**Web Frameworks - Task Manager Web Application**

**T00205890 Michal Kubiak 3rd year Software Development**

# **Summary**

**This is a fully functional application which is now deployed to Heroku at:**

[*http://web-frameworks-task-manager.herokuapp.com/*](http://web-frameworks-task-manager.herokuapp.com/)

*GitHub - https://github.com/michalkubiak98/Node-Express-MongoDB-Task-Manager*

**The frameworks used in this so far are NodeJS, Express and MongoDB. Angular to be added.**

**The app uses PassportJS’ Google OAuth2 Strategy for authentication. Some other packages are also used and specified below during the build process examination.**

Table of Contents

[**Summary** 1](#_Toc54627059)

[**Introduction** 2](#_Toc54627060)

[**MongoDB Connection** 3](#_Toc54627061)

[**Models** 4](#_Toc54627062)

[**Views** 6](#_Toc54627063)

[**Routes** 10](#_Toc54627064)

[**Tasks** 14](#_Toc54627065)

[**Public Tasks** 18](#_Toc54627066)

[**Full Task Page (Read More Button on a Task)** 23](#_Toc54627067)

[**Particular User Tasks Sort** 23](#_Toc54627068)

[**Controllers** 24](#_Toc54627069)

[**Heroku Deployment** 25](#_Toc54627070)

# **Introduction**

**I started by initializing a MongoDB project cluster and copying the web app URI from it, this is later saved into an environmental variables’ configuration file.**

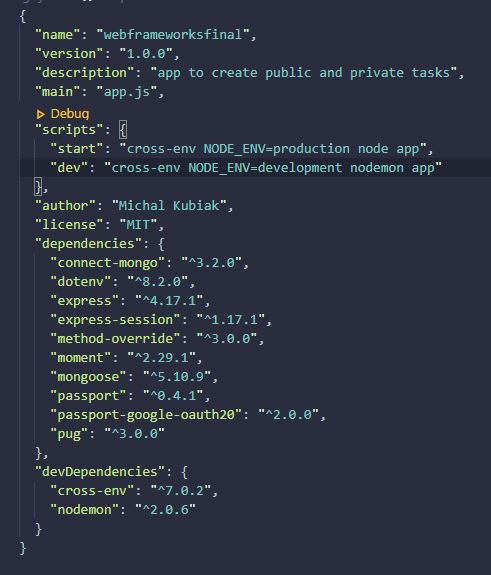
**To start the project, I ran the *npm init* command in the terminal from the project’s folder. Entry point set to app.js as this is a complete backend production application.**

**I installed the packages as I needed them but all can be installed using one command:**

1. ***npm install express mongoose connect-mongo pug express-session dot-env cross-env passport passport-google-oauth20 method-override moment morgan passport***

**I installed nodemon which is used for the development sever reload on save using *npm I -D nodemon***

1. **I edited the start script (which I had to change before deployment to Heroku at the end) in the package.json file.**

****

1. **I then set up the app.js and config.env files. The app.js was initialized with express a localhost port and the config file with environmental variables. Grave accents (``) are used in the app.listen() function to pass the environmental variables from the config file.**

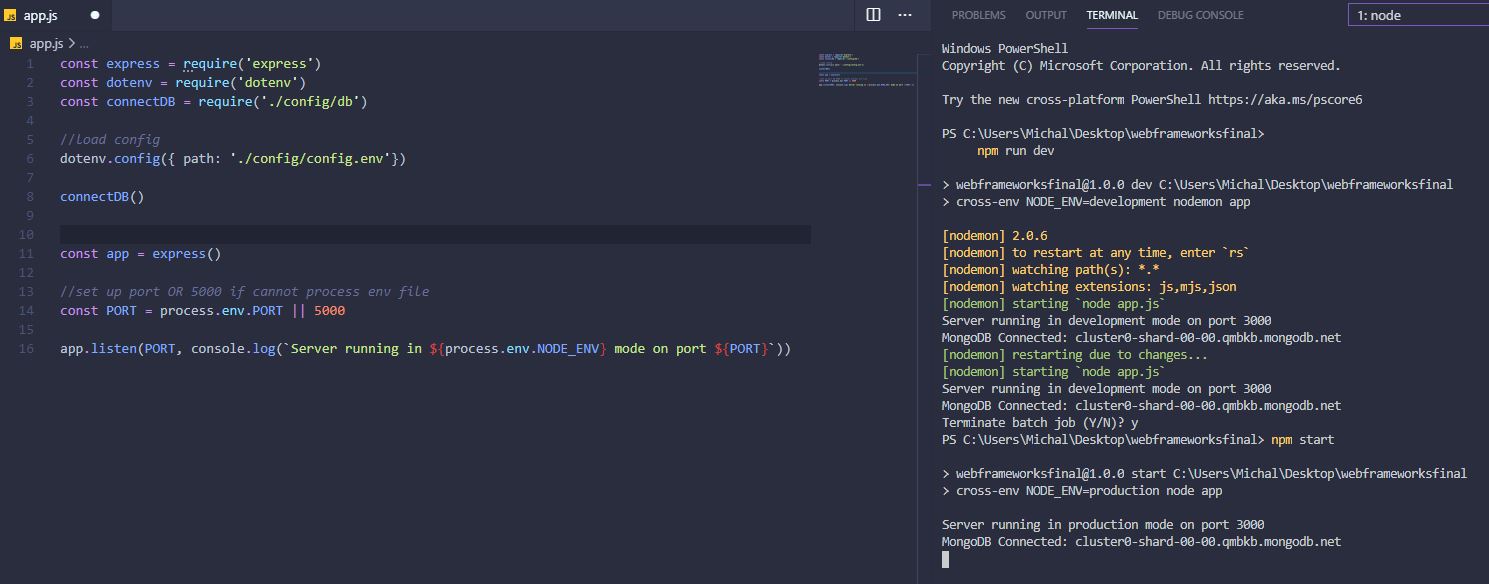
****

# **MongoDB Connection**

1. **Created a const connectDB function which uses async await as I m dealing with MongoDB inside the config folder under db.js. I am working with promises, mongoose.connect for example returns a promise, hence async await. I also passed an object with 3 functions in mongoose.connect with the MongoDB URI to stop warnings in the console. All wrapped in a try catch to handle errors.**

****

1. **I added const connectDB = require(‘./config/db’) in app.js, it now console logged correctly when running the dev environment (npm run dev), production environment started correctly too (npm start).**

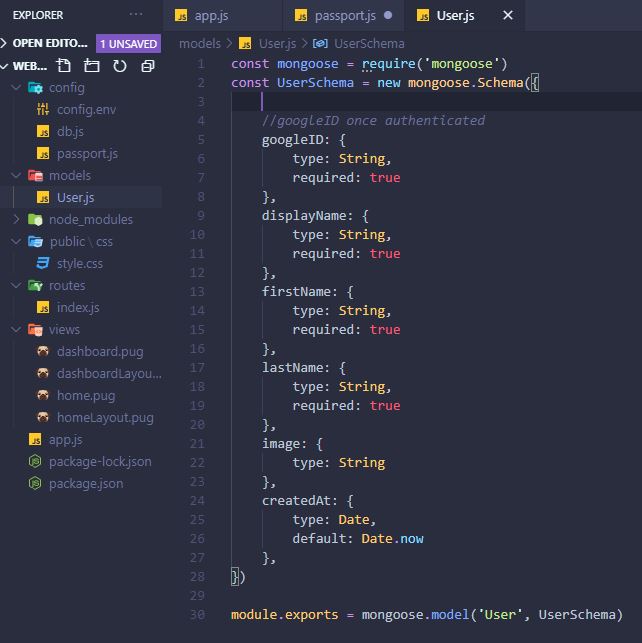
****

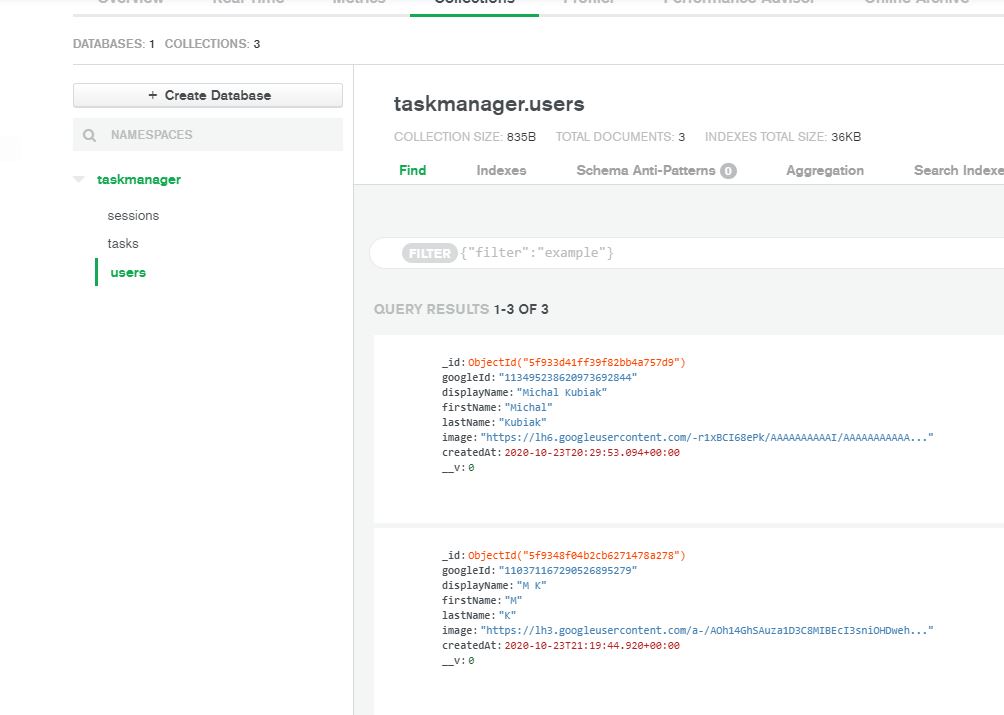
1. **I set up Morgan in app.js (for logging in the console all the requests the app makes for routes and pages for debugging during the development runtime).**

****

# **Models**

1. **I created the User model. googleID should have been googleId – small d - I got errors after the screenshot.**

****

****

# **Views**

1. **I set up the Pug template engine and Views folder, I ultimately put all views in one folder instead of separating them into different folders for ease of access. Added the following lines to app.js after const app = express()**

const path = require('path')

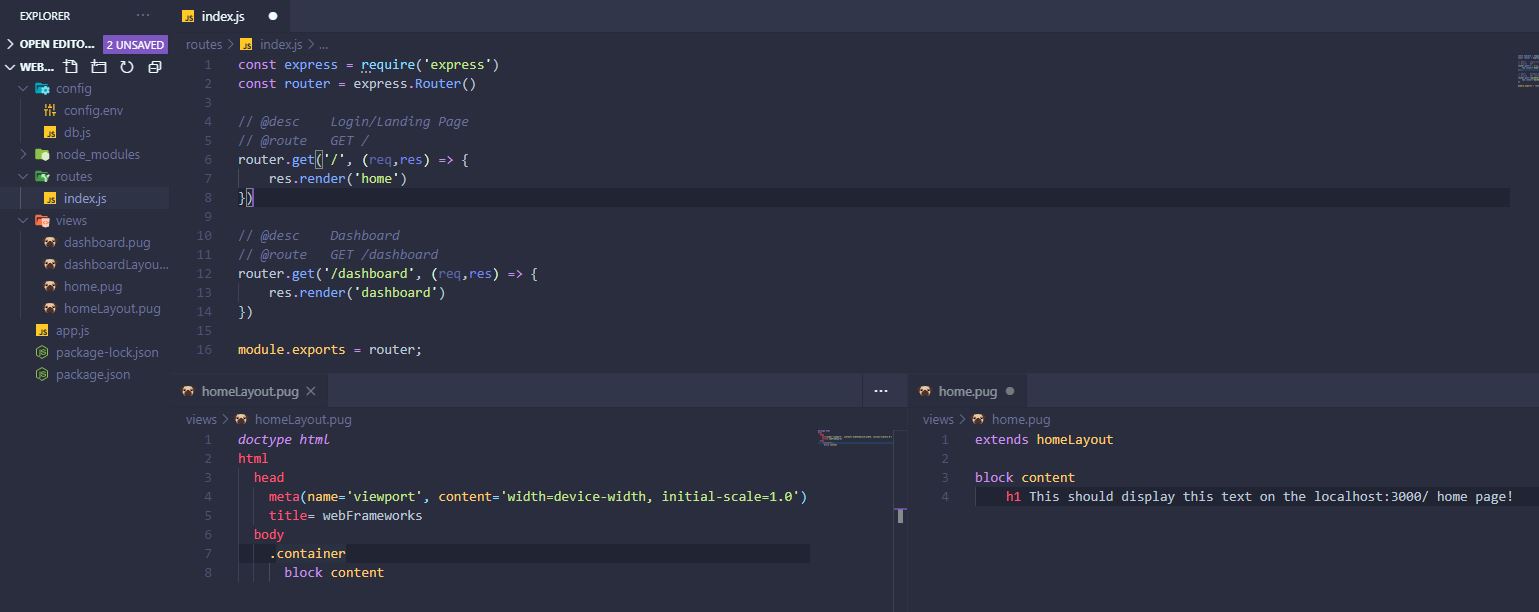
app.set('views', path.join(\_\_dirname, 'views'))

app.set('view engine', 'pug')

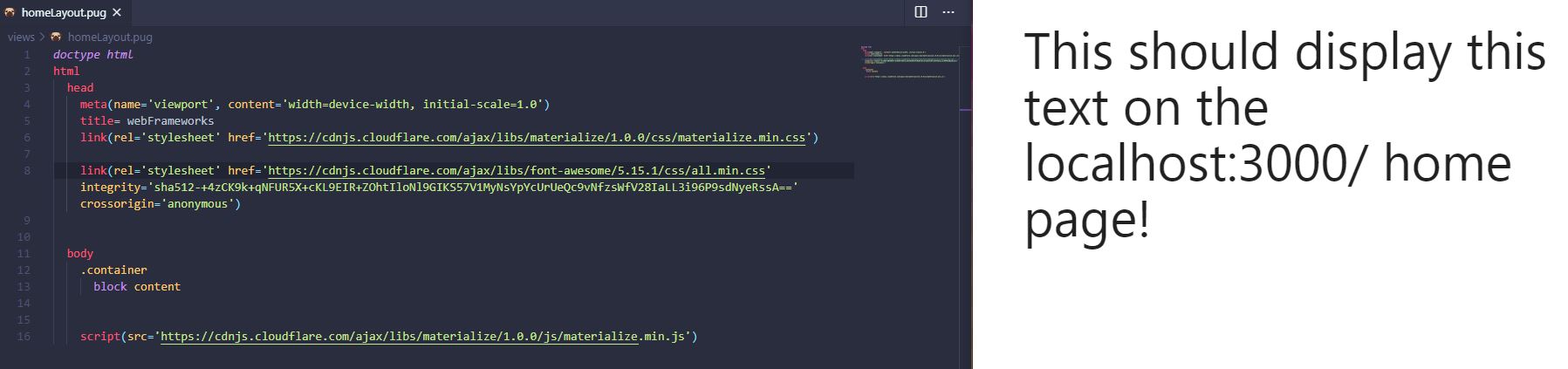
**I set up a main pug view (later called this home.pug) and a login pug view later called dashboard.**

****

1. **I set up routes and rendered the views, created an index.js file in routes which are top level routes, any route that isn’t followed by anything eg. /dashboard. Created a homeLayout and dashboardLayout views. Home is where the login is, this needs a different layout (No navigation bar for example), router.get() sets a route and you then pass in a req,res function => with an object { res.render(‘dashbaord’) }, this will render the dashboard pug view as I already set the view folder and view engine.**

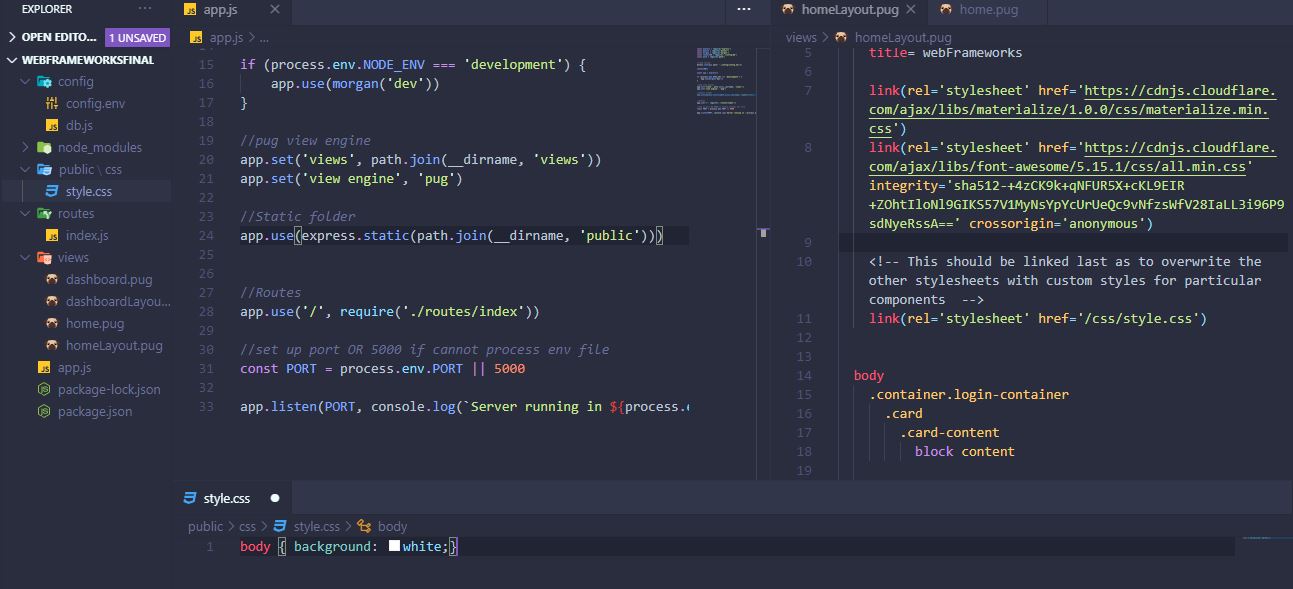
****

1. **I linked Materialize with a CDN in both layout pug views, it’s basically bootstrap but cleaner. Also linked fontAwesome’s CDN. Their classes can now be used in the pug views. In home.pug just wrote extends homeLayout, block content, h3 This should display this text……**

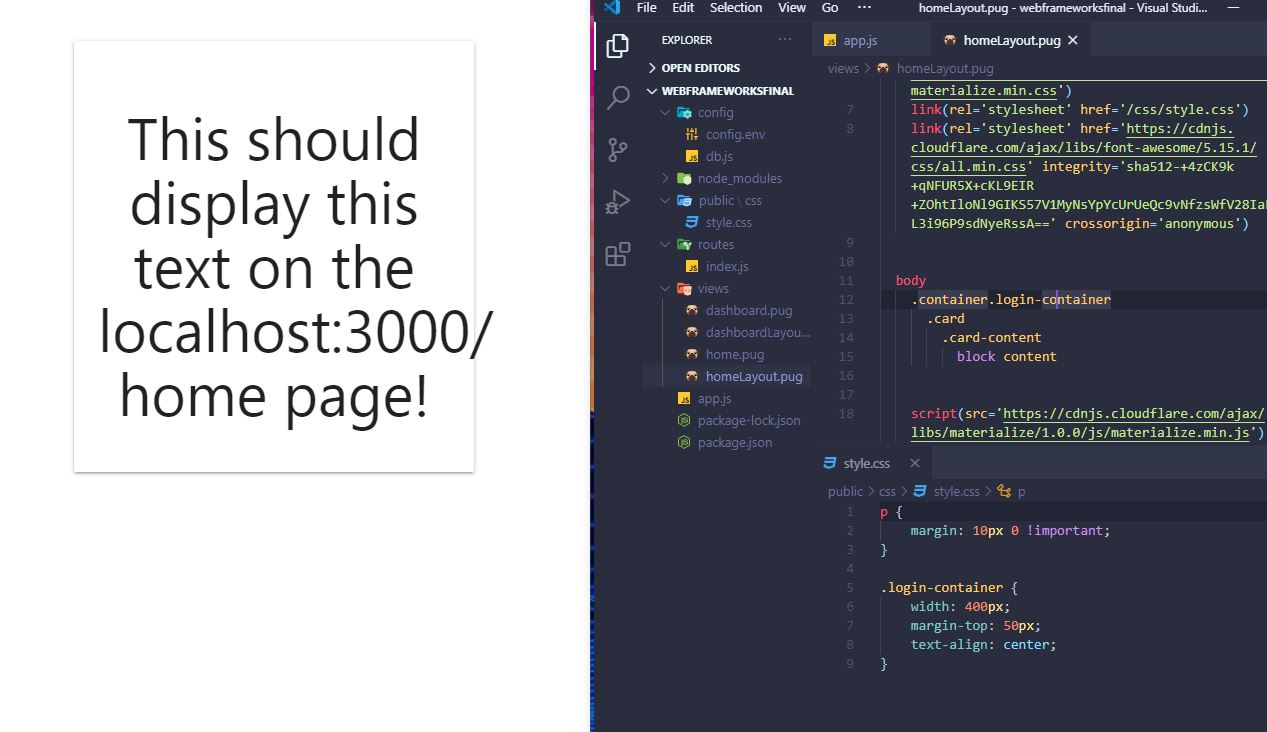
****

1. **I set up a static public folder for public CSS and other files like js if need be. This was linked in app.js. For path.join(\_\_dirname, you need to write const path = require(‘path’) at the top.**

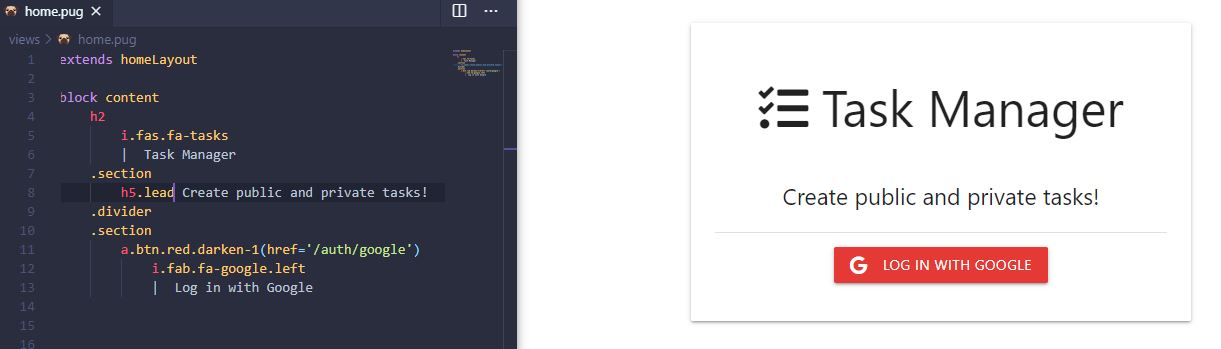
**I then linked the custom css stylesheet in the homeLayout and the dashboardLayout**

****

**I updated the home layout before the block content, anything in home.pug is now on a card, the login container class is a custom css class I wrote in style.css**

****

1. **I styled the home.pug view. The button link links to a route (auth/google) that I am about to initialize.**

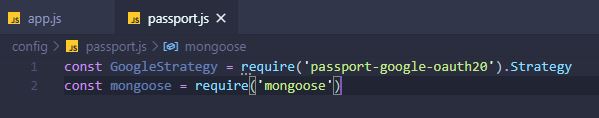
****

1. **To initialize Google OAuth 2.0 I need to get the API keys from the google cloud platform.**

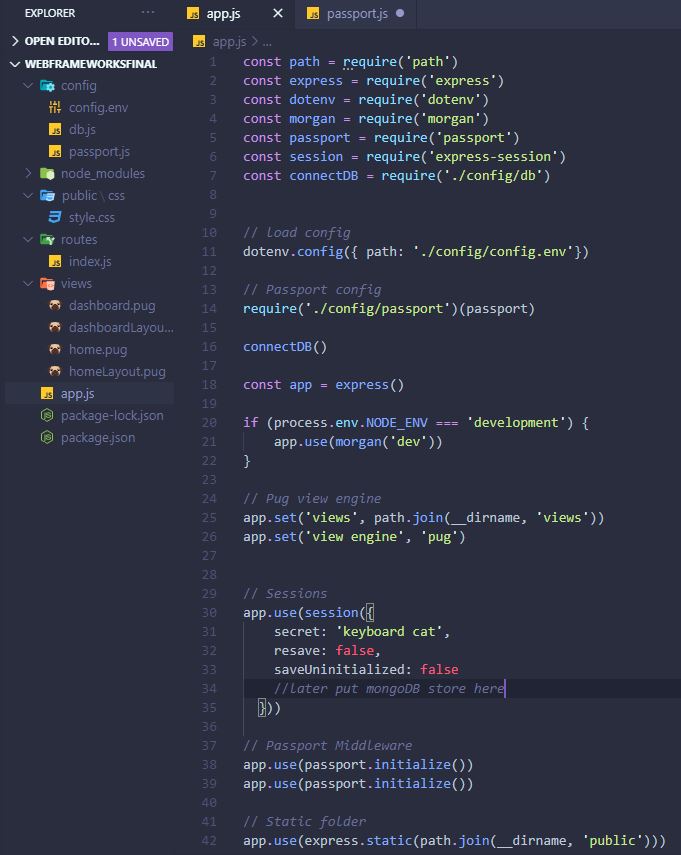
* **Sign in with a google account**
* **Create a new project in console**
* **Go to API and Services**
* **Click enable api and services on top**
* **Search for google+ API addon**
* **Click on it and enable it**
* **Go to credentials section on console or left menu**
* **Click on credentials and API services**
* **Click on create credentials**
* **+ OAuth Client ID**
* **Create OAuth consent screen if shows up**
* **Fill out name of app and save**
* **Create OAuth Client ID**
* **Web Application**
* **Under Authorised redirects URI’s add a callback URI (http not https)**
* **http://localhost:3000/auth/google/callback**
* **- This will later be saved to the Heroku link once its live**
* **Save**
* **Copy client ID and client secret into config.env file**



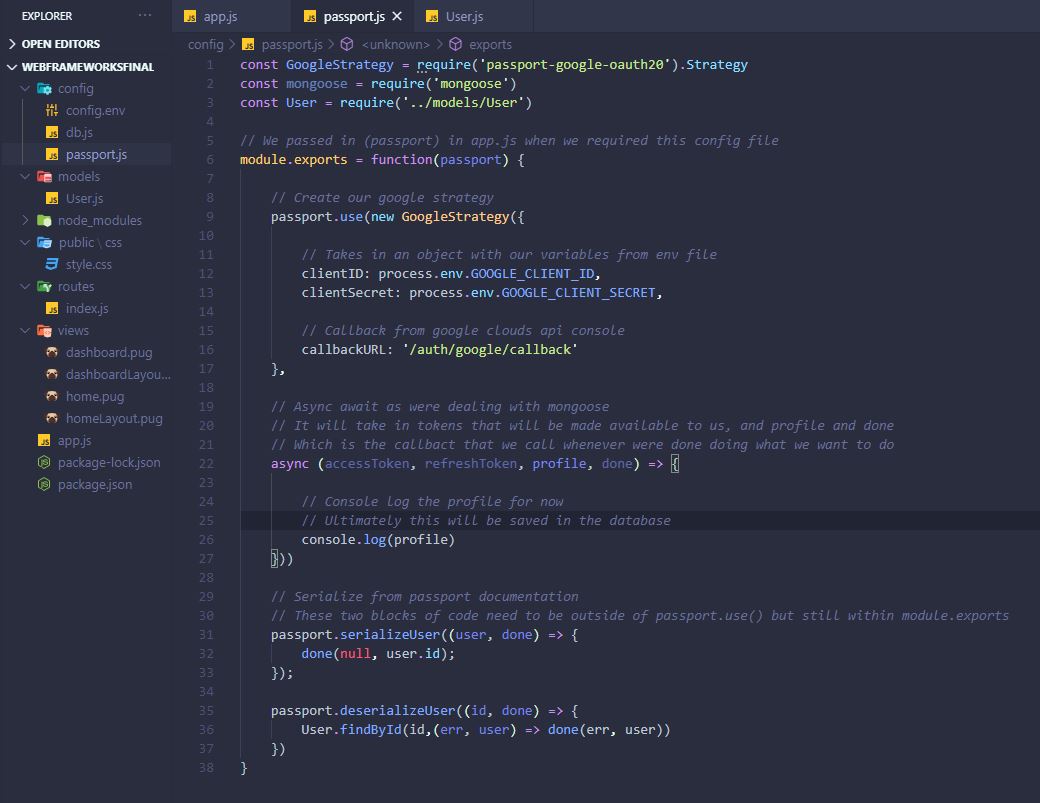
1. **PassportJS – I looked up the getting started documentation. I chose the passport-google-oauth20 strategy on the passportjs website for its documentation. Created a passport.js config file, then created a google strategy. (once user is logged in it will save that user in the database after I initialize the routes and app.js)**

****

**Imported this passport.js file in app.js. U also used passport sessions. Passport middleware one of them should be app.use(passport.session()) – forgot to do it before screenshotting. Cookies were removed from the app.use(session arguments as they will not work without https**

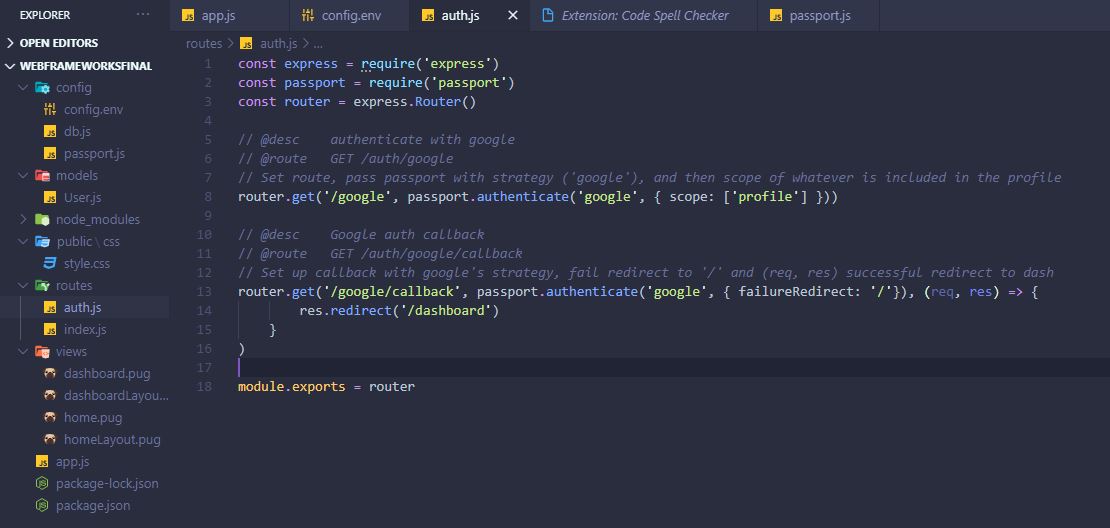
****

**I brought in the user model and set up passport google strategy**

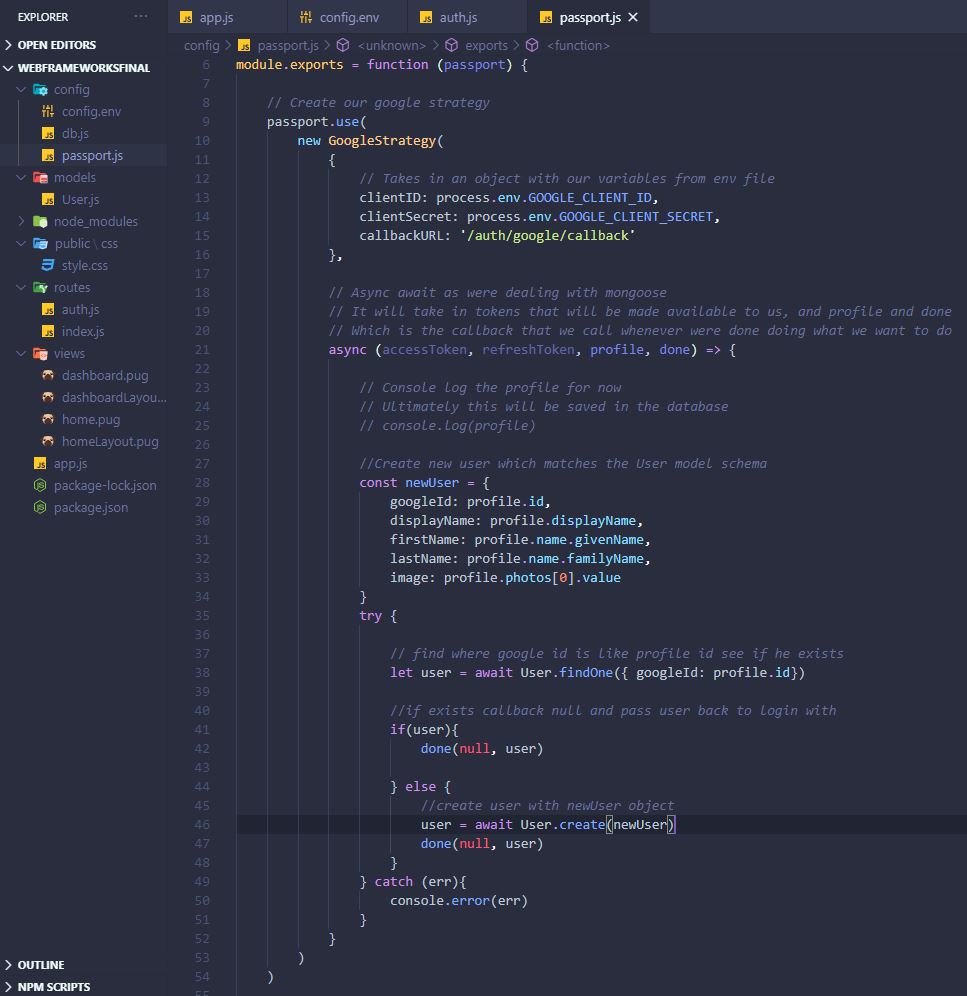
****

# **Routes**

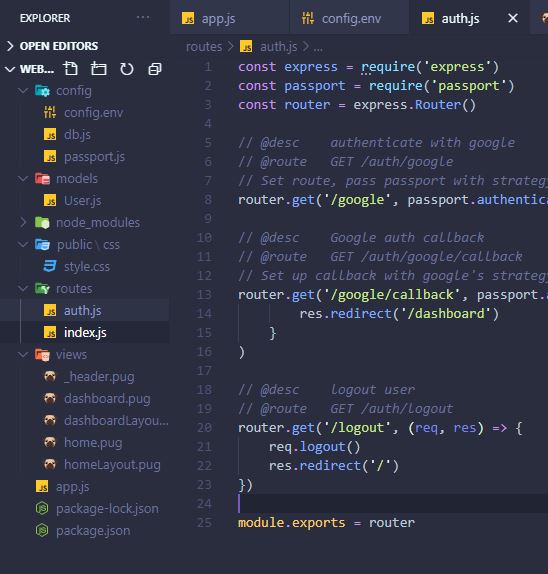
1. **I created an auth.js route file to deal with what happens after the google login button is pressed. Implemented them according to the passport google oauth 20 strategy documentation.**

****

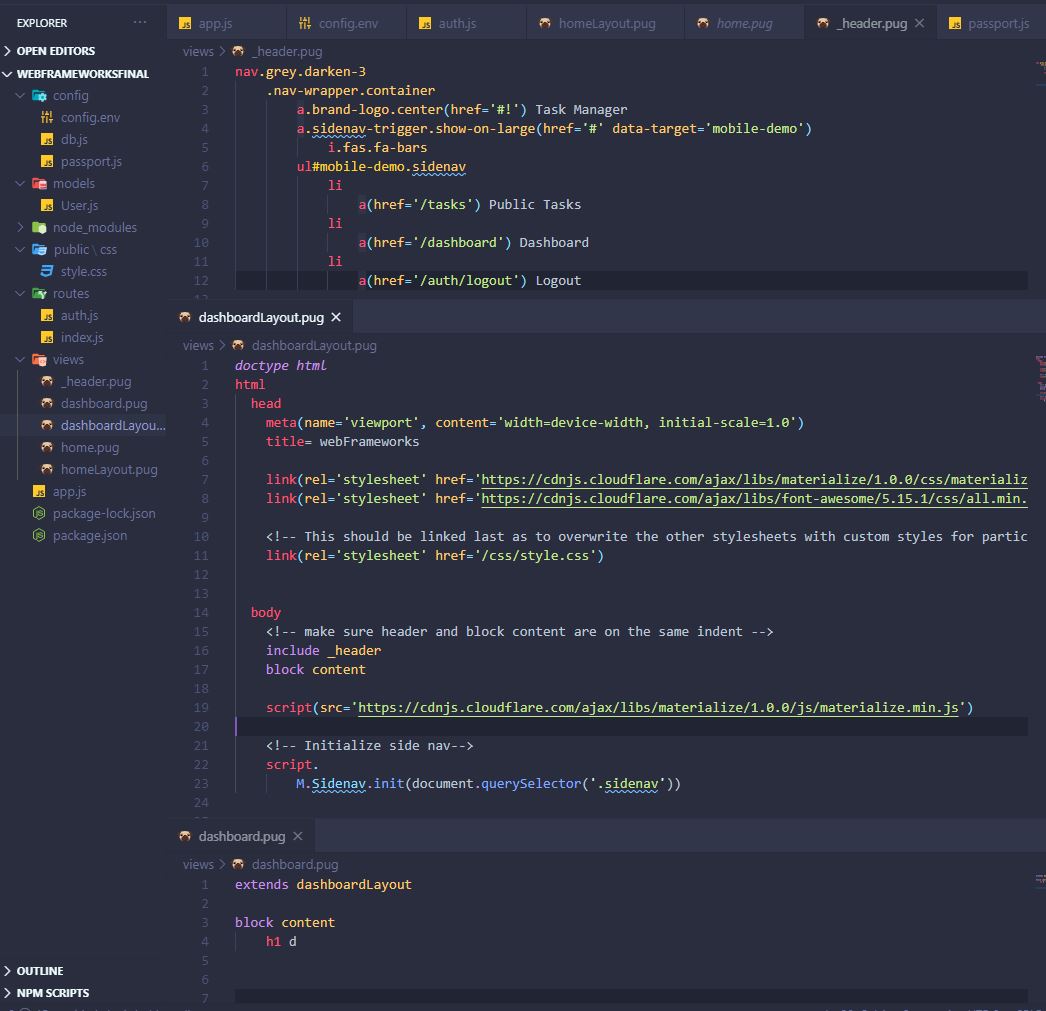
**Updated passport.js with Module exports a function for passport which uses a google strategy which makes a new user object from the google profile details. (Replaced the console.log(profile) with creating a new user)**

****

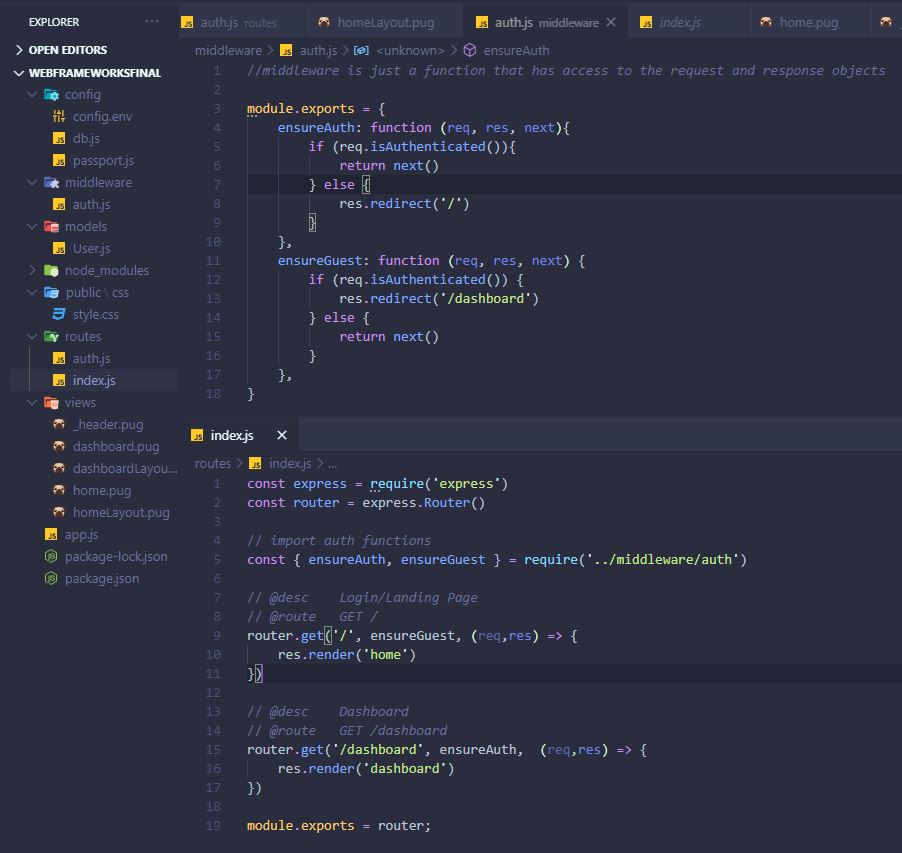
1. **Logout route to log user out and redirect to login screen/home**

****

1. **Created a partial pug file for the header and nav and included it in dashboardLayout, initialized the nav bar side button with a script**

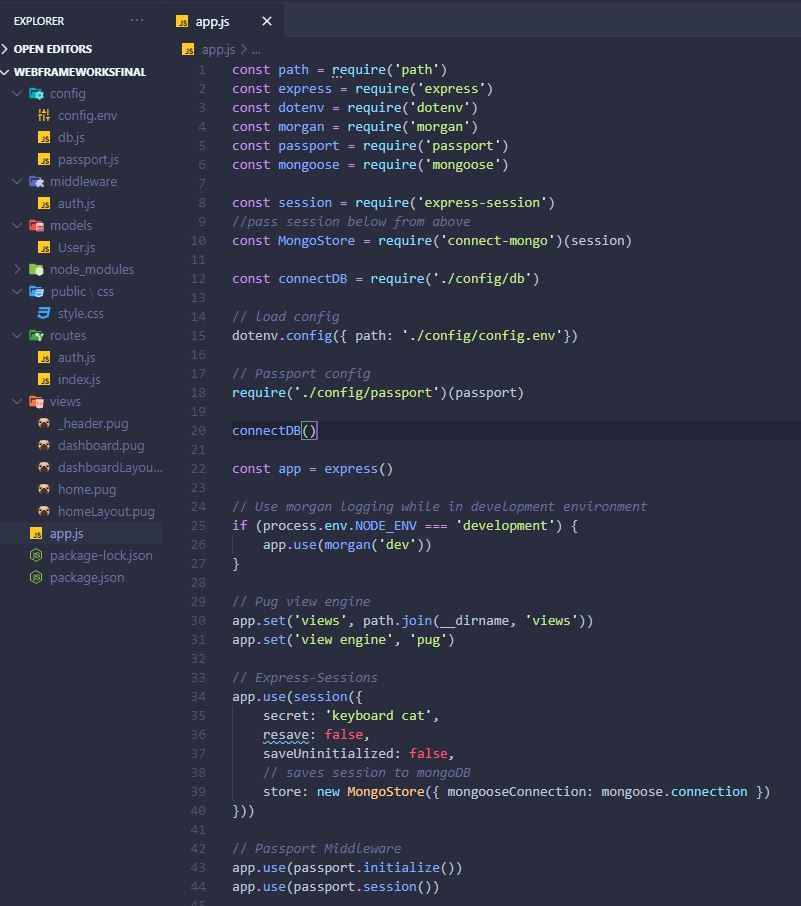
****

1. **I protected the dashboard route so unauthorised users can’t go to it by typing the link, same for the login screen users that are already logged in should not see the login screen. I created a middleware folder and an auth.js file in it. Added the functions in index.js.**

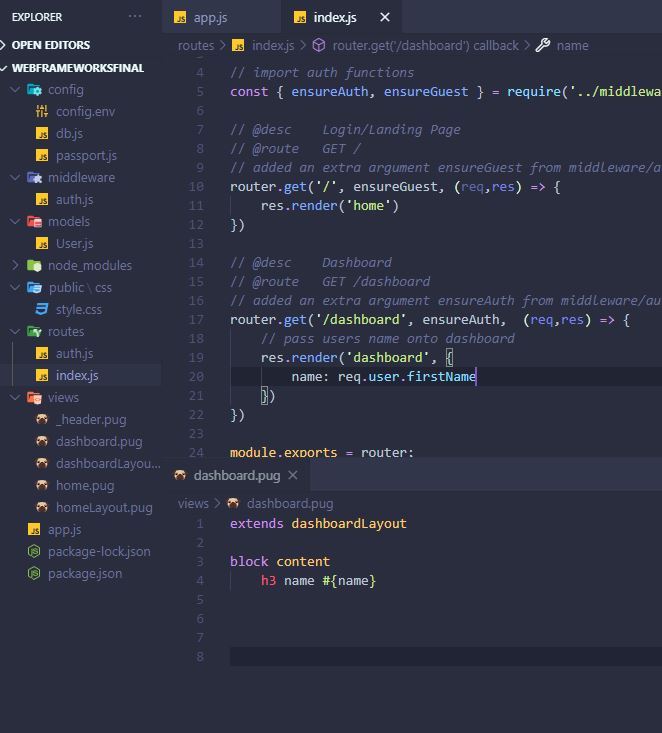
****

**Added Sessions into MongoDB so user stays logged in after page refreshes, or server restarts.**

**Used the connect-mongo package here. Updated the app.js under sessions with MongoStore**

****

**Updated index.js routes file so it displays the name of the logged in person on the dashboard**

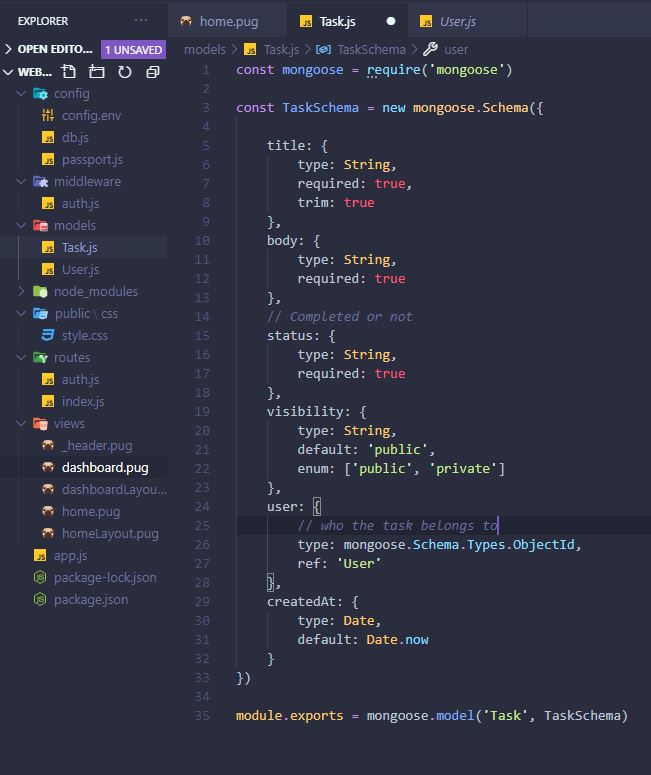
****

**Authentication was now fully completed for this project.**

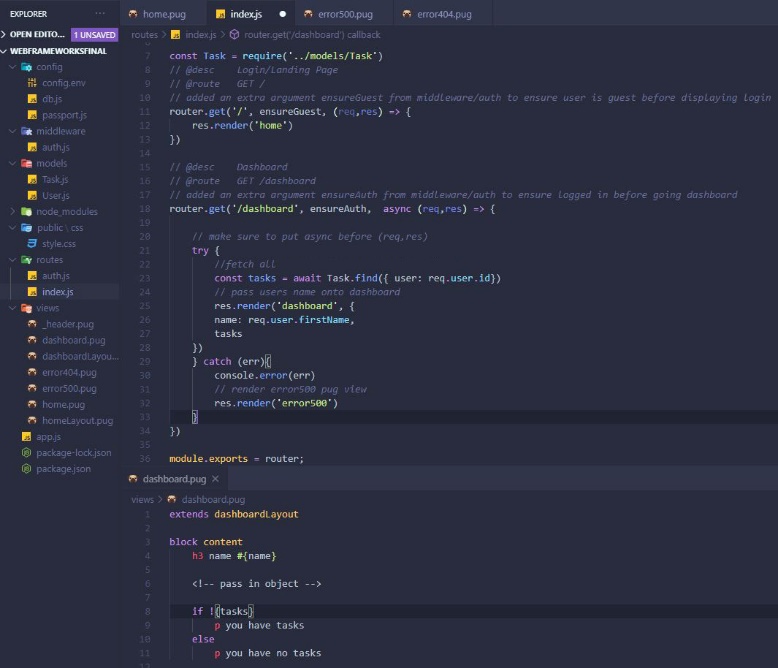
# **Tasks**

**Model, routes, controllers, views.**

1. **Task model with a user field which references the User schema**

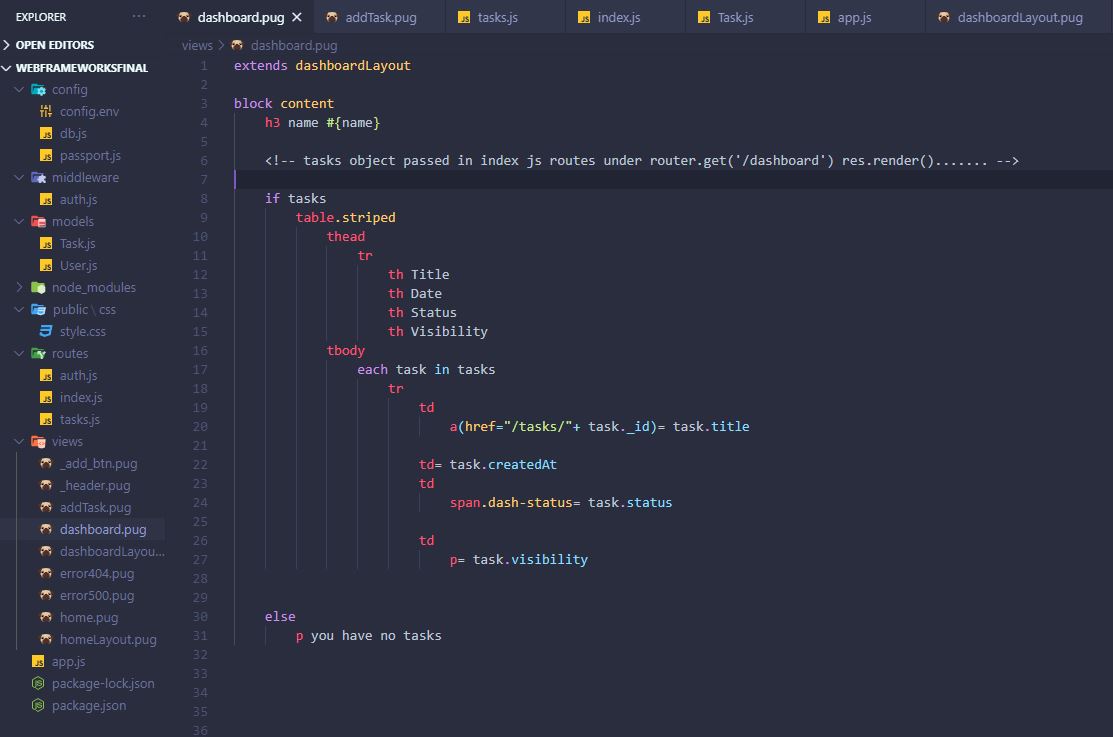
****

1. **Updated the index.js routes file, added tasks to be displayed on dashboard. Fetched all tasks into a variable and passed in into the render dashboard pug view. It can now be used there. (Zoom in)**

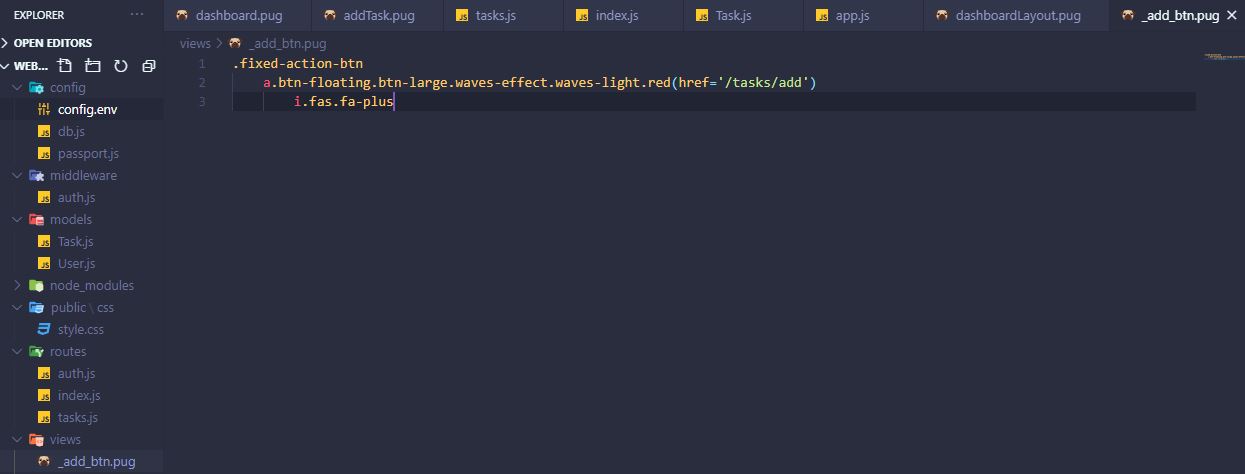
****

**If tasks exist show p you have tasks else p no tasks**

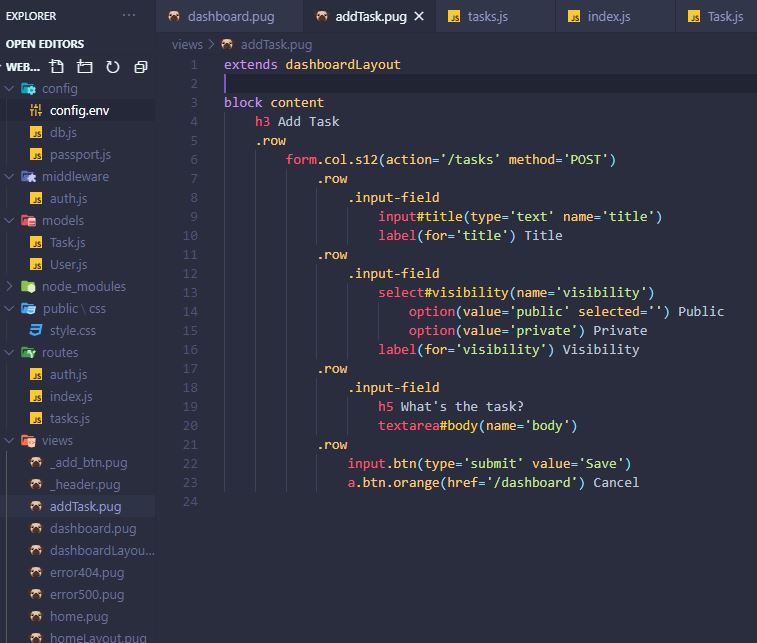
1. **I displayed a table of tasks from the logged in user.**

****

**Added an add\_button pug view, included in dash layout so its visible on any screen (can add a task on any screen you are)**

****

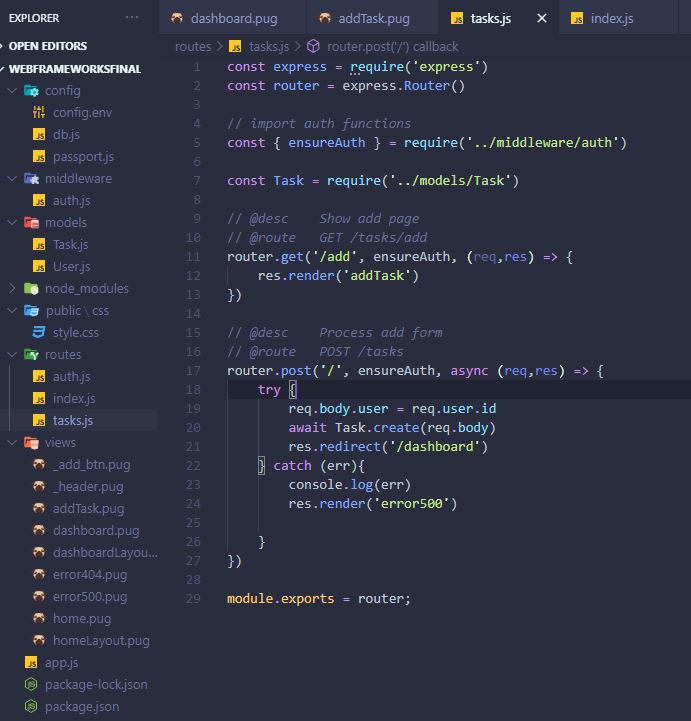
**Added an addTask page pug view**

****

**Created a tasks routing file and brought it into app.js. Added the /add route and addTask pug view in it, added an add task post request route to ‘/’ but this is actually ‘/tasks/’ as this is a tasks route file.**

**Creates a Task object from the addTask form href /tasks.**

**Req.body.user = req.user.id – this parses the logged in user into the req.body and creates the Task object from the req.body which now contains the values from the form and the user to create a task object (look at task schema)**

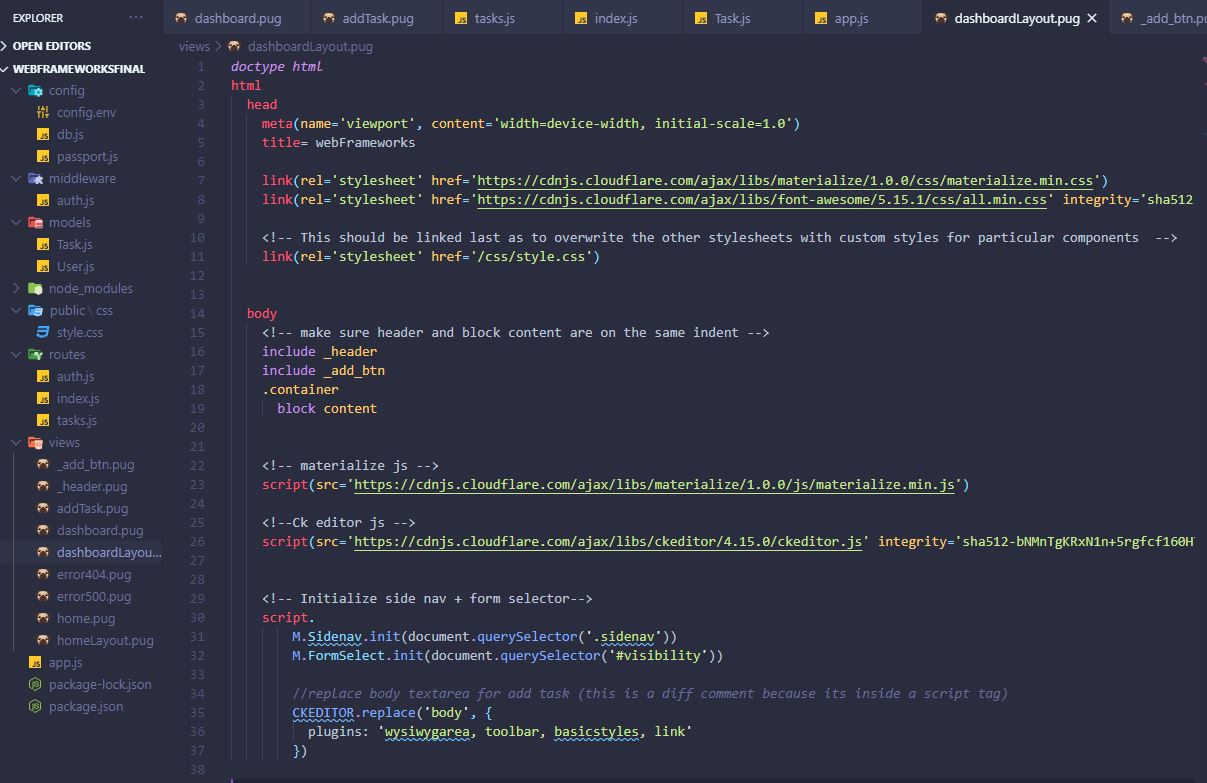
****

**Brought it into app.js**

****

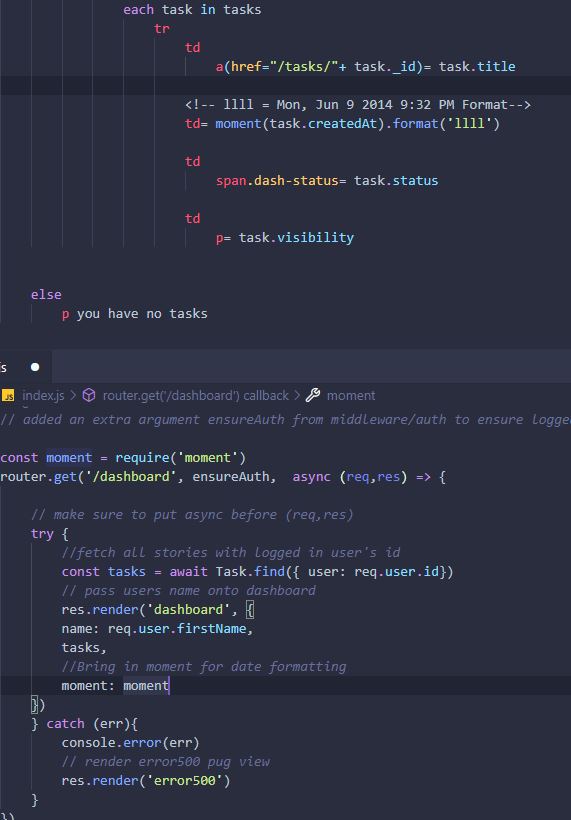
**Initialized the form public/private selector & CKEditor text area**

**Got the CDN from cdnjs.com, got the scripts from online documentation**

****

1. **Formatting the date on the tasks**

**I used the moment package for this, in index.js /dashboard route I brought in moment as moment (can call the first moment variable anything you want to use inside the pug view) just like I brought in tasks but tasks were an object so just tasks. Name was just the logged in user’s first name (there is also an error 500 pug view page I created that just extends the dash layout and has 1 p tag saying server error to display for error handling**

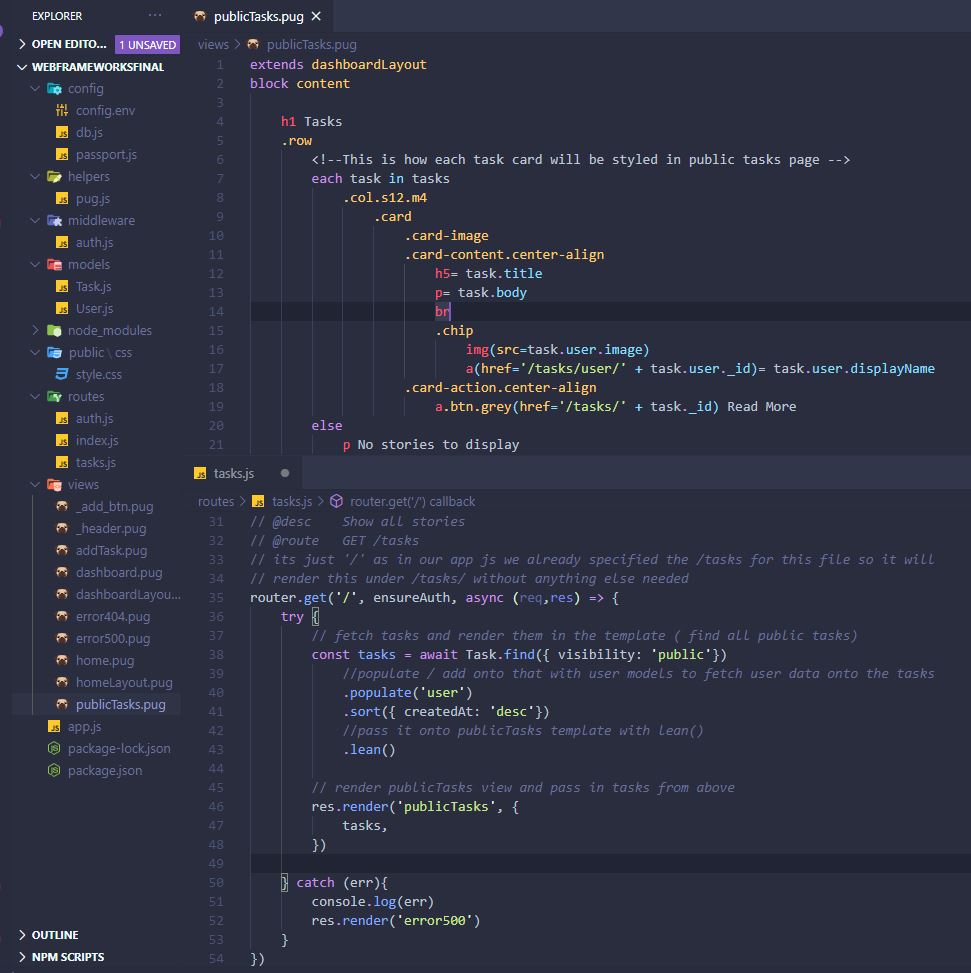
****

# **Public Tasks**

1. **I created a publicTasks pug view page and added to tasks route file.**

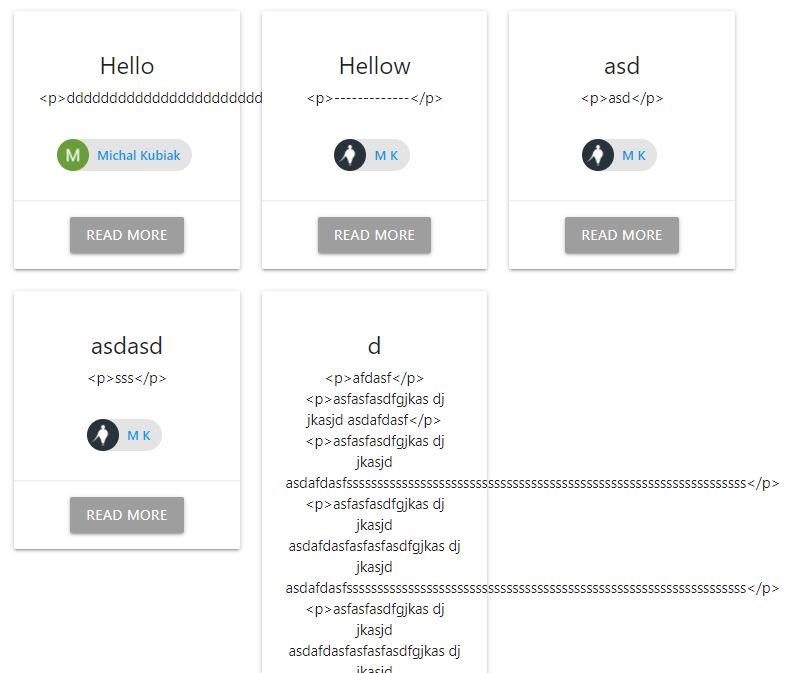
**The pug view displays each task in database as a card with links to specific user’s tasks page (not yet done) and a task page to read more about it (not done yet)**

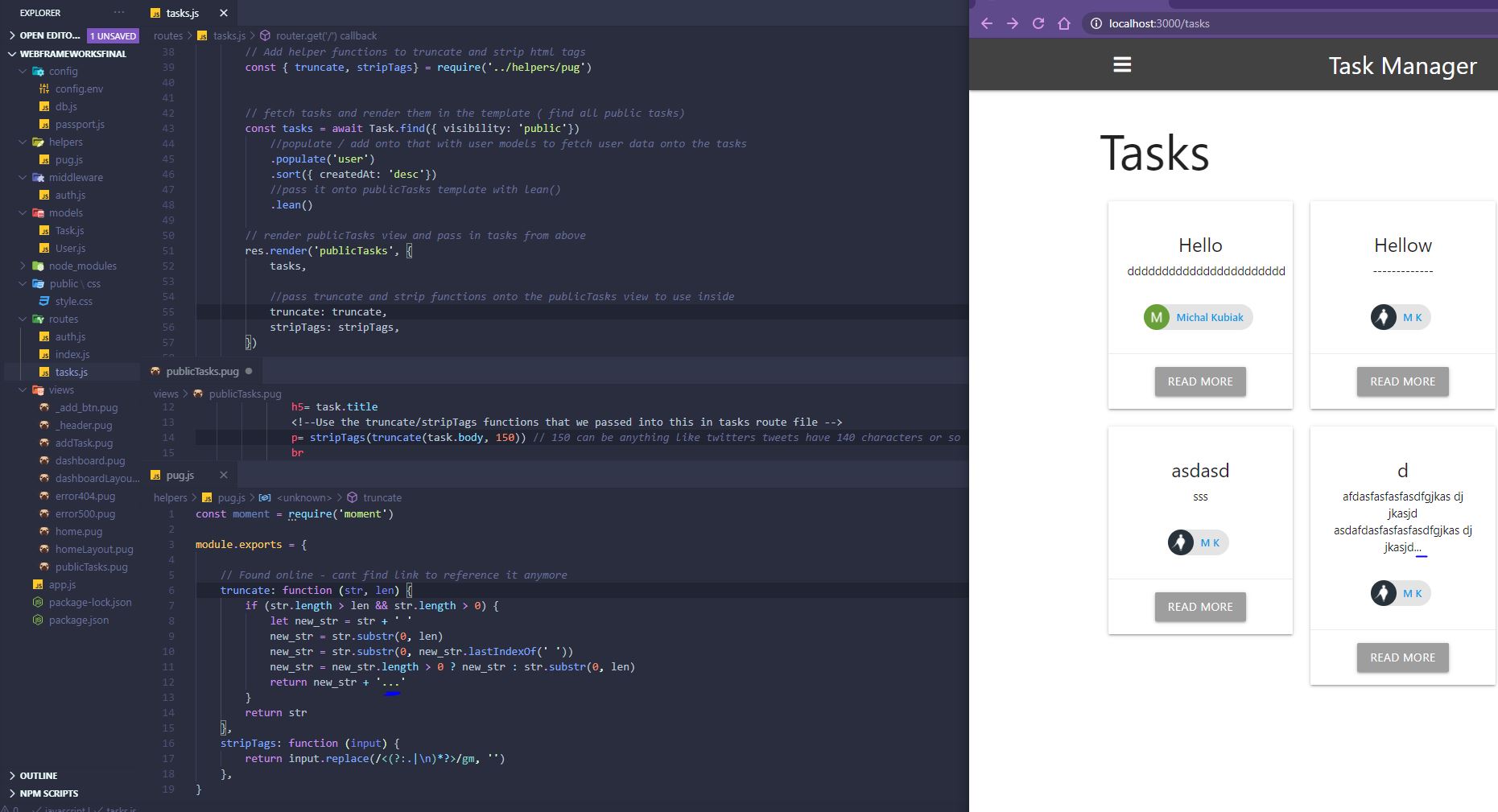
**The route was updated so that the home page in tasks ‘/’ (which is just localhost:300/tasks/) now displays the tasks that are PUBLIC only as I passed that argument into the Task.find mongoose function. I populated the tasks with the user model data, sorted the tasks in ascending order, and leaned them.**

****

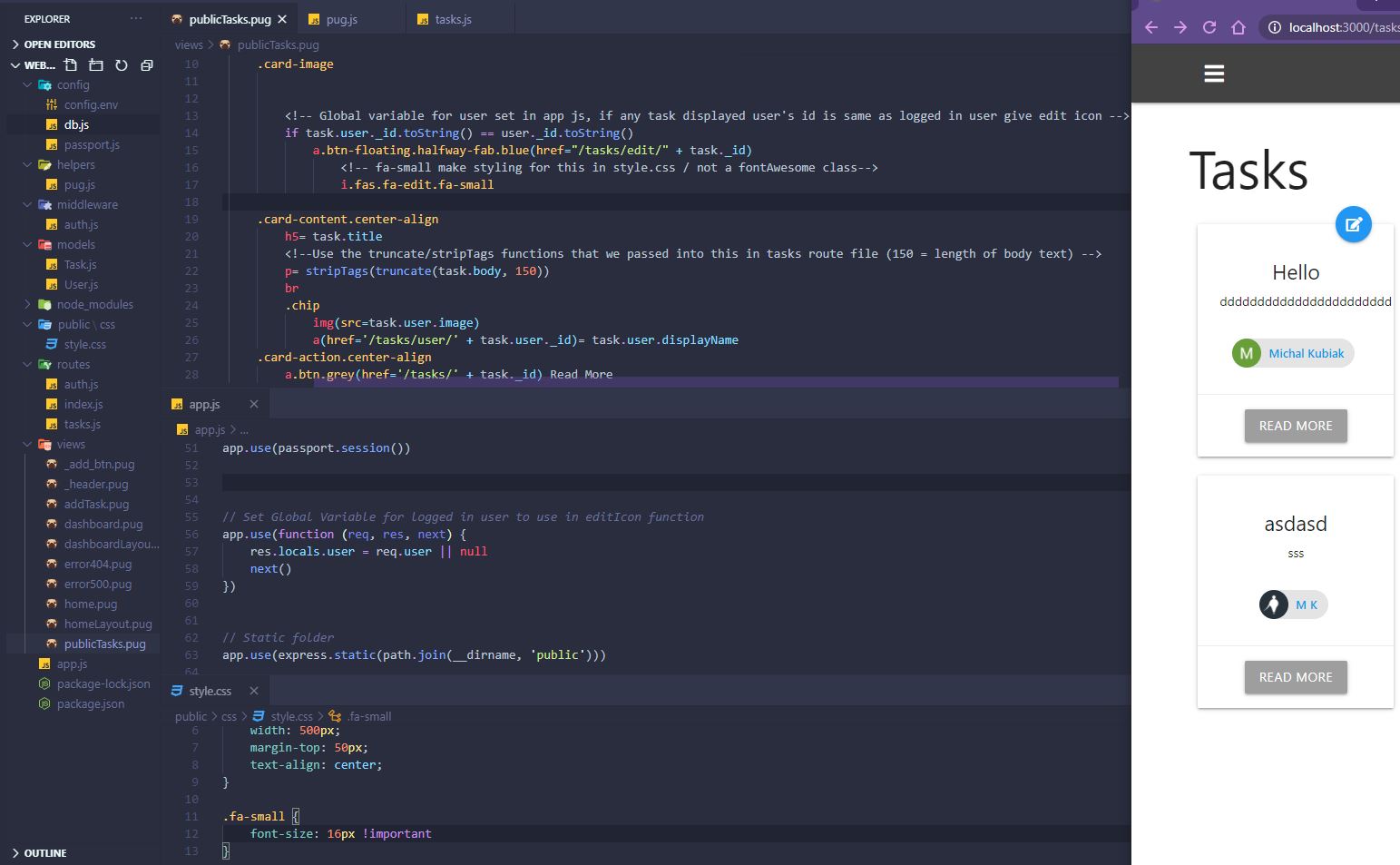
1. **The text on the tasks now displayed the html tags and the body could be very long so I found a truncate and remove tags functions online. I wrote them in a pug helper file in the helper’s folder. I passed them into the dashboard render to use in the pug view.**

**Before and after:**

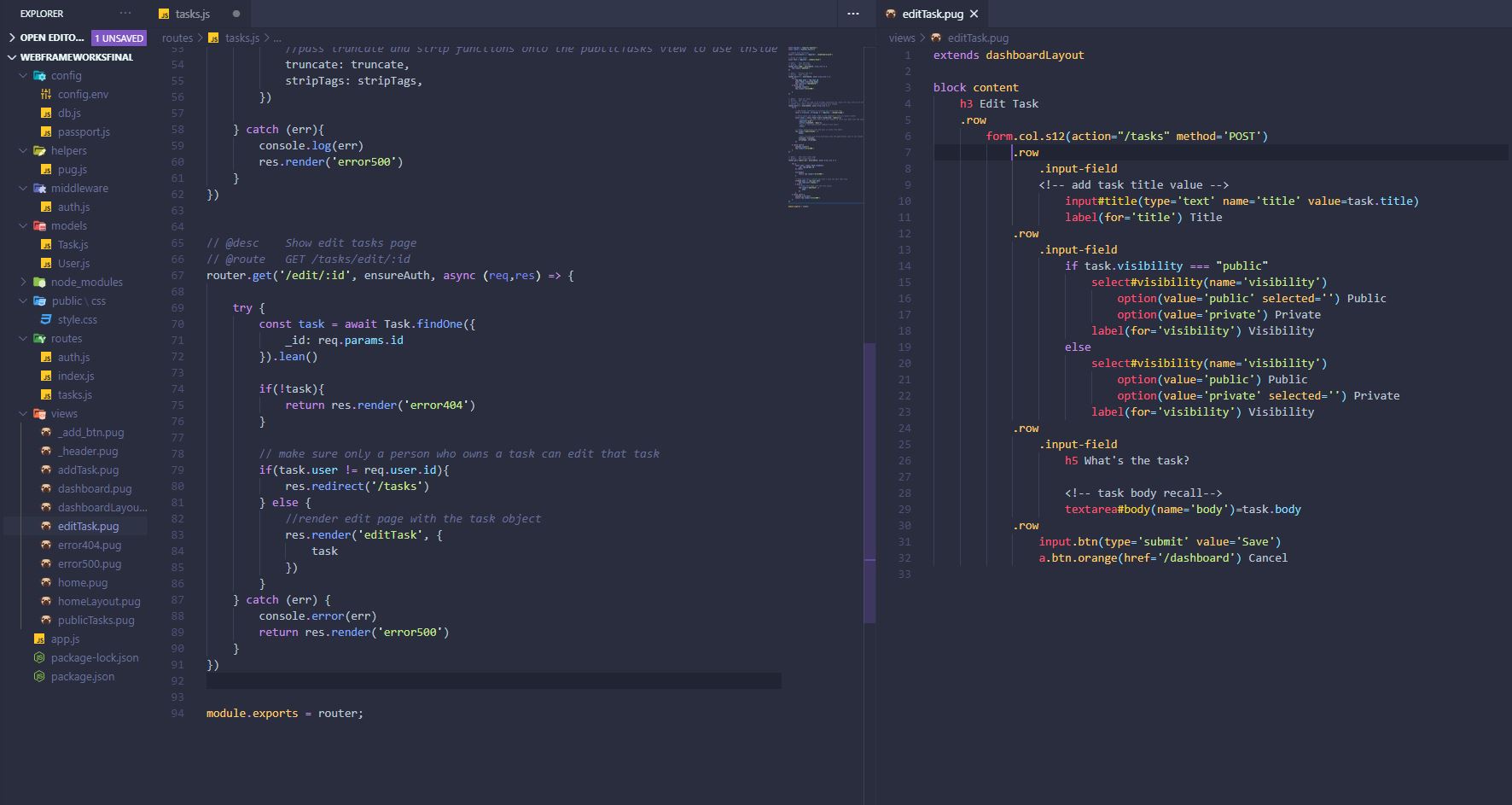
****

****

1. **I added an edit task button and passed it into the dashboard pug view. I used a global variable to use the current user in the pug view. If the logged in user has the same id as the task’s user’s id then they can edit the task.**

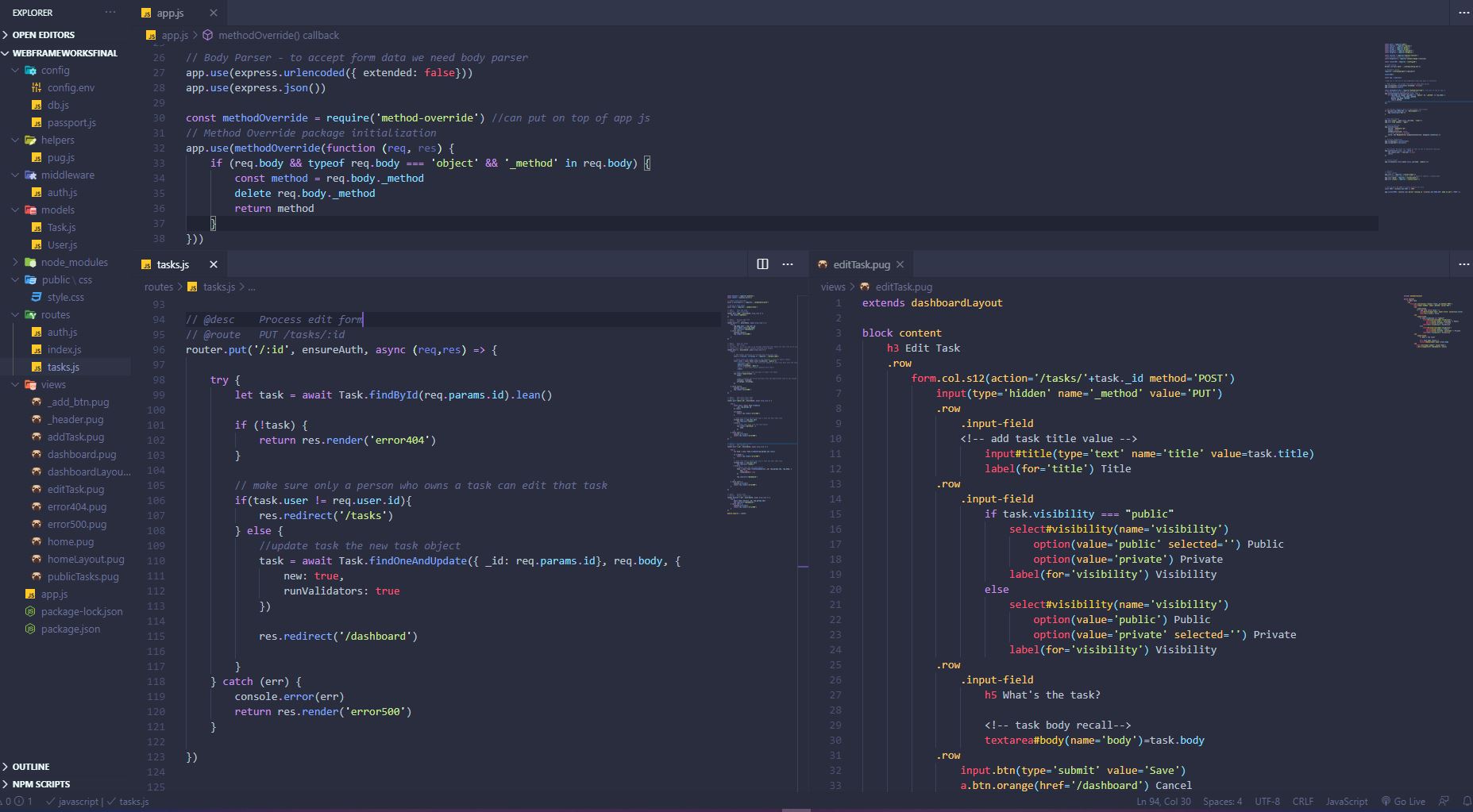
****

1. **I made an editTask pug view page for after clicking the edit button and initialized the route for this. findOne function with passed in id of the task that was clicked. Load the saved data from the task I want to edit into the form using value=**

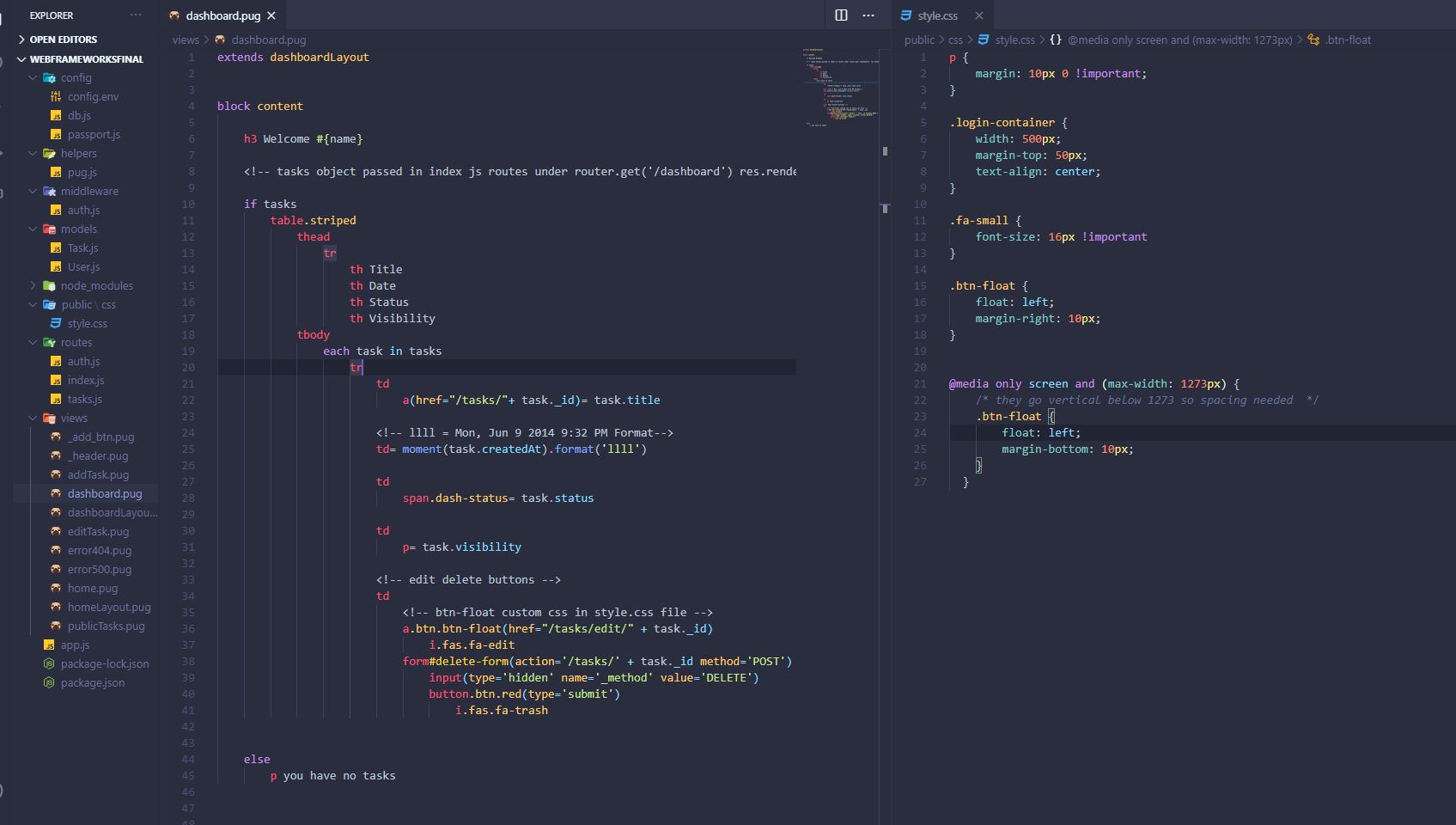
****

**For the form to work I needed to override the POST method into a PUT method using the method-override package. A form can only do GET POST by default, can’t put or delete. This is done below by making an input that’s hidden that has a value of put and a name of \_method.**

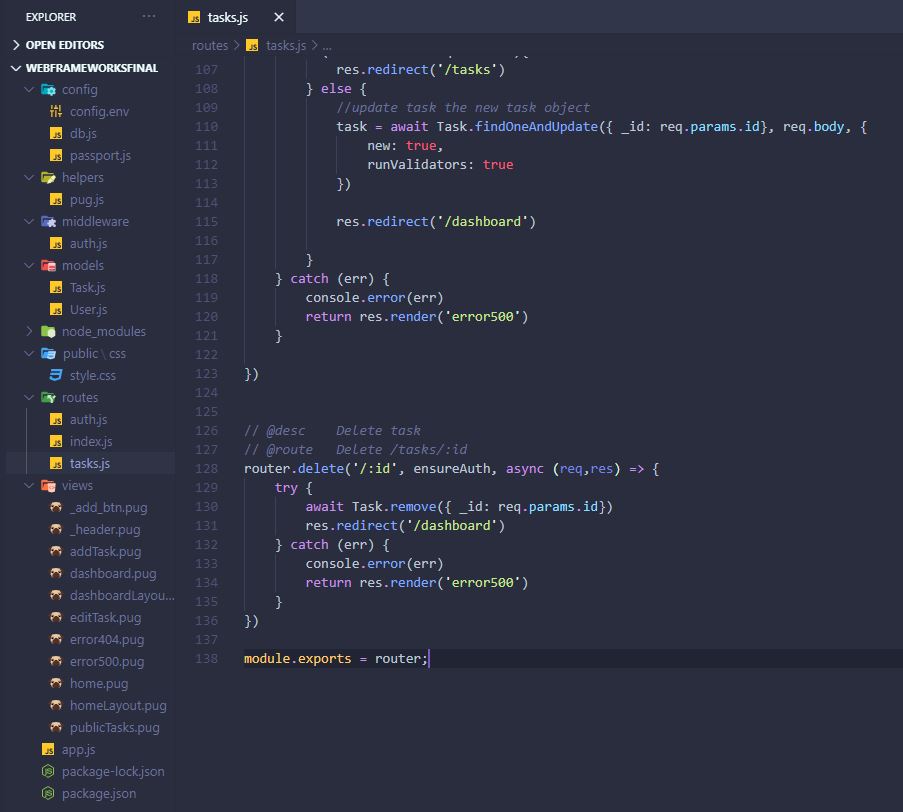
**I initialized this in app.js for use and made a put route when the user updates a tasks they will have the id in the http bar so the :id uses that to PUT any task we clicked and update it in the database with findOneAndUpdate function.**

****

1. **Added edit and delete buttons to dashboard**

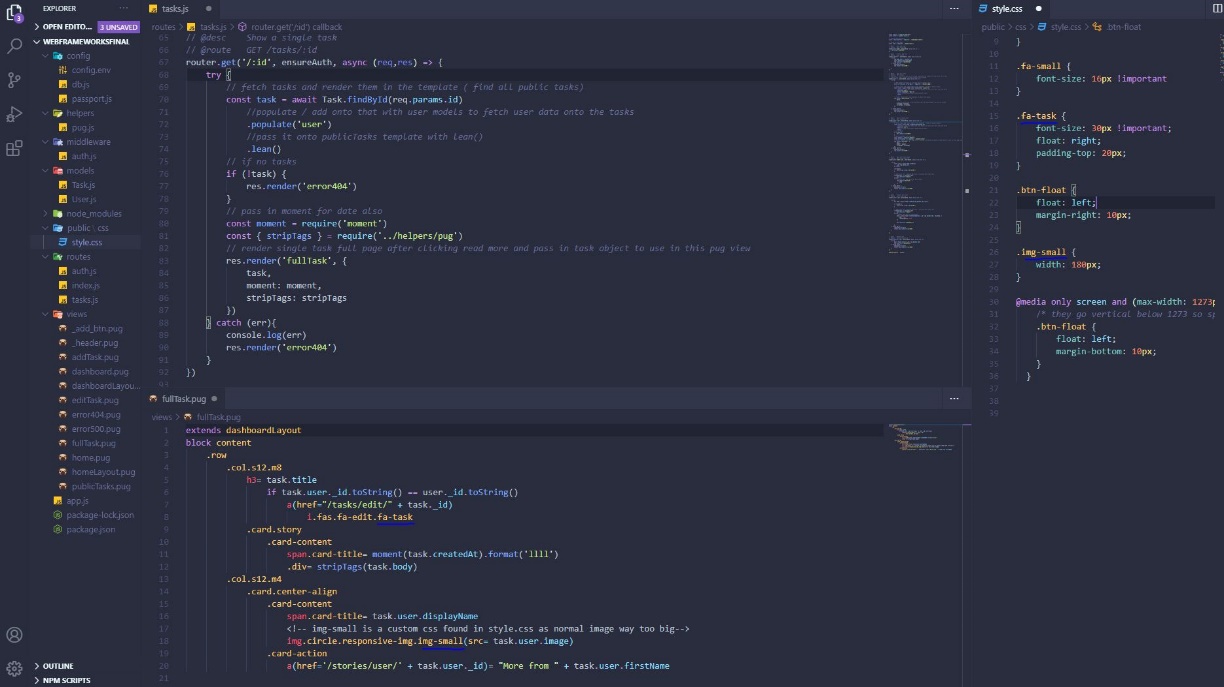
****

1. **Added delete route to tasks route file (the view has a delete button link that links to the delete route)**

****

# **Full Task Page (Read More Button on a Task)**

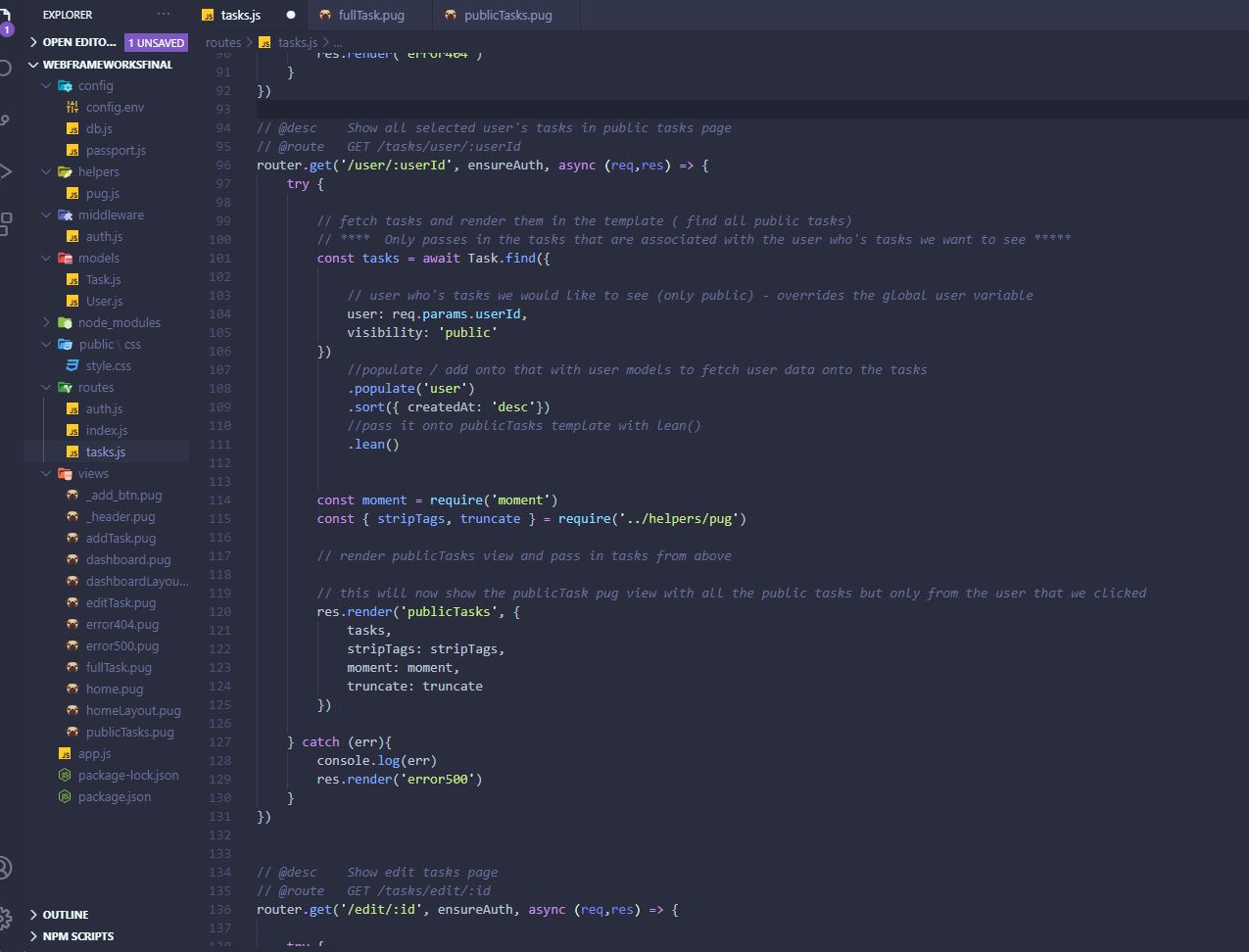
1. Router get a particular task id and display the fullTask pug file



# **Particular User Tasks Sort**

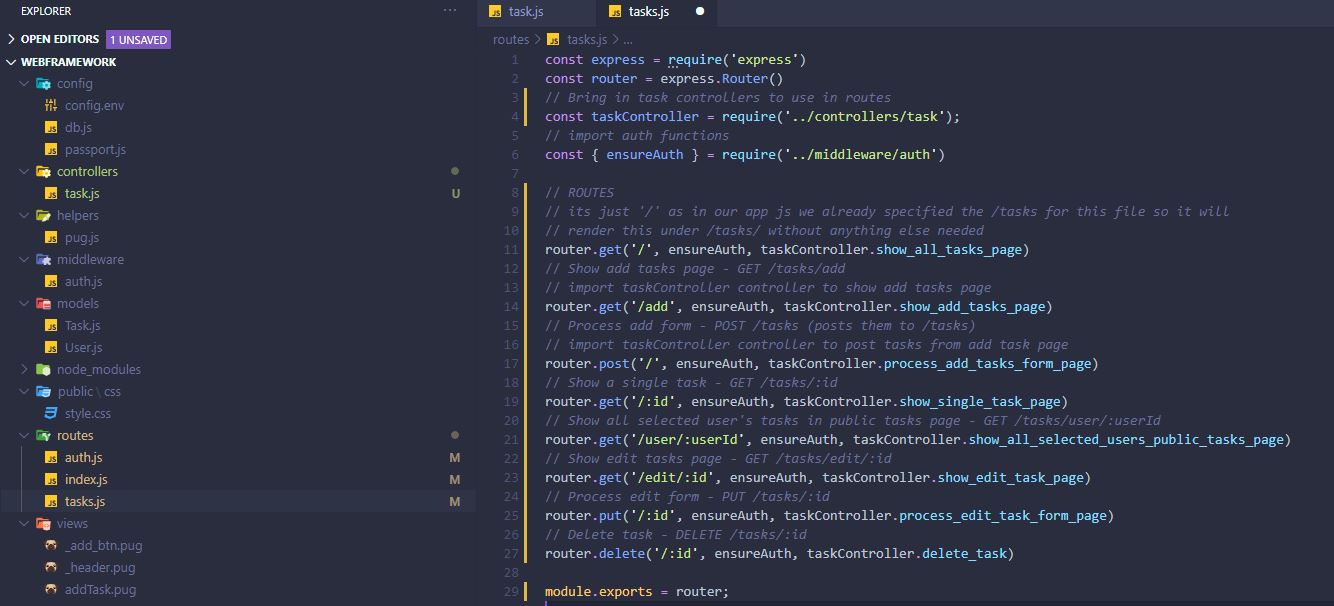
1. User Tasks - /tasks/user route and view. New route in tasks user/:userId.

This will show all the tasks from any user we clicked on the /tasks/ page as we only put through the tasks with specified ID, that’s why we can re-use the publicTasks pug view. When we click a task’s user there is a link there that links to this route.

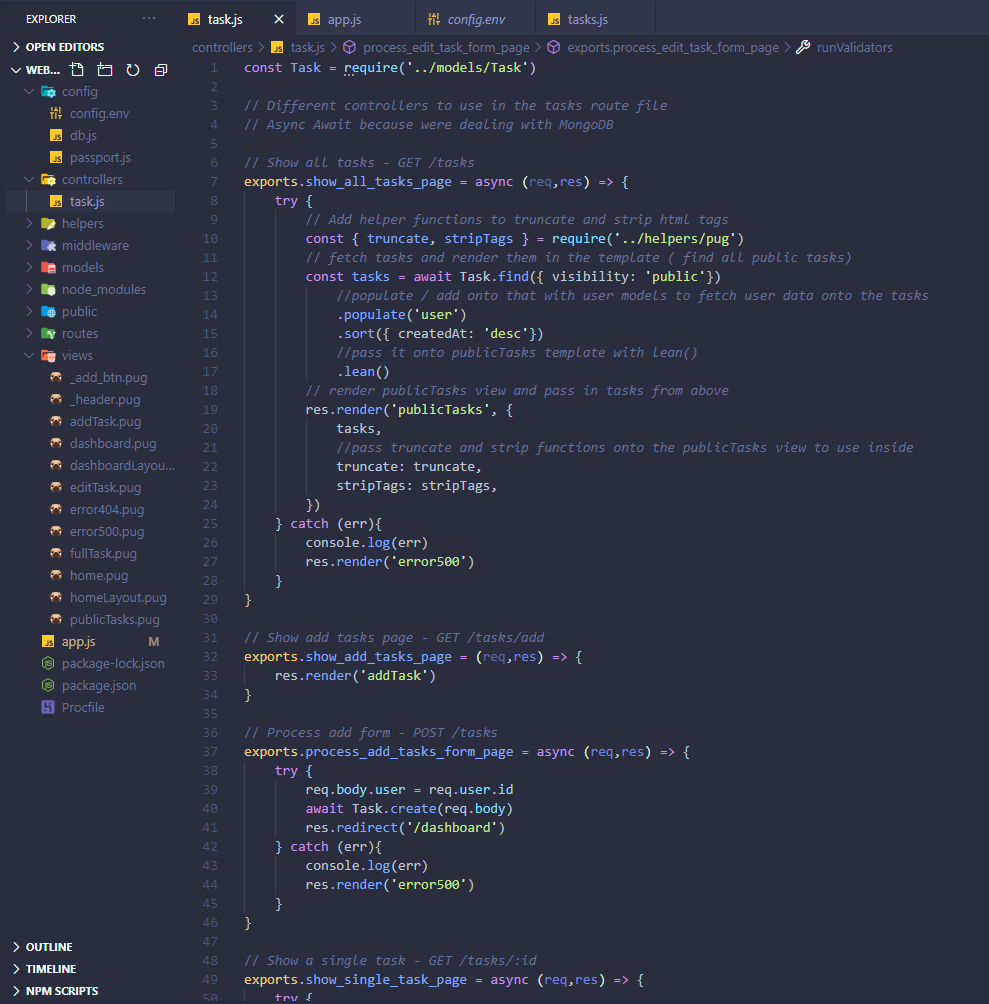


# **Controllers**

1. I cleaned up the tasks route file and made a controller file for all the functions passed into the routes. This is the tasks route file after clean-up

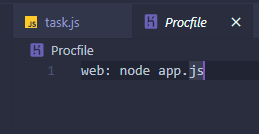


Ill attach a snippet of the top of the controller. I did cut everything after the ensureAuth and pasted them into controller file as functions and exported them and imported them into the tasks route file.

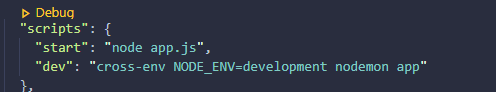


# **Heroku Deployment**

Made a new file called Procfile in the root directory.



Changed the package json start to



Registered on Heroku

Create a new app

Downloaded the installer and installed it on the machine.

Created remote repository with cmd line from my project folder

git push –set-upstream https://github.com/michalkubiak98/projectName.git

cmd line again

heroku login

(At this point I was missing the morgan package so I npm installed it, heroku wouldn’t push)

git add .

git commit -m “commit”

git push heroku master

App was now deployed and got a link in the cmd line for it.

I had to now change the Google Console’s URI and add Heroku’s app link instead the localhost:300/auth/google/callback

And added auth/google/callback to it

I had to create a new credential with it and copied the new client id and secret into the config file in project and repeat git add and push to heroku.