Obligatory Assignment

INFO134

Candidate Number

102

Word count: 1980

Prelude

In this assignment I have worked alone and all the associated code is written by me.

I will assume that the reader of this document is familiar with terms about the subject of HTML, JavaScript and MongoDB.

Part One

In this part I will explain the concept of using a REST API so that data can be extracted efficiently from a MongoDB database and inserted into a HTML document through DOM manipulation. I will show how this was done in the files given to me in the assignment and establish my knowledge about the workings of the different parts that makes up a dynamic Web-site working with an external database using REST API.

The flow of information on index.html

The information flow does not start until a user enters “index.html”. This page then initializes the script “profile.js” in-between the <head> part of the document and the function “setup_frontpage” is initialized. This function initializes an XMLHttpRequest and sets the type→ “get” and target path→ “http://wildboy.uib.no/mongodb/profiles/profiles/?filter_username=” + current_user + “&limit=1, true”. This is a query to retrieve information about the current user, set to “bruker1” for now, as well as a path so that the request can find the information it needs and hit the right mongodb database. It then asks the database to respond in JSON format. At last the request says that it should limit the results to one so that it does not return other information than the one result we are interested in. All that is then left is to set that whenever the query is finished and the response has been received it should start an anonymous function as can be seen on line 8. The XMLHttpRequest is not actually sent before line 42 and what is in-between is just the function that should be called once the request is finished.

The script “profile.js” now has access to the information about the user we want. A quick check as to if the results contains the information we want is done in line 13. If the response has the username it is inserted into the HTML through DOM manipulation. After this is done the two lists that should be inserted is done in the same way as the username was if the information exists.

The flow of information on show_movie.html

This works site works in the same manner as the index.html page. The scripts “query_params.js” and “show_movie.js” is executed in the respective order and “query_params.js” takes the information in the URL and transforms it into an JS-object. The show_movie.js then takes that object and uses it to target the right movie in its XMLHttpRequest. This page initialises the XMLHttpRequest in the same way as index.html and grabs the results after it has been initialized in line 45 of “show_movie.js” This anonymous function takes the movie it retrieves and displays the information in table rows through DOM manipulation again.

The flow of information on search_results.html and advanced_search.html

Here things are a bit different. While advanced_search.html is not implemented I will talk about how search_results.html uses local JSON files to search through and extract the information needed. search_results.html files initializes query_params.js to do what was done in show_movie.js and it also initializes object.js. This is because it needs the massive .json file to search through. Finally it starts search_results.js where it grabs any movie that gets a match in the .json file and displays it in the html tag with the id: res_list.

MongoDB and the REST API

Concerning myself with show_movie.js and profile.js since these are the two script files that builds ut the REST API and uses it, these two scripts show that with simple GET requests over XMLHttpRequests it is very easy to set up and to use. It is very simple to make these XMLHttpRequests on any action that requires an update and therefore get very smooth updates without brute force search on local files. With the use of other types of requests like “PUT”, “POST” or “DELETE” one could easily make this into a ready to use application.

Part Two

Here I will talk about the changes that I would have made in the different types of files. I will divide and talk about HTML, CSS and JavaScript generally. I will show that some parts of the task will stay much the same while other parts will be partially or all-round rewritten. Using generalisation I will use my knowledge about programming to explain how I would use one JavaScript file to extract the information that each Web-page needs through the REST API that is provided to us.

Changes in the HTML

Interestingly enough the HTML would not need much updating at all. One could of course drop some of the script tags in the <head> tags since files like “genres.js” or “objects.js” is no longer needed, but other than that the HTML does not need much change.

Changes in the CSS

The CSS would obviously not need any change since this is connected to the HTML tags and so if there was no new tags there would not be needed any changes in the CSS.

Changes in the JavaScript

Here it becomes obvious that some change is really necessary. Since my functions for displaying information is separated from my functions for finding the right information to display. My solution to the different html pages are somewhat different. This is what I will discuss in the next sections. I will also discuss the script file “generateIndex.js” even if it is not part of the requirements in this task I want to discuss it here since it uses the method of extracting information from the json files and would have to be rewritten to use the REST API.

I would have written an API in a new file called “api.js” file to make calls and extract information from the MongoDB. This will make many of the functions in my JavaScript files obsolete and unnecessary. This would be quite short and only handle some of the common information calls to the database like “get all movies where attribute x = y”, “get one random movie” or “find movies that is rated. These calls, only to the local storage files are used quite often and would have saved me a lot of time. Going forward I would have deleted some of the files that I currently have: all of the json files and “api.js” would replace “generateArray.js”.

Other than that I would replace all the instances of extracting information in the other JavaScript files like “generateProfile.js”, generateIndex.js”, “show_movie.js” and “search_results.js” with calls to the new API file. These calls would include the ones to get the rating of a movie and find the correct genre of a movie. These are quite heavy when two large “json” files is intertwined, but there is no better way that I know.

The search functions would of course be deprecated since the information would get filtered through search queries in the database and not locally. In my display functions I would also remove data extracting and get the information beforehand so that it does not query to a database while executing DOM manipulations.

Other than this I can not think of anything that would need to be done differently. I believe that this would have worked and that it would handle the information efficiently and precisely.

Part Three

In this section I will explain what all the files that I have submitted contains and what their function is. I will separate out what is in the “header” tag in all of the “html” pages and call it “common_header”. This is because it is the same in all aspects, but pathing so I have decided that I would separate it here.

--- HTML---

“common_header”: This is where the title, navigation and the search function components exists. The navigation is simply to the profile and to log out now since we do not have authentication or authorization and assumes that everyone is logged in. The search function can either be used or it can take the user to advanced search directly.

“index.html”: This is the landing page on my Website and presents the user with three horizontal scrollable lists. The first list is the movies that exists in the database with the latest reg_date, the second one is intended for movies that the user has loaned, e.g. (last watched movies), but is for now randomly chosen. Lastly there is one based on the current rating of the movies and displays the ones with the highest first.

There are then a rating section, this is not implemented yet since it is impossible to insert into json files with JavaScript.
All of the movies in the lists and the current one in the rating section are anchors that lead to details about the respective movie. All of these are extracted from
Lastly the rules that apply to the people that loans movies in this institution.

“profile.html”: Uses “object.json” and “reviews.json” to produce two lists of movies. One is movies that the user already has loaned and the other is movies that they wish to loan.
“search_results.html”: Uses “object.json”, “reviews.json” and “genres.json” to display a list of movies that is filtered based on the search terms that is found in the “get” response from the respective Web-page. The list is generated and inserted through the “search_results.js”. The search

terms are carried through so that the user can remove or add information easier, instead of rewriting everything should a more precise search be needed. It is not limited to how many returned items are allowed so the search can take a while to produce.

“movieDetails.html”: Uses the information in the “get” response to filter out the information it needs from “object.json”, “reviews.json” and “genres.json” and generates a dynamic html file to display the information the user could need. It is divided into two parts. One textual information area and one media area. It loads images and/or trailer and displays them properly if they exists.

---- CSS ----

“common.css”: Styles tags that exists in many of the html documents so dupication is not needed.
“header.css”: Styles the header since this exists in all of the html documents and is equal in all aspects.

“master.css”: Styles the index page where it joins together with “common.css” and “rating.css” to make up “index.html”
“movies.css”: Styles the individual html pages of the individual movies.
“profile.css”: Styles the profile page of the individual user.
“rating.css”: Styles the rating system wherever the html is inserted.
“search.css”: Styles the search results and their arrangement.

–-- JavaScript ----

“generateArray.js”: Generates a new instance of a movie-array. This is to extract from the “object.json” files and present a full array that can be manipulated in another function.

“generateIndex.js”: This is the script that generates the three lists and the rating element. It takes lists from “generateArray.js” and splices/sorts the array and uses DOM manipulation to insert it into “index.html”

“generateProfile.js”: This takes now random elements and puts them into the two lists that “profile.html” displays.

“query_params.js”: This takes the “get” array and extracts the information into an object.

“search_results.js”: This takes the search terms from “query_params.js” and applies them to the appropriate filters. Then the array from “generateArray.js” is spliced and what remains is the items that applies to the user.

“show_movie.js”: This takes the id that “query_params.js” outputs and generates a website to the corresponding movie object in “objects.json”.

---- JSON ----

“object.json”: A json file that contains an object made of movie instances based on “id”s.

“genres.json”: A json file that contains an object made of genre instances connected by “object.json” “id” numbers.

“reviews.json”: A json file that contains an object made of reviews instances connected by “object.json” “id” numbers.

---- IMAGE ----

“noFound.png”: This is the image I insert into a movie cover image should an error occur like if the image is not found.