Models & Attributes

movie: title, gross, release_date, mpaa_rating, description

review: comment, star_rating, reviewer_name

Relationships

Movie has many reviews

movie
  has_many :reviews

review
  belongs_to :user

Validations

movie
  title: unique
  gross: defaults to 0
  release_date: year 2010 or later (dropdown?)
  mpaa_rating: G or PG or PG13 or R NC-17
  description: exist

review:
  comment: exist
  (star_rating)
  (username)

Features (User vs. Admin)

user can view all movies
#user can view a movie
POST#user can create a new movie
[user cannot update or delete a movie]

user can view a movie's reviews
POST#user can add a movie review
[user cannot update or delete a review]

admin can view all movies
admin can view a single movie
admin can create a new movie
admin can update a movie
admin can delete a movie

admin can view a movie's reviews
admin can add a movie review
admin can update a review
admin can delete a review

0. Git & GitHub Setup:

- Nicole
    -create a new repo for BananasApi project on Github with a README.md
    -clone repo to local machine
    -in local repo, create the new rails-api app in this directory
- Gina & Cara,
    -fork https://github.com/npupillo/bananas-api
    -clone your new fork to your local machine

Development Workflow, Gina & Cara:

1. Feature branch workflow:

A. Review all the branch names in the repo you just forked:

    $ git branch -a

B. Create a local feature branch to work on, using a unique name that does not already exist from above:

    $ git branch -b joe-feature-branch

C. While in your local feature branch, do your feature work.

D. Continuously add & commit as needed:

    $ git add <file or files>
    $ git commit -m "commit message"

E. Pull from the upstream master to your local master when pull requests from team members are accepted:

    $ git checkout master
    $ git pull upstream master

$ git checkout feature_branch

$ git merge master #merge master into feature_branch

2. Complete the feature branch:
A. When all work in the branch is complete, push your local feature_branch to your GitHub repo:

$ git push origin feature_branch

B. Go to your github account, select your feature_branch in the drop down and make a pull request from that branch.

- Pull request reviewed & accepted b/c you rock! Yay!

C. Then go back to your local master branch & do a git pull from the upstream repo to get any other changes from others

$ git checkout master
$ git pull upstream master

3. Return to Step 1 and start work on a new feature branch.

Development Workflow, Nicole:
1. Feature branch workflow:
A. Review all the branch names in the repo you just forked:

$ git branch -a

B. Create a local feature branch to work on, using a unique name that does not already exist from above:

$ git branch -b joe-feature-branch

C. While in your local feature branch, do your feature work.

D. Continuously add & commit as needed:

$ git add <file or files>
$ git commit -m "commit message"

E. Pull from the upstream master to your local master when pull requests from team members are accepted:

$ git checkout master
$ git pull origin master

merge feature_branch into master & test
$ git merge feature_branch

2. When all work in the branch is complete:
A. Push to origin master branch:

```
$ git push origin master
```

B. Then while still in your local master branch & do a git pull from the upstream repo to get any other changes from others

```
$ git pull origin master
```

3. Return to Step 1 and start work on a new feature branch.

Movie Application Mini-Project

Description

We are going to create an application that will allow users to review movies.

No Authentication or Authorization.

Initially, this app will only have anonymous users. It will **not** implement authentication. At a later date we may decide to add authentication and possibly authorization.

Please, please know the difference between authentication and authorization.

Process

You will follow what is a typical project process or workflow.

Team effort.

This will be a collaborate effort of a team of develelopers. Each member of the team will be given a specific user story to implement.

The project README will list all team members.

Planning

The team will co-operate to determine a plan for this project. The details of this plan are below.

Initial planning will take the most time.

*But, do **NOT** take to much time planning. Guessing about 30 to 40 minutes tops for initial planning*

Maintain User Stories.

These will be kept in the project README.

A list of User Stories will be maintained. Each story will:

- List the implementor/s initials.
- Iteration that story was completed, e.g. Iteration 3.
- Priority, from 1 to 4 with 1 being the highest priority.
- Difficultly, chosen from 1, 2, 4, 8 with 1 being the easiest.

Workflow

One project Github repo will be created for each team. Each team member will typically work in a local feature/topic branch until a feature/story is complete.

- This local feature branch will have a remote tracking branch that will be up to date.
- Squash commits that should be combined before merging the feature branch into the release branch.
- After a feature is completed it *MUST* be merged into the master branch.

Can use git merge or rebase to update the release/master branch with a feature branch.

Iterative process.

Each iteration will be 3 hours long. So, each developer must be assigned one or more stories for a specific iteration.

Note: this will take some planning.

The stories must be prioritized to account for dependencies and *possibly* the importance of the story/feature. For example, one can not implement Reviews before Movies.

After each iteration the feature branches will be merged into the master/release branch. It's at this time that a released version will be stamped using a git tag. *See semantic versioning*

Testing

Each story will require one or more tests. On the backend you'll use using RSpec request specs for these tests.

*You are not required to use TDD but you **MUST** test your code and prove you have a resonable level of code coverage.*

Run rake stats, or some other code coverage tool, to determine test coverage.

Code Review

All code must be reviewed by another developer before it's merged into the master or release branch. *You may want to record who reviewed code in a commit or next to a user story in the README.

Requirements

Movies Stories

Users can:

- View all movies
- View a specific movie.
- Create a new movie with a title, total gross, release date, MPAA rating and description.
- Update a movie.
- Movies must have a title.
- Movie titles must be unique.
- A Movie rating must be a valid MPAA rating.
- The release date may not be greater than the 5 years from the date the movie was created in this app.
- The total gross is 0 by default.

Movie Reviews

- Users can see all the reviews for a specific movie.

- Users can create a new review.
- Users can not update or delete reviews.
- Each review:
    - Must have a comment.
    - May have a star rating between one and five.
    - May have the name of a reviewer.

There can only be one movie review for each **named** reviewer. If the reviewer filled in a name they cannot have more that one review per movie.

Admin users.

Admin users must access movies and reviews via the 'admin' URI namespace.

For example, /admin/movies/5 or /admin/reviews/3

Only Admins are allowed to delete reviews. Only Admins are allowed to edit/update reviews.

Note: There is no authentiation, yet, so being an admin only implies that they access resources via the /admin path.

Bonus Requirements

- Add Versioning to this API.
- Implement Pagination.
- Implement Search and AutoComplete.
- Add features from an online Movie site.
    - TMDb
    - Rotten Tomatoes API
- Populate this app's data from a CSV file. Find a movie data CSV and create a rake task to import it's data into the app DB.
- Get app data from a remote API.
    - TMDb API
    - Rotten Tomatoes API
- Use Active Job to consume another API and populate your DB. This will periodically retrieve new or updated data from the remote api.

Technical Requirements

- Use the Rails API Gem to generate a minimal Rails app that will be used *only* as an API.
- Configure each JSON Resource representation, attributes/properties, using the Active Model Serializer gem.
- Implement Cross-origin resource sharing, CORS, so that Single Page Applications, SPA can access this API.
- Only use ActiveRecord's nested_attributes_for to access reviews. *All reviews should be accessed in the context of a specific movie.*
- Use the annotate gem to annotate you models and tests.
- Optionally, use the rspec-its gem to DRY up tests.

Front-End

Build a Javascript/JQuery front-end. *Make it look perty*