Деструктуризация:

http://localhost:5000/api/user/authorization**?id=5&login=asd\**

code:

const query = req.query  
res.json(query)

result:



---------------------------------------------------------------------------------------------------------

code:

const {id} = req.query  
res.json(id)

result:



1. **Installation**:
2. npm init -y
3. npm install express pg pg-hstore sequelize cors dotenv
   1. express – фреймворк
   2. СУБД postgreSQL – 1 модуль ‘pg’, 2 модуль ‘pg-hstore’
   3. ORM – sequelize
   4. Для обращения из браузера к серверу – cors

CORS (Cross-Origin Resource Sharing) - это механизм, который позволяет веб-страницам получать ресурсы (например, данные) с серверов, расположенных на другом домене, чем тот, на котором находится веб-страница.

Например, представьте себе, что у вас есть веб-страница на домене "example.com", которая хочет получить данные с сервера на домене "api.example.com". Без механизма CORS браузер будет блокировать запрос, и веб-страница не сможет получить данные. Но если сервер на домене "api.example.com" отправит соответствующий заголовок Access-Control-Allow-Origin, то веб-страница сможет получить данные с сервера.

* 1. Чтоб задавать переменное окружение – модуль dotenv

1. Как Dev-зависимость (зависимость для разработки) установим ‘nodemon’, он нужен для того, чтоб при каждом изменение в коде не перезапускать сервер, он перезапускает его автоматически:
   1. npm install -D nodemon
2. В файле package.json напишем скрипт который запускает приложения в режиме разработки:

"scripts": {  
 "dev": "nodemon index.js"  
},

.env – это переменное окружение

db.js – файл где создаём объект “Sequelize” в котором прописываем конфигурацию подключение к БД

1. **Модели данных и связи между ними**

**'**./models/models.js' – в этом описывается все модели (таблицы в БД)

1. **Роутинг и эндпоинты (Каркас приложения)**
2. **'**./routers/**'** Маршруты по которым будут отрабатывать методы
   1. **'**./routers/index.js**'** – файл объединяет остальные роутеры
   2. **'**./routers/category.js**' –!** методудаление доработать/\*--router.delete()--\*/

**В файле '**index.js**' импортируем главный роутер '**./routers/index.js**'**

1. **'**./controllers/**'**
2. **CRUD для типов, брэндов, устройств**

Устанавливаем пакет “express-fileupload” для загрузки картинок

Установим пакет “uuid” – он генерирует случайные неповторяющиеся id-ишки (нужен для файлов)

Цепочка

http://localhost:5000/api/user/authorization

1. **'**index.js**' –** В главном файле задаём URL адрес и вызываем *router*

app.use('/api', *router*)

1. './routers/index.js' – В этом файле собираем все остальные *роутеры,* задаём следующею цепочку URL адреса и вызываем *UserRouter* (и другие.)

const *userRouter* = *require*('./userRouter.js')

*router*.use('/user', *userRouter*)

1. './routers/userRouter.js' – в этом файле (и других файлах) импортируем соответствующий Controller (*userController*), задаём следующею цепочку URL адреса и функцию (без скобок, т.к. мы функцию не вызываем, а передаем как объект)
2. *у контроллера* функцию(и другие.)

const *userController* = *require*('../controllers/userController.js')

*router*.get('/authorization', *userController*.authorizationCheck)

1. './controllers/userController.js' – в этом файле (и других файлах) реализуемфункции(и другие.)

async authorizationCheck(req, res, next){  
 const {id} = req.query  
 if (!id){  
 return next(ApiError.*badRequest*("Не задан ID"))  
 }  
 res.json(id)  
}

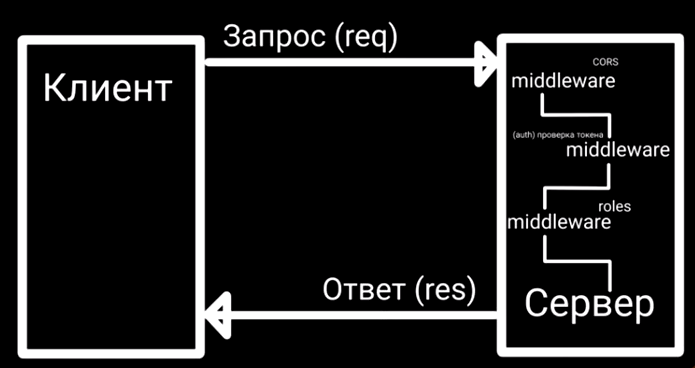
1. **Универсальная обработка ошибок (Handler).**

**'**./error/ApiError.js**'** класс с необходимыми «ошибками»

Можно добавить ещё ошибки

**Первый middleware**

**Middleware –** это некоторое промежуточное звеномежду запросом (от клиента) и функцией ктр этот запрос должен обработать (в сервера), то есть прежде, чем запрос попадёт в функцию, он пройдет по цепочке middleware (их может быть несколько, и они вызываются последовательно одна за другой). Пример, middleware проверка запроса на соответствие CORS, или же допустим функция обрабатывает только авторизованных пользователей (тоже проходить через middleware), если не авторизован, то бросает соответствующую ошибку. Проверка роли пользователя (допустим нектр. функции доступны только для админа), мы проверяем является ли пользователь админом, если да только потом дадим доступ. Только после этой цепочки middleware, запрос доходить до функции и потом ответ на клиент (от сервера).

****

**'**./middleware/ErrorHandlingMiddleware**' -**