IT 202 Project Proposal

Project Name: Movies API Project
Project Summary: This project will primarily use data pull from an API that contains information about movies. It then allows users to create a watchlist with the movies of their choice. Users can create accounts on this website, create/edit watchlists and movie information, and have different roles that give them access to different pages and information.
Github Link:

- **Milestone 1: https://github.com/nafisa37/na569-it202-008/pull/43**
- **Milestone 2: https://github.com/nafisa37/na569-it202-008/pull/63**
- **Milestone 3: https://github.com/nafisa37/na569-it202-008/pull/74**

Website Link: https://na569-prod-74aadc581478.herokuapp.com/project/login.php
API Link: https://rapidapi.com/gox-ai-gox-ai-default/api/ott-details
Video Link: https://www.youtube.com/watch?v=jhZ8DJiK_xQ
Your Name: Nafisa Ahmed

Important: Below is a list of generic core features for everyone's project. Your desired scope/design may be slightly different but each of these requirements must be met to the best of your ability, even if it's just to add an extraneous page to accomplish the feature. The goal of this proposal is to try to fit as many desired project types as possible (which may not be perfectly possible). Each feature is intended to be relatively generic and flexible and can be a little unclear in some cases in how it can fit your project; reach out to me at the earliest if there's any confusion and I can help offer suggestions/guidelines. Due keep in mind your project scope/design/goal as you work towards these features.

Milestone Features:

Milestone 1 (9):

- User will be able to register a new account
  - Form Fields
    - Username, email, password, confirm password(other fields optional)
    - Email is required and must be validated
    - Username is required
    - Confirm password's match
  - Users Table
    - Id, username, email, password (60 characters), created, modified
  - Password must be hashed (plain text passwords will lose points)
  - Email should be unique
  - Username should be unique
  - System should let user know if username or email is taken and allow the user to correct the error without wiping/clearing the form
    - The only fields that may be cleared are the password fields

- User will be able to login to their account (given they enter the correct credentials)
    - Form
        - User can login with **email** or **username**
            - This can be done as a single field or as two separate fields
        - Password is required
    - User should see friendly error messages when an account either doesn't exist or if passwords don't match
    - Logging in should fetch the user's details (and roles) and save them into the session.
    - User will be directed to a landing page upon login
        - This is a protected page (non-logged in users shouldn't have access)
        - This can be home, profile, a dashboard, etc
- User will be able to logout
    - Logging out will redirect to login page
    - User should see a message that they've successfully logged out
    - Session should be destroyed (so the back button doesn't allow them access back in)
- Basic security rules implemented
    - Authentication:
        - Function to check if user is logged in
        - Function should be called on appropriate pages that only allow logged in users
    - Roles/Authorization:
        - Have a roles table (see below)
- Basic Roles implemented
    - Have a Roles table (id, name, description, is_active, modified, created)
    - Have a User Roles table (id, user_id, role_id, is_active, created, modified)
    - Include a function to check if a user has a specific role
- Site should have basic styles/theme applied; everything should be styled
    - I.e., forms/input, navigation bar, etc
- Any output messages/errors should be "user friendly"
    - Any technical errors or debug output displayed will result in a loss of points
- User will be able to see their profile
    - Email, username, etc
- User will be able to edit their profile
    - Changing username/email should properly check to see if it's available before allowing the change
    - Any other fields should be properly validated
    - Allow password reset (only if the existing correct password is provided)
        - Hint: logic for the password check would be similar to login

Milestone 2 (7):

- Define the appropriate table or tables for your API
    - Note: It should have the 3 core columns we'll always be using (id, created, modified) plus additional columns for the incoming API data
        - It's not a valid design decision to use just one column to store the API result as text/JSON/etc, you actually need to do some data mapping
- Data Creation Page
    - Form should have the correct data types for each property being requested
    - Form should have correct validation for each form field
    - Consider how duplicate content is handled
    - Successful creation should have an appropriate message shown
    - Any errors should have user friendly messages shown
    - Manually created entities should be in the same table as the one(s) that is/are populated from the API data
        - Have some column indicator that's used to distinguish between manual and API data (**Hint:** Typically the API will have its own identifier that you'd record as a separate column not directly related to the table's own **id** column)
    - Consider which roles will have access to this form (i.e., is it admin only or can anyone make an entity?)
- Data List Page (many items)
    - Have a page that lists the primary non-user entities of your application
        - Both API generated entities and custom entities must be shown together
    - Consider which roles have access to this page and/or if a user must be logged in to view this page
    - The list page should have logical options for filtering/sorting the results
        - The filter/sort should include a field for the user to specify a limit of records between 1 and 100
            - The server-side should ensure the chosen value is within range or default to 10
        - A filter with no matching records should clearly mention "No results available" or a similar message of your choice
    - Each list item should show a summary of the data you want to show in this view (i.e., likely it won't be the entire entity)
    - Each list item should contain the following links
        - A link to a single view (i.e., a details page)
        - A link to delete (this may be an admin-only thing, but it should be present for the respective role)
        - A link to edit (this may be an admin-only thing, but it should be present for the respective role)

    - Design/Style is your choice but must be applied (i.e., can't just leave it as plaintext dump to the screen)
- View Data Details Page (single item)
    - Entity should be fetched by id passed through query parameters in the URL
    - Invalid id should be redirected back to the list page with an appropriate message
    - Design/Style is your choice but must be applied (i.e., can't just leave it as plaintext dump to the screen)
    - Data shown should be more detailed than that of the list/summary view
    - This page should contain the following links
        - A link to edit the single entity (this may be an admin-only thing, but it should be present for the respective role)
        - A link to delete the single entity (this may be an admin-only thing, but it should be present for the respective role)
- Edit Data Page
    - Entity should be fetch by id passed through query parameters in the URL
    - Invalid id should be redirected back to the list page with an appropriate message
    - Form should be similar to the Create page (**Note**: not all properties should be editable like id, created, modified)
    - Form should prefill with the existing entity's information for the respective fields
    - Form should have the correct data types for each property being requested
    - Form should have correct validation for each form field
    - Successful update should have an appropriate message shown
        - The updated data should be shown in the form
    - Any errors should have user friendly messages shown
- Delete Handling
    - Entity should be fetched by id passed through query parameters in the URL
    - Invalid id should be redirected back to the list page with an appropriate message
    - Handle any necessary roles/permissions (i.e., it's not likely that anyone can delete any entity they choose)
        - Examples:
            - Can only delete entities they created
            - Can only be done by admin
            - Can only delete entities associated with themselves
    - Consider if the deletion of the entity will be a hard delete or a soft delete
    - On successful deletion redirect to the previous page the delete was triggered from with a user friendly success message

        - If deleting from a list page with an active filter/sort applied, the redirect should apply the same filter/sort so the results are the same except without this item showing
- API Handling
    - API data will be fetched from the server-side
    - Consider how you'll transform the data from the API to your table holding the API data
        - Will you be using all the data or just a subset?
    - Consider how duplicate entries are handled (even if unlikely as custom data shouldn't be impacted)
    - Consider how updates to existing entities are handled
        - How will you handle manual updates to API data?
    - Consider how your application will trigger this data fetch
        - Examples:
            - Periodic
            - User triggered via some action
            - Admin triggered via an explicitly click of a button or similar

Milestone 3 (6):

- API Data Association
    - Consider how your API data will be associated with a user
        - Examples:
            - List of favorites
            - Recipe Builder
            - WatchList
            - Purchases
            - Assignment
            - Etc
    - Handling Data Changes
        - When the entity is updated either manually or via the API how is the association affected?
            - Examples:
                - The user sees the old version of the data
                - The user sees the new version of the data
                - The user needs to re-associate the data
                - Etc
- Handle the association of data to a user
    - (Option 1) Update the necessary Pages to include the ability to associate the data with a User
        - **Note:** This is like favorites, shopping cart, wishlist, etc
    - (Option 2) Create a page where the user will have data associated with them by someone else
        - (i.e., in the case of a higher role assigning the association like the UserRoles/Roles logic)

            - **Note:** This is usually when the user can't control their own data like our Roles system
- Logged in user's associated entities page
    - Each line item should summarize relevant information
    - Each line item should include the following links
        - A link to a single view (i.e., a details page)
        - A link to delete (this may be an admin-only thing, but it should be present for the respective role)
            - **Note**: This is to delete the relationship and not the specific entity or user
    - Outside of the list section there should be a link/button to remove all associations to this user
        - This may be an admin only thing, but it should still be present for the respective role
    - The heading of the page should show the total count of items associated to this user
    - The heading of this page should include the total number of items shown on the page
        - This value should be changed based on any applied filter
    - The list page should have logical options for filtering/sorting the results
        - The filter/sort should include a field for the user to specify a limit of records between 1 and 100
            - The server-side should ensure the chosen value is within range or default to 10
        - A filter with no matching records should clearly mention "No results available" or a similar message of your choice
- All Users association page
    - **Note:** This will likely be an admin page and is not the same as the previous item
        - This view is to show multiple associations between entities and many users
    - Each line item should summarize relevant information
        - Include the username this item is associated with
        - Include a column that shows the total number of users the entity is associated with
    - Each line item should include the following links
        - A link to a single view of the entity (i.e., a details page)
        - A link to delete (this may be an admin-only thing, but it should be present for the respective role)
            - **Note**: This is to delete the relationship and not the specific entity or user
        - Clicking the username should redirect to the respective user's profile

    - The heading of the page should show the total count of items associated with users
    - The heading of this page should include the total number of items shown on the page
        - This value should be changed based on any applied filter
    - Include a filter that can filter the result set by partial match of username to show only the matching users' associated items
        - When a filter is applied, there should be a link/button to remove all associations to the matching user(s)
    - The list page should have other logical options for filtering/sorting the results in addition to the required username filter (it's all one form)
        - The filter/sort should include a field for the user to specify a limit of records between 1 and 100
            - The server-side should ensure the chosen value is within range or default to 10
        - A filter with no matching records should clearly mention "No results available" or a similar message of your choice
- Create a page that shows data **not** associated with any user
    - **Note:** This will likely be an admin page and is not the same as the previous item
    - Each line item should summarize relevant information
    - Each line item should include the following links
        - A link to a single view of the entity (i.e., a details page)
    - The heading of the page should show the total count of items not associated to anyone
    - The heading of this page should include the total number of items shown on the page
        - This value should be changed based on any applied filter
    - The list page should have logical options for filtering/sorting the results
        - The filter/sort should include a field for the user to specify a limit of records between 1 and 100
            - The server-side should ensure the chosen value is within range or default to 10
        - A filter with no matching records should clearly mention "No results available" or a similar message of your choice
- Admin can associate any entity with any users
    - **Note:** This may be a form on an existing association page if you rather not have a separate page for this
    - This page should have a form with two fields
        - Entity identifier field (name or some user-friendly property)
            - Partial match
        - Username field
            - Partial match

- Submitting the form will result in a list of the following
    - All partially matched entities (maximum of 25 results)
    - All partially matched users (maximum of 25 results)
    - **Note:** It's likely best to show this as two separate columns
        - **Hint:** similar to the User Role Assignment admin page
    - Each entity and each user will have a checkbox next to them
    - At the top and/or the bottom should be a button to apply the checked associations
- Clicking the association button should apply the associations of the relationship doesn't exist or remove the association if the relationship exists
    - **Hint**: Similar to the User Roles Association admin page