Web Application Development
Technical Design

Title: Budget Application

Introduction:

We will design a dynamic web application to help users manage their monthly spending. Users will be able to document their expected income, planned expenses, and financial transactions for each month. Expenses will be organized by category, i.e. Food, Housing, Lifestyle, and Transportation, to make them more readable.

Architecture:

The software architecture we will follow is shown in **Figure 1**below. The architecture is comprised of the following layers: User Interface, Controllers, Business Logic, and Data Access. The User Interface will be the only client-side layer; the other layers represent server-side processing.

Components within each layer will be organized by page. Specifically, features belonging to the same page will be grouped into the same java package. This will ease the process of sharing and merging code as we develop.

The user interface layer is the most unique, as it contains folders for scripts, includes, and styles. We hope to make the web application responsive and will need separate files to keep our work organized. Finally, Asynchronous Javascript and XML (AJAX) protocols will be used to send and receive XML data from the server. The scripts folder will be used to store the XML parsers and callback functions needed to make this work.

The controller layer will contain java servlets only. This layer is responsible for receiving requests from the client and processing them accordingly. Controllers can call managers in the Business Logic layer but will not be allowed to communicate with the Data Access layer directly.

Such a restriction will allow us to handle server-side validation in the Business Logic layer. As a result, incoming data will be validated before it reaches the database.

The Data Access layer will be responsible for communicating with the database. The Database Factory class will contain a static method for returning connection strings. The other classes will be responsible for executing create, read, update, and delete queries for the features they are named after.

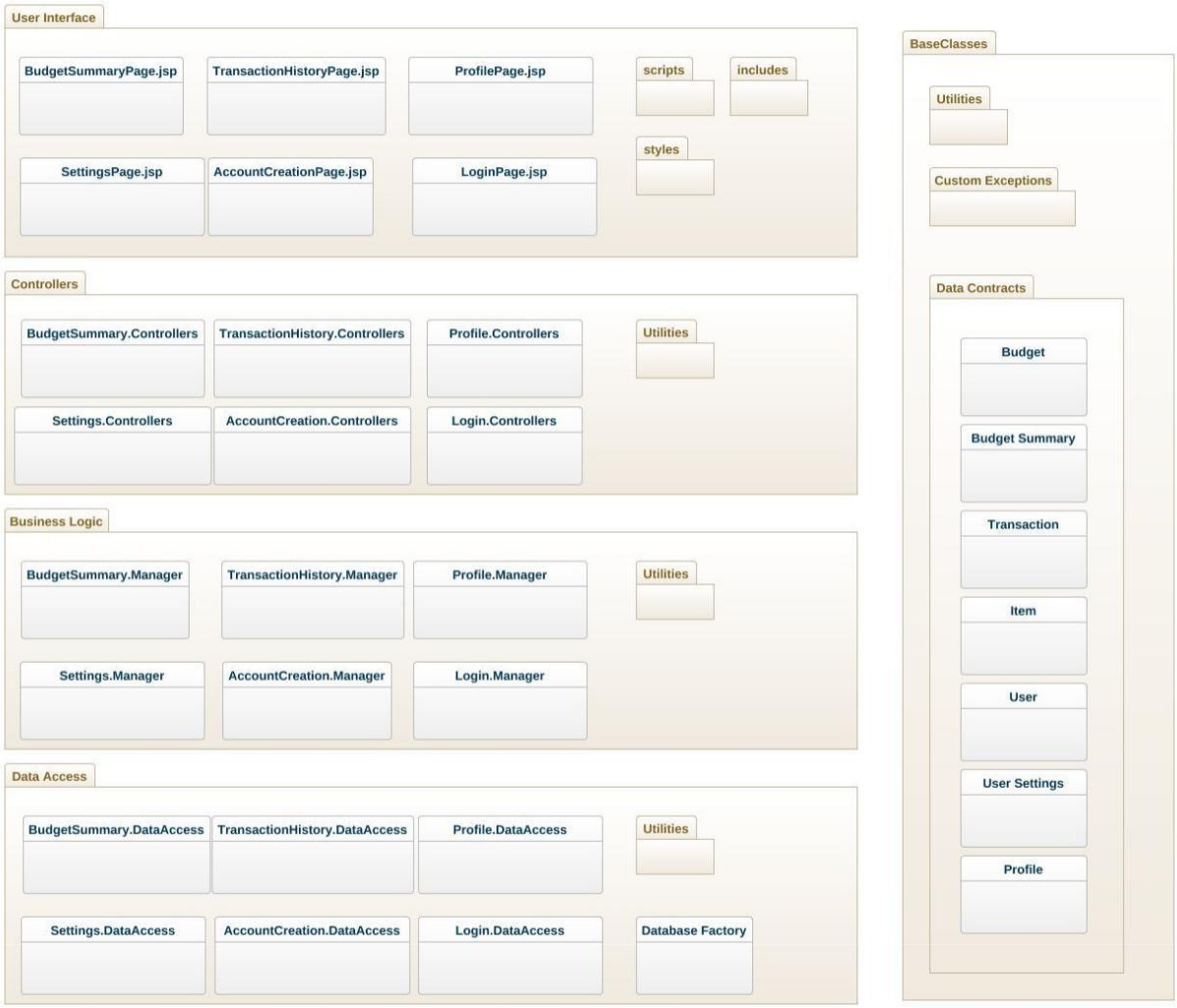

Figure 1. Software Architecture.

Page Navigation:

We plan to use a navigation scheme similar to that of Google Docs or Microsoft Office. Each page will contain a top-side navigation bar which will grant the user access to all pages in the application and allow them to sign out.

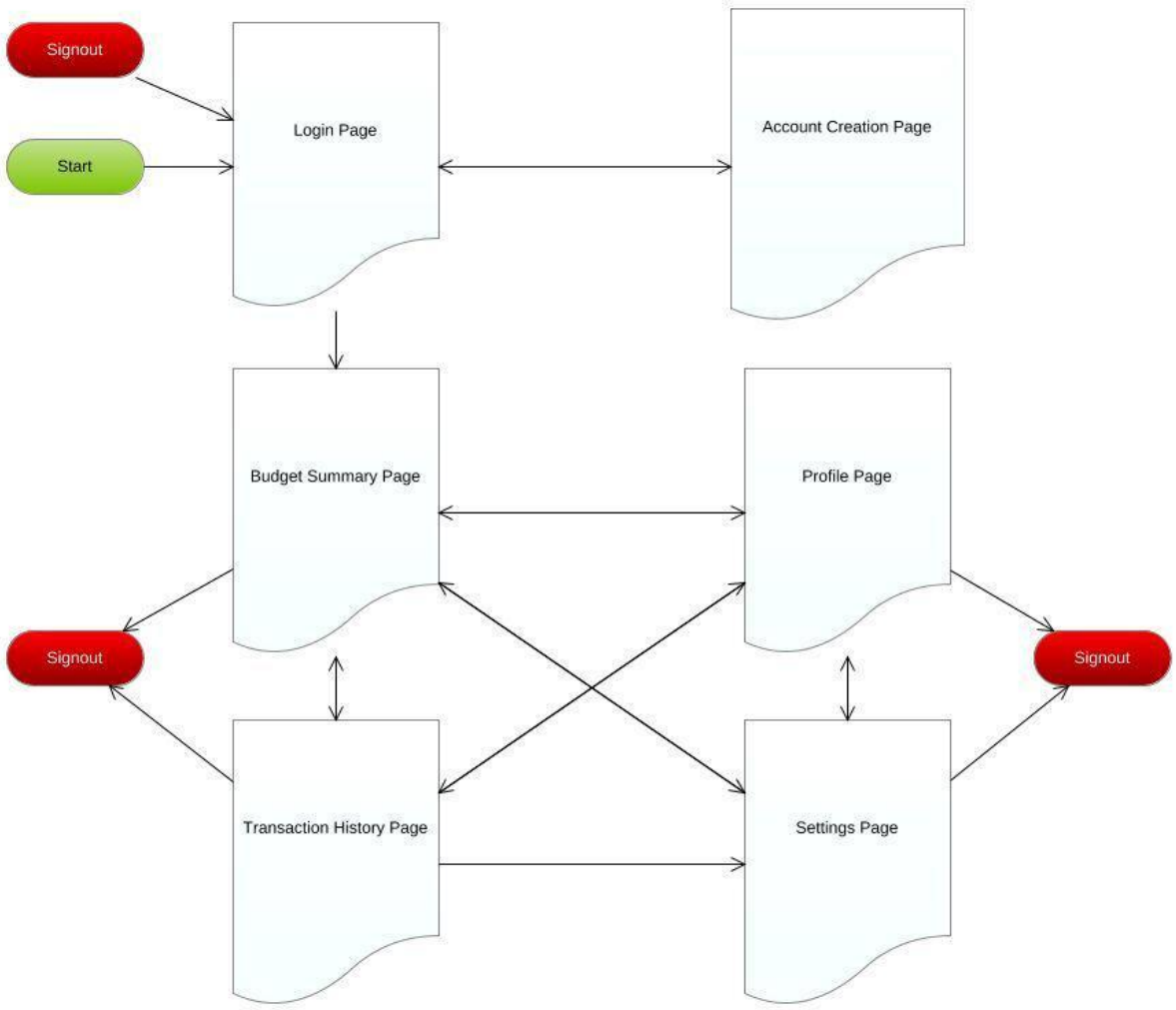

Figure 2. Page Navigation Diagram.

Client-Server Interactions:

The architecture diagram is a general representation of how components will be organized in the project. The following diagrams show how components interact with each other for specific use-cases in the application. Each diagram contains a key describing what the symbols represent:

- The white document symbols represent jsp pages which are viewable by the client.
- The blue boxes represent actions that the user can take while interacting with the jsp page.
- The pink bubbles represent server-side processing that occurs in response to events in the jsp page.
- The purple diamonds represent decisions the controllers must make while processing a request.

There are a few things that need to be mentioned. Typical GET and POST requests will be intercepted by a servlet/controller and handled via a response object. Consequently, the client will be redirected or forwarded to another JSP. This behavior is disruptive as it forces the client to wait for the entire page to reload. To solve this issue, we will implement AJAX requests for small tasks such as modifying individual rows of data. Rather than redirecting the client to a new page, the controller will send a small packet of XML data to be processed by client-side Javascript. This process is shown in **Figure 5**below.

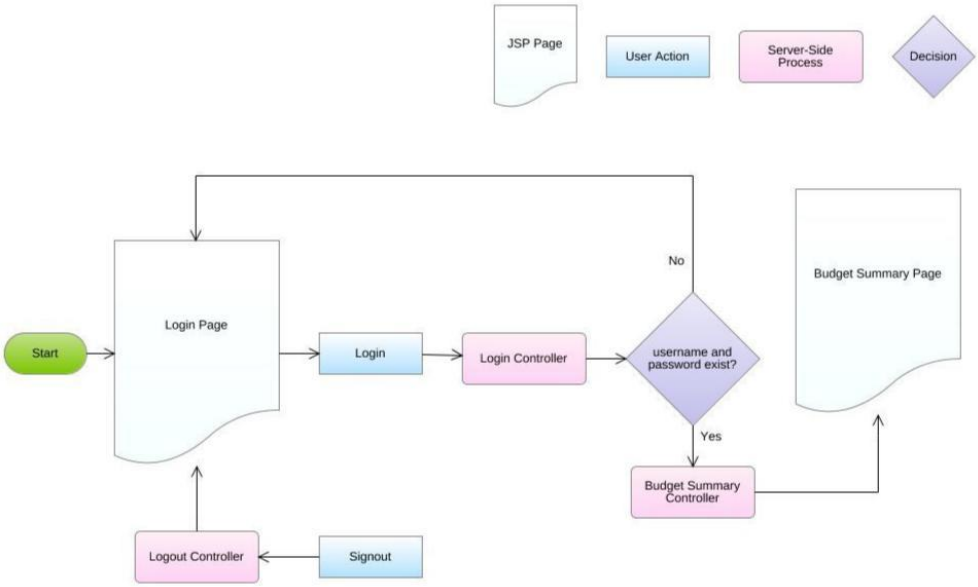

Figure 3. Login Use-Case Diagram.

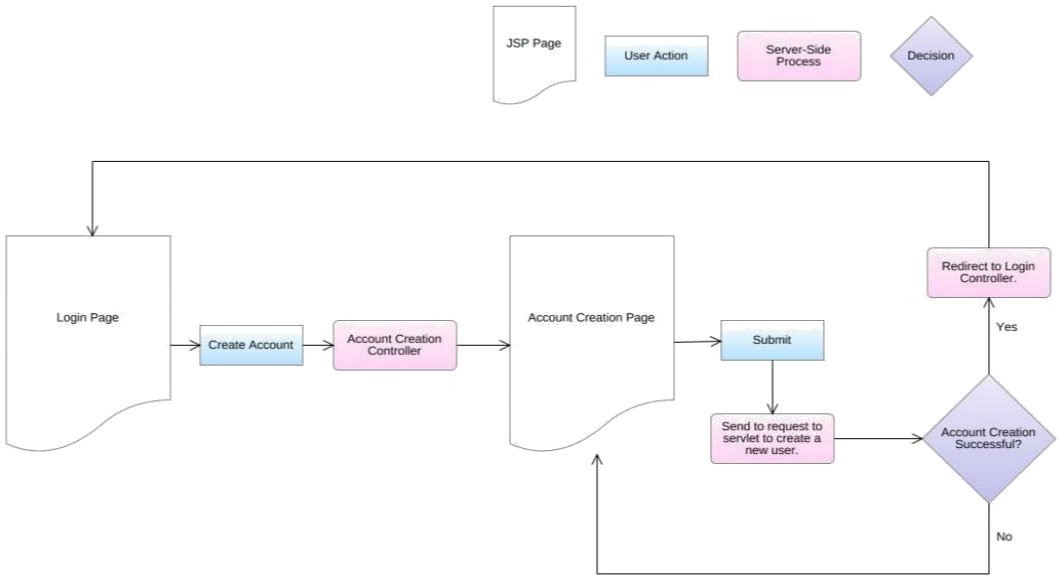

Figure 4. Create Account Use-Case Diagram.

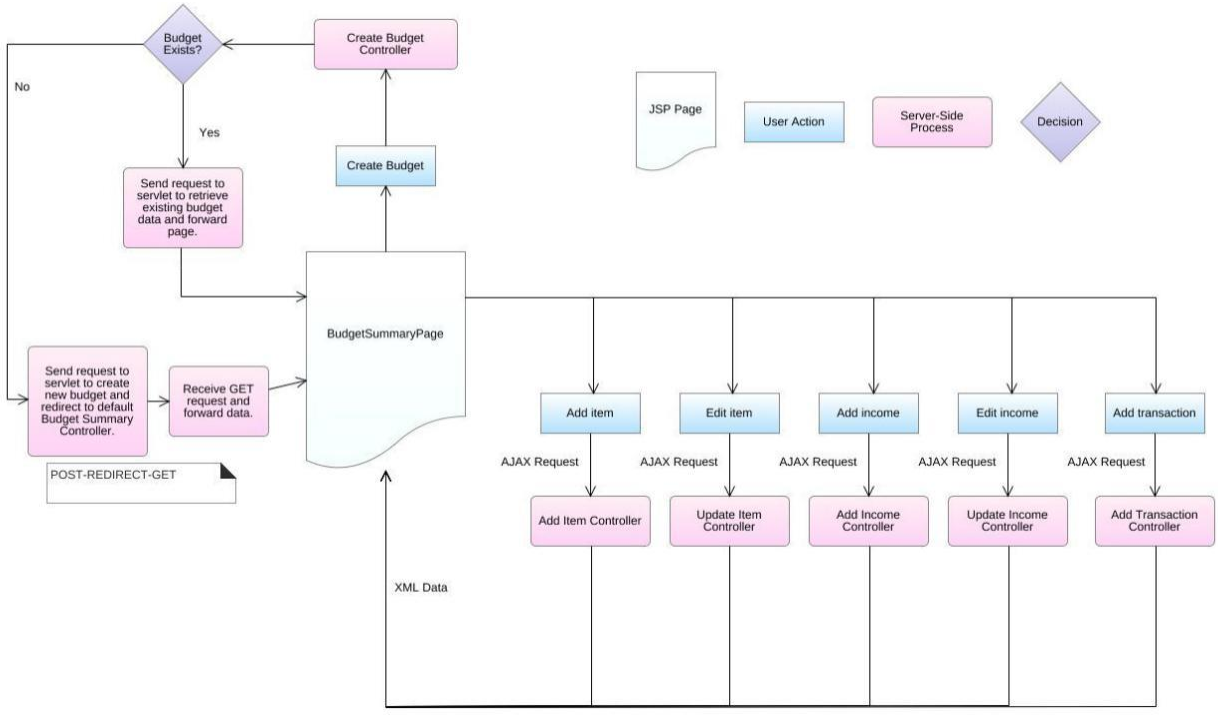

Figure 5. Budget Summary Use-Case Diagram.

MySQL Database:

This project will be built on a MySQL database. The entities we will use for the application are represented in the **Database Diagram**in Figure 6 below. Since we are using a MySQL database, we will use the JDBC MySQL connector to communicate with is.

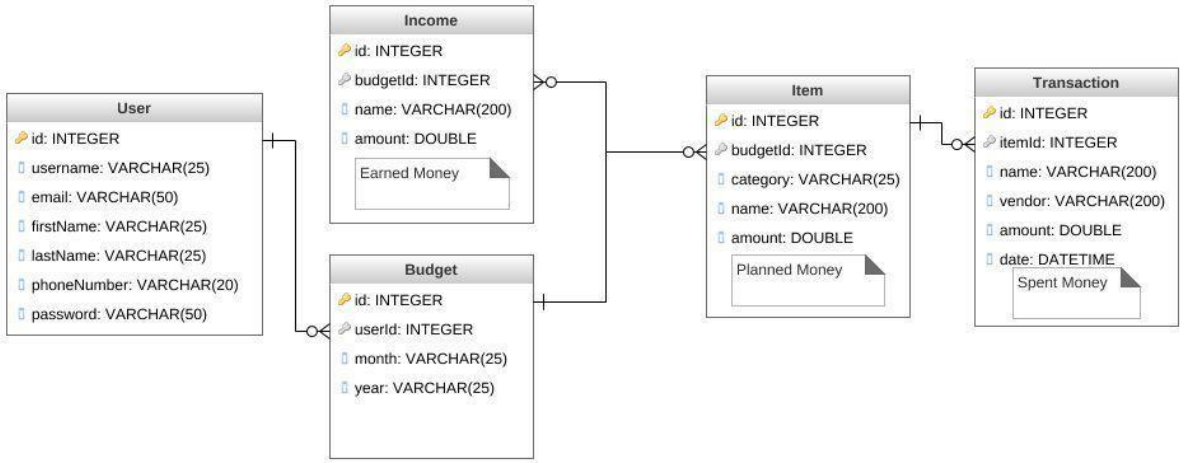

Figure 6. Database Diagram.

Conclusion:

We would like to build a responsive web application to help users manage their monthly spending. The application will organize users' planned expenses and transaction by category to make the data more readable. In addition, the application will allow the user to:

- Generate visual representations of their transactions and budget items.
- Sort through a large history of financial data.
- Identify areas of concern (i.e. where they have overspent).
- Create, edit, modify, and remove large quantities of financial records securely over the web.

Finally, we will use AJAX to process user actions and maintain concurrency between the client and the database.

APPENDIX

Table 1. Project Backlog.

Page	Backlog Item	Description
Budget Summary Page	Design a page that displays a summary of the current month's budget.	The page should allow the user to: add budget items, edit budget items, assign transactions to a budget item.
Budget Summary Page	Add an item to a budget category.	Assign an item to a budget category by category id. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Add a source of income.	Add an income source to the income table and display the changes. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Edit a source of income.	Let the user modify the amount or description of an income source. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Delete a source of income.	Remove a source of income from the income table. The delete request will be made using AJAX.
Budget Summary Page	Edit item information.	Edit and save changes to an item in the budget. The user should also have the ability to reassign the item to another category. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Delete an item from the budget.	This is a high-risk operation. The user should be prompted to confirm and warned that they may lose transactions. The delete request will be made using AJAX.

Budget Summary Page	Add a transaction.	Let the user add a transaction to a budget item. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Category pie chart.	Display a pie chart of all categories in the budget. This is based on the items that are currently assigned and should be updated if changes are made to the budget. The data should be updated everytime the budget is modified. The new data will be serviced using an AJAX request and returned to the client as an XML packet.
Budget Summary Page	Budget progress bars.	Show a progress bar that represents the percentage of transactions relative to each category total. The data should be updated everytime the budget is modified. The new data will be
Budget Summary Page	Budget selector	Let the user select and view another budget they've created.
Budget Summary Page	Create new budget	Let the user create a new budget. They should be prompted to provide a month and year.
Login Page	Design a login form.	The form should include a basic login form and a link to the account creation page.
Login Page	Reset password.	Prompt the user for their email and send them a temporary login password. They should be prompted to create a new password before continuing to the website.
Account Creation Page	Design an account creation page.	The page should include a form for creating an account. Input fields include first name, last name, username, password, email, phone number.
Profile Page	Design a page that displays the user's information.	The page should display the user's information (i.e. username, first name, last name, email, and phone number).

Profile Page	Delete user account.	Allow the user to delete their account. They should be prompted to confirm before the operation is committed.
Profile Page	Edit basic user information.	Let the user change their basic account information (i.e. first name, last name, phone number, and email).
Profile Page	Change password.	Let the user change their password. The user should be prompted to enter the password twice to confirm that they match.
Transaction History Page	Display a table of all transactions in the currently active budget.	The table should include the following fields: description (name), vendor, category, amount, and date.
Transaction History Page	Change range of transaction history table.	Display a lower bound and upper bound above the table. The controls should allow the user to adjust which transactions appear in the table history table based on date.
Transaction History Page	Display a chart of transactions.	The y-axis should represent the sum of all transactions. The x-axis represents the date those transactions took place.
Transaction History Page	Edit transaction data.	Let the user edit transaction data or reassign it to another item/category.
Transaction History Page	Sorting options	Let the user sort by desription (name), vendor, category, amount, or date. They should be allowed to sort in ascending/descending order.
---	User signout	The user is allowed to signout. Doing so invalidates the current session. They user should be prompted to sign back in even if they press the back button.
---	Setup a remote database for the application.	The database should include the following tables: user, budget, item, income, transaction, and category.
---	Deploy the application to Auburn's Mallard server.	Deploy a working version of the application during development and for final release.

User Settings Page	Design a page that displays the user's current application settings.	User settings include: the order in which categories are displayed, turn on email notifications, turn on text notifications, and color picker for categories.
User Settings Page	Edit user settings.	Let the user edit their settings and commit them to the database.