Для разработки сайта, реализующего следующие возможности для пользователя: регистрация, авторизация, возможность задать вопрос и просмотр всех заданных вопросов; выбран паттерн MVC-S (Model-view-controller-services – «модель-вид-контроллер-сервисы»).

Паттерн MVC-S включает ряд компонентов:

* Модели определяют структуру и логику используемых данных;
* Представления (views) определяют визуальную часть (каким образом будут отображаться);
* Контроллеры обрабатывают входящие http-запросы, используя для обработки модели и представления, и отправляют в ответ клиенту некоторый результат обработки, нередко в виде html-кода.
* Сервисы это абстракции, которые уровень обслуживания будет вызывать для получения/обновления необходимых ему данных. Этот уровень обычно вызывает либо базу данных, либо какую-либо другую систему (например, сервер LDAP, веб-служба или NoSql-тип DB).

**Контроллеры** в паттерне MVC позволяют связать представления и модели и выполняют некоторую логику по обработке запроса. В проекте необходимо определить роутер, который будет сопоставляться с адресами (определить обработку перехода по различным страницам сайта). Отдельно необходимо определить директории для хранения роутеров и контроллеров – routes и controllers соответственно (на данный момент это не реализовано в проекте).

В приложении используется фреймворк Express. Для использования его пакетов необходимо подключить фреймворк в проект:

import express from "express";

Рассмотрим представленный ниже код:

initPassportLocal();

let router = express.Router();

let initWebRoutes = (app) => {

router.get("/", loginController.checkLoggedIn, homePageController.handleHelloWorld);

router.get("/comments", commentsController.getPageComments);

router.post("/comments", commentsController.createComment);

router.get("/login", loginController.checkLoggedOut, loginController.getPageLogin);

router.post("/login", checkCaptcha,

passport.authenticate("local", {

successRedirect: "/",

failureRedirect: "/login",

successFlash: true,

failureFlash: true

}));

router.get("/register", registerController.getPageRegister);

router.post("/register", mul.single('avatar'), auth.validateRegister, registerController.createNewUser);

router.post("/logout", loginController.postLogOut);

return app.use("/", router);

};

module.exports = initWebRoutes;

В данном примере рассмотрено применение обработчиков маршрутов. Маршрутизация определяет, как приложение отвечает на клиентский запрос к конкретному адресу (URI).

**Представления** в рамках проекта созданы для отображения сообщения об успешной авторизации пользователей и чтения вопросов. В качестве движка представлений использован движок EJS. Для хранения представлений создана директория views. Файл homapage.ejs содержит код, который позволяет отобразить сообщение об успешной авторизации пользователя и вывести его никнейм. Ниже представлен код файла homepage.ejs:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Home page</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

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

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"

integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

</head>

<body>

<div class="container">

<div class="container mt-5">

<h1>Авторизация прошла успешно!</h1>

<div>

<h2> Добро пожаловать "<%= user.fullname%>"</h2>

<img class="useravatar" src="<%= user.avatarpath%>" alt="Fullname">

<br>

<div>

<form method="POST" action="/logout">

<button type="submit" class="btn btn-primary">Выйти</button>

</form>

</div>

</div>

</div>

</div>

</body>

</html>

Таким образом, движок ejs позволяет динамически подставить необходимые данные в разметку html-файла.

Файл comments.ejs позволяет отобразить заданные вопросы, хранящиеся в базе данных. Ниже представлен код:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css" integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4" crossorigin="anonymous">

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js" integrity="sha384-uefMccjFJAIv6A+rW+L4AHf99KvxDjWSu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm" crossorigin="anonymous"></script>

</head>

<body>

<div class="container">

<div class="container">

<div class="row mt-5">

<div class="col-7 login-left">

</div>

<div class="container">

<div class="row">

<div class="col-lg-12">

<h2 class="text-center">

Вопросы:

</h2>

</div>

<div class="col-lg-6">

<div id="comment-field">

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

<p class="alert alert-primary" role="alert">

<%= comments[i].name\_user %>

</p>

<p class="alert alert-success" role="alert">

<%= comments[i].comment %>

</p>

<% } %>

</div>

</div>

<div class="col-lg-6">

<form method="POST" action="">

<div class="form-group">

<label for="comment-name">Ваше имя:</label>

<input class="form-control" id="comment-name" name="username" placeholder="Введите ваше имя">

</div>

<div class="form-group">

<label for="comment-body">Вопрос:</label>

<textarea class="form-control" id="comment-body" name="comment" placeholder="Текст вопроса"></textarea>

</div>

<div class="form-group form-check text-right">

<button type="submit" id="comment-add" class="btn btn-primary">Добавить вопрос</button>

</div>

</form>

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js" integrity="sha384-cs/chFZiN24E4KMATLdqdvsezGxaGsi4hLGOzlXwp5UZB1LY//20VyM2taTB4QvJ" crossorigin="anonymous"></script>

<script src="js.js"></script>

<div>

Если у Вас уже имеется аккаунт: <a href="/login">Вернуться на страницу авторизации</a>

</div>

<br><br>

<div>

Для связи с нами пишите на почту: niksvergun@gmail.com

Либо звоните по телефону 89648453571 </a>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

При переходе на страницу вопросов выполняется функция считывания информации из таблицы comments в базе данных. В цикле последовательно выводятся все найденные записи прямо в разметку html.

**Сервисы -** уровень служб т.е "connector", "middleman", "server side Controller" в том смысле, что то, что контроллер стороны "client" делает для клиента, "Service" делает для сервера.

Другими словами, на стороне клиента "Controller" только "talks" с контроллером "Service" на стороне сервера.

Контроллер ---> Запросы и получение от уровня обслуживания <-----

Уровень обслуживания извлекает или передает информацию тем слоям на стороне сервера, которые в ней нуждаются.

Сам по себе Сервис ничего не делает, кроме подключения серверных уровней к тому, что им нужно.

Пример кода из файла сервиса на добавление комментариев:

import DBConnection from "./../configs/DBConnection";

import bcrypt from "bcryptjs";

let createComment = (data) => {

return new Promise(async (resolve, reject) => {

let comment = {

name\_user: data.username,

comment: data.comment

};

DBConnection.query(

' INSERT INTO comments set ? ', userItem,

function (err, rows) {

if (err) {

reject(false)

}

resolve("Комментарий успешно добавлен");

}

);

});

};

module.exports = {

createComment: createComment

};

Таким образом, на данный момент в проекте лишь частично применен паттерн MVCS.