```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import {
  AUTH_TOKEN,
  BRANCH_ID,
  FIREBASE_TOKEN,
  PERMIT,
  REFRESH_TOKEN,
  USER_ROLES,
} from "constants/AuthConstant";
import FirebaseService from "services/FirebaseService";
import AuthService from "services/AuthService";
import { setAuthenData } from "utils/helper";
import { cloneDeep } from "lodash";
import { deleteToken, getMessaging } from "firebase/messaging";
import { generateToken } from "auth/FirebaseAuth";

export const initialState = {
  profile: null,
  profileById: null,
  loading: false,
  message: "",
  showMessage: false,
  redirect: "",
  token: localStorage.getItem(AUTH_TOKEN) || null,
  permitList: JSON.parse(localStorage.getItem(PERMIT)) || [],
  userRoles: localStorage.getItem(USER_ROLES) || [],
  forgotPw: {},
  branchId: sessionStorage.getItem(BRANCH_ID) || null,
};

export const getProfileApi = createAsyncThunk(
  "auth/profile",
  async (data, { rejectWithValue }) => {
    try {
      const response = await AuthService.getProfileApi();
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getProfileByIdApi = createAsyncThunk(
  "auth/getProfileByIdApi",
  async (id, { rejectWithValue }) => {
    try {
      const response = await AuthService.getProfileByIdApi(id);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const UpdateProfileApi = createAsyncThunk(
  "auth/UpdateProfileApi",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await AuthService.updateProfile(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signIn = createAsyncThunk(
```

```
  "auth/signIn",
  async (data, { rejectWithValue }) => {
    const { email, password } = data;
    try {
      await generateToken();
      const firebaseToken = localStorage.getItem(FIREBASE_TOKEN);

      const response = await AuthService.login({
        emailOrPhone: email,
        password: password,
        firebaseToken: firebaseToken,
      });
      localStorage.setItem(PERMIT, JSON.stringify(response.data.permits));
      setAuthenData(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signUp = createAsyncThunk(
  "auth/signUp",
  async (data, { rejectWithValue }) => {
    const { email, password } = data;
    try {
      const response = await FirebaseService.signUpEmailRequest(
        email,
        password
      );
      if (response.user) {
        const token = response.user.refreshToken;
        localStorage.setItem(AUTH_TOKEN, response.user.refreshToken);
        return token;
      } else {
        return rejectWithValue(response.message?.replace("Firebase: ", ""));
      }
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const forgotPasswordApi = createAsyncThunk(
  "auth/forgotPassword",
  async (data, { rejectWithValue }) => {
    try {
      data.secretKey = process.env.REACT_APP_SECRETKEY;
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await AuthService.forgotPassword(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const ResetPasswordApi = createAsyncThunk(
  "auth/resetPasswordApi",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await AuthService.resetPassword(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
```

```
);

export const ChangePasswordApi = createAsyncThunk(
  "auth/ChangePasswordApi",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await AuthService.changePassword(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signOutApi = createAsyncThunk(
  "auth/signOutApi",
  async (data, { rejectWithValue, getState }) => {
    try {
      const messaging = getMessaging();
      const token = localStorage.getItem(AUTH_TOKEN);
      const firebaseToken = localStorage.getItem(FIREBASE_TOKEN);
      const refreshToken = localStorage.getItem(REFRESH_TOKEN);
      localStorage.removeItem(REFRESH_TOKEN);
      localStorage.removeItem(FIREBASE_TOKEN);
      localStorage.removeItem(AUTH_TOKEN);
      sessionStorage.removeItem(BRANCH_ID);
      localStorage.removeItem(PERMIT);
      //deleteToken(messaging);
      const response = await AuthService.signOutApi({
        token,
        refreshToken,
        firebaseToken,
      });
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signInWithGoogle = createAsyncThunk(
  "auth/signInWithGoogle",
  async (_, { rejectWithValue }) => {
    const response = await FirebaseService.signInGoogleRequest();
    if (response.user) {
      const token = response.user.refreshToken;
      localStorage.setItem(AUTH_TOKEN, response.user.refreshToken);
      return token;
    } else {
      return rejectWithValue(response.message?.replace("Firebase: ", ""));
    }
  }
);

export const signInWithFacebook = createAsyncThunk(
  "auth/signInWithFacebook",
  async (_, { rejectWithValue }) => {
    const response = await FirebaseService.signInFacebookRequest();
    if (response.user) {
      const token = response.user.refreshToken;
      localStorage.setItem(AUTH_TOKEN, response.user.refreshToken);
      return token;
    } else {
      return rejectWithValue(response.message?.replace("Firebase: ", ""));
    }
  }
);

export const searchByRoles = createAsyncThunk(
  "auth/searchByRoles",
```

```
  async (data, { rejectWithValue }) => {
    try {
      const response = await AuthService.searchByRoles(data);
      return response.data;
    } catch (error) {
      return rejectWithValue(error.message || "Error");
    }
  }
);

export const authSlice = createSlice({
  name: "auth",
  initialState,
  reducers: {
    authenticated: (state, action) => {
      state.loading = false;
      state.redirect = "/";
      state.token = action.payload;
    },
    showAuthMessage: (state, action) => {
      state.message = action.payload;
      state.showMessage = true;
      state.loading = false;
    },
    hideAuthMessage: (state) => {
      state.message = "";
      state.showMessage = false;
    },
    signOutSuccess: (state) => {
      state.loading = false;
      state.token = null;
      state.redirect = "/";
    },
    showLoading: (state) => {
      state.loading = true;
    },
    signInSuccess: (state, action) => {
      state.loading = false;
      state.token = action.payload;
    },
    setForgotPw: (state, action) => {
      state.forgotPw = action.payload;
    },
    setBranchId: (state, action) => {
      state.branchId = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(signIn.pending, (state) => {
        state.loading = true;
      })
      .addCase(signIn.fulfilled, (state, action) => {
        const location = window.location.href;

        const redirectIndex = location.indexOf("redirect=");

        let redirectUrl = "";
        if (redirectIndex !== -1) {
          redirectUrl = location.substring(redirectIndex + "redirect=".length);
        }
        state.loading = false;
        state.redirect = redirectUrl;
        state.token = action.payload.tokenData.token;
        // state.userRoles = action.payload.userRoles;
        state.permitList = action.payload.permits;
      })
      .addCase(signIn.rejected, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(signUp.pending, (state) => {
        state.loading = true;
      })
```

```
.addCase(signUp.fulfilled, (state, action) => {
  state.loading = false;
  state.redirect = "/";
  state.token = action.payload;
})
.addCase(signUp.rejected, (state, action) => {
  state.loading = false;
  state.redirect = "/";
})
.addCase(signInWithGoogle.pending, (state) => {
  state.loading = true;
})
.addCase(signInWithGoogle.fulfilled, (state, action) => {
  state.loading = false;
  state.redirect = "/";
  state.token = action.payload;
})
.addCase(signInWithGoogle.rejected, (state, action) => {
  state.loading = false;
  state.redirect = "/";
})
.addCase(signInWithFacebook.pending, (state) => {
  state.loading = true;
})
.addCase(signInWithFacebook.fulfilled, (state, action) => {
  state.loading = false;
  state.redirect = "/";
  state.token = action.payload;
})
.addCase(signInWithFacebook.rejected, (state, action) => {
  state.loading = false;
  state.redirect = "/";
})
.addCase(getProfileApi.pending, (state) => {
  state.loading = true;
})
.addCase(getProfileApi.fulfilled, (state, action) => {
  state.loading = false;
  state.redirect = "/";
  state.profile = action.payload;
  state.permitList = action.payload.permits;
  localStorage.setItem(PERMIT, JSON.stringify(action.payload.permits));
})
.addCase(getProfileApi.rejected, (state, action) => {
  state.loading = false;
  state.redirect = "/";
})

.addCase(getProfileByIdApi.pending, (state) => {
  state.loading = true;
})
.addCase(getProfileByIdApi.fulfilled, (state, action) => {
  state.loading = false;
  state.profileById = action.payload;
})
.addCase(getProfileByIdApi.rejected, (state, action) => {
  state.loading = false;
})

.addCase(UpdateProfileApi.pending, (state) => {
  state.loading = true;
})
.addCase(UpdateProfileApi.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(UpdateProfileApi.rejected, (state, action) => {
  state.loading = false;
  state.redirect = "/";
})
.addCase(forgotPasswordApi.pending, (state) => {
  state.loading = true;
})
.addCase(forgotPasswordApi.fulfilled, (state, action) => {
  state.loading = false;
```

```
        state.redirect = "/";
        state.forgotPw.email = action.payload;
      })
      .addCase(forgotPasswordApi.rejected, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(ChangePasswordApi.pending, (state) => {
        state.loading = true;
      })
      .addCase(ChangePasswordApi.fulfilled, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(ChangePasswordApi.rejected, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(ResetPasswordApi.pending, (state) => {
        state.loading = true;
      })
      .addCase(ResetPasswordApi.fulfilled, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(ResetPasswordApi.rejected, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(signOutApi.pending, (state) => {
        state.loading = true;
      })
      .addCase(signOutApi.fulfilled, (state, action) => {
        state.loading = false;
        state.token = null;
        state.redirect = "/";
        state.branchId = null;
        state.profile = null;
        state.permitList = [];
      })
      .addCase(signOutApi.rejected, (state, action) => {
        state.loading = false;
        state.token = null;
        state.redirect = "/";
        state.branchId = null;
        state.profile = null;
        state.permitList = [];
      })
      .addCase(searchByRoles.pending, (state) => {
        state.loading = true;
      })
      .addCase(searchByRoles.fulfilled, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      })
      .addCase(searchByRoles.rejected, (state, action) => {
        state.loading = false;
        state.redirect = "/";
      });
  },
});

export const {
  authenticated,
  showAuthMessage,
  hideAuthMessage,
  signOutSuccess,
  showLoading,
  signInSuccess,
  setForgotPw,
  setBranchId,
} = authSlice.actions;

export default authSlice.reducer;
```