Contents

[08-15-2020 Class Homework Help 2](#_Toc47430303)

[Document Goal 2](#_Toc47430304)

[Assignment 2](#_Toc47430305)

[Navigation 2](#_Toc47430306)

[Adding Menus 3](#_Toc47430307)

[Login 4](#_Toc47430308)

[My Mazes Implementation 5](#_Toc47430309)

[Customizing Appearance 5](#_Toc47430310)

[Changing Navigation Look & Feel 5](#_Toc47430311)

[Predefined Colors 7](#_Toc47430312)

[Using a Palette Seletor 7](#_Toc47430313)

[Icons and Images 8](#_Toc47430314)

# 08-15-2020 Class Homework Help

## Document Goal

In order to help you with the homework I am giving you some comments and notes over the next two weeks. This I hope keeps you engaged and gives you a way to be successful without too much frustration if you do not already know the Node language.

## Assignment

For this week’s homework you are to add navigation and login processing, finally you need to implement the pages to support the navigation “My Mazes” and customize the look and feel to your own liking. Once this homework is complete future updates will be file specific and you will need to consider merging changes I make into your code if you need to use my code to complete homework assignments. This document and other help documents are designed to give you knowledge about all files and processes I used to complete the homework myself. Also you are getting insight into the patterns used to complete an entire website design and infrastructure. While this code is not complete, the foundation and how it interacts to provide colors, navigation, db support, api calls etc is in place. This will make it easier for you to learn coding as well as customize it and make this application your own. As the class advances my hope is that you can customize the maze above and beyond our goals as well as complete a template of how websites work behind the scenes using Node and JS.

## Navigation

First lets look at navigation. The navigation is partially laid out for you. I have set a framework for this that includes navigation for my current application plans.

My Mazes, Favorites, Add, Search, Register, Login, Logout (in this order, but you could change that if you like)

I am going to give you some information about where things can be changed.

Your navigation will be in the layout.pug file in the /views folder.

The code for navigation is like the following. This takes advantage of the css and icon files to display a header navigation. You can also move the navigation to the side or bottom by changing other CSS code.

  body

    block header

      header.top

        nav.nav

          .nav\_\_section.nav\_\_section--pages

            li.nav\_\_item

              a.nav\_\_link.nav\_\_link--logo(href="/")

                != h.icon('logo')

            each item in h.menu

              li.nav\_\_item

                a.nav\_\_link(href=item.slug, class=(currentPath.startsWith(item.slug) ? 'nav\_\_link--active' : ''))

                  != h.icon(item.icon)

                  span #{item.title}

          .nav\_\_section.nav\_\_section--search

            .search

              input.search\_\_input(type="text" placeholder="Maze name, Date..." name="search")

              .search\_\_results

          .nav\_\_section.nav\_\_section--user

            if user

              li.nav\_\_item: a.nav\_\_link(href="/hearts", class=(currentPath.startsWith('/hearts') ? 'nav\_\_link--active' : ''))

                != h.icon('heart')

                span.heart-count #{user.hearts && user.hearts.length}

              li.nav\_\_item: a.nav\_\_link(href="/logout", class=(currentPath.startsWith('/logout') ? 'nav\_\_link--active' : ''))

                != h.icon('logout')

                span Logout

              li.nav\_\_item: a.nav\_\_link(href="/account", class=(currentPath.startsWith('/account') ? 'nav\_\_link--active' : ''))

                img.avatar(src=user.gravatar + 'd=retro')

            else

              li.nav\_\_item: a.nav\_\_link(href="/register", class=(currentPath.startsWith('/register') ? 'nav\_\_link--active' : '')) Register

              li.nav\_\_item: a.nav\_\_link(href="/login", class=(currentPath.startsWith('/login') ? 'nav\_\_link--active' : '')) Log In

## Adding Menus

A few things for your menu handling. The top bar color is in the style.css file. You can change the colors with this line.

    background: linear-gradient(90deg, #4878de 0%, #2630bf 20%, #e82cbf 60%, #ff0eaf 85%, #de4848 95%);

When you look at this code in the editor it will show the colors like the below.



You can change your menu options in the helpers.js file with this code.

exports.menu = [

  { slug: '/maze', title: 'My Mazes', icon: 'maze', },

  { slug: '/top', title: 'Favorites', icon: 'favorite', },

  { slug: '/add', title: 'Add', icon: 'add', },

];

You can change the icons in the /images/icons folder. They are svg files. You can convert the svg to a different file type by changing this command in the helpers.js file.

exports.icon = (name) => fs.readFileSync(`./public/images/icons/${name}.svg`);

You can change your mouse over colors for the navigation in the layout.css css file here:

a:active,

a:hover {

  outline-width: 0; }

Also in the \_nav.css file in the /sass/partials folder

    &:hover, &--active {

      border-bottom-color: rgba(0,0,0,0.2);

      border-right-color: rgba(0,0,0,0.05);

      svg {

        transform: scale(1.2)

      }

## Login

Your login already technically works. You only need a DB connection to MONGO and a model. There are a few other minor things but they are laid out for you in the starter code already.

The model is user.js in the /models folder.

We will look at the model more in classes in the future. The following is a core user descriptive schema.

const userSchema = new Schema({

  email: {

    type: String,

    unique: true,

    lowercase: true,

    trim: true,

    validate: [validator.isEmail, 'Invalid Email Address'],

    required: 'Please Supply an email address'

  },

  name: {

    type: String,

    required: 'Please supply a name',

    trim: true

  },

  resetPasswordToken: String,

  resetPasswordExpires: Date,

  hearts: [

    { type: mongoose.Schema.ObjectId, ref: 'Maze' }

  ]

});

Here you notice we enforce required and formats. We store our favorite mazes as a Maze object on the user so we know what the user’s choices are for favorites.

In order to login you will need to be able to access registration and register as a user.

## My Mazes Implementation

You will see the exported function myMazes in the mazeController.js class. This is the logic that will get your pages from the DB. Right now there is not much to make this all happen. The logic calls for a maze db table and a maze model. The logic will need a mazes pug file to hand display. I would recommend a maze mixin so it can be used over and over to create a list of your mazes. You will need to implement the \_pagination mixin if you have more than 4 mazes stored for a single profile. This can be made larger in the mazeController.js class here:

  const page = req.params.page || 1;

  const limit = 4;

  const skip = (page \* limit) - limit;

## Customizing Appearance

You can customize the appearance of your pages with the following files.

Styles are added to our app through the maze-app.js class in the /public/scripts folder with this code:

import '../sass/style.scss';

To change Buttons change this code in styles.css file in the /public/dist folder

.button {

  border: 0;

  background: #0307fd;

  color: #fdfcfc;

  font-family: 'Panama';

  font-weight: 600; }

## Changing Navigation Look & Feel

You can change the top navigation of the page look and feel with this code in the styles.css file inside /public/dist folder.

.nav {

  display: -ms-flexbox;

  display: flex;

  list-style: none;

  margin: 0;

  padding: 0;

  -ms-flex-pack: justify;

      justify-content: space-between;

  background: #fdfbfb; }

  .nav\_\_section {

    display: -ms-flexbox;

    display: flex; }

    .nav\_\_section--search {

      -ms-flex: 1 1 auto;

          flex: 1 1 auto; }

  .nav\_\_item {

    display: -ms-flexbox;

    display: flex; }

  .nav\_\_link {

    background: #f8f7f7;

    color: rgb(7, 7, 7);

    border-right: 1px solid rgb(252, 6, 6);

    text-transform: uppercase;

    padding: 1.2rem 2rem 1rem 2rem;

    display: block;

    display: -ms-flexbox;

    display: flex;

    -ms-flex-direction: column;

        flex-direction: column;

    -ms-flex-align: center;

        align-items: center;

    -ms-flex-pack: center;

        justify-content: center;

    transition: transform 0.2s;

    border-bottom: 5px solid transparent; }

    .nav\_\_link svg {

      width: 40px;

      transition: all 0.2s;

      fill: white;

      margin-bottom: 1rem; }

    .nav\_\_link:hover, .nav\_\_link--active {

      border-bottom-color: rgb(247, 243, 8);

      border-right-color: rgba(247, 243, 8);

      background: linear-gradient(90deg, #2f02f7 0%, #a026bf 20%, #e82c75 60%, #FFC40E 85%, #48ded4 95%) fixed; }

      .nav\_\_link:hover svg, .nav\_\_link--active svg {

        -ms-transform: scale(1.2);

            transform: scale(1.2); }

    .nav\_\_link--logo:hover svg {

      -ms-transform: none;

          transform: none; }

    .nav\_\_link--logo svg {

      width: 200px;

      margin: 0; }

## Predefined Colors

You will notice the $black, $white etc in the css files. These values are shortcuts which are defined in the \_variables.scss file inside the /scss/partials folder.

$black: #303030;

$white: #fdfdfd;

$white: #0602fd;

$width: 800px;

$purple: #262161;

$yellow: #FFC40E;

$pink: #EE7297;

$green: #93FF00;

$red: #E50714;

$grey: lighten(grey, 40%);

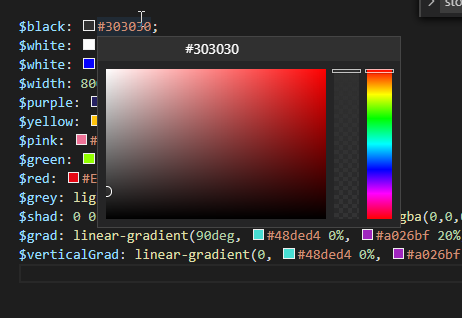
$shad: 0 0 10px rgba(0,0,0,0.1), 0 5px 10px rgba(0,0,0,0.05);

$grad: linear-gradient(90deg, #48ded4 0%, #a026bf 20%, #e82c75 60%, $yellow 85%, #48ded4 95%);

$verticalGrad: linear-gradient(0, #48ded4 0%, #a026bf 20%, #e82c75 60%, $yellow 85%, #48ded4 95%);

## Using a Palette Seletor

You will notice in Visual Studio Code you can hover over the color and it allows you to change it from a user-friendly palette that looks like the following.



## Icons and Images

You can change the Icons of the navigation and other things by changing their names to match svg files or png files which you place in the /public/icons folder.

## Logging In/Registering/Authorization/LogOut

User interaction requires a few things. Besides the obvious, getting logged in, logging out and giving the user access to some functionality based on being logged in there are other things that go with a user. You need to register/signup new users. You need to provide logout and forgot password ability so they can handle their own issues if they have trouble logging in. You need to consider the session around the login which will also be useful in a stateless web environment.

There needs to be rules around the password and the password should normally be encrypted in the db.

Thankfully I this is all implemented in the code you have as a starter. But you will need to be familiar with it as well as be ready to make modifications to this code when needed for making the application do things differently and implement new functionality.

The authController.js file in the /controllers folder is where your authorization process takes place. This is also where most of the above-mentioned functionality is housed.

There are several forms that control the user login experience. They are all in the /views and /views/mixins folders.

\_forgot.pug

\_loginForm.pug

Login.pug

Register.pug

As with the rest of the pages the main views use mixins for detailed implementation. Mixins start with an \_ which is a standard I adopted when I learned NODE.

## Passport

We use a middleware called Passport for authentication. Documentation can be found here: <http://www.passportjs.org/docs/>

Our implementation is in the authController class in the /controllers folder.

You will also find reference to passport in the app.js class. This is where we pre-load information needed for login and session handling.

We have a passport handler in the /handlers folder. This is used for interaction with the userController and model.

In the user.js in the /models folder we implement that Passport tool for interaction with Mongoose so we have encryption capability to Mongo.

This provides reset tokens and other capabilities so it is not easy to hack our software. While this is relatively secure, we would need to do a little bit heavier lifting to make this system secure at a level we would use in a financial system.