# COMP 2537 Web Development 2

# Assignment 2

Introduction:

This assignment builds on the previous assignment, Assignment 1.

You should already have a website that can create users and log in as them using encrypted sessions. Those users are stored in a MongoDB database, hosted on MongoDB Atlas and protected from NoSQL Injection attacks. MongoDB database connection usernames and passwords and all other secrets are stored in a `.env` file and kept out of the github repo. Your site is hosted via Qoddi or some other similar hosting service.

There's a few things we are going to add as part of this assignment: 1) EJS Templates, 2) authorization, and 3) use a CSS Toolkit like Bootstrap.

**EJS Templates.** In Assignment 1, every request we did for a webpage our code looked something like this:

```
app.get('/logout', (req,res) => {
    var html = `You are logged out.`;
    res.send(html);
});
```

In this example, the code for handling the request [`app.get('/logout', …)`] is mixed with the our HTML. This is hard to maintain on anything other than the smallest of projects.

This was ok when we didn't have too many pages and our server-side logic was fairly simple. However, this design doesn't scale well. First off, we'd like to have a separate file for each webpage; this way, when we work as a team, different team members can work on different files at the same time and have fewer Git merge conflicts (you'll learn in COMP 2800 that Git merge conflicts are bad and are to be avoided if possible). Secondly, we'd like to separate our server code from our HTML. HTML files can get quite large once you incorporate JavaScript files, CSS files, images, headers, navigation bars, footers, etc. Having all the HTML in a single file will make this file very large. Splitting the HTML for each page into its own file will make our code more readable and maintainable. Thirdly, templating can give us the ability to identify repeated sets of HTML blocks (like headers, navigation bars, footers, lists, etc.) and reuse them without copy/pasting. Copy/pasting leads to inconsistencies and difficulty editing our code when inevitably we need to make a change, because we need to remember and update all related code blocks.

**Authorization.** Authorization is the ability to tell different types of users apart and providing some types of users more abilities than others. Say for example you want to have some users be administrators who can view all users in the system, delete users, reset their passwords and provide additional functionality. You probably only want to provide these abilities to people you trust. Regular users should not have this much power! Regular users might have the ability to edit their own information but not of other users. Guests may have even less permissions and only have the ability to view the data and not edit anything.

Keep in mind that authorization is different authentication. Authentication is the process of verifying a user is who they say they are. This verification is usually done by confirming they know a password or PIN, but can also be done by any number of other methods such as fingerprinting, facial recognition or retinal scans, sending a one-time code to a phone or email address or using a key fob or card.

**CSS Toolkit – Bootstrap.** Bootstrap is a CSS Toolkit that can do a lot of the heavy lifting of your initial CSS, by providing you boilerplate or a base of CSS. This means you don't have to re-invent the same CSS for all your sites and you can spend more time working on other things like your server-side code, your database, and the business logic that will make your webapp awesome!

The official documentation for bootstrap can be found here: https://getbootstrap.com/

To include Bootstrap's CSS you can add this to your `<HEAD>` section of your HTML:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
```

Note: the link above is for bootstrap **v5.3 alpha**. You may want to get the latest version available from Bootstrap, if one is available.

The examples page on Bootstraps pages shows examples of headers, footers, navbars, heroes, buttons, grids and so much more. https://getbootstrap.com/docs/5.3/examples/

User Stories:

As you know, user stories are a key part of the agile development methodology as they help define valuable software features from the perspective of the end-user.

Here are some examples of user stories being implemented in this assignment:

- As a *user*, I want to be able to **add images** for my account in the members area.
- As a *user*, I want the website to have a **consistent look and feel**.
- As a *user*, I want the website to have a **logical flow**.
- As a *user*, I want the website to be **aesthetically pleasing**.
- As a *user*, I want the website to look good on my **mobile** phone **and** on my **desktop** computer.

- As an *administrator* of the site, I want to be able to **see all the users** and which type they are.
- As an *administrator* of the site, I want to be able to **promote users** that I trust admin privileges and **demote users** if I don't.
- As an *administrator* of the site, I want to make sure **only admins** can see the admin page.

From the perspective of the developer, you might appreciate the following developer stories. While technically not user-stories, these developer stories are worth mentioning and working towards, since having cleaner code makes developers happy (and having happy developers is a good thing!).

Here are some examples of developer stories being implemented in this assignment:

- As a developer, I want to have a code base that is **easy to read and to maintain**.
- As a developer, I want to **minimize** the amount times I **copy/paste** the same or similar code.
- As a developer, I want to **avoid editing the same file** as my teammates to avoid git merge conflicts.
- As a developer, I want to **avoid large code files** (with lots of lines) – instead I prefer several smaller code files.

Node Modules you will need:

In Node we will need some external libraries to make our website work:

- **ejs**
  According to EJS's official site (https://ejs.co/): "EJS is a simple templating language that lets you generate HTML markup with plain JavaScript".

Remember, in order to use these modules, you'll need to install them using:

```
npm i <module_name>
```

Example:

```
npm i ejs
```

EJS:

As a templating engine, EJS can let us separate our HTML from our server routing code.

Instead of (in `index.js`):

```
app.get('/logout', (req,res) => {
    var html = `You are logged out.`;
    res.send(html);
});
```

We can use this instead (`index.js`):

```
app.get('/logout', (req,res) => {
    res.render("logout");
});
```

And put our HTML in a file called `/views/logout.ejs`

```
You are logged out.
```

Notice we switched from `res.send` to `res.render`. All we need to do is provide the name of the file (without the .ejs extension) and EJS will open the file and send the HTML contents to the user.

We also need to tell express that we want it to use EJS to render our ejs files as HTML using this middleware. You need to put this line near the top of your index.js file before your routes:

```
app.set('view engine', 'ejs');
```

By default, EJS will look for our HTML code (which it likes to call 'views') in a folder called `/views/`

Passing in variables in EJS Views:

We can also pass parameters to our views by using a second parameter in the `res.render` function like this:

```
app.get('/members, (req,res) => {
    res.render("members", {user: "Francine"});
});
```

And put our HTML in a file called `/views/members.ejs`

```
Welcome, <%= user %>!
```

When this `members.ejs` gets rendered, EJS will replace the `<% user %>` with the argument passed in – in this case "Francine". The resulting HTML would look like this:

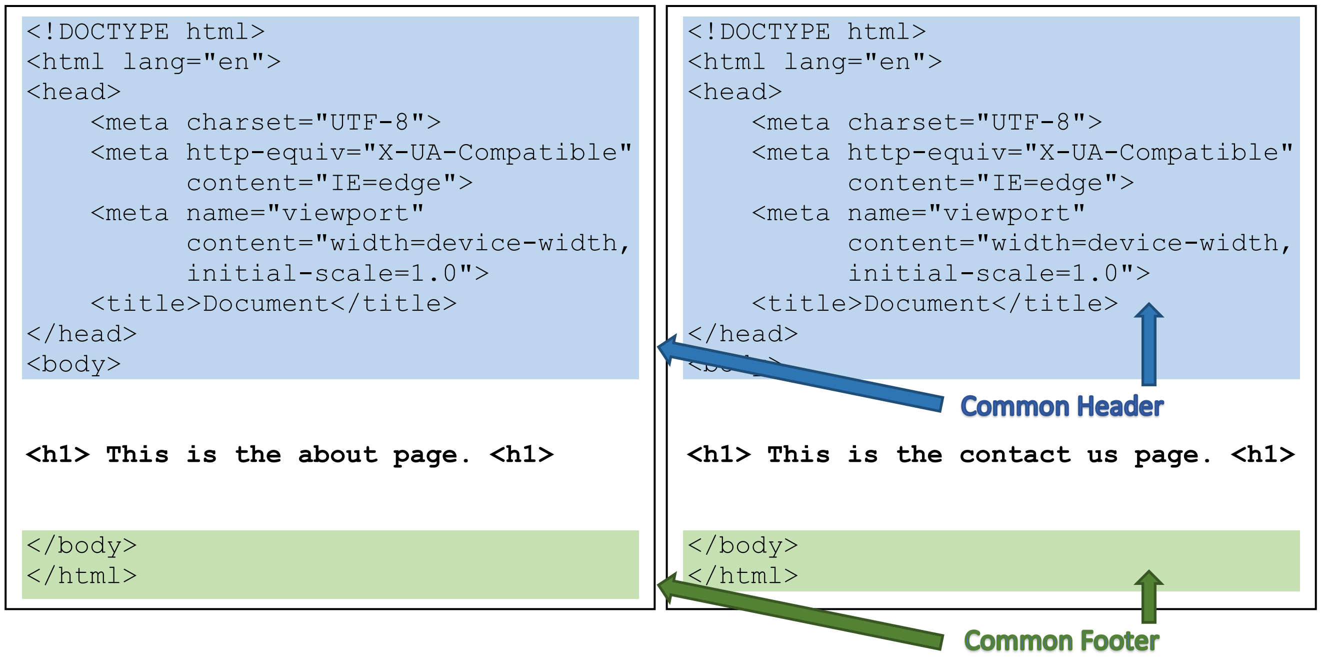
```
Welcome, Francine!
```

Using EJS Templates:

When creating a website with multiple webpages, there is often blocks of HTML that are repeated. Take the following 2 pages – the about page and the contact us pages:

About us page:

Contact us page:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
          content="IE=edge">
    <meta name="viewport"
          content="width=device-width,
          initial-scale=1.0">
    <title>Document</title>
</head>
<body>


<h1> This is the about page. <h1>


</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
          content="IE=edge">
    <meta name="viewport"
          content="width=device-width,
          initial-scale=1.0">
    <title>Document</title>
</head>
<body>


<h1> This is the contact us page. <h1>


</body>
</html>
```

Notice how both pages have the save code at the beginning (header) and the same code at the end (footer). EJS can help separate this code into templates that can be reused so we don't have to copy/paste our HTML.

We put this into `/views/templates/header.ejs`:

```
<!DOCTYPE html>                                              Common Header
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible"
          content="IE=edge">
    <meta name="viewport"
          content="width=device-width,
          initial-scale=1.0">
    <title>Document</title>
</head>
<body>
```

We put this into `/views/templates/footer.ejs`:

```
</body>                                                      Common Footer
</html>
```

And our about page (`/views/about.ejs`) becomes:

```
<%- include("templates/header") %>
    <h1> This is the about page. <h1>
<%- include("templates/footer") %>
```

And our about page (`/views/contact.ejs`) becomes:

```
<%- include("templates/header") %>
     <h1> This is the contact us page. <h1>
<%- include("templates/footer") %>
```

Separating the headers and footers like this has 2 main advantages. First, changes to the common header and footer are easier since they happen in a single place. And second, the about us and contact us .ejs files are now shorter and focus on just the content that makes them different from each other.

For more information see EJS's official documentation here: https://ejs.co/#docs

EJS for code blocks:

One additional thing we can do for reducing code duplication is identifying repeated HTML code blocks and separate those into templates as well. For example, if we have a list of items we want to display using an ordered list in HTML, we might want to use an `<ol>` tag with each item being a list item `<li>` tag (file: `/views/users.ejs`):

```
<ol>
  <% let i=0; %>
  <% for(i=0; i < users.length; i++) { %>
    <%- include('templates/user', {user: users[i]})  %>
  <% } %>
</ol>
```

In our user template file (`/views/templates/user.ejs`):

```
<li>
  <%= user.username %>: <%= user.email %>
</li>
```

A call to `res.render` on this template:

```
res.render("users", users: [{username: "me", email: me@b.ca},
{username:"you", email: "you@b.ca"}]
```

Would produce the following HTML:

```
<ol>
  <li> me: me@b.ca </li>
  <li> you: you@b.ca </li>
</ol>
```

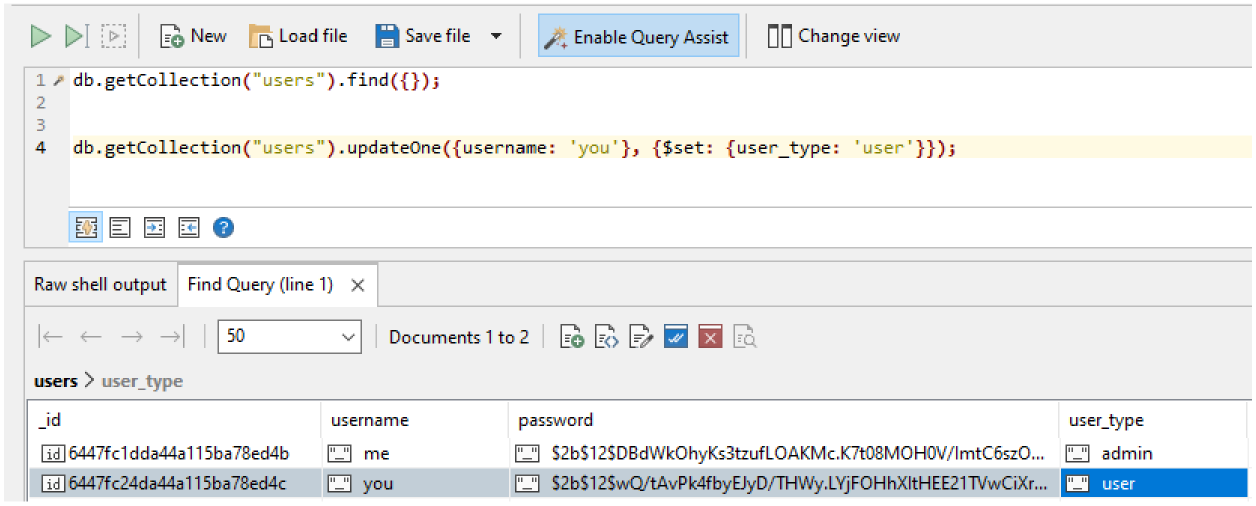
(It loops over our array of 2 users, each user is an object with 2 attributes, username and email).

## MongoDB Updates:

This assignment will have you update an existing document within your MongoDB database.

If we had a user with username = "you" and we wanted to change their user_type to "user" the following code works within Studio 3T:

```
db.getCollection("users").updateOne({username: 'you'}, {$set: {user_type: 'user'}});
```



```
db.getCollection("users").find({});


db.getCollection("users").updateOne({username: 'you'}, {$set: {user_type: 'user'}});
```

| _id | username | password | user_type |
|---|---|---|---|
| 6447fc1dda44a115ba78ed4b | me | $2b$12$DBdWkOhyKs3tzufLOAKMc.K7t08MOH0V/ImtC6szO... | admin |
| 6447fc24da44a115ba78ed4c | you | $2b$12$wQ/tAvPk4fbyEJyD/THWy.LYjFOHhXltHEE21TVwCiXr... | user |

If we wanted to do the same in our Node.js code, we can almost use the same code, the only change we would need it to use the `await` keyword in front like this:

```
await db.getCollection("users").updateOne({username: 'you'}, {$set: {user_type: 'user'}});
```

And remember, that any function that calls a synchronous function using `await` needs to be declared as `async`.

For more information about how to update data inside a MongoDB database in Node.js, here are resources:
https://www.mongodb.com/docs/drivers/node/current/usage-examples/updateOne/

https://mongodb.github.io/node-mongodb-native/3.0/reference/ecmascriptnext/crud/

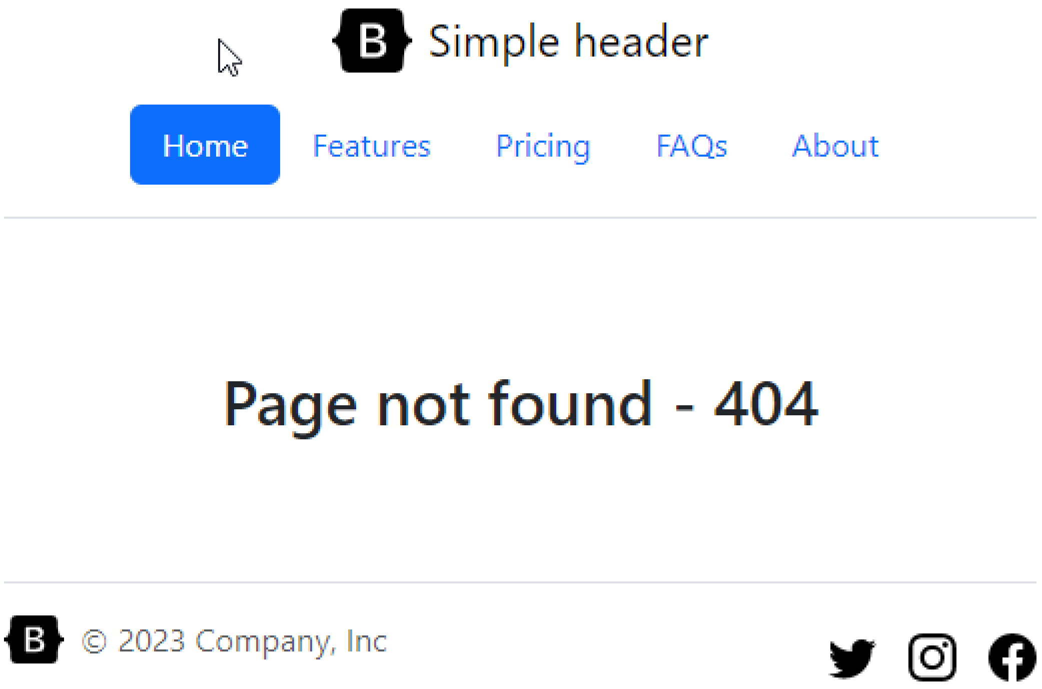
Bootstrap (CSS Framework):

Customize your site with:

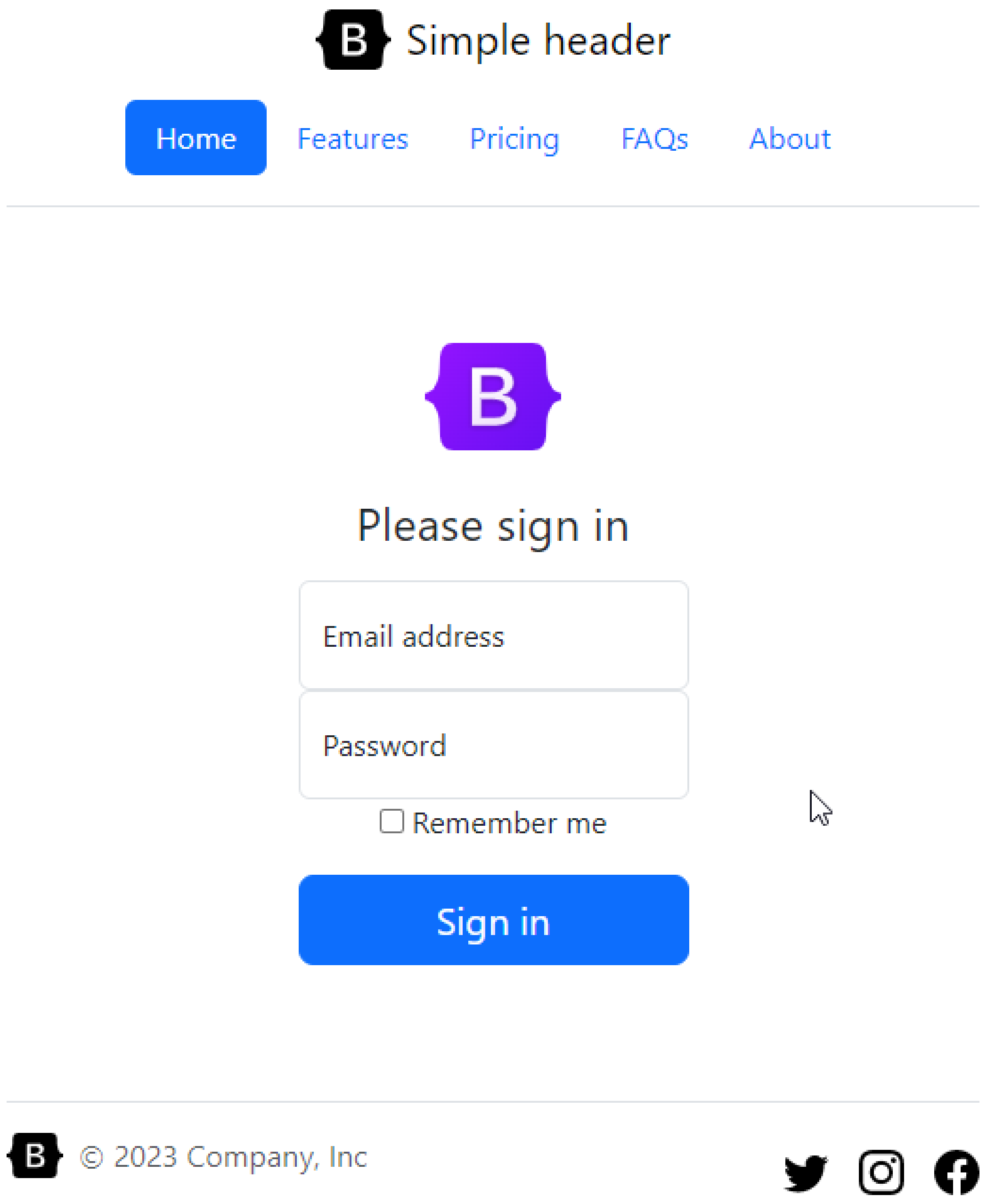
1. a header
2. a footer
3. buttons
4. forms and
5. grids (where applicable)

Bootstrap has a site demoing snippets and components to give you ideas of what you can use on your site. https://getbootstrap.com/docs/5.3/examples/

This example shows the 404 page with a bootstrap header, and footer:



This page shows the login page with the same header and footer, with Bootstrap form styling:



Simple header

Home Features Pricing FAQs About

Please sign in

Email address

Password

Remember me

Sign in

© 2023 Company, Inc

Objectives:

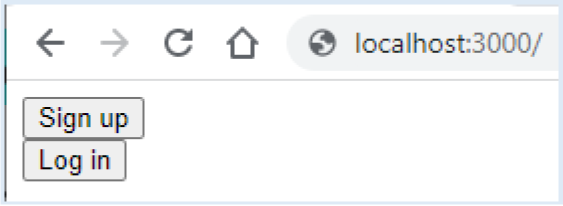
1. Refactor your code from Assignment 1 to use the **EJS Templating**.
   Keep in mind that if you see HTML code that is being used multiple times, avoid repeating this code in multiple files. Instead use EJS to include those **common blocks**.
   Create a **common header** and **common footer** to be reused for all pages.

Your Node.js website should have the following pages and functionality:

Existing functionality (but now refactored code using EJS):
Note also, your site will look different from the screenshots below, since you will be using Bootstrap or another CSS framework to make your site look beautiful.

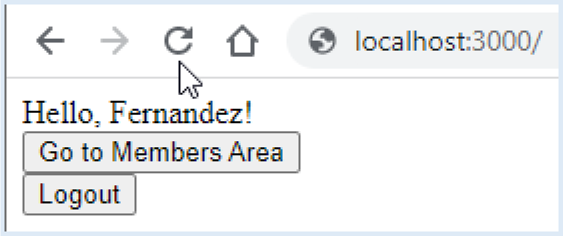
1. Home page – site: `/` method: `GET`

   If the user is **not logged in**, this home page will have links to:

   a. Sign up (`/signup`)
   b. Log in (`/login`)

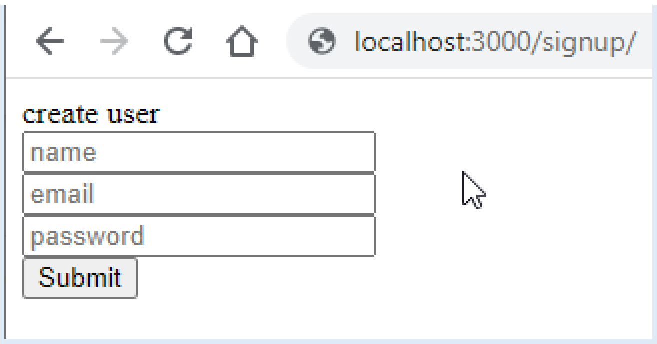
   Else, if the user is **logged in**, this home page will:

   a. Say Hello **and** the name of the user – ex: Hello, Fernadez.
      You should store the user's name in the session during login/signup and display it here.

   b. Provide a link to the members area (`/members`).
   c. Have a link that will log the user out (`/logout`).
      This will end the session and redirect back to (`/`).

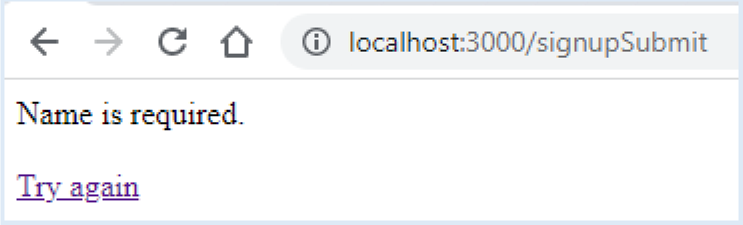
2. Sign up page – site: `/signup` method: `GET`



A form will gather the following information for sign up:

a. Name
b. Email
c. Password

The signup form will `POST` the form fields.
You will validate all 3 inputs to make sure they aren't empty. If the user has forgotten to enter one or more of the 3 inputs, you will send back an appropriate message saying which field was missing. Ex: *Please provide an email address.* You will also provide a link back to the login page so the user may try again:

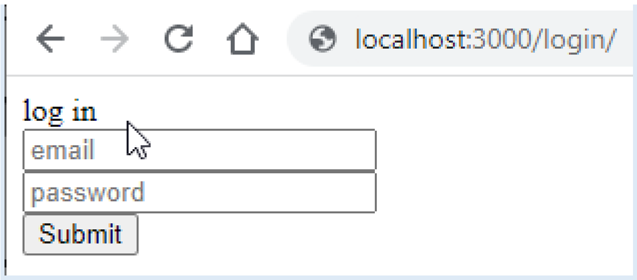


Name is required.

Try again

If the 3 fields are non-empty, add the user to your MongoDB database. Make sure that you are using Joi to validate the input (name, email and password) so that NoSQL Injection attacks are not possible. Add the **name**, **email** and a **<u>BCrypted hashed</u> password** as a user to the database. Then create a session and redirect the user to the `/members` page.

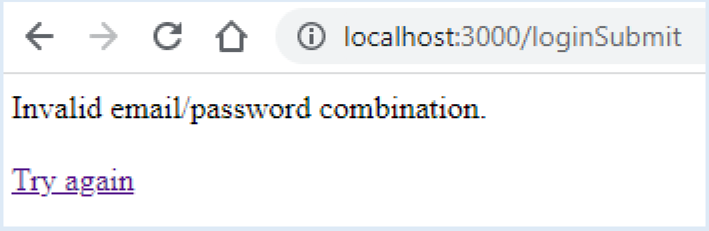
3. Log in page – site: `/login` method: `GET`



A form will gather the following information for sign up:

d. Email
e. Password

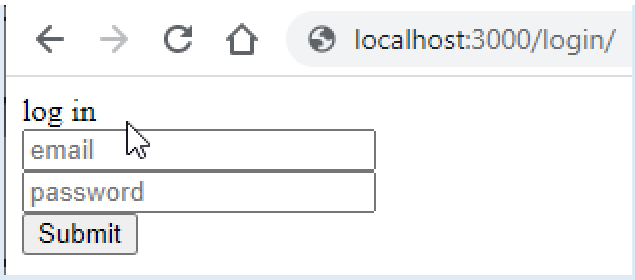
The login form will `POST` the form fields. Check the user against the MongoDB database. Make sure that you are using Joi to validate the input (email and password) so that NoSQL Injection attacks are not possible. If a user with that email is found, check to see if the password matches the BCrypted password. If the email and passwords match store the user's name in a session and log the user in (redirect to `/members`).
If the log in fails, send a message to the user with an appropriate message. Ex: *User and password not found.* Also provide a link back to the signup page so that users can try again:


localhost:3000/loginSubmit

Invalid email/password combination.

Try again

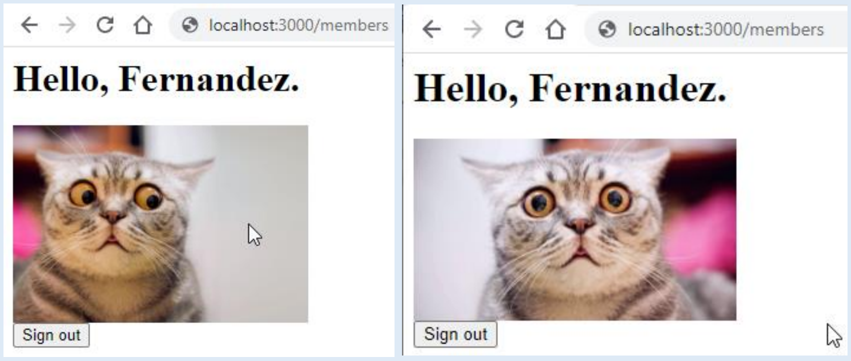
4. Members page – site: `/members` method: `GET`


localhost:3000/login/

log in

If the user has a **valid session** (is logged in):

a. Say Hello **and** the name of the user – ex: Hello, Fernandez.
   (use the name value stored in the session)
b. Have a link that will log the user out (ends the session and redirects back to `/`).
c. Display a random image from a selection of 3 images.
   The images should be stored in a `/public` folder on the server.


localhost:3000/members

Hello, Fernandez.

localhost:3000/members

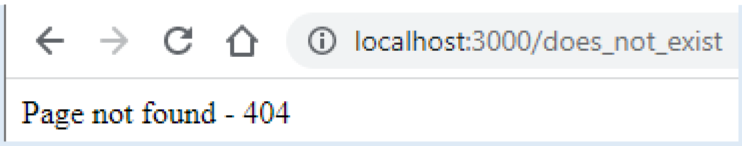
Hello, Fernandez.

If the user has **no session** (or an invalid one – i.e. not logged in):

a. Redirect to the home page (`/`).

5. Log out page – site: `/logout` method: `GET`
   This page doesn't have any content, but should destroy the user's session (`req.session.destroy();`) and redirect back to the home page (`/`).

6. 404 page – site: any non-assigned URLs method: `GET`

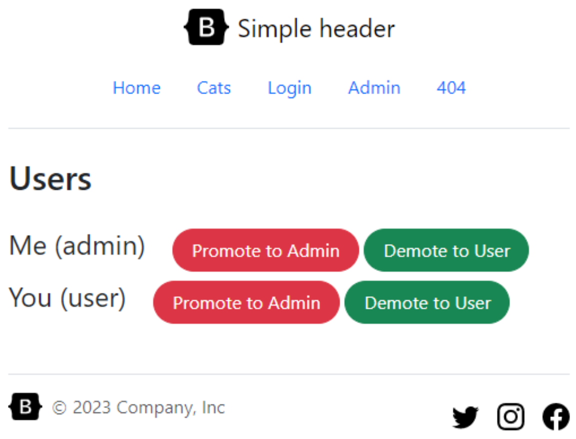


Page not found - 404

   Any page that doesn't match the above sites will send an HTML status code of **`404`** and display page not found – Ex: `/does_not_exist`

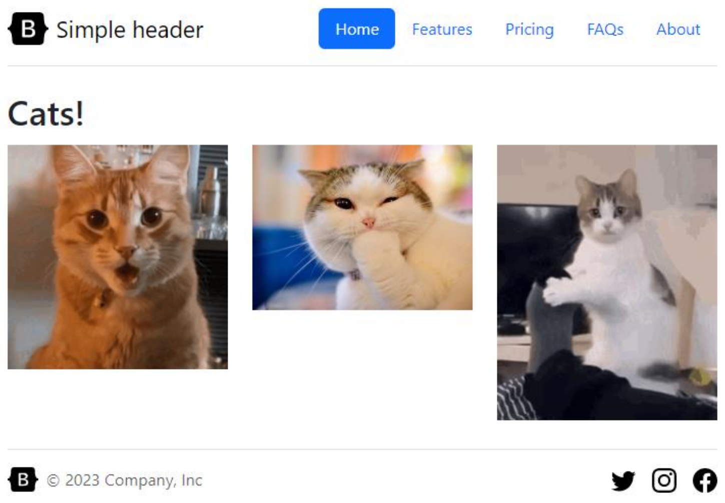
New Functionality:

7. Admin page – site: `/admin` method: `GET`

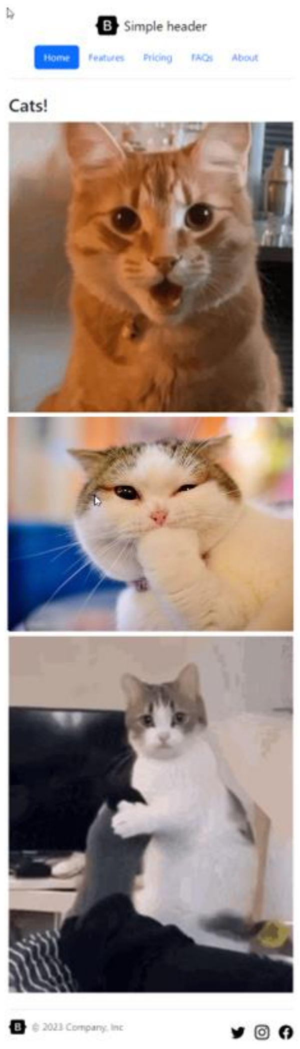


   This page should show a list of **all the users** and **which type they are** (admin or user) within your MongoDB Database (your results will vary depending on which users you have in your database).
   For each of the users, provide a link to **promote them to an admin** or **demote them to a regular user**. (This change needs to be updated in your MongoDB database).

8. Members page – site: `/members` method: `GET`
   Instead of providing the user with a random image (1 of 3), **display all 3 images** in a Bootstrap responsive grid.



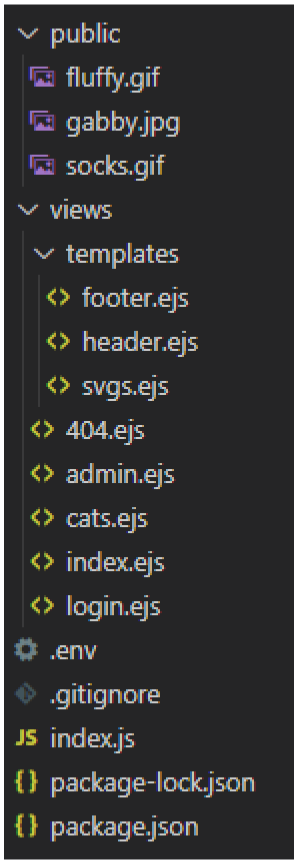
(this is the desktop view)



(this is the mobile view)

EJS Templates:

Your site should be structured like this:



```
public
    fluffy.gif
    gabby.jpg
    socks.gif
views
    templates
        footer.ejs
        header.ejs
        svgs.ejs
    404.ejs
    admin.ejs
    cats.ejs
    index.ejs
    login.ejs
.env
.gitignore
index.js
package-lock.json
package.json
```

You should have a /views folder for your ejs files.

You should have a templates folder ejs files that are included within other ejs files (ex: `header.ejs`, `footer.ejs`, `user.ejs`).

One of the goals of using a CSS Framework like Bootstrap is to minimize how much CSS *we* have to write. Avoid writing your own inline or external style sheets. Instead use Bootstraps built-in modifiers and utility classes.

Your `.env` file:

Make sure that your username and password for connecting to your MongoDB database are stored in your `.env` file. Also, store your encryption keys for your sessions in the `.env` file. Make sure your `.env` is added to the `.gitignore` so that the `.env` file is NOT added to your git repo.
A `.env` file should **NEVER** be in your git repo!

<u>Create a Git Repo:</u>

Create a git repo for your website. Make sure to add all files, commit and **push** them your origin on github.com. A single branch called `main` (or `master`) is fine.

As mentioned earlier, ensure you have excluded the `.env` file from your git repo.

Create a `.gitignore` file with the following contents:

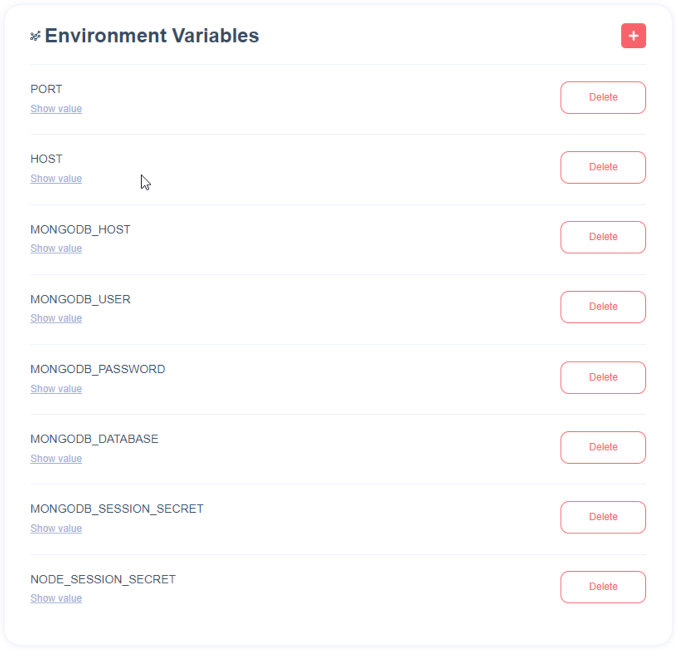
```
/node_modules
package-lock.json
.env
```

<u>Add a Procfile</u>:

Remember, hosting services like Qoddi (and Heroku) can run multiple different types of applications including PHP, Python, Ruby on Rails, Node.js and many more. We need to tell Qoddi which type of application is in our git repo and how it should be run.

<u>Hosting on Qoddi (or other hosting service):</u>

Remember, we need Qoddi to have access to the passwords and secrets in our `.env` file, so we need to transfer all the variables in our `.env` to Qoddi. Qoddi lets us create environment variables in its online interface.



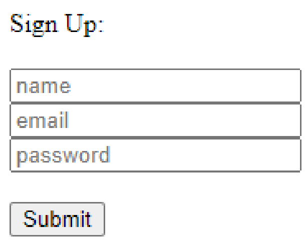
PORT
HOST
MONGODB_HOST
MONGODB_USER
MONGODB_PASSWORD
MONGODB_DATABASE
MONGODB_SESSION_SECRET
NODE_SESSION_SECRET

Create a new environment variable for each variable in your `.env` file.

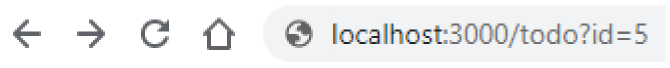
Protect against NoSQL Injection Attacks:

Review your code and make sure any time we query a database and we include input from a user, we protect the query against NoSQL Injection attacks.User input can come in multiple ways such as form fields in POST requests and URL parameters.

Form Fields



URL parameters



User input can come from a form field like `req.body.username` or from a URL parameter like `req.params.id`

Use Joi to ensure all your user input is a string to prevent NoSQL Injection attacks.

Marking Guide:

| Criteria | Marks |
|---|---|
| The `/admin` page redirects to the `/login` page if not logged in. | 5 marks |
| The `/admin` page shows an error message if logged in, but not an admin | 5 marks |
| The `/admin` page shows a list of all users | 5 marks |
| The `/admin` page allows for promoting and demoting users to/from admin type (This is verified in the database) | 5 marks |
| All pages use a CSS Framework like Bootstrap You must incorporate a header, footer, responsive grid, forms, buttons. | 5 marks |
| The site uses EJS as a templating engine | 5 marks |
| Common headers and footers are shared across all pages | 5 marks |
| Code used within loop is templated using EJS (ex: list of users in admin page) | 5 marks |
| The members page has a responsive grid of 3 images. | 5 marks |
| Your site is hosted on Qoddi or other hosting site. | 5 marks |
| Note: You will receive a **mark of 0** if your code doesn't run (i.e. too many errors to run properly, cannot connect to the database, etc.)! I will be running the code on your Qoddi app, please make sure it is available until after you receive your marks! | |
| **Total:** | **50 marks** |

Submission Requirements:

| **Submission:** | **File name:** |
|---|---|
| A txt file containing a Youtube video link to your video demo. | Video_demo_URL.txt |
| A txt file containing the Qoddi app URL. ex: `http://xrb198fpq1a.us11.qoddiapp.com/` | Qoddi_app_URL.txt |
| A zip file containing **all the source code** required to run your Node.js application. Do **NOT** include the `/node_modules/` or the `/.git/` folder | Members_site_code.zip |
| Self-graded checklist | My_checklist.txt |

Self-graded Checklist:

Find the self-graded assignment 2 checklist template and for each item, either **leave it blank** (if you did not complete the task fully) or put an **x** next to it (indicating you completed the task).
Provide a totalled grade out of 50.

Example:

```
[ ]  The signout buttons end the session.    (incomplete)
[x]  The .env file is NOT in your git repo.  (complete)
```

Video Demo Requirements:

| **Your video needs to include:** | **Example:** |
|---|---|
| Your name | Patrick Guichon |
| Your set | 1A |
| Which assignment you are demoing | Assignment 2 |

| **Notes about your video:** |
|---|
| If we don't see all the functionality and features of your website demonstrated in the video, you will lose marks for not having completing this part of the website. |
| You do NOT need to show your face while recording the video, however, you should be narrating the video. Tell us what you are doing, as you are demoing your site. |
| Make sure we can hear your voice clearly, and at an adequate volume. |
| Max video length: 5m:00s. Your video should **NOT be longer than 5 minutes**. |

| **Demo the following functionality in this order:** |
|---|
| 1. Show the home page (`/`) running on Qoddi (or other hosting service) |
| 2. Show the users database in Studio 3T. You should have at least one **normal** user (`user_type = "user"`) and at least one **admin** user (`user_type = "admin"`) |
| 3. Show that there is no active session (by showing all cookies are deleted). Show that going to the `/admin` page (without a session) redirects back to the `/login` page. |
| 4. Click on the Log in link and login as a **normal** user. Return to the `/admin` page and show that this user doesn't have access with a 403 error. |
| 5. Clear your session (by deleteing the cookie and/or the database cookie). |
| 6. Click on the Log in link again and login as an **admin** user. Return to the `/admin` page and show that this user is allowed to see this page. |
| 7. Pick a **normal** user and promote them to **admin**. Confirm within the database that this user now has `user_type = "admin"`. |
| 8. Pick an **admin** user and demote them to **normal**. Confirm within the database that this user now has `user_type = "user"`. |
| 9. Go to a page that doesn't exist example `/doesnotexist` to show the 404 page. Make sure that you are using Bootstrap (or another CSS Framework) |
| 10. Show us your code:<br>- Show us your `header.ejs` template<br>- Show us your `footer.ejs` template<br>- Show us your `user.ejs` template<br>(this `user.ejs` template should be used to loop over each user)<br>- Show us at least 2 ejs pages that include those the `header.ejs` and `footer.ejs` templates<br>(ex: `admin.ejs`, `index.ejs` and `404.ejs`)<br>- Show us your authentication and authorization code<br>(ex: your middleware functions that ensure valid sessions and admins only) |