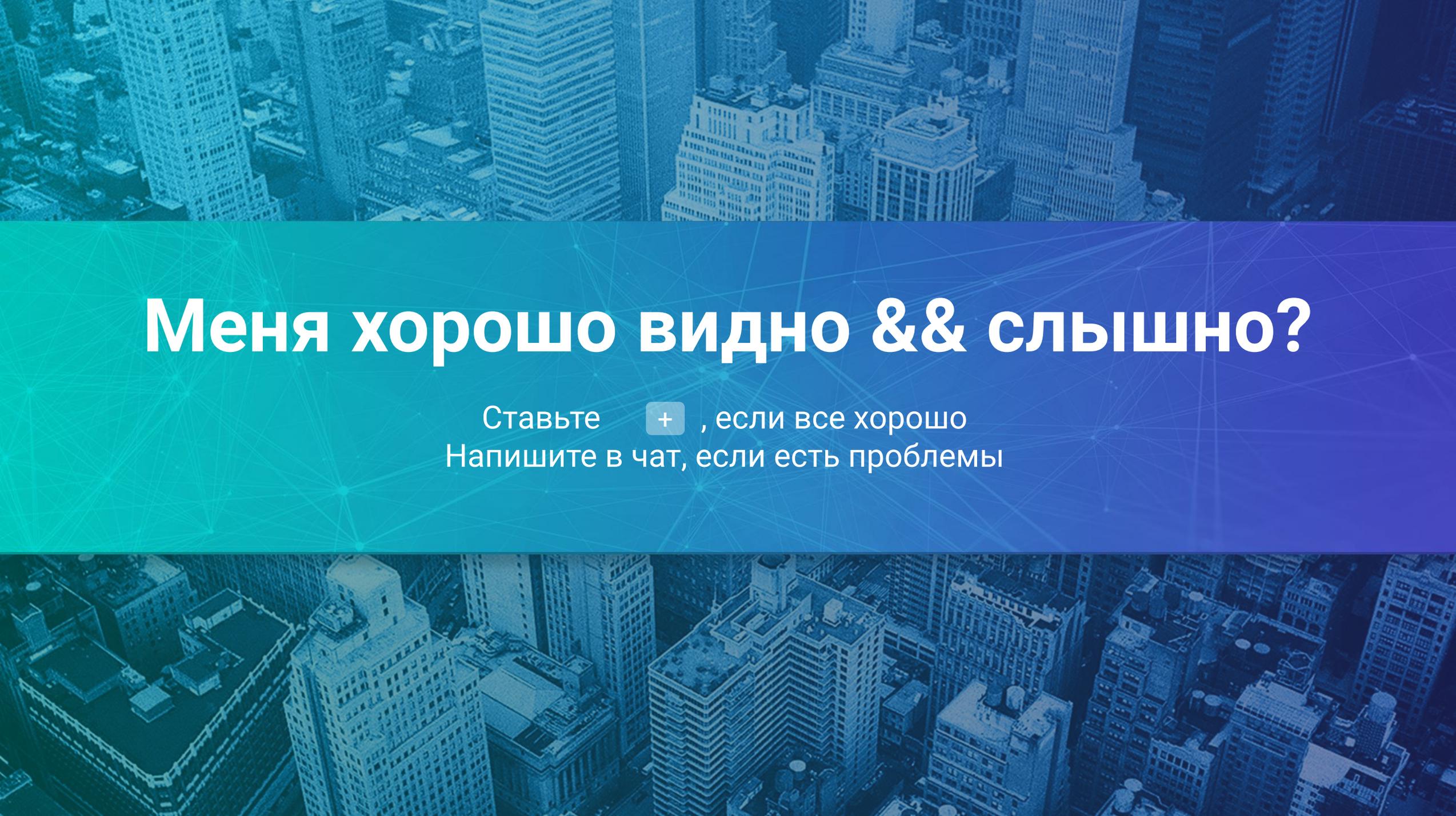




OTUS

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

The background of the slide is a blue-tinted aerial photograph of a city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue band with a white network pattern of dots and lines is overlaid across the middle of the image.

# Особенности Nest.js

---

Николай Лапшин

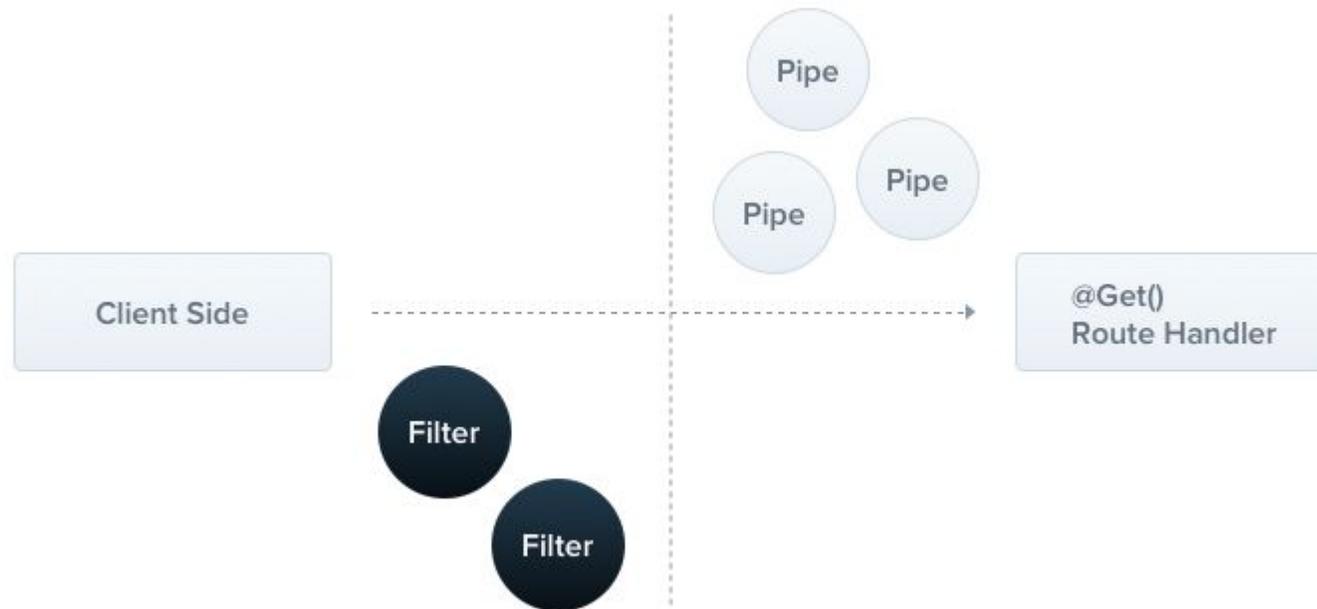
# Маршрут вебинара



The image features a blue-tinted aerial view of a city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue band with a white network pattern of dots and lines runs horizontally across the middle of the image. The text "Exception filters" is centered within this band in a white, bold, sans-serif font.

# 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');
  }
}
```



A screenshot of a web browser window. The address bar shows the URL `localhost:4000/exc/not-handled`. The main content area of the browser displays a JSON response: `{"statusCode":500,"message":"Internal server error"}`. The browser interface includes back, forward, and refresh buttons.

# Exception filters



# Exception filters: Стандартные исключения

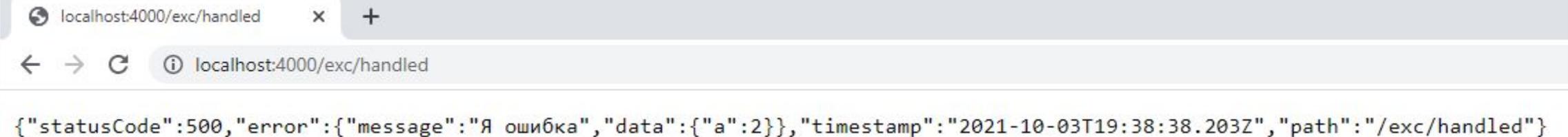
```
@Catch()
export class AllExceptionsFilter implements ExceptionFilter {
  catch(exception: any, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse();
    const request = ctx.getRequest();

    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 } };
}
```



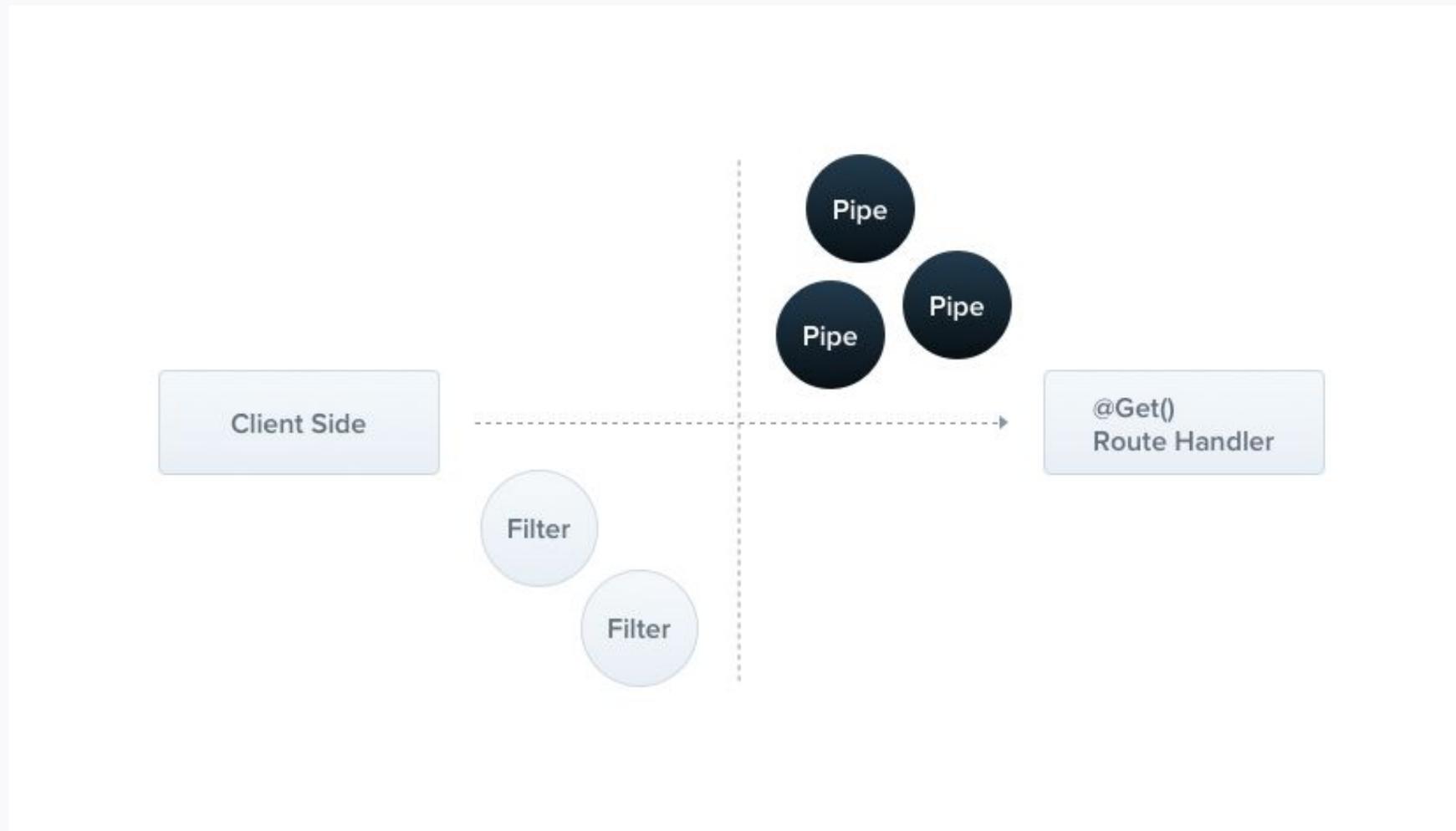
The screenshot shows a web browser window with the address bar displaying 'localhost:4000/exc/handled'. The page content shows a JSON error response: {"statusCode":500,"error":{"message":"Я ошибка","data":{"a":2}},"timestamp":"2021-10-03T19:38:38.203Z","path":"/exc/handled"}. The browser interface includes a tab, navigation buttons, and an information icon.

```
{"statusCode":500,"error":{"message":"Я ошибка","data":{"a":2}},"timestamp":"2021-10-03T19:38:38.203Z","path":"/exc/handled"}
```

The image features a central horizontal band with a blue-to-purple gradient. Overlaid on this band is a network of white lines connecting various points, resembling a data or communication network. The word "Pipes" is written in a large, white, sans-serif font, centered within this network. The background of the entire image is an aerial view of a city skyline, with numerous skyscrapers and buildings, all rendered in a monochromatic blue color scheme.

# 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 };
  }
}
```

← → ↻ ⓘ localhost:4000/pipes/not-transformed/2

```
{"val": "2", "type": "string"}
```

# Nest Pipes

Используются для преобразования аргументов запросов

Также используются для валидации

- Срабатывают 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



# 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

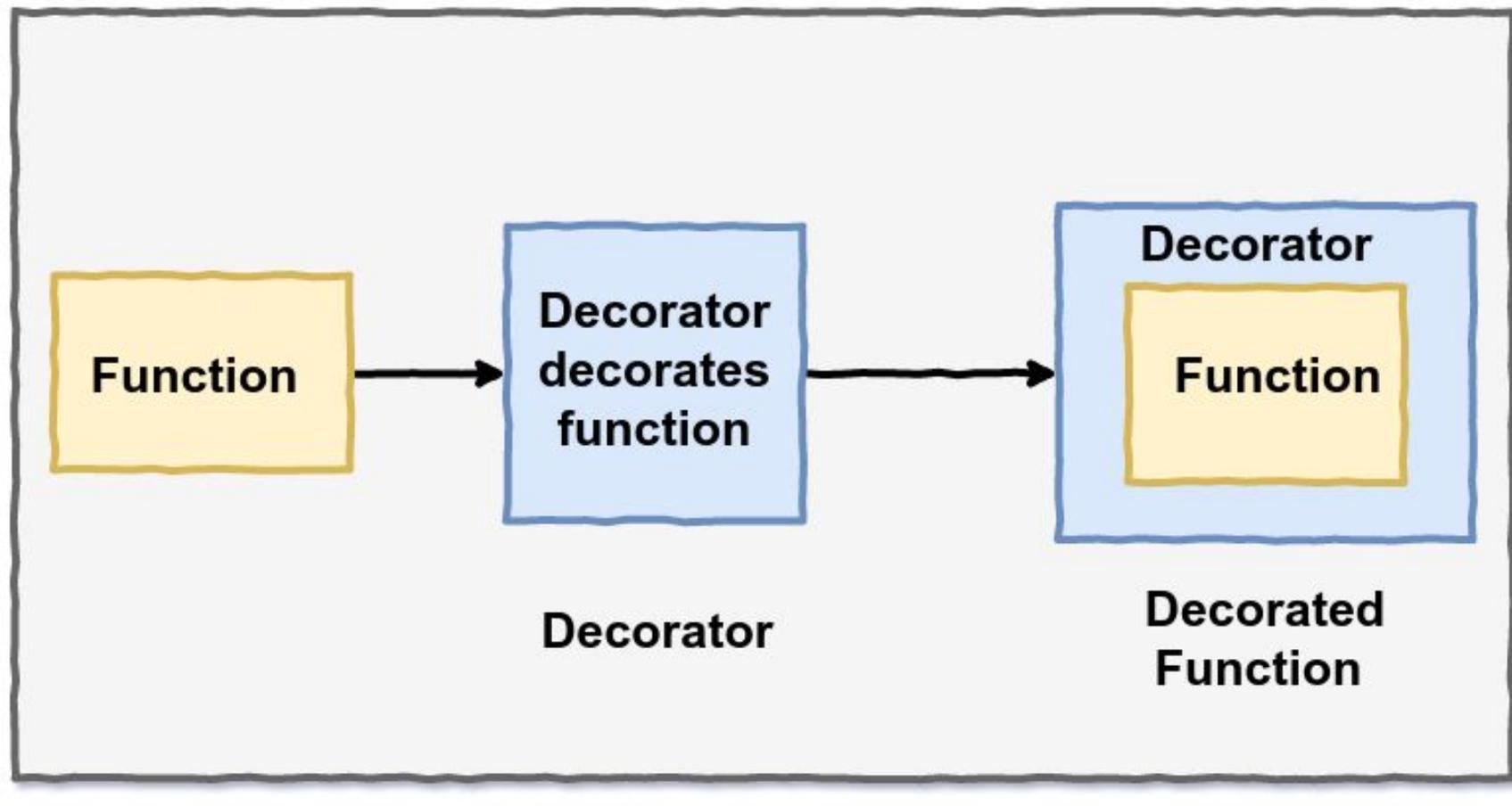
```
1 import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common'
2 import { Observable } from 'rxjs'
3 import { tap } from 'rxjs/operators'
4
5 @Injectable()
6 export class LoggingInterceptor implements NestInterceptor {
7   intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
8     console.log('Before...')
9
10    const now = Date.now()
11    return next.handle().pipe(tap(() => console.log(`After... ${Date.now() - now}ms`)))
12  }
13 }
```

The image features a central horizontal band with a blue-to-purple gradient background. Overlaid on this band is a white network pattern of interconnected nodes and lines. The text "Custom decorators" is centered in this band in a large, white, sans-serif font. The top and bottom portions of the image show an aerial view of a city skyline, with numerous skyscrapers and buildings, all rendered in a monochromatic blue color scheme.

# 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`

`@Response()`

`res`

`@Next()`

`next`

`@Session()`

`req.session`

`@Param(param?: string)`

`req.params / req.params[param]`

`@Body(param?: string)`

`req.body / req.body[param]`

`@Query(param?: string)`

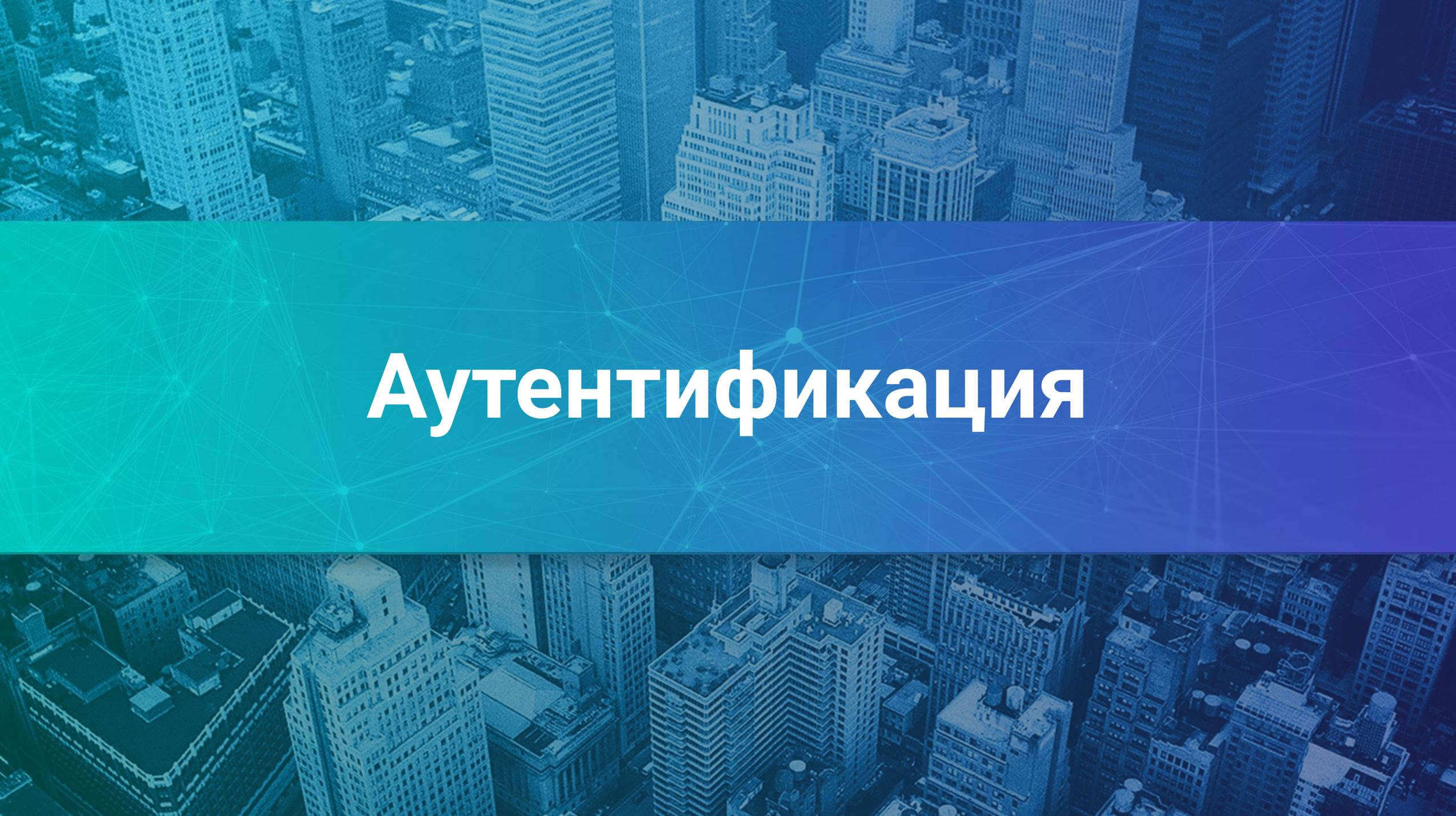
`req.query / req.query[param]`

`@Headers(param?: string)`

`req.headers / req.headers[param]`

# Custom decorators: Создание декоратора

```
1 import { createParamDecorator, ExecutionContext } from '@nestjs/common'
2
3 export const Authorization = createParamDecorator((data: unknown, ctx: ExecutionContext) => {
4   const request = ctx.switchToHttp().getRequest()
5   const token = request.headers.authorization || null
6   return token
7 })
```

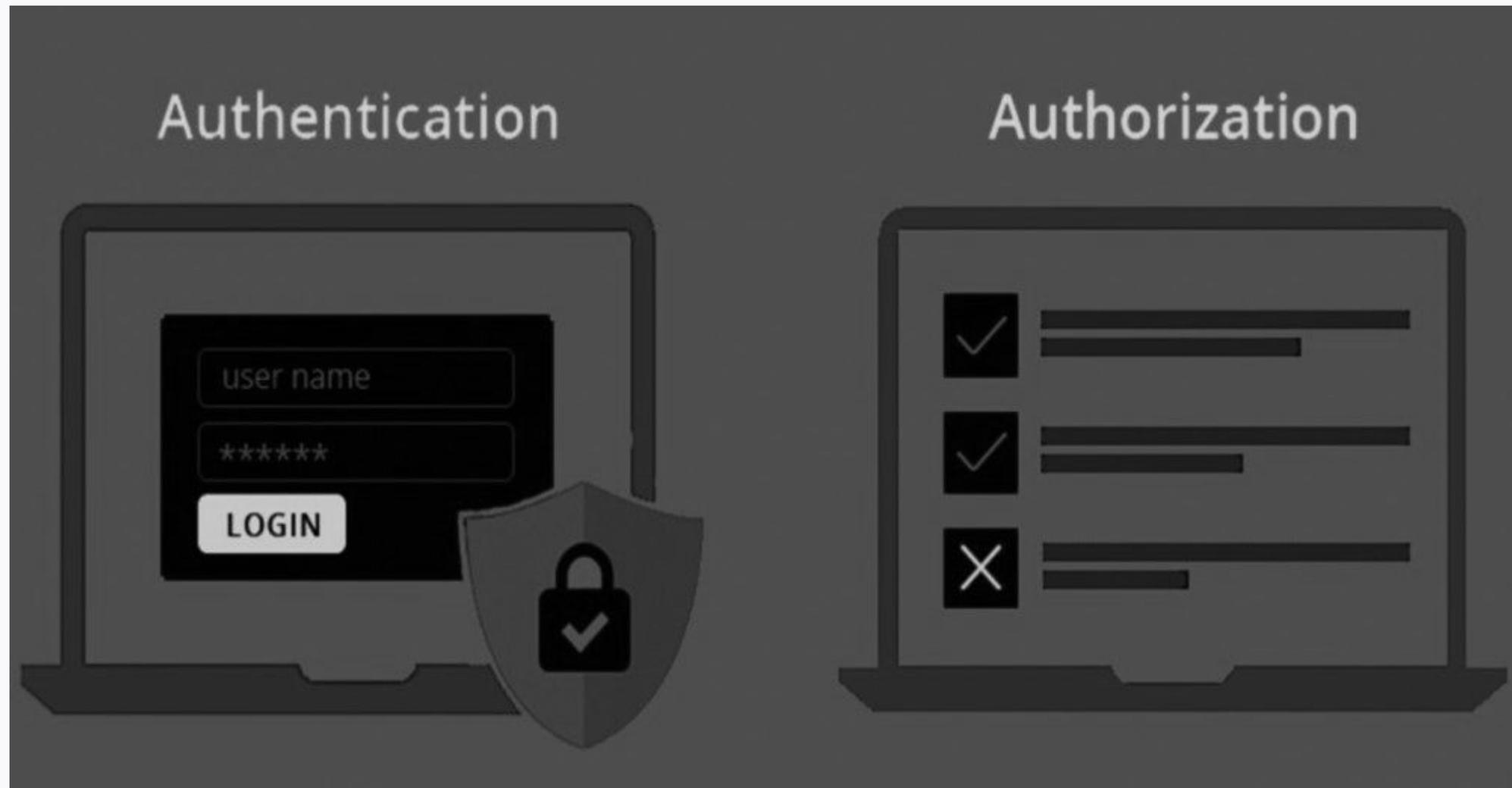
The image features a central horizontal band with a blue-to-purple gradient. Overlaid on this band is a white network pattern of interconnected lines and dots. The background of the entire image is an aerial view of a city skyline, rendered in a monochromatic blue color. The word "Аутентификация" is written in a large, bold, white sans-serif font across the center of the network pattern.

# Аутентификация

# Аутентификация

**В чем различие аутентификации против авторизации?**

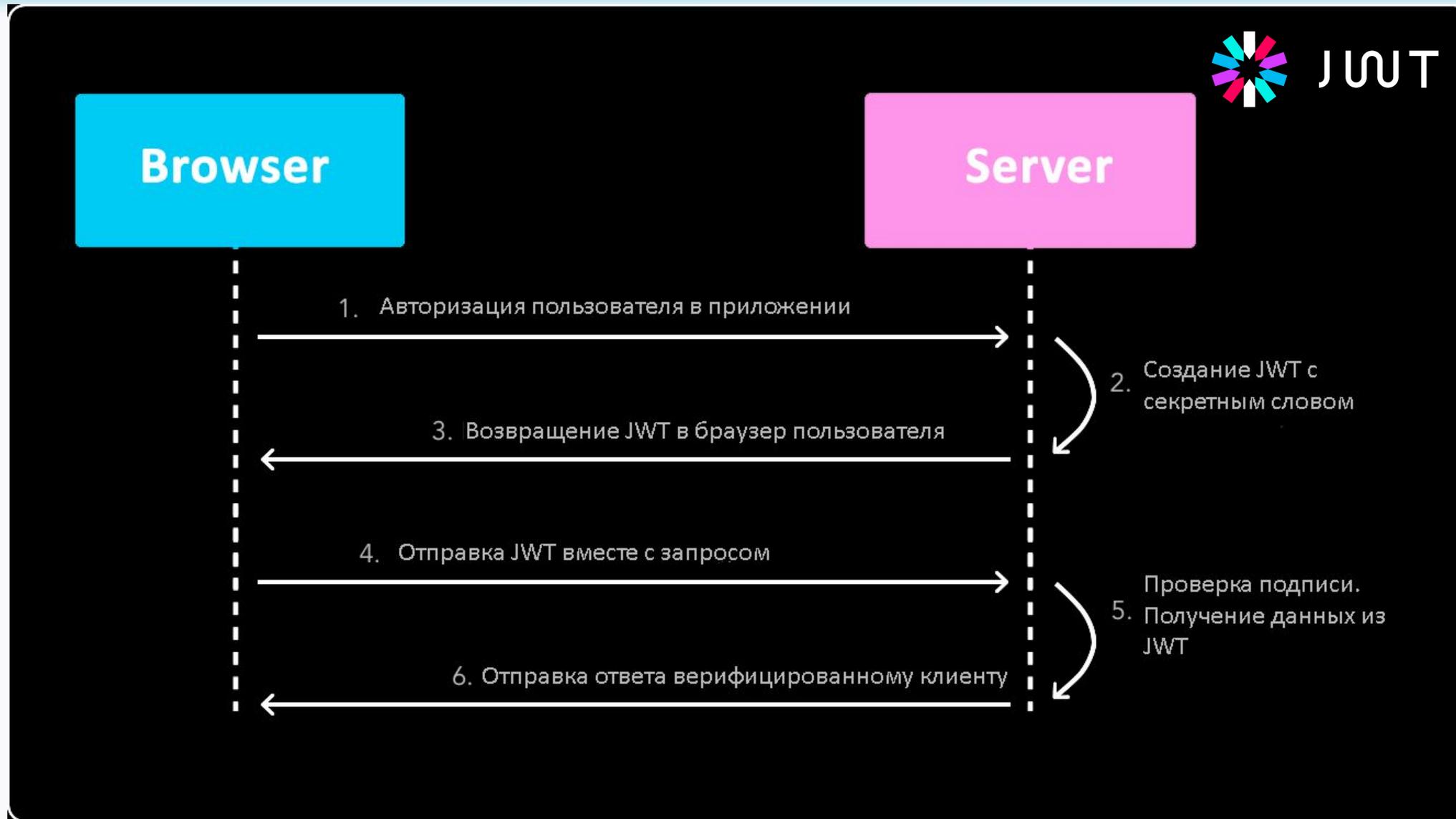
# Аутентификация



# 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

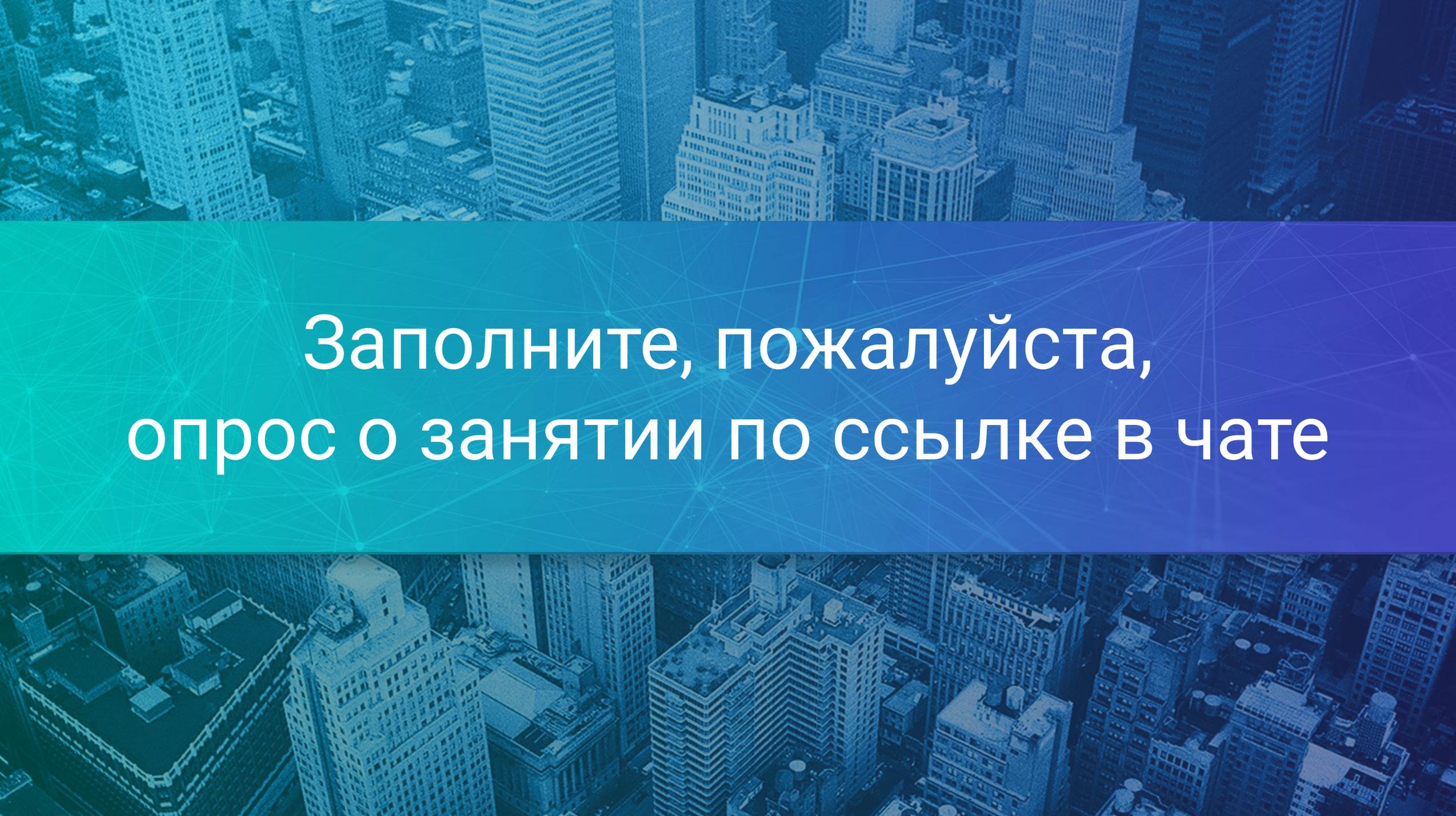
2 Изучили, как преобразовывать параметры запроса при помощи pipes

3 Рассмотрели на практике как работает авторизация/аутентификация  
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](#)

An aerial view of a city skyline, likely New York City, with a blue overlay and a network pattern of white lines and dots. The text is centered in the middle of the image.

Заполните, пожалуйста,  
опрос о занятии по ссылке в чате