Contents

[Class Homework Help 2](#_Toc47354228)

[Document Goal 2](#_Toc47354229)

[Index.js file understanding 2](#_Toc47354230)

[What is Routing 2](#_Toc47354231)

[Implementation of routing in Node.js 2](#_Toc47354232)

[Using Framework 2](#_Toc47354233)

[Routing with Express in Node 2](#_Toc47354234)

[For GET request use app.get() method 3](#_Toc47354235)

[For POST request use app.post() method 3](#_Toc47354236)

[For handling all HTTP methods (i.e. GET, POST, PUT, DELETE etc.) use app.all() method 3](#_Toc47354237)

[Controller For Our Routes 3](#_Toc47354238)

[UI PUG usage 4](#_Toc47354239)

[So what are Mixins? 5](#_Toc47354240)

[Workers 5](#_Toc47354241)

[CSS and SASS (scss) 6](#_Toc47354242)

# 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.

The preliminary coding will be harder than a junior node programmer may understand. But it is all groundwork so we can build a usable application and learn the basic things that node relies on. So, I hope this makes it more fun and challenging so you learn faster and better.

## Index.js file understanding

The routes are important for your first homework. The routes are already really done for you. The controller is created for the maze which will become our root URL handler. You can see this with the code that is in the starter route index.js file.

router.get('/', catchErrors(mazeController.main));

router.post('/', catchErrors(mazeController.main));

To help you understand what this means and what it is for I am providing some explanation for you.

What is Routing**?**  
Routing defines the way in which the client requests are handled by the application endpoints.

Implementation of routing in Node.js**:**

There are two ways to implement routing in node.js which are listed below:

* By Using Framework
* Without using Framework

Using Framework**:**

Node has many frameworks to help you to get your server up and running. The most popular is Express.js. For our class we will be using Express.

Routing with Express in Node**:**

Express.js has an “app” object corresponding to HTTP. We define the routes by using the methods of this “app” object. This app object specifies a callback function, which is called when a request is received. We have different methods in app object for a different type of request.

You will notice the routes I have provided in your template code are:

/

/user

/login

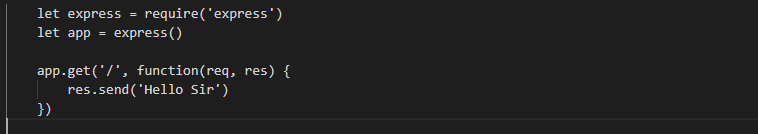
/register

/maze (commented out)

Etc

These show you what the distinct capabilities of the site are if I have correctly separated my controllers and logic.

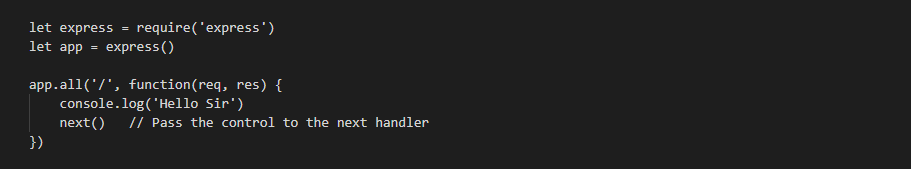
For GET request use app.get() method**:**



For POST request use app.post() method**:**



For handling all HTTP methods (i.e. GET, POST, PUT, DELETE etc.) use app.all() method**:**



## Controller For Our Routes

Instead of writing directly to the routes, we factor that code out to its own controller.

There is a folder called controllers.

I like to make a controller for every specific area of the website.

On the controller, we have to export all of the different functions that happen. This allows them to be accessed and referred to in our index.js file for our route handling. Like the following code:

router.get('/', catchErrors(mazeController.main));

router.post('/', catchErrors(mazeController.main));

router.get('/user', catchErrors(userController.main));

router.post('/user', catchErrors(userController.main));

This handling comes from the MVC pattern we will be building our site based on.

Just like C# MVC patterns.

M = Model (ex: db access)

The part of our application that will deal with the database or any data-related functionality.

Think of the Model as a way to abstract the data into a “model” of something that can store the data. So, we can create functions that do things like save, delete, update and other functions to the data without messy manipulation.

V = View (pug)

Everything the user will see. Basically, the pages that we’re going to send to the client.

C = Controller (ex: controls flow of data )

The logic of our site, and the glue between models and views. Call the models to get the data, then put that data on the views to be sent to the users.

Middleware - Middleware has access to request and response object

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware in the stack.

## UI PUG usage

We are going to be using a node module called PUG to describe our output and UI. You will find these in the /views folder.

I have provided you with a main.pug that is the display page you will be updating for the maze display. The following code expects a maze object to be passed. You will be passing maze model object. So you would expect to modify this under the /models folder. The file is maze.js. This describes the data that makes up a maze and will be eventually stored in the DB. Don’t get too wound up about how that all glues together yet.

extends layout

include mixins/\_mazeForm

block content

  .inner

    h2= title

    +mazeForm(maze)

You will notice I am using Mixins. This code is where it is referenced in the main.pug file above.

    +mazeForm(maze)

## So what are Mixins?

Think of Mixin as a way that you can bypass the fact that in JavaScript and hence NODE you can only inherit from a single object. The mix of two objects, object and object is what you may consider a mixin. In simple terms anyway. “A [mixin](https://en.wikipedia.org/wiki/Mixin) is a class containing methods that can be used by other classes without a need to inherit from it.”

A mixin for PUG is a way to create reusable blocks of PUG code. This makes editing reused logic centralized. Specifically around UI.

Our mixin above comes from our usage at the top of the pug file that is this code:

include mixins/\_mazeForm

So you would expect to look in /views/mixin/\_mazeForm.pug to edit the information that is “re-usable” in the mixin.

## Workers

One of your task was to create a worker class.

For my expected outcome I created a worker who does both the generate a Maze array and create a displayable copy of the Maze array. This is just the code from our last homework with some rework to run in our app website. So these two functions make up my worker class.

function buildMaze(x,y) {

function getMaze(x, y) {

These signatures are different slightly than you currently have. This is because of something you may learn and I can warn you about. Because of the way these two functions work together you will want to call the getMaze which replaces our *display* function and calls the buildMaze for you and replaces our *Maze* function from before. The reason for this change is the random generator does not work the way we were calling it when we use it in a website callback model. I can’t tell you all of the technical reasons why. Because I don’t know. But I do know the randomizer gets broken and therefore the maze will make no sense if you don’t recognize this.

## CSS and SASS (scss)

I have provided you with a lot of CSS code that controls the buttons, colors and menus, to only name a few, functionality as well as look and feel. You can modify this if you know CSS. This will let you make the maze system look the way you like it. If you change it radically, you may find future versions of my code will not work with yours without keeping track of those changes.