MAKERS ACADEMY, PORTFOLIO SUBMISSION

No. 1

Placement: Compare the Market
Apprentice: Krzysztof Kasprzak

Table of Contents

Placement background

A little bit on who I am and how I got here

My background and motivation to work in tech industry

As the title page suggests, my name is Krzysztof (you can read it as Kristof), and I am delighted to be a BGL Academy's apprentice. I managed to secure the apprentice role after deciding to stop my secondary teacher training.
As a relatively fresh maths graduate (I graduated with my first-class B.Sc. (Hons) in Mathematics in 2019), my previous position proved to be not challenging enough in terms of my personal development in STEM subjects.
I quickly found out that working in an environment where you don't get to learn new things, but rather regurgitate information and knowledge you find at most rudimentary, is not a place I want to stay in.
I set myself a challenge to find much more fascinating and stimulating career.
Software development was one of the options I was strongly considering. Having been exposed to basics of C++ programming and usage of maths orientated syntaxes like MATLAB or SAS during my time at university; I knew how engaging and ever-changing work in tech industry could be.

Joining BGL.

Nevertheless, knowing that my previous experience with programming was definitely not enough to secure a junior role without any form of prior training, I decided to apply for various software development/IT apprenticeships.

This way, I have initially filled and submitted only an interest form for the BGL's Academy apprenticeship programme, as according to the website at that time, intake for this year's cohort was closed.
To my surprise, I was soon contacted by the recruiter and invited for the initial assessment. My interest form made the right impact, and it was apparent that BGL was very happy with my academic background.

Later on, I was asked to complete a recruitment assignment for Makers Academy. It required me to learn basics of Ruby and attempting a simple online test on using the language.
It was a great experience; I was learning new things and felt gratified for doing so; something that further solidified my aspiration to get a career as a software engineer.
I have successfully passed this assessment and did well during my last panel interview for the position.
I was offered the role soon after, a role I am proud of!

The company and how I fit in it

BGL Academy

The academy is BGL Group's programme that oversees and lead apprenticeships, graduate schemes and training programmes for BGL's new and existing employees. Academy's main purpose is to aggregate talent and provide quality training for the employees to secure their continuous professional development and job satisfaction; consequently, providing means for a sustainable growth of the company.

BGL Group and Compare the Market

BGL Group specialises in digital distribution of insurance and household financial services.
More specifically, I am a proud meerkat.
That means, I work for the BGL's business which is a nation's beloved insurance comparison online service: Compare the Market (CtM).
Compare the Market strives to improve financial decision making for their customers by providing them with a simple to understand and comprehensive breakdowns and comparisons of variety of insurance products

(provided by BGL Group's partners). In addition, CtM provides household financial services such as easy comparison and switch of energy, broadband and credit card products.

My team, its purpose and where it belongs in the company

Which team I belong to

I belong to an Allspark team. It is a small team of four developers (including me), amongst which, Marc is Allspark's line manager who also oversees a sibling team of two other developers: Teletraan.
Both of the teams share the same business analyst (BA) and delivery manager (DM), as well as team rituals of daily stand-ups and sprint retros. Additionally, Allspark gets to work with Jose, the senior engineer in test who rotates between working with Allspark and Quoting Team.

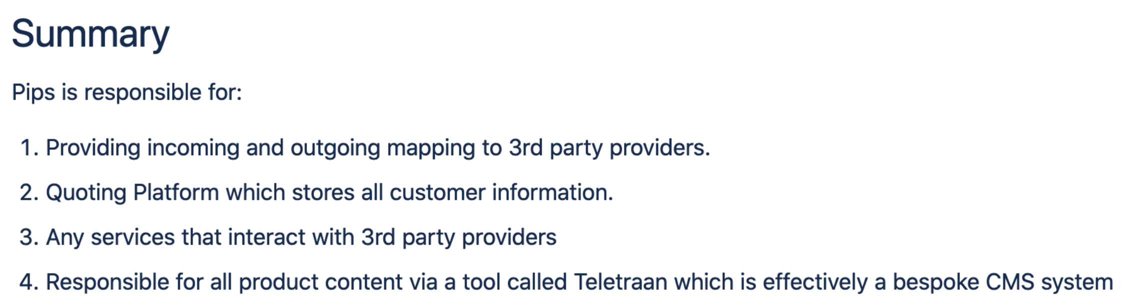
The purpose of Allspark

Both of the teams (Teletraan and Allspark) have their names coined from the main services they are developing and maintaining.
Allspark (the project which team I belong to maintains and continuously develops) is an internal user interface that allows its users to translate and edit schema (data) mappings created in xml files to json file format and vice versa. These mappings are used to carry information about insurance products that are being advertised at the main CtM website.
Translation from xml to json is a translation that takes information from provider and displays the relevant parts of it on the consumer end. The reverse operation happens when the providers want to receive data in xml format for the purposes of their own analysis/business practices.

Wider perspective

In a wider perspective, all four teams:
- Allspark,
- Teletraan,
- Quoting Team,
- Transformers,

Are making up to be a group called PIPS (Partner Integration and Pricing Services).
PIPS, however, is one of Compare the Market Technology teams.
As can be seen on the figure below (fig. 1.1), PIPS responsibilities can be summarised into four main parts, where the first one reflects the need for the Allspark service.



Summary

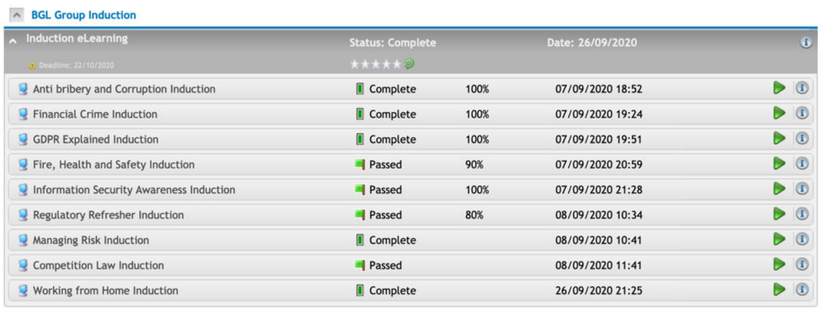
Pips is responsible for:

1. Providing incoming and outgoing mapping to 3rd party providers.
2. Quoting Platform which stores all customer information.
3. Any services that interact with 3rd party providers
4. Responsible for all product content via a tool called Teletraan which is effectively a bespoke CMS system

Fig. 1.1, A snapshot from a Confluence internal article that outlines PIPS responsibilities. Confluence is a company's standard tool that I and other employees use to share and use variety of technical guides/documentations.

Allspark’s ways of working

For the past year, team has been working fully remotely (due to COVID-!9 pandemic) and as I am writing the portfolio, there is no sign for that to change. In a future, however, decision has been made by company executives to cut down on office spaces (as the lockdown is continuedly showing that remote working does not impede the company’s outcomes in any way). With that, I am expected to work on and off site on a 50%-50% basis in a mid-long term.
With this implication, I was inducted how to work from home with compliance to health and safety guidelines (see fig. 1.2 below for evidence).



BGL Group Induction

Induction eLearning	Status: Complete		Date: 26/09/2020
Deadline: 22/10/2020	★★★★★		
Anti bribery and Corruption Induction	Complete	100%	07/09/2020 18:52
Financial Crime Induction	Complete	100%	07/09/2020 19:24
GDPR Explained Induction	Complete	100%	07/09/2020 19:51
Fire, Health and Safety Induction	Passed	90%	07/09/2020 20:59
Information Security Awareness Induction	Passed	100%	07/09/2020 21:28
Regulatory Refresher Induction	Passed	80%	08/09/2020 10:34
Managing Risk Induction	Complete		08/09/2020 10:41
Competition Law Induction	Passed		08/09/2020 11:41
Working from Home Induction	Complete		26/09/2020 21:25

Fig. 1.2, Is a snippet of introductory training I completed/passed as part of my onboarding. Amongst which the bottom module outlined how I should work from home, i.e., reminding me how I should set up an ergonomic working space, time intervals I should strive to work in or how to stay mentally fit while working remotely.

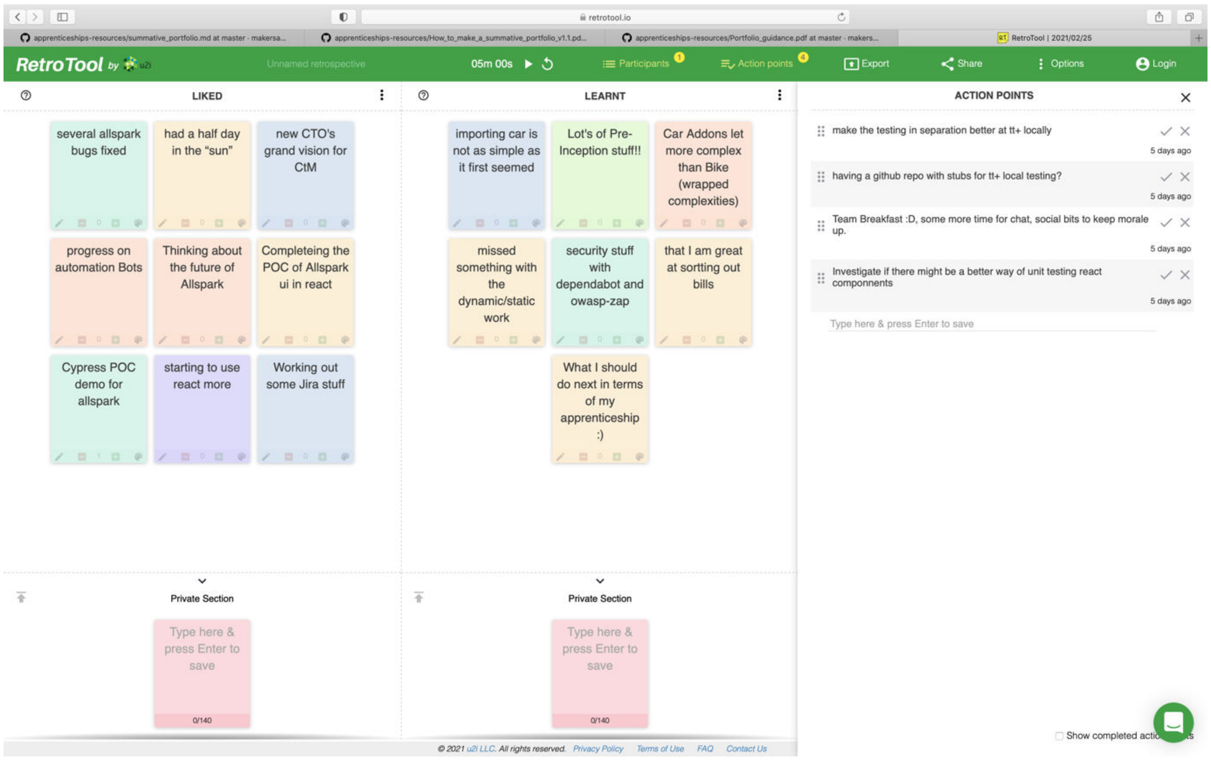
As fig. 1.2 also demonstrates, I have passed and understood all core business operational requirements. Amongst which ones that really are important to me as a software engineer are the ones regarding GDPR (General Data Protection Regulation), information security awareness or competition law, since the project I work on deals with managing providers product data.

The team works in Agile settings. I, together with others in Allspark work in two weeks long sprints. Every sprint finishes with a team retro during which everyone has a chance to reflect on emotional, technical (on a high level) and administrative aspects of team’s practice. Retro is normally led by a single person who is responsible for preparing board on which everyone can anonymously pin their reflections. Once the stage of posting reflections is over, the retro host is also responsible for going over the shared reflections and letting the participants to voice their opinions/experiences on each reflection submitted. Lastly, the host also sums up any actionable points from the retro for the BA (Nicola) to act upon.

It is a team custom to just alphabetically call out retro hosts, with that, I already had a chance to lead a retro. During my retro, I used a so-called Retro Tool website, which is a simple web application that allows users to create retro boards and share the mand update in real life so that it can be used in parallel to collect reflections. On figures 1.3 and 1.4 respectively (next page), you can see how I communicated with my team the relevant retro tool that I used as a host and a snapshot of results of the retro I lead, along with some actionable points I collected.

Krzysztof Kasprzak 10:04 AM
https://retrotool.io/MPpr_HkWJtZfspOOeod6b
retrotool.io
RetroTool - free, simple and flexible online retrospectives
A dead simple tool for running retrospective sessions. Create a board and invite your team to collaborate. (55 kB)

Fig.1.3, A snapshot of me sharing the retro tool just prior to starting the retro. I shared it using the common slack channel for the Allspark/Teletraan teams. Slack is a communication tool the I and others in my team use to communicate with each other. It is a most popular means of communication across the whole company, making it great also for networking with employees from a wider scope.



LIKED

- several allspark bugs fixed
- had a half day in the "sun"
- new CTO's grand vision for CtM
- progress on automation Bots
- Thinking about the future of Allspark
- Completeing the POC of Allspark ui in react
- Cypress POC demo for allspark
- starting to use react more
- Working out some Jira stuff

LEARNT

- importing car is not as simple as it first seemed
- Lot's of Pre-Inception stuff!!
- Car Addons let more complex than Bike (wrapped complexities)
- missed something with the dynamic/static work
- security stuff with dependabot and owasp-zap
- that I am great at sortting out bills
- What I should do next in terms of my apprenticeship :)

ACTION POINTS

- make the testing in separation better at tt+ locally — 5 days ago
- having a github repo with stubs for tt+ local testing? — 5 days ago
- Team Breakfast :D, some more time for chat, social bits to keep morale up. — 5 days ago
- Investigate if there might be a better way of unit testing react componnents — 5 days ago

Type here & press Enter to save
Private Section
Type here & press Enter to save
0/140
Private Section
Type here & press Enter to save
0/140
Show completed action points
© 2021 u2i LLC. All rights reserved. Privacy Policy Terms of Use FAQ Contact Us

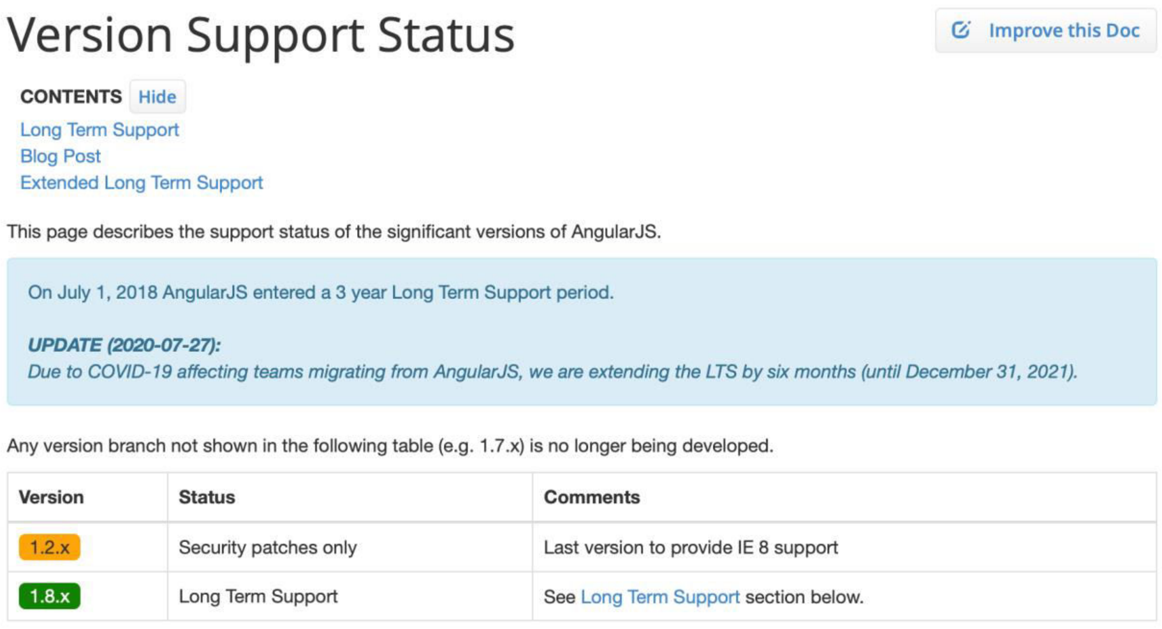
Fig. 1.4, A snapshot of a retro board I used to host a retro. Left-hand side presents to out of four columns participants were able to post their reflections and right-hand side shows the action points I collected.

Each sprint begins with planning phase during which, Allspark meets up together with BA and DM (Ellen), to put forward, point and priorities tickets for the next sprint.
Since the sprint duration spans over two weeks, at the end of first week of each sprint, there is a refinement session. During this half an hour meeting Allspark gathers to generally assess how feasible it is to complete all the planned activates and act accordingly to the current situation.
That is to for example scale down expectations of how many tasks will be completed or add in more from the back log, contacting other teams for aid etc.
I also attend non-compulsory social meetings that take place every other Thursday; these are led by Alex. It feels important for me be able to get along with my colleagues and have a sense of belonging in a workplace; this is a great way to enhance it.
Last, final piece of team rituals that occurs on a regular basis are daily stand-ups. These morning meetings are a chance for everyone to briefly inform others of what they currently working on, whether they would like to get some help with it or to report any blockers. It allows me and others to stay in a feedback loop and be

reassured that everyone is doing a relevant piece of work. On a plus side, it also allows for a bit of an informal chat. During pandemic and the era of remote working on a scale that never happened before, it really is vital. Every additional chance for a not work-related human interaction is good, especially as it solidifies team integrity.

Allspark's new, major task

As mentioned in *"The purpose of Allspark"*, it develops and maintains an internal UI (user interface) and the backend services that it consumes. However, the UI itself is in need of a rewrite. Being created in an old angular.js technology, it is not fit for further development and usage as the technology will come to the end of support with the end of year 2021 (see a snapshot of the relevant information on the fig. 1.5 below). Using the angular.js library beyond this point in time would mean potential security risks for the company and introduction of unmaintainable bugs and errors in the future.



Version Support Status

Improve this Doc

CONTENTS Hide

Long Term Support
Blog Post
Extended Long Term Support

This page describes the support status of the significant versions of AngularJS.

On July 1, 2018 AngularJS entered a 3 year Long Term Support period.

UPDATE (2020-07-27):
Due to COVID-19 affecting teams migrating from AngularJS, we are extending the LTS by six months (until December 31, 2021).

Any version branch not shown in the following table (e.g. 1.7.x) is no longer being developed.

Version	Status	Comments
1.2.x	Security patches only	Last version to provide IE 8 support
1.8.x	Long Term Support	See Long Term Support section below.

Fig. 1.5, a snapshot from official AngularJs documentation website that communicates the end of support date for the technology.

Hence, the major work that has to be carried out by me and my colleagues is to rewrite the frontend of the Allspark service using a newer library or framework.
Under the company's tech governance, angular 2 (the newer version of angular.js) is not recognised as recommended technology for new projects, instead React JS is. Additionally, React JS is widely used by other teams, making it a natural choice for coherency. Allspark will be rewritten using the React JS library. This is going to be my main focus during the placement.

Proof of concept task

The task

The JavaScript library in use

As briefly mentioned in *“Allspark’s new, major task”*, due to upcoming end date of official support for the framework in which the Allspark UI has been created, I and others in the team have to rewrite it into a new project using React JS library.
More precisely, React JS is a JavaScript library.
JavaScript (JS), in its own right, is a multi-paradigm, high-level programming language; along HTML and CSS, it is a core language that help creating web pages.
More specifically, JS allows for creation of dynamic web pages. Unlike purely HTML and CSS based web pages which are limited to reload each time one of its components has to change, pages written with JS allow for dynamic loading of specific components of the web page, without the need to reload it as a whole.
React JS, as a library of JS, provides developers with tools to efficiently build dynamic, component-based UI’s. Being component-based, means it prompts users to build web pages that consist of small, reusable and self-contained elements that can be displayed and interacted with.

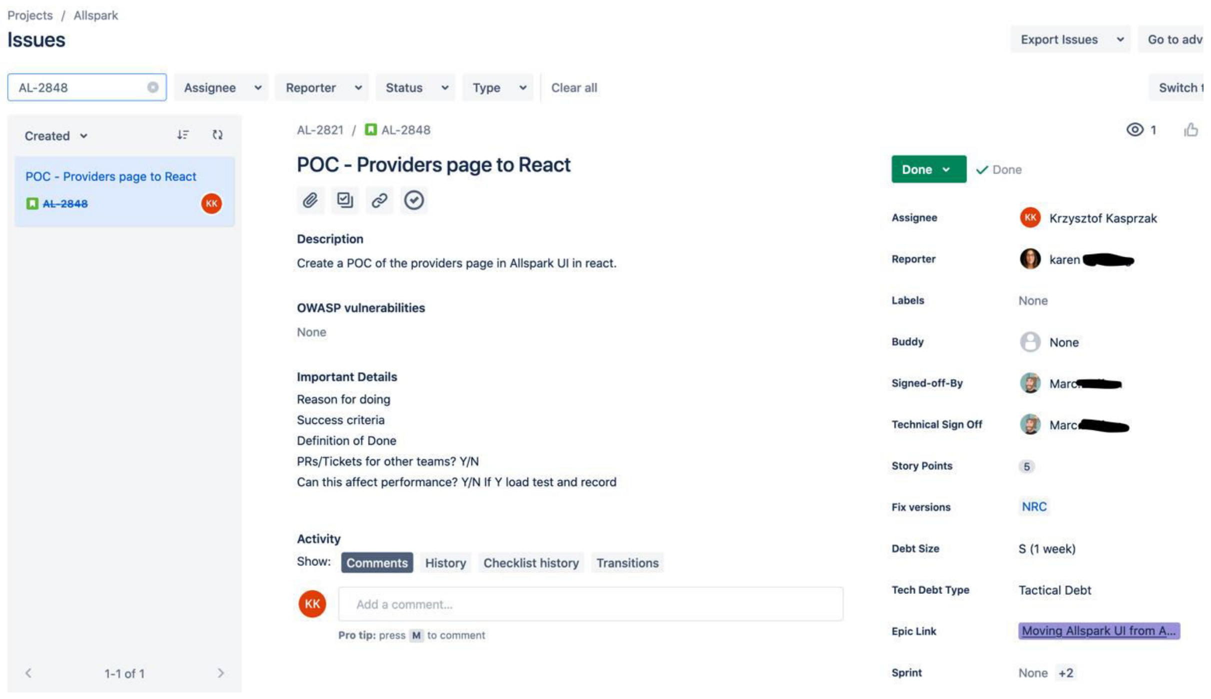
Proof of concept, what is it and why it is needed?

Proof of concept (a small-scale project that can be used as an evidence that a bigger task, using similar technique/resources is feasible) was required as it is a standard procedure (at least in the company) for tech teams to provide such pieces.
It is no surprise to me as it is a standard procedure; completion of POC (proof of concept) is essential to allow for exploration of possible ways the rewrite can happen and to aid planning of user stories to complete in the future. It allows business analysts to carry out analysis and pitch for potential support for the team on the basis of findings gathered upon the completion.

Initial completion requirements and how I got involved

Initial requirements

The Jira ticket (Jira is a scheduling tool that I and my colleagues use to follow and update track of progress during each sprint, it is created with agile management in mind, software development cycle that is employed across CtM Tech) was not very detailed (see fig 1.6 on the next page to see the task ticket description).


Projects / Allspark

Issues

Export Issues | Go to adv

AL-2848 | Assignee | Reporter | Status | Type | Clear all | Switch t

Created

POC - Providers page to React

~~AL-2848~~ KK

1-1 of 1

AL-2821 / AL-2848

POC - Providers page to React

Description

Create a POC of the providers page in Allspark UI in react.

OWASP vulnerabilities

None

Important Details

Reason for doing
Success criteria
Definition of Done
PRs/Tickets for other teams? Y/N
Can this affect performance? Y/N If Y load test and record

Activity

Show: Comments | History | Checklist history | Transitions

KK Add a comment...

Pro tip: press M to comment

Done ✓ Done

Assignee	KK Krzysztof Kasprzak
Reporter	karen [redacted]
Labels	None
Buddy	None
Signed-off-By	Marc[redacted]
Technical Sign Off	Marc[redacted]
Story Points	5
Fix versions	NRC
Debt Size	S (1 week)
Tech Debt Type	Tactical Debt
Epic Link	Moving Allspark UI from A...
Sprint	None +2

Fig. 1.6, as seen on the screen capture above, I was the assignee to complete the ticket. Additionally, it is worth to mention that the ticket did not clearly outline what are the criteria for its completion, leaving me and my pair partner Tim Heap with a great flexibility on what to do.

How I got involved

I expected to get involved with this task prior to the ticket being created.
Having been in contact with my line-manager Marc during the pre-placement bootcamp with Makers, he knew that I studied basics of React JS as part of it.
Furthermore, once I joined Allspark, I did express my interest in getting involved with it as at that point, I knew much more about react than C#, which Allspark uses to develop and maintain backend services that Allspark Ui consumes. I really wanted to make a positive impact in my team; I knew I could do that by participating in the POC as other developers in Allspark team haven't had as much exposure to React JS, the team at that point was even newer to the library than me.
It gave me opportunity to not only further practice my React JS skills, but also to share my already exiting knowledge of it.

Steps to completion of the task

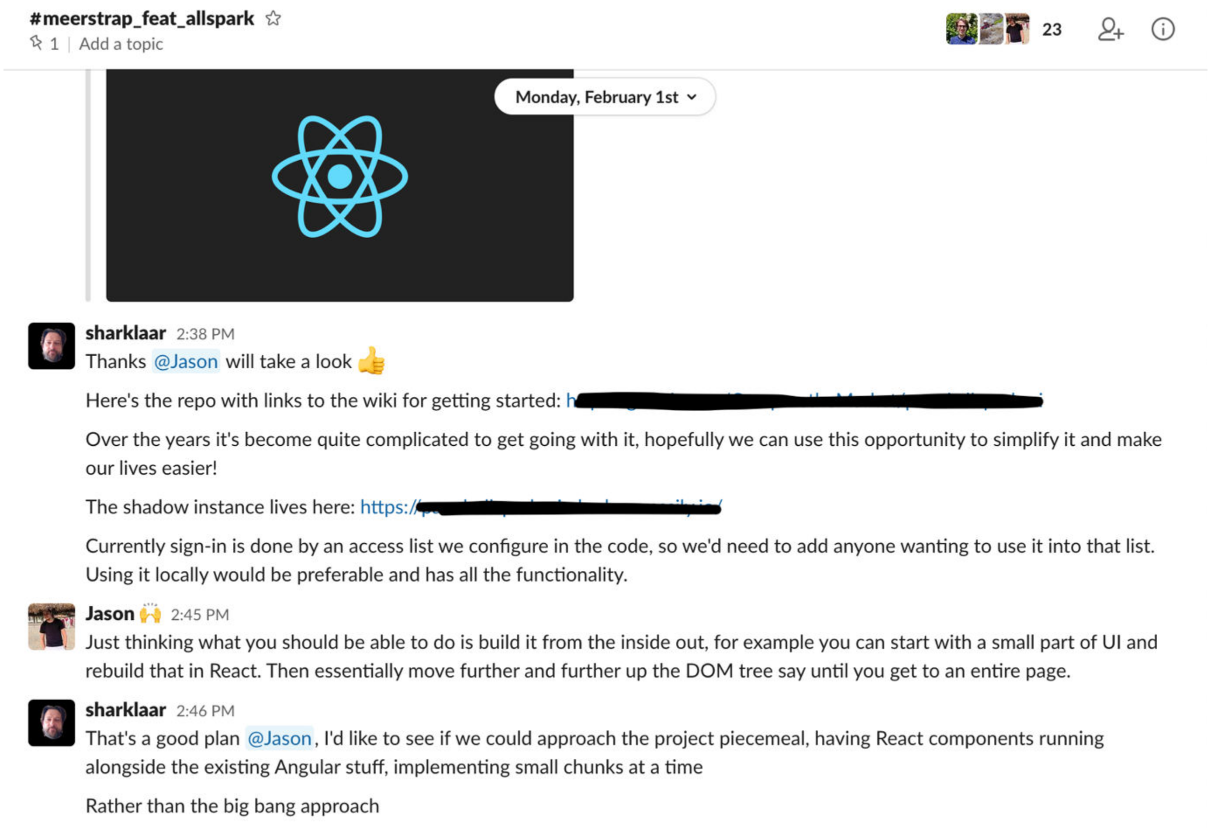
Support from another team

Prior to starting the task, it has been decided by my line-manager and DM that for the time of working on POC, me and my pair partner (Tim who is an experienced developer with over 15 years of experience) will be get aid from another CtM Tech team, Meerstrap.
Meerstrap is a team that creates React JS components that other CtM Tech teams can use so that there is consistency in the presentation (which is important to keep the presentation distinctive for the customers' recognition of brand) and code base amongst company's projects.

With that, they were a great support being so adept in working with the library.

First attempt to the task

The first direction in which I and my partner (Tim) was directed with this task, was an attempt to build some simple React JS components on top of the existing project, replacing angular.js UI.
The direction initially came from Jason, one of the members of Meerstrap team (see fig. 1.7 below for the communication detail).


#meerstrap_feat_allspark

1 | Add a topic

23

Monday, February 1st

sharklaar 2:38 PM
Thanks @Jason will take a look 👍

Here's the repo with links to the wiki for getting started: h[redacted]

Over the years it's become quite complicated to get going with it, hopefully we can use this opportunity to simplify it and make our lives easier!

The shadow instance lives here: https://[redacted]

Currently sign-in is done by an access list we configure in the code, so we'd need to add anyone wanting to use it into that list. Using it locally would be preferable and has all the functionality.

Jason 🙌 2:45 PM
Just thinking what you should be able to do is build it from the inside out, for example you can start with a small part of UI and rebuild that in React. Then essentially move further and further up the DOM tree say until you get to an entire page.

sharklaar 2:46 PM
That's a good plan @Jason, I'd like to see if we could approach the project piecemeal, having React components running alongside the existing Angular stuff, implementing small chunks at a time

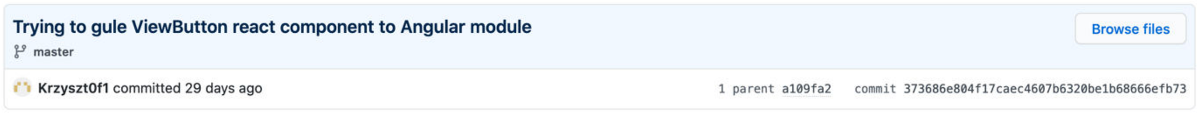
Rather than the big bang approach

Fig. 1.7, depicts decision agreed on what approach I needed to try first.

During this phase of work, I have been driving and navigating in pretty much 50% of time when working with Tim. Tim and I took decision to do so because, attempt to embed React components on top of existing angular project required the driver to be more fluent with angular than I was at the time. Additionally, I was a vital navigator as I was able to quickly navigate Tim with the implementations of React syntax wherever required. However, I really wanted to get some exposure in at least following the flow of how angula.js works (which I will need for further work, rewriting the UI to React), hence I was driving for the other half of the time at that stage of the POC.
This was also a great way of following generic best practice of pair programming, often switching drivers allowed Tim not to become to fatigued with it, maintaining mine and his focus levels.
At that stage, the main, most promising ways I tried to incorporate in the project, were the usages of two, popular external open-source libraries:

- ngReact
- react2angular

Both of these can be used to mount React components on to existing angular pages.
One defining difference is that ngReact is a library that is already archived and largely out of use (even the GitHub repository of the library mentions that the more modern solution is to use react2angular); the reason why I tried applying it was the fact that Allspark UI itself is a legacy code written in an old angular, something the library is technically suitable for.
Nevertheless, I tried applying both of the solutions following the steps outlined in their read.me files, i.e., installing the dependencies and implementing them in the code (see fig. 1.8 -2.1 on the next page).



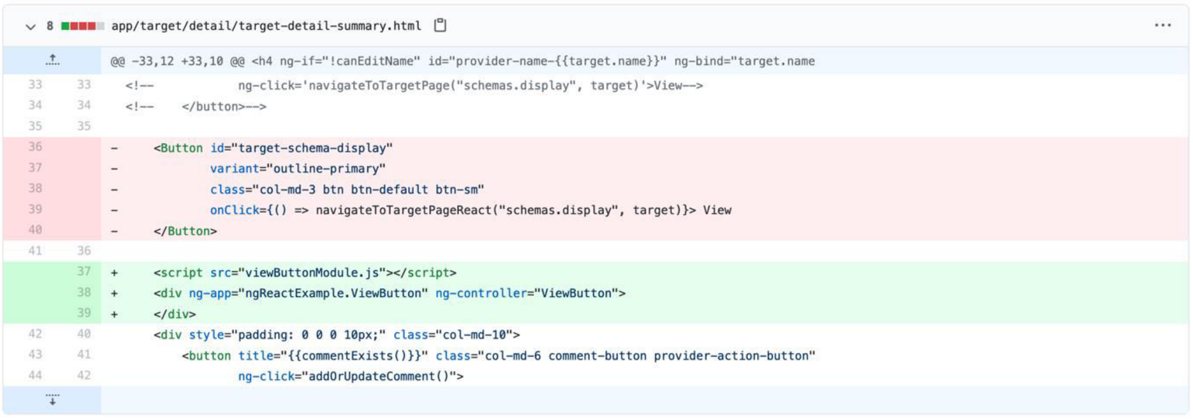
Trying to gule ViewButton react component to Angular module

master

Krzyszt0f1 committed 29 days ago

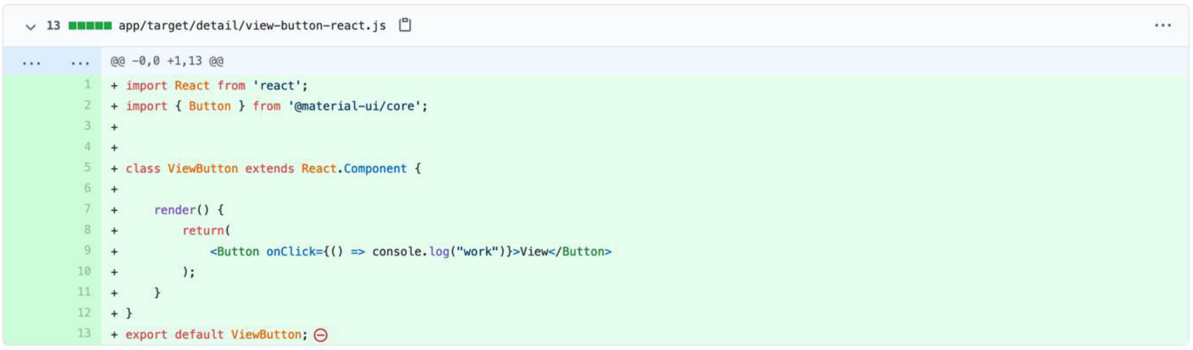
1 parent a109fa2 commit 373686e804f17caec4607b6320be1b68666efb73

Fig. 1.8, snippet that shows the title of commit that included an attempt of using ngReact to display a button on the angular page. It is worth to point out that here I forgot adhering to team's convention of naming GitHub commits. It should have included the ticket number at the front, in this case AL-2848.



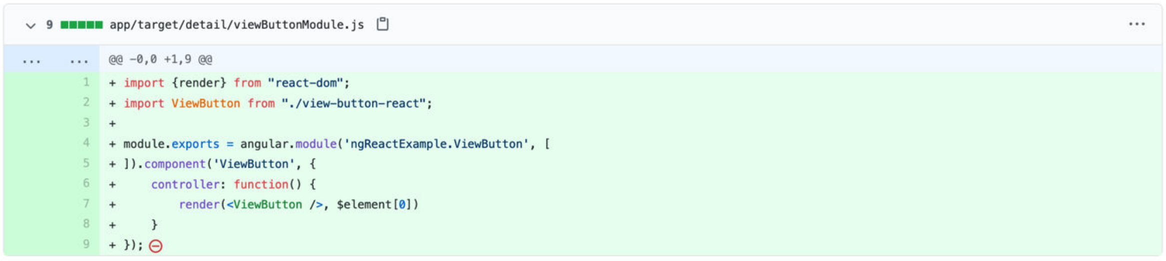
app/target/detail/target-detail-summary.html

```
@@ -33,12 +33,10 @@ <h4 ng-if="!canEditName" id="provider-name-{{target.name}}" ng-bind="target.name
  <!--          ng-click='navigateToTargetPage("schemas.display", target)'>View-->
  <!--     </button>-->

-     <Button id="target-schema-display"
              variant="outline-primary"
              class="col-md-3 btn btn-default btn-sm"
-             onClick={() => navigateToTargetPageReact("schemas.display", target)}> View
-     </Button>

+     <script src="viewButtonModule.js"></script>
+     <div ng-app="ngReactExample.ViewButton" ng-controller="ViewButton">
+     </div>
      <div style="padding: 0 0 0 10px;" class="col-md-10">
          <button title="{{commentExists()}}" class="col-md-6 comment-button provider-action-button"
                  ng-click="addOrUpdateComment()">
```



app/target/detail/view-button-react.js

```
@@ -0,0 +1,13 @@
+ import React from 'react';
+ import { Button } from '@material-ui/core';
+
+
+ class ViewButton extends React.Component {
+
+     render() {
+         return(
+             <Button onClick={() => console.log("work")}>View</Button>
+         );
+     }
+ }
+ export default ViewButton;
```



app/target/detail/viewButtonModule.js

```
@@ -0,0 +1,9 @@
+ import {render} from "react-dom";
+ import ViewButton from "./view-button-react";
+
+ module.exports = angular.module('ngReactExample.ViewButton', [
+ ]).component('ViewButton', {
+     controller: function() {
+         render(<ViewButton />, $element[0])
+     }
+ });
```

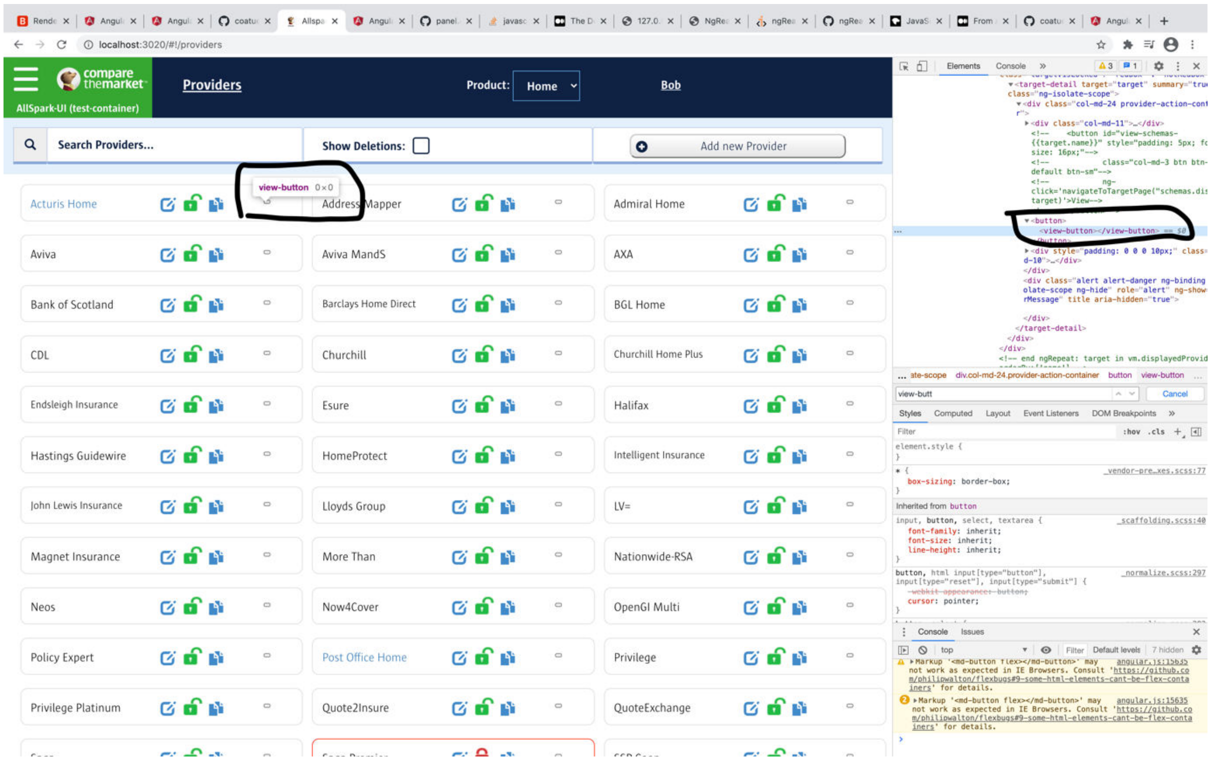
Fig. 1.9 and 2.1, these show the changes I made while driving in order to attempt and incorporate react into angular html view file. As can be seen, in the `target-detail-summary.html` to mount the react component it was required to link it to the script from `viewButtonMojule.js` written in plain JS with a use of angular module. The component itself (`view-button-react.js`) is written as react class in react jsx syntax.

Writing this snippet, I kept it in accordance with generic and team's agreed best practices (I obeyed jshint linting tool), for example, as per agreed linting tool, I kept the code lines short so you to view it all you don't have to scroll to sides, used `Button` styling from Material UI, a library that was recommended to me by colleagues form Teletraan which is deemed safe to use. I also have located the files in a same directory that is responsible for the detail button, file structuring that is widely agreed upon in both, Teletraan and Allspark teams. The button, for the purposes of if the initial trial was made a simple component that only logs string to console, the first obstacle was to make it appear.

I kept the react component as a class there for the ease of communication with my pair partner, at that stage Tim was more accustomed with creating react components as classes, not headless functions, I made the switch further along the line; functions are generally regarded as a better practice because a react class is a function behind the scenes anyways, that's due to JavaScript being mainly prototype-based language. With that, functions work generally faster. Additionally, when it comes to more involved components, functions have better readability and can perform pretty much all the operations that a class component can thanks to the use of react hooks.

Results of the first attempt

In turn, this implementation (as well as react2angular which was almost identically, just with the use of different, updated library) brought some interesting results. Namely, the button was visible in the source html when accessing the Allspark UI. However, it was being rendered as a 0 by 0 pixels element (as can be seen on the fig. 2.2 below)



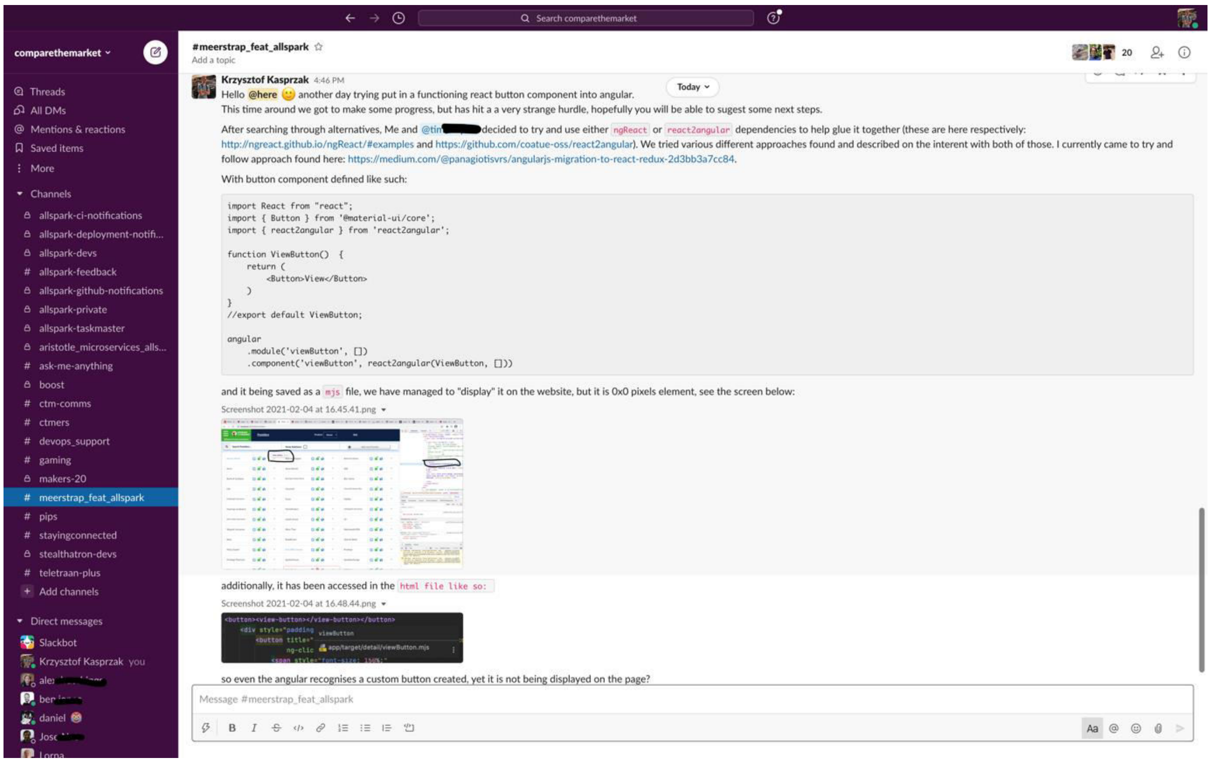
```
▾<target-detail target="target" summary="tru
class="ng-isolate-scope">
  ▾<div class="col-md-24 provider-action-con
  r">
    ▸<div class="col-md-11">…</div>
     <!--    <button id="view-schemas-
     {{target.name}}" style="padding: 5px; f
     size: 16px;"-->
     <!--              class="col-md-3 btn btn-
     default btn-sm"-->
     <!--              ng-
     click='navigateToTargetPage("schemas.di
     target)'>View-->
    ▾<button>
       <view-button></view-button> == $0
     </button>
    ▸<div style="padding: 0 0 0 10px;" class
    d-10">…</div>
   </div>
   <div class="alert alert-danger ng-binding
   olate-scope ng-hide" role="alert" ng-show
   rMessage" title aria-hidden="true">

   </div>
  </target-detail>
 </div>
</div>
<!-- end ngRepeat: target in vm.displayedProvid
```

```
... ate-scope  div.col-md-24.provider-action-container  button  view-button
view-butt

element.style {
}
* {                                        _vendor-pre…xes.scss:77
  box-sizing: border-box;
}
Inherited from button
input, button, select, textarea {           _scaffolding.scss:40
  font-family: inherit;
  font-size: inherit;
  line-height: inherit;
}
button, html input[type="button"],          _normalize.scss:297
input[type="reset"], input[type="submit"] {
  ~~-webkit-appearance: button;~~
  cursor: pointer;
}
```

```
▸Markup '<md-button flex></md-button>' may     angular.js:15635
not work as expected in IE Browsers. Consult 'https://github.co
m/philipwalton/flexbugs#9-some-html-elements-cant-be-flex-conta
iners' for details.
2 ▸Markup '<md-button flex></md-button>' may   angular.js:15635
not work as expected in IE Browsers. Consult 'https://github.co
m/philipwalton/flexbugs#9-some-html-elements-cant-be-flex-conta
iners' for details.
```

Fig. 2.2, shows the outcome of attempted changes, page renders swapped, react button as a 0x0 pixels element.

After consulting this peculiar occurrence with Meerstrap colleagues (see fig. 2.3 on the next page to find how I communicated this expressing technical detail in relevant depth), we came to the conclusion that it is happening because the angular.js html view file does not recognise react component written in jsx (React's native syntax) rather it sees it as a nonsense piece written in JavaScript.
It was most likely due to how complex and potentially unfit for this approach the project is. Tim or I couldn't find a well-documented way to employ these libraries when working on a project that is being spun up in a container and uses grunt to configure it.



Krzysztof Kasprzak 4:46 PM
Hello @here 🙂 another day trying put in a functioning react button component into angular.
This time around we got to make some progress, but has hit a a very strange hurdle, hopefully you will be able to sugest some next steps.

After searching through alternatives, Me and @tim[redacted]decided to try and use either `ngReact` or `react2angular` dependencies to help glue it together (these are here respectively: http://ngreact.github.io/ngReact/#examples and https://github.com/coatue-oss/react2angular). We tried various different approaches found and described on the interent with both of those. I currently came to try and follow approach found here: https://medium.com/@panagiotisvrs/angularjs-migration-to-react-redux-2d3bb3a7cc84.

With button component defined like such:

```
import React from "react";
import { Button } from '@material-ui/core';
import { react2angular } from 'react2angular';

function ViewButton()  {
    return (
        <Button>View</Button>
    )
}
//export default ViewButton;

angular
    .module('viewButton', [])
    .component('viewButton', react2angular(ViewButton, []))
```

and it being saved as a `mjs` file, we have managed to "display" it on the website, but it is 0x0 pixels element, see the screen below:

Screenshot 2021-02-04 at 16.45.41.png

additionally, it has been accessed in the `html file like so:`

Screenshot 2021-02-04 at 16.48.44.png

```
<button><view-button></view-button></button>
    <div style="padding   viewButton
        <button title="
            ng-clic   app/target/detail/viewButton.mjs
              <span style="font-size: 150%;"
```

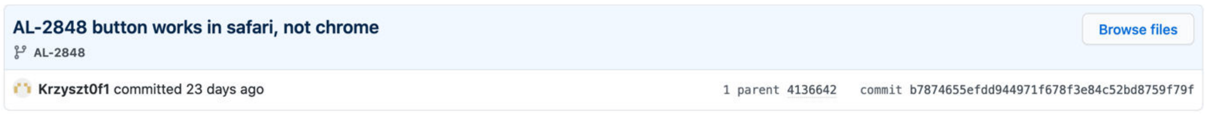
so even the angular recognises a custom button created, yet it is not being displayed on the page?

Fig. 2.3, Example of how I can communicate technical matters with stakeholders with a relevant level of detail.

With it, together with Tim, I made a decision that some other approach should be tried.

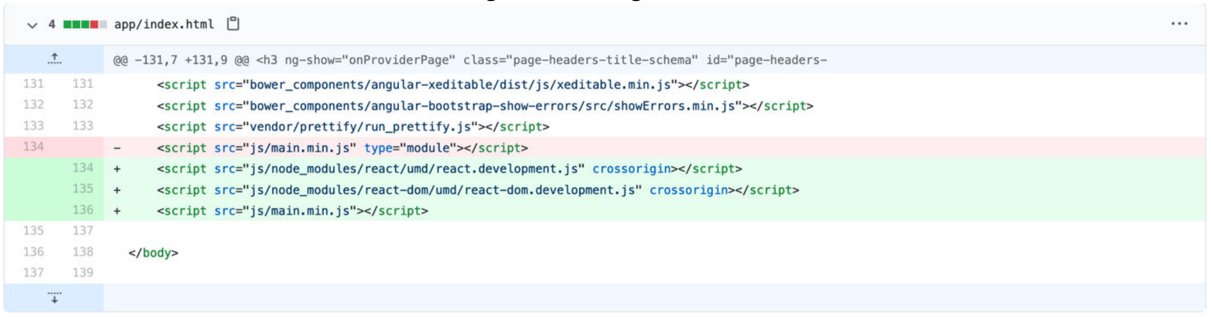
Second attempt

In the light of what has happened, I decided to try and step back even further, simplifying the problem even more for the time being (scaling and solving one problem at the time). Namely I decided that first thing I shall try is to make any react component visible and try to do it by writing it directly in the html view by wrapping it in a script tag. I have done by making the following changes (see fig. 2.4 to 2.8 that presents changes made and describes them briefly):



AL-2848 button works in safari, not chrome
AL-2848
Krzyszt0f1 committed 23 days ago
1 parent 4136642   commit b7874655efdd944971f678f3e84c52bd8759f79f

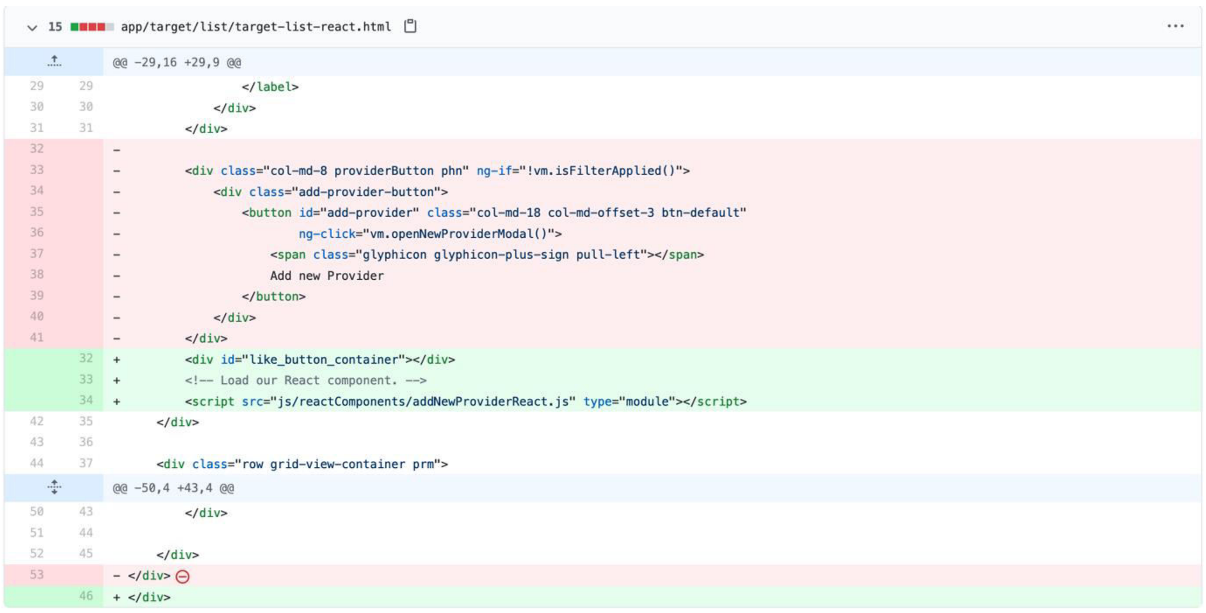
Fig. 2.4, shows how throughout the duration of the project I solidified the correct way of labelling commits according to team's agreed standards.



4 app/index.html

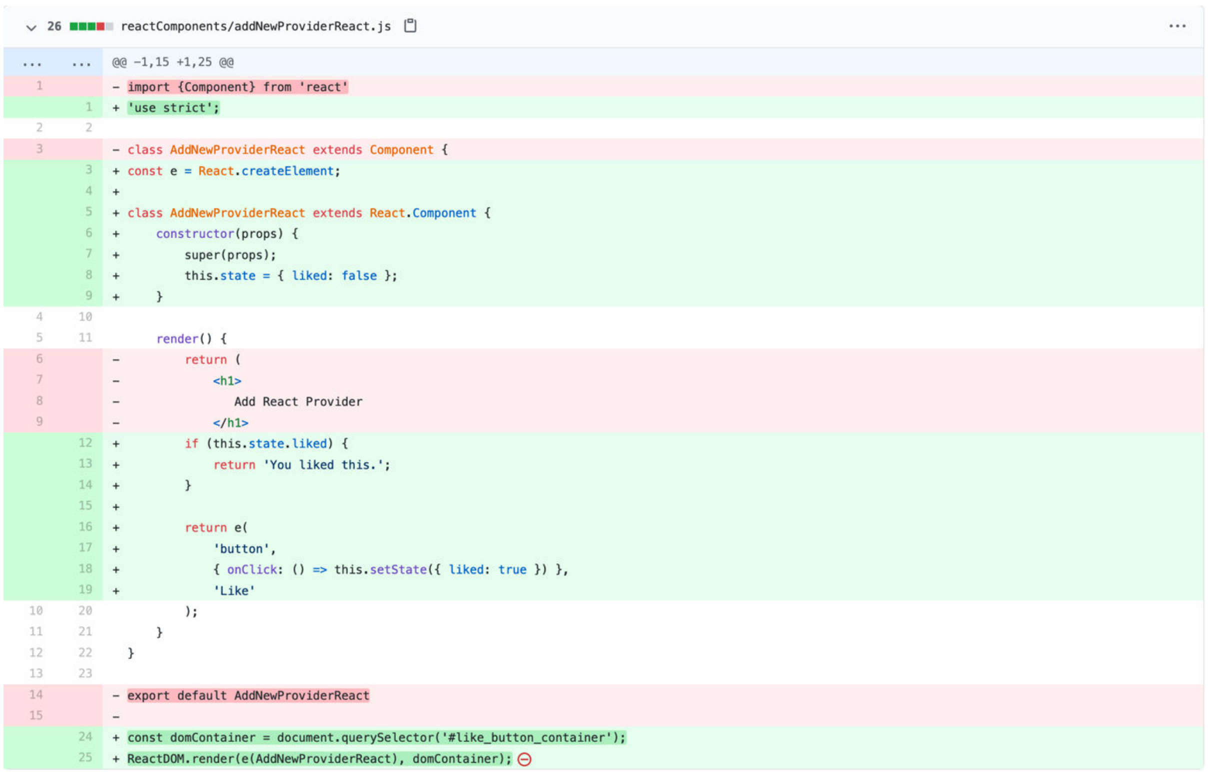
```
@@ -131,7 +131,9 @@ <h3 ng-show="onProviderPage" class="page-headers-title-schema" id="page-headers-
        <script src="bower_components/angular-xeditable/dist/js/xeditable.min.js"></script>
        <script src="bower_components/angular-bootstrap-show-errors/src/showErrors.min.js"></script>
        <script src="vendor/prettify/run_prettify.js"></script>
-       <script src="js/main.min.js" type="module"></script>
+       <script src="js/node_modules/react/umd/react.development.js" crossorigin></script>
+       <script src="js/node_modules/react-dom/umd/react-dom.development.js" crossorigin></script>
+       <script src="js/main.min.js"></script>

    </body>

```

Fig. 2.5, shows how I wrapped the logic from react components directly into the main index.html, avoiding the need to use intermediate libraries such as react2angular or ngReact.



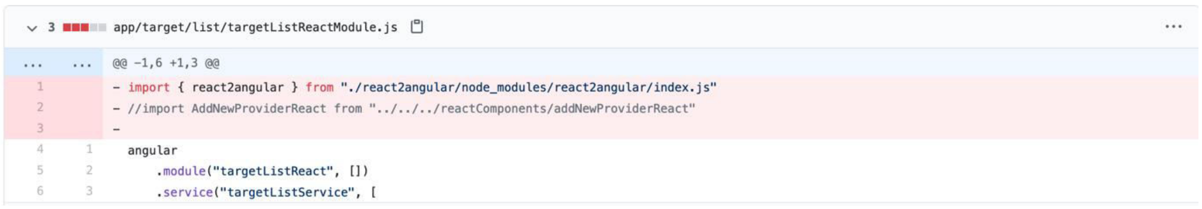
```
15  app/target/list/target-list-react.html

@@ -29,16 +29,9 @@
                    </label>
                </div>
            </div>
-
-           <div class="col-md-8 providerButton phn" ng-if="!vm.isFilterApplied()">
-               <div class="add-provider-button">
-                   <button id="add-provider" class="col-md-18 col-md-offset-3 btn-default"
-                           ng-click="vm.openNewProviderModal()">
-                       <span class="glyphicon glyphicon-plus-sign pull-left"></span>
-                       Add new Provider
-                   </button>
-               </div>
-           </div>
+           <div id="like_button_container"></div>
+           <!-- Load our React component. -->
+           <script src="js/reactComponents/addNewProviderReact.js" type="module"></script>
        </div>

        <div class="row grid-view-container prm">
@@ -50,4 +43,4 @@
            </div>

        </div>
- </div>
+ </div>
```

Fig. 2.6, similarly simplifying implementation of the component in the exact angular view html file.



```
26  reactComponents/addNewProviderReact.js

@@ -1,15 +1,25 @@
- import {Component} from 'react'
+ 'use strict';

- class AddNewProviderReact extends Component {
+ const e = React.createElement;
+
+ class AddNewProviderReact extends React.Component {
+     constructor(props) {
+         super(props);
+         this.state = { liked: false };
+     }

      render() {
-         return (
-             <h1>
-                 Add React Provider
-             </h1>
+         if (this.state.liked) {
+             return 'You liked this.';
+         }
+
+         return e(
+             'button',
+             { onClick: () => this.setState({ liked: true }) },
+             'Like'
          );
      }
  }

- export default AddNewProviderReact
-
+ const domContainer = document.querySelector('#like_button_container');
+ ReactDOM.render(e(AddNewProviderReact), domContainer);
```

Fig. 2.7, I kept the component in a form of a class but, used the most legacy way to define a component in hopes that it will render it directly without a need to use jsx.



```
3 app/target/list/targetListReactModule.js
@@ -1,6 +1,3 @@
- import { react2angular } from "./react2angular/node_modules/react2angular/index.js"
- //import AddNewProviderReact from "../../../reactComponents/addNewProviderReact"
-
  angular
      .module("targetListReact", [])
      .service("targetListService", [
```

Fig. 2.8, I also simplified the angular module which didn't need to use external library anymore.

With the changes made, it actually worked. Defining the react component not using jsx made the button functional (with its simplistic logic) and visible (see fig. 2.9 below).

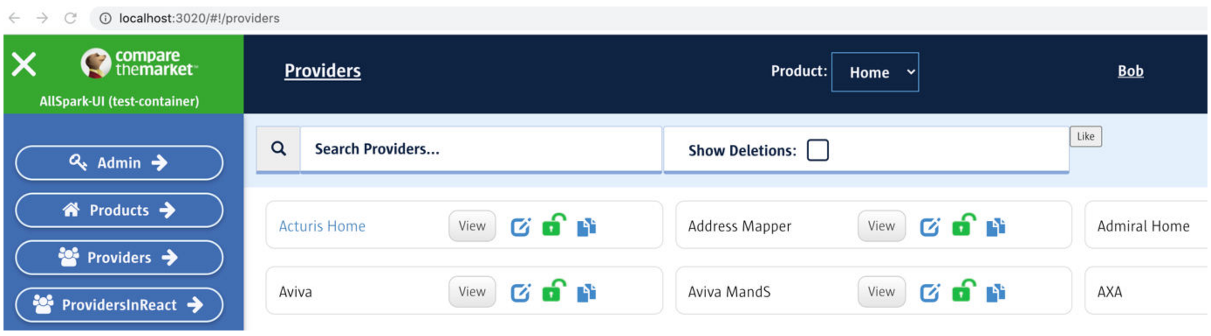


Fig. 2.9, In the right top corner of the interface is the successful implementation of using pure react injection into html.

Results of the second attempt

In turn, this seeming success has fuelled a design decision that potentially, it would be able to rewrite the project from top to bottom, element by element until there would be enough react components that migrating it onto a single, stand-alone react project. Especially that during this work-around, I have centralised the component in. new directory in which intentionally all react elements would be stored.
Unfortunately, this was only a dead end. Next, natural step from having everything injected with react like in the example above, would be to re-write it with the use of jsx. For that, in order to still have elements injected and being read as react, I needed to use babel library. Tool that allows to translate jsx format of React JS into browser friendly JavaScript.
This has not panned out. Tim and I could not make it happen, after discussing it further with Meerstrap team colleagues, conclusion was reached; it was because to include babel and have it working, the project would need to be created as a React one. Eventually, it could have been injected straight into the html elements via scripts, however that would be:

- Not scalable, having to retrieve all the elements embedded in it would not make it easier to later migrate it into stand-alone react project,
- Not efficient, babel needs to "compile" jsx into JS, it is capable of doing it in real time when injected but, it would hit the user experience along the time when Allspark UI would be in the process of rewriting.

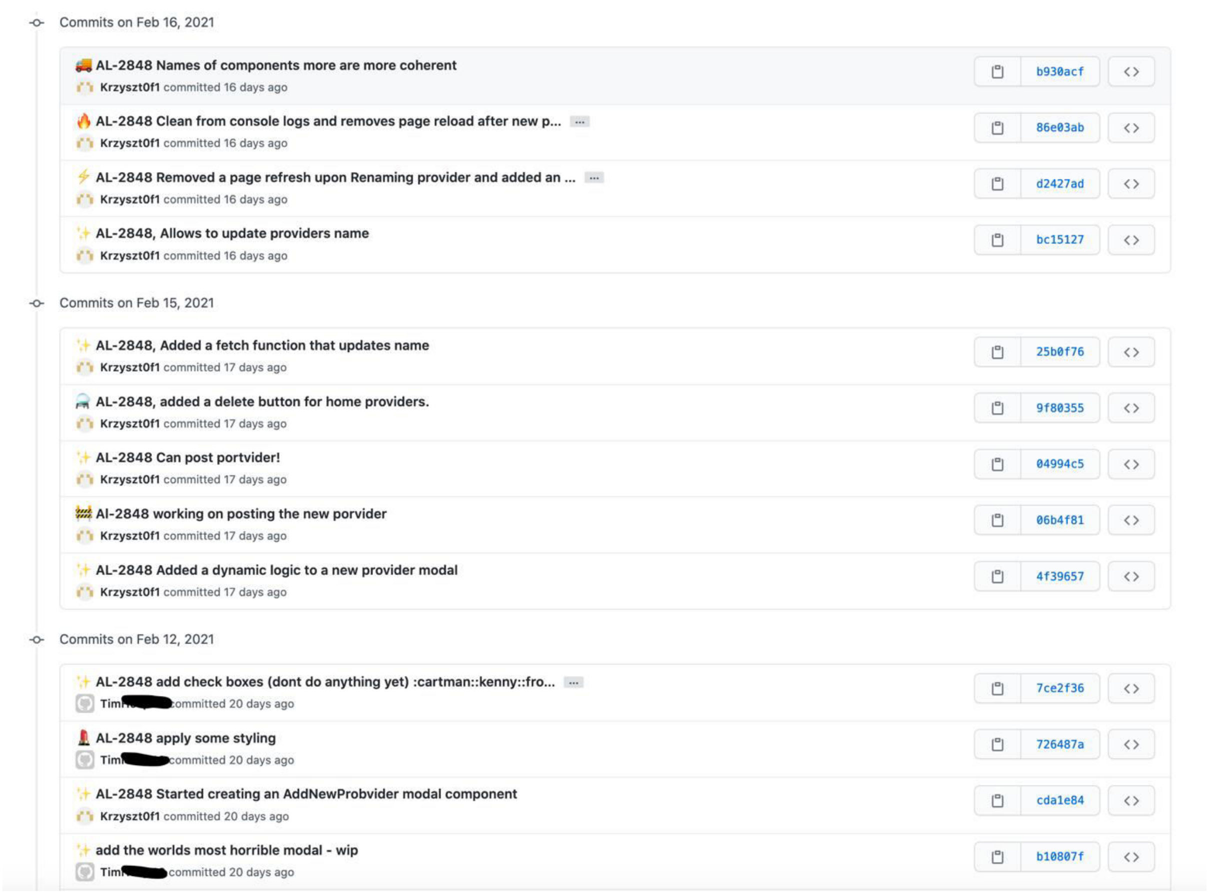
With this problem explored, Meerstrap team offered to try and create a custom, new environment in which angular and react sites of projects would run. I accompanied them as we mobbed plenty during this time, Tim and I were on hand to help them navigate through the existing project; it is a complicated and legacy structure. They came up with a potential way of creating angular and react app hybrid which runs both of those in parallel.
Yet again, the structure of existing project proved to be too convoluted to do it in a feasible way.

This has concluded that both, POC and the future version of Allspark UI will have to be created from scratch as React JS projects.

Final straight, my initiative

After all the previous work, having discovered a way forward Tim and I started creating the POC from scratch. For the purpose of quick and convenient development set up, I decided that `create-react app` command should be used. It is a boiler-plate command that allows for an easy creation of react project, along with a skeleton of repository structure.
The command, same as the whole library was created by Facebook, it is possibly the most popular way to set up react projects.

As the entire project focused more on spiking the possibility of creating a new react app, and already discovering that the most feasible way forward would be to just create it all anew, I was not required to carry out any automated testing, not even on the unit testing level; all tests I carried out were just manual feature tests.
For this part of work, I was the one to carry out most of driving (see fig. 3.1 below). It came as Tim expressed being not confident enough with react syntax; it was not a problem for me. I actually was happy to be able to share what I know about React JS and to familiarise him with function focused style of creating components.



Commits on Feb 16, 2021

AL-2848 Names of components more are more coherent
Krzyszt0f1 committed 16 days ago — b930acf

AL-2848 Clean from console logs and removes page reload after new p... …
Krzyszt0f1 committed 16 days ago — 86e03ab

AL-2848 Removed a page refresh upon Renaming provider and added an ... …
Krzyszt0f1 committed 16 days ago — d2427ad

AL-2848, Allows to update providers name
Krzyszt0f1 committed 16 days ago — bc15127

Commits on Feb 15, 2021

AL-2848, Added a fetch function that updates name
Krzyszt0f1 committed 17 days ago — 25b0f76

AL-2848, added a delete button for home providers.
Krzyszt0f1 committed 17 days ago — 9f80355

AL-2848 Can post portvider!
Krzyszt0f1 committed 17 days ago — 04994c5

Al-2848 working on posting the new porvider
Krzyszt0f1 committed 17 days ago — 06b4f81

AL-2848 Added a dynamic logic to a new provider modal
Krzyszt0f1 committed 17 days ago — 4f39657

Commits on Feb 12, 2021

AL-2848 add check boxes (dont do anything yet) :cartman::kenny::fro... …
Tim[redacted]committed 20 days ago — 7ce2f36

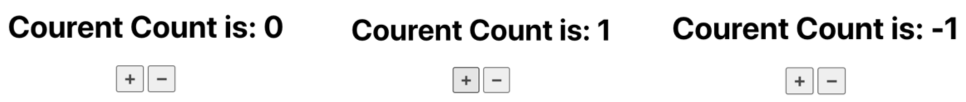
AL-2848 apply some styling
Tim[redacted]committed 20 days ago — 726487a

AL-2848 Started creating an AddNewProbvider modal component
Krzyszt0f1 committed 20 days ago — cda1e84

add the worlds most horrible modal - wip
Tim[redacted]committed 20 days ago — b10807f

Fig. 3.1, shows that from the point of deciding to create the POC purely in React JS, I was the main driver.

For example, I was able to explain to him how `useState` hook works with react functions. As with the simple example of a counter app I created to demonstrate my explanation here (see fig. 3.2 and 3.3 on the next page).


Courent Count is: 0

Courent Count is: 1

Courent Count is: -1

Fig. 3.2 shows the result of my simple example I wanted to use here to show my understanding of react `useState` hook I shared with Tim. The initial state of count is set to 0, after clicking `plus` it changes to one, and if `minus` is clicked twice it goes down to negative 1.

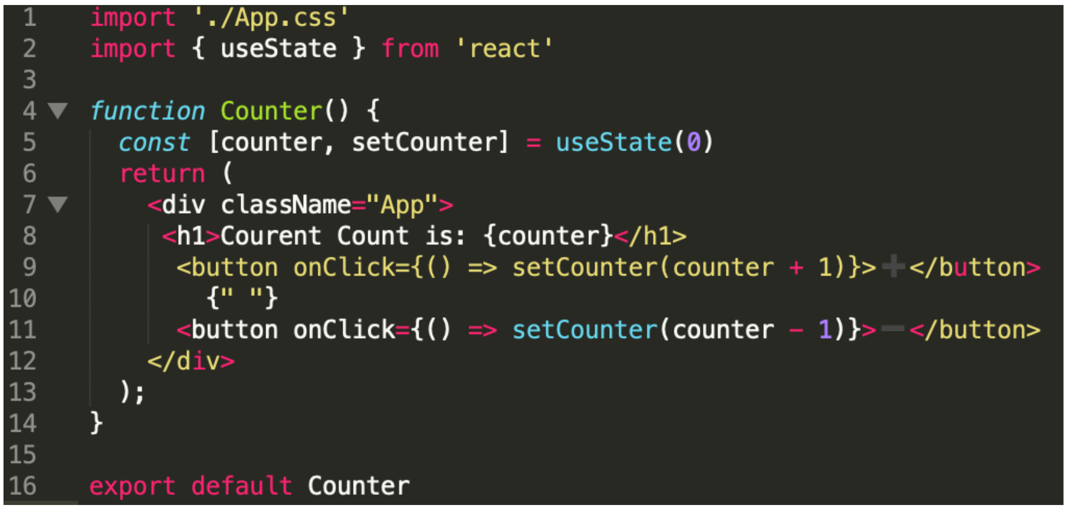

```
1   import './App.css'
2   import { useState } from 'react'
3
4   function Counter() {
5     const [counter, setCounter] = useState(0)
6     return (
7       <div className="App">
8        <h1>Courent Count is: {counter}</h1>
9         <button onClick={() => setCounter(counter + 1)}>+</button>
10          {" "}
11        <button onClick={() => setCounter(counter - 1)}>−</button>
12      </div>
13    );
14  }
15
16  export default Counter
```

Fig. 3.3, shows a simple example of a function react component that uses `useState`. What happens is that on line 5 of the code I initialised a counter as a variable whose state can be changed by `setCounter` function. To be able to use the hook, I had to import it from react library on line 2.
The `useState(0)` means that I initially want the counter to equal to 0.
Lines 9 to 11 are just implementations of the `useState` hook, it is being triggered and used on every click of `plus` or ‘minus` buttons. I did not use here any semicolons because my IDE is set to use linting that adheres to the team’s agreed code convention.

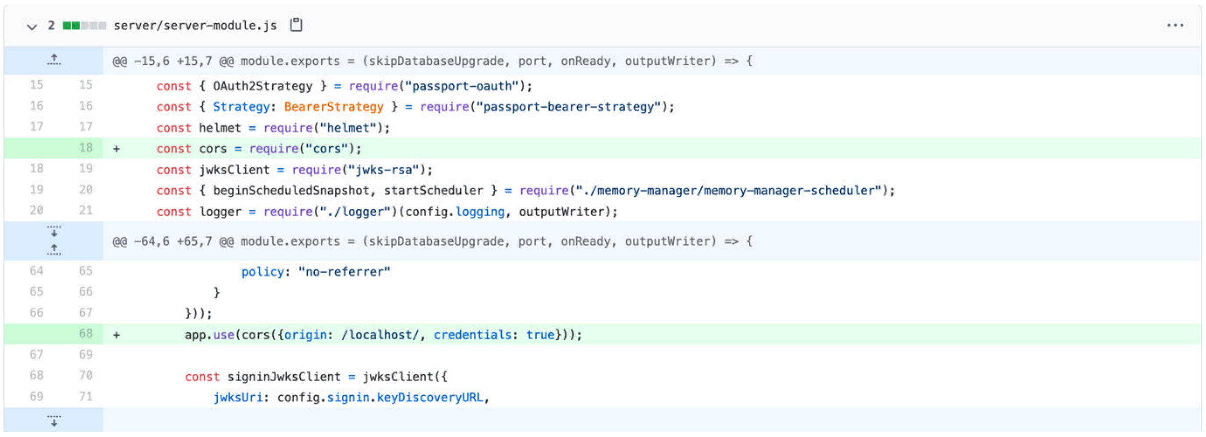
useState hook in more detail, encapsulation in React

As per Object Orientated Programming (OOP) definition of encapsulation, it is a concept of restricting a direct access to certain components of an object. This in fact, ties to general best practice of how to handle state changes in react functional components. It is advised that when developing react functional components (as the example I used to share the knowledge of react hooks on the fig. 3.3 above), any of the react component state, that shall be modified by interacting with, should not be mutated directly. Direct mutation of the `counter` value could result in unexpected value assignment or a loss of value increment/decrement at all. That’s because if the `counter` would be modified directly, i.e. `onClick={() => counter++}`, this would trigger a change of state, forcing the component to re-render. However, the increment could be forgiven, as by default component’s state is being cleaned at re-render unless wrapped in the `useState()`. This is by design as uncleared state could for example get incremented more less/than once, depending on how the operations in the browser would get queued.
Hence, `useState()` allows for the state to be carried over to the next re-render and queues it as desired, ultimately encapsulating the react’s component state away from direct manipulation.

CRUD functionality and the need to adjust cross origin accessibility

Upon determining what should be done for the POC to be complete with the line manager, it was clear that all that I needed to create was a simple example of CRUD app that, similarly as an existing Allspark UI, creates, reads, updates and deletes data (CRUD) that is being consumed by the frontend. It was a great way for me to practice operating these kinds of operations and executing them on data via API requests.
However, prior to writing any code, it was essential to adjust the existing Allspark UI code.

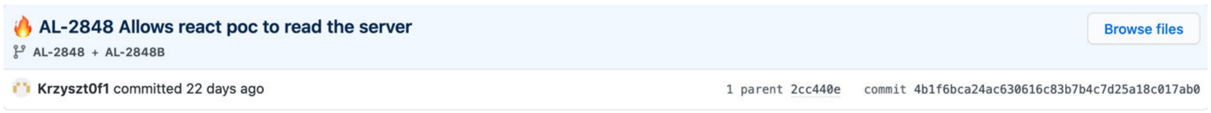
Allspark UI has been written in such way that it encompasses frontend written in angular.js and backend server side written using node.js (a runtime environment that allows to use JavaScript outside of any browser).
The server side then connects to database which is kept using Mongo DB and mongoose JavaScript plugin. This allows for the use of non-tabular (NoSQL) data structures.
With that, to minimise the amount of work needed, Tim and I have decided that the most feasible way to create the POC would be to adjust the cross-origin access defined in the server part of existing Allspark UI to allow the POC access to it (see fig. 3.4 and 3.5 below that shows changes I put into the server configuration to allow POC access it).



```
2 server/server-module.js

@@ -15,6 +15,7 @@ module.exports = (skipDatabaseUpgrade, port, onReady, outputWriter) => {
15  15      const { OAuth2Strategy } = require("passport-oauth");
16  16      const { Strategy: BearerStrategy } = require("passport-bearer-strategy");
17  17      const helmet = require("helmet");
    18  +   const cors = require("cors");
18  19      const jwksClient = require("jwks-rsa");
19  20      const { beginScheduledSnapshot, startScheduler } = require("./memory-manager/memory-manager-scheduler");
20  21      const logger = require("./logger")(config.logging, outputWriter);
@@ -64,6 +65,7 @@ module.exports = (skipDatabaseUpgrade, port, onReady, outputWriter) => {
64  65                  policy: "no-referrer"
65  66              }
66  67          }));
    68  +       app.use(cors({origin: /localhost/, credentials: true}));
67  69
68  70          const signinJwksClient = jwksClient({
69  71              jwksUri: config.signin.keyDiscoveryURL,
```

Fig. 3.4, shows the simple changes I did to allow POC to access the server of existing Allspark UI. I just had to import a new dependency on line 17, that allows for specifying the cross-origin resource sharing (CORS) settings. Then on line 68, It was just a case of allowing the access for all the page of type `localhost` in this case.

Solution of allowing `localhost` pages to access the server is only temporary and applicable only for the POC, as it was only required to run locally. However, this has prompted a decision that for the full-fledged rewrite, will act similarly. That is, the react project will be a purely frontend piece that will be consuming a separated-out server side of Allspark UI (being written in node.js, makes it not affected by angular.js going out of use) to connect with the database.

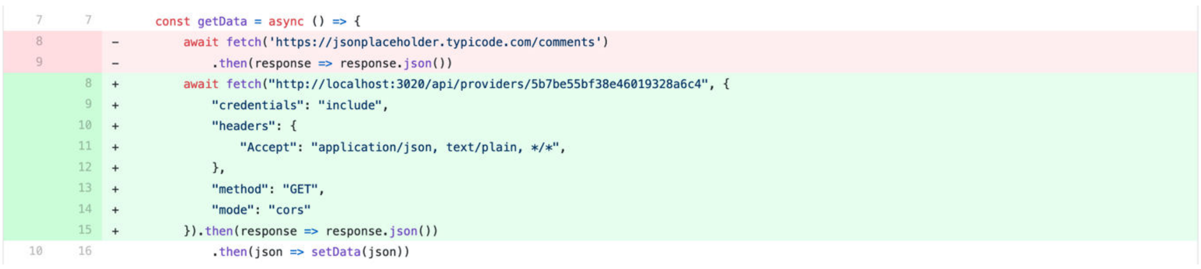


AL-2848 Allows react poc to read the server
AL-2848 + AL-2848B
Krzyszt0f1 committed 22 days ago
1 parent 2cc440e commit 4b1f6bca24ac630616c83b7b4c7d25a18c017ab0

Fig. 3.5, shows the title of the commit I pushed the relevant server changes with. As can be seen, I did push here a relevant change, without overloading the commit with different, not relevant pieces of work. That's one of generic best practices, to commit relevant pieces of work with good frequency.

Read operation

With the server configured to allow the new react project access it, I was ready to start writing the POC. The very first piece I wrote and connected to the server and database was a component that was fetching list of home insurance providers available in the database (see the fig. 3.6 below for an example of how I wrote the function to fetch wanted data).



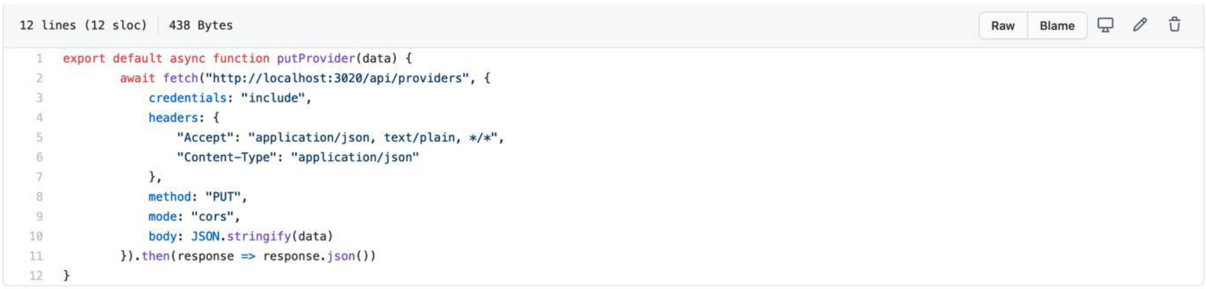
```diff
 7    7          const getData = async () => {
 8       -           await fetch('https://jsonplaceholder.typicode.com/comments')
 9       -               .then(response => response.json())
      8  +           await fetch("http://localhost:3020/api/providers/5b7be55bf38e46019328a6c4", {
      9  +               "credentials": "include",
     10  +               "headers": {
     11  +                   "Accept": "application/json, text/plain, */*",
     12  +               },
     13  +               "method": "GET",
     14  +               "mode": "cors"
     15  +           }).then(response => response.json())
10   16                  .then(json => setData(json))
```

Fig. 3.6, show how I configured a fetch API method, which is a standard yet flexible and powerful JavaScript method to fetch data from servers/databases. I defined it as an asynchronous method to allow for load of information and avoidance of losing it in the loop. Additionally, it is configured to for example include `credentials`, the same way the server side was configured. It also makes sure to read data across origins with the addition of line 14. I specified that this fetch gets data by writing cod on line 13; line 11 specifies that the operation expects to receive "json" or "text" data.

I used simple fetch API as it is vital to stray away from implementing unnecessary dependencies (common, best practice), especially in the case of small scale, proof of concept. It increases readability and accessibility, as other developers do not require to understand specialised, non-standard libraires (i.e., axios).

Create operation

In a similar fashion, using fetch API, I have written create operation that allowed to upload new home insurance provider.
Following code snapshot (fig. 3.7 below) shows and describes how I used fetch method asynchronously to conduct creation of new piece of data through the server that manages the database.



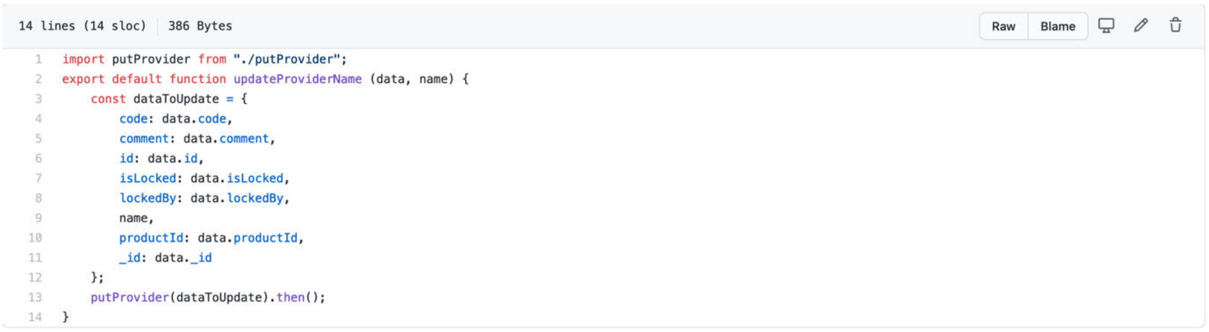
```
 1  export default async function putProvider(data) {
 2          await fetch("http://localhost:3020/api/providers", {
 3              credentials: "include",
 4              headers: {
 5                  "Accept": "application/json, text/plain, */*",
 6                  "Content-Type": "application/json"
 7              },
 8              method: "PUT",
 9              mode: "cors",
10              body: JSON.stringify(data)
11          }).then(response => response.json())
12  }
```

Fig. 3.7, Shows how I created a stand-alone JavaScript function that gets passed a data, makes it a string to be send by `JSON.stringify()` and creates a new entry in the database. It is worth pointing out that I am fully aware that normally, I would use `POST` one-off query to create a new data entry. However, this was the implementation of the Allspark UI API I had to follow. As it is a function to post data, line 10 specifies "body" variable, i.e., what is meant to be send to the database. Additionally, using a `PUT` query subsequently allowed me to re-use this piece of code in the update operation.

Having written the method to post data to the database via API query, made me be one step closer to completion of CRUD-capable POC. Also, as mentioned in the description of fig. 3.7, it scaffolded creation of update operation.

Update operation

The snippet presented below (fig. 3.8), shows and describes my effort to reuse already existing code (using units of code, and reusability of components and functions being core best practices of react and JavaScript respectively), to perform update of selected home insurance provider.



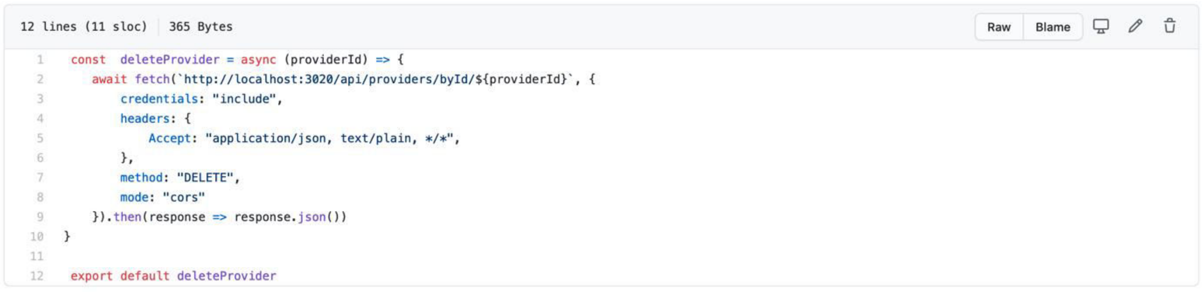
```
14 lines (14 sloc)  386 Bytes
import putProvider from "./putProvider";
export default function updateProviderName (data, name) {
    const dataToUpdate = {
        code: data.code,
        comment: data.comment,
        id: data.id,
        isLocked: data.isLocked,
        lockedBy: data.lockedBy,
        name,
        productId: data.productId,
        _id: data._id
    };
    putProvider(dataToUpdate).then();
}
```

Fig. 3.8, presents how I crated an update request to the database via the API. It is a good snippet as it exemplifies how I created small, reasonable chunks of code and kept them as simple as possible. I used here already created `putProvider` function and used it to create an update query. It also shows how I am capable of working with and creating data structures. The constant `dataToUpdate` is an object later being sent as a JSON string and read as a JSON object at the point of entry in database. To create it, I have pulled data that user puts into a form for creation of a new provider. It presents my ability to work with and manipulate NoSQL data structures.

In turn, what the code in the previous two figures does (when used in the react component), is, it gets a provider and its new desired name passed as a string and overrides the old entry of the provider in the database changing the provider's name. Which was one of the requirements to achieve, to complete the POC.

Delete operation

Lastly, to satisfy requirements, I had to provide the POC with a function that upon obtaining specific's insurance provider ID string, would communicate to the server and command it to delete the provider all together from database. Figure 3.9 below, outlines the creation of such function.



```
12 lines (11 sloc)  365 Bytes
1   const  deleteProvider = async (providerId) => {
2      await fetch(`http://localhost:3020/api/providers/byId/${providerId}`, {
3          credentials: "include",
4          headers: {
5              Accept: "application/json, text/plain, */*",
6          },
7          method: "DELETE",
8          mode: "cors"
9      }).then(response => response.json())
10  }
11
12  export default deleteProvider
```

Fig. 3.9 is a sample of how I can create a delete query to a database. It uses a unique provider id to access the correct element in the database to delete. Again, to let the fetch method know that it should make a query to delete piece of data, line 7 clarifies the intention. The URL on line 2 also allows the means to find the 'providerId' of piece to be deleted.

As can be appreciated, the delete function is in essence the simplest API call out of the four I had to create. Interestingly, the line 9 of the code snippet of the deletion function, shows that despite no real file being received after making a delete call; some response is still being resolved.
The response in this case, is just a HTTP response code message that informs the developer (and potentially user, should such information be passed to him/her) about whether the call was completed or not. For

example, should the deletion be successful, the `response` would be of code type 204. It explicitly means that deletion was completed (OK), but no content was sent back as a result of it.

Connection to database

As mentioned in `CRUD functionality and the need to adjust cross origin accessibility` the functions I wrote to obtain data from the database connect to the intermediate layer of back-end code which then, directly queries database for output. For example, when using the code from fig. 3.6 (here), it triggers following functions (fig. 4.1 to 4.2 below and on the next page):

```
async function getProviders(req, res) {

    const searchObject = {
        productId: req.params.productId,
        deletedOn: null
    };
    res.json(await findAsync(Provider, searchObject));
}
```

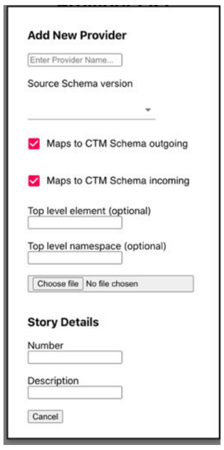
Fig. 4.1, a piece of server code from a server part of the old Allspark application which is responsible for finding a list of providers by an id of a given insurance product. It does not use any specific mongoose syntax, but builds on a `findAsync` (see fig. 4.2 below) function which is directly talking to the data base.

```
module.exports.findAsync = async (model, searchObject) => await
model.find(searchObject).exec();
```

Fig. 4.2, an export of a `findAsync` function that directly queries the mongo DB structure using mongoose (library to use mongo DB). Build in functions of `.find` and `.exec`. The `.find` function looks for a specific object in a given model, whereas `.exec` simply executes the database query. Model is a specific data schema, in this case `getProvider` function is after a provider data schema.

Recreating the complex form

As part of the project, in order to recreate adding a new provider, I have created a complex submission form that allows to upload xml files. (see fig. 4.3 below to see the form created).



Add New Provider

Enter Provider Name...

Source Schema version

☑ Maps to CTM Schema outgoing

☑ Maps to CTM Schema incoming

Top level element (optional)

Top level namespace (optional)

Choose file No file chosen

Story Details

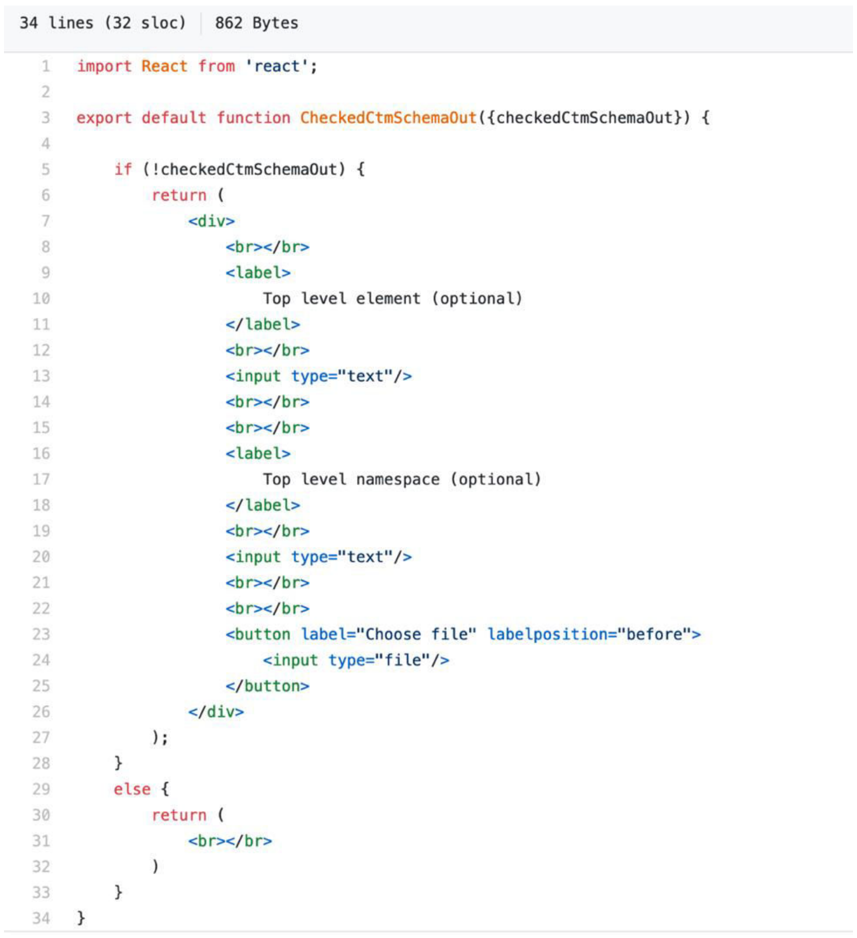
Number

Description

Cancel

Fig. 4.3, the form that was included in a modal that pops up when user wants to add new provider.

To create the component, I have used a pre-set modal that I have found on an opensource library of Material-UI.
This form I created had a logic behind it that allows the user only to upload a file when relevant fields are not filled in, i.e., the checkboxes.
It was interesting to find out how simple it actually is to make a react component to receive a file (see fig. 4.4 below for the code I used to do so).

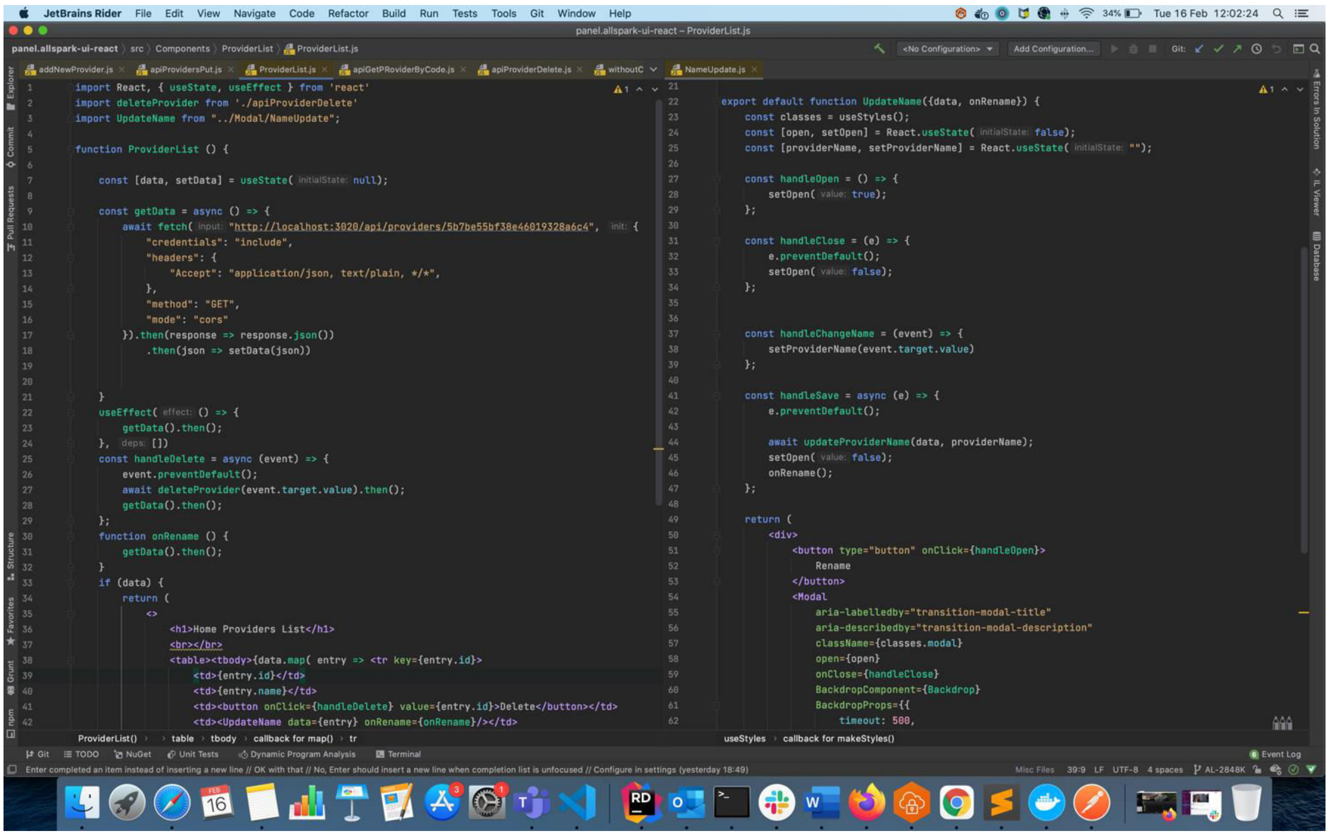


```
34 lines (32 sloc) | 862 Bytes

 1  import React from 'react';
 2
 3  export default function CheckedCtmSchemaOut({checkedCtmSchemaOut}) {
 4
 5      if (!checkedCtmSchemaOut) {
 6          return (
 7              <div>
 8                  <br></br>
 9                  <label>
10                      Top level element (optional)
11                  </label>
12                  <br></br>
13                  <input type="text"/>
14                  <br></br>
15                  <br></br>
16                  <label>
17                      Top level namespace (optional)
18                  </label>
19                  <br></br>
20                  <input type="text"/>
21                  <br></br>
22                  <br></br>
23                  <button label="Choose file" labelposition="before">
24                      <input type="file"/>
25                  </button>
26              </div>
27          );
28      }
29      else {
30          return (
31              <br></br>
32          )
33      }
34  }
```

Fig. 4.4 presents how I used `input type="file"` on lines 23 to 25 in order to conditionally allow for submission of a file. It also shows how I followed the principal of separation of concerns. The react component above is just a child component of the whole form. I did so as smaller, easy to follow components that piece together increase readability and maintainability of the code. I am also, aware that my use of multiple `<br>` is not the best practice. Instead, I could have styled it with CSS.

Making the components pass the state up from child to parent

During the creation of the POC I learned a handy way of allowing the child components to directly update the state of parent components. I learned that from a Meerstrap team colleague Lawrence. It happened when I was showing him almost Finished POC and described that I initially introduced very bad practice way of updating the list of insurance providers. Originally, I achieved it by forcing the page to reload on deleting or updating the list. I understood it is very wasteful as it forces all components to reload unnecessarily, but I didn't know how to do so without using a state management tool like Redux. To keep the POC lightweight, Tim and I were advised to keep it as pure as possible, hence, lack of the use of such tool. Nevertheless, Lawrence navigated me thorough how to make a list refresh (so only single component) upon deletion of a specific provider, and later I have applied the same technique on my won to make it dynamically refresh upon updating the name of the provider.
(see fig. 4.5 on the next page, for a brief description on how on how I did it)



panel.allspark-ui-react › src › Components › ProviderList › ProviderList.js

```
1   import React, { useState, useEffect } from 'react'
2   import deleteProvider from './apiProviderDelete'
3   import UpdateName from "../Modal/NameUpdate";
4
5   function ProviderList () {
6
7       const [data, setData] = useState(null);
8
9       const getData = async () => {
10          await fetch("http://localhost:3020/api/providers/5b7be55bf38e46019328a6c4", {
11              "credentials": "include",
12              "headers": {
13                  "Accept": "application/json, text/plain, */*",
14              },
15              "method": "GET",
16              "mode": "cors"
17          }).then(response => response.json())
18              .then(json => setData(json))
19
20
21      }
22      useEffect( () => {
23          getData().then();
24      }, [])
25      const handleDelete = async (event) => {
26          event.preventDefault();
27          await deleteProvider(event.target.value).then();
28          getData().then();
29      };
30      function onRename () {
31          getData().then();
32      }
33      if (data) {
34          return (
35              <>
36                  <h1>Home Providers List</h1>
37                  <br></br>
38                  <table><tbody>{data.map( entry => <tr key={entry.id}>
39                      <td>{entry.id}</td>
40                      <td>{entry.name}</td>
41                      <td><button onClick={handleDelete} value={entry.id}>Delete</button></td>
42                      <td><UpdateName data={entry} onRename={onRename}/></td>
```

NameUpdate.js

```
21
22  export default function UpdateName({data, onRename}) {
23      const classes = useStyles();
24      const [open, setOpen] = React.useState(false);
25      const [providerName, setProviderName] = React.useState("");
26
27      const handleOpen = () => {
28          setOpen(true);
29      };
30
31      const handleClose = (e) => {
32          e.preventDefault();
33          setOpen(false);
34      };
35
36
37      const handleChangeName = (event) => {
38          setProviderName(event.target.value)
39      };
40
41      const handleSave = async (e) => {
42          e.preventDefault();
43
44          await updateProviderName(data, providerName);
45          setOpen(false);
46          onRename();
47      };
48
49      return (
50          <div>
51              <button type="button" onClick={handleOpen}>
52                  Rename
53              </button>
54              <Modal
55                  aria-labelledby="transition-modal-title"
56                  aria-describedby="transition-modal-description"
57                  className={classes.modal}
58                  open={open}
59                  onClose={handleClose}
60                  BackdropComponent={Backdrop}
61                  BackdropProps={{
62                      timeout: 500,
```

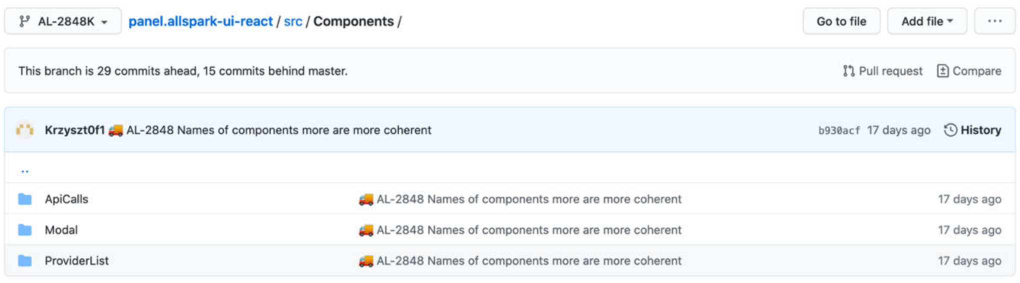
Fig. 4.5, Shows how to use props injection in the child object (onRename in NameUpdate.js line 46 and 22 for definition of props) to pass the state up to a parent, in this case ProviderList in ProviderList.js which uses onRename to trigger a refresh of state in the component (lines 42 and 30-32).

These changes allowed the POC app to work as desired, that is to be fully dynamic (no need for forced page reloads), one page website.

Renaming and cleaning the repository

I have also made sure that towards the end of the project, I will clean up the code base. Although I knew the POC won't be developed any further, I wanted to make sure, that for the presentational purposes, I will leave it all clean, with coherent naming conventions, simple to follow repository structure and no linting errors.

I made sure to keep all the reusable API fetches in separate folder, kept modal and its subcomponents in its own folder and the highest in the hierarchy parent component (ProviderList) has its own folder. Furthermore, all of those components were kept in a relevant, `src` (source) directory (see the fig. 4.6 below that is an example of changes I did to the folder structure).



AL-2848K · panel.allspark-ui-react / src / Components /

This branch is 29 commits ahead, 15 commits behind master.

Krzyszt0f1 AL-2848 Names of components more are more coherent · b930acf 17 days ago · History

ApiCalls	AL-2848 Names of components more are more coherent	17 days ago
Modal	AL-2848 Names of components more are more coherent	17 days ago
ProviderList	AL-2848 Names of components more are more coherent	17 days ago

Fig. 4.6 presents the changes I did to the project structure that follow generic best practice of keeping the code in relevant folders. The naming convention is also followed, camel cased folders and files.

My contributions after completing the project

Help on shaping the road map and inception brief

During the work on the project, I have attended a meeting with team's BA and my pair partner, Nicola at which I have contributed to the creation of a road map of how the actual rewrite should happen.

Similarly, I have also played an important part during a whole team meeting at which everyone was welcomed to comment and add to the road map initially blue printed at the meeting with Tim and Nicola.

That part I played was not only by just participating in a conversation, in both cases I was the one to brief colleagues, BA and DM about the work Tim and I did on the proof of concept. It was a detailed brief outlining not only the end result, but most importantly the discoveries made along the way, for the evidence, please see below a transcript of my notes I used to brief other stakeholders.

Notes for the dev catch up:

- *Why did it take us so long?*

We have been trying various approaches, the final decision of creating the POC form scratch in a separate react project was made after trying to:

— Our first Idea was to try and write a simple react component which could have been put on top of already existing angular body.
We have been trying to achieve it by means of:

1. *Following a story and tutorial that has been documented buy an outside team who manually created "gluing" solutions for putting react on top of angular.js (here: https://tech.small-improvements.com/how-to-migrate-an-angularjs-1-app-to-react/)*
2. *This quickly proved to be out of our reach, even with the help of meerstrap colleagues. We decided to experiment with widely used and written about two particular libraries:*

- *React2angular,*
- *ngReact.*

However, these solutions, despite being popular across web, proved to be unsuitable due to the fact that current all spark-ui is being run in a docker with use of grunt. Combination for which we couldn't find solution for on our own, nor couldn't find well documented solution upon further search.

— Later on, we partially stroke some progress by creating a simple, presentational button in the existing all spark-ui be directly injecting react to the html view. Unfortunately, this has quickly proved to be unscalable. It would only allow for creating "dumb" elements with limited to no logic. Additionally, with such solution, due to the project set up, we were unable to include babel dependency. This would allow us to write such react components in jsx, react's preferred syntax; without it was pointless.

— Lastly, by closely collaborating with meerstrap's Lawrence and Jason we tried creating a hybrid development environment which would allow to run angular and react in parallel. This has borough some results, it could have potentially been possible to set up such environment, however, the complexity of the already existing structure of all spark-ui, it proved to be too complicated, especially for POC.

- *The results:*

1. *With all the problems discovered and explored, it was decided that for now, the POC (and most likely the whole rewrite) should be created from scratch in a separate react project.*
2. *For the POC we have created a simple CRUD page that allow user to:*

- *Load Home insurance Providers List,*
- *Add new Home insurance provider as long as it is being defined with custom outgoing schema,*
- *Rename the providers,*
- *Delete providers.*

3. *For it to work, we have made a new branch on all spark-ui that allows to locally access the service in cross origin manner "restricting" it to any localhost domain.*
4. *This obviously leaves a question of how to integrate server part of all spark ui to be used by react rewrite, but for now leaves a reliable, yet "hacky" development environment.*
5. *Lawrence did a valuable work on trying to separate server part out of all spark-ui, which is one proposed solution to allow react rewrite to consume it as an API. However, it is a work in progress.*

The verbal feedback I was given was positive, I have stated the findings clearly and with appropriate level of technical detail so that technical and non-technical colleagues understood the brief; it has vitally kick started the inception.

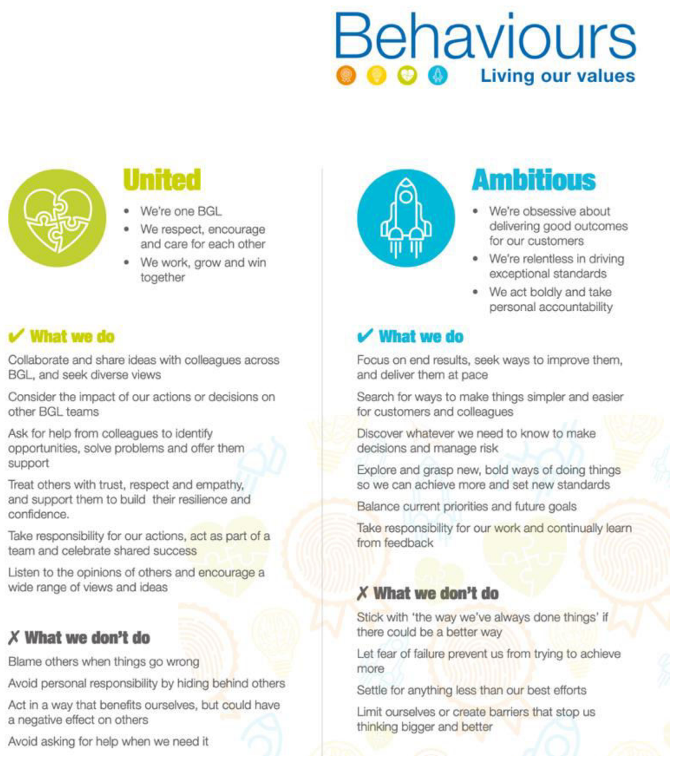
In conclusion

Things I learned and improved upon during the work

Along the way I have learned few new things about React JS.
I got to know how to upload files via components and how to update state from child to parent components.
I further developed my understanding of react also by sharing some key ideas with my pair partner and practiced how to write coherent and clearly named files, components and functions.
I have also gained experience in how to communicate with different stakeholders and external (to my team) developers.

Value I brought and the impact I made

Summarising, I have contributed to the project by being able to share my understanding of the technology in use, was able to work in pair and mob together with people from outside of my standard team. I was proactive in sharing technical issues and findings.
I made sure to follow best common best practices, like favouring the use of functional components rather than class ones. I worked with data and performed CRUD operations on it and was able to work with project's preferable data structures.
As mentioned in My contributions after completing the project, my actions (creation of POC), observations (summary of findings and blockers discovered along the way) and delivery of brief during the inception meetings has impacted the planning for the Allspark UI rewrite, which consequently made a difference to how will one of company's internal tool (Allspark UI) look like in the future and what technologies are going to be used to shape it.
I have also shared my knowledge impacting the collective understanding of how React works within Allspark team by collaborating with people form Meerstrap, which meant I was practicing one of the core behaviour values of the group which is being united (see fig. 4.7 below for a screen that displays detailed description of "united" corporate values).



Behaviours
Living our values

United

- We're one BGL
- We respect, encourage and care for each other
- We work, grow and win together

✓ What we do

Collaborate and share ideas with colleagues across BGL, and seek diverse views

Consider the impact of our actions or decisions on other BGL teams

Ask for help from colleagues to identify opportunities, solve problems and offer them support

Treat others with trust, respect and empathy, and support them to build their resilience and confidence.

Take responsibility for our actions, act as part of a team and celebrate shared success

Listen to the opinions of others and encourage a wide range of views and ideas

✗ What we don't do

Blame others when things go wrong

Avoid personal responsibility by hiding behind others

Act in a way that benefits ourselves, but could have a negative effect on others

Avoid asking for help when we need it

Ambitious

- We're obsessive about delivering good outcomes for our customers
- We're relentless in driving exceptional standards
- We act boldly and take personal accountability

✓ What we do

Focus on end results, seek ways to improve them, and deliver them at pace

Search for ways to make things simpler and easier for customers and colleagues

Discover whatever we need to know to make decisions and manage risk

Explore and grasp new, bold ways of doing things so we can achieve more and set new standards

Balance current priorities and future goals

Take responsibility for our work and continually learn from feedback

✗ What we don't do

Stick with 'the way we've always done things' if there could be a better way

Let fear of failure prevent us from trying to achieve more

Settle for anything less than our best efforts

Limit ourselves or create barriers that stop us thinking bigger and better

Fig. 4.7, snapshot of descriptions for corporate values "united" and "ambitious", as can be seen, thorough my work on the POC, I actioned accordingly to the "dos" of the "united" value.