Паттерн **MVC** (Model-View-Controller) часто используется при разработке web-приложений, в том числе при написании кода на Node.js.

Основными компонентами паттерна являются:

* **модели**, которые определяют структуру и логику некоторых данных;
* **представления**, определяющие визуальную составляющую приложения и то, как будут отображаться данные;
* **контроллеры**, контролирующие поток данных в модели и представления.

В качестве дополнительного компонента при разработке веб-приложений может использоваться **система маршрутизация**, которая сопоставляет http-запросы с маршрутами и выбирает обрабатывающий контроллер.

В общем виде паттерн MVC применительно к проекту веб-приложения можно представить так, как показано на рисунке 1.

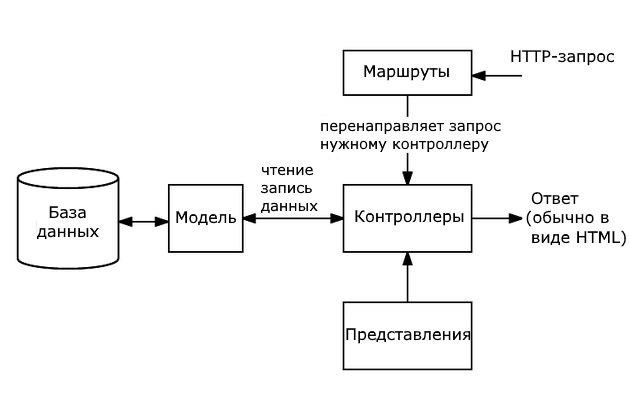


Рисунок 1 – Общий вид паттерна MVC для веб-приложения

Разберем элементы MVC, которые использовались при разработке веб-сайта с авторизацией и обратной связью на Node.js.

Структура файлов и каталогов проекта представлена на рисунке 2.

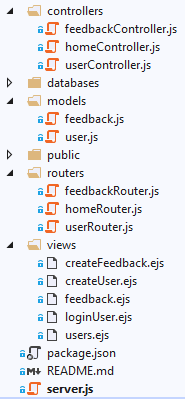


Рисунок 2 – Структура проекта

Модели расположены в каталоге models. Для наглядности представим код модели user.js. Модель описывает сущность (таблицу) из базы данных.

const Sequelize = require('sequelize');

const sequelize = new Sequelize({

dialect: 'sqlite',

storage: './databases/site.db'

});

const User = sequelize.define(

'user',

{

id: {

type: Sequelize.DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

login: {

type: Sequelize.DataTypes.TEXT,

allowNull: false,

},

email: {

type: Sequelize.DataTypes.TEXT,

allowNull: false,

},

password: {

type: Sequelize.DataTypes.TEXT,

allowNull: false,

},

avatar: {

type: Sequelize.DataTypes.TEXT,

allowNull: true,

},

},

{

timestamps: false,

}

);

module.exports = User;

Представления расположены в каталоге views и отвечают за форматированный вывод данных. Приведём код представления feedback.ejs. Оно отображает список вопросов от людей с их именами, почтами и аватарами:

<!DOCTYPE html>

<html>

<head>

<title>Список вопросов</title>

<meta charset="utf-8" />

<link rel="stylesheet" href="styles.css">

</head>

<body>

<a href="/feedbacks/create">Добавить вопрос</a>

<h1>Список вопросов</h1>

<table border="1">

<tr><th>Имя</th><th>Почта</th><th>Тема</th><th>Вопрос</th><th>Изображение</th></tr>

<% for (var i = 0; i < feedbacks.length; i++) { %>

<tr><td><%= feedbacks[i].name %></td>

<td><%= feedbacks[i].email %></td>

<td><%= feedbacks[i].topic %></td>

<td><%= feedbacks[i].message %></td>

<% if (feedbacks[i].image == null) { %>

<td>Отсутствует</td></tr>

<% } else { %>

<td><img src="./scans/<%= feedbacks[i].image %>" width="200" alt="Изображение"></td></tr>

<% }

} %>

</table>

</body>

</html>

Все контроллеры собраны в каталог controllers и нужны для взаимодействия с представлениями (в основном параметризированными) и моделями. Приведём блок кода из userController.js. В блоке присутствует отрисовка обычного и параметризированного представления, а также обращение к модели (массив всех пользователей из базы данных) и передача данных из модели представлению.

const bcrypt = require('bcrypt');

const salt = bcrypt.genSaltSync(10);

const User = require("../models/user.js");

const Op = require('Sequelize').Op;

let firstNum;

let secondNum;

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max - min)) + min;

}

exports.addUser = function (request, response) {

response.render("createUser.ejs");

};

exports.loginUser = function (request, response) {

firstNum = getRandomInt(1, 100);

secondNum = getRandomInt(1, 100);

response.render("loginUser.ejs", {

firstNum: firstNum,

secondNum: secondNum,

});

};

exports.getUsers = function (request, response) {

(async () => {

response.render("users.ejs", {

users: await User.findAll()

});

})()

};

В файлах роутеров, собранных в каталог routers, написан код для сопоставления запросов с определёнными контроллерами, например:

const multer = require("multer");

const fs = require('fs-extra');

const express = require("express");

const userController = require("../controllers/userController.js");

const userRouter = express.Router();

userRouter.use("/postuser", multer({

storage: multer.diskStorage({

destination: (req, file, callback) => {

let path = "./public/avatars";

fs.mkdirsSync(path);

callback(null, path);

}

})

}).single("filedata"), userController.postUser);

userRouter.use("/create", userController.addUser);

userRouter.use("/login", userController.loginUser);

userRouter.use("/enteruser", userController.enterUser);

userRouter.use("/", userController.getUsers);

module.exports = userRouter;

Таким образом, использование паттерна MVC позволило снизить сложность разработки за счёт готовой абстракции конструктивных элементов шаблона и внести некоторую логическую ясность и связность в файловую структуру проекта.