

State of The Art

baryonyx-api

# **Technologies Used**

**Choosing Node.JS:**

**Firstly, what is Node.JS?** Node.js came into existence when the original developers of JavaScript extended it from something you could only run in the browser to something you could run on your machine as a standalone application. We can do much more with JavaScript than just making websites interactive as JavaScript now has the capability to do things that other scripting languages like interpretive Python can do.

Furthermore, Node.JS is built upon Google Chrome’s V8 runtime—written in C++, built for multiple operating systems and super-fast.

**So, why we chose Node.JS?**

Every project starts out small or with the MVP release. But as time goes by, it continues to grow, gaining new features and users with high expectations. Without even noticing, we may end up with a huge project that our team struggles to cope with and introducing new changes or features becomes a nightmare. The answer to this is microservices pattern; it works by splitting our app into small units, which will make every part of it is independently deployable and scalable and individually tested. And Node.JS is perfect for it.

The single-threaded, event-driven architecture of Node.js allows it to handle multiple simultaneous connections efficiently. Most of the popular web platforms create an additional thread for each new request, using up RAM for the whole time it takes to process it. Node, on the other hand, operates on a single thread, making use of the event loop and callbacks for I/O operations or manipulating RDF data, delegating tasks such as database operations as soon as possible. Hence, this will make our use of Proxy pattern and our Rest API much easier.

Also, the latest version of Node.JS 12.14.0 increases the efficiency of the development process as it fills the gap between frontend and backend, and code executes faster than in any other language. And we should not forget about the NPM that offers us, developers, multiple tools and modules to use, thus further boosting productivity.

Bonus: Solid Server is based on Node.JS also.

# **Servers**

**Choosing Node Solid Server:**

For this project, we chose **node-solid-server f**or client identification and reading RDF and the user’s pod information. **node-solid-server** lets us run a Solid server which we’ll use alongside **Docker** on top of the filesystem.

The latest version of the server includes Linked Data Platform, Web Access Control, WebID+TLS Authentication, Real-time live updates, Identity provider for WebID, CORS proxy for cross-site data access, Group members in ACL and Email account recovery.

We were at first debating whether to use inrupt version of the server or the solid version, but looking through their GitHub repositories and official documentation, it seemed that the inrupt version is a work in progress and it’s not stable. Also, from the official solid webpage section for developers **pod server *(***[***https://solidproject.org/for-developers/pod-server***](https://solidproject.org/for-developers/pod-server)*)*, it redirects us to the Solid version (which we will eventually use) and not the Inrupt one.

# **Modelo de Dominio**

**Choosing Allergies:**

We chose the class **Alergia** to be saved in the user’s pod because we found it’s the most personal and interesting one in comparison with height, name, etc.