**1. A Read-only API**

* The SSR technique has two main drawbacks:
  + The server sends to the client an HTML content. However, we can imagine applications that do not need HTML content but only the data (e.g. Android mobile apps, C# native apps, etc.).
  + All the rendering effort is supported by the server and this could cause difficulties in scaling application for multiple concurrent clients.
* APIs approach tries to solve the above problems by only sending data to the client, in a standardized format such as JSON or XML.
* In the 4-api-server-r example we present an API that only offers read option (R letter from CRUD – Create, Reading, Update, Delete).
* See how the server.js become simpler since it does not have to create an actual HTML code. However, the rendering effort and the code complexity goes to the client, as you can see in /public/main.js. That is why, this approach is called CSR – Client-Side Randering.

**2. A CRUD API**

* Analize the 5-api-server-crd example for a Create-Read-Delete API.
* Some explanatory resources:

<https://masteringjs.io/tutorials/express/body>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

<https://stackabuse.com/bytes/how-to-return-status-codes-in-express/>

<https://javascript.info/fetch#post-requests>

* It is very useful to test the API with RestMan and curl.