Hexagonal Architecture

In this lesson, we will have an insight into the Hexagonal architecture.
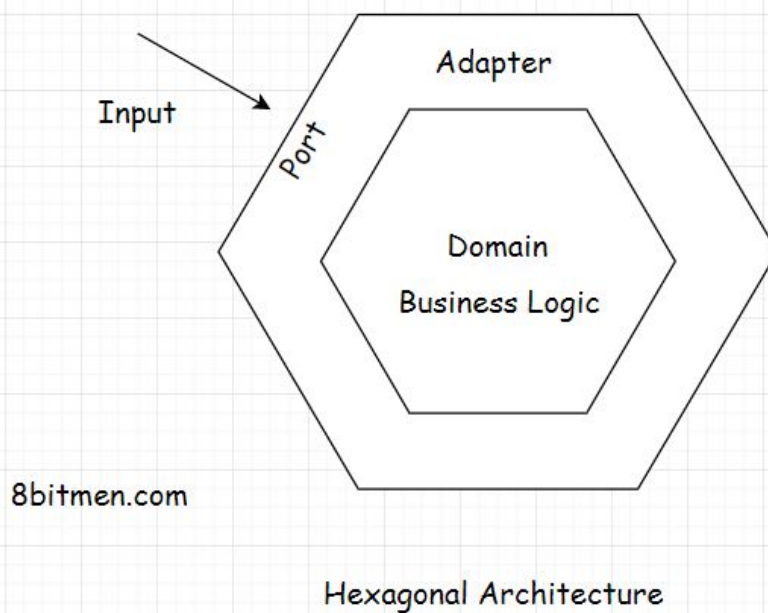
What Is A Hexagonal Architecture?

The architecture consists of three components:

- Ports
- Adapters
- Domain

The focus of this architecture is to make different components of the application: independent, loosely coupled & easy to test.

The application should be designed in a way such that it can be tested by humans,

automated tests, with mock databases, mock middleware, with & without a UI,

without making any changes or adjustments to the code.

The architectural pattern holds the *domain* at its core, that's the *business logic*. On the outside, the outer layer has *Ports* & *Adapters*. *Ports* act like an *API*, as an interface. All the input to the app goes through the interface.

So, the external entities don't have any direct interaction with the *domain*, the business logic. The *Adapter* is the implementation of the interface. Adapters convert the data obtained from the *Ports*, to be processed by the *business logic*. The *business logic* lies at the centre isolated & all the input and output is at the edges of the structure.

The hexagonal shape of the structure doesn't have anything to do with the pattern, it's just a visual representation of the architecture. Initially, the architecture was called as the *Ports and the Adapter pattern*, later the name *Hexagonal* stuck.

The *Ports & the Adapter* analogy comes from the computer ports, as they act as the input interface to the external devices & the *Adapter* converts the signals obtained from the ports to be processed by the chips inside.

Real World Code Implementation

Coming down to the real-world code implementation, isn't that's what we already do with the *Layered Architecture* approach? We have different layers in our applications, we have the *Controller*, then the *Service Layer* interface, *Class* implementations of the interface, the *Business logic* goes in the *Domain model*, a bit in the *Service*, *Business* and the *Repository* classes.

Object Model

User

Controller - REST API

Service

Data Access

Database
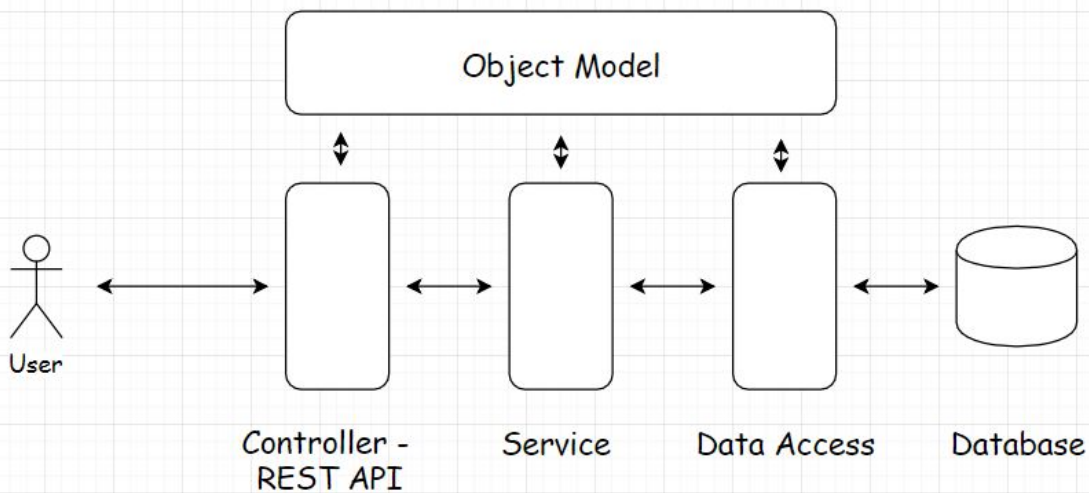
8bitmen.com

Layers of a Web Application

Well, yeah. That's right. First up, I would say that the Hexagonal approach is an evolution of the layered architecture. It's not entirely different. As long as the business logic stays in one place, things should be fine. The issue with the layered approach is, often large repos end up with too many layers beside the regular service, repo & business ones.

The business logic gets scattered across the layers making testing, refactoring & pluggability of new entities difficult. Remember the *Stored procedures* in the databases & the business logic coupled with the UI in *JSP Java Server Pages*?

When working with *JSPs* and *Stored procedures*, we still have the layered architecture, the *UI layer* is separate, the *persistence layer* is separate but the *business logic* is tightly coupled with these layers.

On the contrary, the *Hexagonal pattern* has its stance pretty clear, there is an inside component which holds the *business logic* & then the outside layer, the *Ports* & the *Adapters* which involve the *databases*, *message queues*, *APIs* & stuff.