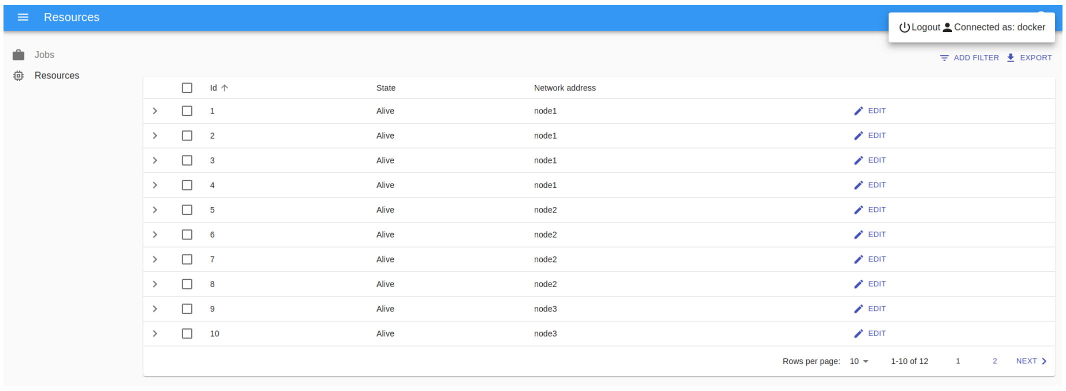
Dashboard for the task manager OAR

Antoine SAGET - Alexis ROLLIN



Id	State	Network address	
1	Alive	node1	EDIT
2	Alive	node1	EDIT
3	Alive	node1	EDIT
4	Alive	node1	EDIT
5	Alive	node2	EDIT
6	Alive	node2	EDIT
7	Alive	node2	EDIT
8	Alive	node2	EDIT
9	Alive	node3	EDIT
10	Alive	node3	EDIT

Rows per page: 10 1-10 of 12 1 2 NEXT

Contents

1 Overview

As fourth-year computer engineering students, we were asked to take part in a project in order to strengthen our computing and project management skills.

This document describes the purpose of the project, the technologies involved, our approach, the problems we faced and ways of improvement. We tried to keep our critical thinking as much as possible while writing this report, so it can serve as a guide for future works.

The source code of our application can be found at:
`https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INFO4/19-20/23/oar-dashboard/`

Our documentation can be found at:
`https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INFO4/19-20/23/docs`

A video demo of our application can be found at:
`https://www.youtube.com/watch?v=CKCQPf8wR6A`

2 Introduction

2.1 Presentation of the project

2.1.1 The problem

OAR is a resource and task manager for HPC clusters. Currently it can mostly be used through an interactive command-line which is not ideal and not very user fiendly. Managing thousands of resources and jobs can be a tedious task. Therefore, the need for an administration dashboard to manage jobs et resources emerge.

2.1.2 The context

Some attemps to develop an **OAR** dashboard have been made in the past. **oar skylight** was made by other students and wasn't considered a good solution as **Angular** isn't mature enough and support for the application would need a lot of work. With this in mind, this new project was created using **React** and **React-Admin**.

2.1.3 The goal

Once the new dashboard has been created a comparison between this environnment and **Angular** can be made. The final goal is to say if yes or no, **React** + **React-Admin** are suitable for a longterm dashboard solution for **OAR**.

2.2 Technologies

2.2.1 React

As said before, our dashboard application is based on **React**. React is a free Javascript library developed by Facebook for building user interfaces. It is commonly used as a base for developing mobile applications or single-page applications like our dashboard as it eases the creation of interfaces thanks to pieces of code called components.

2.2.2 React-Admin

According to **React-Admin** official website, **React-Admin**, developed by Marmelab, is "a frontend framework for building admin applications running in the browser, on top of REST/GraphQL APIs, using ES6, React and Material Design". In other words, it eases (even more than **React** does) the building of an admin interface by providing ready-to-use **React** components and functionalities. For example, **React-Admin** offers several login components which aim to code efficiently an authentication system. Through the `dataProvider` and `authenticationProvider`, it also enables to communicate with APIs, which will be very useful for our application.

It is important to note that our dashboard was developed in Typescript with React-Admin v3, which still hasn't completed its transition from Javascript to Typescript. We will come back to this point at the end of the report.

2.2.3 Oardocker

Oardocker is the tool we used to emulate an **OAR** cluster with many nodes on our laptop. With a simple command we can start a cluster as well as the server answering to API calls without the need to connect to a real cluster. The setup is quite easy and is a real benefit over a connection to a real cluster. **OAR** allow the management of many resources and the scheduling of jobs for those resources. It also handles file management and permission / authentication control.

2.2.4 OarAPI

OarAPI is the link between the dashboard and the cluster. Everything going out of our application will be a call to the API.

3 Installing and starting the application

The instructions for the installation of **docker**, **oardocker** and the application as well as the starting of the application can be found here.

4 oar-dashboard

In the following sections we will take a look at the most important features of the project and how they are implemented using react-admin.

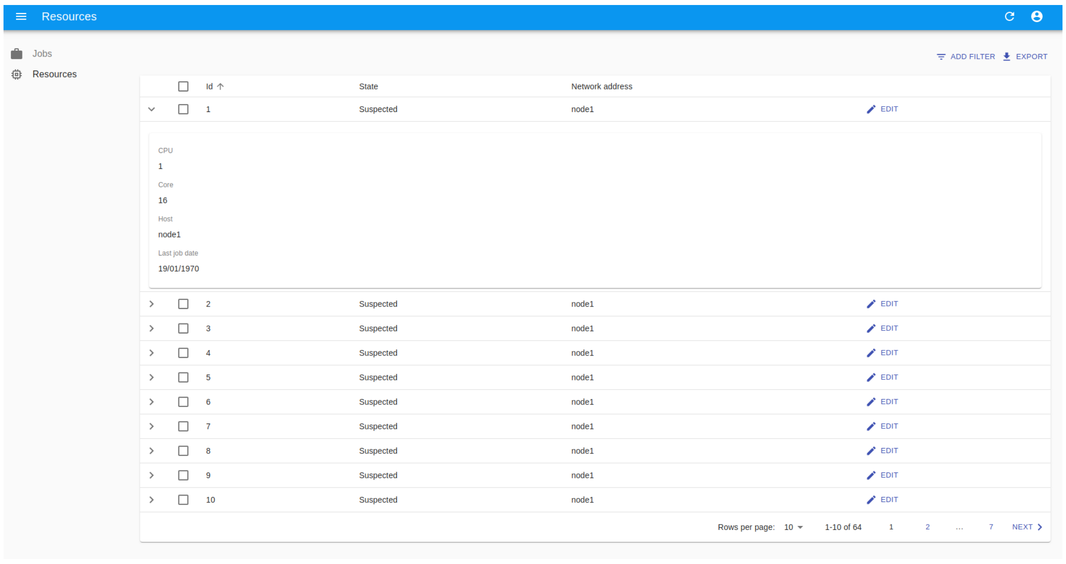
4.1 DataProvider

The **React-Admin** `dataProvider` is the glue layer between the **oarAPI** (or any other api) and the application. Once the `dataProvider` created, the app will make calls to it. A basic `dataProvider` is given but in order to fit to the API it have to be adjusted. The `dataProvider` perform mostly basic CRUD operations : `getList, getOne, update, updateMany, delete, deleteMany, create`.

The `dataProvider` assume a fully featured API with pagination, filtering and sorting which is not the case for **oarAPI**. Therefore, filtering and sorting is done after fetch for API calls that doesn't support it. This is not ideal but not a real issue as it's not **React-Admin** limiting the API but the opposite.

4.1.1 Ressources

In this project we focused on the **OAR** *resources* and *jobs*.
Resources can be viewed as a list, and opened to see the resource details. A resource can be deleted, or its state updated. The same operations are available in bulk which allow to manage multiple ressources quickly.



	Id	State	Network address	
	1	Suspected	node1	EDIT

CPU
1
Core
16
Host
node1
Last job date
19/01/1970

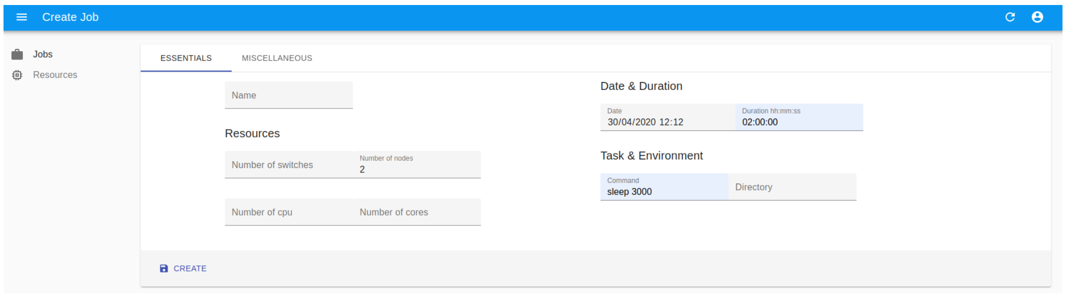
	Id	State	Network address	
	2	Suspected	node1	EDIT
	3	Suspected	node1	EDIT
	4	Suspected	node1	EDIT
	5	Suspected	node1	EDIT
	6	Suspected	node1	EDIT
	7	Suspected	node1	EDIT
	8	Suspected	node1	EDIT
	9	Suspected	node1	EDIT
	10	Suspected	node1	EDIT

Rows per page: 10 1-10 of 64 1 2 ... 7 NEXT

Figure 1: Resources tab. One ressource details is open

4.1.2 Jobs

Jobs can be viewed as a list as well. A job can be created using a form where the most important aspects of the job are defined. Jobs can be deleted.



Name

Resources

Number of switches
Number of nodes: 2
Number of cpu
Number of cores

Date & Duration

Date: 30/04/2020 12:12
Duration hh:mm:ss: 02:00:00

Task & Environment

Command: sleep 3000
Directory

Figure 2: Job creation form

4.2 AuthProvider

Following the architecture advised by the **React-Admin** documentation, the authentication aspect should be handled by a component of type `authProvider`. The `authProvider` is supposed to implement the functions `login`, `logout` and some authorization and error management functions such as checkError. Because the authentication is supposed to give access to certain functionalities of **OAR**, like creating jobs, the `authProvider` must communicate with **oarAPI**.

However, the implementation of the `authProvider` wasn't as simple as in the documentation, as **oarAPI** doesn't simply provide a `login` and a `logout` operation. The status of the user (connected or not) as well as its credentials have to be stored on the application side, and must be reinjected in every request to the API which requires to be authenticated. That is the reason why the `authRequester` component, containing the function `authRequest`, has been created. `authRequest` is used in the sensitive function (like `create`) of the `dataProvider`, checking if the user is authenticated and if so, adding the credentials to the request and sending it. Else, the user is redirected to the login page.

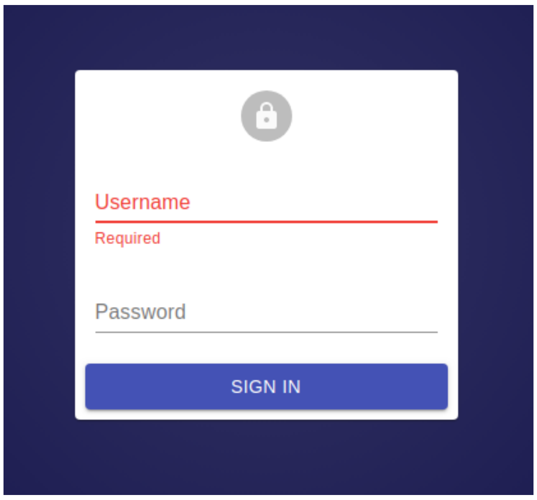
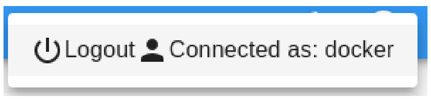
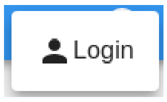


Figure 3: Login page

The basic logout button from **React-Admin** has been overriden to be turned into a login button when nobody is connected. Clicking this button when connected calls the `logout` function, clicking it when disconnected redirects to the login page.



(a) Logout button when connected



(b) Login button when disconnected

Figure 4: Button states

Let's detail a little bit how the `login` function has been developed, because it was tricky. To make it short, the first approach of the login was to retrieve the credentials thanks to the login page provided by **React-Admin** and then calls `whoami` with these credentials in order to get a validation (2xx or 3xx status code returned, usually 200 OK) or not (4xx status code returned, usually 401 Forbidden). This approach was the cleanest and the one adopted by the students of Oar-Skylight project. But it appeared that the response of the API was not only containing the status code but also a specific header (WWW-Authenticate) which causes the browser to display an authentication pop-up when the code is 401. As we didn't find a solution on the application side, making a patch with a new operation on the API side was necessary. Thus, the operation `authentication` was added to the API, returning 400 instead of 401 if the credentials are wrong.

There is a second difficulty we would like to mention, this time coming from **React-Admin**. The redirection to the login page triggers calls to the functions `checkAuth`, `checkError` and `logout`. This is the way the authentication in **React-Admin** was implemented and we had to deal with although it brings some issues. To take an example, when the user is redirected to the login page because an error occured, passing through the `logout` function raises a warning but catching the error earlier stops the redirection, because the redirection is prompted by the presence of an error.
This highlights the fact that when we want to get off the beaten track, development quickly become harder and, instead of being helped by **React-Admin**, we are trying to bypass **React-Admin** mechanisms.

5 Conclusion

5.1 React-Admin: Pros. and Cons.

5.1.1 Pros.

Even if we encountered some issues, objectively most things where quite straightforward. The link with the API throught the `dataProvider` makes sense and is easy to understand. Many integrated components are available already, limiting the code we have to do by hand. Some of theim, such as the `ListGuesser`, allow for fast and convenient prototyping in the beginning. The documentation is quite complete and guide through every important step of the creation of a clean dashboard.

However, things can be complicated when you want to go off track as explained bellow.

5.1.2 Cons.

React-Admin claims to be *"batteries included but removable"*. We found this affirmation to be false, for example, a simple layout issue where we wanted to have two columns instead of one cannot be done due to some react-admin limitations...

As explained in the authentication section, it could be hard to implements some custom logic which doesn't fit the perfect scenario described in the documentation. When we deal with unusual cases, what is meant to be a tool become a limit. That is the problem of having a too high-level framework.

React-Admin is currently migrating towards **Typescript**. **Typescript** strong typing is a major benefit and quality of life improvement over **Javascript**. However, as it's currenlty in migration it was a pain to use for this project. We think **Typescript** support will be a nice addition to **React-Admin** in the future, but currently, sticking to **Javascript** is easier.

5.2 React-Admin vs. Angular

In our opinion, **React-Admin** may not be flexible enough to develop an efficient dashboard for **OAR**. The one we have now works with few jobs and ressources, but isn't suitable nor convenient for the management of hundreds of jobs. Working with a lower level framework may be more appropriate : there could be more work to do but we should gain in flexibility and optimisation.

Compared to **Angular**, react-admin is retro-compatible and **Javascript** is compatible with **Typescript** so, even if we don't have the latest version of **React-admin**, it still works and it is not considered as obsolete whereas **Angular** is too radical in its updates. Hence, the maintenance of **React-admin** should be easier.
In terms of performances, it is hard to compare **Angular** and **React-Admin** as we never tried Angular.
After having looked to the github repo of oar-skylight, we tried to measure the coding efficiency offered by **Angular** but it is hard to guess which line of code was written by the students and which line came from Angular. Nevertheless, our general feeling is that the quantity of source code with **Angular** is higher than with **React-Admin** for an equivalent result.

5.3 Improvements

The application is working well but there still are some warnings to debug as well as feature to implement.

For the dashboard to be usable, we think that media support need to be added, as far as we know there is no pre-made component for file management in **React-admin**. And, our experience with the library leaves us skeptical about the implementation of a file system "by hand"... React-admin allow the uploading of a file very conveniently with the **FileInput** component but this might not be enough for a file system.

An integration of **drawgantt** and **monika** to the main page would be a convenient feature as well.

5.4 Personal thoughts

One of the reasons we decided to choose this project was the web aspect of it. It seems an essential skill today but we are not practicing much in our classes. This project was a good opportunity to get our hands on **Javascript / Typescript** and **React** and we feel more confortable with the basics concepts now.