В моей работе использовался следующий шаблон программирования – **MVC**. MVC (Model View Controller) — это шаблон программирования, который позволяет разделить логику приложения на три части:

**Модели** Содержат или представляют данные, с которыми работают пользователи. Они могут быть простыми моделями представлений, которые только представляют данные, передаваемые между представлениями и контроллерами; или же они могут быть моделями предметной области, которые содержат бизнес-данные, а также операции, преобразования и правила для манипулирования этими данными.

**Представления** Применяются для визуализации некоторой части модели в виде пользовательского интерфейса.

**Контроллеры** Обрабатывают поступающие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

Этот паттерн разработки нужен для того, чтобы разделить логические части приложения и создавать их отдельно друг от друга. То есть писать независимые блоки кода, которые можно, как угодно, менять, не затрагивая другие.

Общий вид паттерна MVC представлен на рисунке 1.

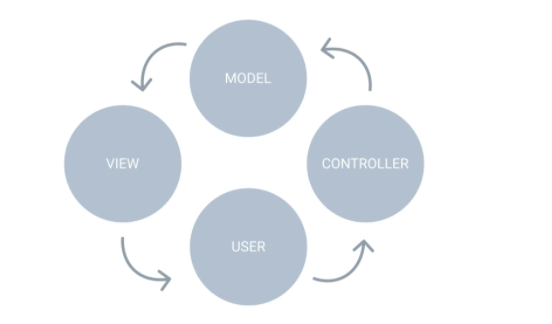


Рисунок 1 – Общий вил шаблона программирования MVC

При работе с веб-сайтом **пользователь** осуществляет ввод данных в формы, таких как: логин, почтовый адрес, номер телефона пароль. А также вводит необходимые данные при отправке комментария: логин, почтовый адрес, а также заголовок и текст комментария.

Данные с формы передаются в **контроллер**,в моей работе он находится в файле index.js:

app.get("/form1", function (request, response) {

response.sendFile(\_\_dirname + "/form/form1.html");

});

app.get("/form2", function (request, response) {

response.sendFile(\_\_dirname + "/form/form2.html");

});

app.get("/form3", function (request, response) {

readMessage();

response.render('\eform3', {dataRes: dataRes});

});

app.post("/form2", urlencodedParser, function (request, response,next) {

if(!request.body) return response.sendStatus(400);

let login=request.body.login;

let mail=request.body.mail;

let header=request.body.header;

let message=request.body.message;

let foto=request.file;

addMessage(login,mail,header,message,foto.filename);

response.sendFile(\_\_dirname + "/form/form2.html");

});

app.post("/form1", urlencodedParser, function (request, response,next) {

if(!request.body) return response.sendStatus(400);

console.log(request.body);

let login=request.body.login;

let mail=request.body.mail;

let number=request.body.number;

let pass=request.body.pass;

addUser(login,mail,number,pass,);

response.send("<script>alert('Hello, "+login+"!'); window.history.back();</script>");

После этого срабатывают вызываемые функции.

Затем **модель** принимает данные, которые передаёт ей контроллер обрабатывает и сохраняет. В моей работе модель представлена в файлах userserver.bd , index.js:

При помощи

function addMessage(login,mail,header,message,foto)

function addUser(login,mail,number,pass)

readMessage()

и соответствующих возвращаемых значений.

После этого **представление**, которое находится в папке views в файле eform3.ejs выводит необходимую информацию c комментариями и соответствующими логинами и аватарами других пользователей на экран пользователя. Ниже представлен код данного представления:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Write a comment from Andrey Trufanov</title>

<link rel="stylesheet" href="/form/style3.css">

</head>

<body>

<h1 title="Comments">User Comments</h1>

<div class="content">

<div class="three\_content">

<div class="form1">

<form action="form1" target="\_self">

<button>Sign Up</button>

</form>

</div>

<div class="form3">

<form action="form2" target="\_self">

<button>Write Comment</button>

</form>

</div>

<div class="butupdate">

<button value="Refresh Page" onClick="window.location.reload();">Update Page</button>

</div>

</div>

<form action="" method="post" name="form3"></form>

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

<div class="commment">

<div class="field">

<div class="span1">

<div class="avatar"> <img src="/form2/<%= dataRes[i].avatar %>" width="150px" height="150px" alt="photo"></div>

<center><span class="nickname"><%= dataRes[i].login %></span></center>

</div>

<div class="span2">

<div class="header"><%= dataRes[i].header %></div>

<div class="comtext"><%= dataRes[i].text %></div>

</div>

</div>

</div>

<% } %>

</form>

</div>

</body>

</html>

В результате данные попадают к **пользователю.**