Serialization

Serializers provides clean abstraction layer towards **data manipulation** before sending the actual response. For instance, sensitive data like user's password should be always excluded from the eventual response. Furthermore, certain properties might require additional transformation, let's say, we don't want to send the whole database entity. Instead, we would like to pick only `id` and `name`. The rest should be automatically stripped. Unluckily, manually mapping all entities may bring a lot of confusion.

Overview

In order to provide a straightforward way to carry out these operations, Nest comes with the `ClassSerializerInterceptor` class. It uses **class-transformer** package to provide a declarative and extensible way of transforming objects. Basically, the `ClassSerializerInterceptor` takes the value returned from the method and call `classToPlain()` function from **class-transformer** package.

Exclude properties

Let's assume that we want to automatically exclude a `password` property from the following entity:

```
import { Exclude } from 'class-transformer';

export class UserEntity {
  id: number;
  firstName: string;
  lastName: string;

  @Exclude()
  password: string;

  constructor(partial: Partial<UserEntity>) {
    Object.assign(this, partial);
  }
}
```

Then, return the instance of this class directly from the controller's method.

```
@UseInterceptors(ClassSerializerInterceptor)
@Get()
findOne(): UserEntity {
  return new UserEntity({
```

```
    id: 1,
    firstName: 'Kamil',
    lastName: 'Mysliwiec',
    password: 'password',
  });
}
```

> **HINT**
> The `ClassSerializerInterceptor` is imported from `@nestjs/common` package.

Now, when you call this endpoint, you'll receive a following response:

```
{
  "id": 1,
  "firstName": "Kamil",
  "lastName": "Mysliwiec"
}
```

Expose properties

If you want to expose earlier precalculated property, simply use `@Expose()` decorator.

```
@Expose()
get fullName(): string {
  return `${this.firstName} ${this.lastName}`;
}
```

Transform

You can perform additional data transformation using `@Transform()` decorator. For example, you want to pick a name of the `RoleEntity` instead of returning the whole object.

```
@Transform(role => role.name)
role: RoleEntity;
```

Pass options

The transform options may vary depending on the certain factors. In order to override default settings, use `@SerializeOptions()` decorator.

```
@SerializeOptions({
  excludePrefixes: ['_'],
})
@Get()
findOne(): UserEntity {
  return {};
}
```

> **HINT**
> The `@SerializeOptions()` decorator is imported from `@nestjs/common` package.

These options will be passed as a second argument of the `classToPlain()` function.

Websockets & Microservices

All these guidelines concern both WebSockets as well as microservices, regardless of transport method that is being used.

Learn more

In order to read more about available decorators, options, visit this page.

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more here.

Principal Sponsor

Sponsors / Partners

Become a sponsor