Contents

[A Little About Pug 2](#_Toc47430203)

[A Little About Helpers 3](#_Toc47430204)

[Promise Usage 4](#_Toc47430205)

[Slugs 4](#_Toc47430206)

[Saving A Maze (using the Model) 4](#_Toc47430207)

[Querying our Database for Mazes 7](#_Toc47430208)

[Homework Assignment 8](#_Toc47430209)

## 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(‘ ‘)

## Homework Assignment

See Class Notes 08-15-2020 Homework Help.docx for extra help on how to do this homework.

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.