Documentation

Project Name: 3 Sigma CRM
Technology Used: React Typescript
Version: **React** - v18.2.0
Typescript -v.4.7.4
Redux - v4.2.0
Redux-Thunk - v2.4.1

Few important dependencies

Validation	-	Yup (v0.32.11)
UI	-	Reactstrap(v9.1.2)
Api call	-	Axios(v0.27.2)

Folder Structure

Public Public folder contains root ***index.html*** file, It contains same code as the reference UI code provided. Few things are added from my side related to facebook login.

```
<script>
  window.fbAsyncInit = function() {
    FB.init({
      appId      : '776734700302578',
      cookie     : true,
      xfbml      : true,
      version    : 'v15.0'
    });
    FB.AppEvents.logPageView()
  };
</script>
```

Inside public folder there is ***css*** folder, where a style.css file is there, code in file is same as per reference code, just modified few styles from my end.

Public -> fonts - Gillory fonts are added here.
Public -> Js - Same as reference code.

Folder Structure -

Actions -

- **actions.tsx**
- **actionTypes.tsx**

Activity names / types are defined in activityTypes.tsx
All redux actions are managed in actions.tsx file.

Components - all common components are placed in this folder.

Config - common config file to change the api endpoint,if api endpoint changed or moved to staging or production, api endpoint can be modified here.

```
export const API_URL = 'http://13.232.122.198:5000/';
```

Reducers - All redux business logic is placed in redux reducers. Here a switch case statement is written where it will check what is the action type & based on that action type it will perform the business logic on the initial state & return updated state.

Inside reducers folders there is one ***rootReducers*** where all other reducers are imported and combined using ***combinedReducer.*** All the reducers are named based on module or functionality so it will not be hard to distinguish between this files.

Routes - this folder contains a ***privateRoute.js*** file where we are checking if the user is logged in or not. If the user is already logged in then it will allow the user to access authenticated pages if the user is not logged in it will redirect to the login page again.

Services service folder is used to perform all api calls, files are named after modules. Example activityService.tsx file will have all api calls related to activities like create activity, get all activities etc.

Inside the services folder there is a file named ***getHeaderOptions.tsx*** it's a common file created to pass header options in api call.

For post apis - axios.post(apiUrl, data, headerOptions)

apiUrl - api endpoint
Data - data that need to be passed as request payload
headerOptions i.e content type, auth_token

Same with put apis

For get apis - axios.get(apiUrl, headerOptions)

Store folder has a file named store.tsx where a common store is created for the whole project. This common store is responsible for all the states or data that will be stored as a single source of truth & used to communicate between components.

Utils All utility functions are placed here within this folder. For example debounce.js is a utility function which is used throughout the project in search feature to optimize frequent api calls.

Another example is ***countdown.tsx*** which is created for a countdown feature in the verify otp page. If an application countdown feature is needed the same code can be reused there.

Views all pages are placed here.like login, signup, leads page, products page, quotation page etc.

Login Page -

In login page ***react-phone-input-2*** is used for country code & phone numbers

This 2 npm packages are used for social media login in login page

```
React-google-login
react-facebook-login
```

Yup is used for validations. Example sign up page. For validations using yup a schema needs to be created, the same is created in every page where required.

```
let schema = yup.object().shape({
        firstName: yup.string().required('First Name is required'),
        lastName: yup.string().required('Last Name is required'),
        email: yup.string()
            .email('Email is invalid')
            .required('Email is required'),
        phone: yup.string()
            .required("Mobile Number is Required")
            .matches(regExp, { message: 'Mobile Number is not valid',
excludeEmptyString: true }),
        companyName: yup.string().required('Company Name is required'),
        category: yup.string().required('Category is required'),
        teamSize: yup.string().required('Team size is required')
    });
```

To validate the schema this code is written, same code will be used wherever we need to add validation.

```
schema.validate(signupData, { abortEarly: false }).catch((err) => {
                const errors = err.inner.reduce((acc: any, error: { path:
string; message: string }) => {
```

```
                return {
                    ...acc,
                    [error.path]: error.message,
                }
            }, {});

            setErrors((prevErrors) =>
                update(prevErrors, {
                    $set: errors,
                })
            );
        });
```

Here signupData is form data that needs to be validated.
abortEarly: false is used to validate all schemas. this will prevent validation to get terminated after an error. All validation errors will be evaluated from it.

Leads All actions related to leads

Common drawer component is used to show add / edit lead form and in other pages as well like add / edit quotation form & add/ edit category form

All the forms like add / Edit lead, add / edit quotation & add / edit category etc have different form fields so forms are passed dynamically as child components to manage the reusability.

drawerTitle
Size
This can be passed dynamically for drawer title as text
Size can be passed in the drawer component for drawer size. Size can be either ***xs, sm, md, lg.***

File names are kept as per functionality so it will not be difficult for user to find the code. Example if you are looking for code of details of lead page, code is present in ***leadDetails.tsx.***