# Hacker News / Hack-or-Snooze
# Project Notes

## JavaScript File #1. Models.js

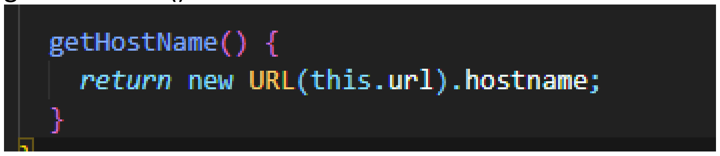
There's one JS file for the "data" layer of the app – models.js. This contains classes to manage the data of the app and the connection to the API. The name models.js to describe a file containing these kinds of classes that focus on the data and logic about the data. UI stuff shouldn't go here.

**BASE_URL –** Server
**Class #1 – Story**
Make Instance of Story from Data Object About Story:
{title, author, url, username, storyId, createdAt)
Constructors:

- storyID
- title
- author
- url
- username
- createdAt

Methods:

- getHostName() – Parses hostname out of URL and returns it.



```
getHostName() {
  return new URL(this.url).hostname;
}
```

**Class #2 – StoryList**
List of Story Instances – Used by UI to Show Story Lists in DOM
Constructors:

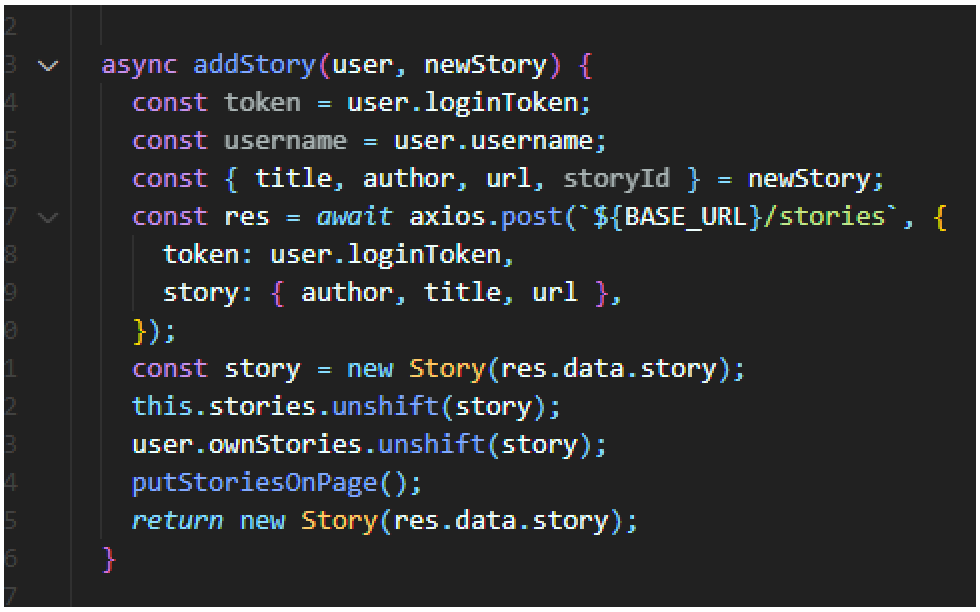
- stories

Generates a new StoryList – It does the following:

- calls the API
  builds an array of story instances
- makes a single StoryList out of that
- Returns the storyList instance

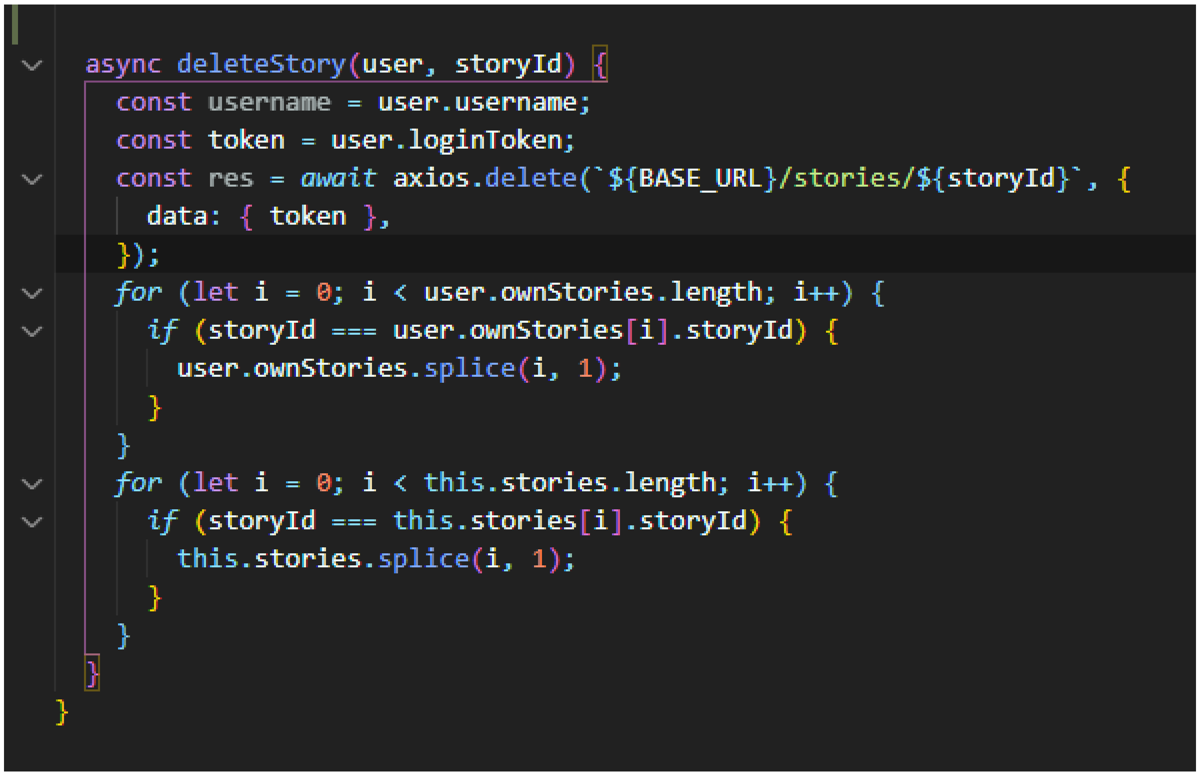
Methods:

- **getStories** – Notice the presence of "static" keyword – this indicates that getStories is **not** an instance method. Rather, it is a method that is called on the class directly. Why doesn't it make sense for getStories to be an instance method? *Assuming because it relates to ALL the stories and not just the one instance of the story!*
  - Response – Queries the /stories endpoint from the Base API URL using Get Method
  - Stories – Turns Plain Old Story Objects from API into New Instances of Story Class by using Map method from **response.data.stories**.
  - Returns a **new StoryList**
- **addStory** – Adds story to API, makes a story instance, and adds it to the story list.
  - User – the current instance of User who will post the story obj of {title, author, url}
  - Returns **new Story**



```
async addStory(user, newStory) {
  const token = user.loginToken;
  const username = user.username;
  const { title, author, url, storyId } = newStory;
  const res = await axios.post(`${BASE_URL}/stories`, {
    token: user.loginToken,
    story: { author, title, url },
  });
  const story = new Story(res.data.story);
  this.stories.unshift(story);
  user.ownStories.unshift(story);
  putStoriesOnPage();
  return new Story(res.data.story);
}
```

- **deleteStory** – Delete's a user both from their ownStory array and from the Stories storage in the API.



```
  async deleteStory(user, storyId) {
    const username = user.username;
    const token = user.loginToken;
    const res = await axios.delete(`${BASE_URL}/stories/${storyId}`, {
      data: { token },
    });
    for (let i = 0; i < user.ownStories.length; i++) {
      if (storyId === user.ownStories[i].storyId) {
        user.ownStories.splice(i, 1);
      }
    }
    for (let i = 0; i < this.stories.length; i++) {
      if (storyId === this.stories[i].storyId) {
        this.stories.splice(i, 1);
      }
    }
  }
}
```

**Class #3 – User**
Make User instance from object of user data and a token {username, name, createdAt, favorites[], ownStories[]}, token
Constructors:

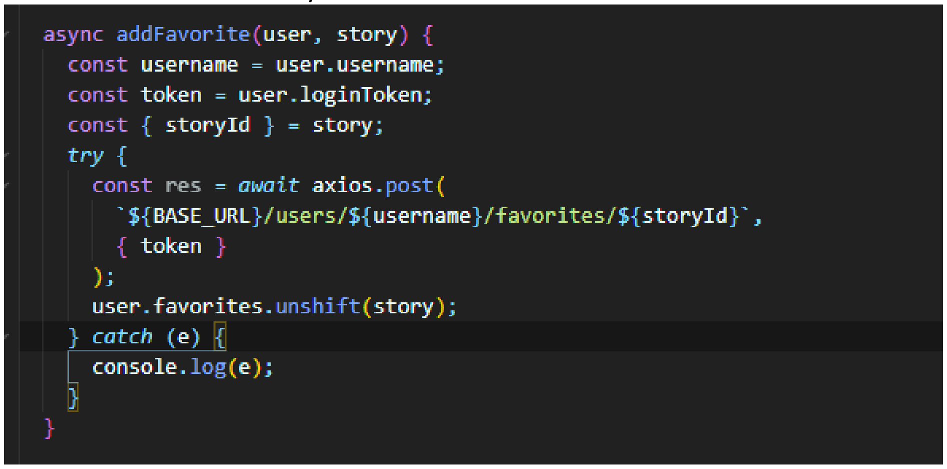
- Object consisting of {username, name, createdAt, favorites [], ownStories []}
- Authentication Token

Instance Variables:

- Username, name, createdAt
- favorites - – instantiate Story instances for user's favorites via map
- ownStories – instantiate Story instances for user's ownStories via map
- loginToken – store the login token of the user so it's easy to find for API calls.

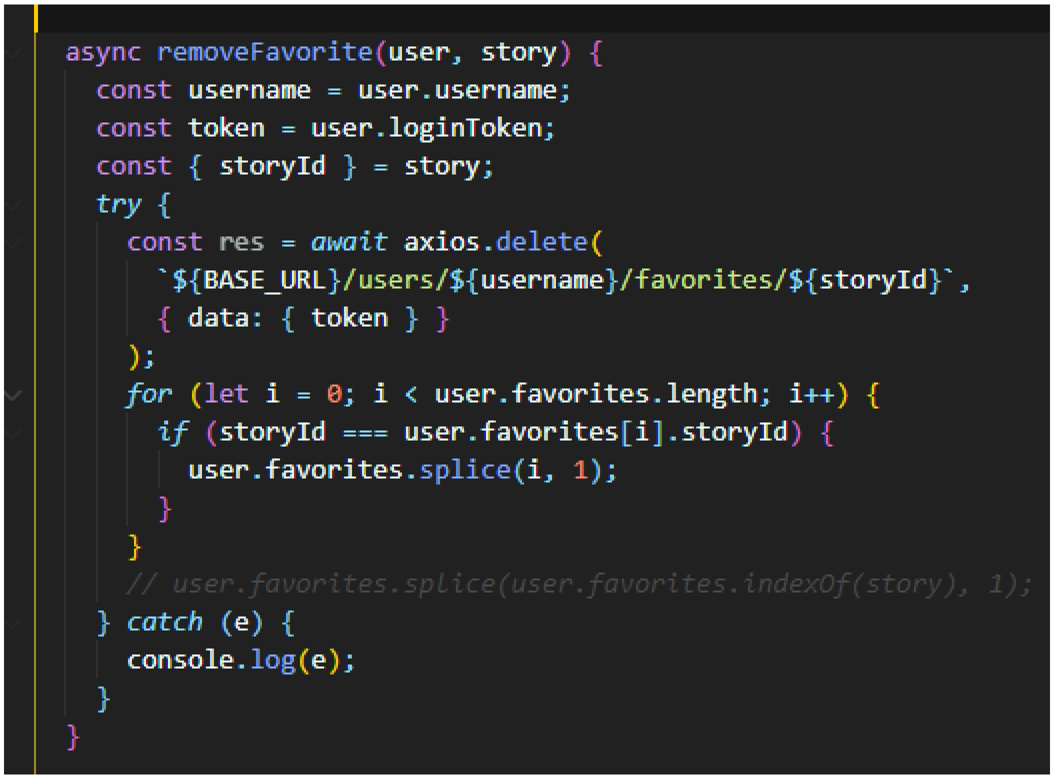
Methods:

- signup – Register new user in API, make **new User** instance and return it:
    - username: a new username
    - password: a new password
    - name: the user's full name
- login – log in user with API, make new User instance, and return it.
    - username: an existing user's username
    - password: an existing user's password
- loginViaStoredCredentials – When we already credentials (token and username) for a user, we can log them in automatically. This function does that.
- addFavorite – adds a story to the user's favorites.



```
async addFavorite(user, story) {
  const username = user.username;
  const token = user.loginToken;
  const { storyId } = story;
  try {
    const res = await axios.post(
      `${BASE_URL}/users/${username}/favorites/${storyId}`,
      { token }
    );
    user.favorites.unshift(story);
  } catch (e) {
    console.log(e);
  }
}
```

- removeFavorite – removes a story from a user's favorites.



```
async removeFavorite(user, story) {
  const username = user.username;
  const token = user.loginToken;
  const { storyId } = story;
  try {
    const res = await axios.delete(
      `${BASE_URL}/users/${username}/favorites/${storyId}`,
      { data: { token } }
    );
    for (let i = 0; i < user.favorites.length; i++) {
      if (storyId === user.favorites[i].storyId) {
        user.favorites.splice(i, 1);
      }
    }
    // user.favorites.splice(user.favorites.indexOf(story), 1);
  } catch (e) {
    console.log(e);
  }
}
```

**JavaScript File #2 – main.js** - contains code for starting the UI of the application, and other miscellaneous things.

**jQuery Variables:**
$body
$storiesLoadingMsg - loading message (removed by JS after stories loaded)
$allStoriesList – list of all stories
$faveStoriesList – favorite stories
$userStoriesList – user’s own stories

$loginForm – hidden by default
$signupForm – hidden by default
$submitForm – area where user can submit a story; also hidden by default

Navbar Variables:
$navLogin – login/signup link on top of navigation bar
$navUserProfile – link to profile for user
$navLogout – logout link
Added NavBar Variables:
$navSubmit – link to submit form
$navFavorites – link to user’s favorites
$navStories – link to user’s own stories

**Function #1 – hidePageComponents** – To make it easier for individual components to show just themselves, this is a useful function that hides pretty much everything on the page using hide(). After calling this, individual components can re-show just what they want.
**Function #2 – start** – Overall function to kick off the app

- "Remember logged-in user" and log-in, if credentials are in localStorage
- If we found a logged-in user, updateUIOnUserLogin;
- Once the DOM is entirely loaded, begin the app.

**JavaScript File #3 – 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.

Initialization variable – **currentUser** – global variable to hold the User instance of the currently logged in user.
**Function #1 – login** – Handle login form submission. If login OK, sets up the user instance.

- Grabs username and password.
- User.login retrieves user info from API and returns User instance which we'll make the globally available, logged-in user.
- Event listener below function ***$loginForm.on("submit", login);***

**Function #2 – signup** – Handles signup for submission.

- User.signup retrieves user info from API and returns User instance which we'll make the globally-available, logged-in user.
- Event listener below function ***$signupForm.on("submit", signup);***

**Function #3 – logout** – Handle click of logout button. Remove user's credentials from localStorage and refreshes page

- Event listener below function: ***$navLogOut.on("click", logout);***

**Function #4 – checkForRememberedUser** – Storing/recalling previously logged-in user with localStorage. If there are user credentials in local storage, it will use those to log in that user. This is meant to be called on page load, just once.

- Tries to log in with the credentials in localStorage (will be null if login failed)

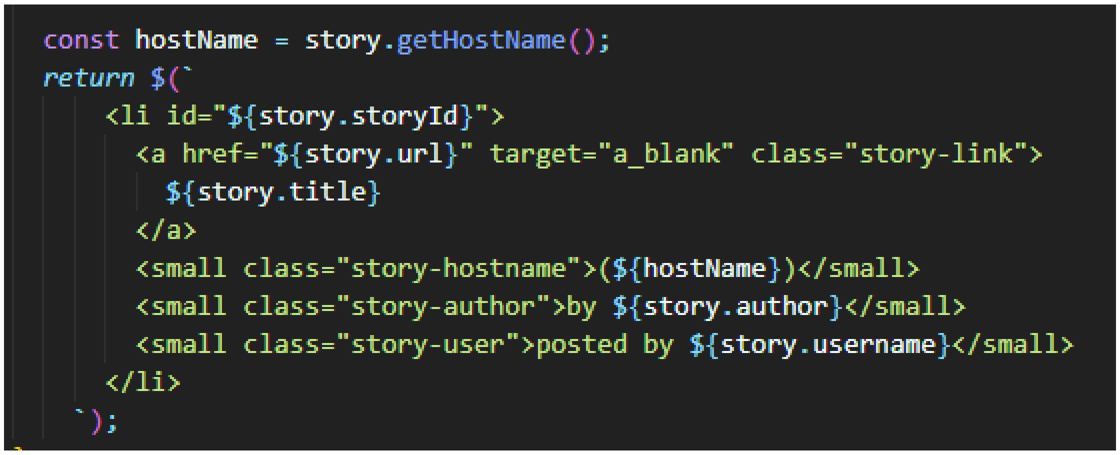
**Function #5 – saveUserCredentialsInLocalStorage** – Sync current user information to localStorage. We store the username/credential-token in localStorage so when the page is refreshed (or the user revisits the site later), they will still be logged in.
**Function #6 – updateUIOnUserLogin** - General UI stuff about Users. When a user signs up or registers, we want to set up the UI for them:

- Show the Stories List
- Update Nav Bar Options for Logged-in User
- Generate User Profile part of page.

**JavaScript File #4 – stories.js** - contains code for UI about listing stories.

Initialization Variable: storyList – This is the global list of the stories, an instance of StoryList

**Function #1 – getAndShowStoriesOnStart** – Get and show stories when the site first loads using the putStoriesOnPage function
**Function #2 – generateStoryMarkup** – A render method to render HTML for an individual Story instance. Story variable represents an instance of Story. Returns the markup for the story.

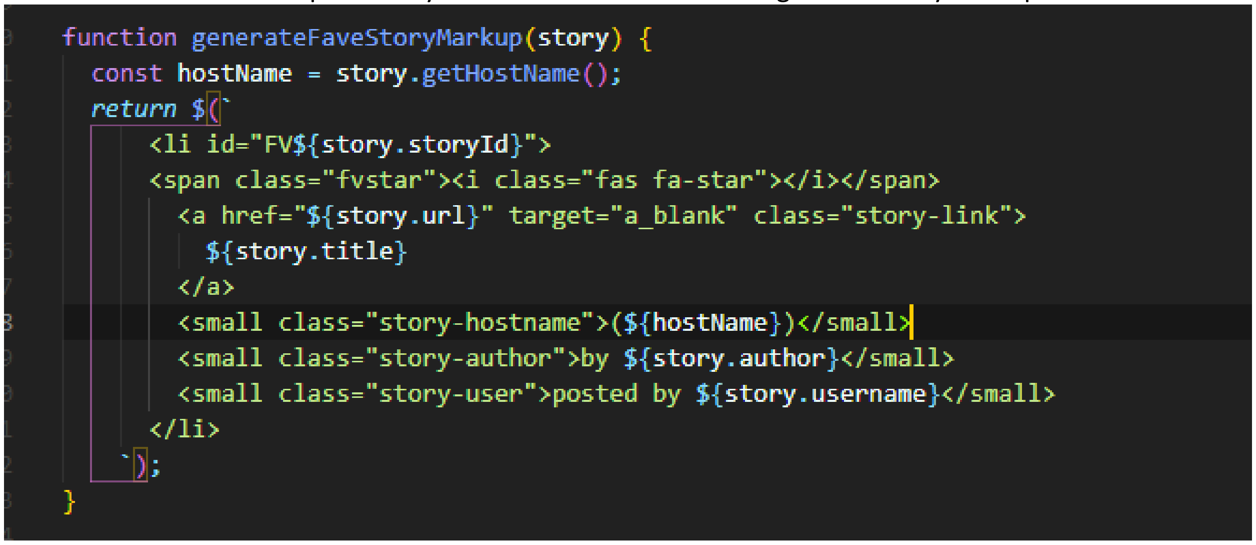


```
  const hostName = story.getHostName();
  return $(`
      <li id="${story.storyId}">
        <a href="${story.url}" target="a_blank" class="story-link">
          ${story.title}
        </a>
        <small class="story-hostname">(${hostName})</small>
        <small class="story-author">by ${story.author}</small>
        <small class="story-user">posted by ${story.username}</small>
      </li>
    `);
```

**Function #3 – putStoriesOnPage** – Gets list of stories from server, generates their HTML, and puts them on the page.

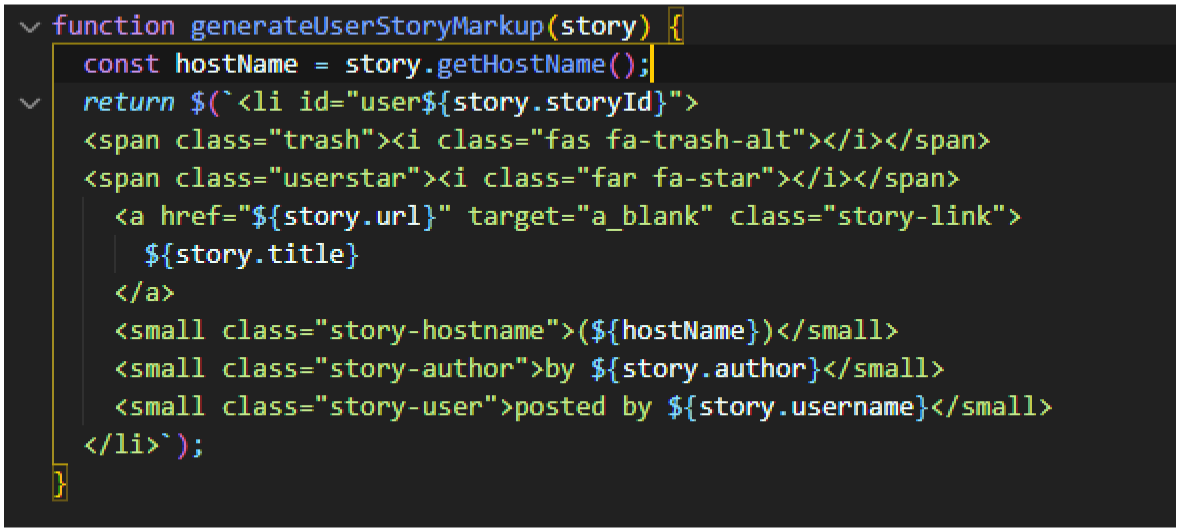
- Loops through all of our stories and generates HTML for them using the generateStoryMarkup function.

**Function #4 – generateFaveStoryMarkup(story)** – similar to generateStoryMarkup but for user favorite stories section. Could be potentially refactored later into earlier generateStoryMarkup section.



```
function generateFaveStoryMarkup(story) {
  const hostName = story.getHostName();
  return $(`
      <li id="FV${story.storyId}">
      <span class="fvstar"><i class="fas fa-star"></i></span>
        <a href="${story.url}" target="a_blank" class="story-link">
          ${story.title}
        </a>
        <small class="story-hostname">(${hostName})</small>
        <small class="story-author">by ${story.author}</small>
        <small class="story-user">posted by ${story.username}</small>
      </li>
    `);
}
```

**Function #5 – generateUserStoryMarkup(story)** – also similar to generateStoryMarkup, but for user's own stories



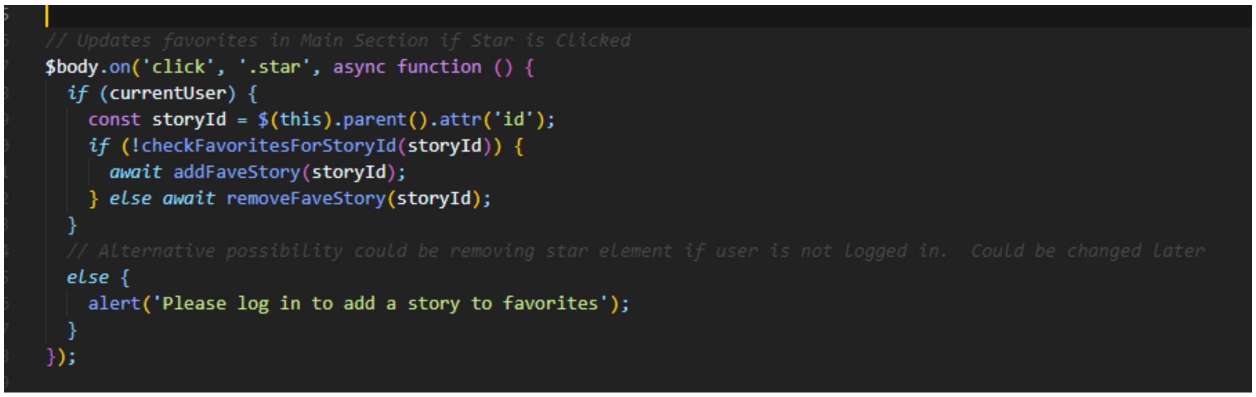
```
function generateUserStoryMarkup(story) {
  const hostName = story.getHostName();
  return $(`<li id="user${story.storyId}">
  <span class="trash"><i class="fas fa-trash-alt"></i></span>
  <span class="userstar"><i class="far fa-star"></i></span>
    <a href="${story.url}" target="a_blank" class="story-link">
      ${story.title}
    </a>
    <small class="story-hostname">(${hostName})</small>
    <small class="story-author">by ${story.author}</small>
    <small class="story-user">posted by ${story.username}</small>
  </li>`);
}
```

**Function #6 – putFaveStoriesOnPage** – Updates list of favorite stories in DOM
**Function #7 – showFaveStories** – Shows Favorite Stories and hides other site components
**Function #8 – showUserStories** – shows user stories and hides other site components
**Function #9 – addFaveStory** – Adds story to favorites based on storyID
**Function #10 – removeFaveStory** – Removes a story from user's favorites based on storyId
**Function #11 – submitStory** – Allows user to submit story via submit form
**Function #12** – **deleteStory** Allows user to delete story
**Function #13 – checkFavoritesForStoryId** – Checks if story is in user's favorites based on storyId
**Function #14 – getStoryById** – Pulls story information from API based on the storyId

**JavaScript File #5 – nav.js** – Contains code to show/hide things in the navigation bar, as well as code for when a user clicks in that bar. Also contains code for when user clicks on elements on the page such as star or trash can icon.

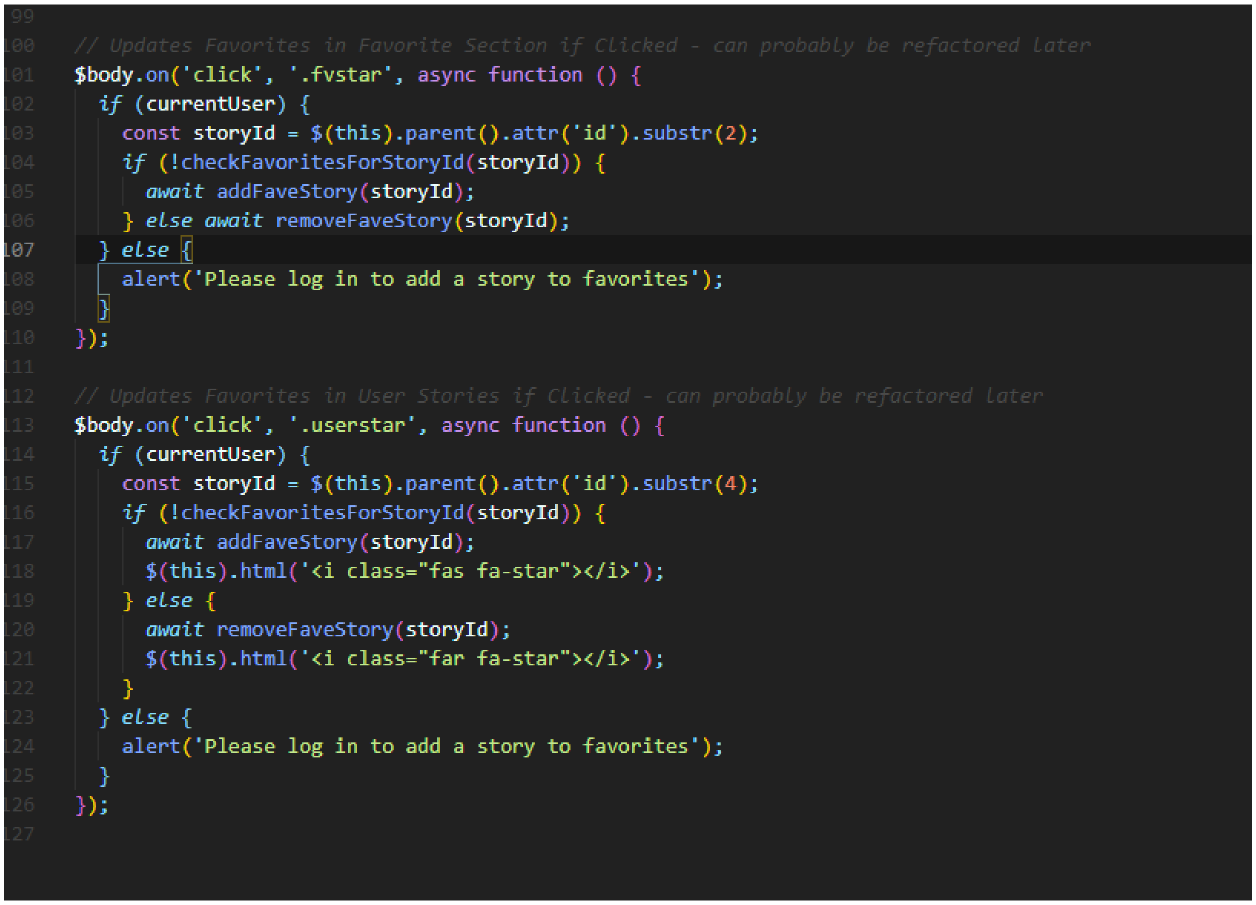
Handling navbar clicks and updating navbar:
**Function #1 – navAllStories** – Show main list of all stories when user clicks site name.
**Function #2 – navLoginClick** – Show login/signup form on click of "login"
**Function #3 – updateNavOnLogin**- When a user first logs in, update the navbar to reflect that – Showing navbar elements (.main-nav-links) such as Submit, Favorites, My Stories. Hide $navLogin. Show $navLogOut. Show user profile.
**Function #4 – navFaveStories** - Show User's Favorite Stories when user clicks "Favorites"
**Function #5 – navUserStories** - Show User's Own Stories when user clicks My Stories
**Function #6 – navLogOutClick** - Remove Main Navbar on click on "logout"
**Function #7 – displayStorySubmitForm -** Display Submit Story Form on click of Submit Link

**Function #8/#9 –** turnFaveStarOn/turnFaveStarOff – Toggles Star on Main Stories List



```javascript
// Updates favorites in Main Section if Star is Clicked
$body.on('click', '.star', async function () {
  if (currentUser) {
    const storyId = $(this).parent().attr('id');
    if (!checkFavoritesForStoryId(storyId)) {
      await addFaveStory(storyId);
    } else await removeFaveStory(storyId);
  }
  // Alternative possibility could be removing star element if user is not logged in.  Could be changed later
  else {
    alert('Please log in to add a story to favorites');
  }
});
```

Event Listeners for .fvstar and .userstar with similar behavior:



```javascript

// Updates Favorites in Favorite Section if Clicked - can probably be refactored later
$body.on('click', '.fvstar', async function () {
  if (currentUser) {
    const storyId = $(this).parent().attr('id').substr(2);
    if (!checkFavoritesForStoryId(storyId)) {
      await addFaveStory(storyId);
    } else await removeFaveStory(storyId);
  } else {
    alert('Please log in to add a story to favorites');
  }
});

// Updates Favorites in User Stories if Clicked - can probably be refactored later
$body.on('click', '.userstar', async function () {
  if (currentUser) {
    const storyId = $(this).parent().attr('id').substr(4);
    if (!checkFavoritesForStoryId(storyId)) {
      await addFaveStory(storyId);
      $(this).html('<i class="fas fa-star"></i>');
    } else {
      await removeFaveStory(storyId);
      $(this).html('<i class="far fa-star"></i>');
    }
  } else {
    alert('Please log in to add a story to favorites');
  }
});

```