Mapped types

As you build out features like **CRUD** (Create/Read/Update/Delete) it's often useful to construct variants on a base entity type. Nest provides several utility functions that perform type transformations to make this task more convenient.

Partial

When building input validation types (also called DTOs), it's often useful to build **create** and **update** variations on the same type. For example, the **create** variant may require all fields, while the **update** variant may make all fields optional.

Nest provides the `PartialType()` utility function to make this task easier and minimize boilerplate.

The `PartialType()` function returns a type (class) with all the properties of the input type set to optional. For example, suppose we have a **create** type as follows:

```
import { ApiProperty } from '@nestjs/swagger';

export class CreateCatDto {
  @ApiProperty()
  name: string;

  @ApiProperty()
  age: number;

  @ApiProperty()
  breed: string;
}
```

By default, all of these fields are required. To create a type with the same fields, but with each one optional, use `PartialType()` passing the class reference (`CreateCatDto`) as an argument:

```
export class UpdateCatDto extends PartialType(CreateCatDto) {}
```

> info **Hint** The `PartialType()` function is imported from the `@nestjs/swagger` package.

Pick

The `PickType()` function constructs a new type (class) by picking a set of properties from an input type. For example, suppose we start with a type like:

```
import { ApiProperty } from '@nestjs/swagger';

export class CreateCatDto {
  @ApiProperty()
```

```
  name: string;

  @ApiProperty()
  age: number;

  @ApiProperty()
  breed: string;
}
```

We can pick a set of properties from this class using the `PickType()` utility function:

```
export class UpdateCatAgeDto extends PickType(CreateCatDto, ['age'] as
const) {}
```

> info **Hint** The `PickType()` function is imported from the `@nestjs/swagger` package.

Omit

The `OmitType()` function constructs a type by picking all properties from an input type and then removing a particular set of keys. For example, suppose we start with a type like:

```
import { ApiProperty } from '@nestjs/swagger';

export class CreateCatDto {
  @ApiProperty()
  name: string;

  @ApiProperty()
  age: number;

  @ApiProperty()
  breed: string;
}
```

We can generate a derived type that has every property **except** `name` as shown below. In this construct, the second argument to `OmitType` is an array of property names.

```
export class UpdateCatDto extends OmitType(CreateCatDto, ['name'] as
const) {}
```

> info **Hint** The `OmitType()` function is imported from the `@nestjs/swagger` package.

Intersection

The `IntersectionType()` function combines two types into one new type (class). For example, suppose we start with two types like:

```
import { ApiProperty } from '@nestjs/swagger';

export class CreateCatDto {
  @ApiProperty()
  name: string;

  @ApiProperty()
  breed: string;
}

export class AdditionalCatInfo {
  @ApiProperty()
  color: string;
}
```

We can generate a new type that combines all properties in both types.

```
export class UpdateCatDto extends IntersectionType(
  CreateCatDto,
  AdditionalCatInfo,
) {}
```

> info **Hint** The `IntersectionType()` function is imported from the `@nestjs/swagger` package.

Composition

The type mapping utility functions are composable. For example, the following will produce a type (class) that has all of the properties of the `CreateCatDto` type except for `name`, and those properties will be set to optional:

```
export class UpdateCatDto extends PartialType(
  OmitType(CreateCatDto, ['name'] as const),
) {}
```