You can see that I run some DB queries to get the tags and the form. So instead of IDs (that come from the request), I have models which can map to `TagData` or `FormData`. Later I'll explain these things in more detail.

And look at the controller:

```php
class CreateSubscriberController extends Controller
{
  public function __invoke(SubscriberData $data):
SubscriberData
  {
    return SubscriberData::from(
      CreateSubscriberAction::execute($data)
    );
  }
}
```

You can inject any Data class, and transformation from the request will happen automatically! And as you can see, a Data object can be returned from a controller action and will be cast to JSON (including the nested properties).

In my opinion, it's a fantastic tool to have, so I'll use it heavily in the demo application later.

One more question to end up this chapter: what's the difference between value objects and DTOs?

- A DTO has an ID because it represents a model.
- A value object never has an ID. It represents a value, not an entity.

That's the main difference. However, in the Laravel community, I often see people mixing up the two concepts. You can even see people writing only value objects, and they use it as a DTO and as a VO.