Hack or Snooze Solution

Download our solution <../../hack-or-snooze-ajax-api-solution.zip>

Markup

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Hack or Snooze</title>
  <link rel="stylesheet"
        href="https://use.fontawesome.com/releases/v5.3.1/css/all.css"
        integrity="sha384-mzrmE5qonljUremFsqc01SB46JvROS7bZs3IO2EmfFsd15uHvIt+Y8vEf7N7fWAU"
        crossorigin="anonymous">
  <link rel="stylesheet" href="style.css" type="text/css">
</head>

<body>
<nav>
  <b><a class="nav-link navbar-brand" href="#" id="nav-all">Hack or Snooze</a></b>
  <section class="main-nav-links hidden">
    <span>|</span>
    <a class="nav-link" href="#" id="nav-submit">submit</a>
    <span>|</span>
    <a class="nav-link" href="#" id="nav-favorites">favorites</a>
    <span>|</span>
    <a class="nav-link" href="#" id="nav-my-stories">my stories</a>
  </section>
  <a class="nav-link nav-right" href="#" id="nav-login">login/create user</a>
  <span class="nav-link nav-right hidden" id="nav-welcome">
      <a href="#" id="nav-user-profile"></a>
    </span>
  <a class="hidden" id="nav-logout" href="#">
    <small>(logout)</small>
  </a>
</nav>
<section class="articles-container container">
```

```
  <form action="" class="hidden" id="submit-form">
    <div>
      <label for="author">author</label>
      <input id="author" required type="text" placeholder="author name">
    </div>
    <div>
      <label for="title">title</label>
      <input id="title" required type="text" placeholder="article title">
    </div>
    <div>
      <label for="url">url</label>
      <input id="url" required type="url" placeholder="article url">
    </div>
    <button type="submit">submit</button>
    <hr>
  </form>
  <article>
    <ol id="all-articles-list" class="articles-list">
      <h3>Loading...</h3>
    </ol>
  </article>
  <article>
    <ul id="favorited-articles" class="hidden articles-list">
    </ul>
  </article>
  <article>
    <ul id="filtered-articles" class="hidden articles-list">
    </ul>
  </article>
  <article>
    <ul id="my-articles" class="hidden articles-list">
    </ul>
  </article>
</section>
<section class="edit-article-container container">
  <form action="#" id="edit-article-form" class="hidden" method="post">
    <h4>Article Information</h4>
    <div class="edit-input">
      <label for="edit-title">Title</label>
      <input id="edit-title" type="text">
      <button type="submit">change</button>
    </div>
  </form>
</section>
<section class="account-forms-container container">
```

```
  <form action="#" id="login-form" class="account-form hidden" method="post">
    <h4>Login</h4>
    <div class="login-input">
      <label for="login-username">username</label>
      <input id="login-username" type="text">
    </div>
    <div class="login-input">
      <label for="login-password">password</label>
      <input id="login-password" type="password">
    </div>
    <button type="submit">login</button>
    <hr>
  </form>
  <form action="#" id="create-account-form" class="account-form hidden"
        method="post">
    <h4>Create Account</h4>
    <div class="login-input">
      <label for="create-account-name">name</label>
      <input id="create-account-name" type="text">
    </div>
    <div class="login-input">
      <label for="create-account-username">username</label>
      <input id="create-account-username" type="text">
    </div>
    <div class="login-input">
      <label for="create-account-password">password</label>
      <input id="create-account-password" type="password">
    </div>
    <button type="submit">create account</button>
  </form>
</section>
<section id="user-profile" class="container hidden">
  <h4>User Profile Info</h4>
  <section>
    <div id="profile-name">Name:</div>
    <div id="profile-username">Username:</div>
    <div id="profile-account-date">Account Created:</div>
  </section>
</section>
</body>
<script
  src="https://code.jquery.com/jquery-3.4.1.js"
  integrity="sha256-WpOohJOqMqqyKL9FccASB9O0KwACQJpFTUBLTYOVvVU="
  crossorigin="anonymous"></script>
<script src="https://unpkg.com/axios/dist/axios.js"></script>
```

```
<script src="api-classes.js"></script>
<script src="ui.js"></script>

</html>
```

API

The OOP Part

```
const BASE_URL = "https://hack-or-snooze-v3.herokuapp.com";

/**
 * This class maintains the list of individual Story instances
 *  It also has some methods for fetching, adding, and removing stories
 */

class StoryList {
  constructor(stories) {
    this.stories = stories;
  }

  /**
   * This method is designed to be called to generate a new StoryList.
   *  It:
   *  - calls the API
   *  - builds an array of Story instances
   *  - makes a single StoryList instance out of that
   *  - returns the StoryList instance.*
   */

  // TODO: Note 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?

  static async getStories() {
    // query the /stories endpoint (no auth required)
    const response = await axios.get(`${BASE_URL}/stories`);

    // turn plain old story objects from API into instances of Story class
    const stories = response.data.stories.map(story => new Story(story));

    // build an instance of our own class using the new array of stories
```

```
    const storyList = new StoryList(stories);
    return storyList;
  }

  /**
   * Method to make a POST request to /stories and add the new story to the list
   * - user - the current instance of User who will post the story
   * - newStory - a new story object for the API with title, author, and url
   *
   * Returns the new story object
   */

  async addStory(user, newStory) {
    const response = await axios({
      method: "POST",
      url: `${BASE_URL}/stories`,
      data: {
        // request body
        // this is the format specified by the API
        token: user.loginToken,
        story: newStory,
      }
    });

    // make a Story instance out of the story object we get back
    newStory = new Story(response.data.story);
    // add the story to the beginning of the list
    this.stories.unshift(newStory);
    // add the story to the beginning of the user's list
    user.ownStories.unshift(newStory);

    return newStory;
  }

  /**
   * Method to make a DELETE request to remove a particular story
   *  and also update the StoryList
   *
   * - user: the current User instance
   * - storyId: the ID of the story you want to remove
   */

  async removeStory(user, storyId) {
    await axios({
      url: `${BASE_URL}/stories/${storyId}`,
```

```
      method: "DELETE",
      data: {
        token: user.loginToken
      },
    });

    // filter out the story whose ID we are removing
    this.stories = this.stories.filter(story => story.storyId !== storyId);

    // do the same thing for the user's list of stories
    user.ownStories = user.ownStories.filter(s => s.storyId !== storyId
    );
  }
}

/**
 * The User class to primarily represent the current user.
 *  There are helper methods to signup (create), login, and getLoggedInUser
 */

class User {
  constructor(userObj) {
    this.username = userObj.username;
    this.name = userObj.name;
    this.createdAt = userObj.createdAt;
    this.updatedAt = userObj.updatedAt;

    // these are all set to defaults, not passed in by the constructor
    this.loginToken = "";
    this.favorites = [];
    this.ownStories = [];
  }

  /* Create and return a new user.
   *
   * Makes POST request to API and returns newly-created user.
   *
   * - username: a new username
   * - password: a new password
   * - name: the user's full name
   */

  static async create(username, password, name) {
    const response = await axios.post(`${BASE_URL}/signup`, {
      user: {
```

```
        username,
        password,
        name,
      }
    });

    // build a new User instance from the API response
    const newUser = new User(response.data.user);

    // attach the token to the newUser instance for convenience
    newUser.loginToken = response.data.token;

    return newUser;
  }

  /* Login in user and return user instance.

   * - username: an existing user's username
   * - password: an existing user's password
   */

  static async login(username, password) {
    const response = await axios.post(`${BASE_URL}/login`, {
      user: {
        username,
        password,
      }
    });

    // build a new User instance from the API response
    const existingUser = new User(response.data.user);

    // instantiate Story instances for the user's favorites and ownStories
    existingUser.favorites = response.data.user.favorites.map(s => new Story(s));
    existingUser.ownStories = response.data.user.stories.map(s => new Story(s));

    // attach the token to the newUser instance for convenience
    existingUser.loginToken = response.data.token;

    return existingUser;
  }

  /** Get user instance for the logged-in-user.
   *
   * This function uses the token & username to make an API request to get details
```

```
   *   about the user. Then it creates an instance of user with that info.
   */

  static async getLoggedInUser(token, username) {
    // if we don't have user info, return null
    if (!token || !username) return null;

    // call the API
    const response = await axios.get(`${BASE_URL}/users/${username}`, {
      params: {token}
    });

    // instantiate the user from the API information
    const existingUser = new User(response.data.user);

    // attach the token to the newUser instance for convenience
    existingUser.loginToken = token;

    // instantiate Story instances for the user's favorites and ownStories
    existingUser.favorites = response.data.user.favorites.map(s => new Story(s));
    existingUser.ownStories = response.data.user.stories.map(s => new Story(s));

    return existingUser;
  }

  /**
   * This function fetches user information from the API
   *  at /users/{username} using a token. Then it sets all the
   *  appropriate instance properties from the response with the current user instance.
   */

  async retrieveDetails() {
    const response = await axios.get(`${BASE_URL}/users/${this.username}`, {
      params: {
        token: this.loginToken
      }
    });

    // update all of the user's properties from the API response
    this.name = response.data.user.name;
    this.createdAt = response.data.user.createdAt;
    this.updatedAt = response.data.user.updatedAt;

    // remember to convert the user's favorites and ownStories into instances of Story
    this.favorites = response.data.user.favorites.map(s => new Story(s));
```

```
    this.ownStories = response.data.user.stories.map(s => new Story(s));

    return this;
  }

  /**
   * Add a story to the list of user favorites and update the API
   * - storyId: an ID of a story to add to favorites
   */

  addFavorite(storyId) {
    return this._toggleFavorite(storyId, "POST");
  }

  /**
   * Remove a story to the list of user favorites and update the API
   * - storyId: an ID of a story to remove from favorites
   */

  removeFavorite(storyId) {
    return this._toggleFavorite(storyId, "DELETE");
  }

  /**
   * A helper method to either POST or DELETE to the API
   * - storyId: an ID of a story to remove from favorites
   * - httpVerb: POST or DELETE based on adding or removing
   */
  async _toggleFavorite(storyId, httpVerb) {
    await axios({
      url: `${BASE_URL}/users/${this.username}/favorites/${storyId}`,
      method: httpVerb,
      data: {
        token: this.loginToken
      }
    });

    await this.retrieveDetails();
    return this;
  }

  /**
   * Send a PATCH request to the API in order to update the user
   * - userData: the user properties you want to update
   */
```

```
  async update(userData) {
    const response = await axios({
      url: `${BASE_URL}/users/${this.username}`,
      method: "PATCH",
      data: {
        user: userData,
        token: this.loginToken
      }
    });

    // "name" is really the only property you can update
    this.name = response.data.user.name;

    // Note: you can also update "password" but we're not storing it
    return this;
  }

  /**
   * Send a DELETE request to the API in order to remove the user
   */

  async remove() {
    // this function is really just a wrapper around axios
    await axios({
      url: `${BASE_URL}/users/${this.username}`,
      method: "DELETE",
      data: {
        token: this.loginToken
      }
    });
  }
}

/**
 * Class to represent a single story.
 */

class Story {

  /**
   * The constructor is designed to take an object for better readability / flexibility
   * - storyObj: an object that has story properties in it
   */
```

```
  constructor(storyObj) {
    this.author = storyObj.author;
    this.title = storyObj.title;
    this.url = storyObj.url;
    this.username = storyObj.username;
    this.storyId = storyObj.storyId;
    this.createdAt = storyObj.createdAt;
    this.updatedAt = storyObj.updatedAt;
  }

  /**
   * Make a PATCH request against /stories/{storyID} to update a single story
   * - user: an instance of User
   * - storyData: an object containing the properties you want to update
   */

  async update(user, storyData) {
    const response = await axios({
      url: `${BASE_URL}/stories/${this.storyId}`,
      method: "PATCH",
      data: {
        token: user.loginToken,
        story: storyData
      }
    });

    const { author, title, url, updatedAt } = response.data.story;

    // these are the only fields that you can change with a PATCH update
    //  so we don't need to worry about updating the others
    this.author = author;
    this.title = title;
    this.url = url;
    this.updatedAt = updatedAt;

    return this;
  }
}
```

Frontend

The jQuery Part

```
$(async function() {
  // cache some selectors we'll be using quite a bit
  const $body = $("body");
  const $allStoriesList = $("#all-articles-list");
  const $submitForm = $("#submit-form");
  const $favoritedStories = $("#favorited-articles");
  const $filteredArticles = $("#filtered-articles");
  const $loginForm = $("#login-form");
  const $createAccountForm = $("#create-account-form");
  const $ownStories = $("#my-articles");
  const $navLogin = $("#nav-login");
  const $navWelcome = $("#nav-welcome");
  const $navUserProfile = $("#nav-user-profile");
  const $navLogOut = $("#nav-logout");
  const $navSubmit = $("#nav-submit");
  const $userProfile = $("#user-profile");

  // global storyList variable
  let storyList = null;

  // global user variable
  let currentUser = null;

  await checkIfLoggedIn();

  /**
   * Event listener for logging in.
   *  If successful, will set up the user instance
   */

  $loginForm.on("submit", async function(evt) {
    evt.preventDefault(); // no page-refresh on submit

    // grab the username and password
    const username = $("#login-username").val();
    const password = $("#login-password").val();

    // call the login static method to build a user instance
    const userInstance = await User.login(username, password);

    // set the global user to the user instance
    currentUser = userInstance;
    syncCurrentUserToLocalStorage();
    loginAndSubmitForm();
  });
```

```
/**
 * Event listener for signing up.
 *  If successful, will setup a new user instance
 */

$createAccountForm.on("submit", async function(evt) {
  evt.preventDefault(); // no page refresh

  // grab the required fields
  const name = $("#create-account-name").val();
  const username = $("#create-account-username").val();
  const password = $("#create-account-password").val();

  // call create method, which calls  API and then builds a new user instance
  const newUser = await User.create(username, password, name);

  currentUser = newUser;
  syncCurrentUserToLocalStorage();
  loginAndSubmitForm();
});

/**
 * Log Out Functionality
 */

$navLogOut.on("click", function() {
  // empty out local storage
  localStorage.clear();
  // refresh the page, clearing memory
  location.reload();
});

/**
 * Submit article event handler.
 *
 * */

$submitForm.on("submit", async function(evt) {
  evt.preventDefault(); // no page refresh

  // grab all the info from the form
  const title = $("#title").val();
  const url = $("#url").val();
  const hostName = getHostName(url);
```

```
    const author = $("#author").val();
    const username = currentUser.username

    const storyObject = await storyList.addStory(currentUser, {
      title,
      author,
      url,
      username
    });

    // generate markup for the new story
    const $li = $(`
      <li id="${storyObject.storyId}" class="id-${storyObject.storyId}">
        <span class="star">
          <i class="far fa-star"></i>
        </span>
        <a class="article-link" href="${url}" target="a_blank">
          <strong>${title}</strong>
        </a>
        <small class="article-hostname ${hostName}">(${hostName})</small>
        <small class="article-author">by ${author}</small>
        <small class="article-username">posted by ${username}</small>
      </li>
    `);
    $allStoriesList.prepend($li);

    // hide the form and reset it
    $submitForm.slideUp("slow");
    $submitForm.trigger("reset");
  });

  /**
   * Starring favorites event handler
   *
   */

  $(".articles-container").on("click", ".star", async function(evt) {
    if (currentUser) {
      const $tgt = $(evt.target);
      const $closestLi = $tgt.closest("li");
      const storyId = $closestLi.attr("id");

      // if the item is already favorited
      if ($tgt.hasClass("fas")) {
        // remove the favorite from the user's list
```

```
        await currentUser.removeFavorite(storyId);
        // then change the class to be an empty star
        $tgt.closest("i").toggleClass("fas far");
      } else {
        // the item is un-favorited
        await currentUser.addFavorite(storyId);
        $tgt.closest("i").toggleClass("fas far");
      }
    }
  });

  /**
   * Event Handler for Clicking Login
   */

  $navLogin.on("click", function() {
    // Show the Login and Create Account Forms
    $loginForm.slideToggle();
    $createAccountForm.slideToggle();
    $allStoriesList.toggle();
  });

  /**
   * Event Handler for On Your Profile
   */

  $navUserProfile.on("click", function() {
    // hide everything
    hideElements();
    // except the user profile
    $userProfile.show();
  });

  /**
   * Event Handler for Navigation Submit
   */

  $navSubmit.on("click", function() {
    if (currentUser) {
      hideElements();
      $allStoriesList.show();
      $submitForm.slideToggle();
    }
  });
```

```
/**
 * Event handler for Navigation to Favorites
 */

$body.on("click", "#nav-favorites", function() {
  hideElements();
  if (currentUser) {
    generateFaves();
    $favoritedStories.show();
  }
});

/**
 * Event handler for Navigation to Homepage
 */

$body.on("click", "#nav-all", async function() {
  hideElements();
  await generateStories();
  $allStoriesList.show();
});

/**
 * Event handler for Navigation to My Stories
 */

$body.on("click", "#nav-my-stories", function() {
  hideElements();
  if (currentUser) {
    $userProfile.hide();
    generateMyStories();
    $ownStories.show();
  }
});

/**
 * Event Handler for Deleting a Single Story
 */

$ownStories.on("click", ".trash-can", async function(evt) {
  // get the Story's ID
  const $closestLi = $(evt.target).closest("li");
  const storyId = $closestLi.attr("id");

  // remove the story from the API
```

```
    await storyList.removeStory(currentUser, storyId);

    // re-generate the story list
    await generateStories();

    // hide everyhing
    hideElements();

    // ...except the story list
    $allStoriesList.show();
  });

  /**
   * On page load, checks local storage to see if the user is already logged in.
   * Renders page information accordingly.
   */

  async function checkIfLoggedIn() {
    // let's see if we're logged in
    const token = localStorage.getItem("token");
    const username = localStorage.getItem("username");

    // if there is a token in localStorage, call User.getLoggedInUser
    //  to get an instance of User with the right details
    //  this is designed to run once, on page load
    currentUser = await User.getLoggedInUser(token, username);
    await generateStories();

    if (currentUser) {
      generateProfile();
      showNavForLoggedInUser();
    }
  }

  /**
   * A rendering function to run to reset the forms and hide the login info
   */

  function loginAndSubmitForm() {
    // hide the forms for logging in and signing up
    $loginForm.hide();
    $createAccountForm.hide();

    // reset those forms
    $loginForm.trigger("reset");
```

```
    $createAccountForm.trigger("reset");

    // show the stories
    $allStoriesList.show();

    // update the navigation bar
    showNavForLoggedInUser();

    // get a user profile
    generateProfile();
  }

  /**
   * Build a user profile based on the global "user" instance
   */

  function generateProfile() {
    // show your name
    $("#profile-name").text(`Name: ${currentUser.name}`);
    // show your username
    $("#profile-username").text(`Username: ${currentUser.username}`);
    // format and display the account creation date
    $("#profile-account-date").text(
      `Account Created: ${currentUser.createdAt.slice(0, 10)}`
    );
    // set the navigation to list the username
    $navUserProfile.text(`${currentUser.username}`);
  }

  /**
   * A rendering function to call the StoryList.getStories static method,
   *  which will generate a storyListInstance. Then render it.
   */

  async function generateStories() {
    // get an instance of StoryList
    const storyListInstance = await StoryList.getStories();
    // update our global variable
    storyList = storyListInstance;
    // empty out that part of the page
    $allStoriesList.empty();

    // loop through all of our stories and generate HTML for them
    for (let story of storyList.stories) {
      const result = generateStoryHTML(story);
```

```
      $allStoriesList.append(result);
    }
  }

  /**
   * A render method to render HTML for an individual Story instance
   * - story: an instance of Story
   * - isOwnStory: was the story posted by the current user
   */

  function generateStoryHTML(story, isOwnStory) {
    let hostName = getHostName(story.url);
    let starType = isFavorite(story) ? "fas" : "far";

    // render a trash can for deleting your own story
    const trashCanIcon = isOwnStory
      ? `<span class="trash-can">
          <i class="fas fa-trash-alt"></i>
        </span>`
      : "";

    // render all the rest of the story markup
    const storyMarkup = $(`
      <li id="${story.storyId}">
        ${trashCanIcon}
        <span class="star">
          <i class="${starType} fa-star"></i>
        </span>
        <a class="article-link" href="${story.url}" target="a_blank">
          <strong>${story.title}</strong>
          </a>
        <small class="article-author">by ${story.author}</small>
        <small class="article-hostname ${hostName}">(${hostName})</small>
        <small class="article-username">posted by ${story.username}</small>
      </li>
    `);

    return storyMarkup;
  }

  /**
   * A rendering function to build the favorites list
   */

  function generateFaves() {
```

```
    // empty out the list by default
    $favoritedStories.empty();

    // if the user has no favorites
    if (currentUser.favorites.length === 0) {
      $favoritedStories.append("<h5>No favorites added!</h5>");
    } else {
      // for all of the user's favorites
      for (let story of currentUser.favorites) {
        // render each story in the list
        let favoriteHTML = generateStoryHTML(story, false, true);
        $favoritedStories.append(favoriteHTML);
      }
    }
  }

  function generateMyStories() {
    $ownStories.empty();

    // if the user has no stories that they have posted
    if (currentUser.ownStories.length === 0) {
      $ownStories.append("<h5>No stories added by user yet!</h5>");
    } else {
      // for all of the user's posted stories
      for (let story of currentUser.ownStories) {
        // render each story in the list
        let ownStoryHTML = generateStoryHTML(story, true);
        $ownStories.append(ownStoryHTML);
      }
    }

    $ownStories.show();
  }

  /* hide all elements in elementsArr */

  function hideElements() {
    const elementsArr = [
      $submitForm,
      $allStoriesList,
      $filteredArticles,
      $ownStories,
      $userProfile,
      $favoritedStories,
      $loginForm,
```

```
    $createAccountForm,
    $userProfile
  ];
  elementsArr.forEach($elem => $elem.hide());
}

function showNavForLoggedInUser() {
  $navLogin.hide();
  $userProfile.hide();
  $(".main-nav-links, #user-profile").toggleClass("hidden");
  $navWelcome.show();
  $navLogOut.show();
}

/* see if a specific story is in the user's list of favorites */

function isFavorite(story) {
  let favStoryIds = new Set();
  if (currentUser) {
    favStoryIds = new Set(currentUser.favorites.map(obj => obj.storyId));
  }
  return favStoryIds.has(story.storyId);
}

/* simple function to pull the hostname from a URL */

function getHostName(url) {
  let hostName;
  if (url.indexOf("://") > -1) {
    hostName = url.split("/")[2];
  } else {
    hostName = url.split("/")[0];
  }
  if (hostName.slice(0, 4) === "www.") {
    hostName = hostName.slice(4);
  }
  return hostName;
}

/* sync current user information to localStorage */

function syncCurrentUserToLocalStorage() {
  if (currentUser) {
    localStorage.setItem("token", currentUser.loginToken);
    localStorage.setItem("username", currentUser.username);
```

```
    }
  }
});
```