Since this is the first formal web application that I build using React JS, I will enumerate all the followed steps in order to you to know/check/evaluate exactly what I have done and why (according to my current knowledge). This will help to identify my weak points and where I should focus on.

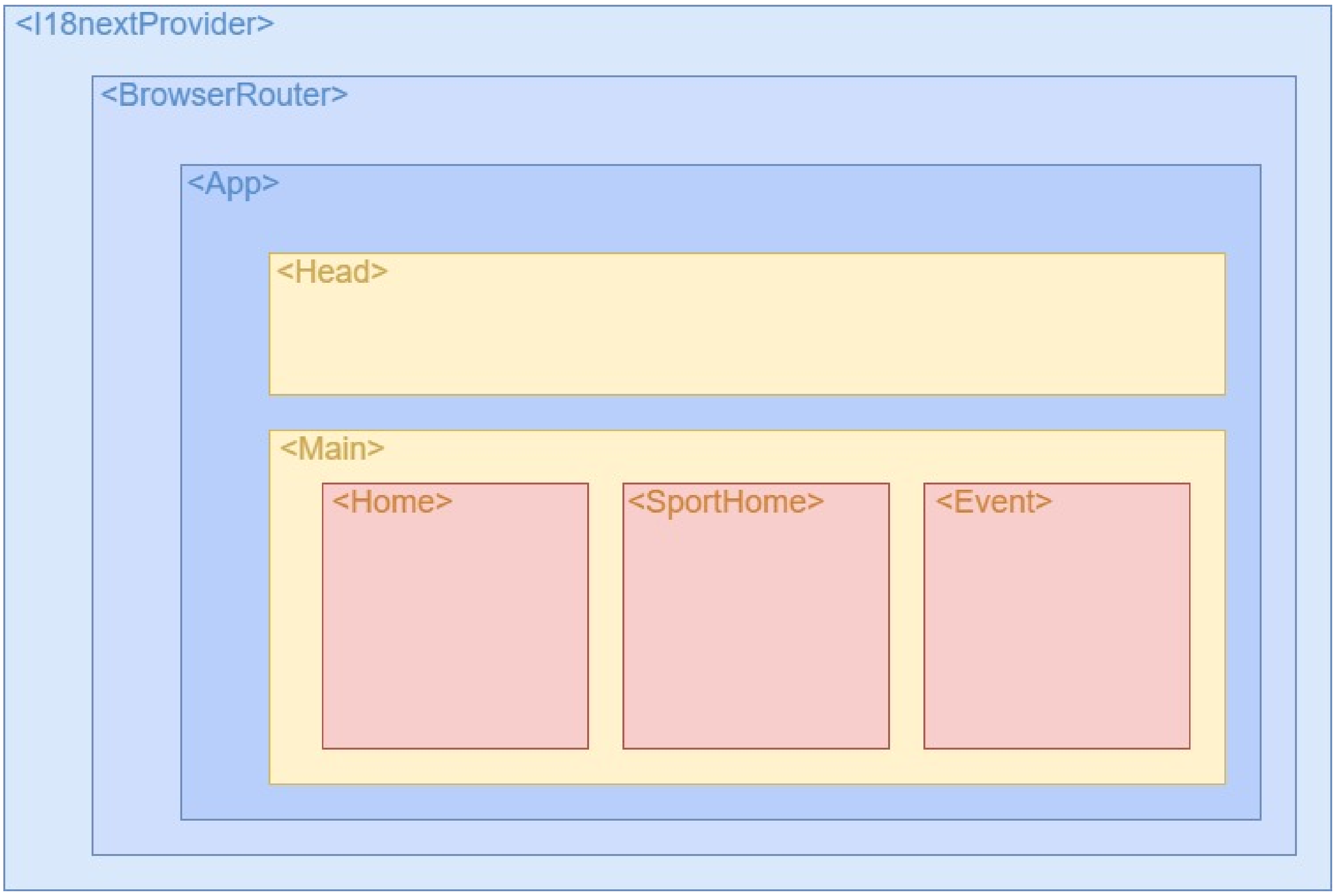
1 – PREPARING ENVIRONMENT

I have installed the following frameworks/modules in order to ease out the building of the application's structure:

- **React JS**
- **Create-React-App**: a single command builds out the application, ready to start coding.
- **React Router**: an essential module that really makes a difference when the application navigation has several levels.
- **i18next / i18next-Language Detector**: Provides quite an interesting way to deal with the multilanguage feature. It even offers the possibility of detecting your system language automatically and apply the proper translation.
- **Jest**: a test runner. It is already included and configurated if we are using Create-React-App. The only requirement is placing the test files into '__test__' folder or naming them as 'filename.test.js'

2 – ROUGH IDEAS BEFORE STARTING CODING

In the beginning, I considered the Sport Tree that is needed to be shown as an independent entity that changes according to the user actions (for example, the typical accordion left menu with Sports/Groups/Competitions levels popped into my mind). But since Events and Outcomes/Markets need also to be displayed, I discarded that idea. Finally I considered every level of the navigation (Home, SportHome with Events, Event's details) as a full React Component. It would help us to fill up every section with more content if we wished. For example, if our retrieved JSON had only prematch matches, we could ask for inplay Events according the selected Sport. So, the basic structure would end up like this:



<Home>, <SportHome> and <Event> components would be loaded according to the Router / URL.

3 – 'SPORTTREE.JS' AS A DATA CONTAINER

Out of this structure, we have the SportTree object, which is some kind of container with its own methods to retrieve the information. Although the incoming data from the JSON is an Array, the right way of keeping that data into our container would be as an associative Array, indexed by ID. That would make our hits or queries against the container rather faster and easier.

At classes/Components level, the idea would be having a basic 'Entity' class, with its own basic states/methods (set 'status'), and more classes such as 'Sport', 'Group', 'Competition', 'Event', 'Outcome', etc… that extend 'Entity' with more methods (set 'score', set 'period_id', set 'odds', etc) that would be useful for rendering and updates purposes.

Despite of that, I have kept the original JSON format and built some basic methods ('allSports', 'getEvents', 'getEvent') to retrieve the required data. Of course, if the data was indexed, I would need only one method, which would work like a SQL query, that would get N parameters: the full route of the element to be found (sport_id -> event_id, etc).

Since I was not able to retrieve the JSON in an normal way (blocked area), I had to download it using a Chrome plugin and catch it into the SportTree Object using 'fetch', inside the 'init' method.

During 'componentDidMount' method from 'App' React Component, SportTree 'init' method is called in order to fill our data container.

4 – KNOWN PROBLEMS

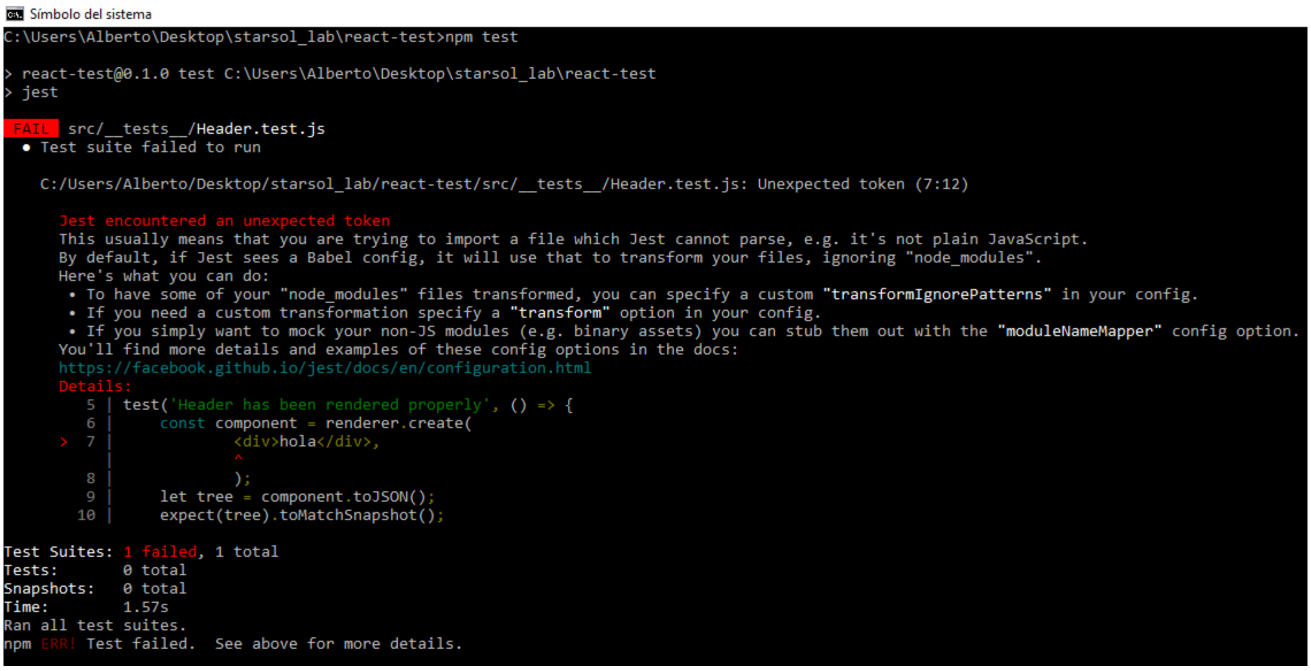
- SportTree container is a basic JS Object, so that means that if we went directly to, for instance, this URL: /sports/17500/event/956732600, even though the data is loaded from the JSON, this Object wouldn't notify other components. So that means that we are getting the "Not Event found" message. According to what I have been reading in some articles/tutorials, the proper solution would be using Redux. We could use a proper Redux container that keeps the data that the application needs, and use Redux 'actions' and 'reducers' to get/set the information.

  In fact, all those elements have been created into 'actions', 'reducers' and 'data' folders, but I haven't been able to do them to work, since I need to dig in more and to understand how all these elements (Redux container, asynchronous petitions, and Router) work together. So in order to make the application works, it uses the basic SportTree JS Object.

- Data is not indexed: keeping, for example, the Events indexed by ID would make easier the typical set/get actions, especially if we keep in mind the big amount of updates that a live Event can receive. Even though we have methods such as 'find' for JS Arrays, accesing directly by ID is always faster.

- Jest tests always retrieve the same error message (see capture below). Even though it offered some possible causes and solutions, I haven't been able to run the test properly in this case (just testing the rendering of the <Head> Component).

- Due to the fact that this is my first React JS app and my lack of knowledge, I am pretty sure that the structure of files/clases/JSX tags could be really improved.


```
Símbolo del sistema
C:\Users\Alberto\Desktop\starsol_lab\react-test>npm test

> react-test@0.1.0 test C:\Users\Alberto\Desktop\starsol_lab\react-test
> jest

 FAIL  src/__tests__/Header.test.js
  ● Test suite failed to run

    C:/Users/Alberto/Desktop/starsol_lab/react-test/src/__tests__/Header.test.js: Unexpected token (7:12)

      Jest encountered an unexpected token
      This usually means that you are trying to import a file which Jest cannot parse, e.g. it's not plain JavaScript.
      By default, if Jest sees a Babel config, it will use that to transform your files, ignoring "node_modules".
      Here's what you can do:
       • To have some of your "node_modules" files transformed, you can specify a custom "transformIgnorePatterns" in your config.
       • If you need a custom transformation specify a "transform" option in your config.
       • If you simply want to mock your non-JS modules (e.g. binary assets) you can stub them out with the "moduleNameMapper" config option.
      You'll find more details and examples of these config options in the docs:
      https://facebook.github.io/jest/docs/en/configuration.html
      Details:
         5 | test('Header has been rendered properly', () => {
         6 |     const component = renderer.create(
      >  7 |             <div>hola</div>,
           |             ^
         8 |             );
         9 |     let tree = component.toJSON();
        10 |     expect(tree).toMatchSnapshot();

Test Suites: 1 failed, 1 total
Tests:       0 total
Snapshots:   0 total
Time:        1.57s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

5 – FINAL THOUGHTS

Definitely, I will carry on studying / practicing with the React JS + Router + Redux combo after delivering this test, since it means going a big step forward if you compare it to the technology we use in my current company.