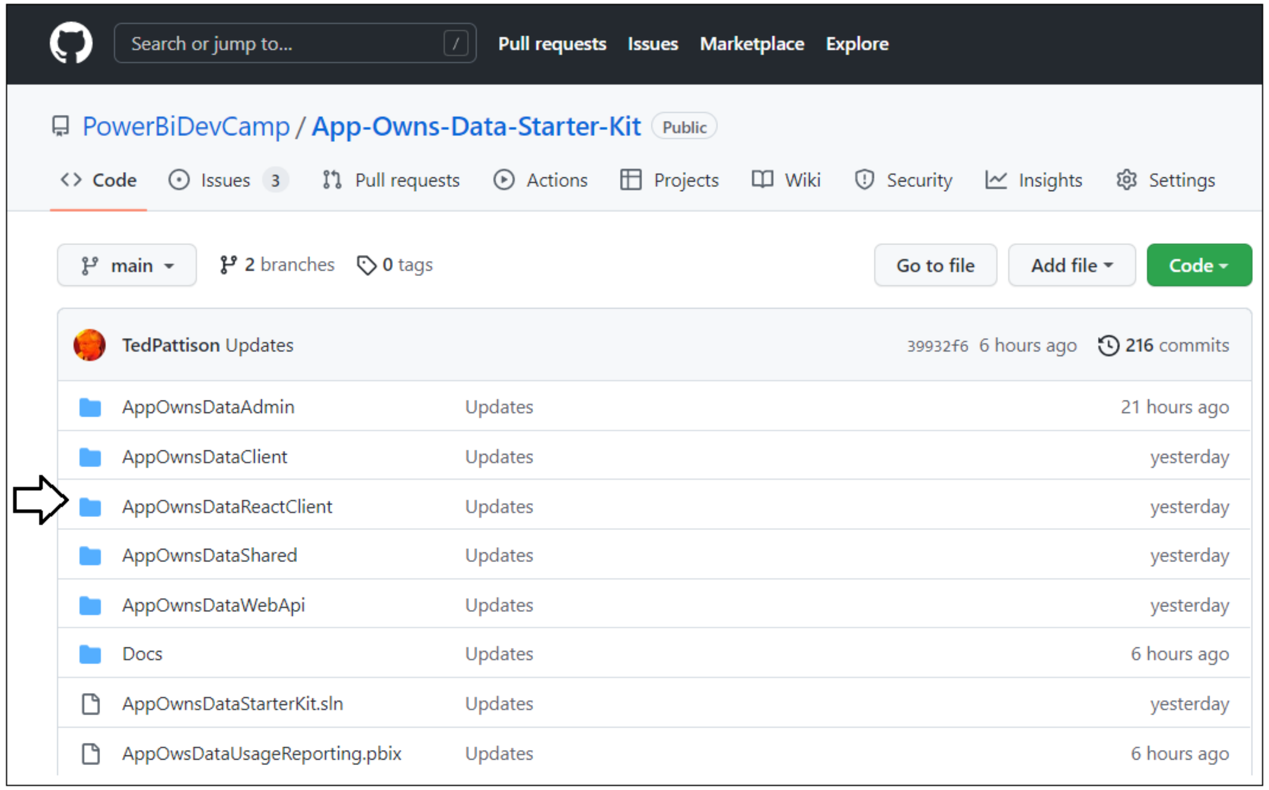
Microsoft Power BI

Modern React-JS Development with App-Owns-Data Embedding

Ted Pattison
Principal Program Manager
Power BI Customer Advisory Team (PBICAT)

Developer Sample used in This Session

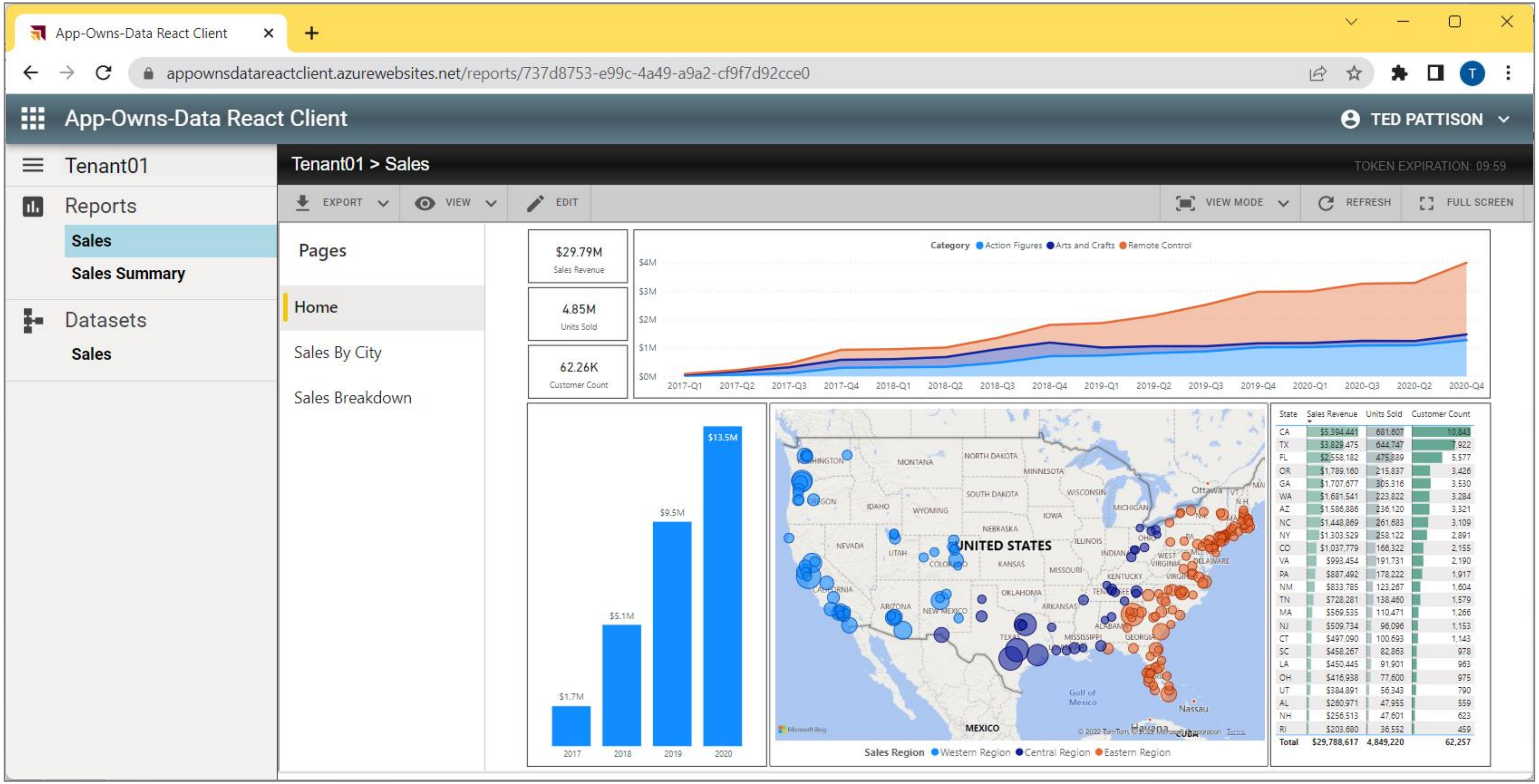
- **AppOwnsDataReactClient** is included as part of the **App-Owns-Data Starter Kit**
  - App-Owns-Data Starter Kit is developer sample demonstrating App-Owns-Data embedding
  - Originally published in July 2021 - Updated in October 2022 with **AppOwnsDataReactClient**
  - **AppOwnsDataReactClient** built using React-JS, Material UI, MSAL.js Typescript and webpack



TedPattison Updates 39932f6 6 hours ago 216 commits

AppOwnsDataAdmin	Updates	21 hours ago
AppOwnsDataClient	Updates	yesterday
AppOwnsDataReactClient	Updates	yesterday
AppOwnsDataShared	Updates	yesterday
AppOwnsDataWebApi	Updates	yesterday
Docs	Updates	6 hours ago
AppOwnsDataStarterKit.sln	Updates	yesterday
AppOwsDataUsageReporting.pbix	Updates	6 hours ago

Intro Demo of AppOwnsDataReactClient



State	Sales Revenue	Units Sold	Customer Count
CA	$5,394,441	681,607	10,843
TX	$3,829,475	644,747	7,922
FL	$2,558,182	475,889	5,577
OR	$1,789,160	215,837	3,426
GA	$1,707,677	305,316	3,530
WA	$1,681,541	223,822	3,284
AZ	$1,586,886	236,120	3,321
NC	$1,448,869	261,683	3,109
NY	$1,303,529	258,122	2,891
CO	$1,037,779	166,322	2,155
VA	$993,454	191,731	2,190
PA	$887,492	178,222	1,917
NM	$833,785	123,267	1,604
TN	$728,281	138,460	1,579
MA	$569,535	110,471	1,266
NJ	$509,734	96,096	1,153
CT	$497,090	100,693	1,143
SC	$458,267	82,863	978
LA	$450,445	91,901	963
OH	$416,938	77,600	975
UT	$384,891	56,343	790
AL	$260,971	47,955	559
NH	$256,513	47,601	623
RI	$203,680	36,552	459
Total	**$29,788,617**	**4,849,220**	**62,257**

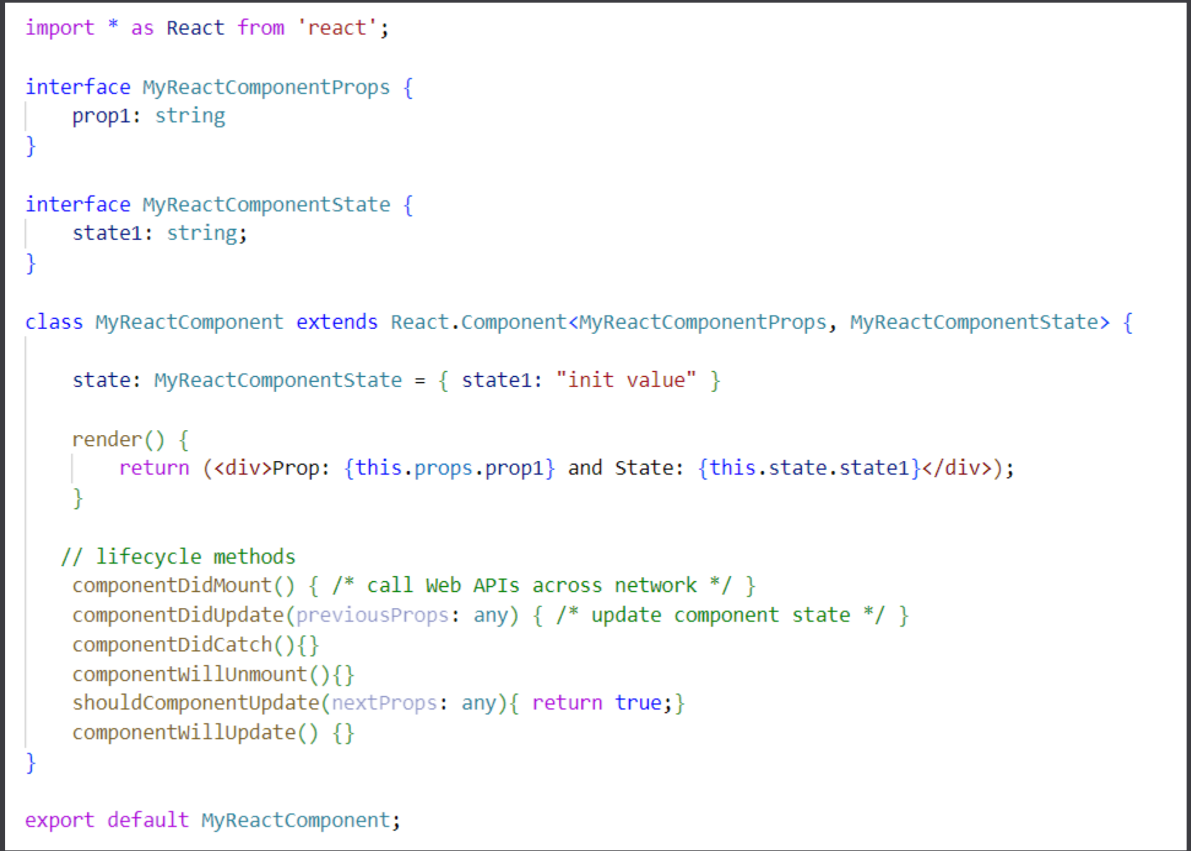
Agenda

- ➢ Quick Review of Modern React
- App-Owns-Data Starter Kit Architecture
- Designing a View Model for a Multitenant Application
- Designing a Functional Component for Embedding Reports
- Managing Embed Token Expiration
- Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

Migrating from Classic React to Modern React

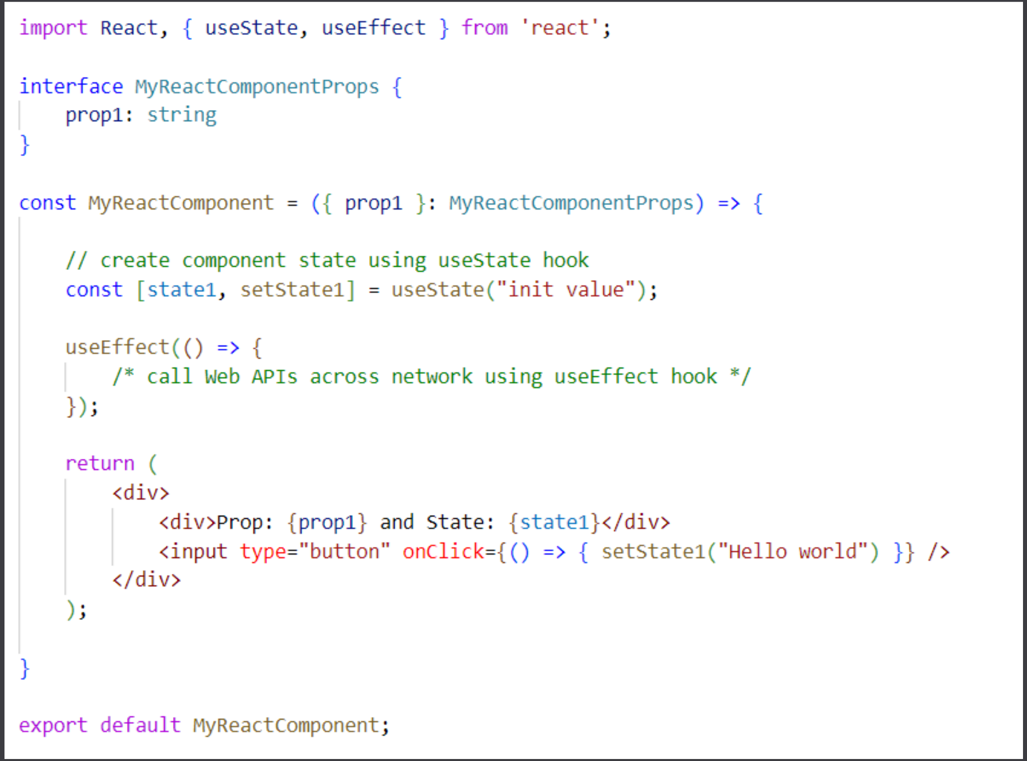
- Classic React based on class-based components and lifecycle method
- Modern React based on functional components and hooks

Class React with Class-based Components



```
import * as React from 'react';

interface MyReactComponentProps {
    prop1: string
}

interface MyReactComponentState {
    state1: string;
}

class MyReactComponent extends React.Component<MyReactComponentProps, MyReactComponentState> {

    state: MyReactComponentState = { state1: "init value" }

    render() {
        return (<div>Prop: {this.props.prop1} and State: {this.state.state1}</div>);
    }

   // lifecycle methods
    componentDidMount() { /* call Web APIs across network */ }
    componentDidUpdate(previousProps: any) { /* update component state */ }
    componentDidCatch(){}
    componentWillUnmount(){}
    shouldComponentUpdate(nextProps: any){ return true;}
    componentWillUpdate() {}
}

export default MyReactComponent;
```

Modern React with Functional Components



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

interface MyReactComponentProps {
    prop1: string
}

const MyReactComponent = ({ prop1 }: MyReactComponentProps) => {

    // create component state using useState hook
    const [state1, setState1] = useState("init value");

    useEffect(() => {
        /* call Web APIs across network using useEffect hook */
    });

    return (
        <div>
            <div>Prop: {prop1} and State: {state1}</div>
            <input type="button" onClick={() => { setState1("Hello world") }} />
        </div>
    );

}

export default MyReactComponent;
```

Hooks with which you should be familiar…

- Hooks provided by React-JS and the React Router
  - **useState**
  - **useEffect**
  - **useLayoutEffect**
  - **useRef**
  - **useContext**
  - **useNavigate**

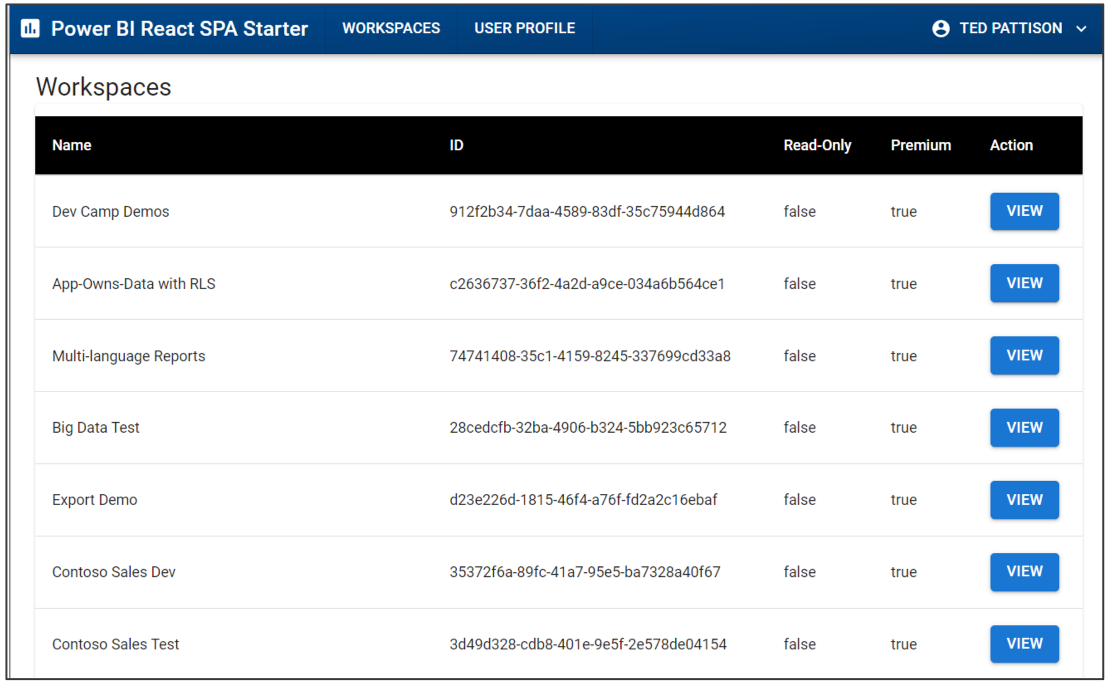
Working with React Components from Material UI

- Import Material UI Components

```
import { Box, Button, Paper, Typography,  } from '@mui/material';
import { Table, TableBody, TableCell, TableContainer, TableHead, TableRow } from '@mui/material';
```

- Build UI using Material UI components

```
<Box sx={{ pt:2 }} >
    <Typography variant='h5' component="h2" >Workspaces</Typography>
    <TableContainer component={Paper}>
        <Table sx={{ marginTop: "12px" }}>
            <TableHead sx={{ "& th": { color: "white", backgroundColor: "black" } }} >
                <TableRow>
                    <TableCell>Name</TableCell>
                    <TableCell>ID</TableCell>
                    <TableCell>Read-Only</TableCell>
                    <TableCell>Premium</TableCell>
                    <TableCell>Action</TableCell>
                </TableRow>
            </TableHead>
            <TableBody>
                {workspaces && workspaces.map((workspace) => (
                    <TableRow key={workspace.id}>
                        <TableCell>{workspace.name}</TableCell>
                        <TableCell>{workspace.id}</TableCell>
                        <TableCell>{String(workspace.isReadOnly)}</TableCell>
                        <TableCell>{String(workspace.isOnDedicatedCapacity)}</TableCell>
                        <TableCell>
                            <Button variant='contained' target="_blank"
                                    href={'https://app.powerbi.com/groups/' + workspace.id}>
                                View
                            </Button>
                        </TableCell>
                    </TableRow>
                ))}
            </TableBody>
        </Table>
    </TableContainer>
</Box>);
```

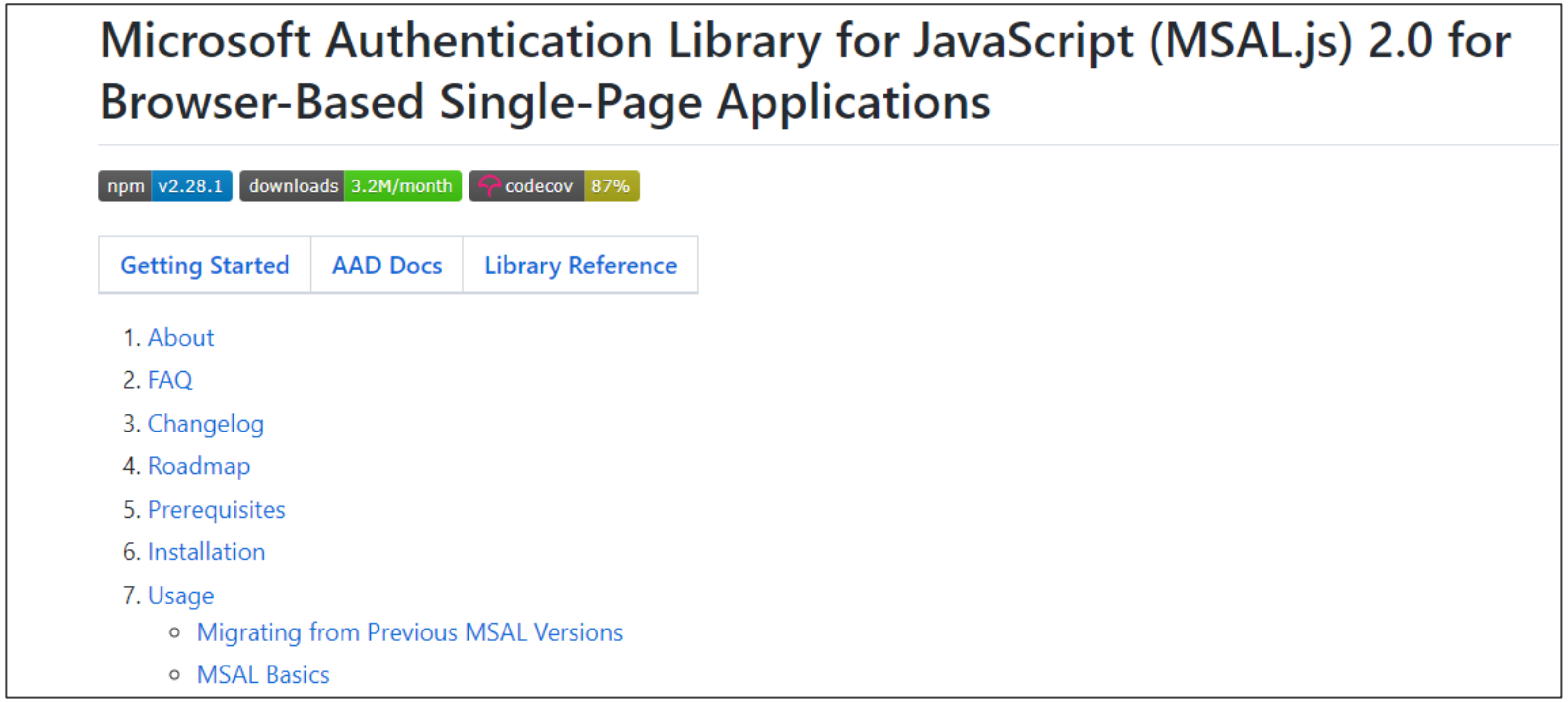


Workspaces

Name	ID	Read-Only	Premium	Action
Dev Camp Demos	912f2b34-7daa-4589-83df-35c75944d864	false	true	VIEW
App-Owns-Data with RLS	c2636737-36f2-4a2d-a9ce-034a6b564ce1	false	true	VIEW
Multi-language Reports	74741408-35c1-4159-8245-337699cd33a8	false	true	VIEW
Big Data Test	28cedcfb-32ba-4906-b324-5bb923c65712	false	true	VIEW
Export Demo	d23e226d-1815-46f4-a76f-fd2a2c16ebaf	false	true	VIEW
Contoso Sales Dev	35372f6a-89fc-41a7-95e5-ba7328a40f67	false	true	VIEW
Contoso Sales Test	3d49d328-cdb8-401e-9e5f-2e578de04154	false	true	VIEW

MSAL.js V 2.0

- Microsoft Authentication Library for JavaScript
  - Authenticates SPA users with Azure AD
  - Install using node.js - **npm i @azure/msal-browser**
  - Docs: https://github.com/AzureAD/microsoft-authentication-library-for-js

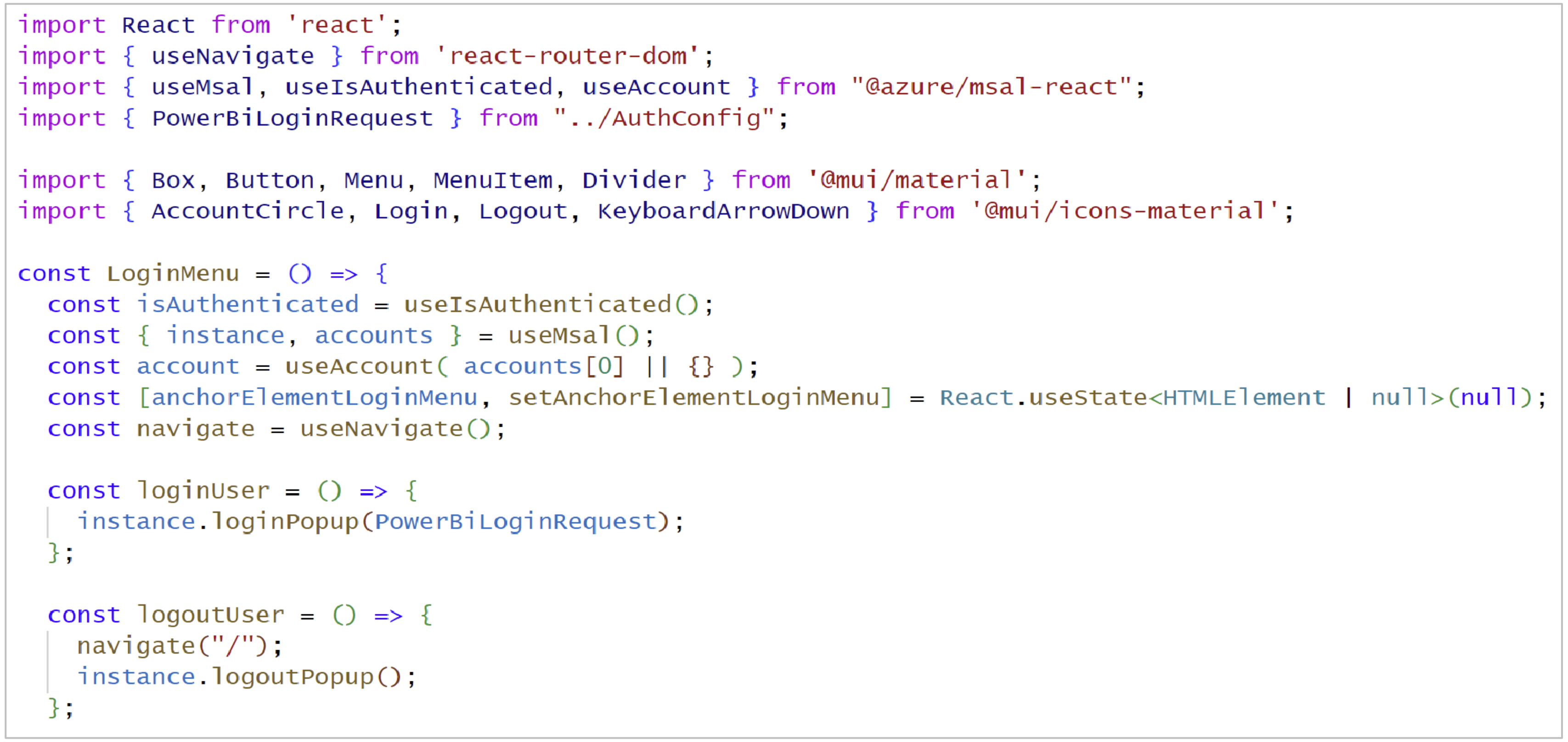


Microsoft Authentication Library for JavaScript (MSAL.js) 2.0 for Browser-Based Single-Page Applications

Getting Started	AAD Docs	Library Reference

MSAL Component for React (@azure/msal-react)

- **msal-react** library is consumed by using hooks
  - useMsal
  - useIsAuthenticated
  - useAccount

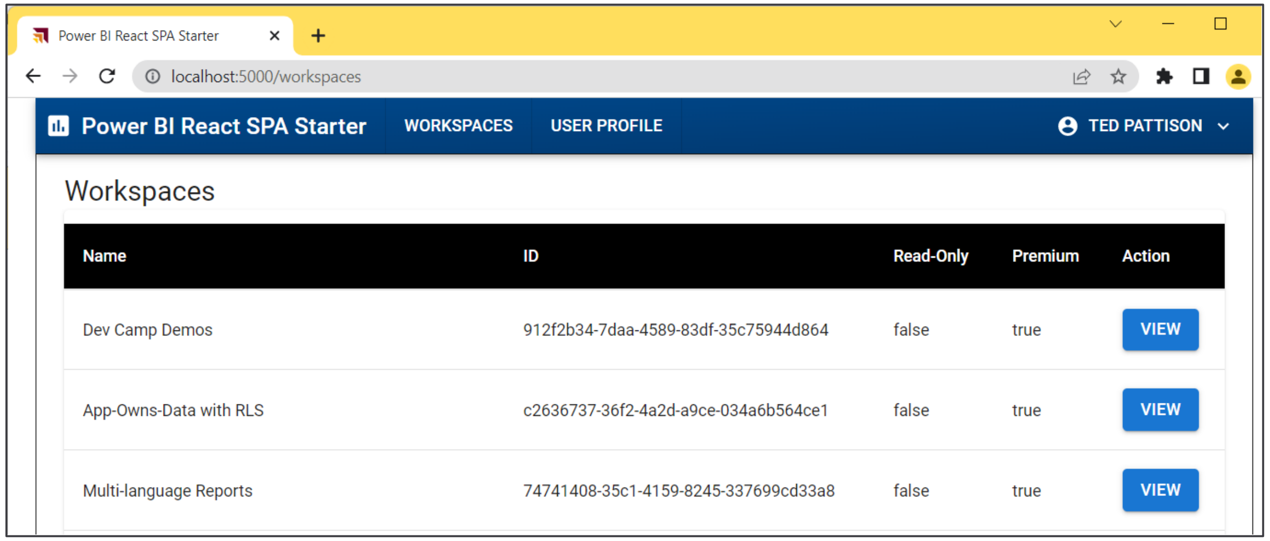


```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { useMsal, useIsAuthenticated, useAccount } from "@azure/msal-react";
import { PowerBiLoginRequest } from "../AuthConfig";

import { Box, Button, Menu, MenuItem, Divider } from '@mui/material';
import { AccountCircle, Login, Logout, KeyboardArrowDown } from '@mui/icons-material';

const LoginMenu = () => {
  const isAuthenticated = useIsAuthenticated();
  const { instance, accounts } = useMsal();
  const account = useAccount( accounts[0] || {} );
  const [anchorElementLoginMenu, setAnchorElementLoginMenu] = React.useState<HTMLElement | null>(null);
  const navigate = useNavigate();

  const loginUser = () => {
    instance.loginPopup(PowerBiLoginRequest);
  };

  const logoutUser = () => {
    navigate("/");
    instance.logoutPopup();
  };
```

Watch the August 2022 Session of Power BI Dev Camp

- **Session 25 - Modern React-JS Development with Power BI Embedding**
  - Provides more detail on technical content covered in this review section
  - Accompanied by developer code samples demonstrating modern React-JS development
  - https://www.powerbidevcamp.net/sessions/session25/

Modern React-JS Development with Power BI Embedding
Ted Pattison
Principal Program Manager
Power BI Customer Advisory Team (PBICAT)
0:00 / 1:07:31


Power BI React SPA Starter
localhost:5000/workspaces
Power BI React SPA Starter
WORKSPACES
USER PROFILE
TED PATTISON

Workspaces

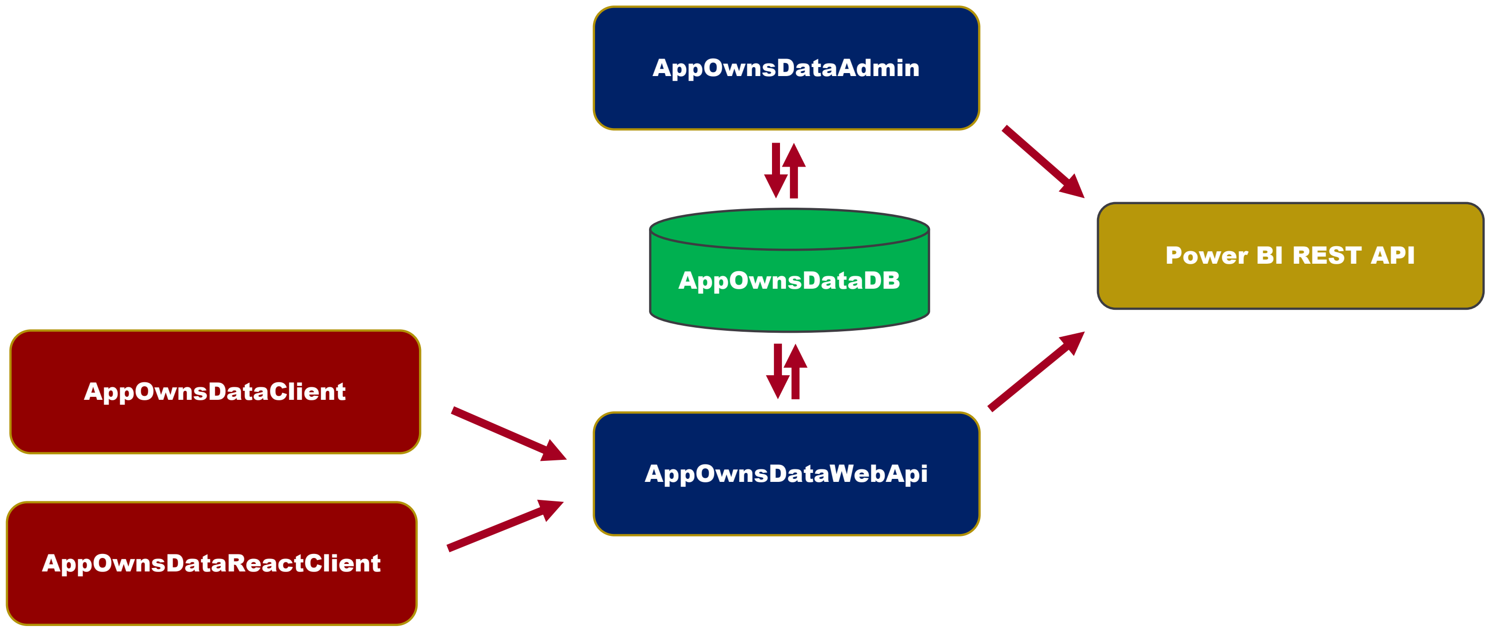
Name	ID	Read-Only	Premium	Action
Dev Camp Demos	912f2b34-7daa-4589-83df-35c75944d864	false	true	VIEW
App-Owns-Data with RLS	c2636737-36f2-4a2d-a9ce-034a6b564ce1	false	true	VIEW
Multi-language Reports	74741408-35c1-4159-8245-337699cd33a8	false	true	VIEW

Agenda

- ✓ Quick Review of Modern React
- ➢ App-Owns-Data Starter Kit Architecture
- Designing a View Model for a Multitenant Application
- Designing a Functional Component for Embedding Reports
- Managing Embed Token Expiration
- Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

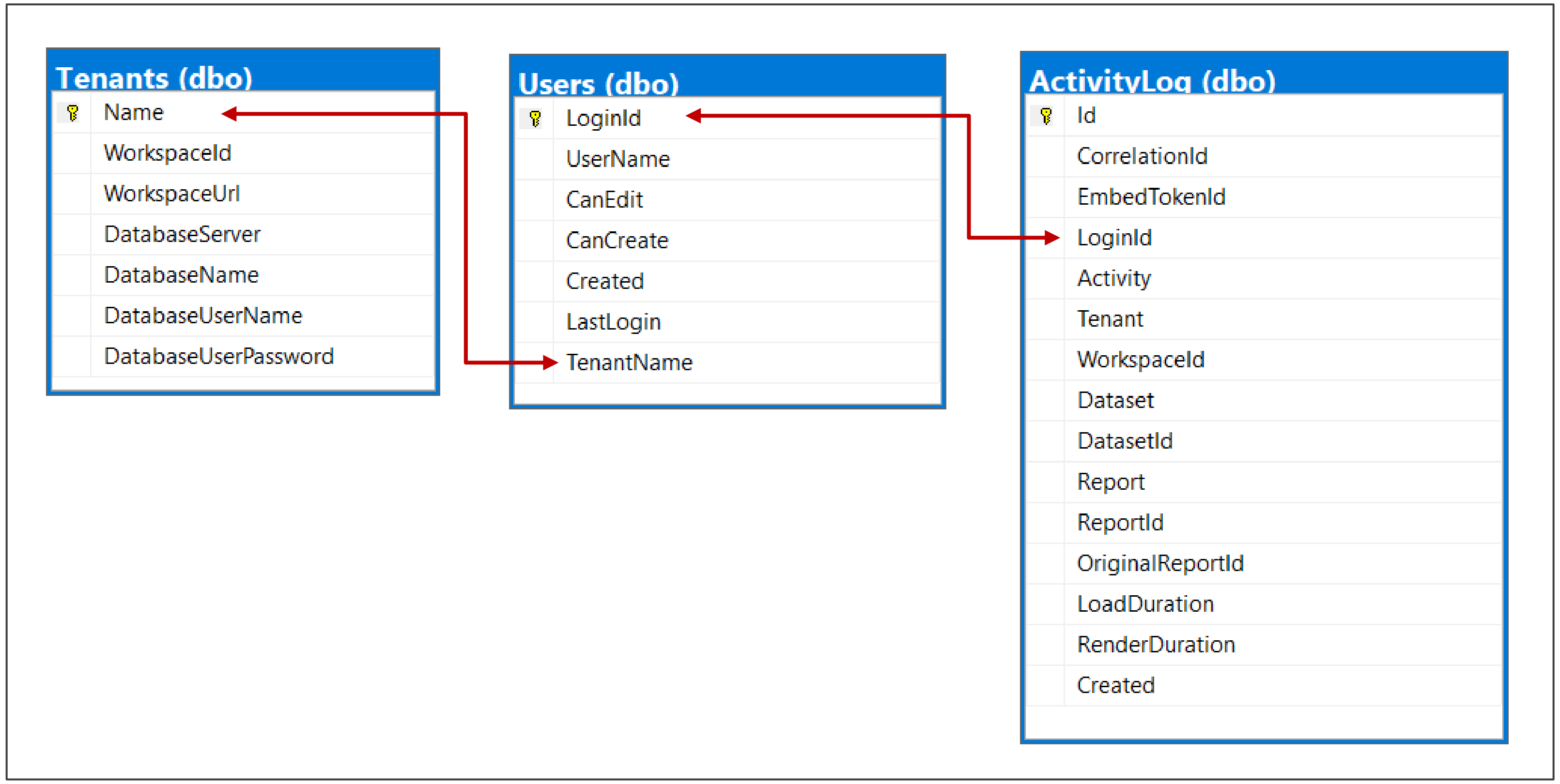
App-Owns-Data Starter Kit Architecture

- **AppOwnsDataDB**: custom database to track tenants, user permissions and user activity
- **AppOwnsDataAdmin**: administrative app to create tenants and manage user permissions
- **AppOwnsDataWebApi**: custom Web API used by client-side SPA applications
- **AppOwnsDataClient**: customer-facing SPA used to view and author reports (JavaScript & JQuery)
- **AppOwnsDataReactClient** : customer-facing SPA used to view and author reports (modern React-JS)



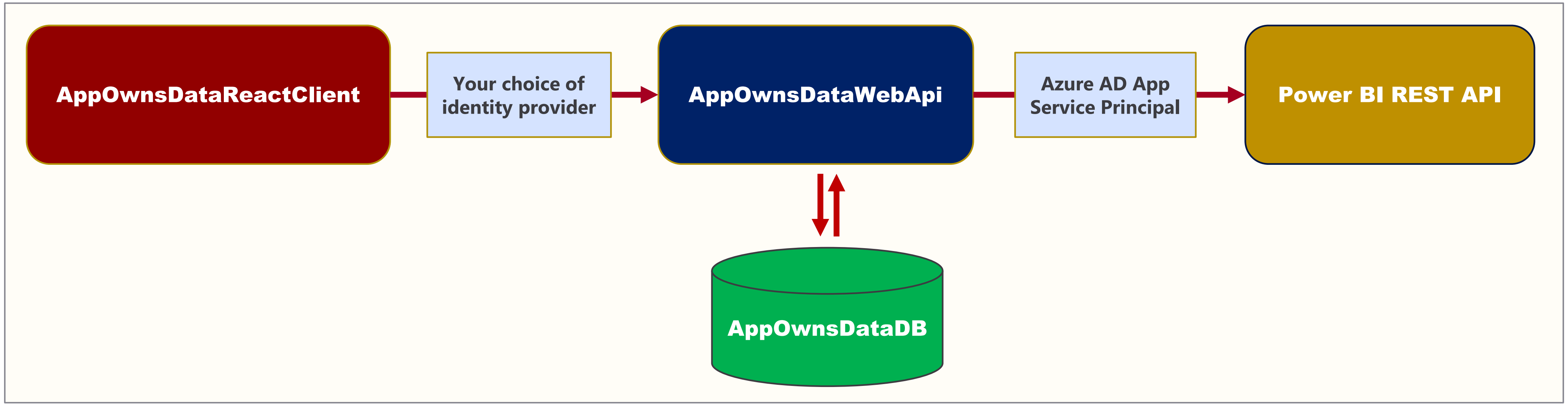
Database Schema for AppOwnsDataDB

- **Tenants** table tracks Power BI workspace Id and customer database connection info
- **Users** table tracks user profile with tenant assignment and permissions within tenant
- **ActivityLog** table tracks telemetry data about user activity and report performance



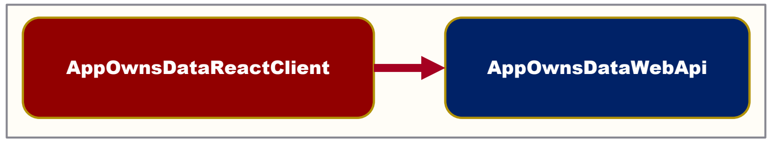
AppOwnsDataWebApi

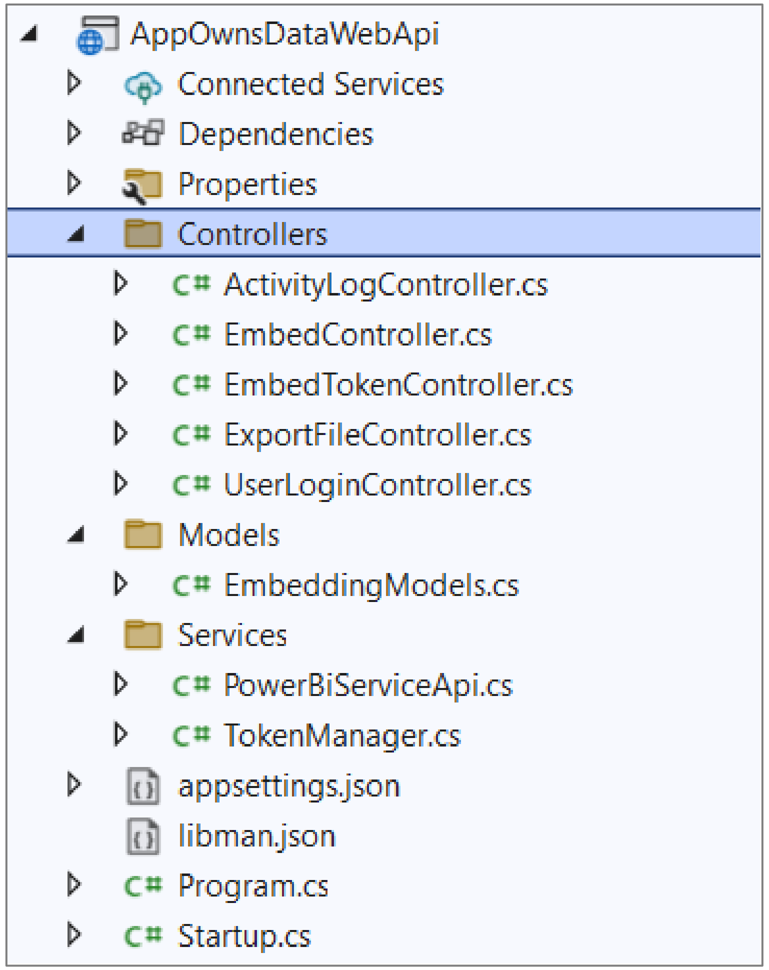
- Provides secure Web API endpoints for **AppOwnsDataReactClient**
  - Used to process user login
  - Used to retrieve embedding data and embed tokens
  - Used to log activity events
  - Used to enable on-demand exporting of reports



AppOwnsDataWebApi Controllers

- **UserLogin**
  - Called by client SPA once after user authenticates
- **Embed**
  - Called to retrieve embedding data for a specific tenant
- **EmbedToken**
  - Called to retrieve new embed token
- **ActivityLog**
  - Called to post activity event to custom audit log
- **ExportToFile**
  - Called in response to user action to export report to file

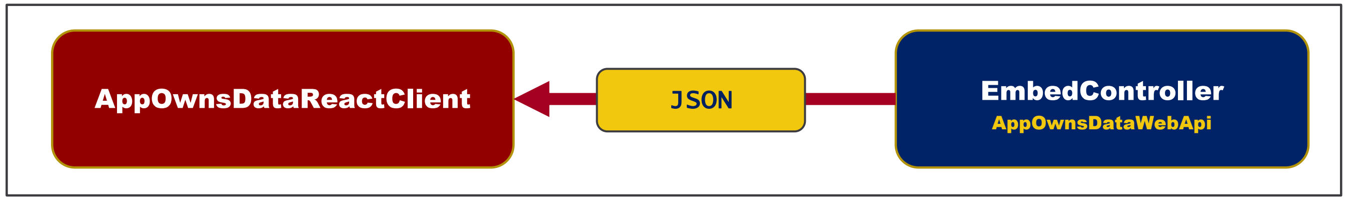


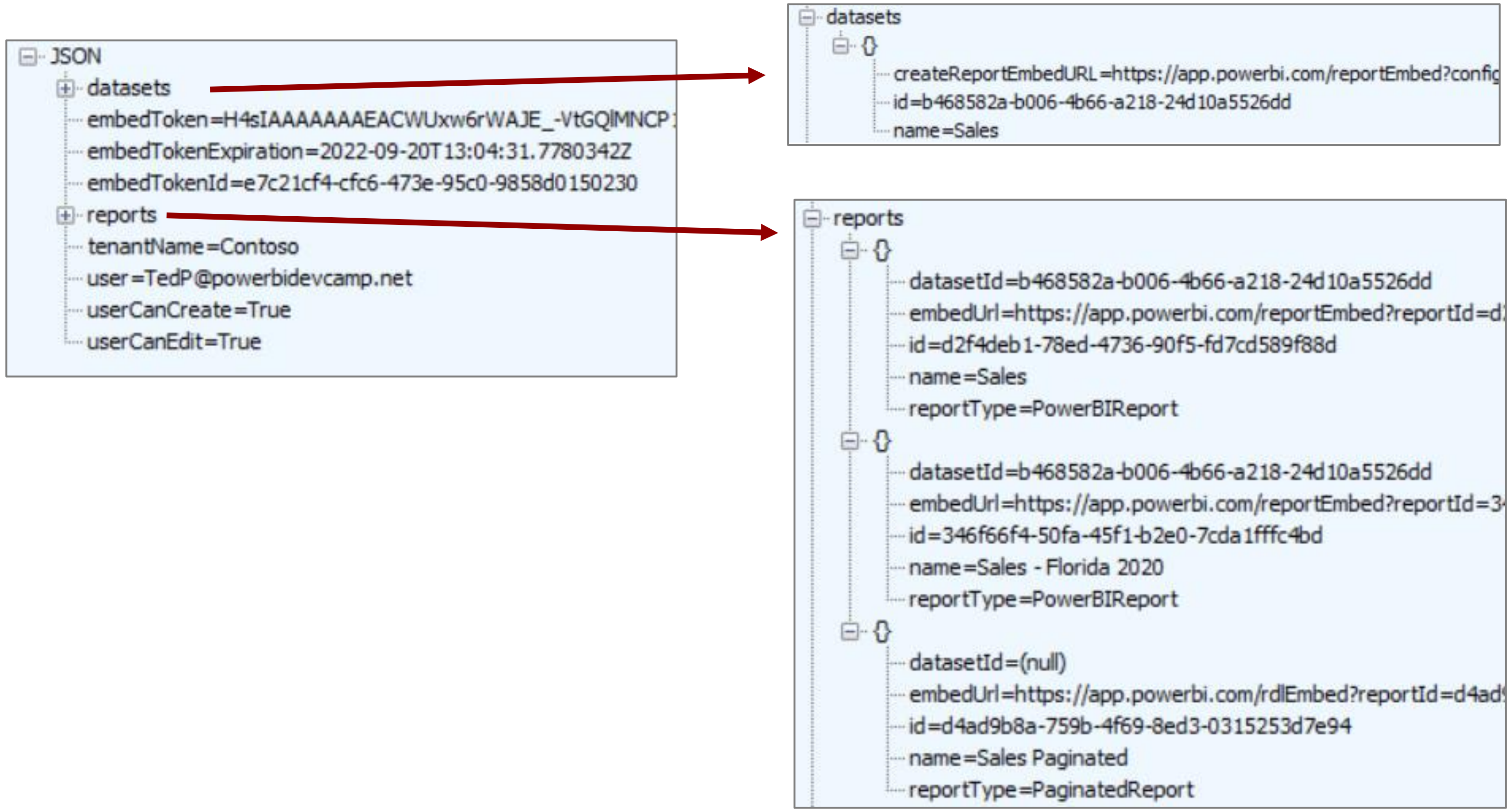


```
◢ AppOwnsDataWebApi
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ◢ Controllers
        ▷ C# ActivityLogController.cs
        ▷ C# EmbedController.cs
        ▷ C# EmbedTokenController.cs
        ▷ C# ExportFileController.cs
        ▷ C# UserLoginController.cs
    ◢ Models
        ▷ C# EmbeddingModels.cs
    ◢ Services
        ▷ C# PowerBiServiceApi.cs
        ▷ C# TokenManager.cs
    ▷ appsettings.json
      libman.json
    ▷ C# Program.cs
    ▷ C# Startup.cs
```

Embedding Data View Model

- **AppOwnsDataWebApi** returns view model to **AppOwnsDataReactClient**





```
JSON
  datasets
  embedToken=H4sIAAAAAAAEACWUxw6rWAJE_-VtGQlMNCP
  embedTokenExpiration=2022-09-20T13:04:31.7780342Z
  embedTokenId=e7c21cf4-cfc6-473e-95c0-9858d0150230
  reports
  tenantName=Contoso
  user=TedP@powerbidevcamp.net
  userCanCreate=True
  userCanEdit=True
```

```
datasets
  {}
    createReportEmbedURL=https://app.powerbi.com/reportEmbed?config
    id=b468582a-b006-4b66-a218-24d10a5526dd
    name=Sales
```

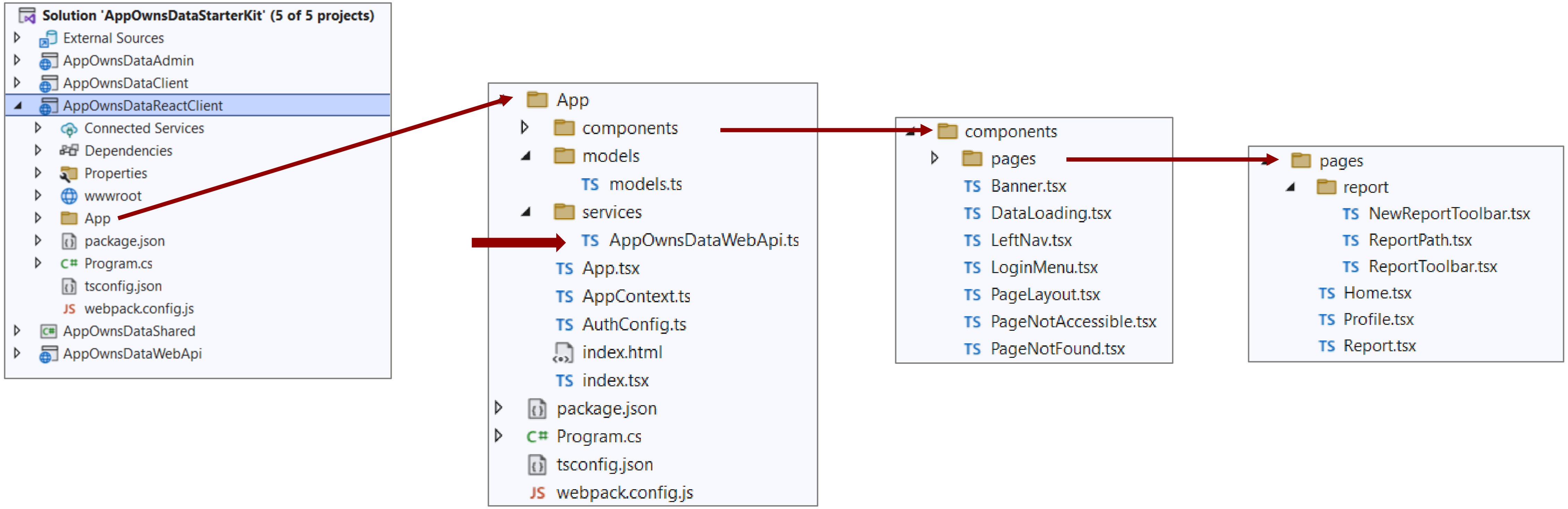
```
reports
  {}
    datasetId=b468582a-b006-4b66-a218-24d10a5526dd
    embedUrl=https://app.powerbi.com/reportEmbed?reportId=d
    id=d2f4deb1-78ed-4736-90f5-fd7cd589f88d
    name=Sales
    reportType=PowerBIReport
  {}
    datasetId=b468582a-b006-4b66-a218-24d10a5526dd
    embedUrl=https://app.powerbi.com/reportEmbed?reportId=3
    id=346f66f4-50fa-45f1-b2e0-7cda1fffc4bd
    name=Sales - Florida 2020
    reportType=PowerBIReport
  {}
    datasetId=(null)
    embedUrl=https://app.powerbi.com/rdlEmbed?reportId=d4ad
    id=d4ad9b8a-759b-4f69-8ed3-0315253d7e94
    name=Sales Paginated
    reportType=PaginatedReport
```

Agenda

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ➢ Designing a View Model for a Multitenant Application
- Designing a Functional Component for Embedding Reports
- Managing Embed Token Expiration
- Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

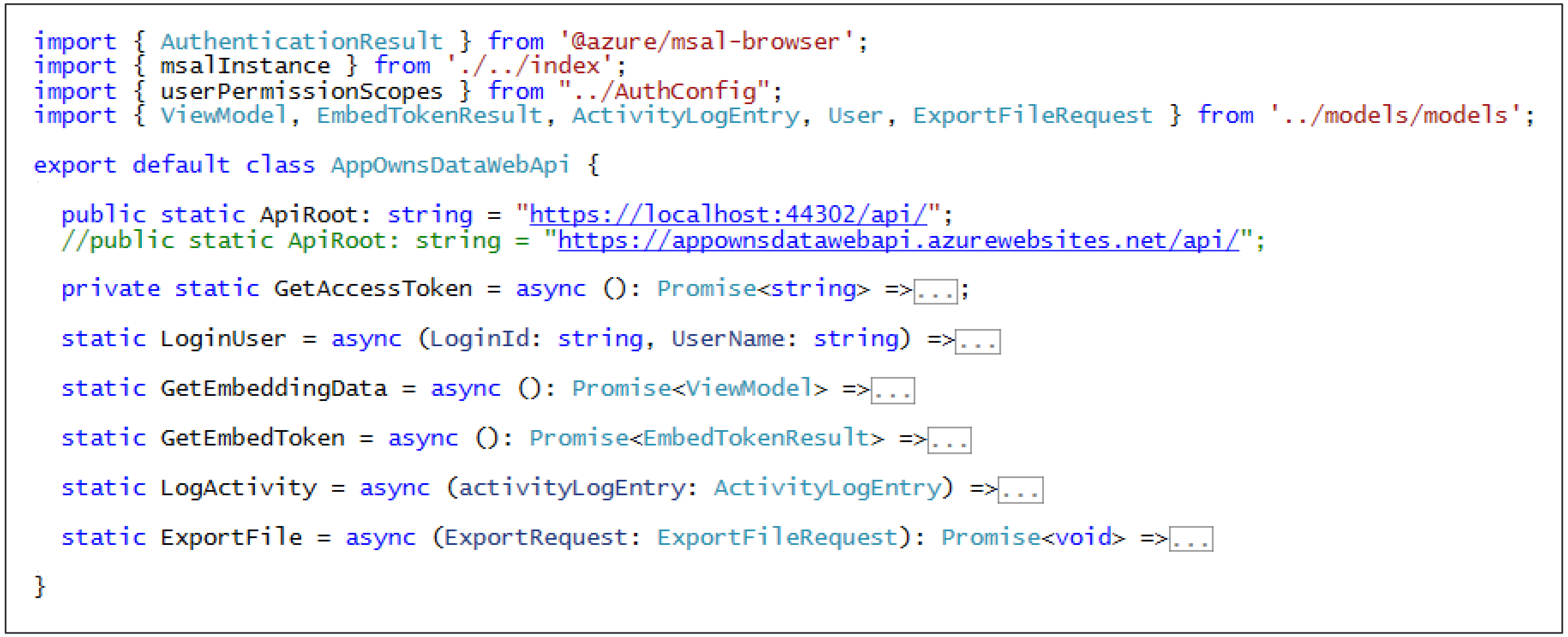
AppOwnsDataReactClent Application Structure

- **AppOwnsDataReactClent** used to develop client-side SPA application with React-JS
  - Webpack utility used to compile Typescript code to JavaScript
  - All Typescript code for SPA located inside **App** folder
  - Code to call Web API encapsulated in **AppOwnsDatWebApi.ts**
  - React components located inside the **components** folder
  - Pages used by React Router located inside the **pages** folder


```
Solution 'AppOwnsDataStarterKit' (5 of 5 projects)
  External Sources
  AppOwnsDataAdmin
  AppOwnsDataClient
  AppOwnsDataReactClient
    Connected Services
    Dependencies
    Properties
    wwwroot
    App
    package.json
    Program.cs
    tsconfig.json
    webpack.config.js
  AppOwnsDataShared
  AppOwnsDataWebApi

App
  components
  models
    models.ts
  services
    AppOwnsDataWebApi.ts
  App.tsx
  AppContext.ts
  AuthConfig.ts
  index.html
  index.tsx
package.json
Program.cs
tsconfig.json
webpack.config.js

components
  pages
  Banner.tsx
  DataLoading.tsx
  LeftNav.tsx
  LoginMenu.tsx
  PageLayout.tsx
  PageNotAccessible.tsx
  PageNotFound.tsx

pages
  report
    NewReportToolbar.tsx
    ReportPath.tsx
    ReportToolbar.tsx
  Home.tsx
  Profile.tsx
  Report.tsx
```

AppOwnsDataWebApi Wrapper Class

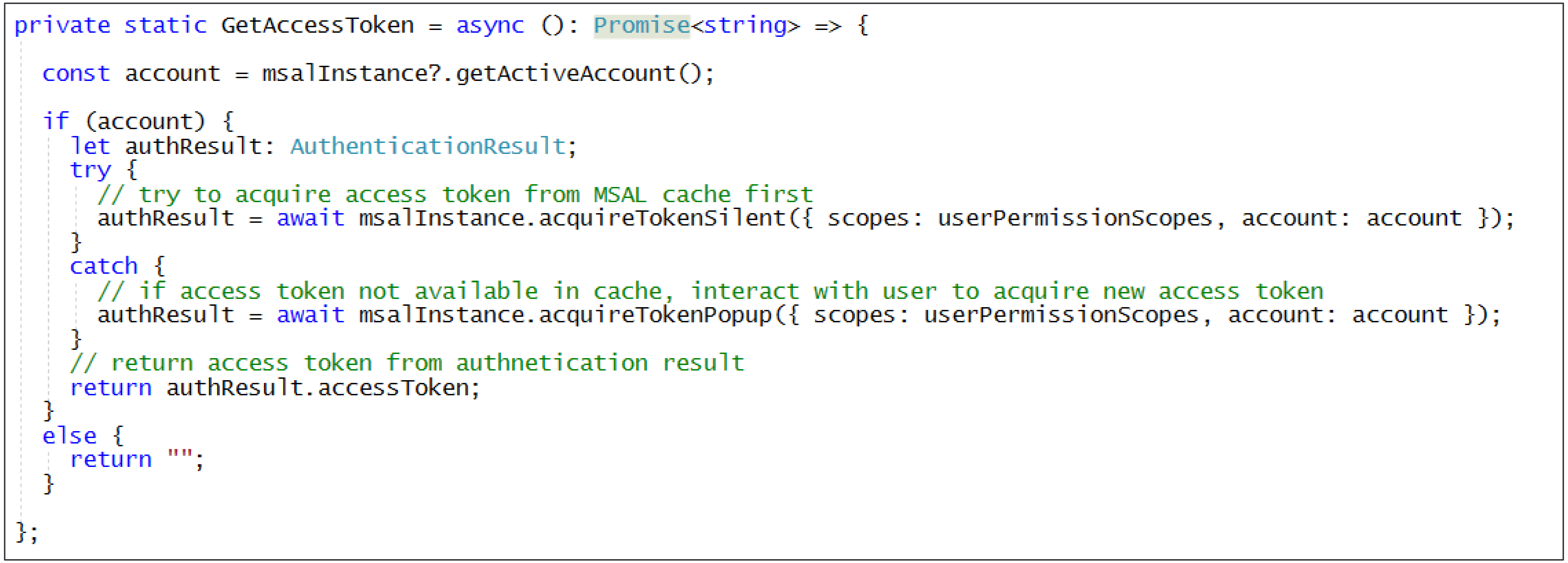
- AppOwnsDataWebApi is wrapper class to call external API



```
import { AuthenticationResult } from '@azure/msal-browser';
import { msalInstance } from './../index';
import { userPermissionScopes } from "../AuthConfig";
import { ViewModel, EmbedTokenResult, ActivityLogEntry, User, ExportFileRequest } from '../models/models';

export default class AppOwnsDataWebApi {

  public static ApiRoot: string = "https://localhost:44302/api/";
  //public static ApiRoot: string = "https://appownsdatawebapi.azurewebsites.net/api/";

  private static GetAccessToken = async (): Promise<string> =>...;

  static LoginUser = async (LoginId: string, UserName: string) =>...

  static GetEmbeddingData = async (): Promise<ViewModel> =>...

  static GetEmbedToken = async (): Promise<EmbedTokenResult> =>...

  static LogActivity = async (activityLogEntry: ActivityLogEntry) =>...

  static ExportFile = async (ExportRequest: ExportFileRequest): Promise<void> =>...

}
```

AppOwnsDataWebApi.GetAccessToken

- **GetAccessToken** uses **msalInstance** object which is initialized in **index.tsx**
  - First attempt is made using **acquireTokenSilent** to acquire token from browser cache
  - Second attempt is made using **acquireTokenPopup** which triggers user interaction



```
private static GetAccessToken = async (): Promise<string> => {

  const account = msalInstance?.getActiveAccount();

  if (account) {
    let authResult: AuthenticationResult;
    try {
      // try to acquire access token from MSAL cache first
      authResult = await msalInstance.acquireTokenSilent({ scopes: userPermissionScopes, account: account });
    }
    catch {
      // if access token not available in cache, interact with user to acquire new access token
      authResult = await msalInstance.acquireTokenPopup({ scopes: userPermissionScopes, account: account });
    }
    // return access token from authnetication result
    return authResult.accessToken;
  }
  else {
    return "";
  }

};
```

AppOwnsDataWebApi.GetEmbeddingData

- **GetEmbeddingData** uses serialization types defined in **models.ts**

```
export class ViewModel {
  tenantName: string;
  reports: PowerBiReport[];
  datasets: PowerBiDataset[];
  embedToken: string;
  embedTokenExpiration: string;
  user: string;
  userCanEdit: boolean;
  userCanCreate: boolean;
}
```

```
export class PowerBiReport {
  id: string;
  name: string;
  datasetId: string;
  embedUrl: string;
  reportType: string;
}
```

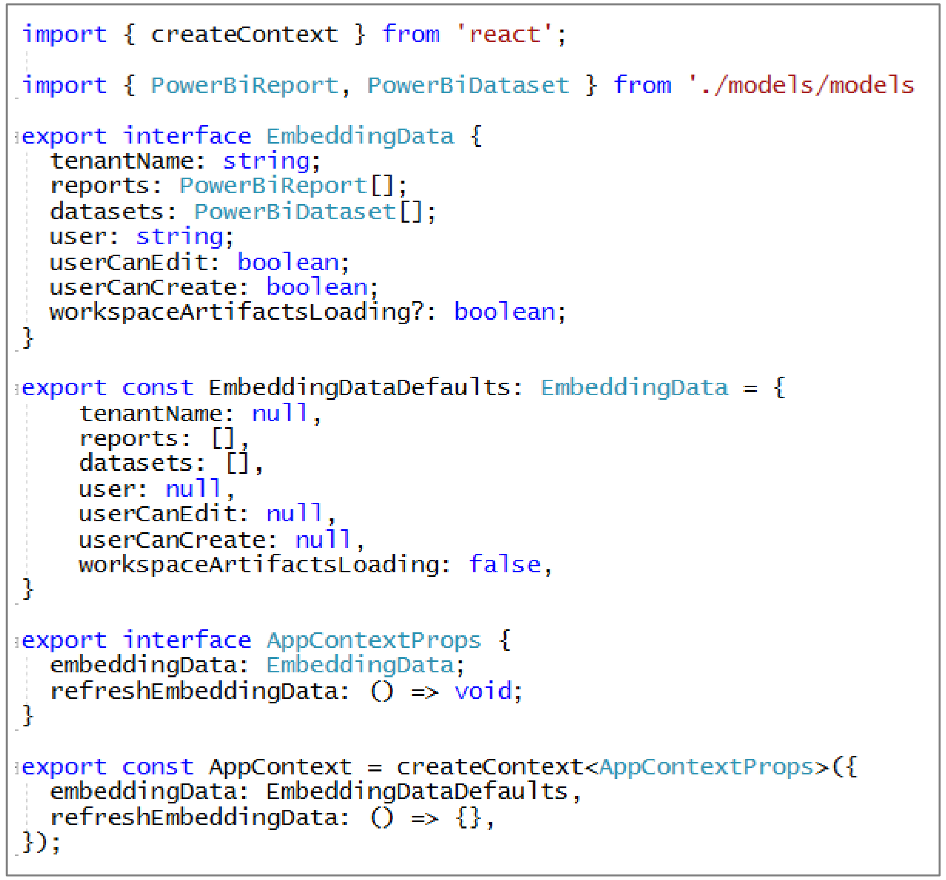
```
export class PowerBiDataset {
  id: string;
  name: string;
  createReportEmbedURL: string;
}
```

- **GetEmbeddingData** uses **fetch** API to call across network

```
static GetEmbeddingData = async (): Promise<ViewModel> => {

  var accessToken: string = await AppOwnsDataWebApi.GetAccessToken();
  var restUrl = AppOwnsDataWebApi.ApiRoot + "Embed/";

  return fetch(restUrl, {
    method: "GET",
    headers: {
      "Accept": "application/json;",
      "Authorization": "Bearer " + accessToken
    }
  }).then(response => response.json())
    .then(response => response);
}
```

AppContext.ts

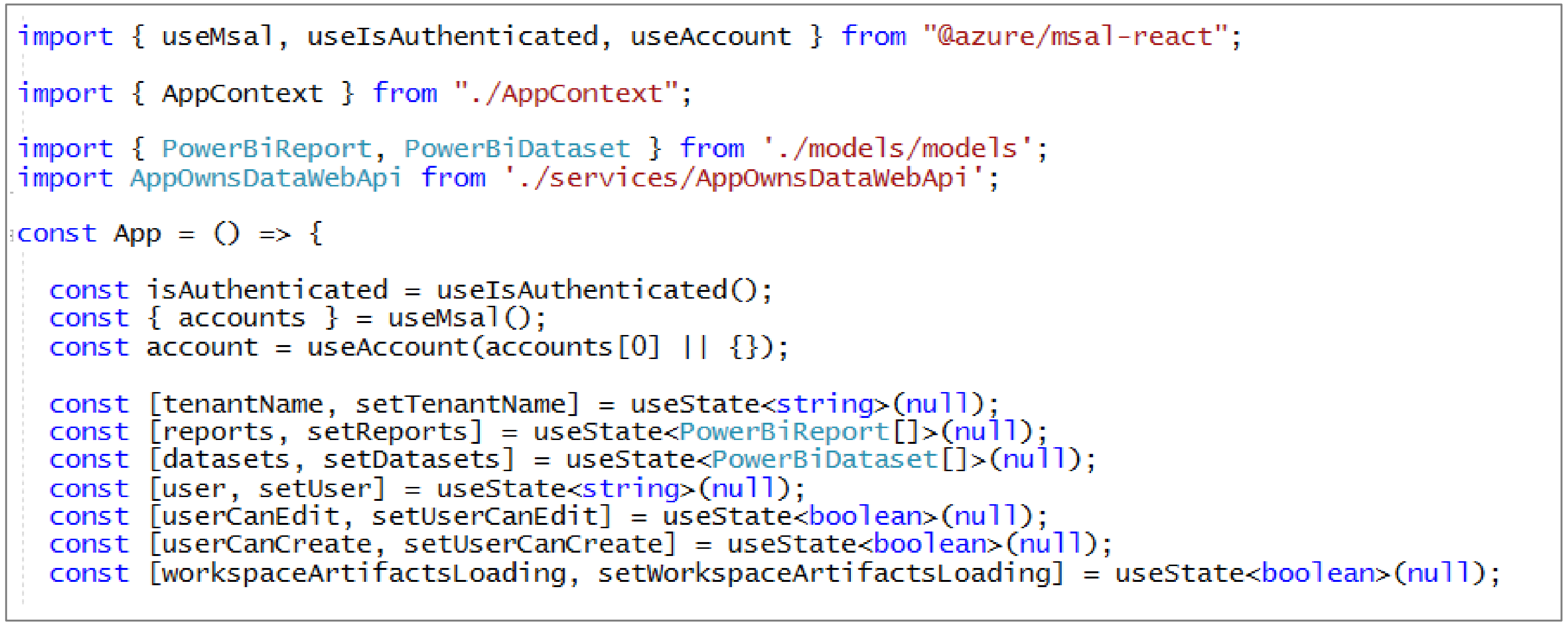
- React-JS provides context feature for sharing state across components
  - Designing with context lessens the need to pass parameters between components
  - Context includes state properties which can be shared with child components
  - Context can also contain modifier methods allowing child components to modify context state



```
import { createContext } from 'react';

import { PowerBiReport, PowerBiDataset } from './models/models

export interface EmbeddingData {
  tenantName: string;
  reports: PowerBiReport[];
  datasets: PowerBiDataset[];
  user: string;
  userCanEdit: boolean;
  userCanCreate: boolean;
  workspaceArtifactsLoading?: boolean;
}

export const EmbeddingDataDefaults: EmbeddingData = {
    tenantName: null,
    reports: [],
    datasets: [],
    user: null,
    userCanEdit: null,
    userCanCreate: null,
    workspaceArtifactsLoading: false,
}

export interface AppContextProps {
  embeddingData: EmbeddingData;
  refreshEmbeddingData: () => void;
}

export const AppContext = createContext<AppContextProps>({
  embeddingData: EmbeddingDataDefaults,
  refreshEmbeddingData: () => {},
});
```

Storing State for Application Context in App.tsx

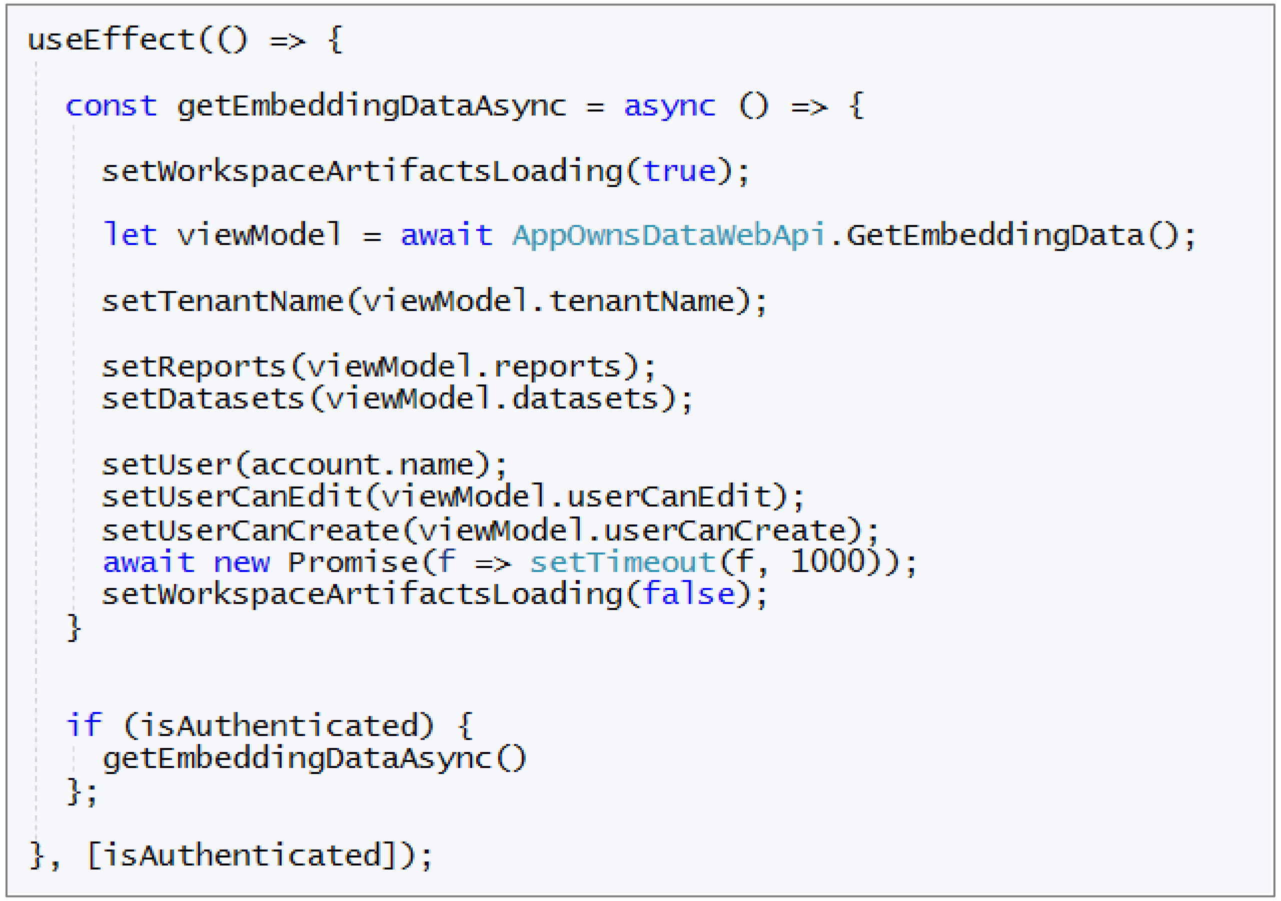
- The state for application context is managed by **App.tsx**
  - **App.tsx** utilizes **useState** hook to track state for application context



```
import { useMsal, useIsAuthenticated, useAccount } from "@azure/msal-react";

import { AppContext } from "./AppContext";

import { PowerBiReport, PowerBiDataset } from './models/models';
import AppOwnsDataWebApi from './services/AppOwnsDataWebApi';

const App = () => {

  const isAuthenticated = useIsAuthenticated();
  const { accounts } = useMsal();
  const account = useAccount(accounts[0] || {});

  const [tenantName, setTenantName] = useState<string>(null);
  const [reports, setReports] = useState<PowerBiReport[]>(null);
  const [datasets, setDatasets] = useState<PowerBiDataset[]>(null);
  const [user, setUser] = useState<string>(null);
  const [userCanEdit, setUserCanEdit] = useState<boolean>(null);
  const [userCanCreate, setUserCanCreate] = useState<boolean>(null);
  const [workspaceArtifactsLoading, setWorkspaceArtifactsLoading] = useState<boolean>(null);
```

Initializing Application Context with useEffect Hook

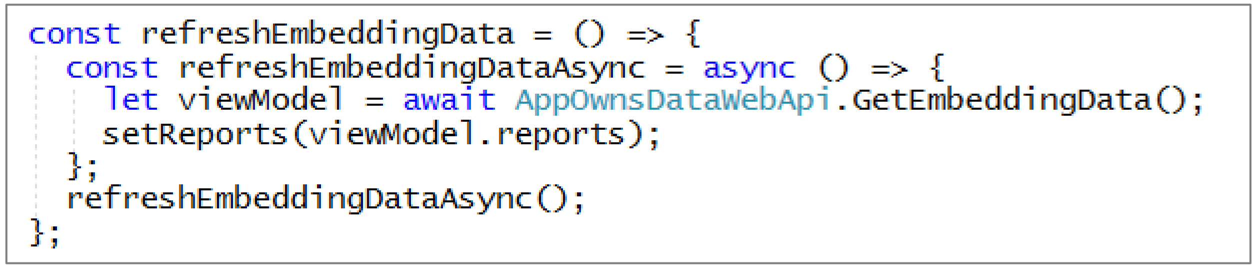
- **useEffect** is hook used to execute code after React has committed changes to DOM
  - Using **useEffect** is best practice when executing code which calls Web APIs across the network
  - Results from Web API call in **useEffect** function can be used to modify component state



```
useEffect(() => {

  const getEmbeddingDataAsync = async () => {

    setWorkspaceArtifactsLoading(true);

    let viewModel = await AppOwnsDataWebApi.GetEmbeddingData();

    setTenantName(viewModel.tenantName);

    setReports(viewModel.reports);
    setDatasets(viewModel.datasets);

    setUser(account.name);
    setUserCanEdit(viewModel.userCanEdit);
    setUserCanCreate(viewModel.userCanCreate);
    await new Promise(f => setTimeout(f, 1000));
    setWorkspaceArtifactsLoading(false);
  }


  if (isAuthenticated) {
    getEmbeddingDataAsync()
  };

}, [isAuthenticated]);
```

refreshEmbeddingData in App.tsx

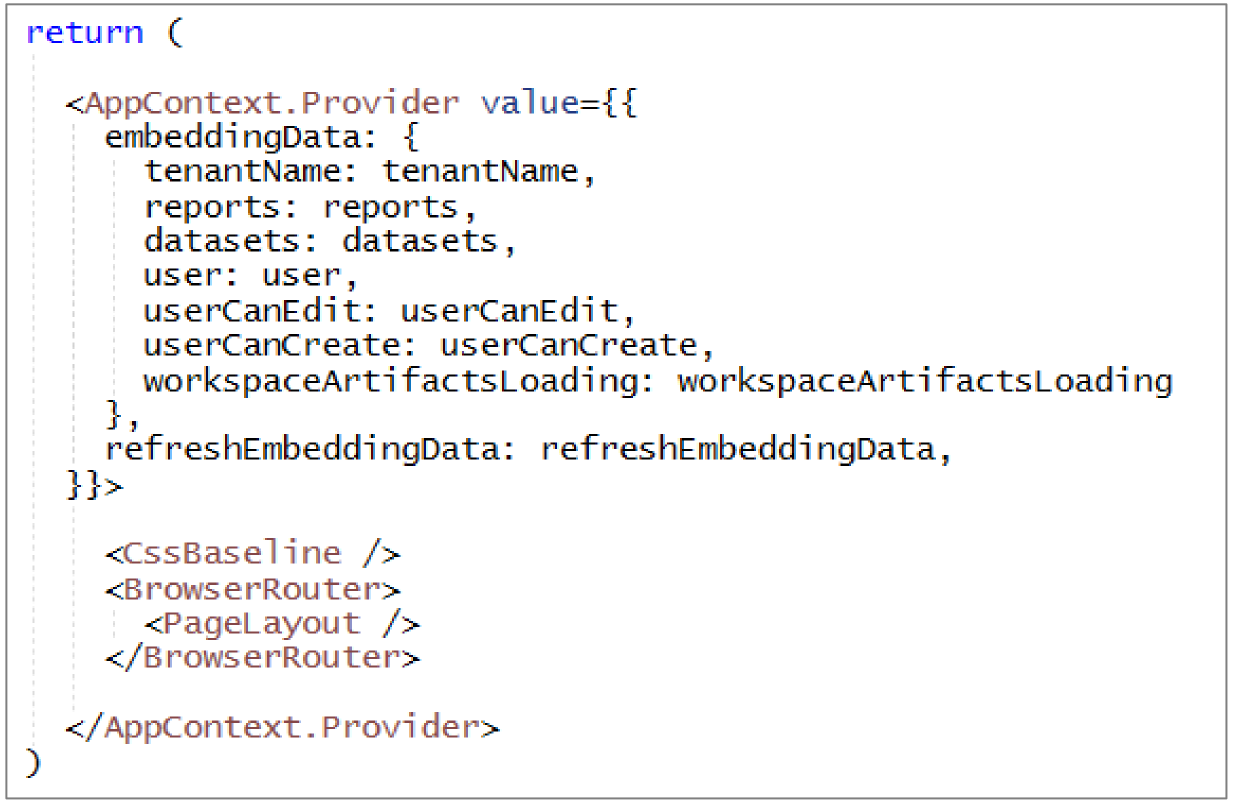
- **refreshEmbeddingData** method used to refresh embedding data
  - This method must be called after the user creates a new report
  - Reference to **refreshEmbeddingData** passed to child components in context



```
const refreshEmbeddingData = () => {
  const refreshEmbeddingDataAsync = async () => {
    let viewModel = await AppOwnsDataWebApi.GetEmbeddingData();
    setReports(viewModel.reports);
  };
  refreshEmbeddingDataAsync();
};
```

Making Context Available to Child Components

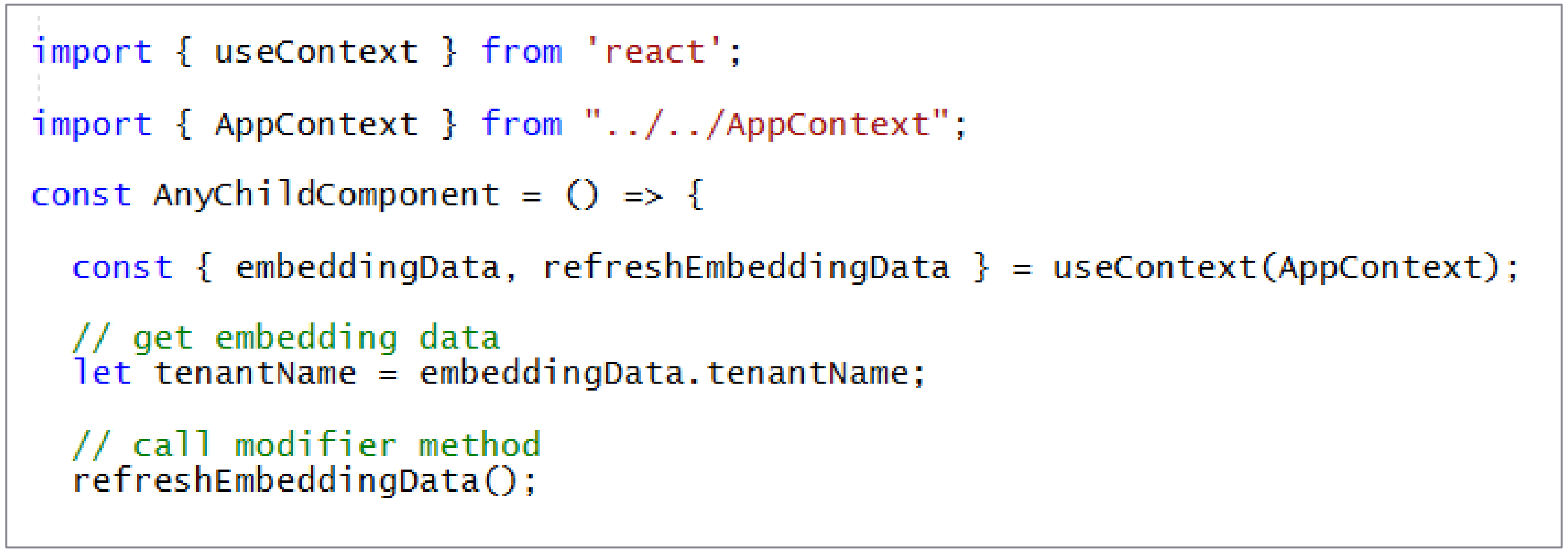
- **App.tsx** contains context **AppContext.Provider** tag
  - **AppContext.Provider** tag used to associate context data with component state in **App.tsx**
  - **AppContext.Provider** tag passes reference to **refreshEmbeddingData**



```
return (

  <AppContext.Provider value={{
    embeddingData: {
      tenantName: tenantName,
      reports: reports,
      datasets: datasets,
      user: user,
      userCanEdit: userCanEdit,
      userCanCreate: userCanCreate,
      workspaceArtifactsLoading: workspaceArtifactsLoading
    },
    refreshEmbeddingData: refreshEmbeddingData,
  }}>

    <CssBaseline />
    <BrowserRouter>
      <PageLayout />
    </BrowserRouter>

  </AppContext.Provider>
)
```

Using AppContext in a Child Component

- Import **AppContext** type into child component
- Create context in child component using **useContext** hook
- Use context state properties and modifier methods anywhere in child component



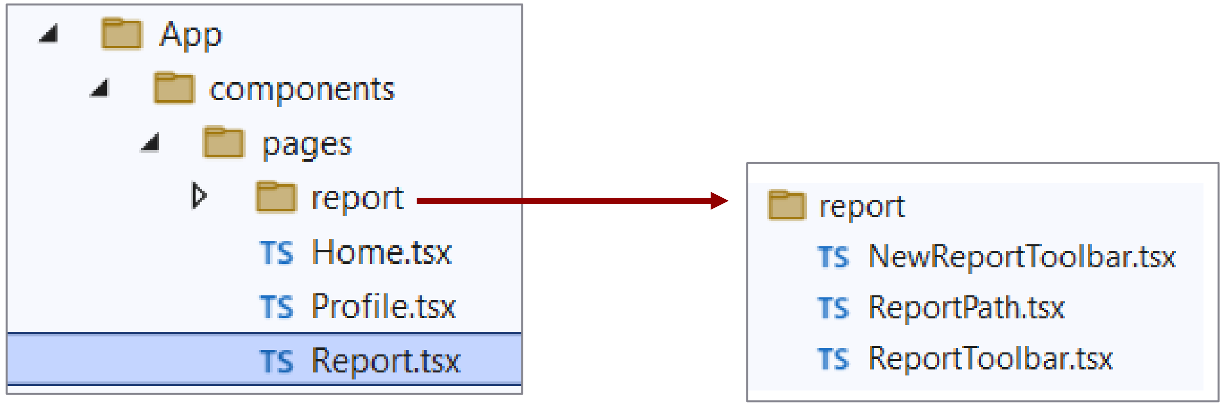
```
import { useContext } from 'react';

import { AppContext } from "../../AppContext";

const AnyChildComponent = () => {

  const { embeddingData, refreshEmbeddingData } = useContext(AppContext);

  // get embedding data
  let tenantName = embeddingData.tenantName;

  // call modifier method
  refreshEmbeddingData();
```

Agenda

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ✓ Designing a View Model for a Multitenant Application
- ➢ Designing a Functional Component for Embedding Reports
- Managing Embed Token Expiration
- Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

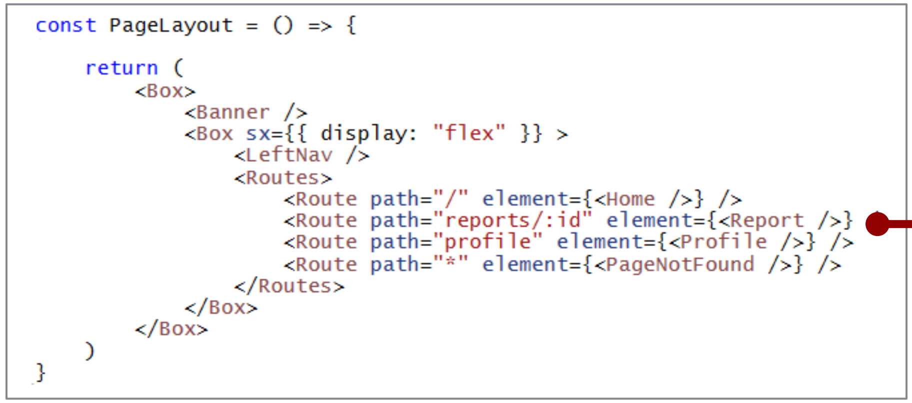
Report.tsx

- **Report.tsx** contains programming logic used to embed reports
  - Uses routing parameters to parse the ID of a report or dataset from URL
  - Acquires embed tokens and manages embed token lifetime
  - Uses the Power BI JavaScript API to embed reports
  - Leverages child components in **report** folder such as **ReportToolbar.tsx** and **ReportPath.tsx**

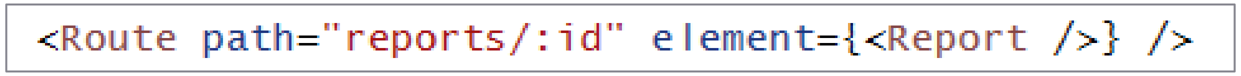


Routing to Report.tsx with a Report ID or Dataset ID

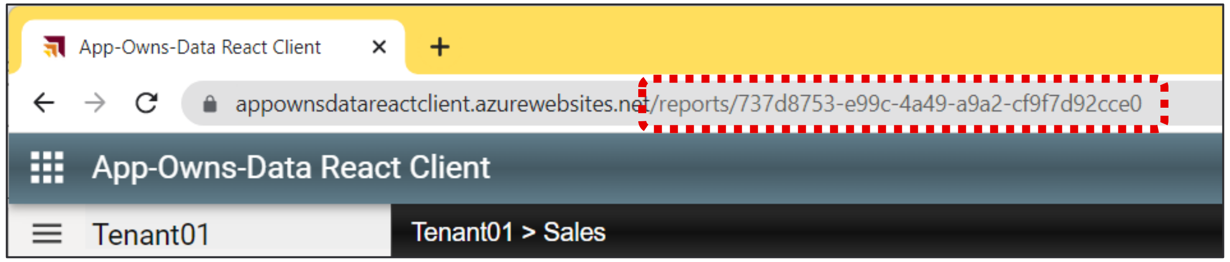
- **AppOwnsDataReactClient** uses a custom routing scheme with React Router
  - Routing scheme for application defined in **PageLayout.tsx**
  - Path for routing to existing report: **/reports/{ReportId}**
  - Path for routing to new report: **/reports/{DatasetId}**
  - **LeftNav.tsx** provides menu allowing user to navigate to route with Report ID or Dataset ID
  - **Report.tsx** contains code to inspect route and determine ID for target report or dataset



```
const PageLayout = () => {

    return (
        <Box>
            <Banner />
            <Box sx={{ display: "flex" }} >
                <LeftNav />
                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="reports/:id" element={<Report />} />
                    <Route path="profile" element={<Profile />} />
                    <Route path="*" element={<PageNotFound />} />
                </Routes>
            </Box>
        </Box>
    )
}
```



```
<Route path="reports/:id" element={<Report />} />
```



App-Owns-Data React Client

appownsdatareactclient.azurewebsites.net/reports/737d8753-e99c-4a49-a9a2-cf9f7d92cce0

App-Owns-Data React Client

Tenant01

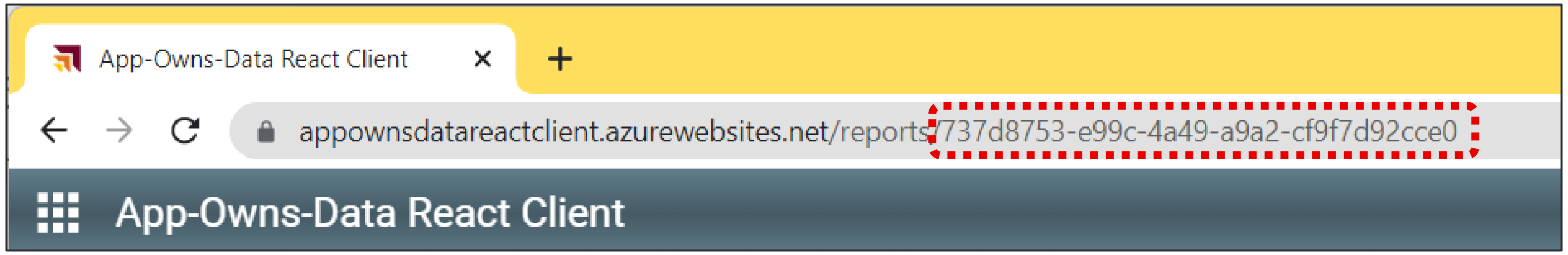
Tenant01 > Sales

Retrieving the ID Parameter from the Route Path

- Route path to **Reports.tsx** defines path parameter named **id**

```
<Route path="reports/:id" element={<Report />} />
```

- The **id** parameter has value of URL segment after **reports/**



appownsdatareactclient.azurewebsites.net/reports/737d8753-e99c-4a49-a9a2-cf9f7d92cce0

App-Owns-Data React Client

- The **id** parameter can be retrieved in **Reports.tsx** using **useParams** hook

```
import { useNavigate, useParams } from "react-router-dom";

const Report = () => {

  const navigate = useNavigate();
  const { id } = useParams();
```

Getting Embed Tokens in Report.tsx

- **Report.tsx** has state properties for storing embed token and related data

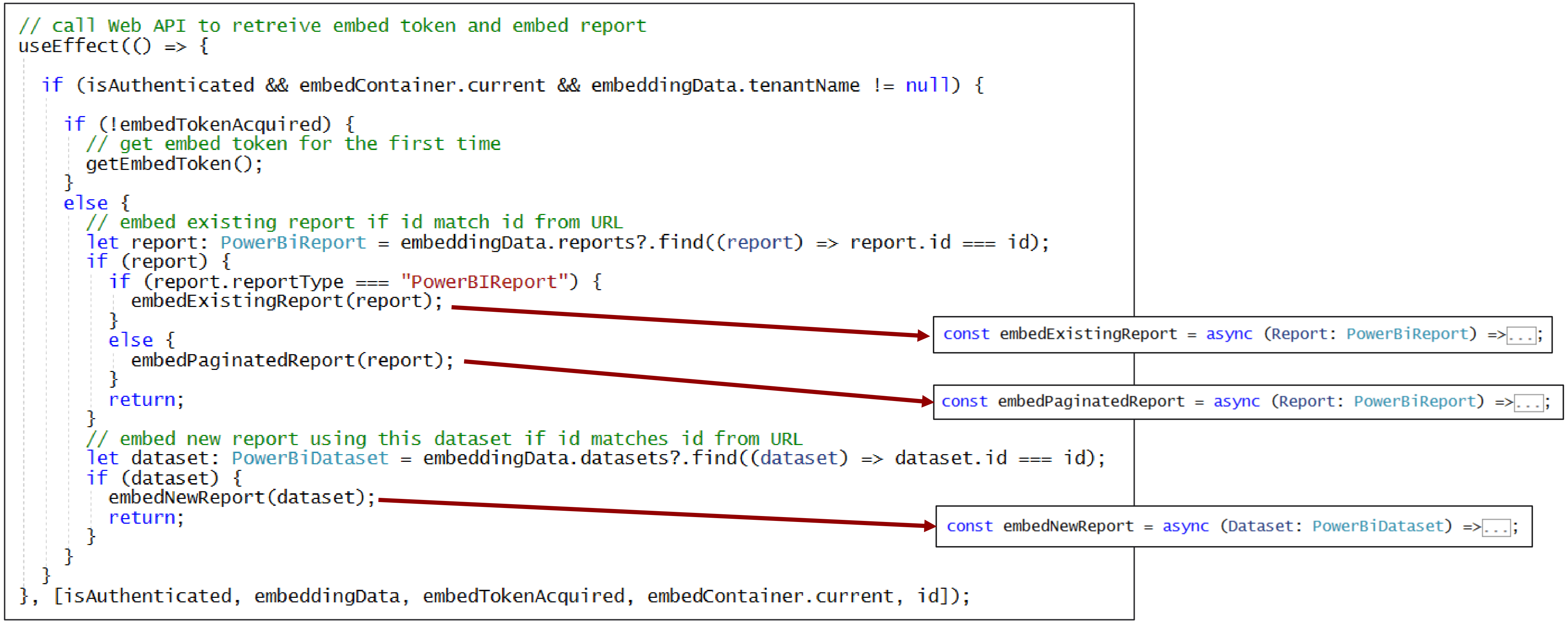
```
const Report = () => {

  const [embedToken, setEmbedToken] = useState<string>(null);
  const [embedTokenExpiration, setEmbedTokenExpiration] = useState<string>(null);
  const [embedTokenAcquired, setEmbedTokenAcquired] = useState<boolean>(false);
  const [embedTokenExpirationDisplay, setEmbedTokenExpirationDisplay] = useState<string>("");

  const [embeddedReport, setEmbeddedReport] = useState<powerbi.Report | null>(null);
  const [embeddedNewReport, setEmbeddedNewReport] = useState<powerbi.Embed | null>(null);
```

- **GetEmbedToken** retrieves embed token data and stores it in state properties

```
const getEmbedToken = async () => {
  let tokenResult = await AppOwnsDataWebApi.GetEmbedToken();
  setEmbedToken(tokenResult.embedToken);
  setEmbedTokenExpiration(tokenResult.embedTokenExpiration);
  setEmbedTokenAcquired(true);
  monitorTokenExpiration(tokenResult.embedTokenExpiration);
};
```

Initialization Logic in Report.tsx

- **useEffect** contains logic to acquire embed token and start embedding process
  - Ensures embed token has been acquired
  - Performs ID lookup on reports and datasets to dispatch to embed processing function



```
// call Web API to retreive embed token and embed report
useEffect(() => {

  if (isAuthenticated && embedContainer.current && embeddingData.tenantName != null) {

    if (!embedTokenAcquired) {
      // get embed token for the first time
      getEmbedToken();
    }
    else {
      // embed existing report if id match id from URL
      let report: PowerBiReport = embeddingData.reports?.find((report) => report.id === id);
      if (report) {
        if (report.reportType === "PowerBIReport") {
          embedExistingReport(report);
        }
        else {
          embedPaginatedReport(report);
        }
        return;
      }
      // embed new report using this dataset if id matches id from URL
      let dataset: PowerBiDataset = embeddingData.datasets?.find((dataset) => dataset.id === id);
      if (dataset) {
        embedNewReport(dataset);
        return;
      }
    }
  }
}, [isAuthenticated, embeddingData, embedTokenAcquired, embedContainer.current, id]);
```

```
const embedExistingReport = async (Report: PowerBiReport) =>...;
```

```
const embedPaginatedReport = async (Report: PowerBiReport) =>...;
```

```
const embedNewReport = async (Dataset: PowerBiDataset) =>...;
```

Report.tsx Output

- State properties in **Report.tsx** used to track report metadata

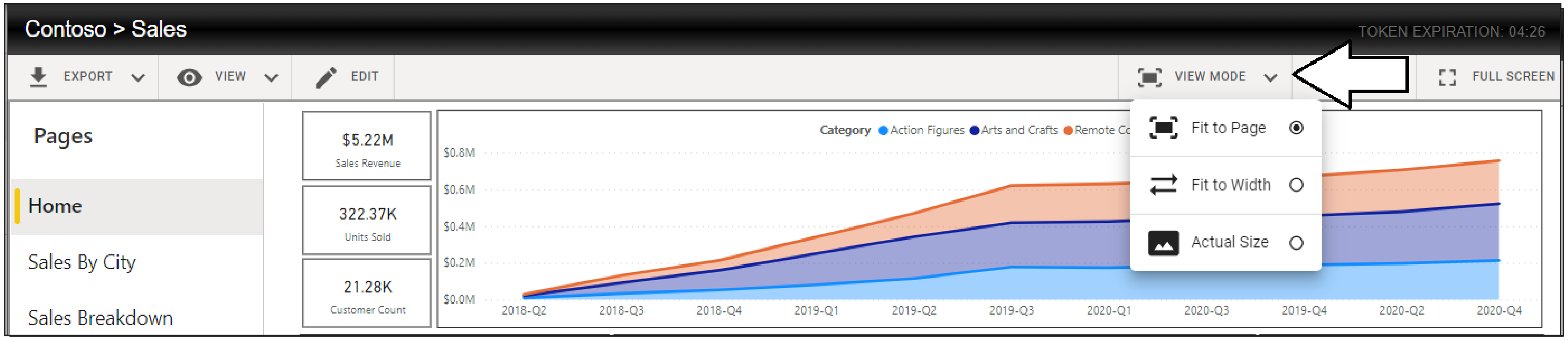
```
const [embeddedReport, setEmbeddedReport] = useState<powerbi.Report | null>(null);
const [embeddedNewReport, setEmbeddedNewReport] = useState<powerbi.Embed | null>(null);

const [embedType, setEmbedType] = useState< "ExistingReport" | "NewReport" | null>(null);
const [reportType, setReportType] = useState< "PowerBiReport" | "PaginatedReport" | null>(null);
const [viewMode, setViewMode] = useState<ViewMode>("FitToPage");
const [editMode, setEditMode] = useState(false);
const [showNavigation, setShowNavigation] = useState(true);
const [showFiltersPane, setShowFiltersPane] = useState(true);
const [showBookmarksPane, setShowBookmarksPane] = useState(false);
const [reportPath, setReportPath] = useState("");
```

- **Report.tsx** output adds **embedContainer** and child components
  - **Report.tsx** state properties passed to **ReportToolbar.tsx**

```
<Box sx={{ display: "grid", gridAutoFlow: "row", width: 1 }}>

  <ReportPath reportPath={reportPath} tokenExpiration={embedTokenExpirationDisplay} refreshEmbedToken={refreshEmbedToken} />

  {embedType === "ExistingReport" && reportType === "PowerBiReport" &&
    <ReportToolbar report={embeddedReport}
      editMode={editMode} setEditMode={setEditMode} showNavigation={showNavigation} setShowNavigation={setShowNavigation}
      showFiltersPane={showFiltersPane} setShowFiltersPane={setShowFiltersPane} viewMode={viewMode} setViewMode={setViewMode}
      showBookmarksPane={showBookmarksPane} setShowBookmarksPane={setShowBookmarksPane} setEmbedToken={setEmbedToken}
      setEmbedTokenExpiration={setEmbedTokenExpiration} />}

  {embedType === "NewReport" && <NewReportToolbar report={embeddedNewReport} />}

  <Box ref={embedContainer} sx={{ width: "100%", borderBottom: 1, borderBottomColor: "#CCCCCC" }} />

</Box>
```

ReportToolbar.tsx

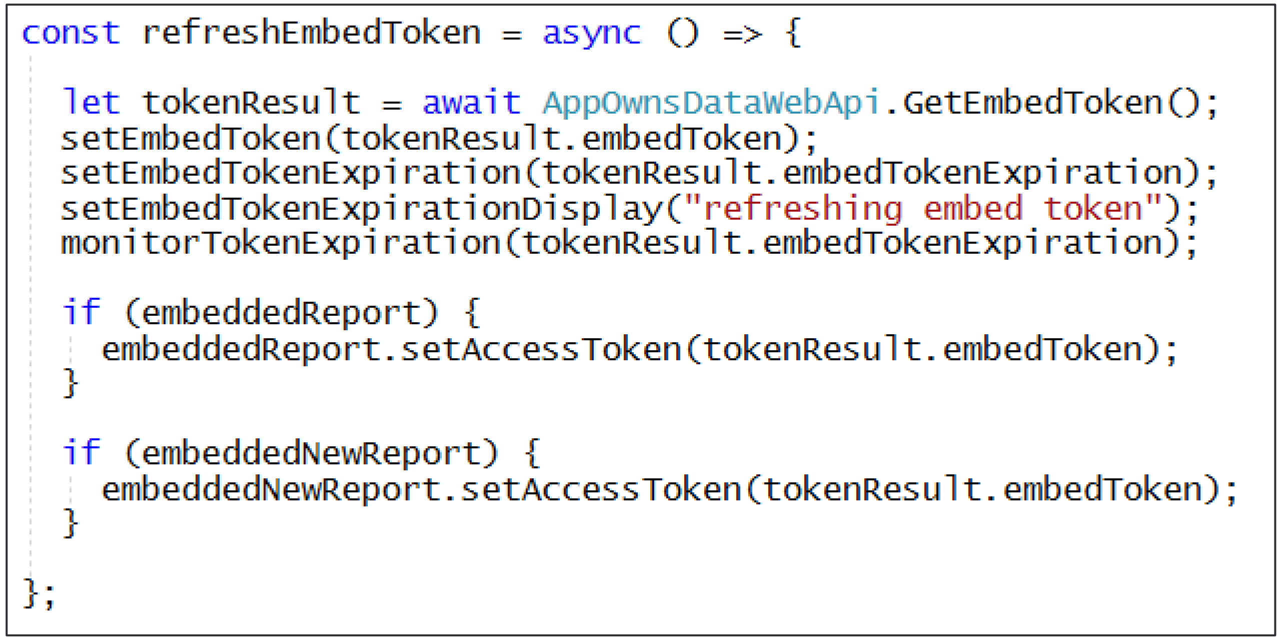
- **ReportToolbar.tsx** provides menu options to report consumers



Agenda

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ✓ Designing a View Model for a Multitenant Application
- ✓ Designing a Functional Component for Embedding Reports
- ➢ Managing Embed Token Expiration
- Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

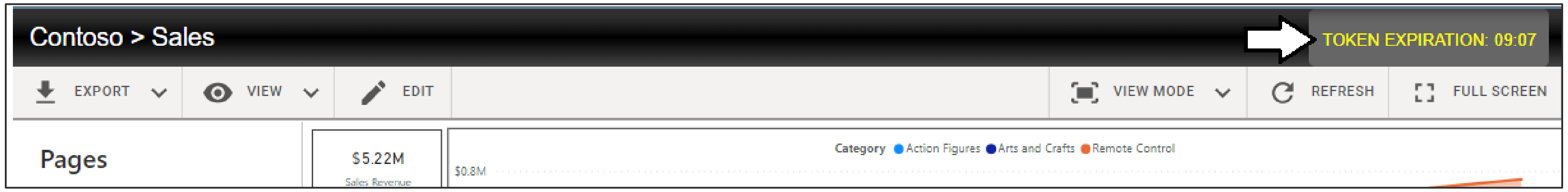
refreshEmbedToken in Report.tsx



```
const refreshEmbedToken = async () => {

  let tokenResult = await AppOwnsDataWebApi.GetEmbedToken();
  setEmbedToken(tokenResult.embedToken);
  setEmbedTokenExpiration(tokenResult.embedTokenExpiration);
  setEmbedTokenExpirationDisplay("refreshing embed token");
  monitorTokenExpiration(tokenResult.embedTokenExpiration);

  if (embeddedReport) {
    embeddedReport.setAccessToken(tokenResult.embedToken);
  }

  if (embeddedNewReport) {
    embeddedNewReport.setAccessToken(tokenResult.embedToken);
  }

};
```

Embed Token Auto-refresh Strategy

- Token expiration time displayed to user with **ReportPath.tsx**



- **monitorTokenExpiration** calls **refreshEmbedToken** if expiration is two minutes away or less

```
const monitorTokenExpiration = (EmbedTokenExpiration: string): void => {

  var secondsToExpire = Math.floor((new Date(EmbedTokenExpiration).getTime() - new Date().getTime()) / 1000);

  // auto-refresh embed token 2 minutes before it expires
  var secondsBeforeExpirationForAutoRefresh = 2 * 60;
  if (secondsToExpire < secondsBeforeExpirationForAutoRefresh) {
    refreshEmbedToken();
  }
  else {
    var minutes = Math.floor(secondsToExpire / 60);
    var seconds = secondsToExpire % 60;
    var timeToExpire = "Token Expiration: " + String(minutes).padStart(2, "0") + ":" + String(seconds).padStart(2, "0");
    setEmbedTokenExpirationDisplay(timeToExpire);
  }
};
```

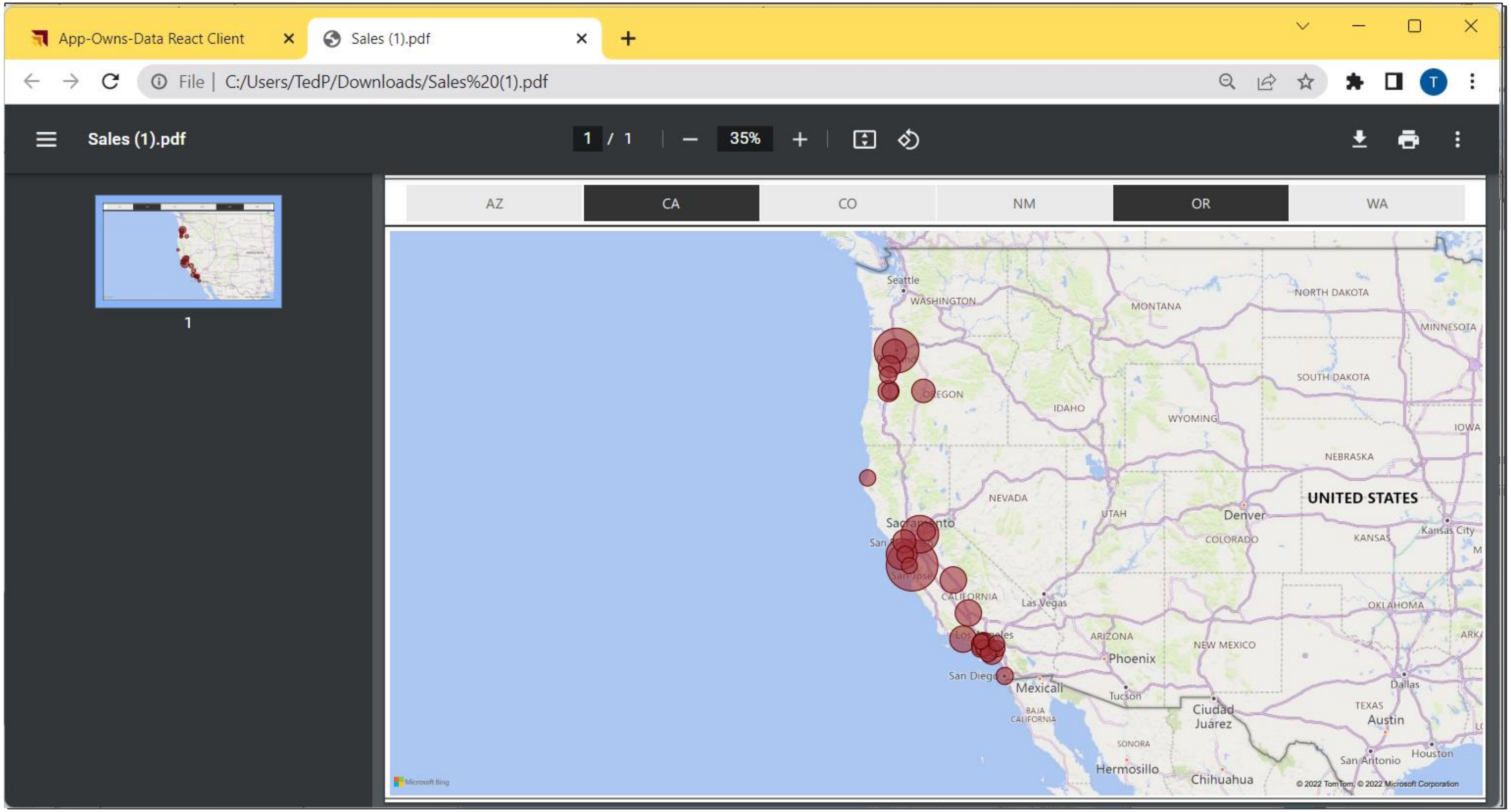
- **useEffect** function calls **setTimeout** to periodically call **monitorTokenExpiration** once a minute

```
// set up repeating effect to update display for embed token expiration time
useEffect(() => {
  if (isAuthenticated && embedTokenAcquired) {
    window.setTimeout(() => {
      monitorTokenExpiration(embedTokenExpiration);
    }, 1000);
  }
}, [isAuthenticated, embedTokenAcquired, embedTokenExpiration, embedTokenExpirationDisplay]);
```

Agenda

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ✓ Designing a View Model for a Multitenant Application
- ✓ Designing a Functional Component for Embedding Reports
- ✓ Managing Embed Token Expiration
- ➢ Providing Users with Export-to-File Commands
- Designing a Self-service Authoring Experience

User Experience for Exporting Reports



Capturing Bookmark State and Active Page Name

- Creating a richer user experience for exporting reports
  - Export report with same filtering as the users sees in browser
  - Give choice to export current page or all visible report page

- Power BI JavaScript API provides access to data about current report
  - Active page name can be captured dynamically using **getActivePage** and **name**
  - Bookmark can be captured dynamically using **bookmarkManager.capture**
  - Custom bookmark state used to pass current filtering state to export job

```
// get report data for ExportFile operation
let reportId = report.getId();
let currentPage = await report.getActivePage();
let currentPageName = currentPage.name;
let bookmark = await report.bookmarksManager.capture({ allPages: false, personalizeVisuals: false });
```

Calling AppOwnsDataWebApi.ExportFile

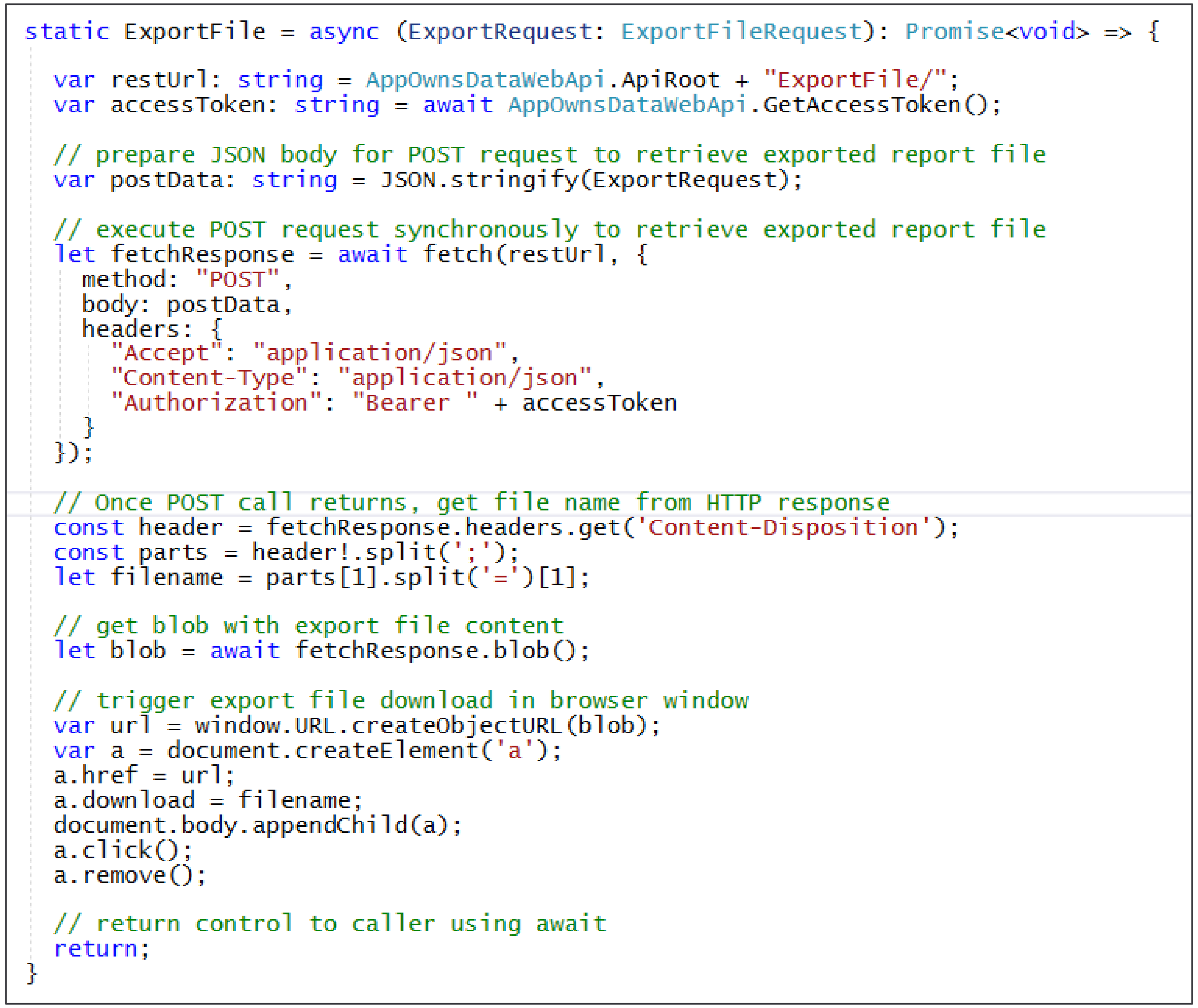
- **ExportFileReqest** defined in **models.ts**

```
export class ExportFileRequest {
  ReportId: string;
  ExportType: "PDF" | "PNG" | "PPTX";
  Filter?: string;
  BookmarkState?: string;
  PageName?: string;
  VisualName?: string;
}
```

- Sample event handler in **ReportToolbar.tsx** for exporting single page to PDF

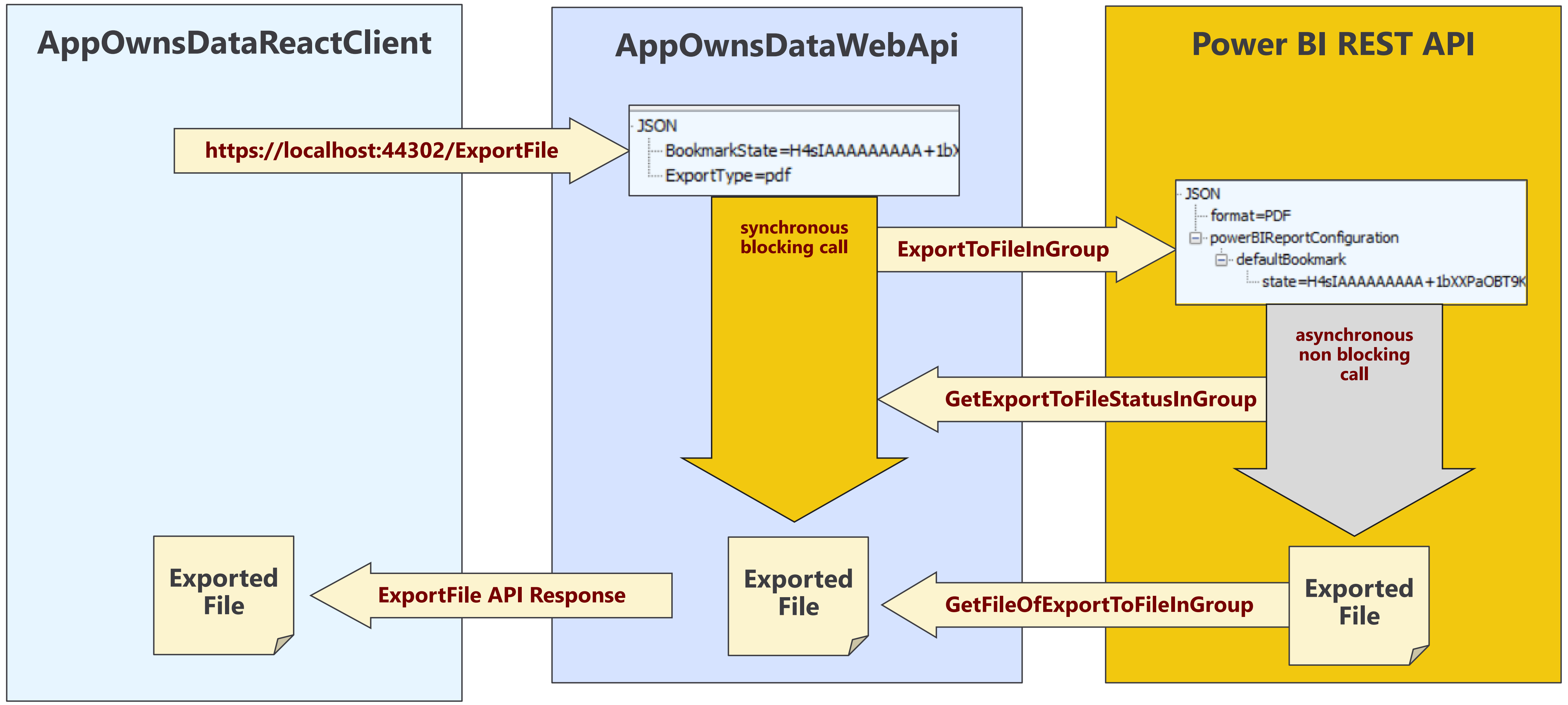
```
const onExportPageToPDF = async () => {

  // close Export menu and open export progress dialog
  setAnchorElementExport(null);
  setOpenExportProgressDialog(true);

  // get report data for ExportFile operation
  let reportId = report.getId();
  let currentPage = await report.getActivePage();
  let currentPageName = currentPage.name;
  let bookmark = await report.bookmarksManager.capture({ allPages: false, personalizeVisuals: false });

  // create ExportFileRequest variable with parameters for export
  const exportRequest: ExportFileRequest = {
    ReportId: reportId,
    ExportType: "PDF",
    BookmarkState: bookmark.state,
    PageName: currentPageName,
  };

  // Call ExportFile from AppOwnsDataWebApi
  await AppOwnsDataWebApi.ExportFile(exportRequest);

  // close export progress dialog
  setOpenExportProgressDialog(false);

};
```

AppOwnsDataWebApi.ExportFile Implementation



```
static ExportFile = async (ExportRequest: ExportFileRequest): Promise<void> => {

  var restUrl: string = AppOwnsDataWebApi.ApiRoot + "ExportFile/";
  var accessToken: string = await AppOwnsDataWebApi.GetAccessToken();

  // prepare JSON body for POST request to retrieve exported report file
  var postData: string = JSON.stringify(ExportRequest);

  // execute POST request synchronously to retrieve exported report file
  let fetchResponse = await fetch(restUrl, {
    method: "POST",
    body: postData,
    headers: {
      "Accept": "application/json",
      "Content-Type": "application/json",
      "Authorization": "Bearer " + accessToken
    }
  });

  // Once POST call returns, get file name from HTTP response
  const header = fetchResponse.headers.get('Content-Disposition');
  const parts = header!.split(';');
  let filename = parts[1].split('=')[1];

  // get blob with export file content
  let blob = await fetchResponse.blob();

  // trigger export file download in browser window
  var url = window.URL.createObjectURL(blob);
  var a = document.createElement('a');
  a.href = url;
  a.download = filename;
  document.body.appendChild(a);
  a.click();
  a.remove();

  // return control to caller using await
  return;
}
```

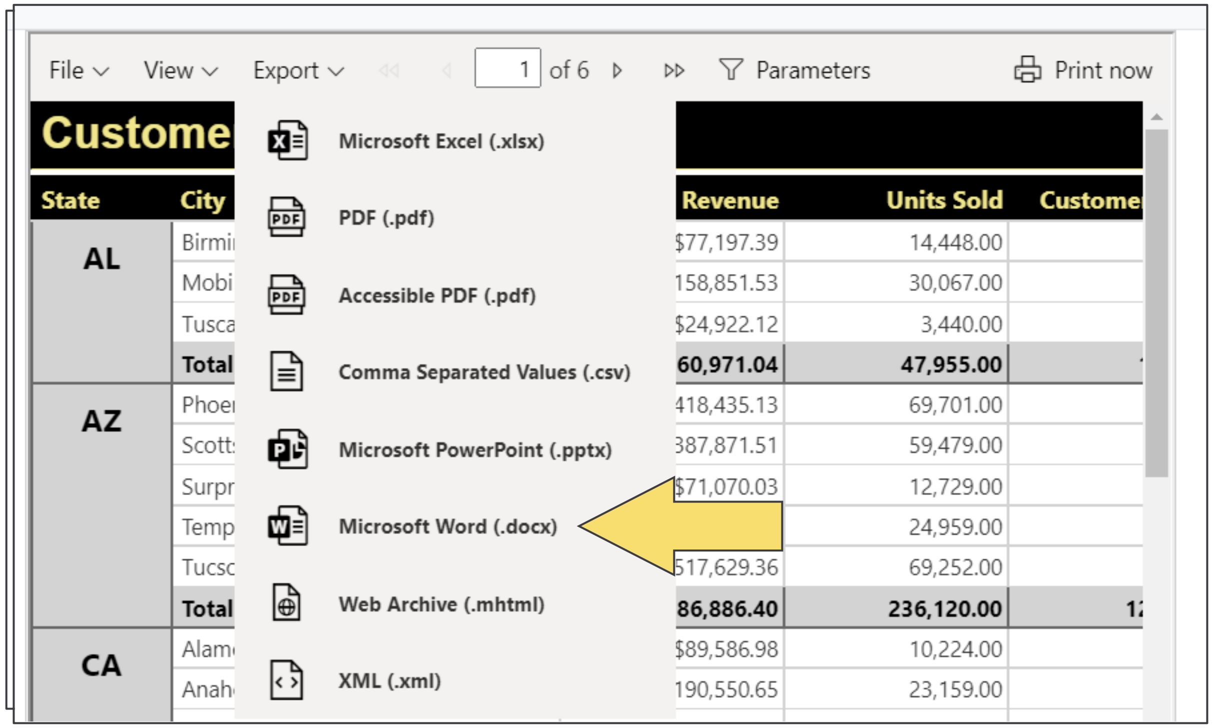
App-Owns-Data Starter Kit Export Processing Architecture

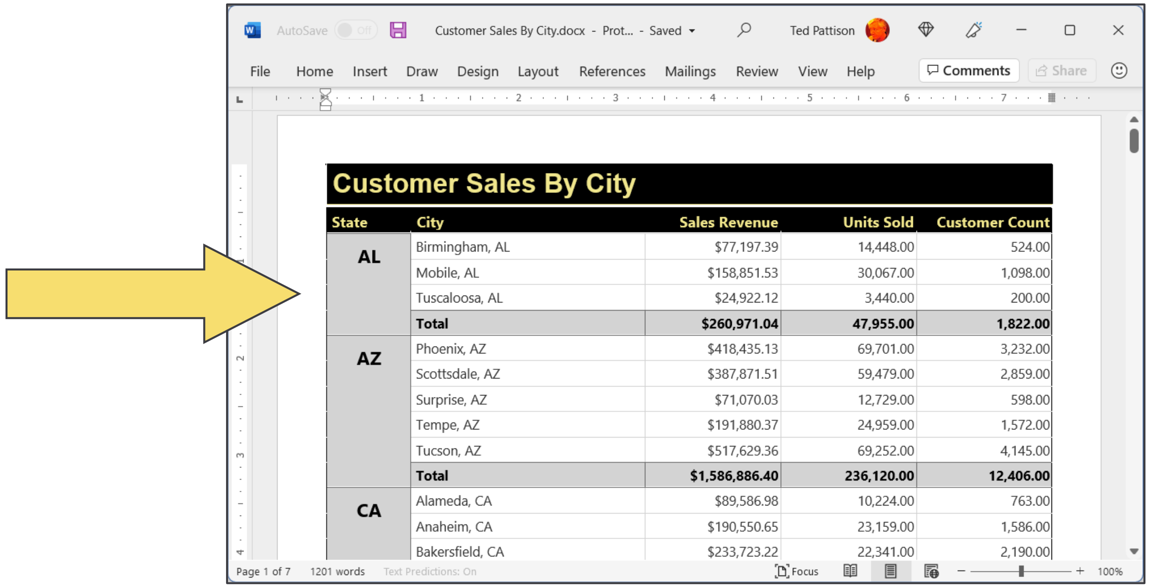

AppOwnsDataReactClient
AppOwnsDataWebApi
Power BI REST API
https://localhost:44302/ExportFile
JSON
BookmarkState=H4sIAAAAAAAAA+1bX
ExportType=pdf
synchronous blocking call
ExportToFileInGroup
JSON
format=PDF
powerBIReportConfiguration
defaultBookmark
state=H4sIAAAAAAAAA+1bXXPaOBT9K
asynchronous non blocking call
GetExportToFileStatusInGroup
Exported File
ExportFile API Response
Exported File
GetFileOfExportToFileInGroup
Exported File

App-Owns-Data Embedding with Paginated Reports

Embedded paginated reports automatically display built-in Export menu

- User can use **Export** menu to download export by hand in any available format
- No need to include custom web API as with exporting Power BI report with App-Owns-Data



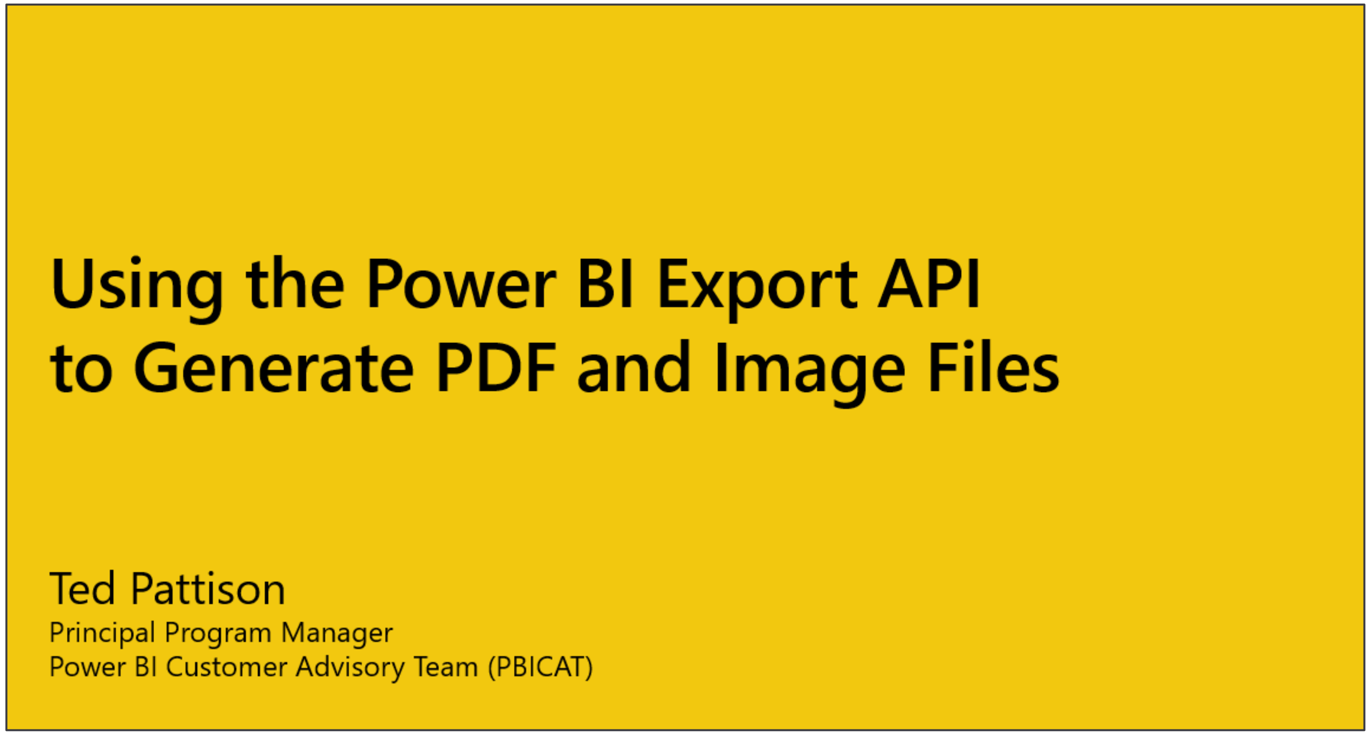


Customer Sales By City

State	City	Sales Revenue	Units Sold	Customer Count
AL	Birmingham, AL	$77,197.39	14,448.00	524.00
	Mobile, AL	$158,851.53	30,067.00	1,098.00
	Tuscaloosa, AL	$24,922.12	3,440.00	200.00
	Total	**$260,971.04**	**47,955.00**	**1,822.00**
AZ	Phoenix, AZ	$418,435.13	69,701.00	3,232.00
	Scottsdale, AZ	$387,871.51	59,479.00	2,859.00
	Surprise, AZ	$71,070.03	12,729.00	598.00
	Tempe, AZ	$191,880.37	24,959.00	1,572.00
	Tucson, AZ	$517,629.36	69,252.00	4,145.00
	Total	**$1,586,886.40**	**236,120.00**	**12,406.00**
CA	Alameda, CA	$89,586.98	10,224.00	763.00
	Anaheim, CA	$190,550.65	23,159.00	1,586.00
	Bakersfield, CA	$233,723.22	22,341.00	2,190.00

Review the Dev Camp Session on the Power BI Export API

- **Session 16: Using the Power BI Export API to Generate PDF and Image Files**
  - Webinar Recording: **https://youtu.be/ybWWTVt_guA**
  - Links and Resources: **https://www.powerbidevcamp.net/sessions/session16/**



Using the Power BI Export API
to Generate PDF and Image Files

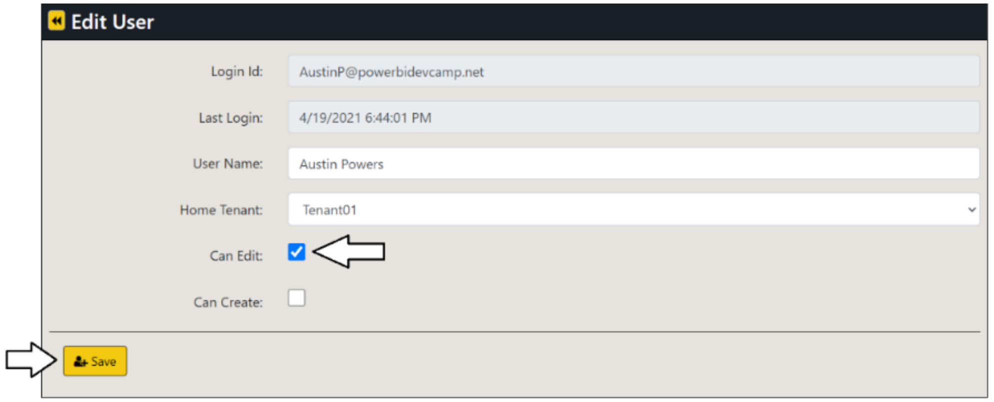
Ted Pattison
Principal Program Manager
Power BI Customer Advisory Team (PBICAT)

Agenda

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ✓ Designing a View Model for a Multitenant Application
- ✓ Designing a Functional Component for Embedding Reports
- ✓ Managing Embed Token Expiration
- ✓ Providing Users with Export-to-File Commands
- ➢ Designing a Self-service Authoring Experience

Setting User Permissions in AppOwnsDataAdmin

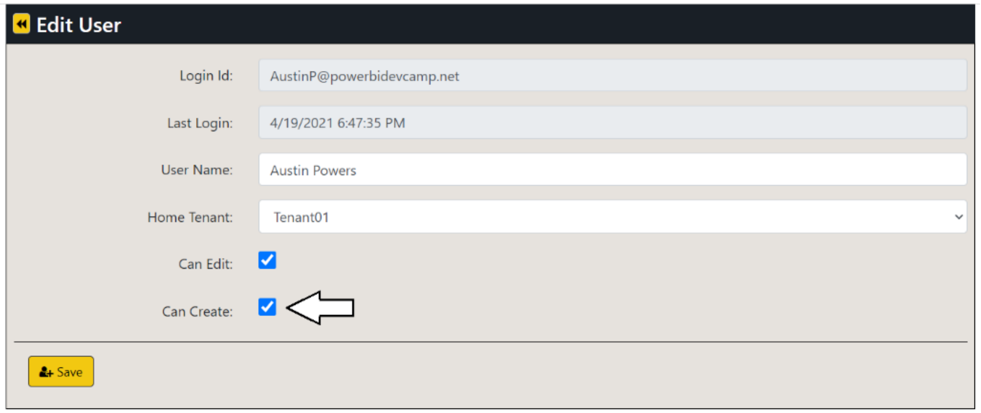
- Setting **Can Edit** allows user to edit and save changes to existing report



Edit User

Login Id:	AustinP@powerbidevcamp.net
Last Login:	4/19/2021 6:44:01 PM
User Name:	Austin Powers
Home Tenant:	Tenant01
Can Edit:	☑
Can Create:	☐

- Setting **Can Create** allows user to create new reports

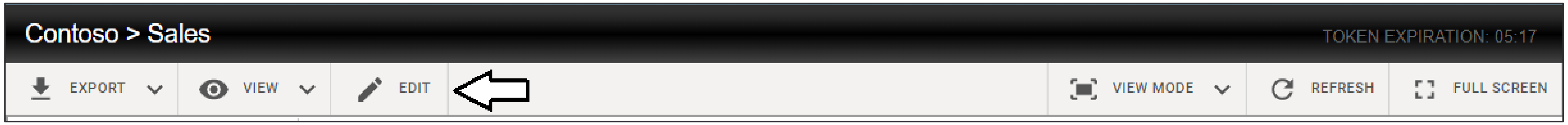


Edit User

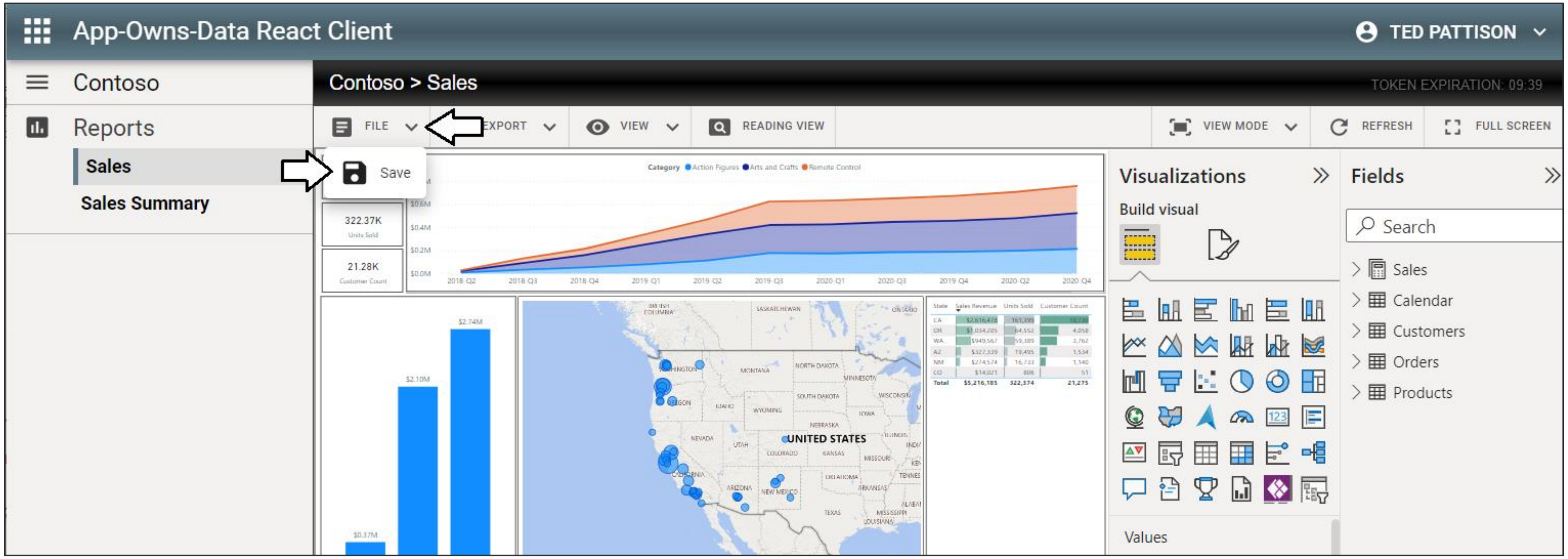
Login Id:	AustinP@powerbidevcamp.net
Last Login:	4/19/2021 6:47:35 PM
User Name:	Austin Powers
Home Tenant:	Tenant01
Can Edit:	☑
Can Create:	☑

Edit reports using AppOwnsDataReactClient

- User with **Can Edit** permissions can move report into edit mode

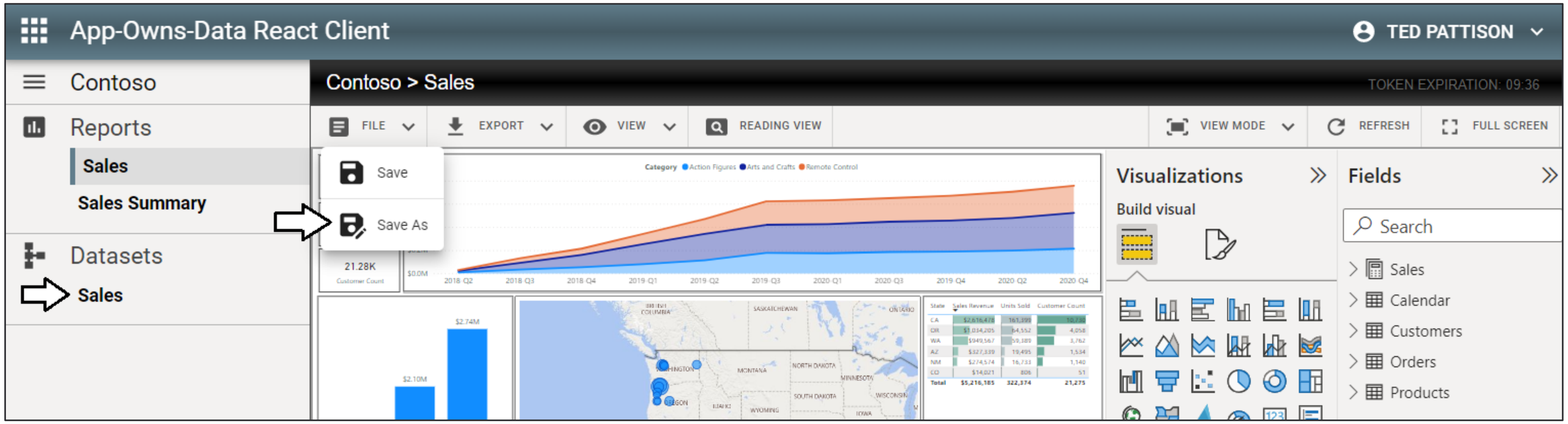


- Once in edit mode, report author can customize report and save changes



Create New Content using AppOwnsDataClient

- User with **Can Create** permissions can create new reports
  - User can use **Save As** command on report in edit mode to copy a report
  - User can click dataset link in **Datasets** section to create new report



Refreshing EmbedToken and EmbeddingData

- Creating new report invalidates embed token and embedding data
  - **EmbedToken** does not contain report ID for new report
  - **EmbeddingData** does not contain new report in **Reports** collection
- What happens when new report is created with **Save As** command?
  - Call to **refreshEmbedToken**
  - Call to **refreshEmbeddinfData**
  - Log **CopyReport** activity event
  - Navigate to URL with ID of new report

```
embeddedReport.on("saved", async (event: any) => {

  if (event.detail.saveAs) {
    // handle save-as with newly created report
    await refreshEmbedToken();
    await refreshEmbeddingData();
    var newReportId = event.detail.reportObjectId;
    var newReportName = event.detail.reportName;
    logCopyReportActivity(Report, newReportId, newReportName);
    navigate("/reports/" + newReportId + "/?edit=true");
  }
  else {
    // handle save to to edit exisitng report
    logEditReportActivity(Report);
  }

});
```

Summary

- ✓ Quick Review of Modern React
- ✓ App-Owns-Data Starter Kit Architecture
- ✓ Designing a View Model for a Multitenant Application
- ✓ Designing a Functional Component for Embedding Reports
- ✓ Managing Embed Token Expiration
- ✓ Providing Users with Export-to-File Commands
- ✓ Designing a Self-service Authoring Experience

Call to Action

- Download and try out the **App-Owns-Data Starter Kit** and **AppOwnsDataReactClient**
  - https://github.com/PowerBiDevCamp/App-Owns-Data-Starter-Kit
- Learn more about modern React
  - https://beta.reactjs.org
- Join Power BI Dev Camp Next Month for More React Coverage
  - Next month's session covers using Azure B2C authentication with App-Owns-Data embedding
  - https://aka.ms/devcamplive
- Live and Love the Power BI Community
  - https://community.powerbi.com/

Questions?

Microsoft Power BI