Contents

[Document Goal 2](#_Toc48504316)

[Assignment 2](#_Toc48504317)

[A Little About Pug 2](#_Toc48504318)

[A Little About Helpers 3](#_Toc48504319)

[Promise Usage 4](#_Toc48504320)

[Slugs 4](#_Toc48504321)

[Saving A Maze (using the Model) 5](#_Toc48504322)

[Querying our Database for Mazes 8](#_Toc48504323)

[CSS 8](#_Toc48504324)

[Styles.css 8](#_Toc48504325)

[Mazecontroller.js 12](#_Toc48504326)

[\_mazeForm.pug 12](#_Toc48504327)

[mazeWorker.js 13](#_Toc48504328)

[Navigation 14](#_Toc48504329)

[Adding Menus 15](#_Toc48504330)

[Login 16](#_Toc48504331)

[My Mazes Implementation 17](#_Toc48504332)

[Customizing Appearance 17](#_Toc48504333)

[Changing Navigation Look & Feel 17](#_Toc48504334)

[Predefined Colors 19](#_Toc48504335)

[Using a Palette Seletor 19](#_Toc48504336)

[Icons and Images 20](#_Toc48504337)

[Logging In/Registering/Authorization/LogOut 20](#_Toc48504338)

[Passport 20](#_Toc48504339)

[The Maze 21](#_Toc48504340)

[Editor for HTML learning 22](#_Toc48504341)

[Information on tables 23](#_Toc48504342)

[Information on borders 23](#_Toc48504343)

## 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 the next 6 weeks of 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.

The next part of the homework will be exploring how we display our maze. There are a number of ways to display a maze. The current code, provided in the starter code used a text display. What I have done and what I recommend is to use a table where you control the cell borders to create the maze. You can choose another way or keep the text output. But you will need to understand how to keep track of the route the user has traversed, where they made mistakes and what paths are possible.

Finally, we will need to know how to save the maze in the DB for redisplay.

Mongo supports a type of Array. So we could save the array(s) we currently have as a base object. We could later add array(s) that track where you have been and your current route found.

## A Little About Pug

How pug works is you don’t write open and closing HTML tags. Instead you just do the open tag that you want, and then the text beside it like so: p Hello!

you have: tag, space and then all of your elements inside of it If you wish to nest elements, you need to indent it one level:

div

p Hello!

You can add classes and ids like so:

div.wrapper

p.hello Hello!

span#yo Yo!!!

If it’s just a div, you don’t need to specify div. If you leave the element type out it will assume that it is always a div.

We also need to know about attributes… you put them in parenthesis like so: img(src=“dog.jpg” alt=“dog”) (you can separate the attributes with commas)

If you want to put something on its own line without it rendering as an element you use | like so:

h2

| Hello

em How are you?

To get info from the request into the “hello” template, you use the second parameter on the render() method.

In pug, you can reference the variables that were passed in the second param of the render() method.

You interpolate a variable inside of text in pug like #{dog}

If you want to use a variable into an attribute, you do it like in javascript:

img.dog(src=“dog.jpg” alt=`dog ${dog}`)

[6F024006 96ED 45EE A14D BF1B4E606524](https://notes.anjagusev.com/static/2fc064e25eca4177d633da5b03a742e9/01387/6F024006-96ED-45EE-A14D-BF1B4E606524.png)

If you need to make a variable in pug, you can do it using a dash like so:

-const upDog = dog.toUpperCase();[5F3ADA7E 71F0 4F5A 9D6B E865738FCA80](https://notes.anjagusev.com/static/b6d995952b987e0ecc88af25f45b4eb9/7adc3/5F3ADA7E-71F0-4F5A-9D6B-E865738FCA80.png)

You can run javascript in Pug.

You can reuse parts of website like footers and headers.

You do that by extending templates, and making “layouts”

We want to extend the layout.pug

extends layout

block content

p Hello

You can also overwrite the layout (default) portions by referencing them in the extends file.

## A Little About Helpers

Sometimes you need data available in every single request.

We will use a helpers.js file and in that file puts any helper libraries or data that is needed in every single template.

exports.menu = [

{ slug: "/mazes", title: "Mazes", icon: "maze" },

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

… etc

]

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}

In middleware we can append variables to all our requests

app.js

// pass variables to our templates + all requests

app.use((req, res, next) => {

res.locals.h = helpers

res.locals.flashes = req.flash()

res.locals.user = req.user || null

res.locals.currentPath = req.path

next()

})

You can export arrays, strings ,or entire libraries

Example: exports.moment = require(‘moment’);

## Promise Usage

You will need to use promise to access the DB so your UI does not feeze.

Get access to promise with a module called promise.

**npm install promise**

When defining promises, it needs to be noted that the "then" method itself returns a promise. So in a sense, promises can be nested or chained to each other.

## Slugs

You are going to see the term SLUG in the code. This refers to a group of data as an object that we are calling slug after the middleware that uses “slugs” to help us display data easier.

Slugs are part of Markdown framework which is another class already added for you.

A slug is a URL markdown friendly way to show a title in the URL or a string. Here is a JAVAScript function to create a slug from a string.

function convertToSlug(Text)

{

return Text

.toLowerCase()

.replace(/[^\w ]+/g,'')

.replace(/ +/g,'-')

;

}

I recommend doing data normalization close to the model. It is where you go for data change and it should be where you go for normalization.

## Saving A Maze (using the Model)

\*\*\* NOTE \*\*\*

If you’re getting a URIError: URI malformed error when running npm start, break out your environment variables. Go into variables.env and split the URI like this MONGO\_URI=mongodb://host.com:port DB\_USER=username and DB\_PASS=password. Then inside your start.js replace mongoose.connect(process.env.DATABASE) with mongoose.connect(process.env.MONGO\_URI, {user: process.env.DB\_USER, pass: process.env.DB\_PASS});. I had issues connecting to my mongodb because my password contained symbols.

\*\*\* MONGOD complains that there is no /data/db folder

Note:

MongoDB also has an option where you can create the data directory in another location, but that's generally not a good idea, because it just slightly complicates things such as DB recovery, because you always have to specify the db-path manually. I wouldn't recommend doing that.

Edit:

the error message you're getting is **"Unable to create/open lock file: /data/db/mongod.lock errno:13 Permission denied"**. The directory you created doesn't seem to have the correct permissions and ownership -- it needs to be writable by the user who runs the MongoDB process.

To see the permissions and ownership of the '/data/db/' directory, do this: (this is what the permissions and ownership should look like)

$ ls -ld /data/db/

drwxr-xr-x 4 mongod mongod 4096 Oct 26 10:31 /data/db/

The left side 'drwxr-xr-x' shows the permissions for the User, Group, and Others. 'mongod mongod' shows who owns the directory, and which group that directory belongs to. Both are called 'mongod' in this case.

**If your '/data/db' directory doesn't have the permissions and ownership above, do this**:

First check what user and group your mongo user has:

# grep mongo /etc/passwd

mongod:x:498:496:mongod:/var/lib/mongo:/bin/false

You should have an entry for mongod in /etc/passwd , as it's a daemon.

sudo chmod 0755 /data/db

sudo chown -R 498:496 /data/db # using the user-id , group-id

You can also use the user-name and group-name, as follows: (they can be found in /etc/passwd and /etc/group )

sudo chown -R mongod:mongod /data/db

that should make it work..

In the comments below, some people used this:

sudo chown -R `id -u` /data/db

sudo chmod -R go+w /data/db

or

sudo chown -R $USER /data/db

sudo chmod -R go+w /data/db

The disadvantage is that $USER is an account which has a login shell. Daemons should ideally not have a shell for security reasons, that's why you see /bin/false in the grep of the password file above.

------------------------

Models are where our data is going to be stored, and before we can create a piece of data we need to describe what that data will look like.

A model can tell what type of data is expected (string ,array), cleanup before the data is saved, creating a slug.

MongoDB can be a loose database, meaning you don’t need to specify what your data will look like ahead of time.

We interface with mongoldb with mongoose package

const mongoose = require(‘mongoose’);

We also need to tell mongoose that the promise to use is the global promise. mongoose.Promise = global.Promise What does that mean?

When we get into querying our db, there are a few ways we can wait for the data to get back because it happens asynchronously. You can uses the built in callbacks, you can use external library like bluebird or since we are learning about async await, we are using the built in es6 promise.

We set the mongoose property to be global (sort of like the window in browser).

Don’t put things on the global, it’s generally not what you want.

we import slugs, which allow us to make url friendly names const slug = require(‘slugs’);

If the main thing you are exporting from a file is going to be importable, then you can put it on modules.exports. for example modules.export=router

However for storeController.js we are exporting exports.homePage .

When you import a package, is the main thing you import from it a function or are you just importing an object that has many properties on it?

Do all your data normalization as close to the model as possible.

For the model, you can just put the property name and then the type, but you can also pass in an object for the property.

How do we make mongo know about the model?

Go to start.js file. You only need to import it once. It’s using a singleton, once you import it you don’t have to import it in every single file.

The slug property in our Store model is auto generated whenever someone saves.

What we use for that is a pre-save hook in MongoDB. Before someone saves a maze, we are going to auto-generate the slug field.

We do that with this code:

mazeSchema.pre(‘save’, function(next){

this.slug = slug(this.name);

})

(Don’t use arrow function because we need to reference this, which is the maze)

We call .next() because this is pre-save and we want to pass it on to the save.

Right now, the pre-save will run overtime we save but we only want it to run when the name is updated.

Modify method like so:

if(!this.isModified(‘name’)){

next();//skip it

return; //stop the function from running ( you can also do return next(); in one line

}

this.slug=slug(this.name);

next();

});

## Querying our Database for Mazes

We are going to display the mazes in the database on the My Mazes tab. We need a controller method that is going to run on both of those routes.

Modify homepage route and add another route for /stores and wrap them in error handlers since they will be async await.

router.get('/maze', catchErrors(mazeController.myMazes));

Now we create another method in mazeController.js and make it async await. This method is responsible for returning all the stores:

exports.myMazes = async (req, res) => {

//1. Query database for a list of all mazes

const mazes = await Maze.find()

console.log(mazes)

res.render("mazes", { title: "My Mazes", mazes })

}

We want to make the stores variable available to our actual template like so: res.render(“mazes”, { title: “My Mazes”, mazes: mazes });

You can access it now in mazes.pug like so: each store in stores

We want to render individual files in a separate mixin however called a “maze card” which I am calling \_mazeView.pug to display each store.

If you want to display the description but limit it to 25 words for example you can do javascript right in pug like so: p= maze.description.split(‘ ‘).slice(0, 25).join(‘ ‘)

# CSS

## Styles.css

You need to add a style for your table. You can get the format from the class table.scss. This is how you do table formatting with CSS in the style.css class. You only need to really have place holders for .table, and tr. Then for TD you need some definitions. This is an example of mine. You can see I am using the keywords table, tr and td. I am adding right, left, top, bottom along with read versions of this. The reason is future use so I can build a red trail for what a user has traversed.

 table {

    font: 11px/24px Verdana, Arial, Helvetica, sans-serif;

    border-collapse: collapse;

    }

  tr {

    border-collapse: collapse;

     }

  td {

    border-bottom: 1px solid #FFFFFF;

    padding: 0 0.5em;

    }

  td.right {

    border-right: 2px solid #000000;

    }

  td.left {

    border-left: 2px solid #000000;

    }

  td.bottom {

    border-bottom: 2px solid #000000;

    }

  td.top {

      border-top: 2px solid #000000;

    }

  td.redTop {

    border-top: 2px solid #000000;

    }

  td.redRight {

    border-right: 2px solid #000000;

    }

  td.redLeft {

    border-left: 2px solid #000000;

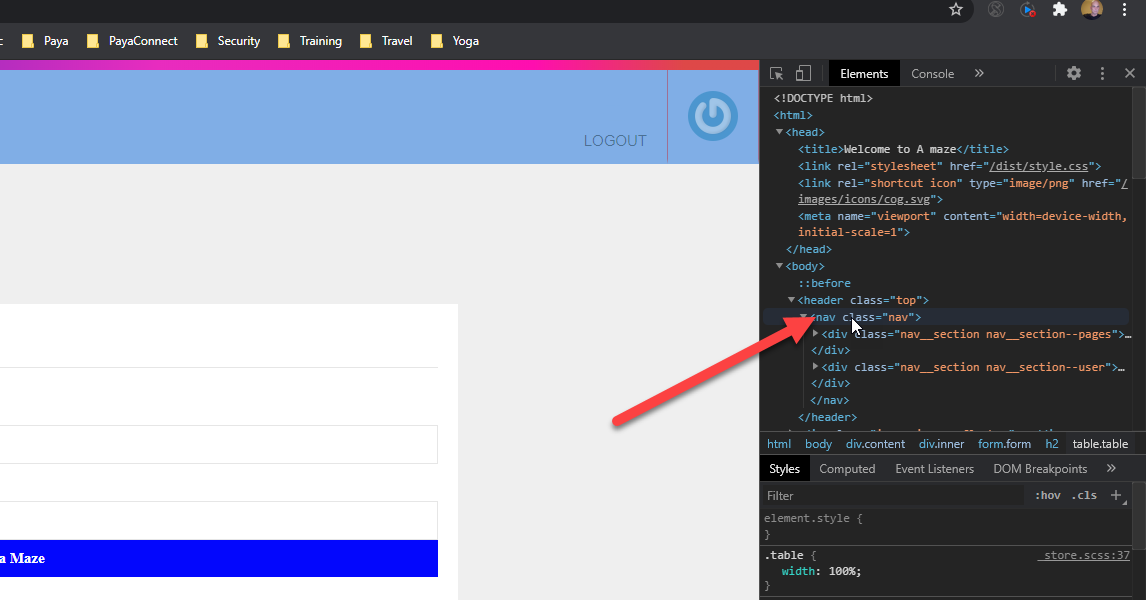
    }

  td.redBottom {

    border-bottom: 2px solid #000000;

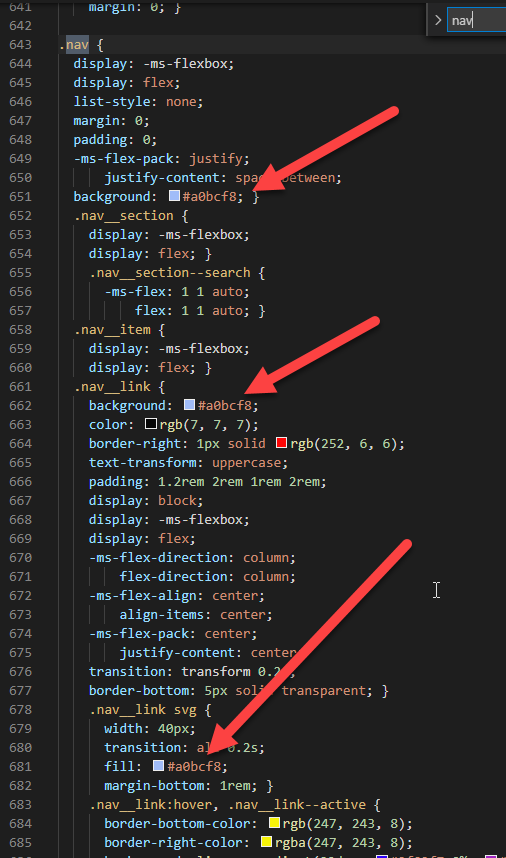
    }

I also changed my header for the page to have some background color. You can figure this out with the developer tools in Chrome.



You can see the class is called NAV. So I modify the code in the NAV section of the style.css class.

I need to change three places to make the color all the same.



# Mazecontroller.js

Not much to change here. Because I only really changed the function name I call.

    //console.log(req.body);

    //console.log("Found request for maze that is " + req.body.height + " by " + req.body.width + " in size!");

    // lets check the size and if ok send us the maze display

    if (req.body.height >= 5 && req.body.width >= 5 && req.body.height <= 70 && req.body.width <= 70) {

      maze.width = req.body.width;

      maze.height = req.body.height;

      mazeWorker.getTableMaze(maze.width, maze.height).then((result) => {

          maze.drawing = result;

          res.render('main', { title: 'Welcome to A maze',  maze });

      }).catch((error) => {

          console.log("Error", error);

          response.end("Error: " + error);

      })

# \_mazeForm.pug

I made quite a bit of change here. I setup a button for saving the maze to the mongo.db and I changed how the maze displays.

mixin mazeForm(maze = {})

  form.form(action='/' method="POST")

    h2 Size

    label(for="width") Width

    input(type="text" name="width" value=maze.width, maxlength='6', style="witdh:5px")

    label(for="height") Height

    input(type="text" name="height" value=maze.height, maxlength='6', style="witdh:5px" )

    input(type="submit" value="Get a Maze" class="button")

    - var x = 0

    - var y = 0

    h2 Maze

      p !{maze.drawing}

  form.form(action='/saveSave' method="POST")

    input(type="submit" value="Save To My Mazes" class="button")

Specifically I changed from a PRE to paragraph object. You can see this here on this line.

    h2 Maze

      p !{maze.drawing}

# mazeWorker.js

This class is the hardest to change of all of them for this work assignment. Here is some sample code you can see what your code should do in order to display a maze. This is just example code of what your code should output.

            let sb = new StringBuilder();

            let newColumn = true;

            // table borders

            sb.append("<table class=\"table\">");

            // uncomment for example table that is simple use of the css classes for

            sb.append("<td class=\"left bottom\">\_</td>");

            sb.append("<td class=\"top bottom\">\_</td>");

            sb.append("<td class=\"top right\">\_</td>");

            sb.append("</tr><tr class=\"noBorder\">");

            sb.append("<td class=\"top bottom right left\">\_</td>");

            sb.append("<td class=\"top bottom right left\">\_</td>");

            sb.append("<td class=\"right left\">\_</td>");

            sb.append("</tr>");

            sb.append("</table>");

Once you understand this code you can modify the function to build a table to display with only the correct columns highlighted.

We are using a string builder so you see the sb.append. This is a more efficient use of memory for all the many strings we need to append to build our output.

## 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%);

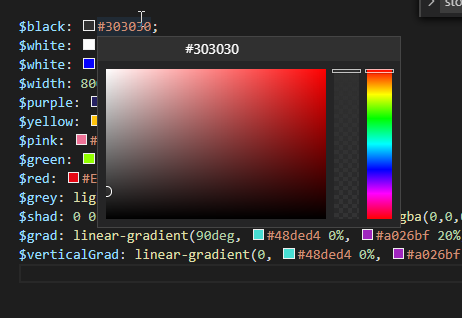
$shad: 0 0 10px rgba(0,0,0,0.1), 0 5px 10px rgba(0,0,0,0.05);

$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.

## The Maze

This part of our homework today we will be exploring how we display our maze. There are a number of ways to display a maze. The current code, provided in the starter code used a text display. What I have done and what I recommend is to use a table where you control the cell borders to create the maze. You can choose another way or keep the text output. But you will need to understand how to keep track of the route the user has traversed, where they made mistakes and what paths are possible.

Finally, we will need to know how to save the maze in the DB for redisplay.

Mongo supports a type of Array. So we could save the array(s) we currently have as a base object. We could later add array(s) that track where you have been and your current route found.

What I did was copy the getMaze function and create a function called getTableMaze. I changed several things about this function and the PUG that displays the maze in order to accomplish a well done maze.

Here are some things that you will need to solve.

* CSS to control the cell borders. (We will later use this to control colored route taken)
  + Styles.css class
* Changes to the maze controller
* Changes to the mazeWorker.js
* Changes to the PUG file \_mazeForm.js
* Changes to the Maze.js model. (optional)

# Editor for HTML learning

I use a couple of online editors to test CSS, JS and HTML working together. They are both free for small things. You can find them here.

<https://htmlcodeeditor.com/>

&

<https://html-online.com/editor/>

You can paste in the code I gave you in the homework helper file combined with the CSS code surrounded with the <style> tag and you can test changes outside your program for display.

So the top of your code pasted in the editors may look like this.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Vertical / Horizontal Line Tester</title>

<style>

table {

font: 11px/24px Verdana, Arial, Helvetica, sans-serif;

border-collapse: collapse;

}

tr {

border-collapse: collapse;

}

td {

border-bottom: 1px solid #FFFFFF;

padding: 0 0.5em;

}

td.right {

border-right: 2px solid #000000;

}

td.left {

border-left: 2px solid #000000;

}

td.bottom {

border-bottom: 2px solid #000000;

}

td.top {

border-top: 2px solid #000000;

}

td.redTop {

border-top: 2px solid #ff0000;

}

td.redRight {

border-right: 2px solid #ff0000;

}

td.redLeft {

border-left: 2px solid #ff0000;

}

td.redBottom {

border-bottom: 2px solid #ff0000;

}

</style>

</head>

<body>

# Information on tables

<https://www.w3schools.com/css/css_table.asp>

# Information on borders

<http://jkorpela.fi/html/cellborder.html>

There are other resources but these are really good ones.

You will need to build a table on the fly. This will require you to keep track of where you are in the table and what elements are required.