You are a software architect and I am the owner of a development framework (BeyondJS).

You are here to help me to make effective communication and documentation about BeyondJS.

To understand the framework, it occurred to me that a good way to understand it is to explain it by comparing it to other existing products.

Let’s call PMJS (Package Modules) to a concept of modern web development that involves the packaging of modules from packages into standard modules, which can be consumed across different execution environments.

We can say that actually PMJS would be made possible through the use of CDNs like Skypack or JSPM, which provide a standardized way of consuming modules without having to worry about the specifics of each package's internal structure. By packaging modules as a single resource, PMJS can simplify the process of module resolution, and reduce the complexity to support different execution environments. The use of PMJS can lead to more maintainable, scalable, and efficient code in modern web development.

Well, BeyondJS is a universal plus modular javascript framework that somehow works on a concept like PMJS.

Developers install BeyondJS from npm (npm i -g beyond), they run BeyondJS just by executing beyond in the projects folder, and then BeyondJS makes a server available to consume the packages through an API that could be comparable to the API of modular CDNs like skypack or JSPM.

When BeyondJS is launched, it looks for the beyond.json file, the workspace configuration file. BeyondJS reads the packages property, and expects to find an array whose entries point to the configuration of the packages being developed locally.

Development mode packages are specified via the package.json file following the standard specification.

At this point, it is necessary to understand that the treatment that BeyondJS gives to the packages is divided into the external packages (installed dependencies) of the packages that are configured in the workspace (pacakges entry of the beyond.json file).

Packages that are dependencies of the packages specified in the workspace are given a treatment similar to that given by CDNs like skypack or JSPM to any package, the modules are packaged and made available by an API similar to theirs, but the packages of the workspace are there to be developed. And for that, they are expected to support capabilities like typescript, style sheet processing, HMR, etc.

The modules of the workspace packages are specified via the module.json file, where they can be configured:

1. the subpath (associated with the subpath of the package.json)

2. the supported platforms (associated with conditional exports), 3. the configuration of various types of processors such as typescript, sass, multi language texts, etc.

BeyondJS is extensible, and for this the processors can be created in a similar way to the plugins of other frameworks.

By following this development model, with BeyondJS it is possible to build modular and multi-platform applications.

BeyondJS compiles packages and applications, therefore it can be used to build:

1. Standard multiplatform packages with the expected features (for example typescript). BeyondJS generates the module declarations (types entry of package.json exports). It helps in type checking because it identifies the consumers of the modules and detects errors when code changes.

2. Compiles for browser, mobile, desktop, web, as well as backend deno, node, and conceptually any other javascript runtimes that support a module system like esm, cjs, or systemjs.

Additionally, thanks to the modular structure of BeyondJS it is very simple to structure applications under the design of microframeworks and microservices. For this, BeyondJS provides a set of packages (in turn created with BeyondJS) that allow you to create web components, which can be easily reused in different projects (@beyond-js/widgets).

@beyond-js/widgets provides a set of view framework adapters that can be used to integrate web components with popular view frameworks such as React, Svelte, and Vue. This can make it easier to create consistent and reusable user interfaces across different projects and applications. Additionally the widgets support server, client and static rendering.

The other package @beyond-js/backend allows you to create realtime backends. Internally @beyond-js/backend uses web sockets (socket.io).

In this way, in BeyondJS the concept of universal framework implies:

1. frontend (browser, web, mobile) and backend (node, deno and conceptually any other javascript environment)

2. SSR/CSR/SR.

3. Multiple view frameworks (React, Svelte, Vue).

4. Realtime backends

This is why we say that BeyondJS is the only one Universal Javascript Framework.