**ТЕОРИЯ**

HTTP – протокол прикладного уровня, самый используемый.

**Методы НТТР запроса:**

* GET – получение
* POST – создание
* PUT - редактирование
* DELETE – удаление

У НТТР запроса есть заголовки, где мы можем указывать конфигурацию и есть тело запроса. В теле запроса передаются данные. После того как запрос дошел до сервера, сервер может вернуть какой-либо статус код. 2хх – success, 3xx – redirect, 4xx – client error, 5xx – server error.

Самый часто используемый архитектурный подход к построению backend – **REST API.** Это стиль архитектуры ПО для построения распределенных масштабируемых веб-сервисов. REST-сервисы позволяют лучшим образом использовать HTTP- протокол. Формат взаимодействия между клиентом и сервером – JSON.

**Слои абстракции –** основная суть в том, чтобы разделять разны по логике куски кода в разные модули. Таким образом получается слабая связанность кода и мы можем любой из кусков кода переписать без вреда для других.

1. **(DAL – data access layer)** - Слой доступа к данным За этот слой можно принять mongoose или sequilize.
2. **Controller** – работа с клиент-серверной составляющей (запросы). Из контролера возвращаем ответ на клиент и указываем статус-код.
3. **Service** – бизнес-логика. Работа с БД и входными параметрами.

**ПРАКТИКА**

1. Инициализируем проект **npm init –y**
2. Установим express: **npm i express** (это фреймворк для создания серверных приложений, является одним из самых популярных для node js.
3. Установим nodemon для автоматической перезагрузки сервера при каких-либо изменениях в коде (npm i –D nodemon)
4. Напишем скрипт для запуска проекта

"scripts": {

"dev": "nodemon index.js"

},

1. Запускаем сервер

import express from "express";

const PORT = 3000;

const app = express();

app.use(express.json());

app.listen(PORT, () => console.log(`Server started on port ${PORT}`));

1. Устанавливаем mongoose, запускаем БД, подключаемся к ней.

mport express from "express";

import mongoose from "mongoose";

const PORT = 3000;

const DB\_URL = `mongodb+srv://root:1234@cluster0.sojwu.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0`;

const app = express();

app.use(express.json());

app.get("/", (req, res) => {

  res.status(200).json("Hello");

});

const start = async () => {

  try {

    await mongoose.connect(DB\_URL);

    app.listen(PORT, () => console.log(`Server started on port ${PORT}`));

  } catch (error) {

    console.log(error);

  }

};

start();

1. Описываем модель данных для БД, например, пост в файле Post.js

import mongoose from "mongoose";

const Post = new mongoose.Schema({

  author: { type: String, required: true },

  title: { type: String, required: true },

  content: { type: String, required: true },

  picture: { type: String },

});

export default mongoose.model("Post", Post);

1. Записываем какой-нибудь объект в БД, проверяем что все работает.

app.post("/", async (req, res) => {

  try {

    const { author, title, content, picture } = req.body;

    const post = await Post.create({ author, title, content, picture });

    res.status(200).json(post);

  } catch (error) {

    res.status(500).json(error);

  }

});

1. Все эндпоинты писать в app нецелесообразно, поэтому для них создадим отдельный файл router.js и перенесем эндпоинты туда.
2. Логику из эндпоинтов вынесем в отдельный файл PostController.js там пропишем все операции

import Post from "./post.js";

class PostController {

  async getAll(req, res) {

    try {

      const posts = await Post.find();

      res.status(200).json(posts);

    } catch (error) {

      res.status(500).json(error);

    }

  }

  async getOne(req, res) {

    try {

      const { id } = req.params;

      if (!id) {

        res.status(400).json({ message: "Id не указан" });

      }

      const post = await Post.findById(id);

      res.status(200).json(post);

    } catch (error) {

      res.status(500).json(error);

    }

  }

  async create(req, res) {

    try {

      const { author, title, content, picture } = req.body;

      const post = await Post.create({ author, title, content, picture });

      res.status(200).json(post);

    } catch (error) {

      res.status(500).json(error);

    }

  }

  async update(req, res) {

    try {

      const post = req.body;

      if (!post.\_id) {

        res.status(400).json({ message: "Id не указан" });

      }

      const updatedPost = await Post.findByIdAndUpdate(post.\_id, post, {

        new: true,

      });

      res.status(200).json(updatedPost);

    } catch (error) {

      res.status(500).json(error);

    }

  }

  async delete(req, res) {

    try {

      const { id } = req.params;

      if (!id) {

        res.status(400).json({ message: "Id не указан" });

      }

      await Post.findByIdAndDelete(id);

      res.status(200).json({ message: "Пост успешно удален" });

    } catch (error) {

      res.status(500).json(error);

    }

  }

}

export default new PostController();

1. В роутере все эти функции подключим

import Router from "express";

import PostController from "./PostController.js";

const router = new Router();

router.get("/posts", PostController.getAll);

router.get("/posts/:id", PostController.getOne);

router.post("/posts", PostController.create);

router.put("/posts", PostController.update);

router.delete("/posts/:id", PostController.delete);

export default router;