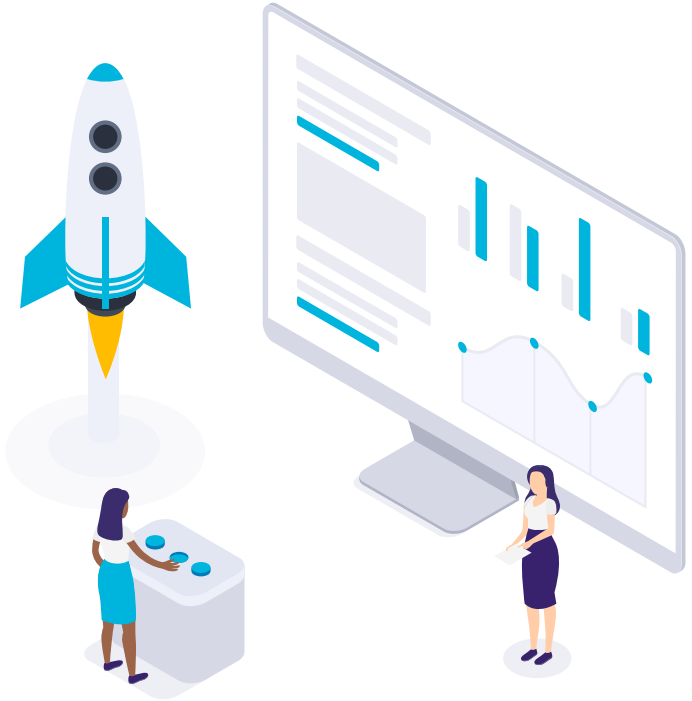
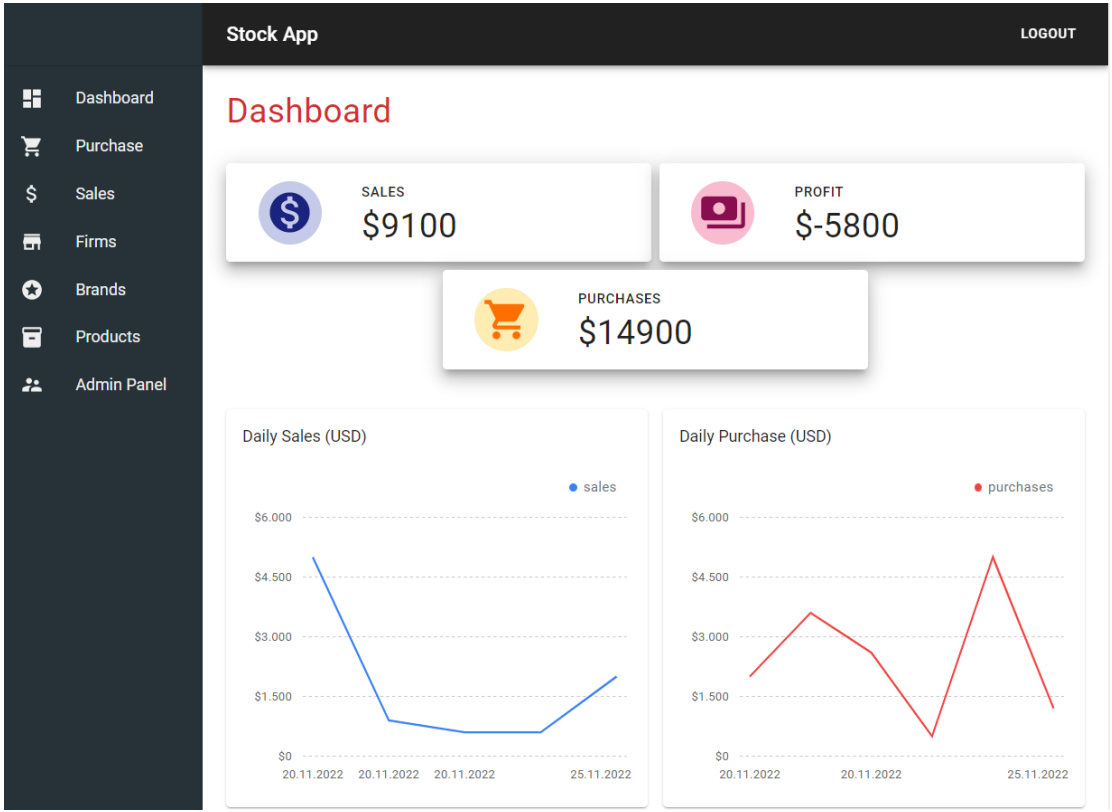
1 Stock App Info



Stock App



The Dashboard of Stock App

CLARUSWAY
WAY TO REINVENT YOURSELF

Used Libraries/APIs

- React-Redux (Redux Toolkit)
- Redux-Persist
- Material UI
- React Router
- Axios
- Formik, Yup
- Toastify
- Tremor

React Redux

- We have a lot of components. So, we should share some information like **authorization** and some **stock data** with the child components.
- To share **these states** between components we used **React-Redux** library.
- **React redux** is **one of the most popular** global state management library.
- It is widely used in the react projects especially if the states are needed to be **frequently** updated.
- The another global state management alternative is Context API. It is an built-in API. It is especially more suitable for **less state** count and **more static states**.
- We might also use **props drilling** method to share information but in this case, our project structure would be **more complicated**.

React Redux

- We used **Redux Toolkit** Package in this project. The Redux Toolkit helps to write more efficient Redux logic.

- It minimizes the **configurations** and **boilerplate** codes. It also includes **RTK Query** package to handle API request with Redux logic.

- We didn't use **RTK Query** in this project but we plan to use it for better performance in near future.

- Moreover, Redux Developers strongly **recommend** using **Redux Toolkit** for all Redux apps.

React Redux

- Example of a redux slice

authSlice.jsx

```jsx
import { createSlice } from "@reduxjs/toolkit";

const authSlice = createSlice({
  name: "auth",

  initialState: {
    currentUser: null,
    loading: false,
    error: false,
    isAdmin: false,
    token: null,
  },
  reducers: {
    fetchStart: (state) => {
      state.loading = true;
      state.error = false;
    },
    loginSuccess: (state, { payload }) => {
      state.loading = false;
      state.currentUser = payload?.user?.username;
      state.isAdmin = payload?.user?.is_superuser;
      state.token = payload?.key;
    },
```

Redux-Persist

- We know that the states always are **reset** to the initial values in every **refresh**. This is not good for user experience.

- **Redux-Persist** takes the Redux state object and saves it to persisted storage. Then on app launch it retrieves this persisted state and saves it back to redux.

- **Redux-persist** provides different storage to persist data like local storage, session storage or async storage.

- We used **Redux-Persist** library to persist our **authorization** states.

Redux-Persist

- Example of redux-persist's configuration

store.jsx

```jsx
import { configureStore } from "@reduxjs/toolkit";
import authReducer from "../features/authSlice";
import stockReducer from "../features/stockSlice";
import storage from "redux-persist/lib/storage/"; // defaults to localStorage for web

import {
  persistStore,persistReducer,FLUSH,REHYDRATE,PAUSE,PERSIST,PURGE,REGISTER,
} from "redux-persist";

const persistConfig = {
  key: "root",
  storage,
};

const persistedReducer = persistReducer(persistConfig, authReducer);

const store = configureStore({
  reducer: {
    auth: persistedReducer,
    stock: stockReducer,
  },

  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
      },
    }),
  devTools: process.env.NODE_ENV !== "production",
});

export const persistor = persistStore(store);
export default store;
```

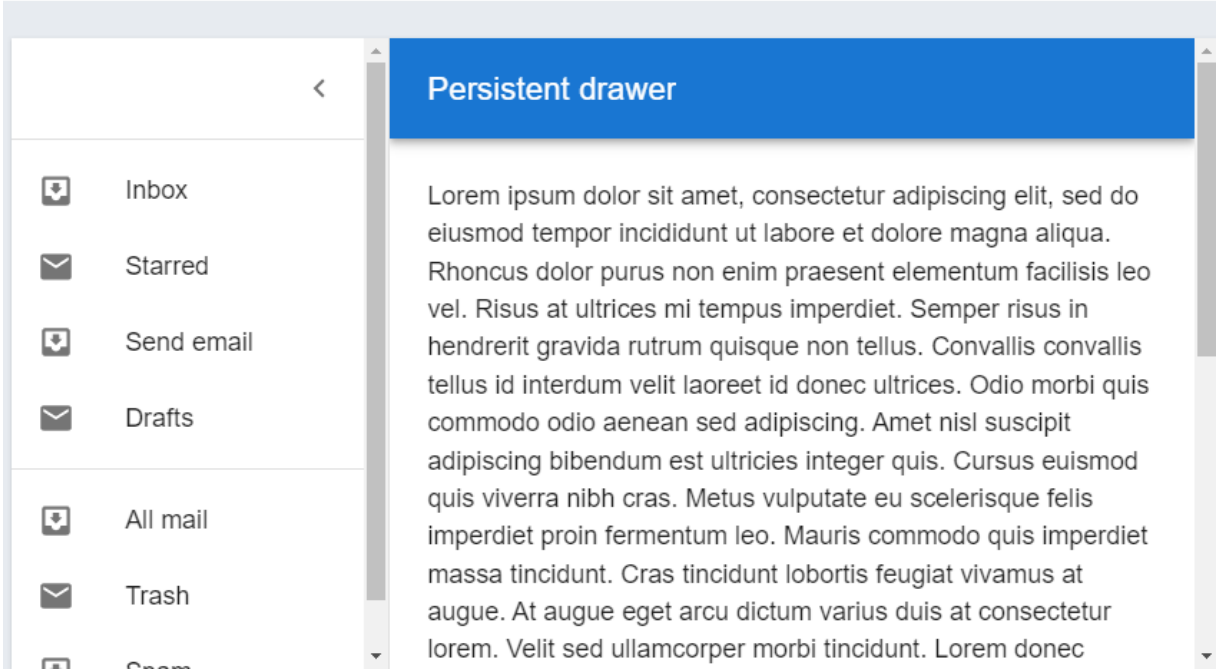
Material UI 

- We used Material UI for User Interface library. It has a lot of ready to use components like Drawer, Appbar etc. to build a dashboard.

- The advantages of MUI.
  - It's based on **Google Material Design**
  - It is widely used in React applications
  - Open-Source and Highly **customizable**
  - **Elegance** design
  - Very **huge** pre-built component library
  - Mobile **compatible**

Material UI

- Example of MUI Drawer component.



React-Router

- We used **React Router** library to enable **Client Side Routing**
- It also allows us to use **browser history** features while preserving the right application view.
- It allows us to build a single-page web application with navigation without the **page refreshing** as the user navigates.
- It is **mostly used routing** library in react applications.

React-Router

- Routes examples

AppRouter.jsx

```jsx
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

const AppRouter = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="register" element={<Register />} />
        <Route path="stock" element={<PrivateRouter />}>
          <Route path="" element={<Dashboard />}>
            <Route index element={<Home />} />
            <Route path="purchases" element={<Purchases />} />
            <Route path="sales" element={<Sales />} />
            <Route path="products" element={<Products />} />
            <Route path="firms" element={<Firms />} />
            <Route path="brands" element={<Brands />} />
          </Route>
        </Route>
      </Routes>
    </Router>
  );
};

export default AppRouter;
```

Axios

- We used **axios** library to handle API request.

- Advantage of axios
  - It supports to create a new **instance** with a custom config.
  - It allows to **intercept** requests or responses before they are handled.
  - Automatic transforms for JSON data

- By the help of axios, we can create the instances with custom config. So We avoided the code **repetition** for every fetch.

Axios

AXIOS

- **Custom hook** example with **axios instances**

useAxios.jsx

```jsx
import axios from "axios";
import { useSelector } from "react-redux";

const BASE_URL = "https://10001.fullstack.clarusway.com/";

//* Axios Instance for Public API Request
export const axiosPublic = axios.create({
  baseURL: BASE_URL,
});

const useAxios = () => {
  const { token } = useSelector((state) => state.auth);

  //* Axios Instance for Private API Request
  const axiosWithToken = axios.create({
    baseURL: BASE_URL,
    headers: { Authorization: `Token ${token}` },
  });

  return { axiosWithToken };
};

export default useAxios;
```

Axios

AXIOS

- Another **custom hook** for fetching data from the API with **axios instances**

UseStockCalls.jsx

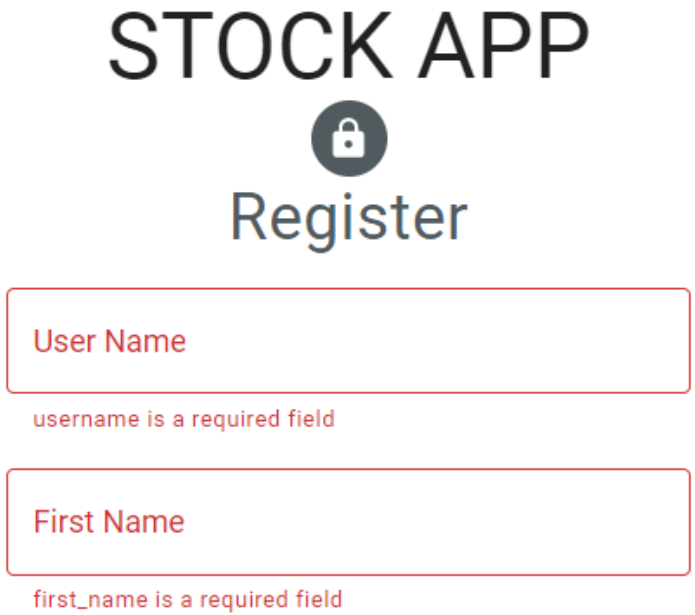
```jsx
import { useDispatch } from "react-redux";
import { fetchFail, fetchStart, getSuccess } from "../features/stockSlice";
import useAxios from "./useAxios";

const useStockCalls = () => {
  const dispatch = useDispatch();
  const { axiosWithToken } = useAxios();

  //!------------- GET CALLS ----------------
  const getStockData = async (url) => {
    dispatch(fetchStart());
    try {
      const { data } = await axiosWithToken.get(`stock/${url}/`);
      dispatch(getSuccess({ data, url }));
    } catch (error) {
      dispatch(fetchFail());
      console.log(error);
    }
  };

  //!------------- POST CALLS ----------------
  const postStockData = async (info, url) => {
    try {
      await axiosWithToken.post(`stock/${url}/`, info);
      toastSuccessNotify(`${url} successfuly added`);
      getStockData(url);
    } catch (error) {
      console.log(error);
      toastErrorNotify(`${url} can not be added`);
    }
  };
  return { getStockData, postStockData };
};

export default useStockCalls;
```

Formik and Yup

- **Formik** simplifies **form handling**. It creates local form states. So we can directly use them.

- **Yup** is a client-side **validation** library. It allows defining validation schema for the forms.

- **Formik**, **Yup** and **Material UI** can work together without problem. By the help of these trios we can easily create beautiful and efficient forms.



Formik and Yup

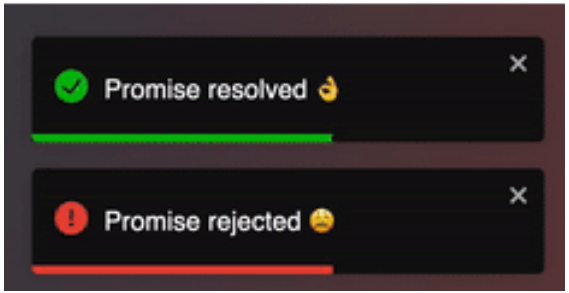
- Example of Formik and Yup usage

Login.jsx

```jsx
export const loginSchema = Yup.object().shape({
  email: Yup.string()
    .email("Please enter valid email")
    .required("Email is mandatory"),
  password: Yup.string()
    .min(8, "Password must have min 8 chars")
    .max(16, "Password must have max 16 chars")
    .matches(/\d+/, "Password must have a number")
    .matches(/[a-z]+/, "Password must have a lowercase")
    .matches(/[A-Z]+/, "Password must have an uppercase")
    .matches(/[!,?{}><%&$#£+-.]+/, " Password must have a special char"),
});

const Login = () => {
  const { login } = useAuthCalls();
  return (
    <Container maxWidth="lg">
      <Formik
        initialValues={{ email: "", password: "" }}
        validationSchema={loginSchema}
        onSubmit={(values, actions) => {
          login(values);
          actions.resetForm();
          actions.setSubmitting(false);
        }}
        component={(props) => <LoginForm {...props} />}
      ></Formik>
    </Container>
  );
};
export default Login;
```

Toastify

- **React-Toastify** allows to add notifications to our app with ease.
- It supports to style our toast messages.
- There are a lot capabilities to inform the user.



Toastify

- Example of Toast Notify functions

ToastNotify.js

```
import { toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
export const toastSuccessNotify = (msg) => {
  toast.success(msg, {
    autoClose: 1500,
    hideProgressBar: false,
    closeOnClick: true,
    pauseOnHover: true,
    draggable: true,
    progress: undefined,
  });
};
export const toastErrorNotify = (msg) => {
  toast.error(msg, {
    autoClose: 2000,
    hideProgressBar: false,
    closeOnClick: true,
    pauseOnHover: true,
    draggable: true,
    progress: undefined,
  });
};
```

Tremor

- Tremor is an open-source UI component library for building insightful **dashboards**. https://www.tremor.so/. It really simplify build a dashboard.

- Tremor is based on **Tailwindcss**, **Recharts** and **React**. It offers components, such as charts, layouts, or input elements, covering the essential parts of a dashboard or analytical interface.

- We used **Tremor** to create the **charts** of our dashboard and **Multi Select** element to filter our table data.

- The apparency of the Multi-Select Component is **very nice** and its usage is **very easy**.

Tremor

- Example of Chart component with Tremor

Chart.jsx

```jsx
import { LineChart } from "@tremor/react";

const Charts = () => {
  const dataFormatter = (number) =>
    `$${Intl.NumberFormat("us").format(number).toString()}`;

  const salesData = sales?.map((item) => ({
    date: item.createds,
    sales: Number(item.price_total),
  }));

  return (
    <Card sx={{ p: 2 }}>
      <Typography>Daily Sales (USD)</Typography>
      <LineChart
        data={salesData}
        dataKey="date"
        categories={["sales"]}
        colors={["blue"]}
        valueFormatter={dataFormatter}
        marginTop="mt-6"
      />
    </Card>
  );
};

export default Charts;
```

THANKS!

Any questions?