Chris Hinson: Developing, deploying, publishing and finding your own web service

Critical Analysis

Foreword

This critical analysis has been broken up into 3 parts. Design patterns used and considered, External libraries used and coding techniques implementing throughout the process of building the app. Each part will be broken down into subsections. Any references to files will not be screenshot but will be clearly mentioned as to where those files can be found in the coursework project.

Design Patterns

Singleton

The Singleton design pattern useful when we only ever want to allow a user to have one instance of an object. It puts control of instantiation into the class itself and removes the capacity for multiple instantiations to happen when that is not something a designer might wish for a class.

The Singleton project has been implemented in the FilmDAO class in the model package. A user should only be able to make a single request of the DAO at a time. Once that transaction is completed the connection should be closed to avoid memory leaks. The Singleton design pattern achieves this goal for and that is the reason it has been implemented in this instance.

MVC

The Model-View-Controller design pattern is useful when conceptualizing a project as it allows designers to have a clear goal when building parts of the project differentiating between the data, the logic and command and presentation elements of the project.

The Coursework app operates under the MVC design pattern. The Servlets and Restful methods act as controllers and are in the controller package. The Film, FilmDAO and the FilmInterface are in the Model package. The JavaScript and HTML pages act as the view managing what the user's requests to the controller and what the model is sending back to the user.

DTO

The Coursework App uses the Film.java as a Data Transfer Object. Film is a representation of attributes that the film databases uses to make up an individual record. Once the film is constructed from user input or retrieved from the DAO the Controller is only interested in transferring this object and is not interested in interacting with the object itself. The View element deconstructs the object into its attributes so that it can be presented in a table. By using a data transfer object, we can move around several linked elements, such as a films title, year and director, in one transfer rather than in 3 separate transfers.

DAO

The Coursework App, as previously mentioned, makes use of a Database Access Object in the model package: the FilmDAO. The goal of the project was to interact with a database and retrieve films. The DAO was selected as the approach to manage this as it fit within the MVC approach and could be set up as a singleton. Throughout development the DAO was useful as the approach to view changed from simple ajax, to jQuery ajax to a REST approach. Similarly, the controller adapted from separate servlets to a RESTful approach. Having a separate DAO which the rest of the app only needed to interact with and was not concerned with its internal functionality made building and testing more reliable and efficient.

REST

The REST architectural style has been utilized in the Coursework app. The example of this is in the RestfulFilmController. The clarity of what function is requested from the user and how the input data is passed from the form, as opposed to being appended in the URL, for the POST PUT and DELETE functions fit the project and that is why REST was implemented. Should the app need to be upscaled or require more expansive functionality in future REST would make this simple and easy to do as it is easy to read and easy to change for any future code maintenance.

Façade

In a general way the Front end JavaScript and HTML pages act as a façade to the back end. The goal of a façade is to conceal the functionality of the app as the user may not understand the working complexity. In this case any user who uses the Coursework app has no inclination if where the database is, how a film is stored or created, or how that information is retrieved from unless they choose to use the inspector or look at the code. In this sense the Model and Controller are being concealed by the façade of the view.

The following design patterns were <u>not</u> implemented but were considered

Factory

A factory method is a design pattern that was considered. It allows the construction of classes that have not been predetermined. However, in our case the information the user is inputting follows a predetermined pattern. The only case where a factory was beneficial was in building a DAO. However, the Singleton method was utilized instead.

Connection Pools

Connection Pools are a method of maintaining connections to the database as opening connections is a costly action. By allowing the connections to be reused we can reduce time and computational power. However, in this instance connection pools were not implemented. This decision was made as the Database is being stored on the google cloud and maintaining those database connections could prove both unnecessary and costly. If the app proved to popular or the see a high frequency of usage, then connection pools would be a solution to scaling the project up.

Prepared Statements

Prepared statements are a method of preparing against an SQL injection attack. This is something that could be implemented in future should the security prove to be an issue in the project's life. However, google cloud is backing the database up so even if the database is subjected to an SQL injection there are precautions taken to restore the database.

External libraries

XStream

One of the goals of the project was to allow a user to receive data in 3 formats: plain text, json and XML. Taking the information from the FilmDAO and moving it into the JavaScript front end whilst parsing it into XML proved challenging. To solve this problem the XStream library has been used to take a film array and change it into XML format. The XStream library was selected over JAXB because it had a very simple implementation, should it be required to be maintained it is a simple process. XStream takes advantage of Lazy initialization, only being instantiated when it is exactly required.

XStream code examples can be found in the AllServlet.java, SearchServlet.java and the GET methods with xml format in the RestfulFilmController.java.

GSON

As previously mentioned, one of the goals of the project was to provide the user the option of receiving data in the json format. Similarly, to with the Xstream, gson was utilized as a way of transferring data from the database to the front end in the json format. It was utilized as it made code more efficient, more readable and code became easier to maintain in the build process.

Gson code example can be found in the AllServlet.java, SearchServlet.java and the GET methods with json format in the RestfulFilmController.java.

jQuery

jQuery has been utilised extensively in handling the front to back end connections. jQuery has also been used extensively in reading user inputs from the forms in the HTML pages. jQuery was utilized because it made the code easier to maintain and more readable than other approaches. jQuery also encouraged a more structured approach to building the front end of the application as it encouraged a standard of readability when dealing with HTML elements.

jQuery code itself is jquery.js file and it has been utilized extensively in the jquery-restful-connections.js and jquery-servlet-connection.js files.

JSON to Table

One of the challenges that proved difficult to overcome was the formatting the information from the models into the front end in a readable way. Each format required a separate approach to building tables. The XML and plain text formats proved to be manageable however the json format proved challenging. To solve this problem the JSON to table library was utilized. It has the drawback of being formatted differently from the other table formats. However, it handles all the json requirements making code maintainable and easy to read.

JSON to Table code itself is JSON-to-Table.js and it has been utilized in the jquery-restful-connections.js and jquery-servlet-connection.js files

Coding techniques

Keep it Simple

This is a coding technique that was at the core of the build process. It has 2 core components, keeping code as simple as possible, and making it as readable as possible.

On the readability front all class names, function names and variable names try are used to try and convey the meaning as obviously as possible. For class names the the app uses names like RestfulFilmController, and UpdateServlet which should make clear what each class does within the context of the app. Using the Jquery-restful-connection.js file as an example it has functions such as hideUpdateFilm, deleteFilm which clearly indicate what the function is attempting to accomplish. It also has variables such as format and info which are there to convey the meaning and make the code easy to follow.

From the aspect of keeping the code as simple as possible, there are no nested logic statements, no function that returns more than one object and all the servlets are smaller than 100 lines of code. The RestfulFilmController.java is a good example of how each function completed in less than 25 lines of code. The reason for doing this is so that a class or function stays within its scope and doesn't become a "god class" or overly convoluted function.

Naming Convention

When naming objects and variables it is important to be consistent. This is something that has been attempted throughout the build process. Looking at the index.html each variable follows a naming convention that is easy to follow. For example, a division is called "add-film-boxes" and another is called "update-film-boxes". This continues throughout the index.html file. For the individual elements inside the forms have been named to prevent confusion. In the add film form the title text input is named "film-title-a". In the update film form the title text input is named "film-title-u".

In java the attributes use a consistency in naming, a String to represent title in the AddServlet is called Title, in the SearchServlet it is also called Title. This has been done to minimise confusion and increase efficiency when working on the files and to make the code readable.

Refactoring

To avoid unnecessary repeated code, it is important to refactor code. This makes it more readable and easier to maintain. As the build has evolved over time the code has been refactored in several places. For example, the formatHandler and dataTypeHandler in the Jquery-servlet-connection.js are examples of code that was being repeated multiple times and was refactored into separate methods.

Similarly, ifNull in RestfulFilmController is an example of code that was repeated in the update and add functions and was refactored into a separate method. There are areas in the code that could be further refactored and perhaps a utilities class could be created to handle a series of abstract methods for the servlets. This task was not undertaken as the app is likely to operate on Restful alone in future.