Algorithms, Data Structures and Security

Artifact Description

Computer Science 340 introduces advanced programming concepts to the student. In essence, the class is based around the idea of incorporating a database into application development. The artifact created in the class is an application which provides a RESTful interface for updating and maintaining a MongoDB as well as an application with a user interface which allows the user to perform basic crud operations on the database.

Justification for inclusion

Category two objectives revolve around secure coding, algorithms and data structures. An application which works with a database, by its very nature, leverages algorithms and data structures as the means of bringing data from the data layer to the presentation layer. Lists, arrays, and other collection objects are the primary means of storing multiple data records. For the most part, algorithms are limited to sorting on the collections themselves while the objects contained therein are manipulated for content.

Including the final product from the CS 340 course makes sense as it provides a reasonably wide scoped view of various data structures and there uses. Additionally, the code must be self-sufficient in error checking. In any situation where user input is involved there is a chance of errors in the content to be pushed into the data layer. Adding error checking algorithms and prompting for correction enables a more solid implementation of the project.

Coverage of Course Objectives

Changes included in the updated artifact begin with a new look and feel which is cleaner and looks more current to today's state of the art. These changes include a user-login feature which allows a user to self-subscribe to the application and gain access to its features. By default, a user is granted read-only access to the system. This means he or she can lookup values using the controls available, but cannot take any actions which will create, update or delete records. When a user has authenticated as an admin, he or she is able to perform full crud operations on any stock record, as well as grant a regular user administration access to the system.

A new feature of the application is the ability to import stock data using a formatted CSV file. The import algorithm is rudimentary and assumes the data is correctly formatted. The idea being that a bulk data import will be used for transferring data from another source using a compatible export format. This functionality mimics the existing JSON import functionality in which the user is expected to know and understand the format for the data to be imported. The application contains a level of error handling which will prevent a poorly formatted file from adding records to the database.

In addition to the creation of the ability to import a CSV file, the algorithm used to convert a web form to JSON object has been enhanced enabling the web page to leverage the existing REST interface for updates and insertions of records.

As with any database driven application data structures are paramount in handling data returned from and sent to the database. Collections in the database contain multiple records each record is correlated to a Java object. As the name implies the collection of records corresponds to a collection of objects. Most commonly these collections are contained in List objects, but some are Sets. In addition to the database, because this is a web-based application a special data structure object known as a ModelAndView is used to pass data from the controller to the presentation layer. The ModelAndView object is essentially an implementation of a Map which is itself two lists which work as a key, value pairing.

Reflection

This exercise was a bit frustrating to me. There is a point of emphasis for algorithms and data structures which to me seems very odd. I have a very difficult time imagining building out an application of any real functionality that does not contain several different data structures. So, in my mind, I've accomplished this point of emphasis by simply building an application with database functionality. While database activities are the focus of the next milestone, to me it's hard to imagine an application with any functionality that does not incorporate some form of permanent storage such as a database.

With regard to security, this exercise has been a blast. I leveraged a basic security library to enable user login and registration. This library gave me some basic functionality but was a starting point only. Integrating it into my existing market tracking application was a bit of a challenge. I needed to ensure that each page in the application is handled under the security and authentication tools. After ensuring each page of the application is covered by security, it then became necessary to restrict portions of the application based on role.

In the original assignment, the project was to contain both traditional user-friendly web components as well as RESTful interfaces. Adding security to the project in the form of user logins caused the REST interfaces to come behind the security protocols. Integration of the interfaces meant creating a new role for users which will grant them REST access without necessarily having to have access to the web site.

Restricting the application based on role can take many forms, and this is where the design decisions have to be made. It is entirely possible to restrict access in the configuration class. Using the configuration eliminates the need to have code checking for a role at various steps. However, as many views require the same logic to bring up the data regardless of role, this can lead to a lot of duplicate code. Taking a modular approach such using view templates and using several helper methods and classes can eliminate much of the duplication of code while creating a secure application.