* **MVC in node js**

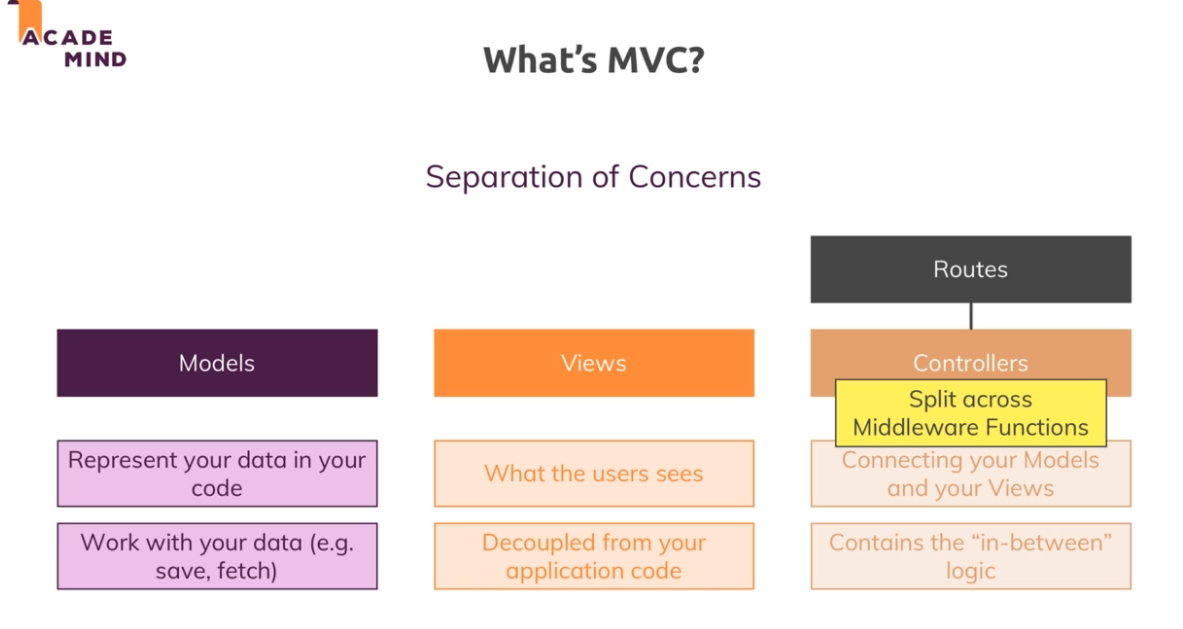
Mvc is used to separation of concern , by mvc you can separate your files and do know that which file is doing what

Mvc stands for model view controller

The model is responsible for representing the data in your code like save fetch even the code in memory

View id responsible for what the user see at the end responsible for rendering the write content into html uses templates

Controllers are the connection point between your models and views it contains the in between logic it controls which model should work with which view like routing the templates to user



So now we have to create our controller folder in which we can create our all the routes logic and export them so the routes of required them can call them

This code is in the controller file

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

    res.render('admin.ejs', { docTitle: 'productPage'});

}

// so you can export this function inside some routes

We are calling the above exported code in the adim.js file

// so that we can require file and can use exports in their

const getController = require('../controllers/products.js');

// here using getAddProduct export in the admin.get request

admin.get('/product',getController.getAddProduct);

and we can use all the routes code in one single file and can export them with different names so that different routes can use it

const addProduct = [];

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

    addProduct.push({title : req.body.title});

    res.redirect('/');

}

exports.getShop = (req,res,next) => {

    res.render('shop',{prods: addProduct, docTitle: 'shopPage'});

}

admin.post('/All-product',getController.postProduct);

routs.get('/',getController.getShop);

* **Modules in mvc**

Now we can use the modules to stores our data and save it and fetch it for this let us use the array of products in different space that we are using above

So for that facility we create a different file of product.js in our module folder which is used

To store thing in the array and use it when ever requires

// so modules plays with the data and the memory storage

const products = [];

module.exports = class product {

    constructor(t) {

        this.title = t;

    }

    save() {

        products.push(this); // push all the variables or function of the constructor

    }

    static fetch() {

        return products;

    }

}

So above we are exporting the class inside our controller file so that the use the product to save and fetch to the server when required

**const product = require('../models/product.js');**

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

**const products = new product(req.body.title);**

**products.save();**

    res.redirect('/');

}

exports.getShop = (req,res,next) => {

**const products = product.fetch();**

    res.render('shop',{prods: products, docTitle: 'shopPage'});

}

So instead of creating array inside the controller we are creating in different folder so that we manage the database at one specific file.

So above code we are getting the exports from the modules and saving the req title in product as product.save and then fetching it when required.

* ***Read and write file instead of pushing array of product in modulo***

So we know let us write file from server to in our data and read from our data to server

const { json } = require('body-parser');

const fs = require('fs');

const path = require('path');

const { setFlagsFromString } = require('v8');

module.exports = class product {

    constructor(t) {

        this.title = t;

    }

    save() {

**const p = path.join(\_\_dirname,'../','Data','product.json');**

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

**let product = [];**

**if(!err) { // check file present in the path or not**

**product = (JSON.parse(fileContent)); // parse because arry in form of array**

**}**

**product.push(this); push this content in array**

**fs.writeFile(p, JSON.stringify(product), (err) => { // write in the file**

**console.log(err);**

**})**

**})**

**}**

    static fetch() { // so to adjust the async we have to use a callback function in it which will run when it read the file

**fs.readFile(p, (err,fileContent) => { // fs.read is async function so to it will throw error if we access it before time**

**if(err) {**

**return [];**

**}**

**return JSON.parse(fileContent);**

**})**

**return products;**

**}**

}

So we have to pass the array all time and because we are storing into some file that’s why we have to stringify it because we are writing the data and to readFile we have to parse it

To save file we have to check for error and that is if file is present in the file if not then we write the empty array

**But if we return something inside the read file it will throw an error because the return value get run before the file reads completely because read file is a async function so for that we have call a function which can hold the return values and it will run only when the async read run completely.**

**static fetch(cb) { // so to adjust the async we have to use a callback function in it which will run when it read the file**

**const p = path.join(\_\_dirname,'../','Data','product.json');**

**fs.readFile(p, (err,fileContent) => { // fs.read is async function so to it will throw error if we access it before time**

**if(err) {**

**cb([]); // we are using function so it will run only once the read file runs**

**}**

**cb(JSON.parse(fileContent)); // read file is async so it will run when it reads the whole**

**})**

**return products;**

**}**

**}**

**So here we are calling cb as call back function which run when the async function comes in the call back stack**

**exports.getShop = (req,res,next) => {**

**product.fetch((products) => {**

**res.render('shop',{prods: products, docTitle: 'shopPage'});**

**});**

**So we can call fetch as like above in which we are calling a product which now holding an array**

**So now we add all the necessary files which is required to create product page so we just bit modified our pages**