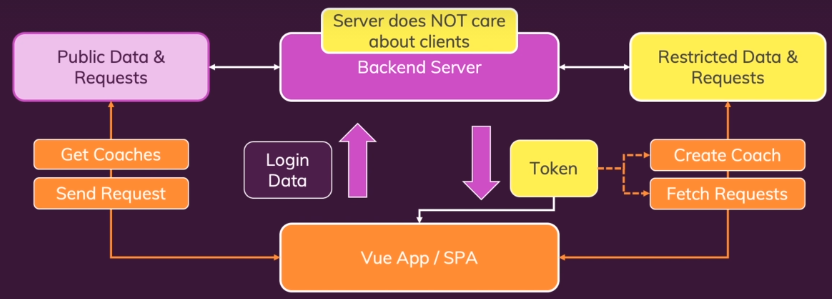
**Section 17 – Vue And Authentication**

Chapter 257 – How Authentication Works In Vue Apps (Or Any SPA)



Chapter 258 – Locking / Protecting Backend Resources

1. Make the rule like below.

{

"rules": {

"coaches": {

".read": true,

".write": "auth != null"

},

"requests": {

".read": "auth != null",

".write": true

}

}

}

Chapter 259 – Adding An Authentication Page (Login And Signup)

1. Make folder Auth and file UserAuth.vue. Write the template like below.

<template>

    <form>

        <base-card>

            <div class="form-control">

                <label for="email">E-Mail</label>

                <input type="email" id="email">

            </div>

            <div class="form-control">

                <label for="password">Password</label>

                <input type="password" id="password">

            </div>

            <base-button>Login</base-button>

            <base-button type="button" mode="flat">Sign Up Instead</base-button>

        </base-card>

    </form>

</template>

1. Write a method to validate the input data and switch mode like below.

methods: {

        submitForm() {

            this.formIsValid = true;

            if (this.email === '' ||

                !this.email.includes('@') ||

                this.password.length < 6) {

                    this.formIsValid = false;

                    return;

            }

        },

        switchAuthMode() {

            if (this.mode === 'login') {

                this.mode = 'signup';

            } else {

                this.mode = 'login';

            }

        }

    }

1. We also make a computed caption for login and sign up button like below.

computed: {

        submitButtonCaption() {

            if (this.mode === 'Login') {

                return 'Login';

            } else {

                return 'Signup';

            }

        },

        switchModeButtonCaption() {

            if (this.mode === 'login') {

                return 'Signup instead';

            } else {

                return 'Login instead';

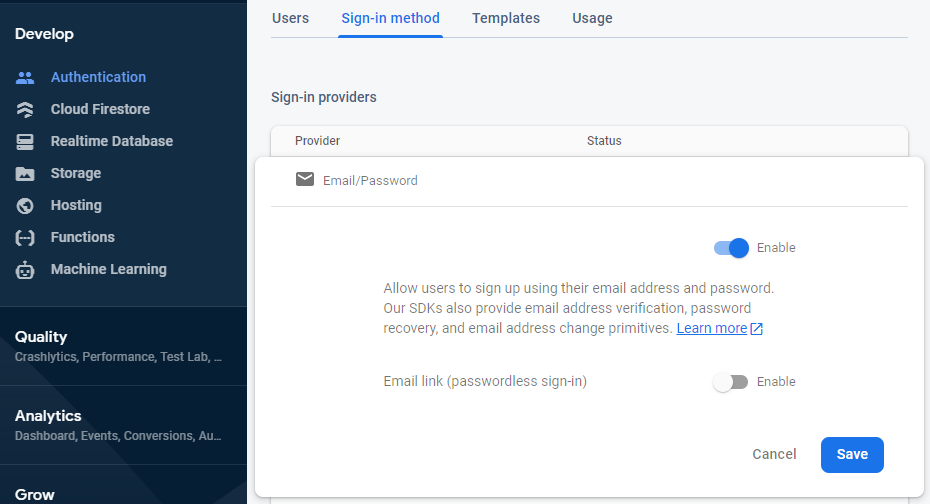
            }

        }

},

Chapter 260 – Preparing Vuex

1. Enable email setting. Authentication, Sign-in method, Enable Email, and Save to support Login.



1. Create some file like below.

Action.js

export default {

    login() {

    },

    signup(context, payload) {

        fetch()

    }

};

Getters.js

export default {

    userId(state) {

        return state.userId;

    }

};

Index.js

import mutations from './mutations.js';

import actions from './actions.js';

import getters from './getters.js';

export default {

    state() {

        return {

          userId: 'c3'

        };

    },

    mutations,

    actions,

    getters

}

Mutations.js

Chapter 261 – Adding A “Signup” Action And Flow

1. Write sign up method in actions.js.

export default {

    login() {

    },

    async signup(context, payload) {

        const response = await fetch('https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc', {

            method: 'POST',

            body: JSON.stringify({

                email: payload.email,

                password: payload.password,

                returnSecureToken: true

            })

        });

        const responseData = await response.json();

        if (!response.ok) {

            console.log(responseData);

            const error = new Error(responseData.message || 'Failed to authenticate.');

            throw error;

        }

        console.log(responseData);

        // Add below because we have added mutations to get return data

        context.commit('setUser', {

            token: responseData.idToken,

            userId: responseData.localId,

            tokenExpiration = responseData.expiresIn

        });

    }

};

1. Write below in mutations.js to get return data.

export default {

    setUser(state, payload) {

        state.token = payload.token;

        state.userId = payload.userId;

        state.tokenExpiration = payload.tokenExpiration;

    }

}

1. In index.js, we set return null value for userId, token, and tokenExpiration.

export default {

    state() {

        return {

          userId: null,

          token: null,

          tokenExpiration: null

        };

    },

    mutations,

    actions,

    getters

}

Chapter 262 – Better UX: Loading Spinner And Error Handling

1. Make the spinner and error handling using this code below.

<base-dialog :show="!!error" title="An error occured" @close="handleError">

<p>{{ error }}</p>

</base-dialog>

<base-dialog :show="isLoading" title="Authenticating..." fixed>

<p>Authenticating ...</p>

<base-spinner></base-spinner>

</base-dialog>

1. The handle error function will be like below.

handleError() {

this.error = null;

}

Chapter 263 – Adding A “Login” Action And Flow

1. Make Log In Function same likes Sign Up Function, only different URL.

In actions.js

async login(context, payload) {

        const response = await fetch('https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc', {

            method: 'POST',

            body: JSON.stringify({

                email: payload.email,

                password: payload.password,

                returnSecureToken: true

            })

        });

        const responseData = await response.json();

        if (!response.ok) {

            console.log(responseData);

            const error = new Error(responseData.message || 'Failed to authenticate. Check your login data.');

            throw error;

        }

        console.log(responseData);

        // Add below because we have added mutations to get return data

        context.commit('setUser', {

            token: responseData.idToken,

            userId: responseData.localId,

            tokenExpiration: responseData.expiresIn

        });

},

In Auth.vue

const actionPayLoad = {

                email: this.email,

                password: this.password

            };

            try {

                if (this.mode === 'login') {

                    await this.$store.dispatch('login', actionPayLoad);

                } else {

                    await this.$store.dispatch('signup', actionPayLoad);

                }

            } catch (err) {

                this.error = err.message || 'Failed to authenticate, try later.'

}

Chapter 264 – Attaching The Token To Outgoing Requests

1. Get token in getters.js

export default {

    userId(state) {

        return state.userId;

    },

    token(state) {

        return state.token;

    }

};

1. We need to add token when send a request.

async fetchRequests(context) {

    const coachId = context.rootGetters.userId;

    const token = context.rootGetters.token;

    const response = await fetch(

      `https://vue-http-demo-a1d4c.firebaseio.com/requests/${coachId}.json?auth=` +

        token

    );

    const responseData = await response.json();

    if (!response.ok) {

      const error = new Error(

        responseData.message || 'Failed to fetch requests.'

      );

      throw error;

    }

    const requests = [];

    for (const key in responseData) {

      const request = {

        id: key,

        coachId: coachId,

        userEmail: responseData[key].userEmail,

        message: responseData[key].message

      };

      requests.push(request);

    }

    context.commit('setRequests', requests);

  }

Chapter 265 – Updating The UI Based On Auth State

1. Add isAuthenticated data to return the condition of Login.

export default {

  userId(state) {

    return state.userId;

  },

  token(state) {

    return state.token;

  },

  isAuthenticated(state) {

    return !!state.token;

  }

};

1. In CoachList.vue we want to know now is Log in or not (the status), so we add computed function like below.

 computed: {

    isLoggedIn() {

      return this.$store.getters.isAuthenticated;

    },

    isCoach() {

      return this.$store.getters['coaches/isCoach'];

    },

    filteredCoaches() {

      const coaches = this.$store.getters['coaches/coaches'];

      return coaches.filter((coach) => {

        if (this.activeFilters.frontend && coach.areas.includes('frontend')) {

          return true;

        }

        if (this.activeFilters.backend && coach.areas.includes('backend')) {

          return true;

        }

        if (this.activeFilters.career && coach.areas.includes('career')) {

          return true;

        }

        return false;

      });

    },

    hasCoaches() {

      return !this.isLoading && this.$store.getters['coaches/hasCoaches'];

    },

  },

Add 1 button to login and print it if we are not login. We also add 1 condition on register coach button, it means we need to login to register a new coach.

<base-button link to="/auth" v-if="!isLoggedIn">Login</base-button>

<base-button v-if="isLoggedIn && !isCoach && !isLoading" link to="/register"> Register as Coach

</base-button>

Chapter 266 – Adding A “Logout” Action And Flow

1. Add Logout button in TheHeader.vue

<li v-if="isLoggedIn">

<base-button @click="logout">Logout</base-button>

</li>

1. Add logout function in Auth/actions.js

logout(context) {

    context.commit('setUser', {

      token: null,

      userId: null,

      tokenExpiration: null

    });

  }

Chapter 267 – Authentication And Routing (Include Navigation Guards)

1. After login we can redirect our pages into another pages like below.

Auth/UserAuth.vue

try {

        if (this.mode === 'login') {

          await this.$store.dispatch('login', actionPayload);

        } else {

          await this.$store.dispatch('signup', actionPayload);

        }

        this.$router.replace('/coaches');

      } catch (err) {

        this.error = err.message || 'Failed to authenticate, try later.';

      }

1. When logout, we also can redirect to another link like below.

TheHeader.vue

methods: {

    logout() {

      this.$store.dispatch('logout');

      this.$router.replace('/coaches');

    }

}

1. If we want to send a parameter through the link, we can add ?param=value like below, so after login our link will be directed to ‘/register’

<base-button link to="/auth?redirect=register" v-if="!isLoggedIn">

Login to Register as Coach

</base-button>

In UserAuth.vue, we can catch the status of parameter like below.

try {

if (this.mode === 'login') {

await this.$store.dispatch('login', actionPayload);

} else {

await this.$store.dispatch('signup', actionPayload);

}

const redirectUrl = '/' + (this.$route.query.redirect || 'coaches');

this.$router.replace(redirectUrl);

} catch (err) {

this.error = err.message || 'Failed to authenticate, try later.';

}

1. Add Guard (Middleware) in our apps. We need code below in router.js.

{ path: '/register', component: CoachRegistation, meta: { requiresAuth: true } },

{ path: '/requests', component: RequestsReceived, meta: { requiresAuth: true } },

{ path: '/auth', component: UserAuth, meta: { requiresUnauth: true } },

And we need beforeEach function.

router.beforeEach(function(to, \_, next) {

  if (to.meta.requiresAuth && !store.getters.isAuthenticated) {

    next('/auth');

  } else if (to.meta.requiresUnauth && store.getters.isAuthenticated) {

    next('/coaches');

  } else {

    next();

  }

});

Chapter 268 – Adding “Auto Login”

1. We can modify the login and signup function because the function is same, only the url is different, so the code will be like below.

export default {

  async login(context, payload) {

    return context.dispatch('auth', {

      ...payload,

      mode: 'login'

    });

  },

  async signup(context, payload) {

    return context.dispatch('auth', {

      ...payload,

      mode: 'signup'

    });

  },

  async auth(context, payload) {

    const mode = payload.mode;

    let url = 'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    if (mode === 'signup') {

      url = 'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    }

    const response = await fetch(

      url,

      {

        method: 'POST',

        body: JSON.stringify({

          email: payload.email,

          password: payload.password,

          returnSecureToken: true

        })

      }

    );

    const responseData = await response.json();

    if (!response.ok) {

      console.log(responseData);

      const error = new Error(

        responseData.message || 'Failed to authenticate. Check your login data.'

      );

      throw error;

    }

    console.log(responseData);

    context.commit('setUser', {

      token: responseData.idToken,

      userId: responseData.localId,

      tokenExpiration: responseData.expiresIn

    });

  },

  logout(context) {

    context.commit('setUser', {

      token: null,

      userId: null,

      tokenExpiration: null

    });

  }

};

1. Add auto login function with store the data in local storage. You can check after login, go to application, local storage, and you can see user and token from the user.

// We dont just commit it to Vuex, but also store it

// in the browser storage to make auto login function.

localStorage.setItem('token', responseData.idToken);

localStorage.setItem('userId', responseData.localId);

1. Add Auto Login in auth/actions.js

tryLogin(context) {

    const token = localStorage.getItem('token');

    const userId = localStorage.getItem('userId');

    if (token && userId) {

      context.commit('setUser', {

        token: token,

        userId: userId,

        tokenExpiration: null

      });

    }

  },

1. In App.vue we need to add try login function when we access the apps for the first time like below.

export default {

  components: {

    TheHeader

  },

  created() {

    this.$store.dispatch('tryLogin');

  }

}

Chapter 269 – Addiang “Auto Logout”

1. We need to remove the item in logout.

logout(context) {

    localStorage.removeItem('token');

    localStorage.removeItem('userId');

    context.commit('setUser', {

      token: null,

      userId: null,

      tokenExpiration: null

    });

  }

1. In login function, we need determine the expiration date like below.

async auth(context, payload) {

    const mode = payload.mode;

    let url = 'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    if (mode === 'signup') {

      url = 'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    }

    const response = await fetch(

      url,

      {

        method: 'POST',

        body: JSON.stringify({

          email: payload.email,

          password: payload.password,

          returnSecureToken: true

        })

      }

    );

    const responseData = await response.json();

    if (!response.ok) {

      console.log(responseData);

      const error = new Error(

        responseData.message || 'Failed to authenticate. Check your login data.'

      );

      throw error;

    }

    const expiresIn = responseData.expiresIn \* 1000;

    const expirationDate = new Date().getTime() + expiresIn;

    // We dont just commit it to Vuex, but also store it

    // in the browser storage to make auto login function.

    localStorage.setItem('token', responseData.idToken);

    localStorage.setItem('userId', responseData.localId);

    localStorage.setItem('tokenExpiration', expirationDate);

    // console.log(responseData);

    context.commit('setUser', {

      token: responseData.idToken,

      userId: responseData.localId,

      tokenExpiration: responseData.expirationDate

    });

  },

1. TryLogin function become like below.

tryLogin(context) {

    const token = localStorage.getItem('token');

    const userId = localStorage.getItem('userId');

    if (token && userId) {

      context.commit('setUser', {

        token: token,

        userId: userId

      });

    }

  },

1. Add this in the beginning of auth/actions.js

let timer;

1. In login function, we add this method.

async auth(context, payload) {

    const mode = payload.mode;

    let url = 'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    if (mode === 'signup') {

      url = 'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    }

    const response = await fetch(

      url,

      {

        method: 'POST',

        body: JSON.stringify({

          email: payload.email,

          password: payload.password,

          returnSecureToken: true

        })

      }

    );

    const responseData = await response.json();

    if (!response.ok) {

      console.log(responseData);

      const error = new Error(

        responseData.message || 'Failed to authenticate. Check your login data.'

      );

      throw error;

    }

    const expiresIn = 5000;

    const expirationDate = new Date().getTime() + expiresIn;

    // We dont just commit it to Vuex, but also store it

    // in the browser storage to make auto login function.

    localStorage.setItem('token', responseData.idToken);

    localStorage.setItem('userId', responseData.localId);

    localStorage.setItem('tokenExpiration', expirationDate);

    // console.log(responseData);

    timer = setTimeout(function() {

      context.dispatch('logout');

    }, expiresIn);

    context.commit('setUser', {

      token: responseData.idToken,

      userId: responseData.localId,

      tokenExpiration: responseData.expirationDate

    });

  },

1. In logout function, we add this method.

logout(context) {

    localStorage.removeItem('token');

    localStorage.removeItem('userId');

    localStorage.removeItem('tokenExpiration');

    clearTimeout(timer);

    context.commit('setUser', {

      token: null,

      userId: null

    });

  }

1. In try login function, we need calculate how long the time left for auto login like below.

tryLogin(context) {

    const token = localStorage.getItem('token');

    const userId = localStorage.getItem('userId');

    const tokenExpiration = localStorage.getItem('tokenExpiration');

    const expiresIn = +tokenExpiration - new Date().getTime();

    if(expiresIn < 0) {

      return;

    }

    timer = setTimeout(function() {

      context.dispatch('logout');

    }, expiresIn);

    if (token && userId) {

      context.commit('setUser', {

        token: token,

        userId: userId

      });

    }

  },

1. In auth/actions.js, we add autoLogout function like below.

autoLogout(context) {

    context.dispatch('logout');

    context.commit('setAutoLogout');

}

1. In auth/mutations.js, we add setLogout function.

setAutoLogout(state) {

    state.didAutoLogout = true;

}

1. Set the initial state for didAutoLogout = false in auth/index.js

export default {

  state() {

    return {

      userId: null,

      token: null,

      // tokenExpiration: null,

      didAutoLogout: false

    };

  },

  mutations,

  actions,

  getters

};

1. We need to change on the timer into AutoLogout function.

async auth(context, payload) {

    const mode = payload.mode;

    let url = 'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    if (mode === 'signup') {

      url = 'https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyDYjNbs47lkkbVD9OX1ZSIrsJAhAou1ZDc';

    }

    const response = await fetch(

      url,

      {

        method: 'POST',

        body: JSON.stringify({

          email: payload.email,

          password: payload.password,

          returnSecureToken: true

        })

      }

    );

    const responseData = await response.json();

    if (!response.ok) {

      console.log(responseData);

      const error = new Error(

        responseData.message || 'Failed to authenticate. Check your login data.'

      );

      throw error;

    }

    const expiresIn = responseData.expiresIn \* 1000;

    // const expiresIn =  5000;

    const expirationDate = new Date().getTime() + expiresIn;

    // We dont just commit it to Vuex, but also store it

    // in the browser storage to make auto login function.

    localStorage.setItem('token', responseData.idToken);

    localStorage.setItem('userId', responseData.localId);

    localStorage.setItem('tokenExpiration', expirationDate);

    // console.log(responseData);

    timer = setTimeout(function() {

      context.dispatch('autoLogout');

    }, expiresIn);

    context.commit('setUser', {

      token: responseData.idToken,

      userId: responseData.localId,

      tokenExpiration: responseData.expirationDate

    });

  },

tryLogin(context) {

    const token = localStorage.getItem('token');

    const userId = localStorage.getItem('userId');

    const tokenExpiration = localStorage.getItem('tokenExpiration');

    const expiresIn = +tokenExpiration - new Date().getTime();

    if(expiresIn < 0) {

      return;

    }

    timer = setTimeout(function() {

      context.dispatch('autoLogout');

    }, expiresIn);

    if (token && userId) {

      context.commit('setUser', {

        token: token,

        userId: userId

      });

    }

},

1. In auth/getters.js, we need to add didAutoLogout function.

export default {

  userId(state) {

    return state.userId;

  },

  token(state) {

    return state.token;

  },

  isAuthenticated(state) {

    return !!state.token;

  },

  didAutoLogout(state) {

    return state.didAutoLogout;

  }

};

1. In App.vue, we may add computed and watch for didAutoLogout function.

computed: {

    didAutoLogout() {

      return this.$store.getters.didAutoLogout;

    }

  },

watch: {

    didAutoLogout(curValue, oldValue) {

      if (curValue && curValue !== oldValue) {

        this.$router.replace('/coaches');

      }

    }

  }

1. If you try your AutoLogout has been done, we can make it into false in auth/mutations.js.

setUser(state, payload) {

    state.token = payload.token;

    state.userId = payload.userId;

    // state.tokenExpiration = payload.tokenExpiration;

    state.didAutoLogout = false;

},

And in auth/index.js we can delete the tokenExpiration.

export default {

  state() {

    return {

      userId: null,

      token: null,

      // tokenExpiration: null,

      didAutoLogout: false

    };

  },

  mutations,

  actions,

  getters

};

Last, in auth/action.js switch back the timer out (expiresIn) setting.

const expiresIn = responseData.expiresIn \* 1000;

// const expiresIn =  5000;

Chapter 270 – Summary

1. Use Vuex for different action and for sending the HTTP request
2. Used Vuex to manage the log in and the logout status
3. Log the user in and out automatically when token expired or after a page reload.