92)Module introduction

Here we will dive into very important aspect of building web applications. We want to follow a certain pattern for structuring our code and with that I dnt mean how we split it over files or how we write the code but I mean how we logically separate our code and the different functions it fulfils or different things it does. We will dive into mvc pattern.

93)What is MVC

Mvc separation of concern. So making sure that different parts of your code do different things and you clearly know which part is responsible for what.

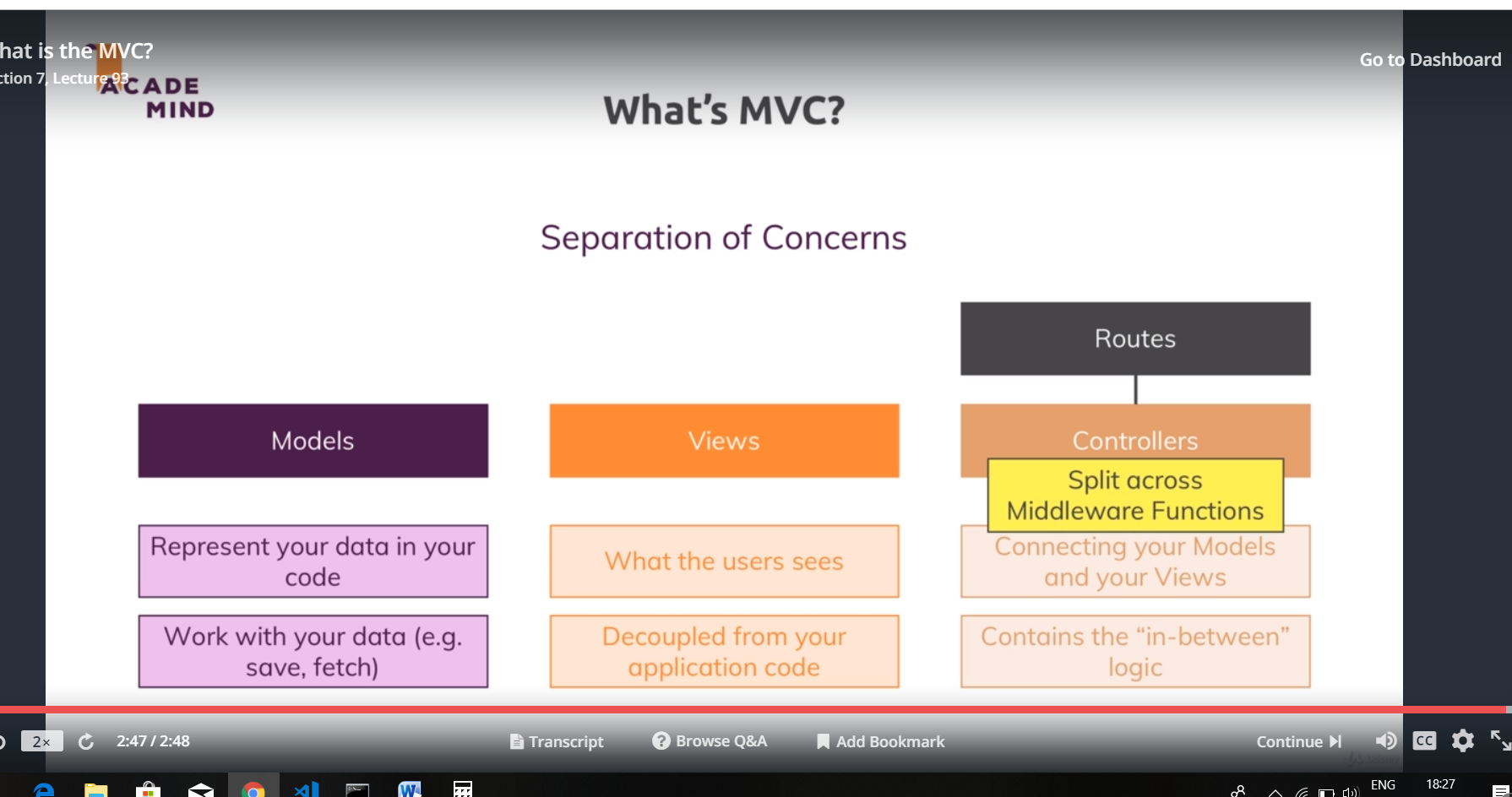
Mvc stands for model, view and controller. So we work with these. You already know views. Mdoels are basically objects or is a prt of your code that is responsible for representing data in your code and allowing you to work with data. so things like saving data, fetching data to or from a file or even if it’s just in memory data(like we are doing right now), this should be handled by model.

Views we already know. They are responsible for rendering right content in our html documents and sending that back to user. So they are decoupled from your application code and are just having some light or minor integrations regarding the data we inject into our templating engine to generate these views.

Controllers are connection points between models and views. Because since the views should’nt care about application logic and models do care about how to save and fetch data and so on , the controllers are the things working with models, saving that data or triggering that save process and so on. And also the part where they then pass that data that was fetched to your views, for ex. So controller is middleman, it contains the in between logic.

Now routes define, on which path and on which http method, which controller code should execute. And as I said then controller is thing defining with which model to work and which view to render.

So this is the pattern and app built with express(as we are doing it) which relies heavily on this middleware , the controller are also kind of split up across middleware functions or some of the logic might be separated and moved into another middleware function but we will see all of that and we will get there step by step. Lets implement mvc pattern in our app.



94)Adding controllers

We have mall our views in views folder. what is missing is controllers and models. Right now that is mixed into our route files or into our route function here(shop.js and admin.js) they way we route we wnt change, we have function which will be executed fro matching route,

router.get('/', (req, res, next) => {

const products = adminData.products;

console.log(products.length > 0);

// res.sendFile(path.join(\_\_dirname, '..', 'views', 'shop.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views', 'shop.html'));

// res.sendFile(path.join(rootDir, 'views', 'shop.html'));

res.render('shop', {

prods: products,

pageTitle: 'Shop',

path: '/',

hasProducts: products.length > 0,

activeShop: true,

productCSS: true

});

});

But the logic executed here is typical controller logic. In first line we interact with data and then we re returning the view and that is exactly in between logic that makes up the controller. Then you can say that we already got our controllers, shop.js and admin.js hold our controller logic. And you wnt be wrong. But as our app grows these files will grow in size and therefore separating this into separate files will be good idea. ten you can see which routes you have and if you want to see the code which executes per route, you simply go into respective controller file and function.

Create a new folder called controllers. Now you can have one to one mapping between your route file names and route file number and your controller file, but you can aslo split it differently. May be you want to group your routes by prefix, these are all admin routes, but thing you execute there might fit different controller. Lets ay you have admin routes which allow you to change admin user profile and products, you might have all these routes in admin.js files in routes folder but you do have 2 different controllers- te suers or admin controller and product controller. This is upto you and will create different files here. I will have have all product related logic in product.js in controller folder, later I I have user logic, that will go into user controller. May be we even split product controller into user product and an admin product controller. Lets split up our logic.

We had this-

Admn.js-

router.get('/add-product', (req, res, next) => {

// res.sendFile(path.join(\_\_dirname, '../', 'views','add-product.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views','add-product.html'));

// res.sendFile(path.join(rootDir,'views', 'add-product.html'));

res.render('add-product', {

pageTitle: 'Add Product',

path: '/admin/add-product',

formsCSS: true,

productCSS: true,

activeAddProduct: true

});

}

);

After splitting-

Admin.js-

const productController = require('../controllers/products');

router.get('/add-product', productController.getAddProduct);

dnt execute the function, we are just passing its refrence.

products.js-

exports.getAddProduct = (req, res, next) => {

// res.sendFile(path.join(\_\_dirname, '../', 'views','add-product.html'));

// res.sendFile(path.join(\_\_dirname, '../', 'views','add-product.html'));

// res.sendFile(path.join(rootDir,'views', 'add-product.html'));

res.render('add-product', {

pageTitle: 'Add Product',

path: '/admin/add-product',

formsCSS: true,

productCSS: true,

activeAddProduct: true

});

};

Here if we press f12 by going to property name in admn.js, we will go directly to function definition in product.js. also when you press productController., you get suggestion about functions avalaible on productController.

We do this for all rotes, now our app.js file is very lean. App.js-

const express = require('express');

const bodyParser = require('body-parser');

const path = require('path');

const app = express();

app.set('view engine', 'ejs');

app.set('views', 'views');

const adminRoutes = require('./routes/admin');

const shopRoutes = require('./routes/shop');

app.use(bodyParser.urlencoded({extended: false}));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/admin',adminRoutes);

app.use(shopRoutes);

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

res.render('404', {pageTitle: 'Page Not Found'});

});

app.listen(3000);

we have 2 controller files in controller folder, product.js and error.js. error.js is for rendering 404 route.

96)Adding a Product model

You can define your model in which ever way you want, you can for example simply export a constructor function. which you can use to create new object. this es5 constructor function pattern. But if you are using es6, like we are doing here, you can create a class.

Create a folder called model, in it create file called model.js-

const products = [];

module.exports = class Product {

constructor(t) {

this.title = t;

}

save() {

products.push(this);

}

static fetchAll() {

return products;

}

}

Now we used static keyword because we want to call fetchAll method on Product class and not single object.

Then we use this model in our controller like this-

Controller/product.js-

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {

res.render('add-product', {

pageTitle: 'Add Product',

path: '/admin/add-product',

formsCSS: true,

productCSS: true,

activeAddProduct: true

});

};

exports.postAddProduct = (req,res,next) => {

const product = new Product(req.body.title);

product.save();

res.redirect('/');

};

exports.getProducts = (req, res, next) => {

const products = Product.fetchAll();

res.render('shop', {

prods: products,

pageTitle: 'Shop',

path: '/',

hasProducts: products.length > 0,

activeShop: true,

productCSS: true

});

};

Question-

<https://www.udemy.com/nodejs-the-complete-guide/learn/v4/questions/5463650>

97)Storing Data in Files Via Model

By the way to read very big files, there are more efficient ways because you don’t want to read them all into memory before you work with them, you can read them as stream them, there is such a function(createReadStream is such a function), but here file is not large, so we will use simple method. When we read the file, callback will be executed. Here we will get error or fileContent, fileContent will be buffer though. Now we have not created products.json file in data folder.

Model/products.js-

save() {

products.push(this);

const p = path.join(process.mainModule.filename, 'data','products.json');

fs.readFile(p, (err, fileContent) => {

console.log(fileContent);

});

}

We get undefined printed on screen. Lets print err obect also, now we will see the error on console, that file does not exist. Lets handle this condition-

Now in order to make sure that this points to our product object, we should pass arrow function to readFile. Otherwise this will loose its context, and will not refer to object.

This is how we handle it.

Models/product.js-

const fs = require('fs');

const path = require('path');

const products = [];

module.exports = class Product {

constructor(t) {

this.title = t;

}

save() {

const p = path.join(path.dirname(process.mainModule.filename),'data','products.json');

fs.readFile(p, (err, fileContent) => {

let products =[];

if(!err) {

products = JSON.parse(fileContent);

}

products.push(this);

fs.writeFile(p, JSON.stringify(products),(err) => {

console.log(err);

});

});

}

static fetchAll() {

const p = path.join(path.dirname(process.mainModule.filename), 'data', 'products.json');

fs.readFile(p, (err, fileContent) => {

if(err) {

return [];

}

return JSON.parse(fileContent);

});

}

}

This is how we save file. after saving file, we want to retrieve contents of file. so we define fetch for it. This is how we call fetchAll in controllers/product.js-

const Product = require('../models/product');

exports.getProducts = (req, res, next) => {

const products = Product.fetchAll();

res.render('shop', {

prods: products,

pageTitle: 'Shop',

path: '/',

hasProducts: products.length > 0,

activeShop: true,

productCSS: true

});

};

If we run , we get this error-

TypeError: Cannot read property 'length' of undefined

This error is because fetchAll returns value asynchronously and when try to execute products.length, product is undefined. And hence the error.

98)Fetching data from files via model

So fetchAll method puts asynchronous code in queue, and finishes with this function. as fetchAll does not return anything(in synchronous mode), value of products in controller is undefined. There are multiple ways of fixing it. For now we will use callback approach.

In controllers/product.js-

exports.getProducts = (req, res, next) => {

Product.fetchAll((products) => {

res.render('shop', {

prods: products,

pageTitle: 'Shop',

path: '/',

hasProducts: products.length > 0,

activeShop: true,

productCSS: true

});

});

};

In models/products.js-

static fetchAll(cb) {

const p = path.join(path.dirname(process.mainModule.filename), 'data', 'products.json');

fs.readFile(p, (err, fileContent) => {

if(err) {

cb([]);

}

console.log('returning the value');

cb(JSON.parse(fileContent));

});

}

99)Refactoring the file storage code

Here we reused some code. We created a function called getProductsFromFile. It reads the data from file.

Model/Products.js-

const fs = require('fs');

const path = require('path');

const products = [];

const p = path.join(path.dirname(process.mainModule.filename), 'data', 'products.json');

const getProductsFromFile = (cb) => {

fs.readFile(p, (err, fileContent) => {

if (err) {

cb([]);

} else {

cb(JSON.parse(fileContent));

}

});

};

module.exports = class Product {

constructor(t) {

this.title = t;

}

save() {

getProductsFromFile((products) => {

products.push(this);

fs.writeFile(p, JSON.stringify(products),(err) => {

console.log(err);

});

});

}

static fetchAll(cb) {

getProductsFromFile(cb);

}

}

101)Useful Resources & Links

Attached, you find the source code for this section.

When using my source code, make sure to run npm install in the extracted folder!

Useful resources:

* More on MVC: <https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture>