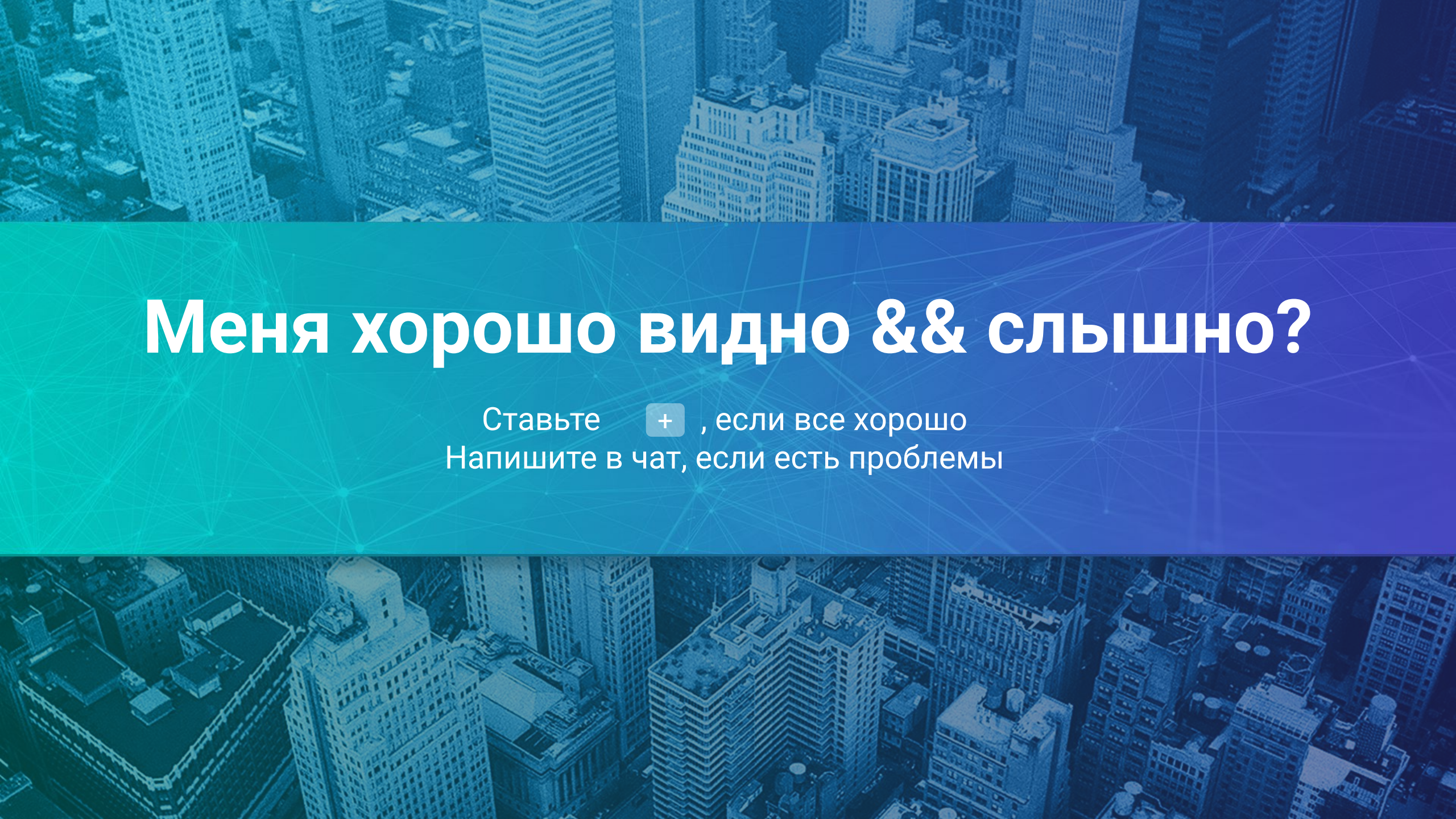
OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование


Меня хорошо видно && слышно?
Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

Проверить, идет ли запись!

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

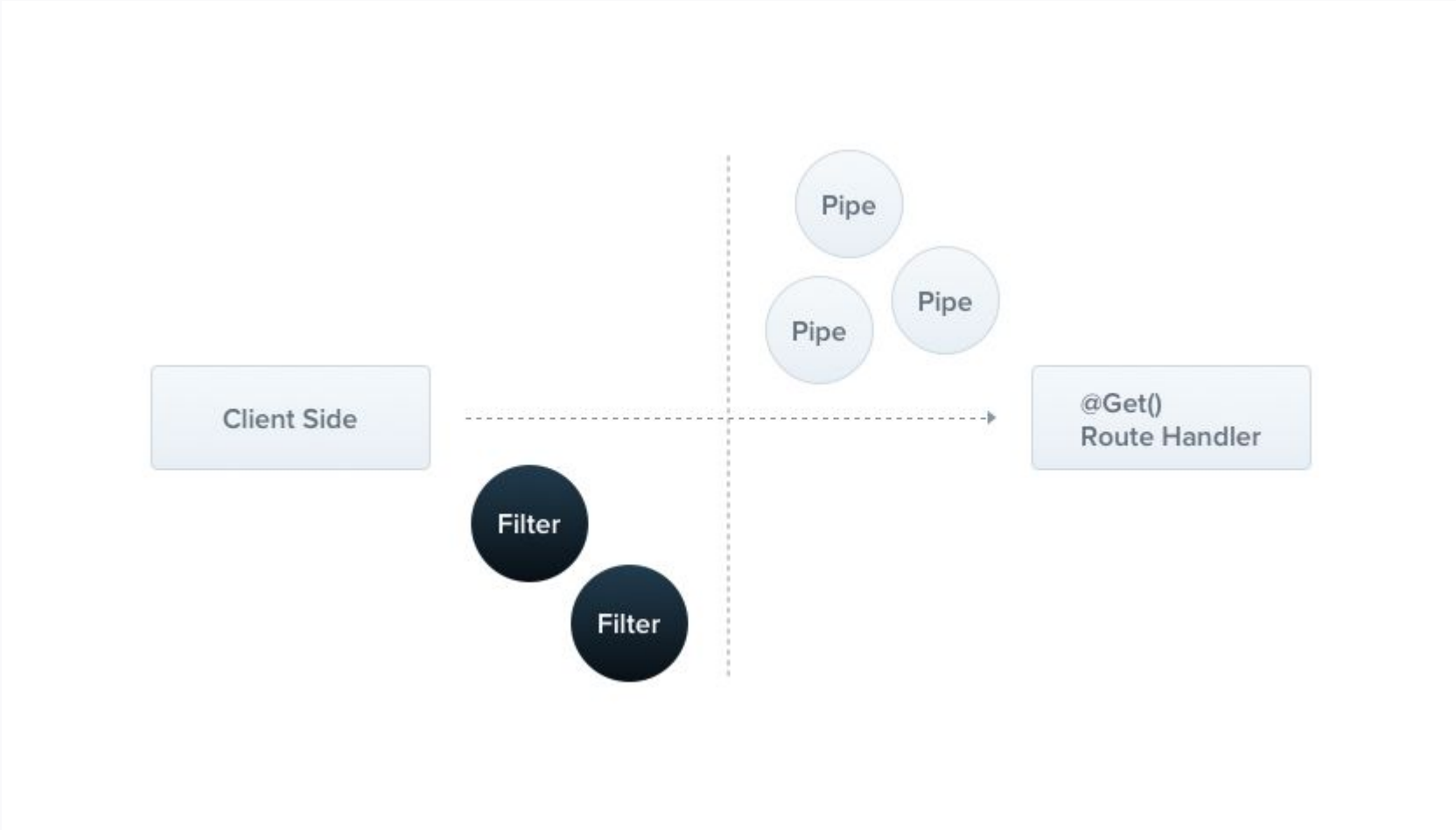
Особенности Nest.js

Николай Лапшин

Маршрут вебинара


Exception filters

Exception filters



Exception filters

Nest имеет встроенный уровень исключений, который отвечает за обработку всех необработанных исключений в приложении.

Когда исключение не обрабатывается кодом вашего приложения, оно перехватывается этим уровнем.

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Фильтры по умолчанию

```
@Controller('exc')
export class ExceptionDemoController {
    @Get('not-handled')
    notChecked(){
        throw new Error('Я ошибка');
    }
}
```

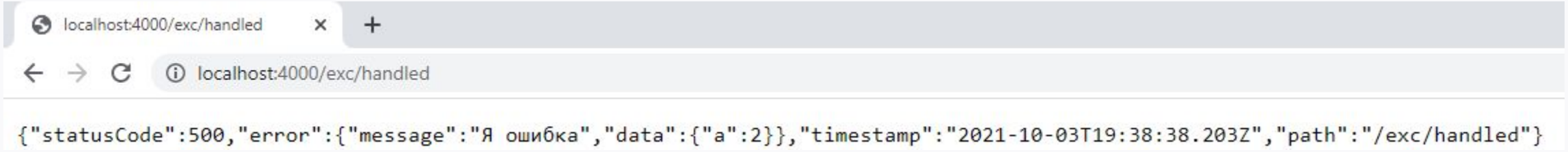
Exception filters

Exception filters: Стандартные исключения

```
@Catch()
export class AllExceptionsFilter implements ExceptionFilter {
    catch(exception: any, host: ArgumentsHost) {
        const ctx = host.switchToHttp();
        const response = ctx.getResponse();
        const request = ctx.getRequest();

        const status =
            exception instanceof HttpException
                ? exception.getStatus()
                : HttpStatus.INTERNAL_SERVER_ERROR;

        response.status(status).json({
            statusCode: status,
            error:exception,
            timestamp: new Date().toISOString(),
            path: request.url,
        });
    }
}
```

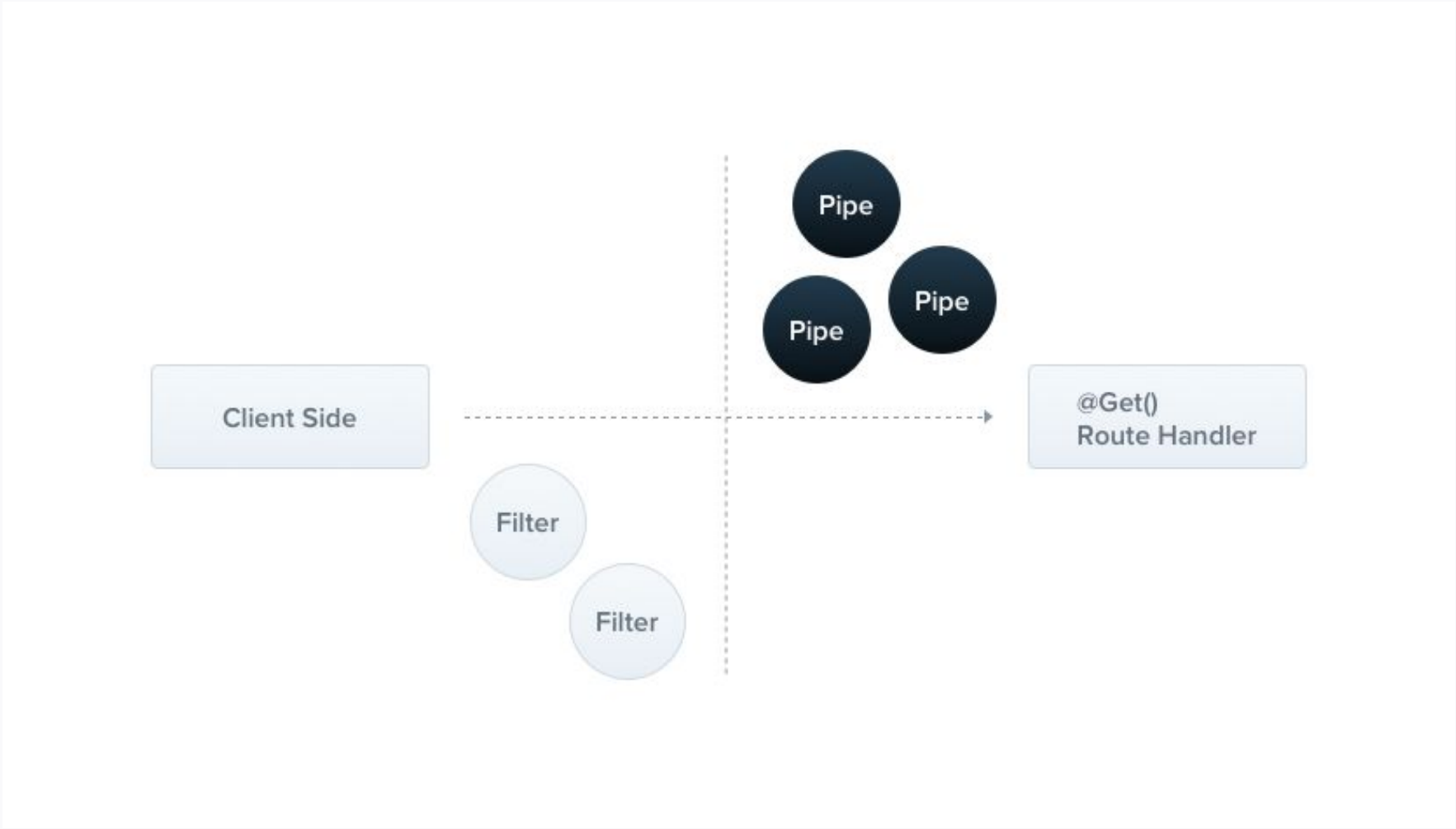
Exception filters: Стандартные исключения

```
@Get('handled')
@UseFilters(new AllExceptionsFilter())
handled(){
    throw { message: 'Я ошибка', data: { a: 2 } };
}
```



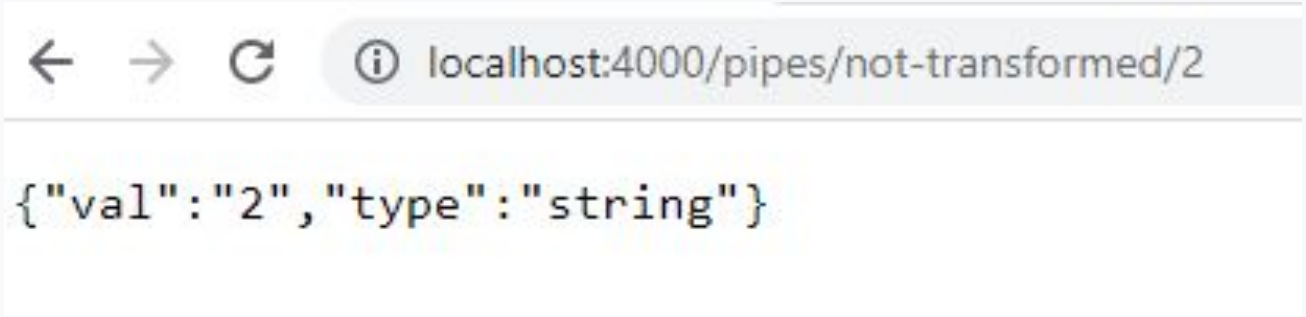

Pipes

Nest Pipes



Nest Pipes

```
@Controller('pipes')
export class PipesController {
    @Get('not-transformed/:num')
    getNotTransformed(@Param('num') num: number) {
        console.log(num);
        return { val: num, type: typeof num };
    }
}
```



Nest Pipes

Используются для преобразования аргументов запросов

Также используются для валидации

- Срабатываю перед вызовом метода
- Аннотируются декоратором **@Injectable()**
- Реализуют интерфейс **PipeTransform**

Nest Pipes

Pipes - класс, аннотированный с помощью декоратора **@Injectable()**

Выполняются перед вызовом метода

Реализуют интерфейс **PipeTransform**

Pipes имеют два типичных варианта использования:

- **Преобразование**
- **Валидация**

Nest Pipes

Pipes из коробки:

1. ValidationPipe
2. ParseIntPipe
3. ParseBoolPipe
4. ParseArrayPipe
5. ParseUUIDPipe
6. DefaultValuePipe

```
@Get(':id')
async findOne(@Param('id', ParseIntPipe) id: number) {
  return this.catsService.findOne(id);
}
```

```
@Get()
async find(@Query('id', ParseArrayPipe) id: string[]) {
  return this.catsService.find({ id });
}
```

Nest Pipes

Каждый **pipe** должен реализовывать метод **transform**

Метод **имеет два параметра:** входящее значение (***value***) и данные запроса (***metadata***)

```
1   import { PipeTransform, Injectable, ArgumentMetadata } from '@nestjs/common'
2
3   @Injectable()
4   export default class CustomQueryParseIntArrayPipe implements PipeTransform {
5     transform(value: string, metadata: ArgumentMetadata): any {
6       if (metadata.type === 'query') {
7         return value.split(',').map(v => +v)
8       }
9       return value
10    }
11  }
```


Guards

Guards


Client Side
HTTP Request
Guard
Route Handler
@RequestMapping

Guards

Guard - это класс, определяющий, доступен ли запрос для выполнения по условиям

- аннотируется **@Injectable()**
- должен реализовать интерфейс **CanActivate**
- выполняется после middleware, но перед любым перехватчиком (Interceptor) или Pipe

Guards: Authorization guard

```
import { Injectable, CanActivate, ExecutionContext } from '@nestjs/common';
import { Observable } from 'rxjs';

@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    const request = context.switchToHttp().getRequest();
    return validateRequest(request);
  }
}
```


Interceptors

Interceptors

Interceptors

Interceptor (Перехватчик) - это класс, аннотированный с помощью декоратора @Injectable()

Перехватчики должны реализовывать интерфейс ***NestInterceptor***.

Interceptors

Interceptor (перехватчик) – класс, аннотированный @Injectable()

Реализует интерфейс **NestInterceptor**

Позволяют

- **привязать** дополнительную логику до или после вызова метода
- **преобразовать** результат, возвращаемый функцией
- **преобразовать** исключение, выброшенное из функции
- **расширить** основные функции поведения
- полностью **переопределить** функцию в зависимости от конкретных условий

Interceptors: Создание перехватчика

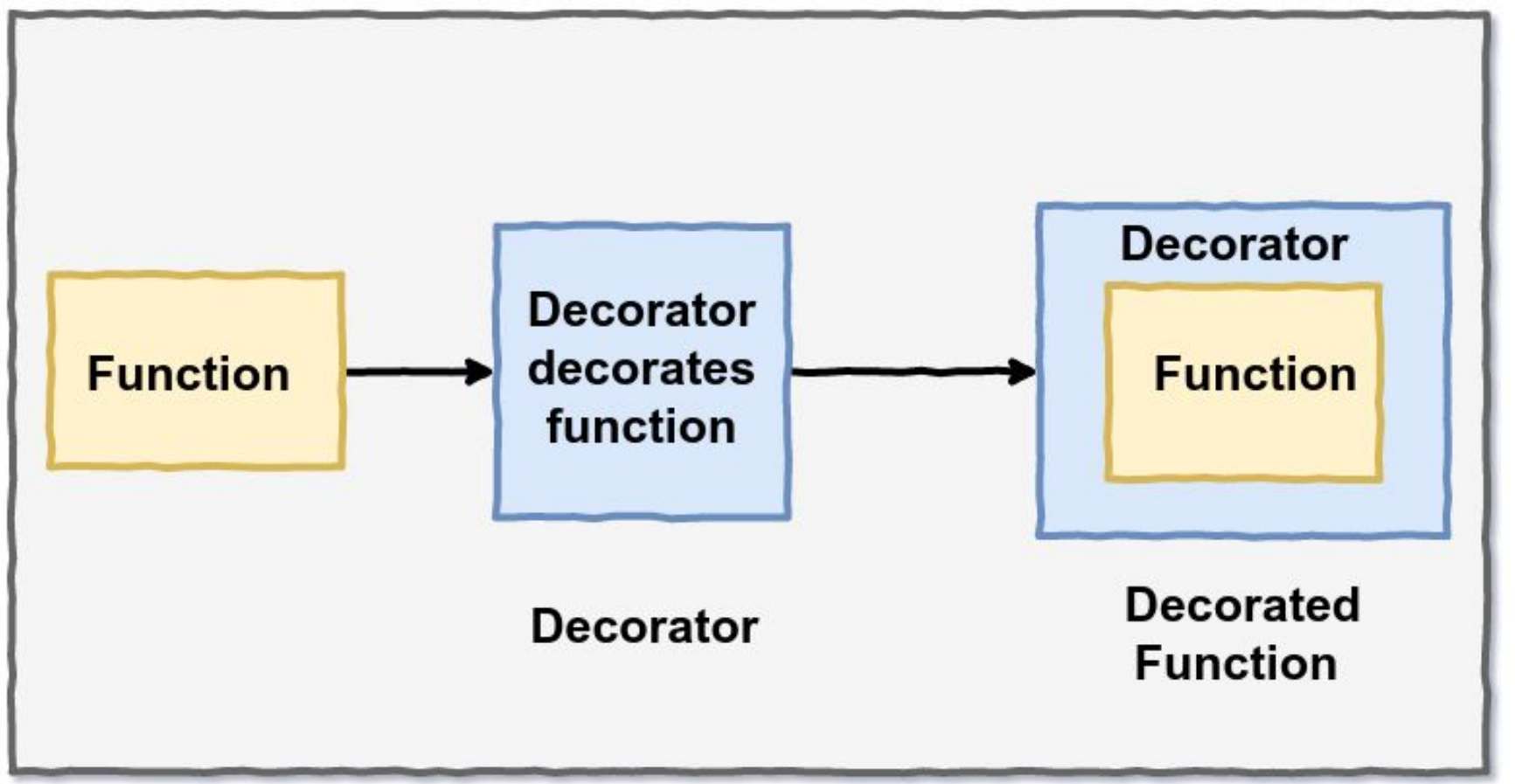
```ts
import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common'
import { Observable } from 'rxjs'
import { tap } from 'rxjs/operators'

@Injectable()
export class LoggingInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
    console.log('Before...')

    const now = Date.now()
    return next.handle().pipe(tap(() => console.log(`After... ${Date.now() - now}ms`)))
  }
}
```

Custom decorators

Custom decorators

Что такое декоратор?

Custom decorators



Custom decorators

Nest использует концепцию **декораторов**.

Декораторы в JavaScript на текущий момент находятся в Stage 2 (черновик)

Декораторы могут быть определены для **класса**, **метода** или **свойства**.

Custom decorators

Как мне работать с декораторами в других проектах?

Javascript: Для работы с декораторами необходим babel версии ^7.1.0 + плагин @babel/plugin-proposal-decorators

Typescript: В файле tsconfig.json в свойство compilerOptions выставить флаг **"experimentalDecorators": true**

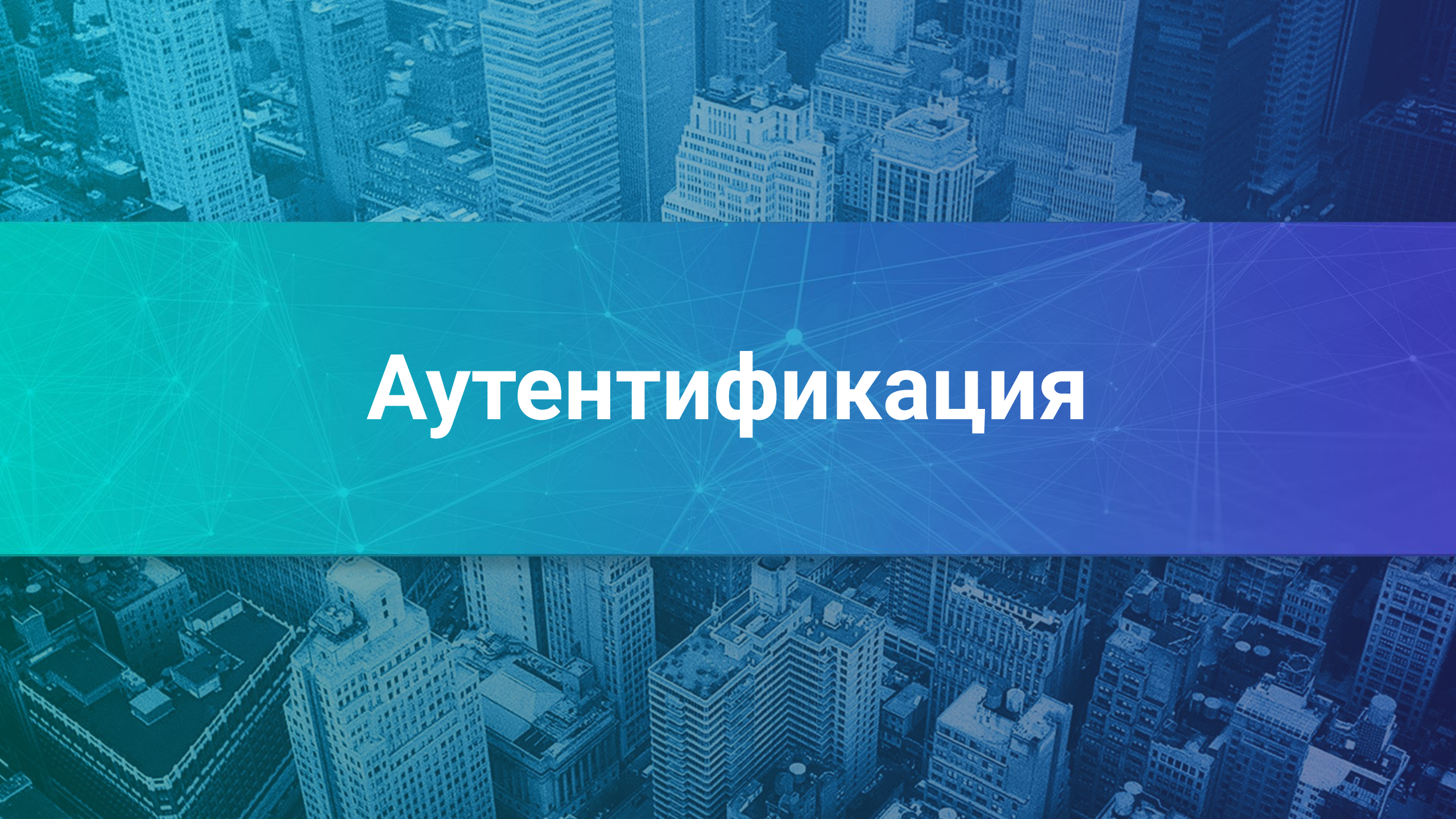
Custom decorators

В Nest по умолчанию входит стандартный набор декораторов параметров (из пакета @nestjs/common)

@Request()	req	@Param(param?: string)	req.params / req.params[param]
@Response()	res	@Body(param?: string)	req.body / req.body[param]
@Next()	next	@Query(param?: string)	req.query / req.query[param]
@Session()	req.session	@Headers(param?: string)	req.headers / req.headers[param]

Custom decorators: Создание декоратора

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common'

export const Authorization = createParamDecorator((data: unknown, ctx: ExecutionContext) => {
  const request = ctx.switchToHttp().getRequest()
  const token = request.headers.authorization || null
  return token
})
```

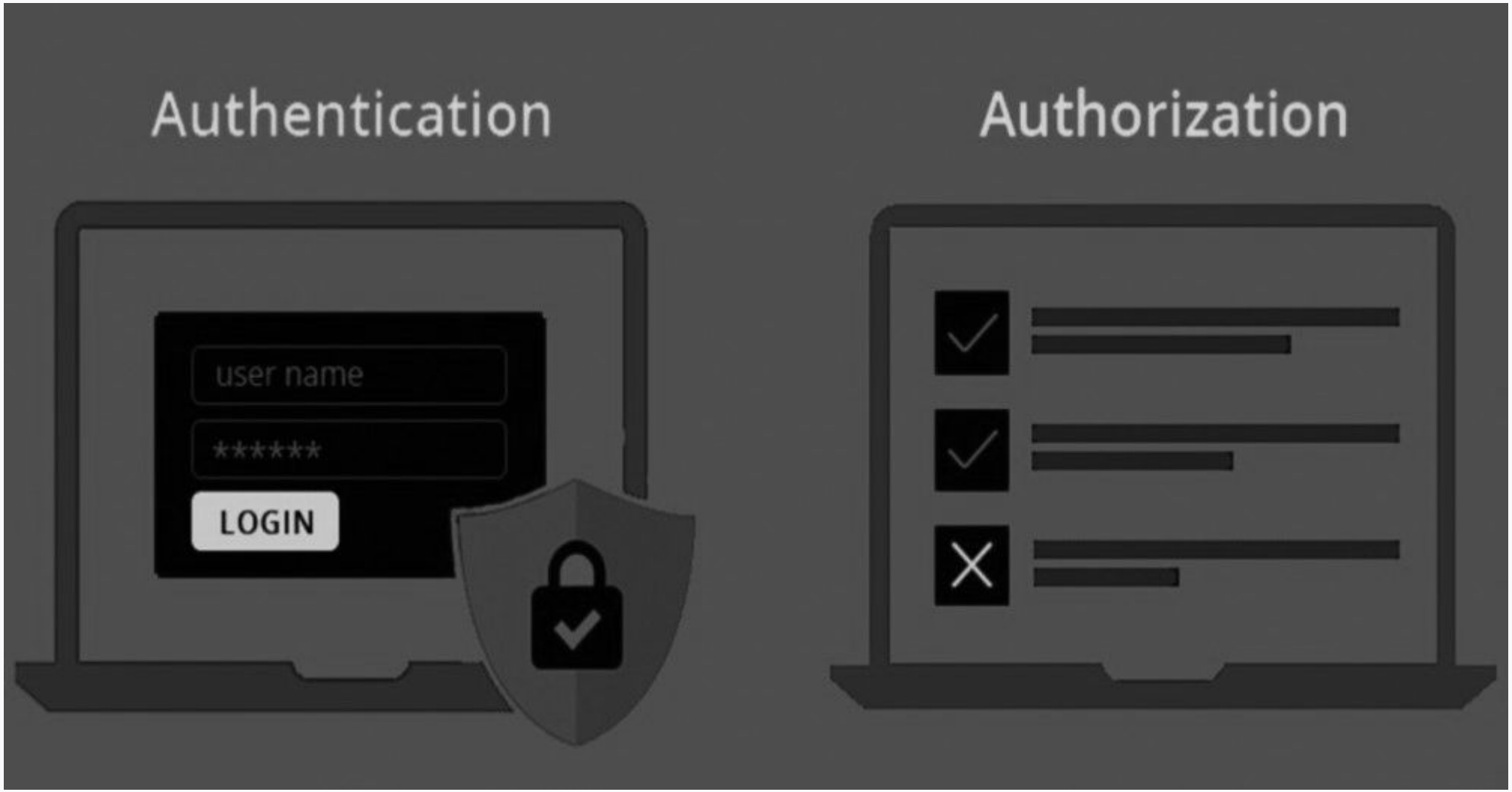


Аутентификация

Аутентификация

В чем различие аутентификации против авторизации?

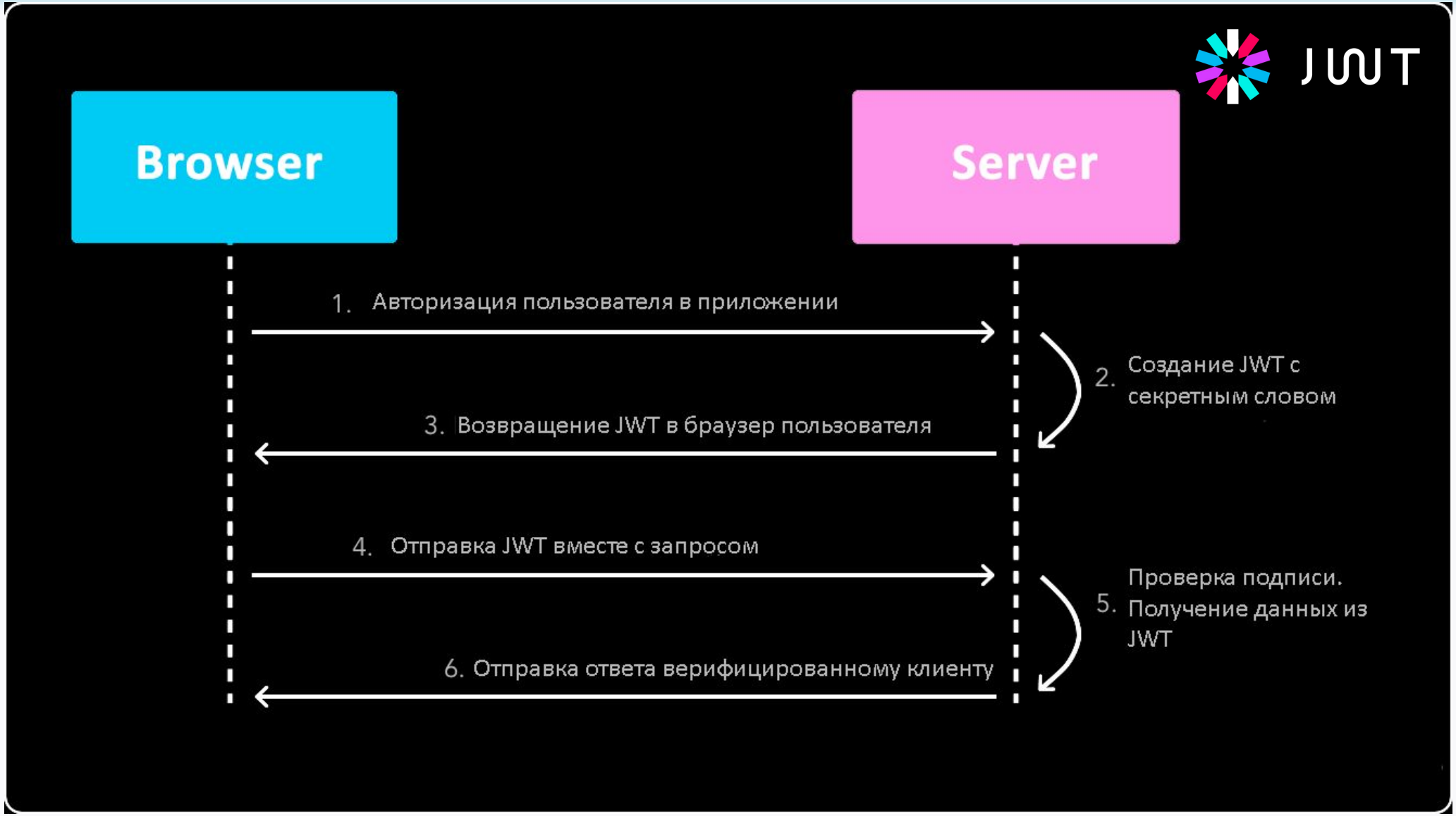
Аутентификация



Аутентификация: Passport

Passport - самая популярная библиотека аутентификации node.js, хорошо известная сообществу и успешно используемая во многих производственных приложениях.

Аутентификация: JWT



Аутентификация: Установка Passport

Passport предоставляет стратегию, называемую локальным паспортом, которая реализует механизм аутентификации по имени пользователя и паролю

```
$ npm install --save @nestjs/passport passport passport-local
$ npm install --save-dev @types/passport-local
```

Итоги - тезисы

1. Познакомились с основными механизмами работы с запросами в nest.js: exception filters, interceptors

2. Изучили, как преобразовывать параметры запроса при помощи pipes

3. Рассмотрели на практике как работает авторизация/аутентификаци в nestjs

Список материалов для изучения

1. Nest Pipes
2. Class validator
3. Passport JS
4. Custom Decorators
5. Proposal-decorators
6. Про токены, JSON Web Tokens (JWT), аутентификацию и авторизацию. Token-Based Authentication

Заполните, пожалуйста,
опрос о занятии по ссылке в чате