Bootcamp Class Project – Purchase Request System

The goal of this project is to create a multi-page, data driven web site using Java, SQL, HTML, CSS, JavaScript, and Angular. *You will be building this project in iterations over the course of the bootcamp.*

Description

You will create an intranet site to implement a purchase request system (PRS). This system will allow company employees to submit requests to purchase specific items from specific vendors with approval from management as determined through business rules. The status of each request should be readily visible and allow all users to see where things stand with each request that is in process.

Technical Requirements

The project should include the use of all major technologies presented in the course, including Java, SQL, HTML, CSS, JavaScript, and Angular. Additional requirements are:

1. Consistent look and feel across pages should be custom CSS pages
2. An Eclipse project that should be published to your GitHub account to publicize your development chops and allow your teammates and instructor the opportunity to review
3. Strong Java fundamentals should be demonstrated by utilizing the Factory Pattern and other standards discussed in class
4. Your project should adhere to the object model discussed at the beginning of the project

Pages

You will be building a Single Page Application (SPA). This means you'll have, approximately, one page for each entity. Build each page out to address the requirements above. Each page in the site should have a consistent look, feel and navigation.

1. Login – This page will handle user login.
2. Home/Default – This page is the landing page in the application. The menu/navigation bar will be at the top and will include links to all pages noted in this document. The center will display the results of the users interaction with the navigation bar. This includes creating a new request, viewing the status of requests, maintenance of the various tables and management approval.
3. New Request data entry form – This page should present the 'header' portion of the Purchase Request and then allow the user to add line items. Once submitted, a unique purchase request id, a status of "New", and a submitted date should be assigned by the system. A success message should be displayed to the user. This page should have the ability to cancel the submission of the request.
    a. Request data. This should allow the entry of a short description, longer justification, the date by which the item(s) are needed, and the mode of delivery (pickup or via mail). The phone number

and email of the requestor should be auto-filled from the User record. Each of these fields should be required. A visual cue should be presented to the user when they have left a required field blank or have entered bad data and should go away when fixed.

b. Line Items: The submitter should have the option to select multiple products on each purchase request. The products will be listed in order of product name. The user should be able to filter and sort this list as needed. As each product is added to the request, a running total should be displayed on the page. This total should be calculated on the back end service.

4. Purchase Request Submit For Review Function – This will be part of the PurchaseRequest Lines page. A button will be available to submit a request for review. Once submitted for review, the PurchaseRequest status should be set to "Review". Anything under $50 total should be automatically approved.
5. Request Review page – This page should show a table or grid with all submitted purchase requests in REVIEW status. This table should be sortable by submitted date and requested date. The reviewer will pick the 'review' button associated with the request and then a 2nd page displayed with ability to approve or reject. Once one of these options are selected the purchase request will be flagged with the appropriate status. If the request is rejected, the reviewer should be prompted for a rejection reason. This page should not allow them to edit the details of the request, but only see the data and mark it as either approved or rejected and save the item back to the database. This page should only ever show reviewers requests that need their attention, so it should automatically filter requests to show only those that are in "Review" status.
6. Vendor Page – This page should have a table or grid that displays the details for all vendors. This should include a 10 digit alphanumeric vendor Code, the vendor's name, a street address, city, state with only two characters (i.e., "KY"), zip, a phone number, email ~~and an indicator as to whether the vendor has been pre-approved~~. The user should able to perform all CRUD functions, if they have authority.
7. Products Page – This page should show all items in the catalog in a table or grid format. The user should be able to click on the product and open a page that shows just that product's details. If possible, a photo of the product (if available) should appear on this page along with the other details. The user should able to perform all CRUD functions, if they have authority.
8. [Bonus – Google Search for Products] If possible, a page should exist to allow the submitter or manager to search for a product on Google and have as a means of comparative shopping and information gathering. This page should display search results or redirect to Google's site. This does not have to integrate programmatically with the rest of the site, but should have the same look and feel.
9. [Bonus – Import a catalog]

Commented [GD1]: *If there is a business reason for this, we could make the user a link to the actual user detail to get the phone and email.*

Commented [SB2R1]: *On review it seems like these could be auto-filled from the user table. Is this OK with you?*

Commented [SB3R1]:

Commented [GD4]: *I don't get the need to have a user entering a request be concerned by the vendor that may be supplying the product being requested.*

Is there a reason for this?

Commented [SB5R4]: *Typically I see the vendor listed by each product in any purchasing app. In this case, if there is more than one provider for the product (ream of paper for example) they could see who the vendor is for each product.*

Commented [SB6R4]: *Making vendor display on the item details page optional, per discussion with Greg.*

Commented [GD7]: *That have been marked as ready for review (Status = REVIEW)*

Commented [SB8R7]: *Agreed. Making this change.*

Commented [SB9R7]:

Commented [GD10]: *To users with Reviewer access*

Commented [SB11R10]: *I agree. I thought that was implied but see I didn't call that out as a rule. Adding in first sentence in this section.*

Commented [SB12R10]:

Data Elements

The following tables should exist in the final solution: User, Request, Product, LineItem, and Vendor. Fields for these tables should be collected from the class diagram and data model presented in class. Any additional tables deemed necessary may be created as well. Each table should have a unique primary key (ID). There will be a minimum set of relationships between Products and Vendors, Requests and Products in the LineItems table, Requests and Users. Other relationships will also be needed but may be subject to your interpretation.

General Requirements and Business Rules

1. Any request over $50 will require approval. If the total request is $50 or less, it is automatically approved and the reviewer should not have to view it on their page.
2. Only users with the role of 'Reviewer' should have access to the reviewer approval flow.
3. Valid delivery modes are: 'Mail' and 'Pickup'.
4. Request status flow:
    a. New -> Review -> Approved or Rejected
    b. [Optional] Once a request is rejected there could be a mechanism to re-open the request
5. Roles: Each person in the User table is considered a user. A user can also have the distinction of being a reviewer or an admin. Only reviewers can approve/reject requests. Only admins can maintain table data.
6. Table Maintenance Rule:
    a. An Admin should be able to maintain data in the User, Vendor, Product and Request.
    b. The status and total fields of the Request table should NOT be editable via the Admin screen.
    c. An admin should be able to change the user associated with a PR. This would be needed in the instance if a user were to create a PR then leave the company. The admin could reassign the PR to another user to either submit it or delete it.
7. PR Approval
    a. Only users with 'reviewer' role can approve requests.
    b. A Reviewer should not be allowed to approve their own requests.

Suggested Steps

- Step 1: Database: Define your data model, create your SQL script(s), and build base data.
- Step 2: Model: Define your object model and create all POJOs and other business classes necessary to manage your application.
- Step 3: Controller: Use the Spring Framework to expose your Java entities via RESTful services returning JSON. Test the functionality using Postman and ensure all CRUD methods work.
- Step 4: UI: Create UI during Angular instruction and complete during final project days.

Commented [GD13]: *There is no data indicating a user is a manager.*

Commented [SB14R13]: *Mistyped… that should read 'reviewer'. Correcting.*

Commented [GD15]: *This is the same as #1 as to reviewing the PR.*

Commented [SB16R15]: *I thought we wanted to call out that only users with a Reviewer role could access the review page. I see this as different than #1, the business rule that calls out auto approval.*

Commented [GD17]: *An admin should be able to change the user associated with a PR. If a user were to create a PR then leave the company, reassigning the PR to another user to either submit it or delete it.*

Commented [SB18R17]: *Agreed. Adding as a business rule.*

Commented [SB19R17]: