Basic Concepts

Domain-Driven Design

First of all, we have to answer the most obvious question: what is Domain-Driven Design.

DDD is a software development approach that tries to bring the business language and the code as close together as possible. This is the most critical attribute of this approach. But for some reason, DDD is one of the most misunderstood and overcomplicated topics in the developer community, so I'll try to make it easy to understand.

DDD teaches us two main things:

- Strategic Design
- Technical Design

In my honest opinion, strategic design is way more important than the technical aspects. It's hard to summarize it in one cool sentence, but you will see what I mean in the rest of the book. For now, these are the essential pillars:

- Domains and namespaces. Later, I'll talk about what a domain is, but DDD teaches us to structure our code very expressive and logical.
- Choosing the proper names. For example, if the business refers to the users as "customers" or "employees, " you should rename your User class to follow that convention.
- The classes and objects should express the intention behind them. In the most simple Laravel application, you have models and controllers. What do you think when you see a project with 50 models and 50 controllers? You can see the application's primary domain, but you have to dig deeper if you want to have a good understanding of the features, right? Now, what about 300 models and 500 controllers? You have no chance to reason about this kind of application.

In most projects, developers prefer technical terms over business concepts. That's natural. After all, we're technical people. But I have a question: are those technical terms significant?