Architecture flow application

Application Specifics:

Blockchain: Cardano
Tech stack:
Backend: Node.js + Typescript
Database: Mongodb + PostgreSQL
SmartContract: Solidity, OpenZeppelin, SafeMath
Testing: Mocha
Environment: Cloud Provider + Rancher + kubernetes + ceph + docker + Ingress / Outgress + NGINX+ + Gitlab
The importance of good architecture for Liqwid

Having a good starting point when it comes to our project architecture is vital for the life of the project itself and how you will be able to tackle changing needs in the future. A bad, messy project architecture often leads to:

- **Unreadable and messy code**, making the development process longer and the product itself harder to test
- **Useless repetition**, making code harder to maintain and manage
- **Difficulty implementing new features**. Since the structure can become a total mess, adding a new feature without messing up existing code can become a real problem

With these points in mind, we can all agree that our project architecture is extremely important, and we can also declare a few points that can help us determine what this architecture must help us do:

- Achieve clean and readable code
- Achieve reusable pieces of code across our application
- Help us to avoid repetitions
- Make life easier when adding a new feature into our application

Establishing a flow

We can discuss what I usually refer to as the application structure flow. The application structure flow is a set of rules and common practices to adopt while developing our applications.

The goal is to create a quick reference guide to establishing the perfect flow structure when developing Node.js application. Let’s start to define our rules:

#1: Correctly organize our files into folders

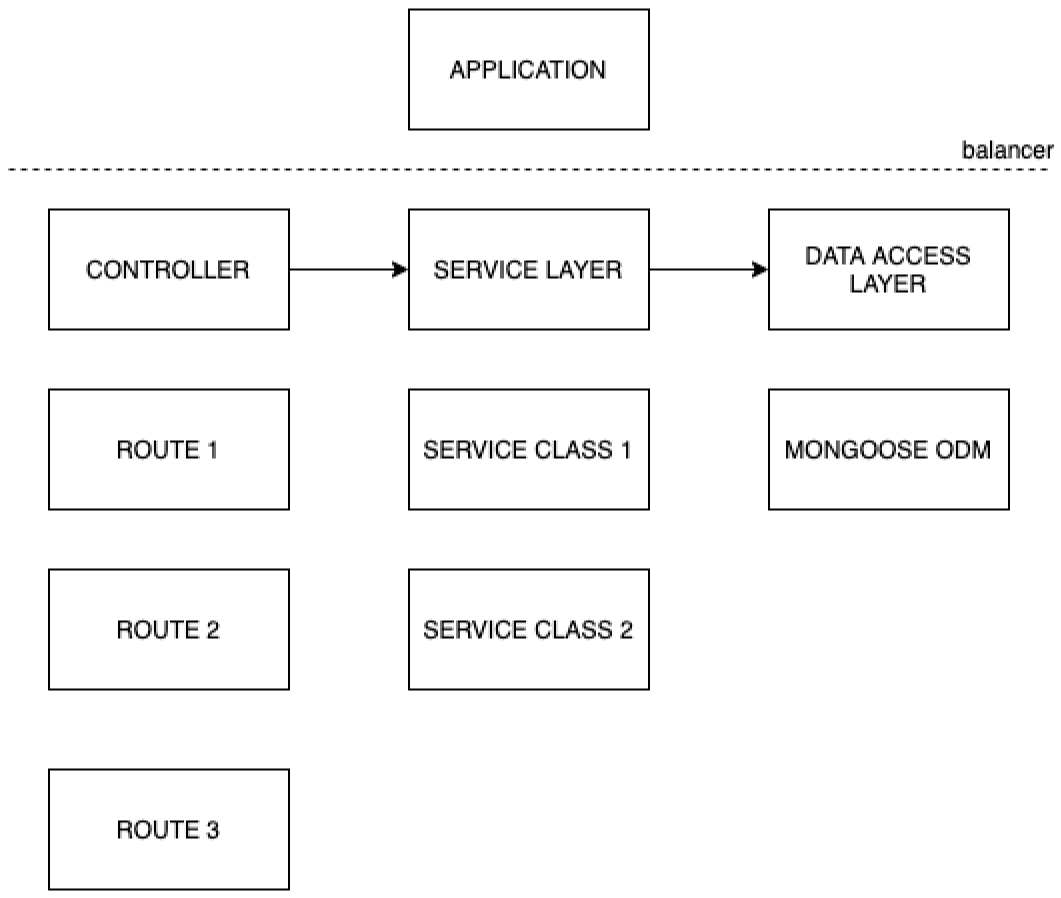
Everything has to have its place in our application, and a folder is the perfect place to group common elements. In particular, we want to define a very important separation, which brings us to rule number #2:

#2: Keep a clear separation between the business logic and the API routes

We will use Express.js It provide us with incredible features for managing requests, views, and routes. With such support, it might be tempting for us to put our business logic into our API routes

#3: Use a service layer (3 Layer architecture)

This is the place where all our business logic should live. It’s basically a collection of classes, each with its methods, that will be implementing our app’s core logic. The only part you should ignore in this layer is the one that accesses the database; that should be managed by the data access layer.



And the subsequent folder structure sending us back to rule #1 can then become:

```
  src
     app.js          # App entry point
  api              # Express route controllers for all the endpoints of
the app
  config           # Environment variables and configuration related
stuff
  jobs             # Jobs definitions for agenda.js
  loaders          # Split the startup process into modules
  models           # Database models
  services         # All the business logic is here
  subscribers      # Event handlers for async task
  types            # Type declaration files (d.ts) for Typescript
```

#4: Use unit testing

We can also implement unit testing for our project. Testing is an incredibly important stage in developing our applications. The whole flow of the project — not just the final result — depends on it since buggy code would slow down the development process and cause other problems.

A common way to test our applications is to test them by units, the goal of which is to isolate a section of code and verify its correctness. When it comes to procedural programming, a unit may be an individual function or procedure. This process is usually performed by the developers who write the code.

Benefits of this approach include:

1. Improved code quality

2. Bugs are found earlier
3. Cost reduction

#5: Use another layer for third-party services calls

Often, in our application, we may want to call a third-party service to retrieve certain data or perform some operations. And still, very often, if we don’t separate this call into another specific layer, we might run into an out-of-control piece of code that has become too big to manage.

A common way to solve this problem is to use the pub/sub pattern. This mechanism is a messaging pattern where we have entities sending messages called publishers, and entities receiving them called subscribers.

The publish-subscribe model enables event-driven architectures and asynchronous parallel processing while improving performance, reliability, and scalability.

#6: Use a linter

This simple tool will help you to perform a faster and overall better development process, helping you to keep an eye on small errors while keeping the entire application code uniform.

#7: Use a style guide

Still thinking about how to properly format your code in a consistent way? Why not adapt one of the amazing style guides that Google or Airbnb ha ve provided to us? Reading code will become incredibly easier, and you won’t get frustrated trying to understand how to correctly position that curly brace.

#8: Use promises

Promises bring in more pros than cons by making our code easier to read and test while still providing functional programming semantics together with a better error-handling platform.

Advantages of Node.js Architecture

- **Handling multiple concurrent client requests is fast and easy**With the use of Event Queue and Thread Pool, the Node.js server enables efficient handling of a large number of incoming requests.
- **No need for creating multiple threads**Event Loop handles all requests one-by-one, so there is no need to create multiple threads. Instead, a single thread is sufficient to handle a blocking incoming request.
- **Requires fewer resources and memory**Node.js server, most of the time, requires fewer resources and memory due to the way it handles the incoming requests. Since the requests are processed one at a time, the overall process becomes less taxing on the memory.