```
        public readonly ?int $id,
        public readonly string $email,
        public readonly string $first_name,
        public readonly ?string $last_name,
        /** @var DataCollection<TagData> */
        public readonly ?DataCollection $tags,
        public readonly ?FormData $form,
    ) {}
}
```

All DTO will look similar to this one:

- The ID is always optional because there's no ID when a POST request comes in, and we transform it into a DTO. But when a PUT request comes in, there's an ID. And remember, this DTO will also be used when querying the subscribers and returning a response.
- Properties are in `$snake_case` format. By default, laravel-data will map the request or the model attributes with the DTO's properties. By using `snake_case` variables, there's no extra work to do. This also applies to plain DTO classes (when you're not using the laravel-data package). If you want your properties to be in `camelCase` you have to write the logic that transforms the `model_attributes` to `dtoAttributes`. I have done it in the past, but after a while, it gets messy. So nowadays, I'm using `snake_case` everywhere:
  - Models
  - DTOs
  - Request parameters
- `$form` is a nested property. As you can see, there's a `FormData` class that is a nested property of the `SubscriberData`. Just as the `Subscriber` model has a `Form` attribute. laravel-data helps us make this nesting very easy. We'll talk about it in more detail later.
- `$tags` is also a nested property, but a subscriber has many tags, and as you can see in DTOs, we can use the `DataCollection` class to do this mapping. It comes from the laravel-data package.