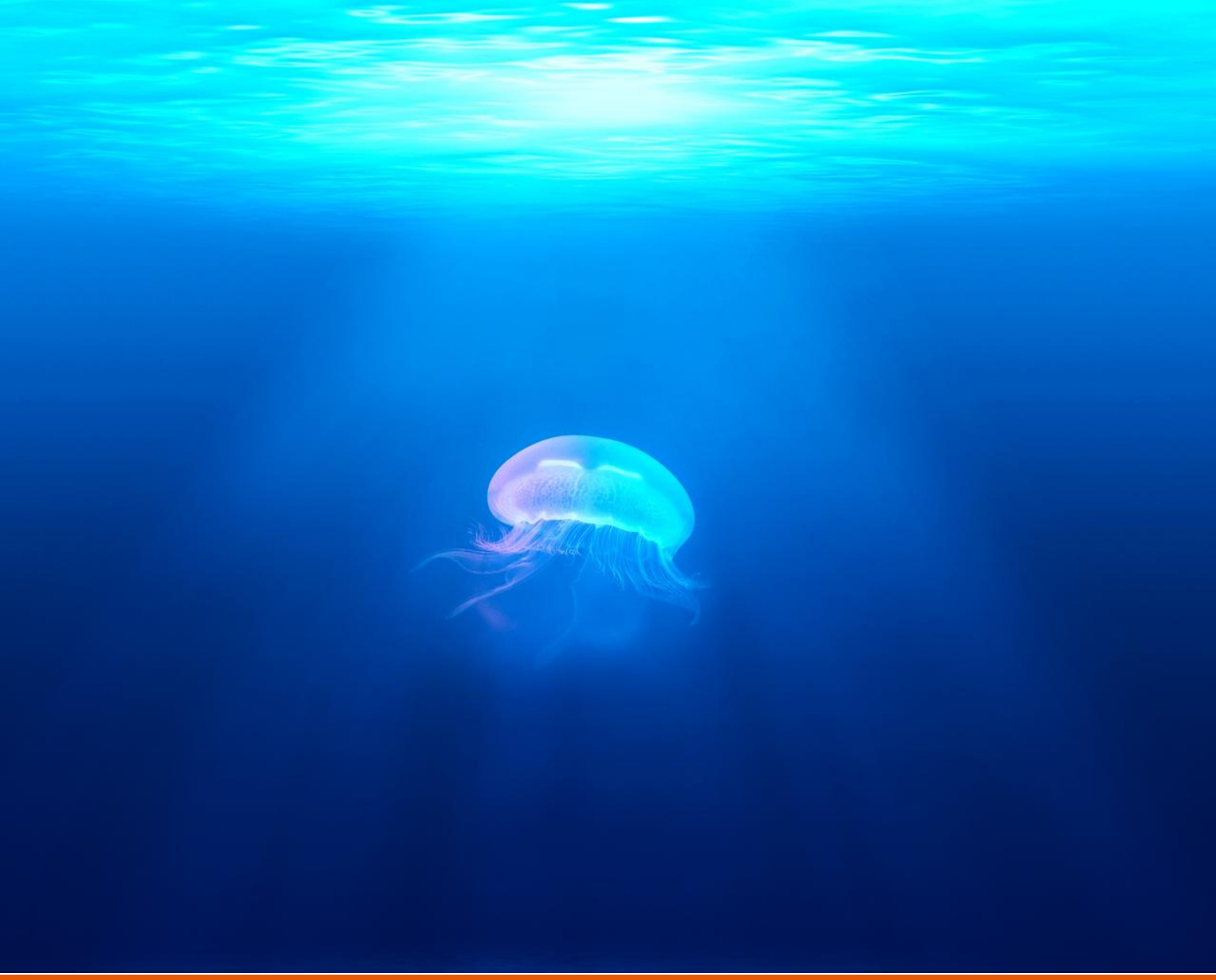
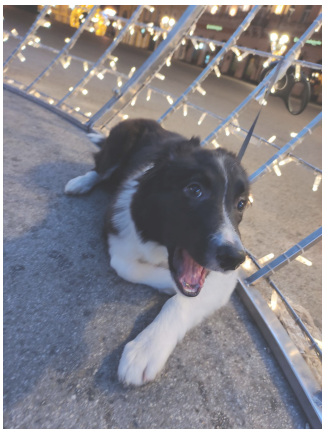


Pragmatic Node.js development

Primer in Nest.js



Author: Zlatibor Zed Veljkovic

Date: June 5, 2022

Version: 0.1

bioinfo 1: bioinfo 2

Some extra info

Contents

Chapter 1 Developer tools

Since ancient times, mankind has constantly spent effort to create new or to improve existing tools. Even now, after thousands and thousands of years we are doing the same. We are making new tools that will make us more efficient or at least to do our tasks easier. In software development there are so many tools available it is hard to choose which set should be used. Next sections will give simple overview of most prominent tools for each section.

1.1 Command line interfaces

Commad line interfaces (CLI) are programs that use textual interface and allow you to interact with it. Every operating system comes with one or more of these, Windows has Command Prompt (aka cmd.exe) and Power Shell, Linux has sh and bash with many alternatives (commonly known as shell), and Mac OS has Terminal.app. One issue that novice developers struggle is that when someone tells you to "open the terminal", they mean one for your system.

Before mentioned apps are the ones that allow you to execute some commands or run different programs. There are also some CLI that is specifically built for one purpose. One example would be Nest.js CLI which allows you to quickly create new projects, update dependencies or start the Nest app.

1.1.1 Command prompt

Command prompt comes preinstalled on Windows systems. It supports batch scripts and usually the file is with .bat extension.

Pros	Cons
• Available on all Windows systems • Allows executing programs in current directory without `.\` prefix	• Batch scripting language is really outdated and hard to write more complex stuff • No command history search

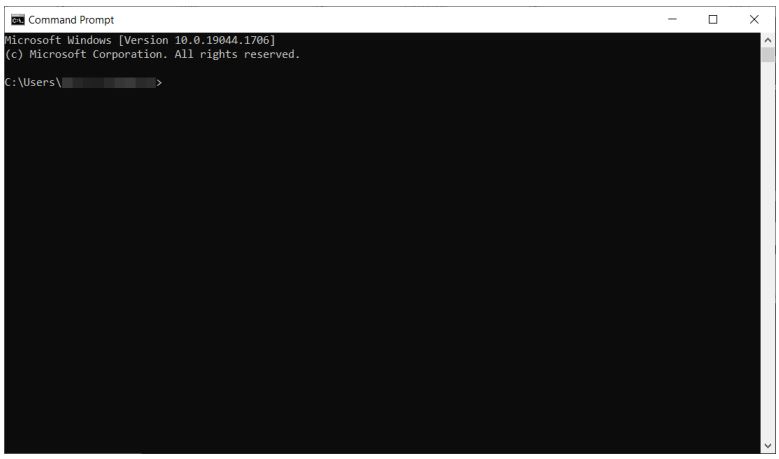


Figure 1.1: Command Prompt

Reference for available Command Prompt commands can be found at Windows Commands

1.1.2 PowerShell

Another shell for Windows systems is called PowerShell and it is available from Windows 7 or later operating systems. Open source version PowerShell Core was released in 2016 and it is based on .Net Core which also made it cross-platform. It has better integration with various functionalities available in Windows so it is prefreed choice over Command Prompt when working with system administration. For developer work it might be an overkill.

Pros	Cons
• Available on all Windows systems • Better integration with Windows functionalities • Has command history search (with F8 key)	• Does not allow executing programs in current directory without `.\` prefix

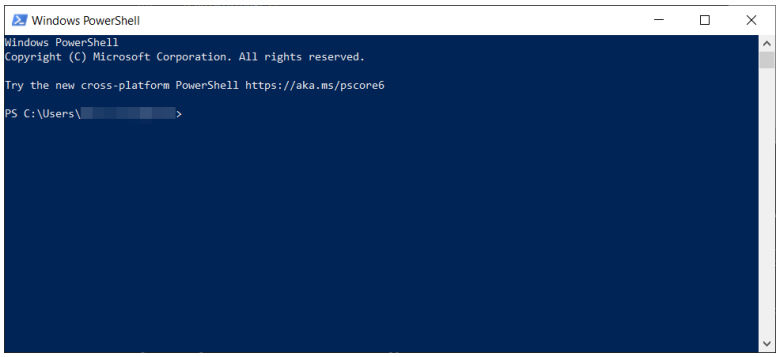


Figure 1.2: PowerShell

1.1.3 Linux/Mac terminals/shells

Linux and Mac have much bigger choice of terminals. Terminal refers to a program that allows you to run programs which are known as shells. Shells come in lots of varieties sh, bash, ksh, csh, zsh. . . . Linux has many command line programs that allow you to manipulate output of commands and offers a lot for power users.

1.2 Package managers

Package managers can be used to install additional software on your PC. They usully automate process of downloading, installing and configuring software. Later it also helps with keeping the installed software up to date or with removal.

- Chocolatey is most prominent Windows package manager. It can be downloaded from https://chocolatey.org/. Searching for packages is done with `choco search postgresql` and installation with `choco install postgresql` will install Postgres.
- Homebrew is a Mac OS X package manager written in Ruby. It can be downloaded from https://brew.sh. Command `brew search postgres` is used to search for package while `brew install postgresql` to install a package.
- Linux has many package managers and preferred way is to use package manager that comes with operating system.

Chapter 2 NestJS application recommendations

2.1 NestJS

I couldnt explain it better then the words of the author ”Nest (NestJS) is a framework for building efficient, scalable Node.js server-side applications. It uses progressive JavaScript, is built with and fully supports TypeScript (yet still enables developers to code in pure JavaScript) and combines elements of OOP (Object Oriented Programming), FP (Functional Programming), and FRP (Functional Reactive Programming).”

The documentation is found on where you can read all about framework components. Some of the components are not up to task and in further sections we will focus on good and bad sides of available components and some alternatives.

2.2 Environments

Modern apps usually run in multiple environments. My recommendation for environments is with code name in parenthesis.

- Local development environment (***a***)
- Local testing environment (***b***)
- Public development environment (***d***)
- Public stable development environment (***s***)
- CI testing environment (***t***)
- Public quality assurance environment (***q***)
- Public production environment (***p***)

2.2.1 Local development environment

Local development environment is pretty obvious as each developer is running the application on development machine. Code name to be used for this environment is ***a*** which as a letter represents first letter in alphabet so this environment represents first step in the application development.

2.2.2 Local testing environment

Local testing environment (***b***) is where developer runs tests on development machine. This environment is necessery to isolate testing of the app completly from local development environment. Without this isolation data that has been entered during development could influence test running. We usually use same variables as ***a*** env but we usually want to change external dependencies like database/queues/Kafka/storage so test runs are properly isolated.

2.2.3 Public development environment

Public development environment is the environment which contains the latest application code. It is configured to be automatically built from the ***develop*** branch. This environment is meant to be broken/reset at any time. It is also first public environment for use mostly by developers.

2.2.4 Public stable development environment

Public stable development environment ***s*** is the environment that is manually updated. When team is happy with the state of ***d*** environment they can promote it to this environment. This allows team leads to test the app without interuptions that would come if they used ***d*** environment.

2.2.5 CI testing environment

CI testing environment is used during the PR verifications in the build pipelines. This is necessary for same reasons as local testing environment, just on the public side.

2.2.6 Public quality assurance environment

Public quality assurance environment ***q*** is environment meant for external evaluation. When all issues identified in ***s*** environment are fixed and team lead is happy with it's state, this build can be promoted to ***q*** environment. This environment should be same in specs as public production environment, so it can be also used as database migration verification or as performnce test target.

2.2.7 Public production environment

Public production environment ***p*** is environment that is available to the users of the application. This makes it last step in build - release cycle. As real users are using it this environment should be under constant monitoring.

2.3 Environment zones

All of the enivornments can be grouped in production and non-production zones. The app code can then provide developers with:

- More information about errors for example by outputing stack traces, displaying error data, and various other info which should be hidden from the real users.
- Improve testability by including features to login as certain type of users, commands that will precreate certain data sets, etc. . .
- Distinguish between seeding production data and developer data.

2.4 Application Configuration

Application configuration in modern development is unescapable. NestJS offers a component ***ConfigService*** which is using ***dotenv*** package internally. What we get is a way to load .env environment file and access the variables defined in environment.

What we are missing is:

- Checking that all required variables are supplied
- That variables conform to the expected type (configuration schema)
- Ability to use multiple .env files so that you can easily configure local development environment (***a***) and local testing environment (***b***) for example by having ***.env.b*** file overriding database name from ***.env*** used by local development environment.
- Type safety as we need to use string identifiers when getting the values

Package convict has almost all of the above but has a requirement that each setting has default value which will not fail the startup if required variable is not found in environment variables.

Zeddy Config

Zeddy Config is my library heavily inspired by convict but offers type safety without default values and is offering just enough functionality for any app. This package also provides variable loading from the ***.env*** and ***.env.env-name*** files with the ***dotenvize*** method. With this simple packages we cover all missing elements for good conifg:

- Single method load the environment variables into JS variable.
- Validation of the configuration schema.

- Full type safety as schema type propery is trasferred to config object properties.
- It also has additional method to load multiple .env files (***dotenvize***).

```
import { configz, dotenvize } from "zeddy-config";

dotenvize();

export const config = configz({
  env: {
    description: "Running environment",
    envVar: "NODE_ENV",
    type: "string"
  },
  server: {
    description: "Server info",
    type: "object",
    properties: {
      port: {
        description: "Server port",
        envVar: "SERVER_PORT",
        type: "int",
        validator: (port) => {
          return port === 3000;
        }
      },
      host: {
        description: "Server host",
        envVar: "SERVER_HOST",
        type: "string"
      },
    }
  }
});

// resulting config interface is {env: string; server: {port: number; host: string}}
}
```