CSCI-4830 Web App 2

**Budgeting Tool**

Software Design Document

**Members:** Alec Bokowski, Christian Woodle, Gabriel Perez, James Triplett, Mark Turpen, Thomas Cordwin

March 7th, 2019

1. **Introduction**
   1. **Purpose**

This Software Design Document (SDD) describes the architecture and design of a budgeting application, as well as the implementation details necessary to satisfy the requirements set forth in Software Requirements Specification (SRS).

* 1. **Scope**

The application is for use by any user who has a need to balance their income against their expenses. The application will provide a simple interface for entering different sources of income and expenses, and will provide the user with a budget to meet their goals.

* 1. **Audience**

This document is intended for use by members of the team developing the application. Unlike the SRS, this document uses terminology and describes concepts that may be complicated for the client to understand.

**2. Architectural Design**

**2.1 Core Modules**

* **Front-End:** The front-end of the application will be composed of various HTML pages that are supported with additional Javascript and CSS code. There will be a page for users to signup, create accounts, as well as a page where the user enters their differing incomes and expenses.
* **Back-End:** The back-end of the application will be composed of various Java Servlets. When end-users submit requests to the server, A Java Servlet will handle that request and respond with the appropriate HTML page.
* **Database:** User data will be encapsulated in a PostgreSQL database. Data that will be saved includes user account information, along with the ability to save specific income / expense sheets.
* **Heroku:** We will host the application using Heroku, a platform as a service (PaaS). Heroku will enable the front-end and back-end, which is all stored in one directory, to communicate with the PostgreSQL database, allowing us to achieve the desired functionality within the application.

**2.2 Decomposition Description**

* **Front-End:**

/users/sign\_in.html: Users get directed here if they are not currently signed in. In the case of a new user, there will be a link to sign up, which will redirect to /users/sign\_up/.html.

/users/sign\_up.html: Page for new users to create a username and password. Once created, they will be redirected to index.html.

/users/#{id}/edit.html: Users will be able to go here to change username and password. There will be validation that the new username is not one that is already in use.

/sheets/index.html: Page that will index all of the users saved sheets. Sorted by id, clicking a sheet will take a user to /sheet/#{id}/, which allows them to view that specific income/expense sheet. Additionally there will be an edit button, taking the user to /sheet/edit/#{id}/, allowing them to change the description of the sheet, as well as a button to add a new sheet, redirecting to /sheet/new.html. In the project, this page will show up as /index.html (the homepage when a user signs in).

/sheets/new.html: Allows user to create a new income/expense sheet. Upon creating sheet user is redirected to /sheet/#{id}.html.

/sheets/#{id}/edit.html: Allows user to edit the description of a sheet they have created. Updating the sheet redirects to /sheet/#{id}.html.

/sheets/#{id}.html: Shows a specific sheet the user has created. Page will be composed of two tabs, one for incomes, and one for expenses. Both pages will list their respective object (income or expense) indexed by id, and each object will have all of its fields listed as well as an edit button if the user wishes to update these values. Rather than have additional pages for the income and expense objects (such /edit, /new), you will update their values on this page.

User Menu: There will additionally be a menu icon on all pages other than /sign\_up and /sign\_in pages. This menu will contain buttons to edit account information, view sheets (takes user back to /sheets/index.html), and sign the user out.

* **Back-End:**

User.Java: Model for the user of the application. Along with data described in the data section, User object may be used to authenticate user when making requests to database so the correct data is grabbed.

Sheet.Java: Model for a income/expense sheet. Along with data described in the data section, there will be a SheetItem array associated with each sheet used to build the list of incomes and expenses for the user.

SheetItem.Java: Model for a specific income or expense of a sheet. Data will include data described in the data section.

UserController.Java: Used to handle routing of pages related to the User, as well as application logic related to the User pages. These pages would be: /users/sign\_in.html, /users/sign\_up.html, and /users/edit.

SheetController.Java: Used to handle routing of pages related to users sheets, as well as application logic related to the sheet pages. These pages would be: /sheets/new.html, /sheets/#{id}.html, /sheets/edit.html, and /sheets/index.html.

JSP Pages: Each HTML page mentioned in the above front-end section will be included as a jsp page. For example, the url /sheets/new.html would actually be /sheets/new.jsp.

* **Database:** PostgreSQL will be used to store a database that consists of users. Associated with users will be income/expense sheets that the user wants persisted. This allows the user to not have to re-enter data as that can be a tedious task.
* **Heroku:** Heroku is a platform as a service (PaaS) that will be used to host our production level website. The entire web-app will be stored in one directory that acts as a Java Web Application. Additionally, the database will be located on Heroku.

**2.3 Design Rationale**

We chose to structure the system how it is due to the simple nature of the application. Rather than have several different components interacting, keeping it as barebones as possible reduces many operations, such as refining and testing the system.

**3. Data Design**

**4.1 Data Description**

The information in our system will be stored as objects. There will be two primary objects: users and income/expense sheets. Associated with income/expense sheets will be an income/expense object. Each of these objects will have their attributes listed below, and in the system these objects will have getters/setters to change the values of these attributes. Data will be stored on a PostgreSQL database and referenced when necessary. The database will consist of four tables. A user table will exist to store all users of the system. There will be an income/expense table to document the different items of a specific budget sheet. Then there will be an income/expense sheet table to persist the entire collection of incomes and expenses. The relation between income/expense and the sheets will be many-to-many, meaning a sheet can contain many income and expense items, and income/expense items can belong in many different sheets.

**4.1.1 Database Schema**

User(

id: int - Primary Key

username: varchar(255)

encrypted password: varchar(255)

)

SheetItem(

id: int - Primary Key

sheet\_id: int - Foreign Key

expense: bool

description: varchar(255)

value: int

frequency\_times: int

frequency: varchar(255)

comment: varchar(255)

)

IncomeExpenseSheet(

id: int - Primary Key

description: varchar(255)

)

**4.1.2 Table Descriptions**

User

id: unique identifier for each user

username: string used to identify a user on the system

password: string used to protect users account

SheetItem

id: unique identifier for each income item

sheet\_id: relates each income item to a specific sheet

expense: boolean where true indicates that it is an expense and false indicates an income

description: description of what the income is

value: dollar value of the income

frequency\_times: number of times income occurs for a given frequency

frequency: period with which the income occurs, valid options are:

* Daily
* Weekly
* Monthly
* Yearly

comment: notes for the user to leave about the income

IncomeExpenseSheet

id: unique identifier for each income/expense sheet

description: description of what the sheet is being used for

**5. User Interface Design**

**5.1 Overview of User Interface**

The User Interface is a critical component of the system, as it is how the end-user interacts with the product. Before diving into a wireframe of how our UI will work and how users get to the various end pages (HTML), It would be good to describe the basic layout of the UI. Below you will find the User Interface broken up into three parts:

* Menu
* User pages
* Sheet pages

The menu will be available on all pages where a user is signed in, and provide easy access to the dashboard, which shows a list of all sheets, as well as access to account info and the ability to sign out. The user pages are all that relate to users, namely the sign in page, sign out page, and edit user page. Lastly, the sheets pages are those which represent the pages where the user actually creates a budget sheet. There is an index page (the dashboard) which shows the user all of their created sheets, a edit page for each sheet, a new sheet page to add a new budget, as well as a page for each sheet to view the actual contents of each sheet.

**5.2 Menu**

**5.3 User Pages**

**5.4 Sheet Pages**