2

Creating User Interfaces with Ant Design

Following the design principles of Ant Design, the Ant Financials’ user experience design team created the Antd library, which offers a variety of React components you can use to accelerate the user interface development.

In this chapter, we'll study the Antd library and create user interfaces using it. The first section will introduce you to the project we will develop, a CRM (Customer Relationship Management) application. Next, we configure the plugin and the layout theme. We'll create the home page and configure internationalization support (also known as i18n) that we'll use on every page from then on. Finally, we'll make the opportunities page, customers page, and reporting page.

In this chapter, we’ll cover the following main topics:

* Introduction to the project and the Ant Design
* Setting up the layout and theme
* Creating the home page and setting up i18n
* Creating the opportunities and customers pages
* Creating the reports page

By the end of this chapter, you’ll have learned how to search and find the right component to meet your needs in the Antd library. You’ll have learned how to configure the plugin-layout, customize your application's default theme, and define global styles. You’ll also know how to set up support for internationalization using the plugin-locale.

Technical requirements

To complete this chapter exercises, you only need a computer with any OS (I recommend Ubuntu 20.04 or higher) and the software installed in chapter one (VScode, Node.js, and yarn).

You can download the complete project on GitHub at:

[https://github.com/PacktPublishing/Enterprise-React-Development-with-UmiJs](https://github.com/PacktPublishing/Enterprise-React-Development_x0005_-with-UmiJs)

Introduction to the project and the Ant Design

This section will introduce you to the project we'll develop and the Ant Design React library.

To illustrate the real-world use of UmiJS and Ant Design, we’ll develop a front-end application for a CRM system.

CRM (Customer Relationship Management) system is a business application that allows a company to approach a customer, offer a solution, and develop a relationship from various strategic contacts to sell the right solution to the customer and guarantee their satisfaction.

In our example, the application has three main features: dashboard with various reports, registry and track of opportunities, and registry of customers.

We’ll also guarantee that our application is easy to be extended and modified in the face of business requirements, has a clean code style, and supports internationalization.

To build the interfaces of our front-end application, we’ll use the Ant Design React library. Let’s learn more about this library and the Pro Components.

Introduction to Ant Design components

The Ant Design library is a React components library created following the design principles of the Ant Design system. The Ant Design library was written in TypeScript and offers predictable static types, internationalization support, and theme customization. The library is also deeply integrated with UmiJS, so it's easy to customize the theme and set internationalization support using it with UmiJS.

You can browse the library and look for components at https://ant.design/components/overview/. On this documentation page, you will find detailed descriptions of every library component and use case examples followed by their respective code.

We’ll also use some components from Pro Components, a set of components derived from Ant Design, to provide a high level of abstraction, making easier the task of building complex interfaces. You can look for Pro Components at <https://procomponents.ant.design/en-US/components.>

In this section, you learned about the Ant Design React library and were introduced to the project we will build. Let's start building the interfaces by setting the default layout and theme.

Setting up the layout and theme

In this section, we’ll set up a default layout using plugin-layout and customize our application theme changing the default LESS variables used by Antd. To do that, follow these steps:

1. We’ll use the project we created in the previous chapter. Add the plugin-layout configuration to the config.ts file as follows:

layout: {

navTheme: 'light',

layout: 'mix',

contentWidth: 'fluid',

fixedHeader: false,

fixSiderbar: true,

colorWeak: false,

title: 'Umi CRM',

locale: true,

pwa: false,

logo: 'https://img.icons8.com/ios-filled/50/ffffff/customer-insight.png',

iconfontUrl: '',

},

This configuration adds a page header and a menu for all pages, defines the application name and logo, and enables the plugin-locale in layout components.

You can also change the layout as you need. For example, you can set the menu to appear in the header instead of a side menu, changing the layout property to top.

1. Let’s also change the primary color that all interfaces and components use. Add this configuration to the config.ts file:

theme: {

'primary-color': '#1895bb',

},

The theme configuration changes the default values of LESS variables used by Ant Design components.

Important Information

You can find all the default LESS variables at: https://github.com/ant-design/ant-design/blob/master/components/style/themes/default.less

1. Next, we need to add some configurations for plugin-layout in the app.tsx file. Add the following to the app.tsx file:

import routes from '../config/routes';

import { RunTimeLayoutConfig } from 'umi';

export const layout: RunTimeLayoutConfig = () => {

return {

routes,

rightContentRender: () => <></>,

onPageChange: () => {},

};

};

With this configuration, we set the routes the plugin that will build on the side menu.

1. To display menu items correctly in the side menu, adjust the route configuration as follows:

routes.ts

export default [

{

path: '/login',

component: '@/pages/Login',

layout: false,

},

{

path: '/',

name: 'home',

icon: 'home',

component: '@/pages/Home',

},

];

We defined the routes to the login page and the home page. The property layout: false will make the default layout not appear on the login page. The properties name and icon define how the Home page will appear on the side menu.

Ant Design provides the icon, and you can look for other icons at https://ant.design/components/icon/.

Now let’s finish our default layout by adding a quick menu, a language selector, and changing its style to use our primary color.

Adding right side content to the layout header

First, let’s create two new components: the HeaderMenu, which will contain the user’s avatar, user’s name, and the logout menu item; and the HeaderOptions component, which will include the HeaderMenu and the SelectLang components. The SelectLang is a component provided by UmiJS to change between languages supported by the application through the plugin-locale.

Follow these steps to create the HeaderMenu component:

1. Create a new folder named components in the src folder and inside it, create a new folder named HeaderMenu.
2. In the HeaderMenu folder, create two files: index.tsx and index.less.
3. In the index.tsx file create the component as follows:

import { Avatar, Dropdown, Menu } from 'antd';

import styles from './index.less';

import { LogoutOutlined } from '@ant-design/icons';

export default function HeaderMenu() {

const options = (

<Menu className={styles.menu}>

<Menu.Item key="center">

<LogoutOutlined /> Logout

</Menu.Item>

</Menu>

);

return (

<Dropdown

className={styles.dropdown}

overlay={options}>

<span>

<Avatar

size="small"

className={styles.avatar} />

<span

className={`${styles.name} anticon`}>

John Doe

</span>

</span>

</Dropdown>

);

}

In this component, we use the Antd library Menu component to render the logout menu item and the Dropdown and Avatar components to render the user avatar and user’s name. The logout option will appear when you mouse over the username or avatar.

1. Create the CSS classes for Avatar and Dropdown components in the index.less file as follows:

.avatar {

color: white;

background-color: #1895bb;

margin: 0px 10px;

}

.dropdown {

display: flex;

flex-flow: row nowrap;

cursor: pointer;

align-items: center;

float: right;

height: 48px;

margin-left: auto;

overflow: hidden;

}

Now follow these steps to create the HeaderOptions component:

1. In the components folder, create a new folder named HeaderOptions. Inside it, create a file called index.tsx.
2. In the index.tsx file create the component as follows:

import { Space } from 'antd';

import { SelectLang } from 'umi';

import HeaderMenu from '../HeaderMenu';

export default function HeaderOptions() {

return (

<Space>

<HeaderMenu />

<SelectLang />

</Space>

);

}

In this component, we use the Space component from Antd and the recently created HeaderMenu component with the SelectLang component from UmiJS to render the layout header options.

Ícone

Descrição gerada automaticamente

Figure 2.1: The language selector (SelectLang component)

Now to add the HeaderOptions component to the layout, follow these steps:

1. Import the HeaderOptions component adding this line to the app.tsx file:

import HeaderOptions from './components/HeaderOptions';

1. Add the HeaderOptions component to the configuration rightContentRender as follows:

export const layout: RunTimeLayoutConfig = () => {

return {

routes,

rightContentRender: () => <HeaderOptions />,

onPageChange: () => {},

};

};

Now the HeaderOptions component should appear in the layout header as follows:

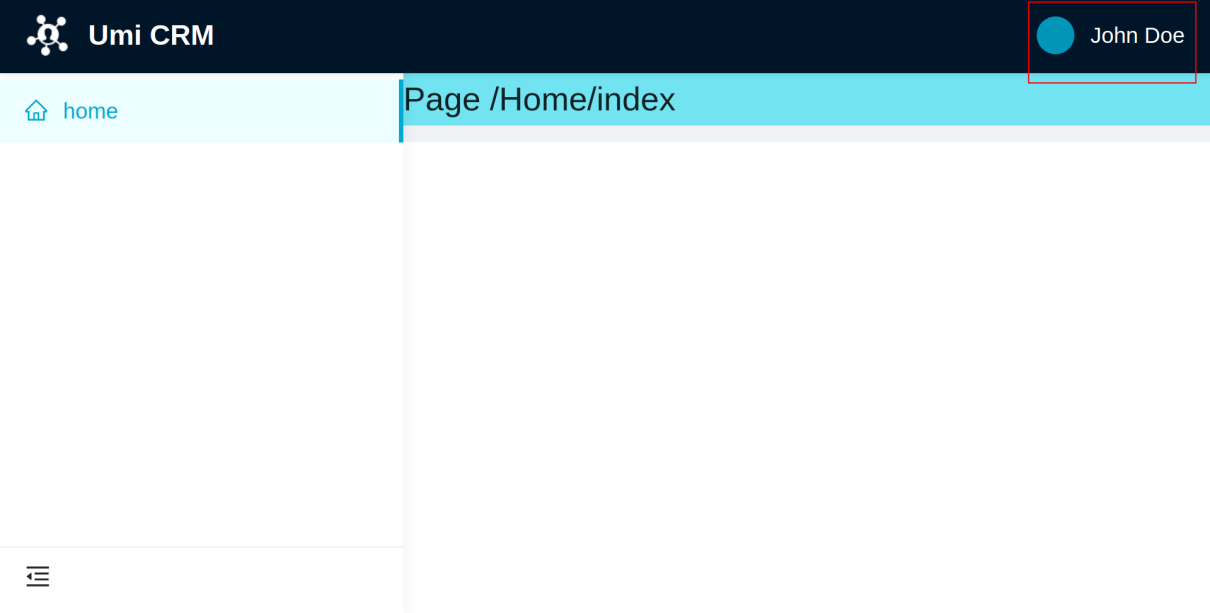


Figure 2.2: Layout right content

You may have noticed that the language selector did not appear. It will appear once we add language support to the project.

To finish setting up our layout, let’s add the primary color. We can customize the CSS class applied to the layout header using the global.less file to add the primary color.

UmiJS will apply the global.less file before all other style sheets, so when you need to customize some style or apply it across all interfaces, you can do that using this file.

Follow these steps to customize the CSS class applied to the layout header:

1. Create a new file under the src folder named global.less.
2. Add this style to the global.less file:

.ant-pro-global-header-layout-mix {

background: #1895bb;

background: linear-gradient(50deg, #1895bb 0%, #14cfbd 100%);

}

We added a background gradient using our primary color to the CSS class and applied that to the global header.

Tip

You can find CSS classes applied to HTML elements inspecting the page with your browser dev tools. Usually, you need to press F12 and look for the elements tab.

Now the layout header should look like this:



Figure 2.3: Layout header with primary color applied

In this section, we set up the default layout for all pages by configuring the plugin-layout and customizing the layout using the global.less file. We also created the components to render the user avatar, name, and language selector. Now let’s build the home page and set up internationalization.

Creating the home page and setting up i18n

In this section, we’ll create the home page and set up the application’s internationalization for Portuguese and English.

Our home page will be composed of two main components: PageContainer and the ProTable. When users log in to the application, we want them to see some information such as the user’s name, role, and a list of recently opened opportunities. To do that, follow these steps:

1. Let’s start by adding the PageContainer component to the index.tsx file under src/pages/Home folder as follows:

import styles from './index.less';

import { PageContainer } from '@ant-design/pro-layout';

export default function IndexPage() {

return (

<PageContainer

header={{ title: undefined }}

style={{ minHeight: '90vh' }}

content={}

></PageContainer>

);

}

By default, the PageContainer component will render the page title you defined as the route name in the routes.ts file, but we set it to undefined as we don’t want to display the title on this page.

1. Now we’ll add some basic information to the content of PageContainer. We want that when the user logs into the application, they see a greeting followed by their name, role, and avatar, so go ahead and add the following information to the content property of PageContainer as follows:

content={

<div className={styles.pageHeaderContent}>

<div className={styles.avatar}>

<Avatar

alt="avatar"

className={styles.avatarComponent}

size={{ xs: 64, sm: 64, md: 64, lg: 64, xl: 80, xxl: 100 }}

icon={<UserOutlined />}

/>

</div>

<div className={styles.content}>

<div className={styles.contentTitle}>Hello John Doe, welcome.</div>

<div>Inside Sales | Umi Group</div>

</div>

</div>

}

Here we added the Avatar component from Antd followed by the greeting, user’s name, and role.

1. We also need to define the styles in the file index.less. Add these styles to the index.less file under src/pages/Home folder as follows:

@import '~antd/es/style/themes/default.less';

.pageHeaderContent {

display: flex;

.avatar {

flex: 0 1 72px;

& > span {

display: block;

width: 72px;

height: 72px;

border-radius: 72px;

}

.avatarComponent {

color: white;

background-color: @primary-color;

}

}

.content {

position: relative;

top: 4px;

flex: 1 1 auto;

margin-left: 24px;

color: @text-color-secondary;

line-height: 22px;

.contentTitle {

margin-bottom: 12px;

color: @heading-color;

font-weight: 500;

font-size: 20px;

line-height: 28px;

}

}

}

Notice that we imported a file called default.less from Antd. This file contains the default LESS variables used by Ant Design components to define the styles. We are using some of these variables in our CSS classes too.

I highly recommend you familiarize yourself with these variables; this will help you maintain a consistent style with the Ant Design specification. You can access the default.less file by pressing Ctrl and clicking on its import path, or you can see the file on GitHub at https://github.com/ant-design/ant-design/blob/master/components/style/themes/default.less.

The next component we'll add to our page is ProTable; this is a Pro Components component that abstracts the logic for manipulating a batch of data in a table.

1. To add the component, we need to install its package, so run this command to install it:

$ yarn add @ant-design/pro-table

1. Next, add the ProTable component as a child of PageContainer to the index.tsx file under src/pages/Home as follows:

<div style={{ width: '100%' }}>

<ProTable<any>

headerTitle="Recent opportunities"

pagination={{ pageSize: 5 }}

rowKey="id"

search={false}

/>

</div>

At this point, your home page should look like this:

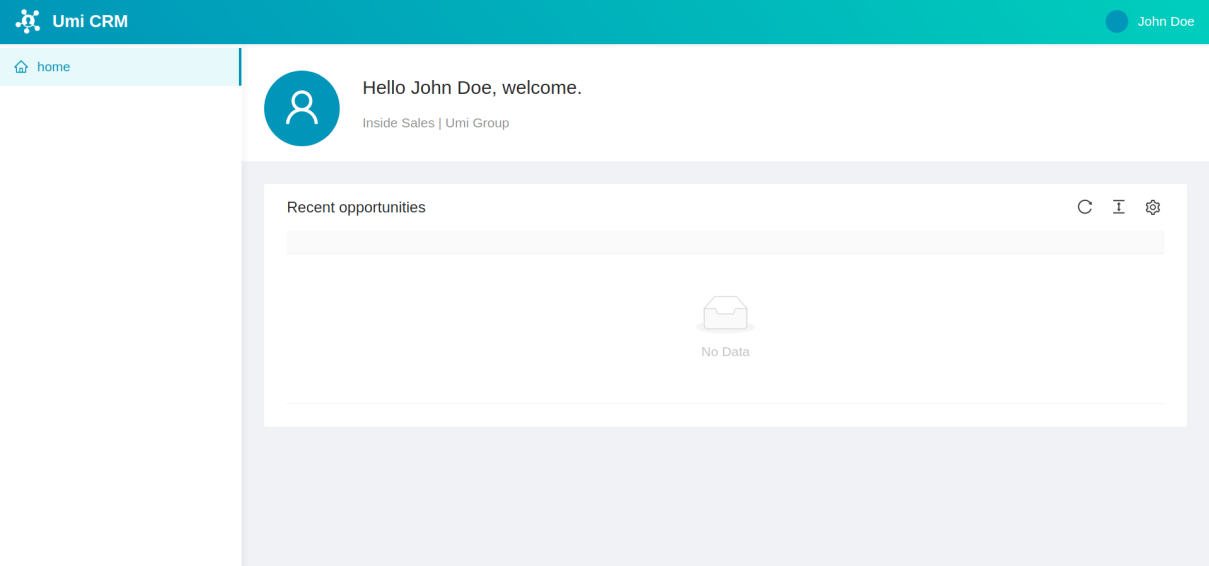


Figure 2.4: Home page interface

Now it’s time to add support for internationalization (i18n) in our application.

Setting up internationalization

To add support to i18n using the plugin-locale, first, we must move all the text we want to translate to multi-language files under the src/locales folder. I’ll build the entire application in English and Portuguese to demonstrate this feature, but you don’t need to worry about it; you can download the Portuguese files available at https://github.com/PacktPublishing/Enterprise-React-Development-with-UmiJs. Follow these steps to create our language files:

1. Create a file named en-US.ts under the src/locales folder, download the file pt-BR.ts, and place it under the same folder.
2. Type the texts for the home page in the en-US.ts file as follows:

export default {

'home.recents': 'Recent opportunities',

'greetings.hello': 'Hello',

'greetings.welcome': 'welcome',

};

1. We need to change the texts on the home page by the component FormattedMessage. Import the component in the index.ts file under the src/pages folder adding this line:

import { FormattedMessage } from 'umi';

1. And change the text by the component as follows:

<div className={styles.content}>

<div className={styles.contentTitle}>

<FormattedMessage id="greetings.hello" /> John Doe,{' '}

<FormattedMessage id="greetings.welcome" />.

</div>

<div>Inside Sales | Umi Group</div>

</div>

1. Also change the ProTable headerTitle property as follows:

headerTitle={<FormattedMessage id="home.recents" />}

The FormattedMessage component property id must match the same key in the en-US.ts and the pt-BR.ts file. As you select the language, the component will render the corresponding text.

We want the menu titles translated, so let’s add files to translate menu items. Follows these steps:

1. Create a new folder named en-US under the src/locales folder. Under the en-US folder, create a new file named menu.ts.
2. Add the text to render in the menu item to the menu.ts file as follows:

export default {

'menu.home': 'Home',

};

The key for the text needs to match the name property in the routes.ts file. The plugin-locale will render the correct text as you change between languages.

1. Import the menu.ts file into the en-US.ts file as follows:

import menu from './en-US/menu';

export default {

...menu,

'home.recents': 'Recent opportunities',

'greetings.hello': 'Hello',

'greetings.welcome': 'welcome',

};

1. We also need to add the menu.ts file to the Portuguese language, so create a new folder named pt-BR, under the src/locales folder, download the file menu.ts available at https://github.com/PacktPublishing/Enterprise-React-Development-with-UmiJs, and place it under the pt-BR folder.

Now you can change the application’s language using the language selector on the top of the page, as shown in the following screenshot:

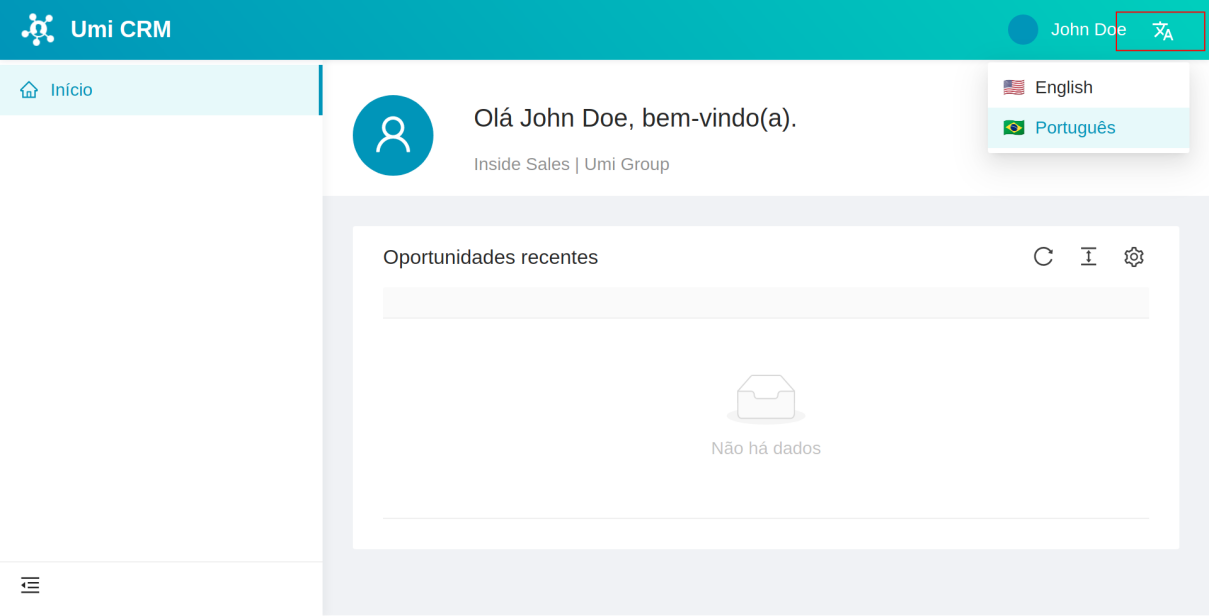


Figure 2.5: Home page with the Portuguese language selected

In this section, we created the home page using the PageContainer and ProTable components. We also set up internationalization by creating multi-language files under the folder src/locales and using the component FormattedMessage to replace the texts with their corresponding translations.

Now you’ll use what you learned to create the opportunity and customers pages.

Creating the opportunities and customers pages

In this section, we’ll build the opportunities and customers pages.

The opportunities page allows users to browse and register a new sale opportunity. A sale opportunity occurs when a customer seems interested in buying a product or service. On this page, we register and track all activities taken until the opportunity is won, that is, when the customer buys the product or service, or until opportunity is lost, or when the customer buys the product from a competitor or gives up on buying the product or service.

The customers page allows users to register and search for customers' contact information.

These two pages are similar; they use the ProTable component to list the opportunities and customers registered. Run the following commands to generate the two pages:

$ yarn umi g page /Customers/index --typescript --less

$ yarn umi g page /Opportunities/index --typescript --less

Now, let’s start with the customer’s page. Follow these steps to build the costumer page interface:

1. Add the PageContainer and the ProTable component to the file index.tsx under the src/pages/Customer folder as follows:

import { PlusOutlined } from '@ant-design/icons';

import { Button } from 'antd';

import ProTable from '@ant-design/pro-table';

import { FormattedMessage, getLocale } from 'umi';

import { PageContainer } from '@ant-design/pro-layout';

export default function Page() {

return (

<PageContainer style={{ minHeight: '90vh' }}>

<ProTable<any>

rowKey="id"

headerTitle={<FormattedMessage id="table.customer.title" />}

search={{ labelWidth: 'auto' }}

pagination={{ pageSize: 5 }}

dateFormatter="string"

locale={getLocale()}

toolBarRender={() => [

<Button key="button" icon={<PlusOutlined />} type="primary">

<FormattedMessage id="table.new" />

</Button>,

]}

/>

</PageContainer>

);

}

Notice that we use the FormattedMessage component to render the texts on this page, so we need to add these texts to multi-language files in the src/locales folder.

1. Add the texts present on the customers page to the en-US.ts file as follows:

import menu from './en-US/menu';

export default {

...menu,

'home.recents': 'Recent opportunities',

'greetings.hello': 'Hello',

'greetings.welcome': 'welcome',

'table.new': 'New',

'table.customer.title': 'Customers',

};

1. Now, to access the customer page, we need to define its route in the file routes.ts as follows:

{

path: '/customers',

name: 'customers',

icon: 'user',

component: '@/pages/Customers',

},

1. Add the customer menu item title to the menu.ts file under src/locales/en-US folder as follows:

export default {

'menu.home': 'Home',

'menu.customers': 'Customers',

};

Now, let’s build the opportunity page following the steps demonstrated previously:

1. For the headerTitle property of ProTable, type the following:

headerTitle={<FormattedMessage id="table.opportunity.title" />}

1. Define the opportunities page route as follows:

{

path: '/opportunities',

name: 'opportunities',

icon: 'AccountBook',

component: '@/pages/Opportunities',

},

1. Don’t forget to add the texts to the en-US.ts and menu.ts files.

The result should look like this:

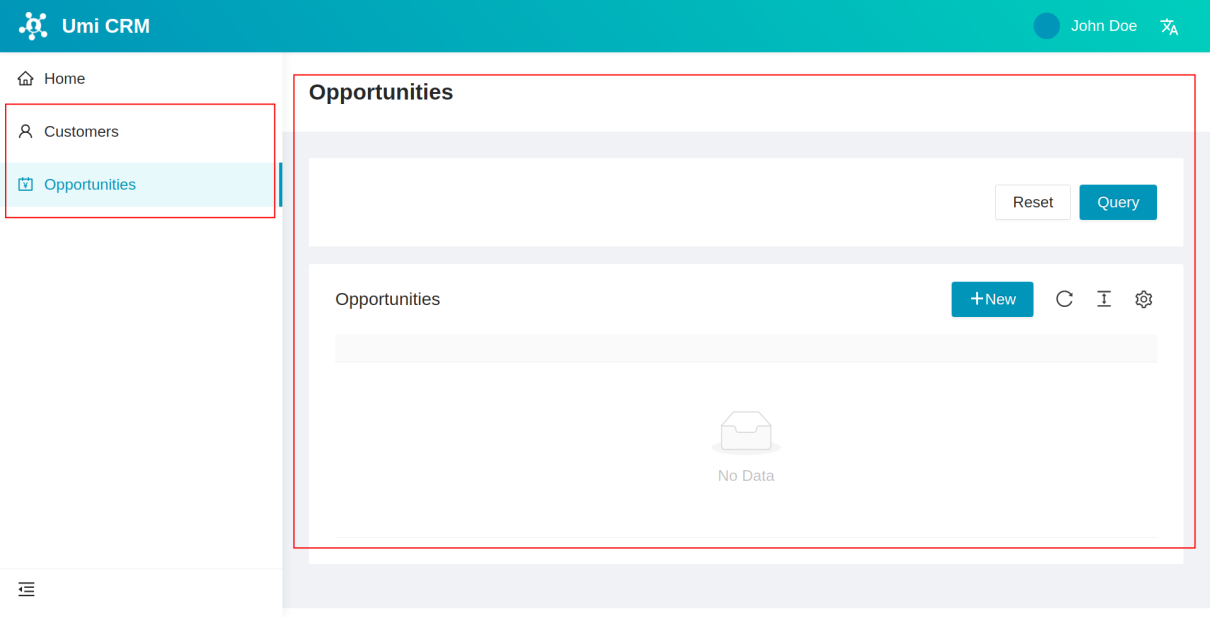


Figure 2.6: Opportunities page layout and menu items

In this section, we created the opportunities and customers pages using the ProTable component and supporting internationalization. Next, we’ll build the last page in this chapter, the reports pages.

Creating the reports page

Now, we’ll build the reports page. Users can access helpful information on this page to get insights into the sales lifecycle. We’ll add three charts to this page using the chart component library bizcharts.

The bizcharts library is focused on business scenarios and dedicated to creating professional data visualization solutions. It’s also an open-source project licensed under MIT license. You can learn more about bizcharts at <https://bizcharts.net/.>

1. First, run this command to install the bizcharts package:

$ yarn add bizcharts

1. Next, run this command to generate the reports page:

$ yarn umi g page /Reports/index --typescript --less

Now, follow these steps to create the reports page interface:

1. Let’s define the page layout with antd components. Add the following components to the index.tsx file under the src/pages/Reports folder:

import { PageContainer } from '@ant-design/pro-layout';

import { Row, Col, Card } from 'antd';

import { FormattedMessage } from 'umi';

import {

Chart,

Coordinate,

Axis,

Legend,

Interval,

Tooltip,

Interaction,

} from 'bizcharts';

const colProps = {

style: { marginBottom: 24 },

xs: 24,

sm: 12,

md: 12,

lg: 12,

xl: 12,

};

export default function Page() {

return (

<PageContainer>

<Row gutter={24}>

<Col {...colProps}></Col>

<Col {...colProps}></Col>

</Row>

<Row gutter={24} style={{ padding: 10 }}></Row>

</PageContainer>

);

}

We defined the layout with two responsive rows, which the first row has two responsive columns. The variable colProps set how the columns should adjust their size at different breakpoints.

1. Now, let’s add the first chart. This chart will show the four most important opportunities classified by the CRM analytics service. Add the Chart component from bizcharts inside the first column of the first row as follows:

<Col {...colProps}>

<Card title={<FormattedMessage id="chart.top" />}>

<Chart height={200} data={[]} autoFit>

<Coordinate transpose />

<Axis name="name" label={false} />

<Axis

name="revenue"

label={{

formatter: (text) => `$ ${text}`,

}}

/>

<Interval

type="interval"

label={["name", (name) => <>{name}</>]}

tooltip={{

fields: ["name", "revenue"],

callback: (name, revenue) => {

return { name: name, value: `$ ${revenue}` };

},

}}

color={["name", "#3936FE-#14CCBE"]}

position="name\*revenue"

/>

<Interaction type="element-single-selected" />

</Chart>

</Card>

</Col>

We can configure the Chart component with its children components. We set the chart to invert the X and Y axis with the Coordinate component. With the component Axis, we defined a new axis called revenue. The Interval component described the chart type and the keys that will populate the axis using the position property.

Notice that we set an empty array in the data property. We’ll put the information we want to display to the data property in the future.

1. Let’s add the second chart. This chart will show where the customers come from and in what proportion. Add the Chart component to the second column of the first row as follows:

<Col {...colProps}>

<Card title={<FormattedMessage id="chart.leads" />}>

<Chart

height={200}

data={[]}

scale={{

percent: {

formatter: (val: any) => {

val = val \* 100 + "%";

return val;

},

},

}}

autoFit

>

<Coordinate type="theta" radius={0.95} />

<Tooltip showTitle={false} />

<Axis visible={false} />

<Legend position="right" />

<Interval

position="percent"

adjust="stack"

color="source"

style={{

lineWidth: 1,

stroke: "#fff",

}}

/>

<Interaction type="element-single-selected" />

</Chart>

</Card>

</Col>

In this chart, we set the Coordinate component to cylindrical coordinates to generate a pie chart. With the Interaction component, we set the chart to react when it is mouse over or clicked.

1. The last chart shows the opportunities gained and lost by month. Add the Chart component to the second row as follows:

<Row gutter={24} style={{ padding: 10 }}>

<Card

style={{ width: '100%' }}

title={<FormattedMessage id="chart.month" />}

>

<Chart height={300} padding="auto" data={[]} autoFit>

<Interval

adjust={[

{

type: 'dodge',

marginRatio: 0,

},

]}

color={['name', '#3776E7-#14CCBE']}

position="month\*value"

/>

<Tooltip shared />

</Chart>

</Card>

</Row>

1. To finish the reports page, let’s add the following texts to the en-US.ts file under the src/locales folder:

'chart.top': 'Top opportunities',

'chart.leads': 'Leads by source',

'chart.month': 'Oppotunities Won/Lost by month',

1. And add the following text to the menu.ts file under the src/locales/en-US folder:

'menu.reports': 'Reports',

1. Finally, configure the reports page route in the routes.ts file as follows:

{

path: '/reports',

name: 'reports',

icon: 'BarChartOutlined',

component: '@/pages/Reports',

},

Now, the reports page is finished and should look like this:

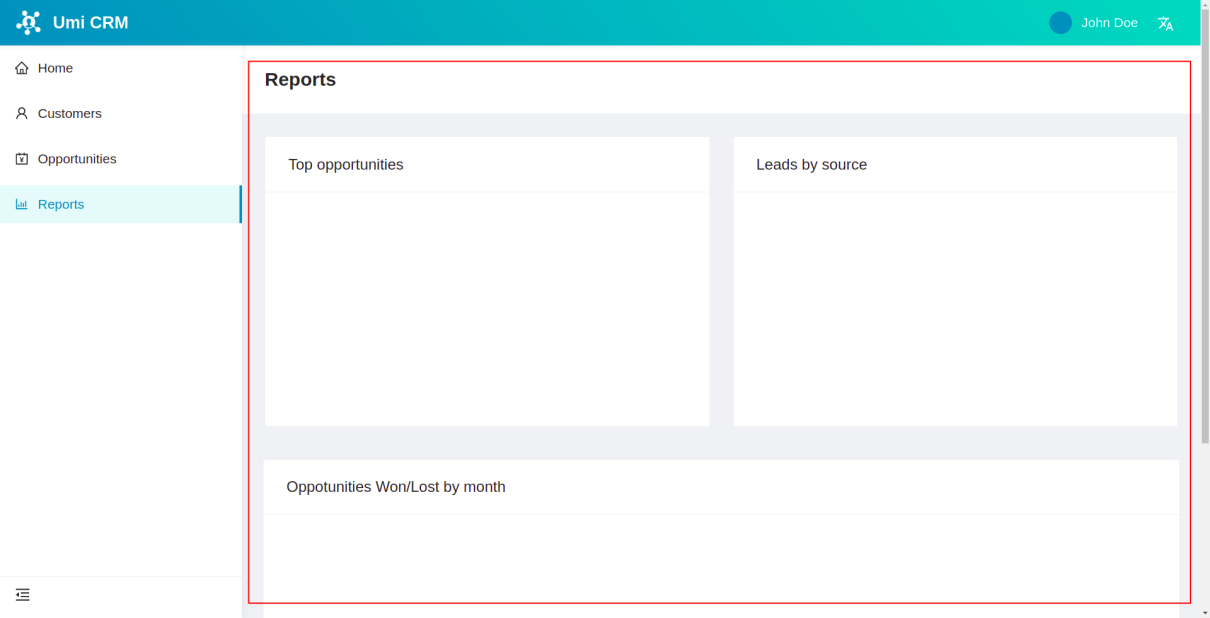


Figure 2.7: Reports page layout

Notice that all chart cards are empty because we defined empty arrays in all chart data properties. We’ll generate the data required to show the charts in the next chapter.

In this section, we create the reports page using the bizcharts library. We added three charts to our page: a bar chart called Top opportunities, a pie chart called Leads by source, and a bar chart called Opportunities Won/Lost by month.

Summary

In this chapter, you were introduced to the project we’ll build, the Ant Design React library, and the Pro Components. You also learned how to configure the layout using the UmiJS layout plugin and define and customize the global layout using the global.less file. You learned how to customize the application theme by changing the default LESS variables used by Ant Design components.

We also created and defined our application layout’s right content to show the user’s name, avatar, and a language selector. You learned how to set up internationalization using the UmiJS locale plugin and created the home page. Next, we made the customer and opportunities page using the ProTable component.

Finally, we built the reports using Antd components to define the layout and the bizcharts library to render three charts.

In the next chapter, we’ll generate a mock API, make requests to the backend and learn how to use services and models.