**Material UI refactor (Web App): MyGamePlan Internship**

By Bogdan Lazar (R0779031 - 3AI01)

# Initiation phase: preparation

Before I got my hands on the code written by the developers of MyGamePlan, I decided to take a while to catch up by studying some of the very necessary chapters to me from Maximilian Schwarzmuller’s React complete course which was on MyGamePlan’s academic Udemy account that I got access to.

I mostly looked at some chapters concerning app wide state (the context API) and TypeScript as well as seeing some real examples of working with React and TypeScript. After two weeks of essentially preparing myself for the real deal, I agreed with the team to cut the studying short and learn by doing.

It was time to do it the IT way. Being ready for massive changes at any point and embrace the struggle.

# Foreword: Material UI vs AtlasKit

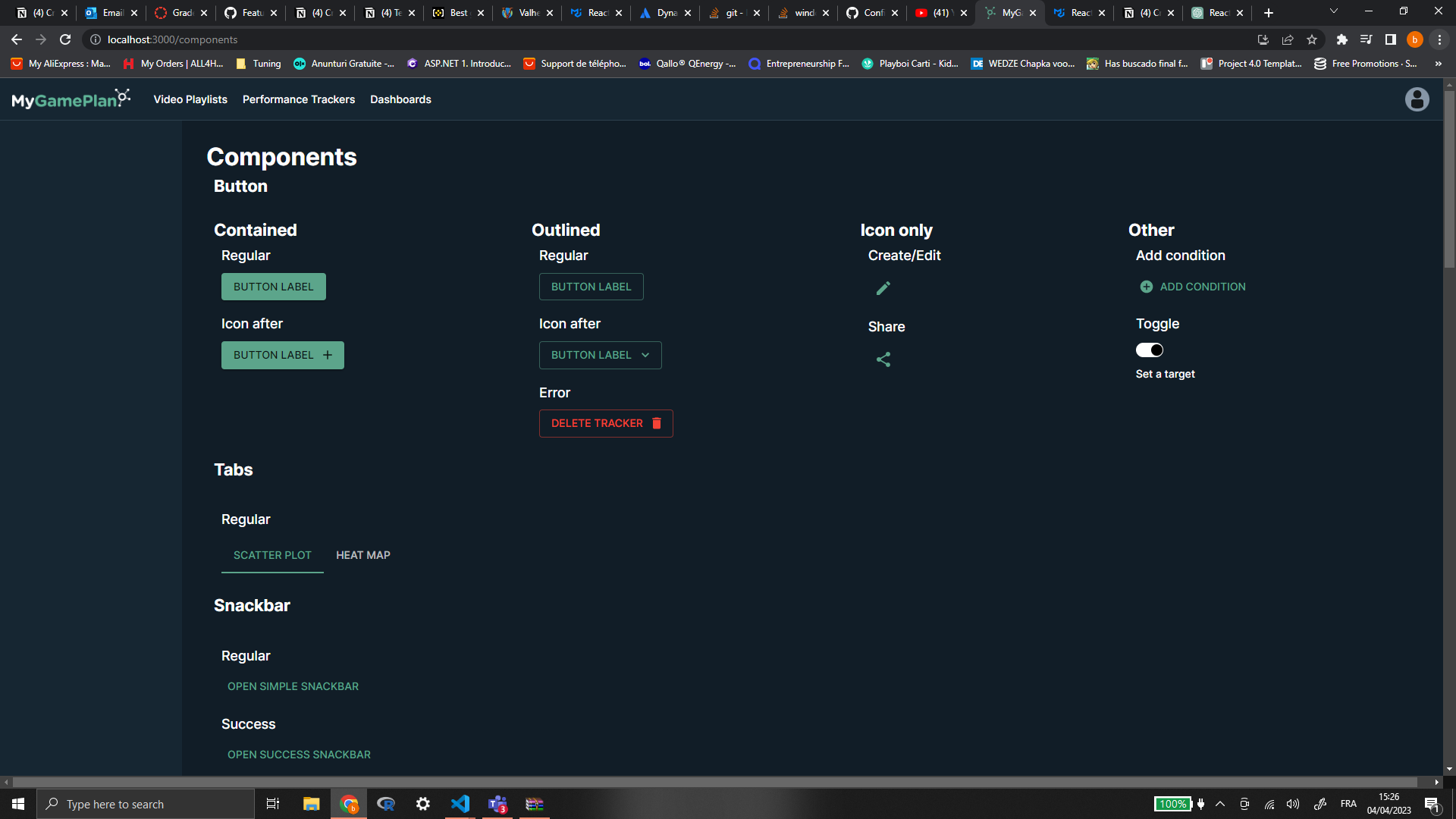
Now, why Material UI? Well, this component library gave a smoother, more animated feel to the web application and it ultimately turned out to cut down on lots of code from AtlasKit integrations that was rendered useless once most of the Material components were integrated.

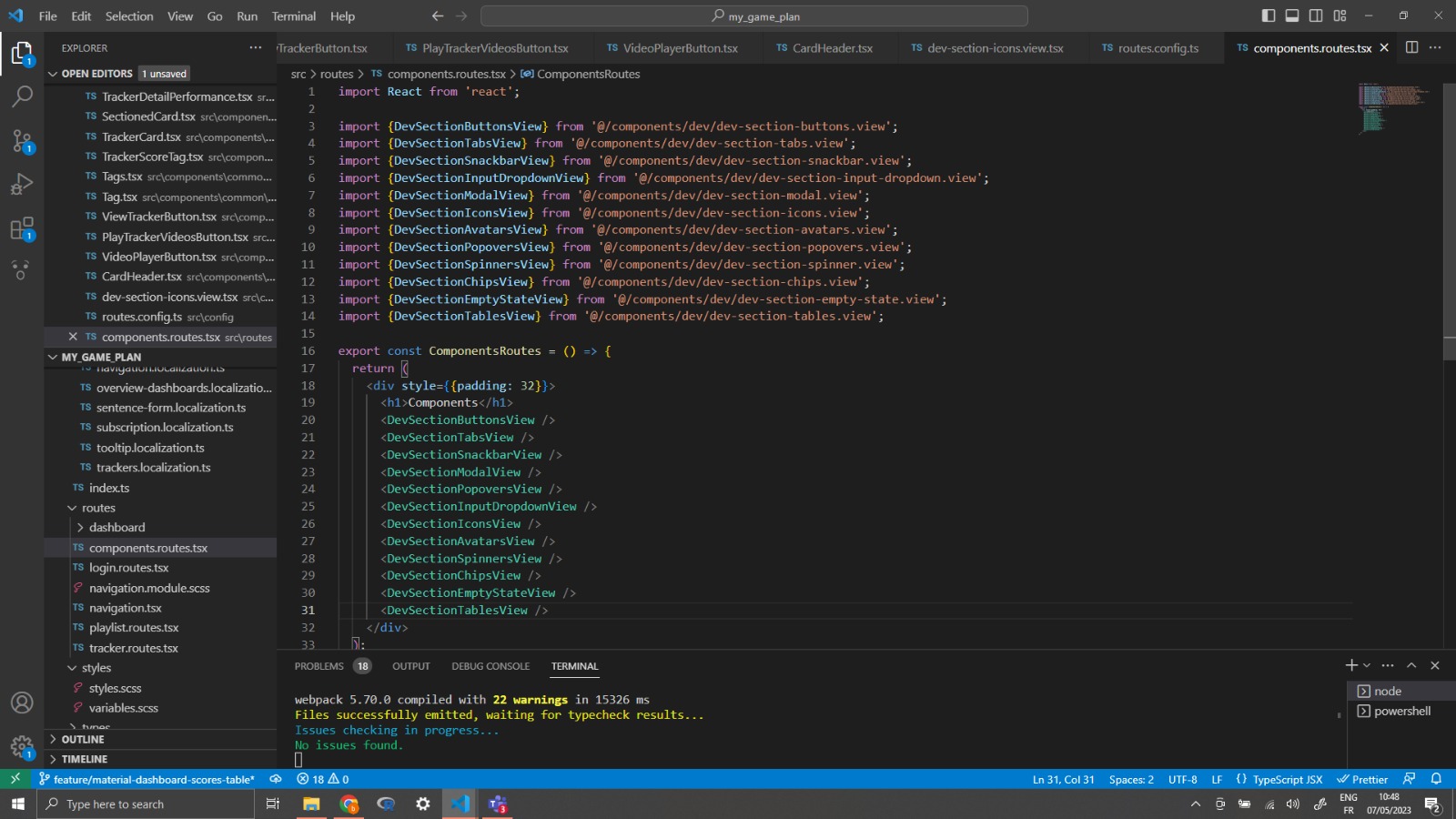
The reason the fact of cutting down on code is important, is because the more code is executed in order to run the application, the slower it will run. The ultimately leaner code would provide more responsiveness to the app itself.

# Task 1: Strategy for Replacing AtlasKit with MaterialUI

MyGamePlan wanted to replace its long lasting AtlasKit component library for something smoother and more subtle.

MaterialUI was the best choice and, in order to start refactoring, a special page would have to be created within the app in order to not only guide the average developer when looking for how to implement a certain component but also to provide a solid strategy on what to replace and with what.





In order to set the foundation for this feature, due to the structure of the app, a page had to be set up with its respective route.

After creating the components.routes.tsx file, this page would be found via /components in the app. To also lean out the code it was a much better idea to then create a view for each set of components that would be shown.

In order for the actual component bit to fall into place, a scalable React component had to be made. This is the DevSection component which can be told by the naming convention of each view.

Each view basically takes this component, binding the right data to the appropriate props of DevSection and returning it as JSX code. Here is the DevSection code with an example of a view:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

In order for the view to work properly, a way to store the data for each component was needed. This was done in the end via JSON with an array of objects where each object contained an array of variants of each component where the components would be stored by type.

To fix any potential compilation issues, special interfaces were made to instruct the app that a property of the object was of React.Node type. After putting all of these in place, for each DevSection it was simply rinse and repeat with certain caveats of logical functions to make them work properly for a basic demo in each case.

The only problem that remained was that some components were either not available in the base MaterialUI library and either required experimental code or paying for the X library. There was also an issue with finding some components to replace AtlasKit ones with like tables that the company was using.

It was a good learning experience. It really put the developer life into perspective.

# Task 2: Replacing AtlasKit with MaterialUI

Now it was time to actually replace the AtlasKit components with MaterialUI ones. This started small, with buttons that could be swapped out immediately. It just took the code and imports for the Material UI items that would be used instead.

Then the right bindings had to be made to the props of the Material UI component and it would be ready to go. Here’s an example:

Text

Description automatically generated

In this simple case anything that was still AtlasKit component code was commented out and the MaterialUI code was dropped in. Commenting helped in case errors were encountered and made it easy to simply switch the codebases in order to visualize which one was working.

Everything worked well until the components such as modals were encountered which were working with logic that was programmed at way deeper levels and required more experience with the overall code of the application in order to be done.

The team then decided that this task was better escalated to more experienced personnel. After seeing that there were no more simple components to replace, it was time to take on another task.

It turned out that the tech lead managed to find a way to replace the data tables with regards to shots and crosses of the players in a team.

# Task 3: Replacement for shots and crosses tables (strategy)

This task would briefly revisit the components page. The point was to replace the AtlasKit table that was being used for shots and crosses data in the dashboards of a team.

After discussions with the tech lead, it was decided that using a dummy json object array we could have a proof of concept for such a replacement table. In this case the idea was to recreate this table with basic React and TypeScript.

It was required of the user to be able to select columns, rows and cells. With enough research, it turned out that this could actually be done with a Material basic table but with a bit more vanilla React. Here is the code with an explanation:

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

The basic idea is that with a bit of logic to enable the required selection, Material’s basic table could easily be used for this. This task has been completed about 75%. The table was displaying the data as required and rows and columns were selectable.

The only problem was that cells were not selectable and rows could be selected by clicking any cell within a row. The idea was only to enable selection for clicking the first cell of a row.

Now this problem was an easy fix within the logic by simply only enabling selection when clicking the first cell of a row. Column selection was already a bit more extensive with the code.

After consultation with the rest of the team, it was decided another task could be taken on instead of ruminating over cell selection and wasting time.

# Task 4: Easier access to User manual from app

This task entailed a bit more than it let on. Not only was it required to provide easier access to the user manual from the user settings popup menu but also to try refactoring the main navigation of the app to Material UI. This is how it was done:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

This was done by first creating a view for this navbar to follow the new convention the dev team agreed upon. In this view the navigation itself with all of its respective logic was coded.

It included Material’s AppBar with a Tabs component entailed within. Lots of logic function programming was required therefore. This was then declared as a React component in JSX code in the file that displayed the navigation itself. This was a lot of developer fun.

The whole set of problems encountered as a developer on the regular were faced during this task but alas it was carried out quite well. Of course, feedback followed that was therefore resolved by fixing the issues within the code and committing once more.

## Task 5: Replacing the main navigation

For this feature, I had to replace the AtlasKit main navigation with a Material UI AppBar.  
A screenshot of a computer

Description automatically generated

In order to accomplish the same feel while using Material UI instead, the AppBar had to be mixed with a Tabs component. This included refactoring the user menu options along with the manual access as well.

At first, a basic implementation was attempted to see what fell into place and what still needed work for the implementation to work fine.

app-bar.view.tsx

import {AppBar} from '@mui/material';

import {ProductHomeContainer} from '../layout/ProductHomeContainer';

import Box from '@mui/material/Box';

import Toolbar from '@mui/material/Toolbar';

import IconButton from '@mui/material/IconButton';

import Typography from '@mui/material/Typography';

import Menu from '@mui/material/Menu';

import Tabs from '@mui/material/Tabs';

import Tab from '@mui/material/Tab';

import Avatar from '@mui/material/Avatar';

import Tooltip from '@mui/material/Tooltip';

import MenuItem from '@mui/material/MenuItem';

import LogoutIcon from '@mui/icons-material/Logout';

import MenuBookIcon from '@mui/icons-material/MenuBook';

import ListItemIcon from '@mui/material/ListItemIcon';

import {useAuth} from '@/context/auth.context';

import {useTrackers} from '@/context/tracker/tracker.context';

import {useTeams} from '@/context/team.context';

import FeedbackIcon from '@atlaskit/icon/glyph/feedback';

import LightbulbFilledIcon from '@atlaskit/icon/glyph/lightbulb-filled';

import React, {useEffect, useRef, useState} from 'react';

import {Link, Routes, useLocation, useNavigate} from 'react-router-dom';

import {useTranslation} from 'react-i18next';

import {IRoute, NAVIGATION, ROUTES} from '@/config/routes.config';

function filterRoutes(): IRoute[] {

return ROUTES.filter((route) => route.navigation === NAVIGATION.TOP);

}

export const AppBarView = () => {

const [anchorElUser, setAnchorElUser] = useState<null | HTMLElement>(null);

const [value, setValue] = React.useState(0);

const \_routes = useRef<IRoute[]>(filterRoutes());

const \_location = useLocation();

const \_navigate = useNavigate();

const {user, logoutUser} = useAuth();

const \_trackers = useTrackers();

const \_teams = useTeams();

const {t} = useTranslation();

useEffect(() => {

const \_foundIndex = \_routes.current.findIndex(

(route) => \_location.pathname.indexOf(route.path) > -1,

);

setValue(\_foundIndex);

}, [\_location.pathname, \_routes.current]);

if (!\_routes) return null;

function \_onButtonClick(event: React.MouseEvent<HTMLElement>) {

event.preventDefault();

const \_target = event.currentTarget as HTMLAnchorElement;

\_navigate(\_target.pathname);

}

const handleLogout = () => {

\_trackers.clear();

\_teams.clear();

logoutUser();

};

const handleOpenUserMenu = (event: React.MouseEvent<HTMLElement>) => {

setAnchorElUser(event.currentTarget);

};

const handleCloseUserMenu = () => {

setAnchorElUser(null);

};

const handleChange = (event: React.SyntheticEvent, newValue: number) => {

setValue(newValue);

};

return (

<AppBar position="fixed">

<Toolbar>

<ProductHomeContainer />

<Box sx={{flexGrow: 1, display: {md: 'flex'}}}>

<Tabs value={value} onChange={handleChange}>

{\_routes.current.map((route) => (

<Tab

sx={{

'&:hover': {

color: 'primary.main',

},

}}

key={route.label}

label={route.label}

component={Link}

to={`/${route.path}`}

/>

))}

</Tabs>

</Box>

<Box sx={{flexGrow: 0}}>

<Tooltip title={t('settings.open')}>

<IconButton onClick={handleOpenUserMenu} sx={{p: 0}}>

<Avatar

alt={`${user?.firstName} ${user?.lastName}`}

src="/static/images/avatar/2.jpg"

/>

</IconButton>

</Tooltip>

<Menu

id="menu-appbar"

anchorEl={anchorElUser}

anchorOrigin={{

vertical: 'bottom',

horizontal: 'right',

}}

open={Boolean(anchorElUser)}

onClose={handleCloseUserMenu}>

<MenuItem onClick={handleLogout}>

<ListItemIcon>

<LogoutIcon />

</ListItemIcon>

<Typography textAlign="center">

{t('settings.log\_out')}

</Typography>

</MenuItem>

<MenuItem

href="https://meadow-teal-b64.notion.site/MyGamePlan-User-Manual-fc09f0ff6bd042c292f89cfb109c2ed8"

target="\_blank"

rel="noopener">

<ListItemIcon>

<MenuBookIcon />

</ListItemIcon>

<Typography textAlign="center">

{t('settings.user\_manual')}

</Typography>

</MenuItem>

</Menu>

</Box>

</Toolbar>

</AppBar>

  );

};

The appbar itself needed some more major refactoring. A whole new view file had to be made to include all of the required logic and JSX code.

Because of starting from scratch, lots of compatibility issues were had for which code had to be refactored in other pre-existing parts of the app. The company logo and the user menu also needed to be reconfigured.

For each part of the app to be accessible with the new AppBar, routes had to be reconfigured. AtlasKit and Material UI Tabs simply didn’t have props that were identical for their value type expectations and their names.  
A screenshot of a computer

Description automatically generated

The settings for the user menu were also replaced in order to give easier access to the manual while getting the fresh Material UI feel.

A picture containing text, software, multimedia software, graphics software

Description automatically generated

Lastly, the AppBar component needed to be called within the navigation.tsx file which represented the main navigation of the app as the component of the highest order within the app’s component tree.

A screenshot of a computer

Description automatically generated with medium confidence

And in order for the user to have settings, now with the help of Material UI, the respective menu items had to be added to the user dropdown menu which was part of the appbar view component.

A picture containing screenshot, text, software, multimedia software

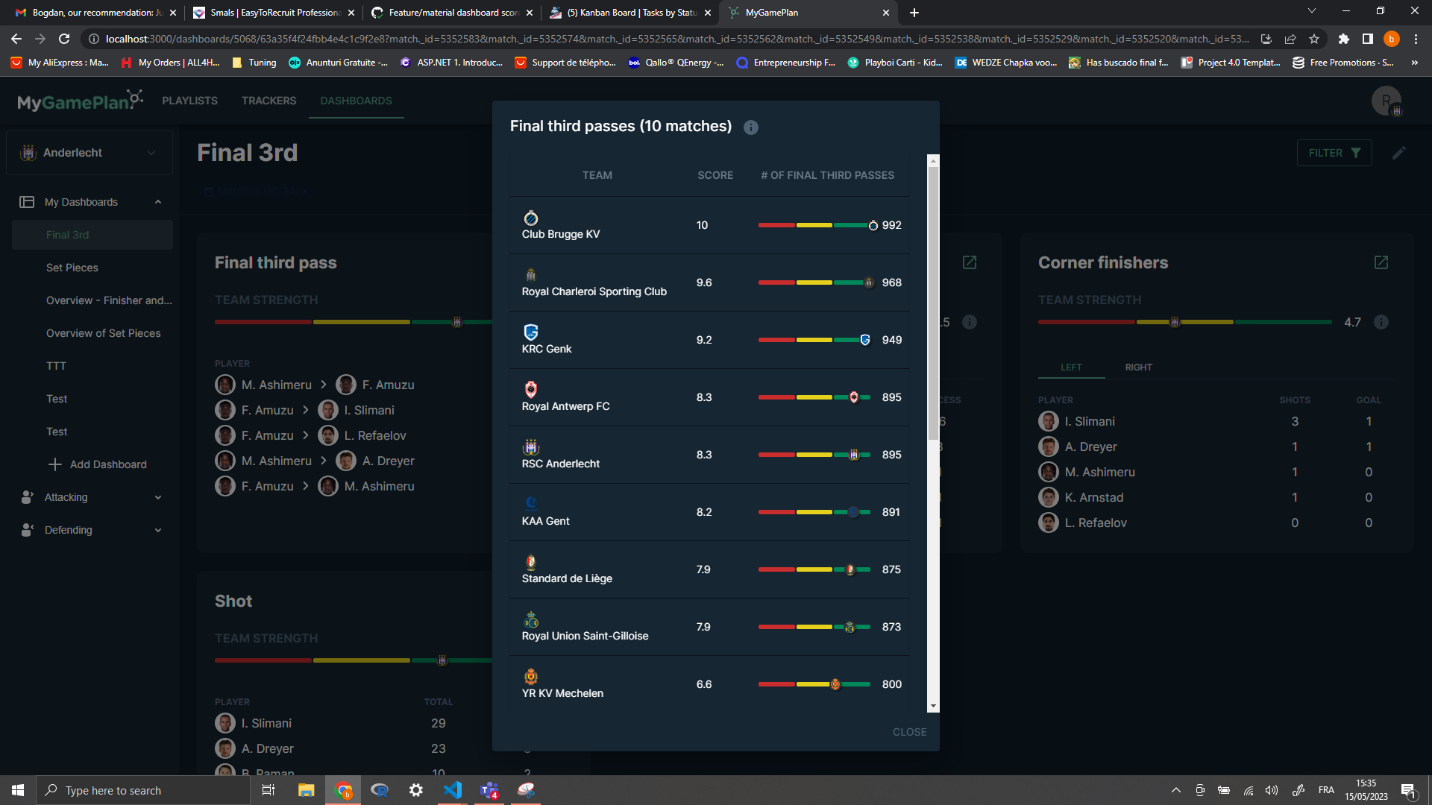
Description automatically generated

This implementation was a real developer challenge. It required a deep dive into the app’s pre-existing code to make something completely new work.

Due to how the app was previously structured using AtlasKit, a whole new component in the app’s tree had to be made just to make the Material UI replacement work properly. In many cases, AtlasKit’s components’ props are different by either the value type that is expected or the name of the prop.

The layout of such components also tends to differ drastically in the case of such complex components as navigation.

## Task 6: Replacing AtlasKit components from the dashboard scores table

For this task, I had to replace the scores table that was present in the dashboard of the web app.   


A screenshot of a computer

Description automatically generated

The first table was had to do with what the dashboard represented from a game that a team played. Data was used in order to display where they stood in terms of executing a certain move at a certain point during a game.  
The second visible table was more of an informative one giving more insight on the structure of the previously represented table.

The following code was written in order to achieve this result:

For the informational table:

import DynamicTable from '@atlaskit/dynamic-table';

import React from 'react';

import {useTranslation} from 'react-i18next';

import {getTranslatedSubscoreProperty} from '@/helpers/translation.helper';

import {IDashboardScore} from '@/types/dashboard/dashboard-score.types';

import {getSubscoresWeights} from '@/helpers/dashboard/dashboard.helper';

import Table from '@mui/material/Table';

import TableBody from '@mui/material/TableBody';

import TableCell from '@mui/material/TableCell';

import TableContainer from '@mui/material/TableContainer';

import TableHead from '@mui/material/TableHead';

import TableRow from '@mui/material/TableRow';

import Paper from '@mui/material/Paper';

interface IndicatorInformationModalTableProps {

score: IDashboardScore;

action?: string;

}

export const IndicatorInformationModalTable = ({

score,

action,

}: IndicatorInformationModalTableProps) => {

const {t} = useTranslation();

return (

// <DynamicTable head={head()} rows={rows(score, action)} />

<div>

<TableContainer component={Paper}>

<Table aria-label="simple table">

<TableHead>

<TableRow>

<TableCell align="center">

{t('dashboard.scores.table.component')}

</TableCell>

<TableCell align="center">

{t('dashboard.scores.table.weight')}

</TableCell>

</TableRow>

</TableHead>

<TableBody>

{score.subscores?.map((subscore) => (

<TableRow key={score.team.name}>

<TableCell component="th" scope="row" align="center">

{getTranslatedSubscoreProperty(

action || '',

subscore.property,

)}

</TableCell>

<TableCell component="th" scope="row" align="center">

{`${subscore.weight}%`}

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

</div>

);

};

// const head = () => {

// return {

// cells: [

// {

// key: 'component',

// content: t('dashboard.scores.table.component'),

// shouldTruncate: true,

// },

// {

// key: 'weight',

// content: t('dashboard.scores.table.weight'),

// shouldTruncate: true,

// },

// ],

// };

// };

// const rows = (score: IDashboardScore, action = '') => {

// const cells = [];

// const subscores = getSubscoresWeights(score);

// if (subscores) {

// for (const {property, weight} of subscores) {

// const temp = [];

// temp.push({

// key: property,

// content: getTranslatedSubscoreProperty(action, property),

// });

// temp.push({

// key: `${property}-weight`,

// content: `${weight}%`,

// });

// cells.push({key: property, cells: temp});

// }

// }

// return cells;

// };

And for the actual table:

import React from 'react';

import {useTranslation} from 'react-i18next';

import {IDashboardScore} from '@/types/dashboard/dashboard-score.types';

import Table from '@mui/material/Table';

import TableBody from '@mui/material/TableBody';

import TableCell from '@mui/material/TableCell';

import TableContainer from '@mui/material/TableContainer';

import TableHead from '@mui/material/TableHead';

import TableRow from '@mui/material/TableRow';

import Paper from '@mui/material/Paper';

import Indicator from '../Indicator';

import {getMinMaxSubscores} from '@/helpers/dashboard/dashboard.helper';

import {getTranslatedSubscoreProperty} from '@/helpers/translation.helper';

import Logo from '@/components/common/Logo/Logo';

interface IndicatorTableProps {

scores: IDashboardScore[];

action?: string;

}

export const IndicatorTable = ({scores, action}: IndicatorTableProps) => {

const {t} = useTranslation();

const subscoresMinMax = getMinMaxSubscores(scores);

return (

<div>

<TableContainer component={Paper}>

<Table aria-label="simple table">

<TableHead>

<TableRow>

<TableCell align="center">

{t('dashboard.scores.table.team')}

</TableCell>

<TableCell align="center">

{t('dashboard.scores.table.score')}

</TableCell>

{scores[0].subscores?.map((subscore) => (

<TableCell align="center">

{getTranslatedSubscoreProperty(

action || '',

subscore.property,

)}

</TableCell>

))}

</TableRow>

</TableHead>

<TableBody>

{scores.map((score) => (

<TableRow key={score.team.name}>

<TableCell component="th" scope="row">

<div style={{display: 'flex', alignItems: 'center'}}>

<Logo src={score.team.image\_url} size="small" />

&nbsp;

{score.team.name}

</div>

</TableCell>

<TableCell align="center">{score.score}</TableCell>

{score.subscores &&

score.subscores.map((subscore) => (

<TableCell align="center">

<Indicator

value={subscore.events}

teamId={score.team.\_id}

min={subscoresMinMax[subscore.property].min}

max={subscoresMinMax[subscore.property].max}

scores={scores}

/>

</TableCell>

))}

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

  </div>

  );

};

I want this code to be the focus. This is where the most important modification was made for the app. The original AtlasKit table was using custom logic and placing rows and columns in variables that were then mapped over in order to get the final result in JSX code.

With the refactor, a different approach had to be taken. After a closer look at the data that was worked with using postman and sending a request, a general idea of the JSON format was available to define a new JSX heavy approach.

The data was now taken directly and mapped over accordingly rather than working with the data in variables storing column/row data. The respective properties of the JSON objects were injected as contents/props for the JSX code of the Material UI table.

Some code was also removed which served as a helper file to define methods that helped with displaying the data in the AtlasKit way.

This was yet another example of how one needs to be prepared for the stormy changes in the IT world. New technologies appear fast and they have to be implemented as quickly as possible to stay ahead.

# Task 7: Refactoring notifications to Notistack snackbars

For this feature, AtlasKit’s flags which were parsed using a helper and then called within every component that required notifications to be displayed for certain actions, were replaced with the Notistack suite and its version of snackbars (notification components).

The following excerpt represents the case of the dashboard and its respective notifications. Keep in mind this was also done for many other components that used notifications such as the video playlist, opponent automations etc.

A screenshot of a computer program

Description automatically generated with medium confidence

The end result was not only that way less code was written per call to notification but some helper files were done away with as well in order to keep the application lean.

In the case of a continuous notification like the one meant for downloading videos from the playlist, some modifications to the props passed on by the context component were also required. More specifically, a “downloading” prop meant to represent the state made the Notistack code incompatible, and it turned out to be redundant for the new solution.

Needless to say, another success was achieved for the UI refactor. Once again, albeit less challenging this time, a proper developer task was completed with persistence and lots of effort.

Here is the result:  
A screenshot of a computer

Description automatically generated

# Task 8: Top filter tags refactor (dashboard)

The app was still using AtlasKit’s lozenge as a filter tag for the dashboards. Naturally, we had to replace this with Material UI Chips.

This was by far the easiest implementation. It posed no compatibility issues with expected return types for props and other such hassle encountered in some of the previous tasks.

The following excerpt represents the main idea in code of this refactor:

A screen shot of a computer program

Description automatically generated with low confidence

As can be seen, it was a simple drag and drop of JSX code along with the respective imports. Everything worked out of the box with the right information assigned to the right props as was done with the AtlasKit component.

Here is the result:

A screenshot of a computer

Description automatically generated



# Task 9: Refactor overview dashboard builder to Material UI

This final task was meant to give the Material UI experience while creating a dashboard. The previous version was using AtlasKit input fields as well as a vanilla React widget selector built from the ground up.

Technically, the input fields had to become Material TextFields and the widget selector had to become a Drawer with list items consisting of widget names and a Material Switch which when enabled, would add the respective widget to the dashboard.

The following excerpts represent the replacement of the text input fields:

A screenshot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

After this quite easy implementation, the time came to work on the widget selector to achieve what was sought after. Initially it was difficult to replace everything as this was another case of being made to work with AtlasKit at a deeper level.

Here are the excerpts piecing together the whole code that was written.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

Due to the AtlasKit structure, lots of brand new JSX code had to be written to accommodate the new components properly. The nested components received a different DOM structure in order for the widgets to be listed properly with the switch next to them.

It’s also worth mentioning that the weird container id’s and inline style on the Drawer and its respective container were meant to make it a contained component within the dashboard builder. The Drawer takes up the whole browser window by default and sticks to either side of the screen depending on prop configuration.

This was yet another proper developer challenge. A proper farewell to this amazing internship.

Here is the final result of this task:

A screenshot of a computer

Description automatically generated