MVC

Uses and importance in Web Development

What is MVC?

No it’s not Marvel vs. Capcom

MODEL

VIEW

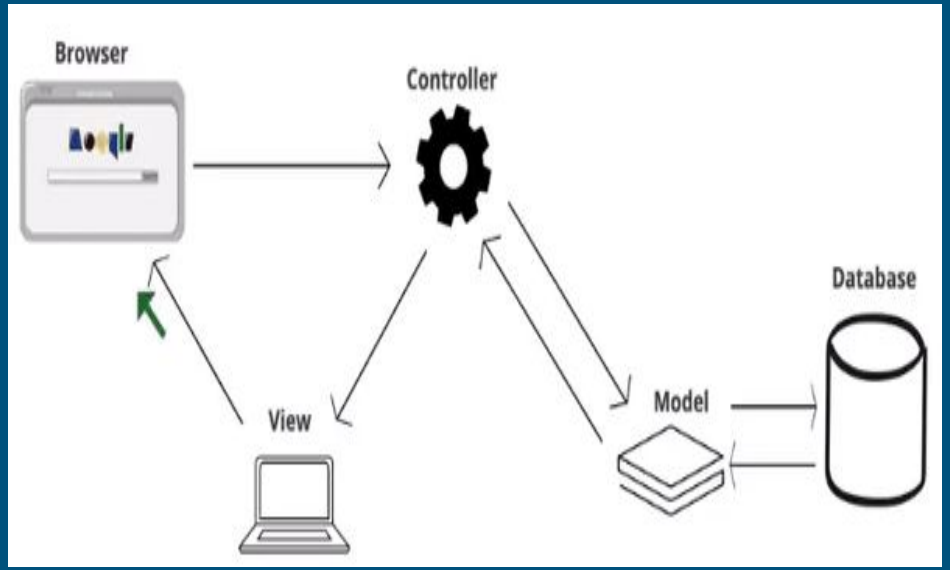
CONTROLLER

MVC is a design pattern that separates an applications functionality into 3 parts:

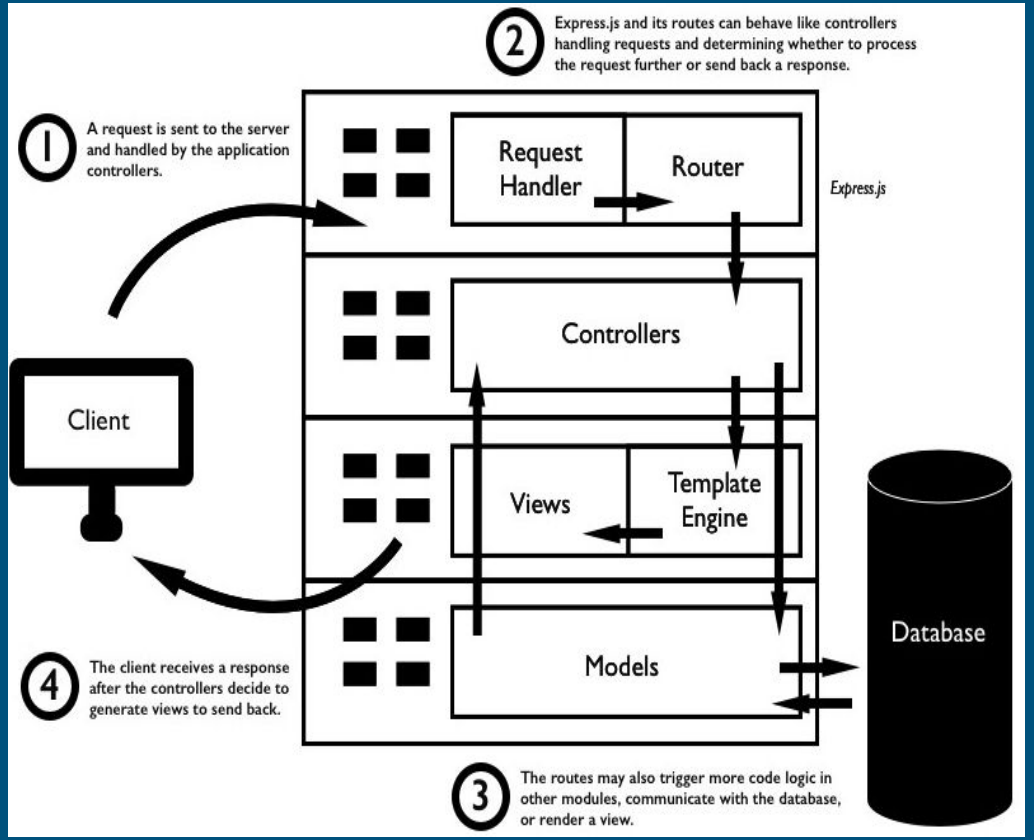
-Model is the data (database) that communicates with the controller

-View is the representation of the data to user (client side specific) that communicates with the controller

-Controller process requests and gets data from Model to the View (server)



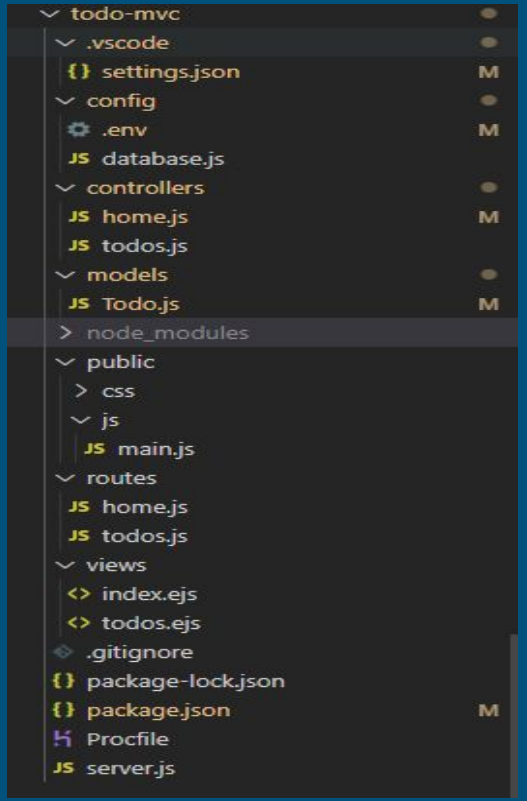
MVC with routing included in node.js



So…what does all that mean?

Let’s break it down step by step

Let’s check out this todo-mvc folder in VSCode and specifically look at the models/views/controllers/routes folders



First: Models

The model sends info to the controller based on requests that controller received from the user. Model interacts with the database and handles updating/deleting/saving data. It doesn’t have to concern itself with user requests since that is the controllers job.

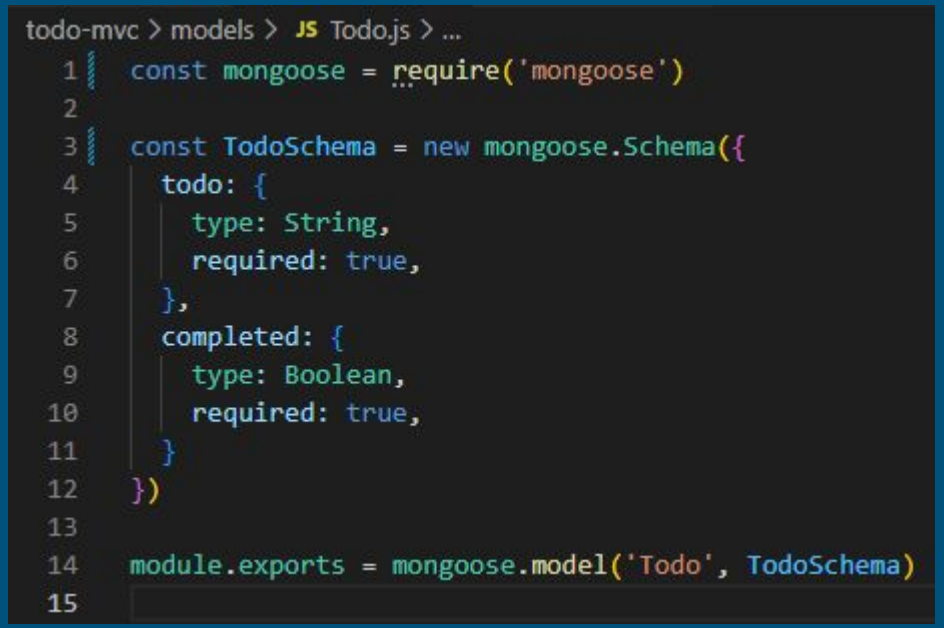
A look at the flow

In the todos.js in controllers folder, a variable Todo is set up to use the Todo file in the models folder

todo-mvc > controllers > JS todos.js > ...

```
1   const Todo = require('../models/Todo')
2
3   module.exports = {
4       getTodos: async (req,res)=>{
5           try{
6               const todoItems = await Todo.find()
7               const itemsLeft = await Todo.countDocuments({completed: false})
8               res.render('todos.ejs', {todos: todoItems, left: itemsLeft})
9           }catch(err){
10              console.log(err)
11          }
12      },
13      createTodo: async (req, res)=>{
14          try{
15              await Todo.create({todo: req.body.todoItem, completed: false})
16              console.log('Todo has been added!')
17              res.redirect('/todos')
18          }catch(err){
19              console.log(err)
20          }
21      },
22      markComplete: async (req, res)=>{
23          try{
24              await Todo.findOneAndUpdate({_id:req.body.todoIdFromJSFile},{
25                  completed: true
26              })
27              console.log('Marked Complete')
28              res.json('Marked Complete')
29          }catch(err){
30              console.log(err)
```

In the Todo.js file in models, mongoose is used to set up information for the database



WAIT WAIT WAIT

...WHAT THE HECK IS MONGOOSE?????

We’re not talking about these little cuties

Or these BADASSES

…I just wanted an excuse to post this epic pic <_<

Mongoose is described on its site as “elegant mongodb object modeling for node.js”.

In layman terms, it makes working with MongoDB easier. You don’t have to spend so much time writing MongoDB validations/casting and structures can be set with schema

Oh come on…what’s a SCHEMA?

(sounds like if someone from Boston decided to make a rap name to me…)

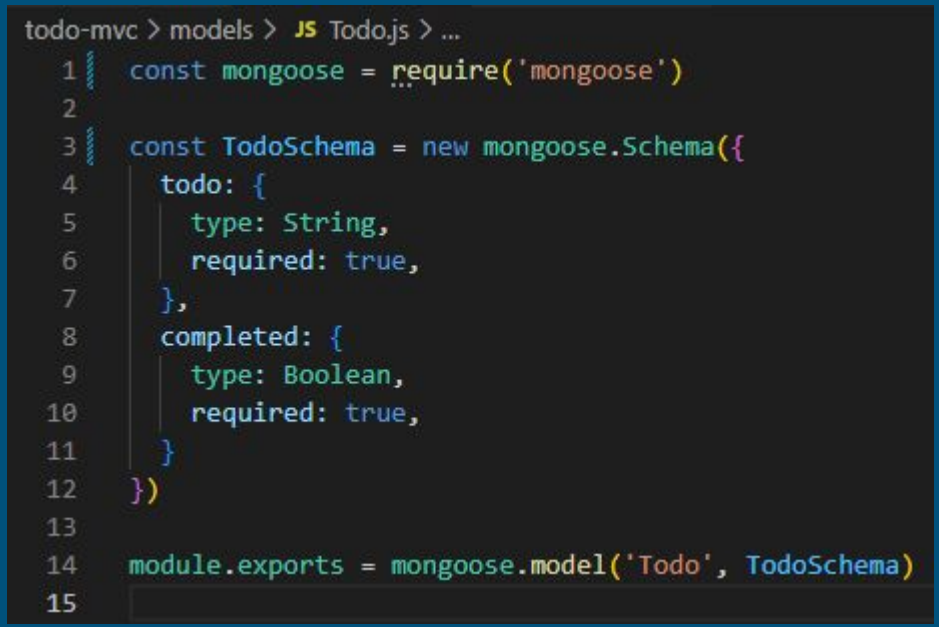
MongoDB is great for storing documents with dynamic structure, but with a database that is large in scale it can get…a little crazy. A little out of control. A little “what is this and why is this in my database”

Schema validation sets specific structures for documents and if a document appears with a different structure the database will reject it.

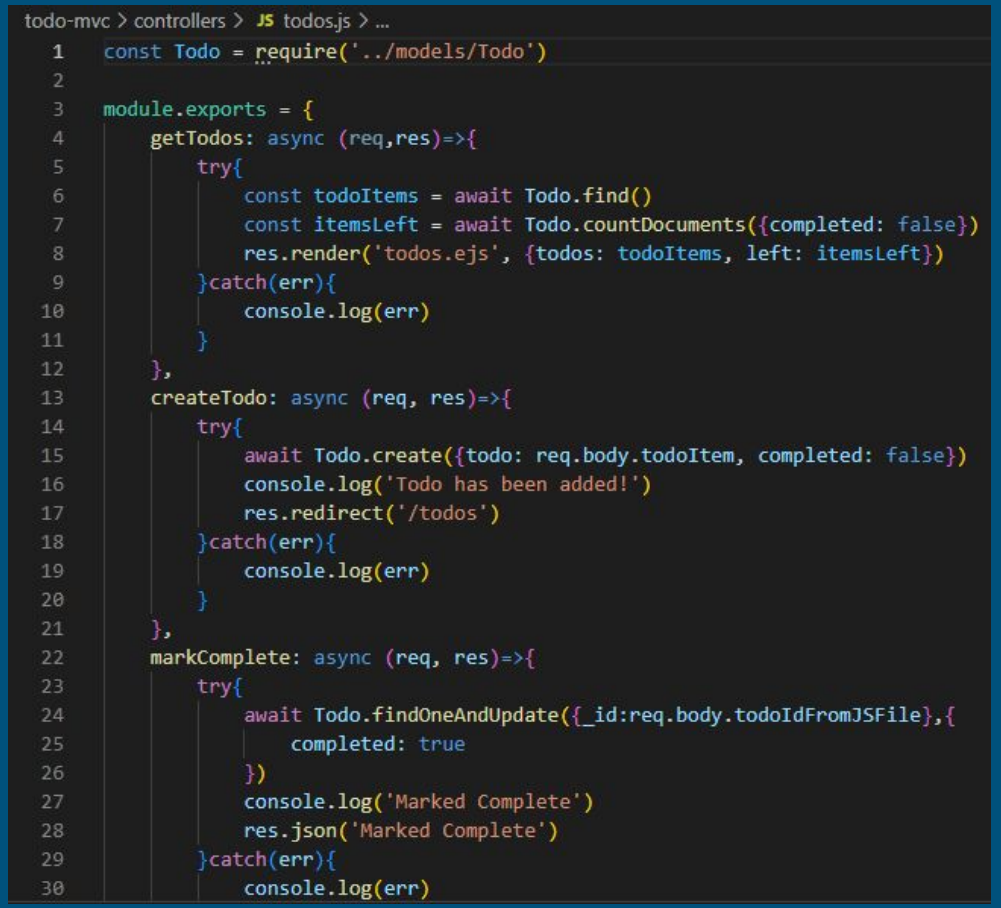
Long story short, with the help of schema validation Mongoose makes managing relationships between data in MongoDB easier and quicker

Back to the main topic

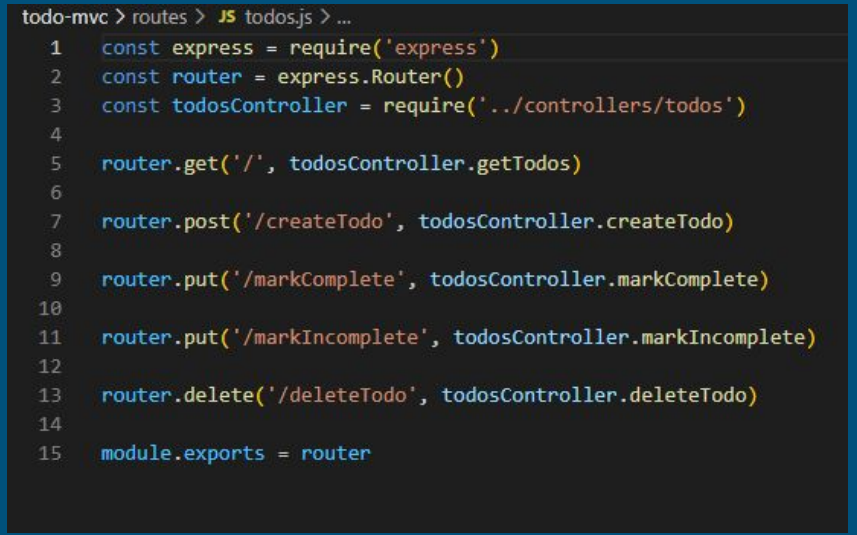
mongoose.model is a function that creates the model for the schema. First parameter takes in name of model (‘Todo’: this is the name that will be in database) and second parameter is TodoSchema passed in. module.exports will allow it to be used in other places



The Todo model set up in the models folder will be used here and the actions used (getTodos/createTodo/markComplete) will depend on request from the router.



Sends request to the todos in the controller using CRUD



And files in views renders information from the controller

todo-mvc > views > todos.ejs > html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>Todos</h1>
    <ul>
    <% todos.forEach( el => { %>
        <li class='todoItem' data-id='<%=el._id%>'>
            <span class='<%= el.completed === true ? 'completed' : 'not'%>'><%= el.todo %></span>
            <span class='del'> Delete </span>
        </li>
    <% }) %>
    </ul>

    <h2>Things left to do: <%= left %></h2>

    <form action="/todos/createTodo" method='POST'>
        <input type="text" placeholder="Enter Todo Item" name='todoItem'>
        <input type="submit">
    </form>

    <script src="js/main.js"></script>
</body>
</html>
```

To be more precise, let’s say a user goes to todos section of webpage (‘/todos’) and creates a new item

```
app.use('/todos', todoRoutes)
```

The server hears this and sends a request to the Router

```
const todosController = require('../controllers/todos')
```

```
router.post('/createTodo', todosController.createTodo)
```

Since user wants to create a new item, router sends request to controller

Controller hears the request, uses todo model that was established and refreshes todo webpage once information has been successfully inputted

todo-mvc > models > JS Todo.js > ...

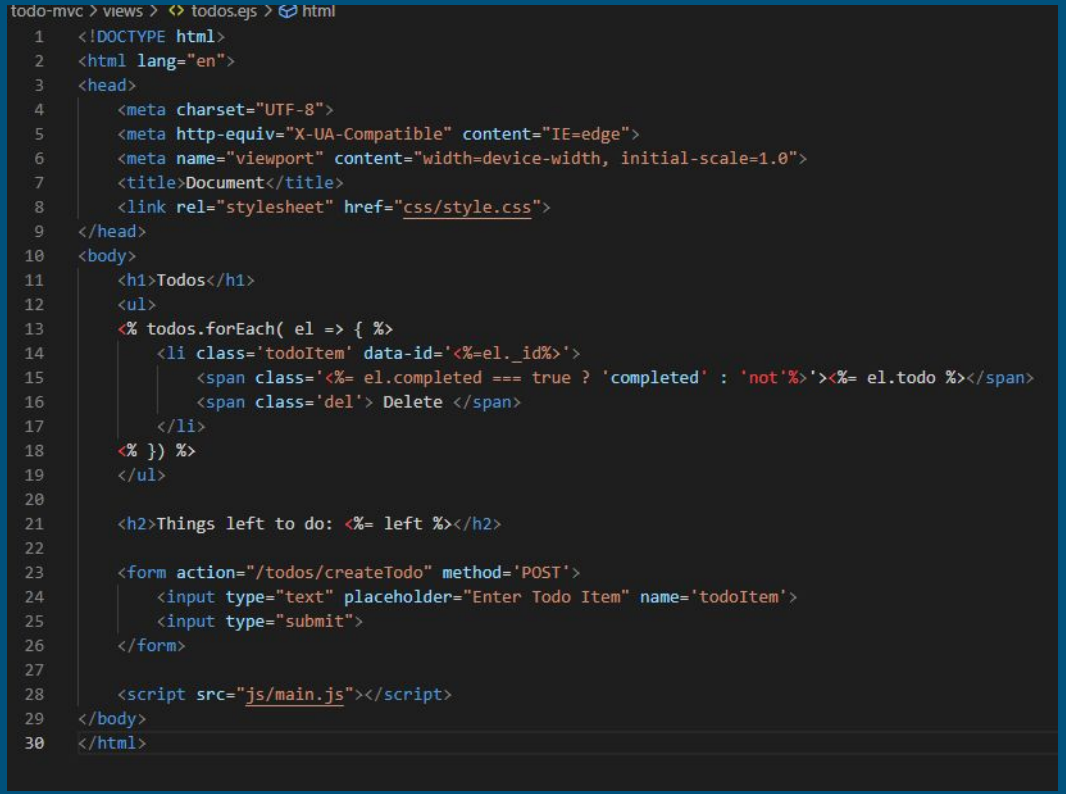
```
const mongoose = require('mongoose')

const TodoSchema = new mongoose.Schema({
  todo: {
    type: String,
    required: true,
  },
  completed: {
    type: Boolean,
    required: true,
  }
})

module.exports = mongoose.model('Todo', TodoSchema)
```

```
createTodo: async (req, res)=>{
    try{
        await Todo.create({todo: req.body.todoItem, completed: false})
        console.log('Todo has been added!')
        res.redirect('/todos')
    }catch(err){
        console.log(err)
    }
},
```

The new item will show up in the ul once user enters the item in the form.



That is the basics of MVC! Hopefully, you will have gained a little more knowledge with this lecture. Until next time!