УДК 004

Обзор и реализация MVC подхода к созданию веб-приложения

Халиманенков Андрей Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статьей рассматривается создание веб-приложения, основанного на архитектуре MVC. Для этой цели используется фреймворк Express для языка JavaScript. Итогом исследования является веб-приложение с реализацией MVC.
Ключевые слова: создание веб-сайтов, MVC, архитектура веб-приложений, JavaScript.

Overview and implementation of the MVC approach to creating a web application

Khalimanenkov Andrey Sergeevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses the creation of a web application based on the MVC architecture. For this purpose, the Express framework for the JavaScript language is used. The result of the study is a small web application with an MVC implementation.
Keywords: websites creating, MVC, web application architecture, JavaScript.

За последние несколько лет взаимодействие человека с компьютерами и приложениями для них сильно изменилось. То, что раньше делали с помощью командной строки, сегодня работает с помощью графических интерфейсов. В это определение входит известные нам формы ввода, различные checkbox и radio переключатели, а также кнопки подтверждения, сброса или перехода на другие страницы. При этом логика приложений могла остаться прежней и не меняться за несколько десятков лет или измениться не так сильно. При этом интерфейсы обновляют каждые несколько лет и тут поднимается вопрос – а как изменить интерфейс в разрезе от логики приложения. На помощь в решении этого вопроса пришла архитектура MVC, а именно Model-View-Controller или Модель-Представление-Контроллер.

Цель исследования – описать архитектуру MVC на примере фреймворка Express для JavaScript.

Вопрос выбора архитектуры при создании сайтов волнует некоторых исследователей и специалистов: Юрочкин А. Г. и Жулябин Д. Ю. [1] посвятили статью анализу современной архитектуры веб-приложения для решения корпоративных задач. Представили многослойную архитектуру системы. Базаревский В. Э. [2] предложил архитектуру програмной системы, предназначенной для обработки вибрационных данных. Матвеев А. Г. И Якубайлик О. Э. [3] обсудили архитектуру и методы построения разделенного веб-приложения. Сергеев О. А. [4] описал основные черты REST подхода к созданию веб-приложений.

Принцип работы MVC выглядит следующим образом и показан на рисунке 1.

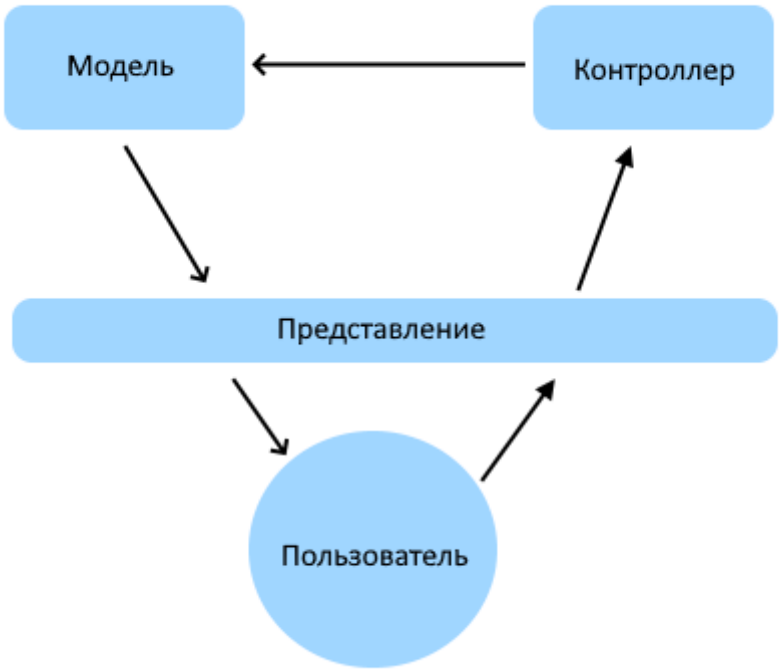

Рисунок 1 – Принцип работы архитектуры MVC

С помощью интерфейса, который является представлением, пользователь взаимодействует с веб-приложением. Данные после взаимодействия отправляются в контроллер, где они обрабатываются для записи в модель. В моделях все данные хранятся с чётко заданной структурой.

Для реализации этого подхода на практике понадобится инициализировать файл package.json в папке с проектом с помощью команды «npm init», введённую в терминал Visual Studio Code или любого другого редактора кода. Эти команды позволяет выполнять Node.JS [5]

Рисунок 2 – инициализация package.json

Затем нужно загрузить фреймворк Express [6] в папку с проектом, использую команду npm install express –save.

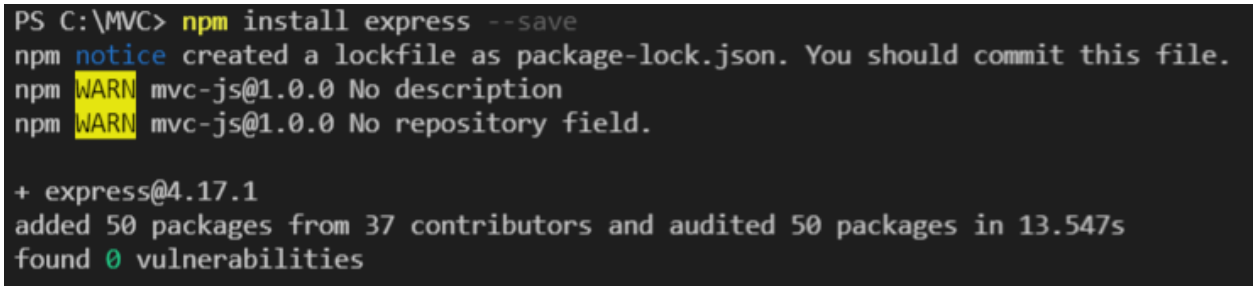

Рисунок 3 – загрузка фреймворка Express

А также нужны модули под названием «hbs» и «body-parser» со встроенным движком для вывода представлений для фреймворка Express и их парсинга соответственно. Вводим команду «npm install hbs body-parser –save» и загружаем.

```
PS C:\MVC> npm install --save hbs
npm WARN mvc-js@1.0.0 No description
npm WARN mvc-js@1.0.0 No repository field.

+ hbs@4.1.2
added 9 packages from 43 contributors and audited 59 packages in 5.489s
found 0 vulnerabilities

PS C:\MVC> npm install --save body-parser
npm WARN mvc-js@1.0.0 No description
npm WARN mvc-js@1.0.0 No repository field.

+ body-parser@1.19.0
updated 1 package and audited 60 packages in 2.077s
found 0 vulnerabilities
```

Рисунок 4 – загрузка модуля hbs и body-parser

Затем создаём директорию и файлы, которые требуются для работы веб-приложения.

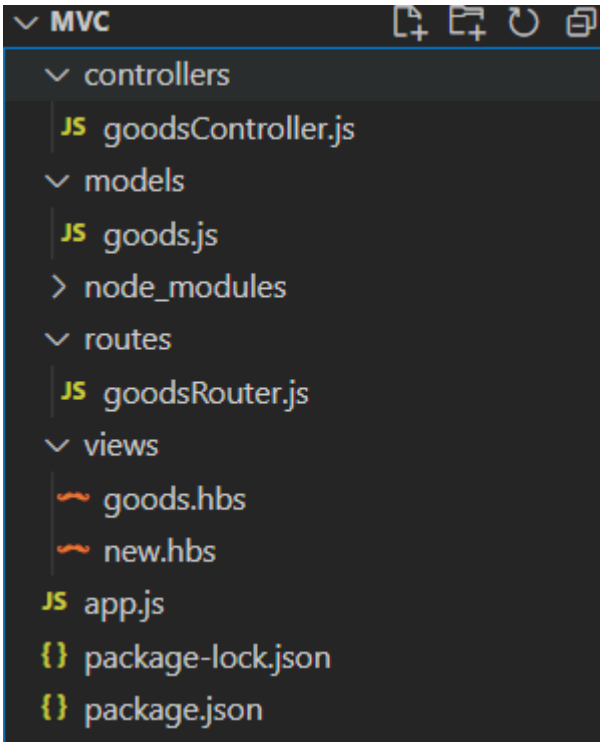

Рисунок 5 – Директория папки с проектом

Главный файл app.js содержит в себе запросы на использование: фреймворка Express, движка представлений hbs, парсера body-express, а также ссылку на страницу с товарами и файл goodsRouter.js, по которому контроллер понимает в какую модель отправлять данные на хранение. Код файла:

```
const express = require("express");
```

```
const app = express();
const bodyParser = require("body-parser");
const goodsRouter = require("./routes/goodsRouter.js");

app.set("view engine", "hbs");
app.use(bodyParser.urlencoded({ extended: false }));

app.use("/goods", goodsRouter);

app.use(function (req, res, next) {
   res.status(404).send("Not Found")
});

app.listen(3000);
```

В папке models находится файл goods.js, который будет хранить код модели:

```
const goods = [];

module.exports= class Goods{

   constructor(name, price){
      this.name = name;
      this.price = price;
   }
   save(){
      goods.push(this);
   }
   static getAll(){
      return goods;
   }
}
```

Модель – это класс Goods. Он содержит price и name, который определяется ценой и названием товара. Все наименования и цены хранятся в массиве goods. Сохранение товаров и их чтение из массива осуществляется методами all и getAll.

В папке views находится файл goods.hbs. Он отображает список товаров из файла goods.js на странице веб-браузера и имеет следующий код:

```
<!DOCTYPE html>
<html>
<head>
   <title>Список товаров</title>
   <meta charset="utf-8" />
</head>
<body>
```

```
<h1>Список товаров</h1>
<table>
<tr><th>Наименование</th><th>Цена</th></tr>
{{#each goods}}
<tr><td>{{this.name}}</td><td>{{this.price}}</td></tr>
{{/each}}
</table>
<br>
<a href="/goods/new">Добавить товар на склад</a>
</body>
```

<html>Также в этой же папке находится файл new.hbs, в свою очередь, он содержит HTML форму для создания нового товара. Вот его код:

```
<!DOCTYPE html>
<html>
<head>
<title>Добавить товар</title>
<meta charset="utf-8" />
</head>
<body>
<h1>Добавить товар на склад</h1>
<form action="postGoods" method="POST">
<label>Наименование</label>
<input name="name" /><br><br>
<label>Цена</label>
<input name="price" type="number" min="0" max="100000" /><br><br>
<input type="submit" value="Отправить" />
</form>
<a href="/goods">К списку товаров/a>
</body>
<html>
```

В папке controllers содержится файл goodsController.js, который хранит три функции:

1. getGoods() – возвращает список товаров с помощью Goods.getAll() в представление new.hbs;
2. postGoods() – принимает данные от пользователя при создании товара, отправляет их в модель, где создаёт новый объёкт в классе Goods. После этого отправляет пользователя на страницу со списком товаров;
3. addGoods() – добавляет новый товар в представление new.hbs

Вот его код:

```
const Goods = require("../models/goods.js");

exports.addGoods = function (request, response){
response.render("new.hbs");
};
```

```
exports.getGoods = function(request, response){
    response.render("goods.hbs", {
        goods: Goods.getAll()
    });
};
exports.postGoods= function(request, response){
    const goodsname = request.body.name;
    const goodsprice = request.body.price;
    const goods = new Goods(goodsname, goodsprice);
    goods.save();
    response.redirect("/goods");
};
```

Для запуска приложения используем команду «node app.js».

```
PS C:\MVC> node app.js
```

Рисунок 6 – Запуск приложения

Затем переходим по ссылке http://localhost:3000/goods/new для того, чтобы открыть страницу добавления товаров. Это и есть одно из двух представлений, с которым взаимодействует пользователь. Его можно менять как угодно, при этом данные будут обрабатываться точно также как они это делали до изменений в интерфейсе.

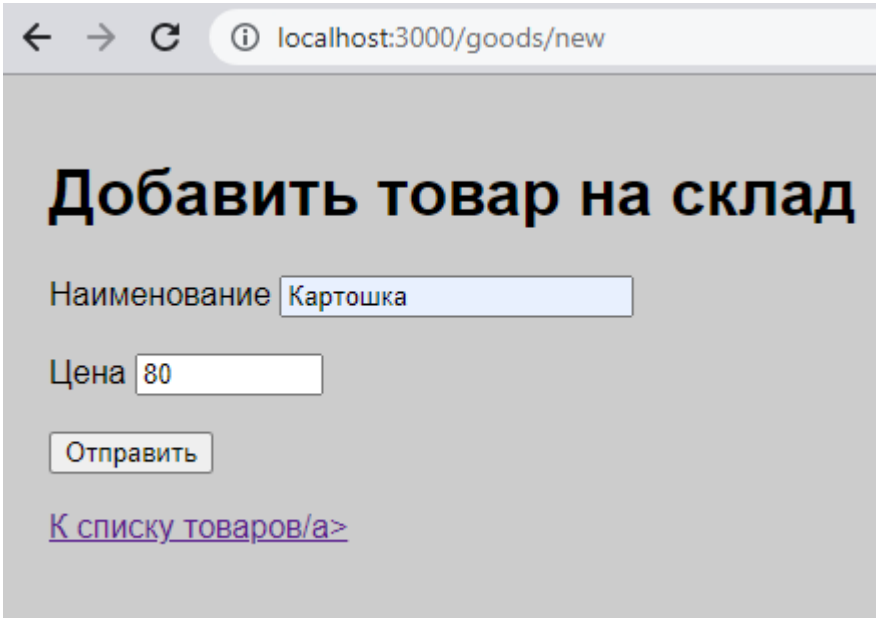

Рисунок 7 – Страница добавления товара на склад

По адресу http://localhost:3000/goods доступен список товаров. Это тоже представление, как и предыдущая страница.

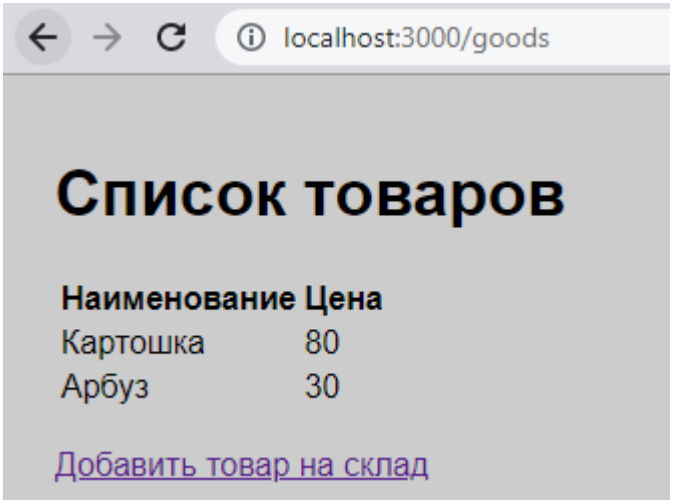

Рисунок 8 – Страница списка товаров

По итогу создано веб-приложение, построенное по архитектуре MVC, которая позволяет разрабатывать и изменять отдельные части приложения без вмешательства в остальные. Это помогает распределить разработку по отдельным узконаправленным специалистам, что позволит снизить расходы на создание и поддержание сайта, а также повысить его отказоустойчивость.

Библиографический список

1. Юрочкин А. Г., Жулябин Д. Ю. Разработка современной архитектуры веб-приложения для решения корпоративных задач //Вестник Воронежского института высоких технологий. 2018. №. 2. С. 101-106.
2. Базаревский В. Э. Архитектура мобильного веб-приложения для обработки сигнальных данных //Доклады Белорусского государственного университета информатики и радиоэлектроники. 2013. №. 1 (71).
3. Матвеев А. Г., Якубайлик О. Э. Разработка веб-приложения для обработки и представления пространственных метаданных геопортала //Сибирский журнал науки и технологий. 2012. №. 2 (42).
4. Сергеев О. А. Использование REST-архитектуры в современных веб-приложениях //Современные научные исследования и инновации. 2019. №. 2. С. 7-7.
5. Node.JS URL: https://nodejs.org/en/ (дата обращения: 21.08.2021)
6. Фреймворк Express URL: https://expressjs.com/ru/ (дата обращения 23.08.2021)