Bookkeeping Design Plan

Jennifer Brady

Matthew Dobson

Andrew Eissen

Kevin Ramirez

Christian Rondon

Steven Wu

University of Maryland University College

CMSC 495

Dr. Robinson

November 12, 2018

Contents

Front-end Design

Language Overview

The bookkeeping project application's front-end is constructed by means of three primary languages, namely HTML$_5$, CSS$_3$, and ES6 JavaScript. The use of each language in the application interface framework may be visualized by means of an analogy related to the human body. HTML serves as the skeleton framework providing a raw outline of the page, JavaScript as the muscular substructure that facilitates animation and motion, and CSS as the integumentary system that gives the application its visible look and feel and shared aesthetic design ethos.

General Approach

Rather than hardcode a set of raw HTML files with specific handlers built in from the outset, the front-end side of the program dynamically constructs what the front-end team has dubbed "scenes." Instead of reloading the page and fetching a separate HTML file every time the user presses a new button in the interface, the program clears the old document object model (DOM) framework in real time and dynamically constructs a new view-model corresponding to the user's request while employing a series of jQuery-inspired fade out/in and swiping animations to make the transition process smooth and seamless from the viewer's perspective. Each of these new view-models, called a "scene," is tailored to certain aspects of the implemented functionality, with scenes dedicated to the login, account creation, and application dashboard pages of the project design. This dynamic approach to scene construction ensures that the user is never left waiting for a new page to be loaded and displayed every time a different functionality button included in the interface is pressed.

HTML

As a result of this heavy JavaScript-driven approach, the sole HTML file included in the program codebase is the initializing `index.html` file. This file is simply used to load the various JS and CSS files that do most of the heavy lifting on the front-end. Due to this dynamic design decision, the program's use requires a supported browser with the user-configurable "JavaScript enabled" permission set to true. Without the use of native browser JavaScript, the program will simply not function.

CSS

CSS styles for use in each of the various scenes in the program are provided via the `styles.css` file. These styles make use of industry-standard device width breakpoints at `576px`, `768px`, `992px`, `1200px` and beyond as defined by the Bootstrap CSS framework to denote the device-specific styles to be applied in each of the cases. In accordance with the content portability and responsive design paradigms, the use of `@media` queries and device-specific widths ensures that all users are fully empowered to use the program without having to default to one or two devices that are capable of displaying its contents as intended. As the user may wish to access the program's functionality on the fly while using a standard mobile device, this approach will ensure that the bookkeeping application is fully usable in its entirety while remaining divorced from the specific devices used to access it.

As far as organization is concerned, the CSS file is largely divided into sections corresponding to the aforementioned page scenes, with each scene having its own clearly denoted set of identifiers and class names containing the relevant styling specific to that scene. In addition to these specific styles, several general-purpose, universallyapplicable styles are included as well, used and applied to multiple scenes to give a sense of visual consistency in accordance with the shared aesthetic. These styles range from the application of a standard color palate to the use of the shared Open Sans and Montserrat fonts that appear in all scenes to add a bit of flavor.

JavaScript

The JavaScript functionality of the front-end side of house is presently stored in a single file called `app.js` and makes use of no external dependencies, libraries, or frameworks of any kind, relying entirely on the inherent flexibility of pure ES6 JavaScript as defined in its present ECMAScript specification. While a few templating libraries dedicated to dynamic table updating were considered, including the jQuery plugins DataTables and Tabular, the relative simplicity of the ledger table design removed the need for a dedicated library and resulted in the development of several vanilla JS handler functions specifically built for the addition and removal of rows that are used in conjunction with a simple HTML table.

Rather than make use of the ES6's `class` keyword and associated framework, the front-end team instead chose to rely on the use of the standard practice ES5 immediately-invoked function expression (IIFE) module approach. This design paradigm was chosen as it offers the use of varied levels of internal and external access scopes while the `class` approach is presently little more than syntactic sugar disguising the standard JavaScript prototyping system in a slightly more readable fashion. The module paradigm involves the wrapping of all logic and application code inside a named IIFE from which is returned an object containing pointers to globally accessible functions or dedicated functions specifically meant for external invocation. This differentiation between inaccessible logic code and externally-accessible code allows for the use of Java-like `private` and `public` style access differentiation scopes and the associated obfuscation of sensitive inner code that should not be accessed from outside the IIFE.

To this end, the script-global scope contains a pair of objects called `accessible` and `inaccessible`, of which only the former is returned from the script upon invocation. This `accessible` object presently contains an externally-invoked `init` function used to start the program upon the successfully loading of the DOM as well as a set of getters allowing external view-only access to the immutable utility `enums`. However, most of the actual script functionality is housed in the `inaccessible` object to which global access is restricted to forbid malicious outside redefinition of essential program functions. This object is thus not returned from the function but remains isolated inside the script-global scope and unavailable for external manipulation. The only way in which this object's property functions may be employed is via the use of the default interface buttons and interactive elements.

Like most of the interface elements, these buttons and interactive elements are also constructed dynamically by a set of helper functions included in the `inaccessible` object. To facilitate the easy addition and modification of these elements by the development team, all interactive elements are included in dedicated arrays of objects (`!Array<object>`). In these arrays, each button, field, and menu has its own individual configuration object containing data related to the element identifier and related class names, a `String` representation of the element's event handler function, names of the element's intended parent node in the DOM, and assorted variability `booleans` that allow the development team to wrap certain buttons in containing outer `<div>`s or the like. This array-based configuration system makes the addition of new elements easier for the development team, providing at a glance a picture of what buttons are included where and the specifics of the functionality they handle in the interface and parts they play in the overall design.

Database Interaction

The integration of the front-end interface with the back-end database is primarily mediated by the common use of JSON files for the return and manipulation of requested data. The front-end interface is responsible for issuing GET and POST requests to the server via a pair of utility functions included in a dedicated `api` object that exists as a property of the module `inaccessible` access scope object. These functions, conveniently named `inaccessible.api.get` and `inaccessible.api.post`, make use of the standard `XMLHttpRequest` to request or provide the server user data in JSON form. Returned data, once parsed by the front-end handler, can be passed to the row addition handlers and added to the in-scene ledger HTML table used to display document data to the user.

Back-end Design

Language Overview

The server-side back-end of the bookkeeping application is implemented in PHP 7, SQL and JSON. PHP is used to define the behavior of the server-side application. In addition to a SQL script used to initiate the MariaDB database, SQL shows up inline within the PHP code. Database logon credentials are stored in human-readable JSON files rather than within the PHP source-code, facilitating convenient access for both the system administrator and the PHP application.

Server

XAMPP and, therefore, Apache HTTP Server and MariaDB, is used as the back-end server/database. The server will only use HTTPS.

Database

All user data are stored using MariaDB across three databases: `UsersDB` contains usernames, `PasswordsDB` contains salted and hashed passwords, and `BooksDB` contains bookkeeping data. Four MariaDB users have varied degrees of access to two or more of the databases depending upon their needs.

The core of the PHP code's database connection infrastructure is an abstract class, `DatabaseConnection`, an instance of which contains and manages a connection to the MariaDB databases using PHP's built-in `mysqli` extension. `DatabaseConnection` defines several generally useful methods for querying the database which are used by its concrete extenders: `AuthenticationConnection`, `PasswordsConnection`, and `PhpConnection`. Each concrete extension corresponds to one MariaDB user. `AuthenticationConnection` is used to query the database for a user's salted and hashed password, `PasswordsConnection` is used to set or update a user's salted and hashed password, and `PhpConnection` is used to enter and query bookkeeping information. In order to prevent SQL injection, all SQL statements executed from PHP are parameterized, prepared statements.

Password Security

Passwords are salted and hashed using PHP's `password_hash()` method using the default algorithm (bcrypt, which is based on the Blowfish cipher) before they are stored.

Back-end Interface to the Front-end

As the front-end is implemented entirely as a dynamic JavaScript application, several PHP endpoint files are used to handle `GET` and `POST` requests from the JavaScript front-end, exchanging information serialized in JSON. PHP sessions are used to maintain users' access from logon to logout.