Hexagonal Architecture

In this lesson, we will introduce hexagonal architecture.

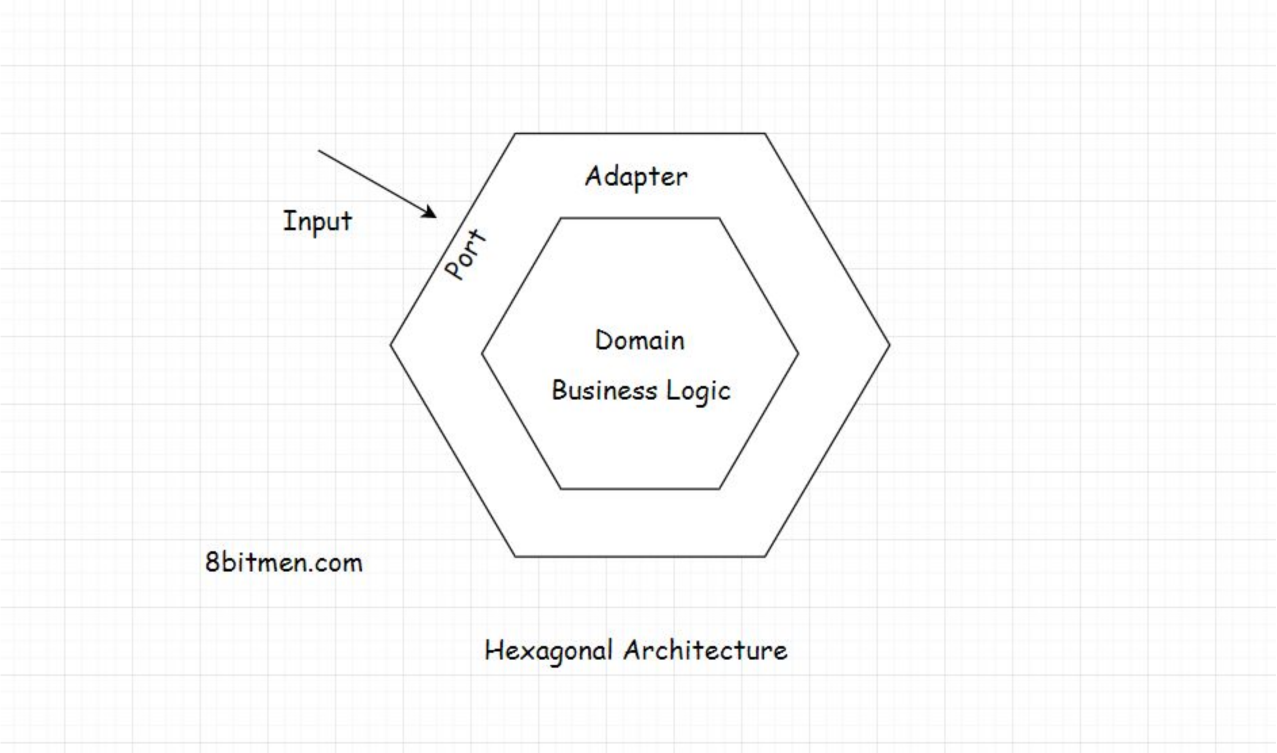
We'll cover the following

- What is a hexagonal architecture?
- Real-world code implementation

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, and easy to test.

The application should be designed in a way such that it can be tested by humans and automated tests, with mock databases and mock middleware, with and without a UI, and without making any changes or adjustments to the code.

The architectural pattern holds the *domain* at its core, meaning the *business logic*. On the outside, the outer layer has *ports* and *adapters*. *Ports* act like an *API* as an interface. All the input to the app goes through the interface.

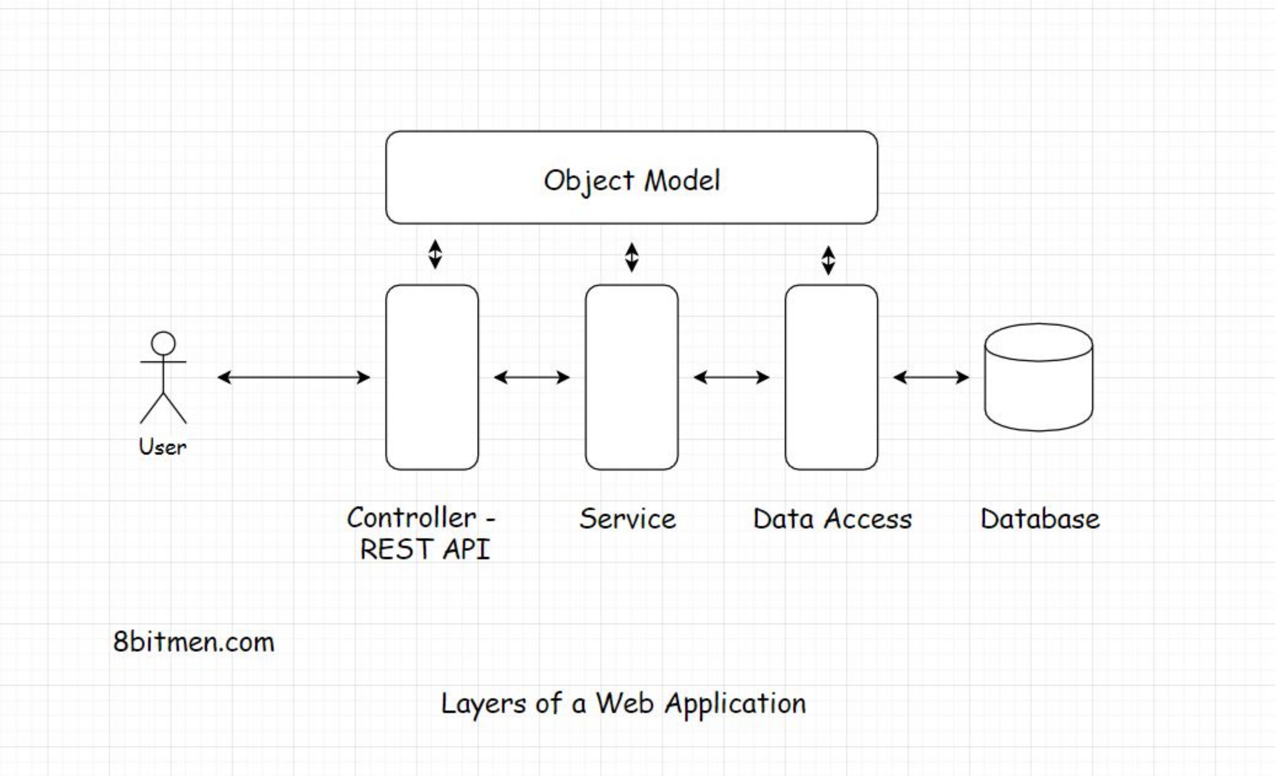
So, the external entities don't have any direct interaction with the *domain*, the business logic. The *adapter* is the implementation of the interface. Adapters convert the data obtained from the *ports*, to be processed by the *business logic*. The *business logic* lies isolated at the center, and 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 the *ports and the adapter pattern*. Later, the name *hexagonal* stuck.

The *ports & the adapter* analogy comes from computer ports because they act as the input interface to the external devices, and 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 this 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, the *class* implementations of the interface, the *business logic* that goes in the *domain model*, and a bit in the *service*, *business*, and the *repository* classes.



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, and business ones.

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

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* makes its stance pretty clear: there is an inside component, which holds the *business logic*, then the outside layer, and the *ports* and the *adapters*, which involve the *databases*, *message queues*, *APIs*, and whatnot.

← Back

Shared-Nothing Architecture

Next →

More on Architecture Quiz – Part 1

☑ Mark as Completed

Report an Issue