CSE 134B – Summer Session 2019
Homework #5 A Basic Working CRUD App

Due date: Saturday, August 3rd, 12PM (Noon)
NO LATE SUBMISSIONS! We will not be accepting late submissions for HW5

Groups of 1 or 2 are allowed, no larger groups allowed

In this assignment we will continue to explore the underpinnings of a simple CRUD (Create, Read, Update, and Delete) application. In the previous iteration we saved data to a local array we will now save such data to a remote database via an end point we have set up. If you did the previous assignment in an abstract manner you may be able to substitute out code to get things to work.

In the previous assignment you created a file named `crud.html` where you built a small "application" that stored basic information about movies including:

- Movie title
- Release year
- Rating (ex: G, PG, PG-13, etc.)

You will add to this small "application":

- Genre - like sci-fi, comedy, etc.
- Image a picture to associate with the movie such as a poster or logo
- UserRating a number between 0 and 5 (think of this as a quality rating)

You will also have to build some mechanism to login and logout of your simple CRUD application. You can provide a simple creation page as well. You will not have to provide more advanced user management such as roles, admin levels, account recovery, etc. A set of REST endpoints are provided in this assignment's appendix, as well as an overview of how the backend works for your interest.

Like in HW4 you should use JavaScript modules to perform your work to keep things clean and

modern ES6. Do not bother with Babel style transpiling for backwards compatibility.

Required Features

- Account Creation
- Standard CRUD actions (add, edit, delete, list - all and a single movie)
- Access of the REST endpoints via Ajax or Fetch APIs
- Reasonable client-side validation
- A thoughtful and simple user-interface that uses both built-in HTML widgets (<select>, <dialog>, <button>, etc.) as well as appropriate extended widgets you may acquire or write yourself for image upload and rating (star or smiley style).
- A standard web or SPA style design
- Some degree of messaging or fallback in noscript or API failure situations
- Desktop style interface

Extra Credit Features (% is based off of total available points)

- Offline First pattern approach using Service Workers (5%)
- A flashier interface (5%)
- A mobile interface (5%)
- Web Component use (5%)
- If you accomplish all these you can then reimplement the app using Vue or React as a comparison and provide a comparison report on the amount of code, byte count, performance and complexity. (20%)
- If you accomplish all these and have some backend proficiency you can see if it is possible to create a version that works when script has been disabled (20%) You will have to figure out how to host that though.

For Extra Credit Features you MUST list the ones you complete in your README.md in order to receive credit!

Grading and Approach Tips

- Watch the video posted to the slack channel for the idea of the application
- Do not assume the shown interface is good enough for an interface, it is purposefully unstyled. If thoughtful is confusing consider the style of Twitter Bootstrap to be a minimal execution of thoughtfulness.
- The complex widgets (image upload and rating) can be self-written or borrowed with credit by you. Such widgets have to be stand-alone (no framework dependencies) and you are responsible for any issues they might introduce.
- You do not have to make this assignment complex, the previous assignment gets you much of the way there.
- Use only the latest version of Chrome only for development to keep things easy. If more time was available we would make
- Do not transpile or set up a complex tool stack even if you know how, we just need simple plain vanilla JavaScript. If you get to the final extra credit that point is loosened as necessary.
- You do not have to do a SPA style to execute this application, but it is encouraged for practice. The fallback version may be difficult with this style though.
- Your code MUST pass both the CSS and HTML validators. Make sure to check all your efforts using the w3c HTML validator (https://validator.w3.org/) and CSS validator (https://jigsaw.w3.org/css-validator/). If your various pages do not validate you must provide HTML comments or CSS comments in your submission indicating the validation error and why it has occurred just as you did in the previous assignment.

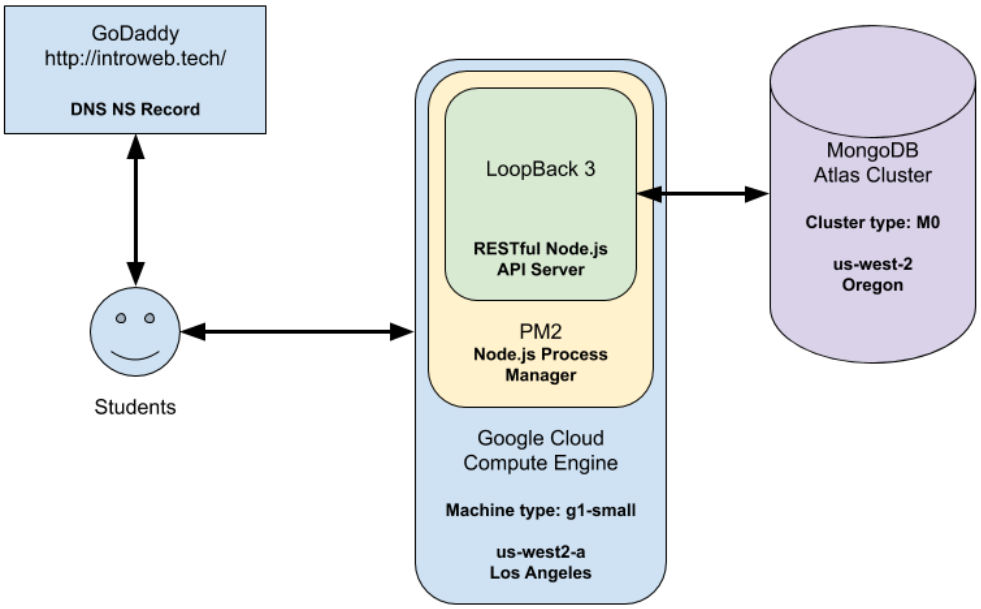
Your Gradescope submission should include:

- You should link this final effort to your notebook to keep all your efforts together and provide any appropriate documentation there. If there is some reason to host it elsewhere please make sure that you also note the external link in a submission.
- **README.md** with your name, PID, a link to your solution and any notes you think that will help the grader to make sense of your solution **(Including a list of any extra credit items you completed)**
- **link.md** a file containing ONLY the link to your solution
    - Should not include https:// or / at the end, do NOT use .webapp link
    - i.e. example.com NOT https://example.com/

- **ALL files you CREATED for this assignment**
- If you work with a partner you only need to submit one assignment on Gradescope, you can add your partner when you submit. Don't forget!

Appendix: RESTful Mock Server Design

System Overview



System diagram

Available Endpoints

Base API URL: http://introweb.tech/api

Authentication Database				
URL Endpoint	Description	HTTP Method	Expected Request Body	Expected Response

`/Users`	Signup a new user	`POST`	`x-www-form-urlencoded`
			
			`"username": "string"`
			`"email": "string"`
			`"password": "string"`
				A JSON object contains:
				registered username
				email
				userId

`/Users/login`	Login user	`POST`	`x-www-form-urlencoded`
			
			`"username": "string"`
			`"password": "string"`
			
			`OR`
			
			`"email": "string"`
			`"password": "string"`
				A JSON object contains:
				id (session id, which is access_token)
				ttl (time to live)
				created
				userId

`/Users/logout?access_token=`	Logout user	`POST`	empty	empty

Movies Database				
Url Endpoint	Description	HTTP Method	Expected Request Body	Expected Response
`/movies/movieList?access_token=`	Retrieve all movies	`GET`	empty	A JSON object contains all movies information

`/movies?access_token=`	Add new movie	`POST`	`x-www-form-urlencoded`
			
			`"title": "string",`
			`"year": number,`
			`"genre": "string",`
			`"rating": "string",`
			`"userRating": number,`
			`"image": "string",`
				A JSON object contains the new movie information

`/movies/{id}?access_token=`	Get movies by movie id	`GET`	empty	A JSON object contains the information about movie
`/movies/{id}?access_token`	Delete	`DELETE`	empty	the number of deletion

=	target movie			
/movies/{id}/replace?access_token=	Update movie	POST	x-www-form-urlencoded "title": "string", "year": number, "genre": "string", "rating": "string", "userRating": number, "image": "string",	A JSON object contains the updated movie information