Aaditya Chopra, Erica Li

COMP 2406 PROJECT CHECK-IN

RESTful Design Outline

1. What types of resources your server will store (e.g., movies, etc.)

Our server will store movie objects (in movies.json), user objects (in users.json), person objects (in persons.json), feed posts objects, notification objects, review objects

2. The URL naming scheme that you will use to identify each resource (e.g., /movies, /movies/{someMovieId}, etc.)

When representing multiple objects of a “category” (movies, users, persons…), the route would be /{category (plural form)}
When representing a specific object of a category, the route would be /{category (plural form)}/{item id}

“/movies” - all movies in the database
“/movies/{movieID}” - a specific movie. movieID is an integer
“/” or “/index” - profile page for logged-in user
“/createMovie” - submit a new movie to the database
“/createPerson” - submit a new person to the database
“/login” - login page
“/index/feed” - JSON representation of the logged-in user’s feed,
“/index/notifications” - JSON representation of the logged-in user’s notifcations,
“/persons” - all persons in the database
“/persons/{personID}” - a specific person. personID is an integer
“/users/{userID}” - a specific user. userID is an integer

3. What types of requests your server will support for each URL (GET, POST, etc.), as well as what the goal of those requests will be (e.g., create a new movie, retrieve a movie, etc.)

3. Type of Requests

GET

These types of requests would be used for whenever the user/client wants to get a resource, such as a page or json data, or any static files.

POST

These types of requests would be used for whenever the user wants to add a new object to the database, such as adding a new movie.

PUT

These types of requests would be used for whenever the client is making any updates to an existing resource, like updating their information.

4. What MIME type(s) each request your server supports will be capable of returning in response (e.g., JSON, HTML, both)

4. MIME Types Supported

With the current implementation of our server, we mainly support the HTML and JSON MIME types, alongside any other static file (such as CSS and PNG files)

HTML

When the server receives a GET request for any page (such as the movies page, users page or people page), the server returns a rendered pug template (which is HTML), or a static HTML page (such as the login page).

JSON

When the server makes a request for the feed on the index page, the server returns JSON, which the client-side javascript then renders and displays the feed data onto the page.

Other MIME Types

Any resources used by the HTML pages (such as CSS or JS files) are handled by the server by looking through the public server. For example, the login page uses two css pages, which is requested under the “styles” folder. The server receives those css page requests as get

requests and returns the static css files with that file path (under the ‘public’ folder which is defined as the root of the statics files). One of the css pages then makes a request to an image (background image). That image is also requested as a get request, which the server returns with the JPG MIME type.

5. Data Each Request is Expected to Contain

GET /movies?q=*:squery*&f=*:searchby*

This request would search through the movies that contain **:squery** in the **:searchby** field (title, genre…)**.** The results would then be displayed on the movies page.

GET /movies/*:mid*

This request would contain the id of the movie that the client wants to view (denoted by **:mid**)

POST /movies

This request would contain the newly created person object that is to be added to the database

POST /reviews

This request would contain the newly created review object that is to be added to the database.

GET /users/*:uid*

This request would contain the id of the user that the client wants to view (denoted by **:uid**)

POST /users

This request would contain the newly created user object that is to be added to the database

PUT /users/*:uid*

This request would contain the user object with the edited attributes. The user with id **:uid** is to be edited in the database.

GET /persons/:*pid*

This request would contain the id of the person that the client wants to view (denoted by **:pid**)

GET /persons?q=:*name*&t=*:type*

This request would search through the persons that have the name **:name** and are a **:type** (i.e. producer, actor, director…)

POST /persons

This request would contain the newly created person object that is to be added to the database.

6. Data Each Response will Contain

GET /login

This route is expected to return the login page, which is a static HTML page. Once sessions are properly implemented, if the user is authenticated/authorized, the user will be redirected to the index page.

GET / or GET /index

This route is expected to return the index page of the currently logged in user, which is a rendered pug template (i.e. an HTML page). Once sessions are properly implemented, if the user is not authenticated/authorized, the user will be redirected to the login page. This will also be true for all the other routes from hereon.

GET /index/feed

This route returns the JSON data that represents the user's timeline/feed data. The client-side javascript code takes care of formatting the json data that is received by the client.

GET /movies?q=:*squery*&f=*:searchby*

This route returns the movies search results page. The database is searched for movie titles containing **:squery** in the **:searchby** field of the movie (title, genre…)

GET /movies/*:mid*

This route would return a rendered pug page of the movie with id **:mid**

GET /users/*:uid*

This route would return a rendered pug page of the users with id **:uid**

Aaditya Chopra, Erica Li

GET /persons?q=:*name*&t=*:type*

This route returns the JSON representation of the person with name **:name** and is of type **:type**

GET /persons/*:pid*

This route returns a rendered pug page of the person with id **:pid**

All POSTs and PUTs

The POST and PUT requests could potentially contain the recently updated or created object (depending on need) and would contain any error message if needed.