NodeJS source code directory structure

Context

There are no guidelines on where to put the source code for your NodeJS projects.

- Do I suppose to use `src`, `lib`, `dist` or `build` directories, or maybe put everything at the root of the project?

Not having a clear guideline adds cognitive load, as well as add friction contributing to projects, and sometimes relying on a particular group of people that know the unspoken/undocumented rule.

In other completely valid cases, like putting everything at the root of the project, don't take into consideration caveats like having so many metadata such as `.gitignore`, `.prettierrc`, `.eslintrc`, `.markdownlint.json`, `tsconfig.json` files and many other files and directories.

Subjectively speaking, they add noise and hard to navigate or concentrate when put all your files at the root of the project.

Furthermore,

- Where do I suppose to put my test files? What are the differences between X and Y directories?
- How to structure the project when I have multiple testing frameworks, or multiple documentation tools?

This ADR is an attempt to follow the steps of other ecosystems like Elixir, Ruby or Rust came to a consensus about this topic.

About compilations

We should tell what requires compilation in the project, which is extremely useful in library projects since you should be aware of such detail to consume the project. Or in cases that you would like to run the project in production.

Having a directory that reflects such information is crucial to mitigate mistakes in application deployments, or misuse of libraries.

About test frameworks

Sometimes you may have multiple test frameworks for your project, most likely for `end-to-end`, `integration`, or `unit` tests, but not limited to those concepts. Therefore, people have tried to come up with `./tests/integration`, `./tests/unit`, `./tests/end-to-end`.

Although that may be a clean way to structure your tests, the problems arrive when you try to articulate the differences between those categories without getting into an in-depth conversation about X vs. Y and end up with unclear boundaries, relying too much on the programmer's experiences to articulate the most optimal definition or decision.

Additionally, if you try to migrate your project to another testing tool that falls into one of those possible categories, you will mingle tests that are run differently depending on the underline tool, and that may be really important in some cases.

About documentation

Documentation tools are amazingly diverse in the ecosystem; such tools require different configurations and code structure, therefore mingle tools may cause incompatible situations, or confusing documentation source code.

Also, it is unclear today what should I do if I have multiple documentation websites (internal or public as an example), and how to manage them.

Resolution

- You **MUST** follow the following directory structures:

```
.
├── package.json
├── bin
├── dist
├── docs
│   ├── storybook
│   └── website
│   └── website-next
│   └── [another website or tool]
├── examples
├── lib
├── src
└── tests
    ├── cypress
    └── jest
    └── puppeteer
    └── [another testing tool]
```

- In the `./tests` directory, you **MUST** use the tool's name to reflect the tool used to run the tests.
- In the `./docs` directory, you **MUST** separate documentation based on the underline tool or context (for example, **internal** or **public**) unless you only have one documentation tool or context.
- You **MUST** place `storybook` documentation under `./docs`.
- `./src` **MUST** required compilation step, and you **MUST** compile the source code into `./dist` directory.
- `./dist` **MUST** required to compile `./src` directory to generate the source code, or any other compilation step in required cases.

About files and directories

- **./package.json:** the NodeJS package configuration.
- **./bin:** contains executables files.
- **./dist:** contains compiled source code.
- **./docs:** contains the documentation of the project.
- **./examples:** contains examples of the project's usage, primarily used in library projects.
- **./lib:** source code that does not require compilation.
- **./src:** source code that requires some compilation.
- **./tests:** contains the tests files.