Technologies used

1. What is MVC and why it is used for making application?

MVC stands for model view and controller. MVC Is a good coding practise that most of the software developers follow when the complexity of the implementation becomes more. MVC is used as separation of concern and it leads to better code understanding and scalability of code.

The model consists of data in form of data and classes are used to manipulate the data with specified function to do so. The Views are the front-end representation of the application through the user interacts. This deals with the front end logic. The controller is the connection with the Model and the Views which is predominantly used for routing and accessing data from model by controller

1. What is node js and Express ?

Node js is a back-end JavaScript technology which is used for server side logic implementation. It is a server side programming language.

Implementation of Node can be cumbersome for scalability purpose so we have a third party package to handle such issues Express.

1. What is Bootstrap ?
   1. Bootstrap Is a front end framework build with CSS and jQuery for responsible design of the application bootstrap is used for front end animations.
2. What is Mongo DB ?
   1. Mongo dB is NoSQL database which is used to implement document styles JSON and BSON data collections. NoSQL database are ideal for scalability and provide high performance. Mongo DB is generally used with data intensive application

Code snippets

1. App.js is the index JavaScript project that is server first by your node cli. This setting can be changed In the package.json file. For creating a node js application with using express framework export it from node’s environment

const express = require('express');

const app = express();

1. For parsing the body of incoming requests

const bodyParser = require('body-parser')

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

1. For using the functions and classes declared In other file than the present file, use require() keyword and specifying the path of the file to be imported. The constant is used for accessing the methods and the classes declared in other file. Use relative addressing scheme

const errorController = require('./controllers/error');

const mongoConnect = require('./util/database').mongoConnect;

const User = require('./models/user');

1. The express also provides the static public folder that can be used for static files like css javascript and images which can be configured by

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

1. The experess is a sequence of middleware functions that keeps on serving incoming requests and handle those. The middleware function takes 2 arguments the first one is the incoming link for request to be and the another will be a call back function for handling the request that takes 3 arguments request and response objects and n next object that passes the request to the other successive middleware function for handling the request
2. app.use((req, res, next) => {
3. User.findById('5baa2528563f16379fc8a610')
4. .then(user => {
5. req.user = new User(user.name, user.email, user.cart, user.\_id);
6. next();
7. })
8. .catch(err => console.log(err));
9. });

6 . For starting the server use

app.listen(3000);

7. Best practice is to use the routes for handling conceptual similar kind of requests

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

here the admin router is the folder that deals with the admin related routing logic

The contents of the file has the location and implementation of the controller that has middleware functions for handling requests

const path = require('path');

const express = require('express');

const adminController = require('../controllers/admin');

const router = express.Router();

*// /admin/add-product => GET*

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

*// /admin/products => GET*

router.get('/products', adminController.getProducts);

*// /admin/add-product => POST*

router.post('/add-product', adminController.postAddProduct);

router.get('/edit-product/:productId', adminController.getEditProduct);

router.post('/edit-product', adminController.postEditProduct);

router.post('/delete-product', adminController.postDeleteProduct);

module.exports = router;

8. The controller in place preforms the middleware that implements the conceptual handling of all the middleware for handling requests.

For the admin related processing of requests we have a admin controller. Sample code is given below

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

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

res.render('admin/edit-product', {

pageTitle: 'Add Product',

path: '/admin/add-product',

editing: false

});

};

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

const title = req.body.title;

const imageUrl = req.body.imageUrl;

const price = req.body.price;

const description = req.body.description;

const product = new Product(

title,

price,

description,

imageUrl,

null,

req.user.\_id

);

product

.save()

.then(result => {

*// console.log(result);*

console.log('Created Product');

res.redirect('/admin/products');

})

.catch(err => {

console.log(err);

});

};

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

const editMode = req.query.edit;

*if* (!editMode) {

*return* res.redirect('/');

}

const prodId = req.params.productId;

Product.findById(prodId)

*// Product.findById(prodId)*

.then(product => {

*if* (!product) {

*return* res.redirect('/');

}

res.render('admin/edit-product', {

pageTitle: 'Edit Product',

path: '/admin/edit-product',

editing: editMode,

product: product

});

})

.catch(err => console.log(err));

};

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

const prodId = req.body.productId;

const updatedTitle = req.body.title;

const updatedPrice = req.body.price;

const updatedImageUrl = req.body.imageUrl;

const updatedDesc = req.body.description;

const product = new Product(

updatedTitle,

updatedPrice,

updatedDesc,

updatedImageUrl,

prodId

);

product

.save()

.then(result => {

console.log('UPDATED PRODUCT!');

res.redirect('/admin/products');

})

.catch(err => console.log(err));

};

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

Product.fetchAll()

.then(products => {

res.render('admin/products', {

prods: products,

pageTitle: 'Admin Products',

path: '/admin/products'

});

})

.catch(err => console.log(err));

};

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

const prodId = req.body.productId;

Product.deleteById(prodId)

.then(() => {

console.log('DESTROYED PRODUCT');

res.redirect('/admin/products');

})

.catch(err => console.log(err));

};

Each result Is associated with promises and cllback function to check if the execution of the promises results in success or failure and all of that conditions are properly handled or not.

9. The views consists of several ejs files. EJS is a JavaScript parsing engine which helps in outputiing dynamic content on html files by passing javascript objects from the controller. Also the ejs files do support javascript code that helps in rendering those contents which can conditionally vary. One such file from the project is given below

For using ejs tempelating engine there is a need to configure it. This can be done by

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

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

Also demo ejs code liiks like javascript embedded in html file

<%- include('../includes/head.ejs') %>

<link rel="stylesheet" href="/css/forms.css">

<link rel="stylesheet" href="/css/product.css">

</head>

<body>

<%- include('../includes/navigation.ejs') %>

<main>

<form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else { %>add-product<% } %>" method="POST">

<div class="form-control">

<label for="title">Title</label>

<input type="text" name="title" id="title" value="<% if (editing) { %><%= product.title %><% } %>">

</div>

<div class="form-control">

<label for="imageUrl">Image URL</label>

<input type="text" name="imageUrl" id="imageUrl" value="<% if (editing) { %><%= product.imageUrl %><% } %>">

</div>

<div class="form-control">

<label for="price">Price</label>

<input type="number" name="price" id="price" step="0.01" value="<% if (editing) { %><%= product.price %><% } %>">

</div>

<div class="form-control">

<label for="description">Description</label>

<textarea name="description" id="description" rows="5"><% if (editing) { %><%= product.description %><% } %></textarea>

</div>

<% if (editing) { %>

<input type="hidden" value="<%= product.\_id %>" name="productId">

<% } %>

<button class="btn" type="submit"><% if (editing) { %>Update Product<% } else { %>Add Product<% } %></button>

</form>

</main>

<%- include('../includes/end.ejs') %>

All the code that is enclosed inside the <%= %> is the ejs code for outputting javascript code. Details are available on the official documentation

10. Working with Model

A model is a conceptual construct that is used to manipulate with the data. This can be used for all the data related activities. There is no one to one connection between the controllers and the models. One such example of model is the product model. Model has in generl several static methods for manipulating data is some kind of database is used

const mongodb = require('mongodb');

const getDb = require('../util/database').getDb;

class Product {

constructor(title, price, description, imageUrl, id, userId) {

*this*.title = title;

*this*.price = price;

*this*.description = description;

*this*.imageUrl = imageUrl;

*this*.\_id = id ? new mongodb.ObjectId(id) : null;

*this*.userId = userId;

}

save() {

const db = getDb();

let dbOp;

*if* (*this*.\_id) {

*// Update the product*

dbOp = db

.collection('products')

.updateOne({ \_id: *this*.\_id }, { $set: *this* });

} *else* {

dbOp = db.collection('products').insertOne(*this*);

}

*return* dbOp

.then(result => {

console.log(result);

})

.catch(err => {

console.log(err);

});

}

static fetchAll() {

const db = getDb();

*return* db

.collection('products')

.find()

.toArray()

.then(products => {

console.log(products);

*return* products;

})

.catch(err => {

console.log(err);

});

}

static findById(prodId) {

const db = getDb();

*return* db

.collection('products')

.find({ \_id: new mongodb.ObjectId(prodId) })

.next()

.then(product => {

console.log(product);

*return* product;

})

.catch(err => {

console.log(err);

});

}

static deleteById(prodId) {

const db = getDb();

*return* db

.collection('products')

.deleteOne({ \_id: new mongodb.ObjectId(prodId) })

.then(result => {

console.log('Deleted');

})

.catch(err => {

console.log(err);

});

}

}

module.exports = Product;

This is a model of class Product which is used to manipulate the product related logic from the database or from the controller.

11. Working with mongodb database

To implement mongodb databse there is a need to import the database.js file in the utility folder that contains the code for creating the database connection and this file is used to create database collection that does not have one. This is handled by the mongoConnect custom function

In the database.js file in util folder, code for connecting and fetching database is

const mongodb = require('mongodb')

const mongoClient = mongodb.MongoClient; *// calling the mogodb client costructor*

let url = "mongodb://localhost:27017/practiceBookWithMe";

let \_db *// this underscore variable is used in this file only to fetch the type of database schema*

const mongoconnect = (callback) => { *// callback function passed on creating and connecting application to the database server*

mongoClient.connect(url,{useNewUrlParser:true}).

then(client =>{

console.log('Connected');

\_db = client.db()

callback(client);

})

.catch(err =>{

console.log('Error in connecting database');

*throw* err;

})

}

*// this function connects to the database \_db or else returns error on not finding database*

const getDb = () =>{

*if*(\_db)

{

*return* \_db;

}

*else*{

*throw* 'No database found';

}

}

exports.mongoconnect = mongoconnect;

exports.getDb = getDb;

1. Accessing database with MongoConnect and mongoClient inside the model and CRUD oeration in database

const mongodb = require('mongodb');

const getDb = require('../util/database').getDb;

class Product {

constructor(title, price, description, imageUrl, id, userId) {

*this*.title = title;

*this*.price = price;

*this*.description = description;

*this*.imageUrl = imageUrl;

*this*.\_id = id ? new mongodb.ObjectId(id) : null;

*this*.userId = userId;

}

save() {

const db = getDb();

let dbOp;

*if* (*this*.\_id) {

*// Update the product*

dbOp = db

.collection('products')

.updateOne({ \_id: *this*.\_id }, { $set: *this* });

} *else* {

dbOp = db.collection('products').insertOne(*this*);

}

*return* dbOp

.then(result => {

console.log(result);

})

.catch(err => {

console.log(err);

});

}

static fetchAll() {

const db = getDb();

*return* db

.collection('products')

.find()

.toArray()

.then(products => {

console.log(products);

*return* products;

})

.catch(err => {

console.log(err);

});

}

static findById(prodId) {

const db = getDb();

*return* db

.collection('products')

.find({ \_id: new mongodb.ObjectId(prodId) })

.next()

.then(product => {

console.log(product);

*return* product;

})

.catch(err => {

console.log(err);

});

}

static deleteById(prodId) {

const db = getDb();

*return* db

.collection('products')

.deleteOne({ \_id: new mongodb.ObjectId(prodId) })

.then(result => {

console.log('Deleted');

})

.catch(err => {

console.log(err);

});

}

}

module.exports = Product;

Other third party application used:

1. Nodemon
2. Jquery
3. Body-parser
4. Popper.js
5. Bootstrap
6. Mongodb
7. Ejs
8. express

this all information Is given in the package.json file

{

"name": "bookwithme",

"version": "1.0.0",

"description": "bookwothme is an online rental service",

"main": "app.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "nodemon app.js"

},

"repository": {

"type": "git",

"url": "git+https://github.com/PrayagDesaiweb/BookWithMe.git"

},

"author": "Prayag Desai",

"license": "ISC",

"bugs": {

"url": "https://github.com/PrayagDesaiweb/BookWithMe/issues"

},

"homepage": "https://github.com/PrayagDesaiweb/BookWithMe#readme",

"dependencies": {

"body-parser": "^1.18.3",

"bootstrap": "^4.2.1",

"ejs": "^2.6.1",

"express": "^4.16.4",

"jquery": "^3.3.1",

"mongodb": "^3.1.10",

"nodemon": "^1.18.9",

"popper.js": "^1.14.6"

}

}