CS 408 - Purdue University

MyCritic

Regression & Incremental Testing Log -Sprint 2

Anubhav Saxena, Avadhoot Joshi, Brandon Kent, Jacob Conley, Jordan Hagedorn

Classification of components

Server Side API Wrapper

- Module that houses the various API call parameters to the different APIs in order to retrieve information regarding the media. This module is called directly by **Browse Media, Search Media, and Media Summary Page** components
- **Inputs:** Request Type + Parameters
    - Request Type specifies the type of media the query is for, as well as the information that is expected return.
    - Parameters are optional in the case of returning a list of most popular content, but are instead used for searching and specifying a single specific title
- **Outputs:** A JSON object containing the requested data
    - JSON Structure depends on Request Type

Browse Media

- Module consists of a window Tab for each type of media. Each tab is responsible for calling functions on the **Server Side API Wrapper** module to get a list of popular media within each category, and updating the graphical interface accordingly. The Apple Music API is a component that has no need in the actual server and is only a single line that pulls the top 50 songs, therefore it is within this module.
- **Inputs:** User clicks on a tab labeled “movies”,”music”,”games” or “books”
- **Outputs:** The client **graphical interface** displays a list of clickable elements of the top 50 items within the respective category

Search Media

- Contains the search bar within the header where queries are processed through the **Server Side API Wrapper** where the top 5 matches of each media category are obtained and displayed on the client graphical interface
- **Inputs:** A search query which will be processed as a string
- **Outputs:** The client **graphical interface** will display 4 different lists, labeled by each media type, where contained within the list are the top 5 matches for the search query

Media Summary Page

- Module is comprised of a web page customized specifically for a single piece of media. This information is obtained by calling the **Server Side API Wrapper**.
- **Inputs:** User clicks on an element in either list output from the **Browse Media** module or from the **Search Media** module
- **Output:** The client **graphical interface** is updated with title, summary, and rating details of the media that was clicked.

User Account Registration and Login/Out

- This module contains the necessary base functionality of having user accounts. The accounts and account details are stored within a MySql database on the server.
- **Inputs:**
    - Registration
        - User clicks the ‘register’ button and is prompted to enter their email, display name, and password. The server processes this into database input and queries the running MySql database
    - Login
        - User chooses to login and is prompted to enter email and password. Server processes this into database input and queries the MySql database
    - Logout
        - User chooses to logout of their account
- **Outputs:**
    - Success
        - User entered valid/correct details when registering/logging in and is presented with a success message on the client **graphical interface**, as well as an updated navbar containing ‘MyProfile’ and ‘Logout’ buttons. On the backend, a client-side session is stored.
    - Failure
        - User entered invalid/incorrect details when registering/logging in and is presented with a failure message on the client **graphical interface**.
    - Logout
        - Client-side session is destroyed and user no longer can click the ‘MyProfile’ and ‘Logout’ links on the client **graphical interface**

Graphical Interface

- The GUI presented to the client containing the necessary components to interface with everything below it.
- **Input:** The user clicks, types, and submits requests through interacting with interface buttons, links, and textbox forms, generating appropriate GET requests to call the **Search Media**, **Browse Media**, and **User Account Registration and Login** components directly
- **Output:** The user is presented with a graphical representation of the output generated by the component it called

Follow Users

- Module allows scenario where user A follows user B and user C, populating user A’s feed with user B and C ‘s reviews.

- **Input:** While viewing another user's profile, click the 'follow' button, which sends the both user's IDs to the server, where the Follows table in the database is updated with this information.
- **Output:** When the user who followed another user views their feed, the database is queried and the feed is populated with reviews from the users they follow

Write a Review

- Allows logged in users to submit their media reviews to the database for others to view
- **Input:** The user must log in and navigate to and individual media page. At the bottom is a drop down input and textarea input, which are sent to the database via the 'submit' button.
- **Output:** The UI is refreshed, and the user's new review and rating are visible to anyone on that media page.

MyProfile Page

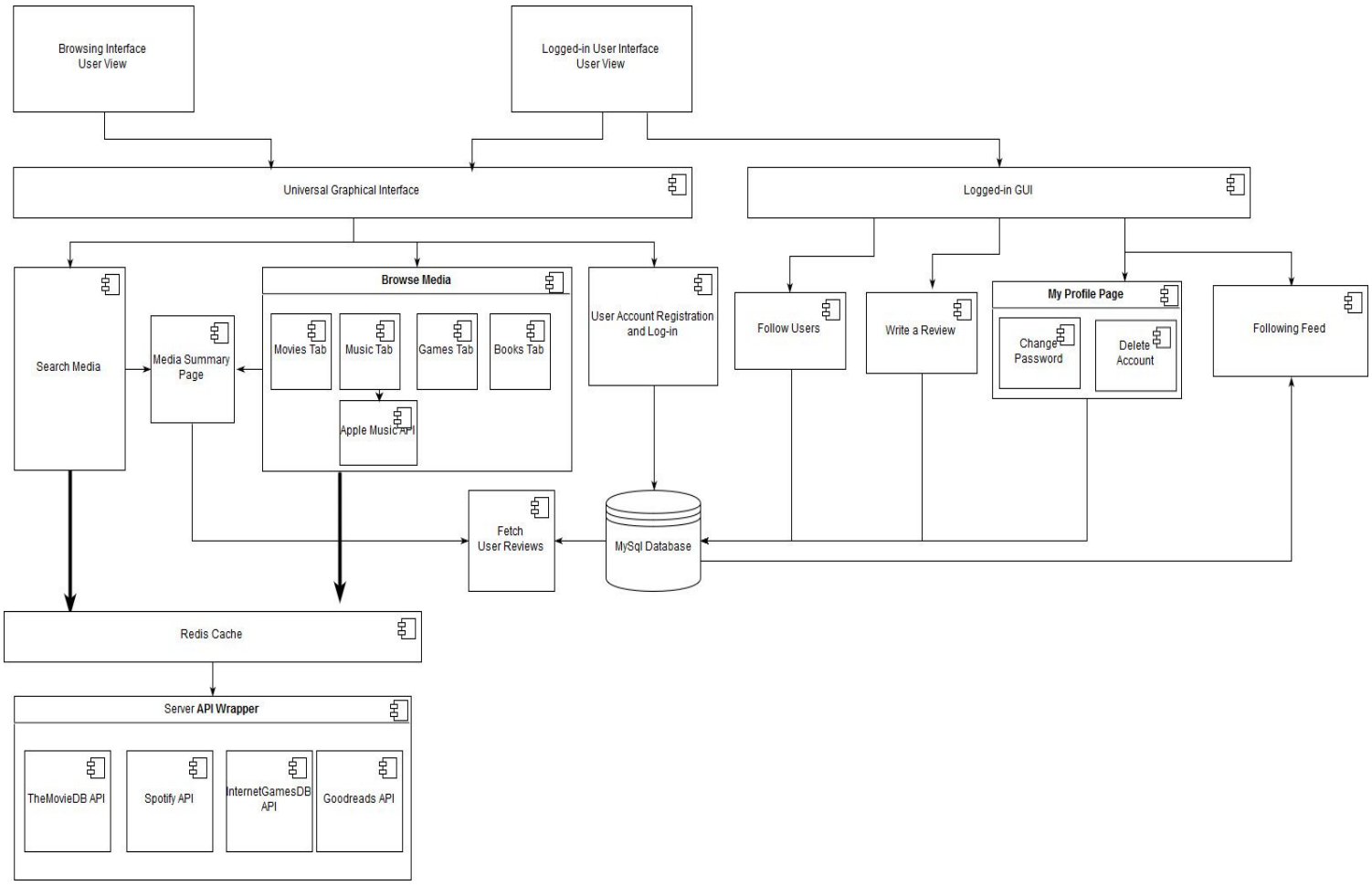
- The module allows the user to view, modify, and delete aspects relating to their own account
- **Input:** The user must log in, and navigate to the 'MyProfile' page, where they can provide additional inputs that correspond to submodules '**Change Password**' and '**Delete Account'**
    - **Change password:** The user inputs their current password, along with their desired new password, and their account entry in the database is updated.
    - **Delete Account:** The user inputs their current password, and confirms that they would like their account, and all associated data deleted
        - Any reviews that this user left, or follower instances, will be deleted.
- **Output:** The GUI shows the user all of the reviews they've left in the past

Following Feed

- This module interacts with the **Follow Users** module and the **Write a Review** module to display a feed of reviews left by users that the user follows
- **Input:** A user can either log in, or navigate to 'home' to trigger their feed
- **Output:** The GUI is updated with a list of reviews

Redis Cache

- This module acts as a layer between calls to the external API's and the server by interacting with the **Browse Media, Search Media, and Media Summary Page** by caching responses, thus limiting API calls.
- **Input:** A JSON object string that comes from an API when the user requests popular media, individual media, or initiates a search query
- **Output:** A JSON object string that is sent to the client



Given the structure of our product we felt it was best to implement a bottom up testing methodology. We noticed that we had four smaller APIs that were going to be the basis of our whole project so we tested each individual API first to test for basic functionality. Then we moved on to test the search module that combines the functionality of the four separate API's. Using this bottom up methodology we were able to find some bugs and fix them to enhance the functionality and experience of our website.

Incremental And Regression Testing

Server-Side API Wrapper

- Automatic testing using node module TestCafe to mimic interactions within the user's scope, such as generating calls from the Search Media, Browse Media, and Media Summary Modules and checking against NULL values and for correctness regarding that the displayed details are in fact what was returned by the API. Regression tests are for ensuring that old defects such as client pages not rendering, are not reintroduced .

Browse Media

- Automatic testing using node module TestCafe. Tests call the Server-Side API Wrapper to ensure that final output on the end-user display is correct, and regression tests ensure that future fixes do not interfere with new fixes

Search Media

- Automatic testing using node module TestCafe. Tests inject the Server-Side API Wrapper module with search queries, and expect outputs from each of the APIs
- Stress test on search module using most popular 1000 words

Media Summary Page

- Automatic testing using node module TestCafe where tests are generated to mimic the user clicking on links in the Bowse Media and Search Media Module. Ensures no errors and no undefined/null values are presented on the graphical interface. Regression tests ensure that our text formatting bugs ('undefined' being listed for example) are replaced by messages meaningful to the user.

User Account Registration and Login/Out

- Testing by sending various random strings into the register/login forms and ensuring that the server and MySql database appropriately respond to submissions that are outside of bounds, edge cases, or use illegal characters.

Follow Users

- Testing by creating random users, following each of the random users, and ensuring that the follow and unfollow buttons are displayed properly and check they are followed in the database.

Cache Searches and Media Pages

- Tested by logging to the console whenever a request was fulfilled by the cache or by the API and comparing the results

Feed Page

- Testing by navigating to the feed page and ensuring that the reviews from the users followed are displayed. Then assert that the filter option displays the expected elements for each filter.

Write a review

- Tests conducted by leaving accounts across multiple media types with multiple accounts and asserting that the reviews are present in the review list.

User Page

- Testing by asserting an existing user with reviews ensure that the reviews are displayed from most recent to least recent.
- Testing by asserting an existing user with no reviews displays no reviews.
- Testing by asserting a non-existing user displays that the user doesn't exist.

Graphical Interface

- Testing links, buttons, and client-side regex for forms

Defect Log

Module Browse Media			
Incremental			
Defect No.	Description	Severity	Solution
1	Movie average score is the total score not the average score	3	Make change the json to get the average score not the total score

Module Browse Media			
Regression			
Defect No.	Description	Severity	Solution
1	If the API returned a null value a song, the music div id would be off, making the song count incorrect and impossible to identify a specific song in the list.	3	As opposed to using the iterator in the for loop, use a counter variable that only increments whenever the song is not null
2	Occasionally, api results would incorrectly return null or undefined, but were still cached, making all future calls undefined	2	Check for null before adding the browse media results into the cache
3	Because certain functionality was moved to the server, the loading bar stopped appearing	3	Make the loading bar appear before the server requests are made

Module Media Summary Page			
Defect No.	Description	Severity	Solution
1.	Each element in the reviews list was incorrectly assigned an id using a colon, which didn't actually set the value and made the row inaccessible by number	2	Change the colon to a equals sign

Module Search Media			
Incremental			
Defect No.	Description	Severity	Solution
1	Search results were being added to the cache even if the API malfunctioned and sent null	2	Check for null values before adding search results to the cache
2	Filtering did not filter, there is a parsing error	2	Remove the parsing to json because its already json.
3	When finding search results with no results, the results page left a blank space	3	Wrote “no results available” to eliminate confusion

Module User Account Registration and Login/Logout and Deletion			
Incremental			
Defect No.	Description	Severity	Solution
1	Deleting an account did not delete all information associated with that account, such as reviews and follow information	2	Before deleting the user from the db, delete the rows in the other tables that contain that users info.

Module Users Page			
Incremental			
Defect No.	Description	Severity	Solution
1	follow button still displays when user doesn't exist	3	Check if user is null and remove buttons
2	Follow and Unfollow buttons can be displayed for the logged in user	3	Remove the follow and unfollow button if the user is the logged in user
3	Trying to load a user that doesn't exist loads a blank page	3	Check for null users and display User Doesn't Exist
Regression			
Defect No.	Description	Severity	Solution
1	Follow and Unfollow buttons can be displayed for the logged in user	3	When checking for null users make sure the check for user == logged in user is reached

Module Feed Page			
Incremental			
Defect No.	Description	Severity	Solution
1	Song titles were appearing as NULL, as this is how they were being added to the database, caused crashes later when accessing	1	Set the title BEFORE loading the review form, so that the title gets sent to the database

Other			
Incremental			
Defect No.	Description	Severity	Solution
1	NCONF module became unusable in later versions of node	2	Instead of nconf, use JSON parser to configure MySql connection

These tests were automated using testcafe for javascript. It is an addon that allows us to simulate user inputs on the website. This will allow us to test multiple inputs for each module, test edge cases, and is a good basis for regression testing to make sure bugs don't integrate back into the system.