Startup Submission 05: Adding a Web Server
Team: BBQForte

Report: HTTP Requests / Routes

1. GET /user/:userid
    a. This HTTP request returns a User object from the mock database. It replaces the getUserData function in the mock server.
    b. Various components utilize this request throughout the app, most notably the forums and the private chat.
2. GET /carousel/
    a. This HTTP request returns the Carousel object stored in database. It replaces the getCarousel function in mock server
    b. Everyone has access to the carousel because it will be public for everyone to see on the main page
3. GET /news-updates/
    a. This HTTP request returns the NewsUpdates object stored in database. It replaces the getNewsUpdates function in mock server
    b. Everyone has access to the News Updates because it will be public for everyone to see on the main page
4. GET /spotify/user/:userid
    a. This HTTP request returns an authorization URL generated by the Spotify Web API that either logs the user into Spotify or prompts them to log in.
    b. Displayed in a separate window.
5. GET /spotifycallback
    a. This HTTP request is used by Spotify to deliver the authorization code.
    b. It sets an accessToken and refreshToken that allows the user to perform tasks with Spotify.
    c. This login is required for most playlist functions.
    d. It redirects to the home page. (I cannot figure out how to redirect to the specific user's playlist feed page).
6. GET /spotify/loggedin/user/:userid
    a. This HTTP request checks if the user is logged into Spotify.
    b. It returns 'true' or 'false'.
7. DELETE /spotify/user/:userid
    a. This HTTP request logs the user out of Spotify.
8. POST /spotify/songlist
    a. This HTTP request searches Spotify tracks and returns the top 20 results.
    b. Returns a list of Song objects for the user to choose from.

9. POST /spotify/playlistresults/:userid
    a. This HTTP request searches Spotify for premade playlists made by any user of Spotify.
    b. It returns an array of Playlist objects.
10. GET /playlistfeed/user/:userid
    a. This HTTP request retrieves the playlist feed for the given user id.
    b. Can only be called by the given user.
11. POST /playlist
    a. This HTTP request creates a new, empty playlist and adds it to the database.
12. PUT /playlistfeed/user/:userid/playlist
    a. This HTTP request adds a given playlist from Spotify to the database.
    b. It also pulls as many of the tracks as it can into the playlist (limit: 100)
13. PUT /playlist/:playlistid/songs/:userid
    a. This HTTP request adds a song to both the playlist in the database as well as the playlist on Spotify.
    b. If the user does not own the playlist, it doesn't do anything.
14. DELETE /playlist/:playlistid/songs/:songindex
    a. If the user owns the playlist on Spotify, this HTTP request deletes the song from both the Spotify and local playlists.
15. DELETE /playlist/:playlistid
    a. This deletes the local copy of a playlist.
    b. It does not affect the Spotify version.
16. PUT /playlist/:playlistid
    a. This edits the local information for the playlist.
    b. It has no effect on the Spotify version.
    c. This allows the user to rename a playlist, associate it with a game, assign it a genre, and give it a description.
17. PUT /playlist/:playlistid/votes/:userid
    a. This HTTP request adds a 'vote' to a playlist.
18. DELETE /playlist/:playlistid/votes/:userid
    a. This HTTP request removes the given user's vote
19. GET /user/:userID/nickName
    a. This HTTP request returns the nickname of the user with ID userID
    b. No authentication is needed for this, any user can see it
    c. This is used in the follower/following list for the profile
20. PUT /user/:userID/name
    a. This HTTP request requires a singular string in the body
    b. The user sending this request must be the same as userID

    c. It is used to edit the full name of the user as seen on the profile page
21. PUT /user/:userID/about
    a. This HTTP request requires a singular string in the body
    b. The user sending this request must be the same as userID
    c. It is used to edit the about section of a user as seen on the profile pag
22. GET http://api.steampowered.com/ISteamUser/ResolveVanityURL/v0001/?key=:steamKey&vanityurl:userName
    a. This HTTP request returns the steamid of the given userName
23. GET http://api.steampowered.com/IPlayerService/GetOwnedGames/v2/?key=:steamKey&steamid=:steamid
    a. This returns the owned games of the account belonging to the given steamid
24. GET https://api.steampowered.com/ISteamUserStats/GetSchemaForGame/v2/?key=:steamKey&appid=:appid
    a. This request returns information about the game tied to the given appid
25. GET https://api.steampowered.com/ISteamUserStats/GetPlayerSummaries/v2/?key=:steamKey&steamids=:steamid
    a. This HTTP request returns information about the user tied to the given steamid

Matthew's Honors HTTP Requests/Routes:

1. PUT /user/:userID/recommendations/:key
    a. This route accepts recommendation keys for users and moves the playlist from the recommendation list to the playlist
2. DELETE /user/:userID/recommendations/:key
    b. This route deletes the keyed recommendation without accepting it.

Logan's Honors HTTP Requests/Routes:

1. GET /forum/
    a. This route returns the forum object.
2. GET /forum/category/:category/topic/:topicId
    a. This route returns a topic object (dependent on the topic id)
3. PUT /forum/category/:category/topic/:topicId/newThread
    a. This request returns a new Thread object, *and* adds to the thread and post count of the parent Topic entry on the Forums page

4. PUT /forum/category/:category/topic/:topicId/thread/:threadId
    a. This request returns a new Post object, *and* adds to the reply count on the parent Thread entry on the Topic page, and post Count of the parent Topic entry on the Forums page

Stanley's Honors HTTP Requests/Routes:

1. GET /private-chat/recent/:userid
    a. This HTTP request returns the list of users in the 'recent conversations' tab for a given user.
    b. This is requested every time the state of the 'recent conversations' tab is changed (via deleting a user or adding a user - see below)
2. POST /private-chat/recent/:userid/add/:otheruserid
    a. This HTTP request adds a user to the 'recent conversations' tab of a given user if the person that the user chose is not already displayed in the tab.
    b. The user has access to this request, which occurs when they select to chat with a user that is not already on the list.
3. DELETE /private-chat/recent/:userid/remove/:otheruserid
    a. This HTTP request removes a user from the 'recent conversations' tab of a given user.
    b. The user has access to this request, which occurs when the 'x' next to one of the entries is clicked.
4. GET /private-chat/live-help/:userid
    a. This HTTP request returns the list of user with the status 'online' or 'away' and displays them in the 'live help' tab of the private chat based on their selected favorite music genre.
5. PUT /private-chat/switch/:userid/to/:otheruserid
    a. This HTTP request switches the chatbox to display a conversation with a different user (once selected from either the recent conversations tab or the live help tab). It does so by updating the 'chattingWith' attribute in the database to the selected user.
    b. The user has access to this request, as each user is the main actor in changing who they wish to chat with.
6. GET /private-chat/conversations/:userid
    a. This HTTP request returns the main chat box of the private chat, initially displaying whoever the user most recently talked to (as well as the chat messages), which is determined by the 'chattingWith' attribute in the database.
7. POST /private-chat/conversations/:userid/create-chat/:otheruserid

    a. This HTTP request creates a new chat log (initially with an empty array of chat messages) when the user selects to chat with another user that they have never chatted with (determined by the ‘chattingWith’ attribute).
    b. The user has access to this request, which occurs when the user selects a new unique user to chat with.
8. POST /private-chat/conversations/:userid/chat-with/:otheruserindex
    a. This HTTP request sends a message to the other user and displays it in the chat box.
    b. This request utilizes the chat-message_schema.json to validate that the message is structured properly to its JSON object.
    c. The user has access to this request, who is the main actor in sending messages to other users.

Report: Special Server Setup Procedure

1. Run npm install in /client and /server
2. Run npm run watch in /client
3. Run node src/server.js in /server
4. For Spotify user this free account (I’m not sure how it would work with other accounts until we set up login for the site):
    a. bbqforte
    b. cs326
    c. *This account will be deleted after the class is done, by the way.

Report: Individual Contributions

Brendan Kelly:

- Collaborated with Matt in attempting to fix a persistent (bugged?) syntax error
- Responsible for the profile page, the GET /user/:userID/nickName, PUT /user/:userID/name and PUT /user/:userID/about requests
- **KNOWN BUGS/ISSUES**: Follow button doesn’t work, Favorite Games not implemented(waiting on Steam API), Favorite Artists not implemented(found it difficult to use Spotfiy)

Jess Hendricks:

- Added the Spotify Web API Node framework.
- Implemented Spotify user authorization.

- Migrated the following playlist, playlist feed, and Spotify song search functions to server/server.js
    - Retrieving the playlists in the playlist feed
    - Searching for songs in Spotify and adding them to a playlist
    - Creating, deleting, editing, and importing playlists (from Spotify).
- Implemented all of the necessary calls to Spotify when a user creates a playlist, adds songs, or removes songs.
    - All of these features are now reflected in the user's Spotify.
- Created the initial schema for playlist and song as well as augmenting the user schema to have fields related to their Spotify accounts.

Ka Wo Fong:

- Fixed the search bar bug where the search terms are not passed down to search result page
- Migrated getCarousel and getNewsUpdate functionality from the client/server.js to server/server.js
- Collaborated with Stanley (peer coding) in his private-chat features
- Set up the general format of this document
- Migrated getNewRelease, getMostPopular, getRising, and getHighestRated from client to server

Logan Rennick:

- Researched the Steam API and started implementing Steam HTTP Requests so we can start pulling information from the Steam Database
- Fixed a lot of startup 4 bugs in the forums section:
- the Forums, Topics, Thread, NewThread, and NewPost pages no longer have hard coded data: forum-row, forum-thread-row, and forum-post-row were created to be used with the map function so the Forum, Topic, and Thread pages are dynamically filled with data from the database. Additionally, the Forum data in the database was restructured to simplify this process.
- The NewThread and NewPost pages now work properly for the most part. Entering data into the text fields and submitting will indeed update the database and the appropriate Topic/Thread pages.
- The Topics page pulls the latest posts from the appropriate Threads, and the NewPost page pulls relevant information from the appropriate Thread for context useful for the user devising their post.
- I also migrated the getForum, getTopic, postThread, and postComment functions from the client to the server
- Finadded database schema for the Thread and Comment

Matthew Zenzie:

- Integrated server changes with settings page
- Added server route for setting user data
- Minor database schema updates

Stanley Lok:

- Fixed previous bugs from startup 4 involving the private chat - clicking on a different user to chat with now immediately switches the chat to that user and adds their name to the ‘recent conversations’ tab if it is not already there
- Created initial schema for posting chat messages validation
- Migrated getUserData functionality from client/server.js to server/server.js (this is used in various components in the web app)
- Migrated all private chat functionality from client side to server side (client/server.js to server/server.js)

Report: Lingering Bugs/ Issues/ Dropped Features

1. Spotify playback is only possible at this time with standalone SDKs such as Android or iOS. As such, we have links that will open the Spotify app or website to the playlist in question.
2. The Spotify login is ugly on Chrome. It opens another tab, then after the login it res.redirect(‘back’) to the home page. We would much rather find a way for it to return to the page it was previously on.. Or even better *close* the login window.
3. The search bar doesn’t re-render new search result when the user is searching for another search term on the search result page.
4. The playlist data still require timestamp for New Release Page to function properly
5. Rising and Highest Rated Page returns the same data because we have yet found a way to determine what quality rising and highest rated playlists have
6. The setUserData function suffers from an “Unexpected syntax error” which despite countless attempts at rewriting and bracket-checking seems to persist. The error doesn’t seem to come from the function itself, as it is syntactically identical to many other such functions but seems to be due to some inconsistency elsewhere. This bug has constrained a great deal of things.
7. Recommendation deletion (without adding the recommendation) was added in the back-end, and will be added to the front-end for the next release. Recommendation generation (triggered by these adds and deletes) will be added as well.
8. The faulty checked state of the settings page persists as well. Ultimately, the page may just be switched over into an entirely other format to avoid this.

9. Profile Page (Brendan): Follow button doesn’t work, Favorite Games not implemented(waiting on Steam API), Favorite Artists not implemented(found it difficult to use Spotfiy),
10. Even though the correct ids are attributed to authors of posts, their userNames do not properly appear on the Thread Page
11. Although schemas were created for the comment and thread objects, when running validate on the schemas within the app.post functions, 400-code errors occured. Thus validate was temporarily removed from the two functions so the fact that they work sans the schema validation can be easily verified and demonstrated.