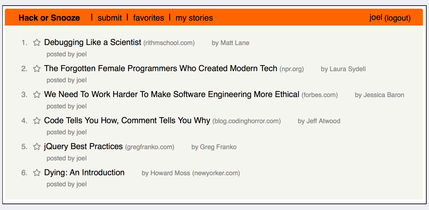
AJAX with jQuery Exercise: Hack-or-Snooze

In this exercise, you'll add features to a news-aggregator site (based loosely on a popular one called Hacker News). It will allow users to create accounts, log in, create articles, mark articles as favorites, and more!



We've already built the backend server API, so you'll focus on learning to use an API and adding features to the front-end Javascript.

Part 0: Explore Working Version and API

Explore the working copy of our solution. It will help you to try how the app works, and what features you'll build *before* digging into the source code.

Note: Turn on your browser console!

In the browser console, you'll see a message explaining how the front-end app can show you useful debugging messages — those will help you get a handle on how the app works, so do what it says :)

Once you've had a chance to try out the app, you should learn about our API. The API docs are at quickstart. Read the first section and try out some API calls in Insomnia or curl. You don't need to read and understand *everything* about the API right now, but get a sense of the basics before moving on.

Part 1: Explore the Starter Code

Download starter code

Download the starter code and start it with `python3 -m http.server`. You can then visit the site at ***http://localhost:8000/***.

You will see that stories are displayed and there is functionality to log in and create a user. (Later, you'll write the features to let users add new stories, favorite a story, and delete a story.)

Our front-end app consists of two parts:

- Classes and methods for the big data ideas: a ***Story*** class for each story, a ***StoryList*** class for the list of stories, and a ***User*** class for the logged-in user (if any). These methods also handle interacting with the API.
- Functions for the UI, handling things like reading form values from forms and manipulating the DOM.

Note: Separation of Concerns and Organization

We've divided the code up into those different parts for readability and maintenance. It's often useful to think about the data and the UI separately, (a *separation of concerns*). Many apps are written this way.

There's one JS file for the "data" layer of the app:

js/models.js

contains classes to manage the data of the app and the connection to the API. The name *models.js* to describe a file containing these kinds of classes that focus on the data and logic about the data. UI stuff shouldn't go here.

Read this file thoroughly. There is a new keyword here, ***static***. Make sure you understand what it means before moving on.

For the UI layer, we've broken this into several files by topic:

js/main.js
contains code for starting the UI of the application, and other miscellaneous things.

js/user.js
contains code for UI about logging in/signing up/logging out, as well as code about remembering a user when they refresh the page and logging them in automatically.

js/stories.js
contains code for UI about listing stories.

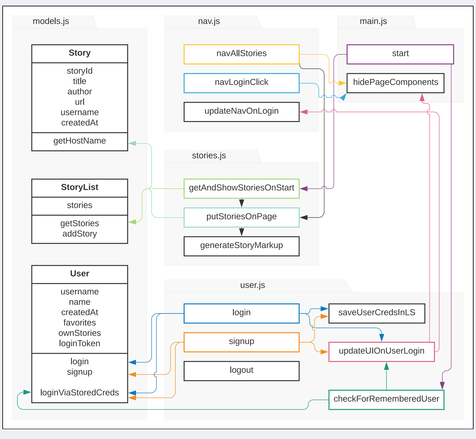
js/nav.js
contains code to show/hide things in the navigation bar, and well as code for when a user clicks in that bar.

Preparing to Read the Code

When meeting a new codebase, be thoughtful about *how* to read the code. It's usually *not* helpful to just read everything in detail, top to bottom. You won't remember it all, and it won't help you understand what the pieces are and how they fit together.

Instead, in the beginning, think about *skimming* the codebase first to just see what the classes and functions are. Look at which functions call other functions. Read the documentation comments before a function or class to get an idea of what it should do and return.

It can be very helpful to make a pen-and-paper drawing of the names of the important functions and how the call the other functions.



Part 2: Creating New Stories

In this part, you'll design and write the functionality to let logged-in users add new stories. We've broken this task into two parts. It will help you to tackle them in this order.

Subpart 2A: Sending Story Data to the Backend API

Here, you'll need to write a method that adds a new story by sending the right data to our API.

We've given you a comment string and a stub method for this, ***addStory***, in the ***StoryList*** class. Complete this function, making sure your function takes in the same parameters and returns the same result as our comment said.

Test that this works, and that your method returns an instance of ***Story***. You can do this in the browser console with:

```
let newStory = await storyList.addStory(currentUser,
  {title: "Test", author: "Me", url: "http://meow.com"});
```

And make sure that returns an instance of the ***Story*** class:

```
newStory instanceof Story;      // should be true!
```

Subpart 2B: Building The UI for New Story Form/Add New Story

Now, we'll add the UI for the story-adding feature:

- Add a form in the HTML for the story. This should initially be hidden.
- Add a link in the navbar with the text of "submit".
- Write a function in ***nav.js*** that is called when users click that navbar link. Look at the other function names in that file that do similar things and pick something descriptive and similar.
- Write a function in ***stories.js*** that is called when users submit the form. Pick a good name for it. This function should get the data from the form, call the ***.addStory*** method you wrote, and then put that new story on the page.

Part 3: Favorite stories

In this step, you'll add a feature marking/unmarking a story as a favorite.

As before, it's best to write the data-logic and API-call part first, and do the UI afterwards.

Subpart 3A: Data/API Changes

Allow logged in users to "favorite" and "un-favorite" a story. These stories should remain favorited when the page refreshes.

Allow logged in users to see a separate list of favorited stories.

The methods for adding and removing favorite status on a story should be defined in the User class.

Part 4: Removing Stories

Allow logged in users to remove a story. Once a story has been deleted, remove it from the DOM and let the API know its been deleted.

Further Study

- Add some error handling for when a username has already been taken or if credentials are incorrect!
- Allow users to edit stories they have created.
- Add a section for a "user profile" where a user can change their ***name*** and ***password*** in their profile.
- Style the application so that it is presentable on mobile devices.
- Add infinite scroll! When a user scrolls to the bottom of the page, load more stories.
- Come up with some other features you can build using what our Hack or Snooze API makes available to you!

Solution

View our solution