Migrating Multi-factor Authentication: FreeIPA WebUI's journey from Dojo to React

FOSDEM 2026

Carla Martínez Poveda, Software Engineer, Red Hat

Agenda

- WebUI's migration context
- The legacy state (Dojo)
- The modern approach (React)
- Conclusion and next steps

WebUI's migration context

FreeIPA's WebUI

- FreeIPA's WebUI → A way to manage FreeIPA functionality via GUI
- From 2022 → Migration and adaptation of the WebUI to a new infrastructure using modern tools
- Why to update it?
  - FOSDEM 24 Talk: Empowering FreeIPA: a dive into the modern WebUI
  - In a nutshell:
    - Legacy code, difficult to maintain
    - Deprecated UI libraries and framework tools
    - Poor accessibility, responsiveness issues, and security
      - This includes: **adapting the existing authentication methods**

FreeIPA's WebUI

- Authentication methods available
  - Username + Password
    - Two-Factor Authentication (OTP)
  - Kerberos Tickets (Single Sign-On)
  - X.509 Client Certificates
- Goals of the new adaptation / The challenge
  - Preserve Multi-factor authentication functionality
  - Modernize implementation following better practices

The legacy state

Before: The “Dojo way”

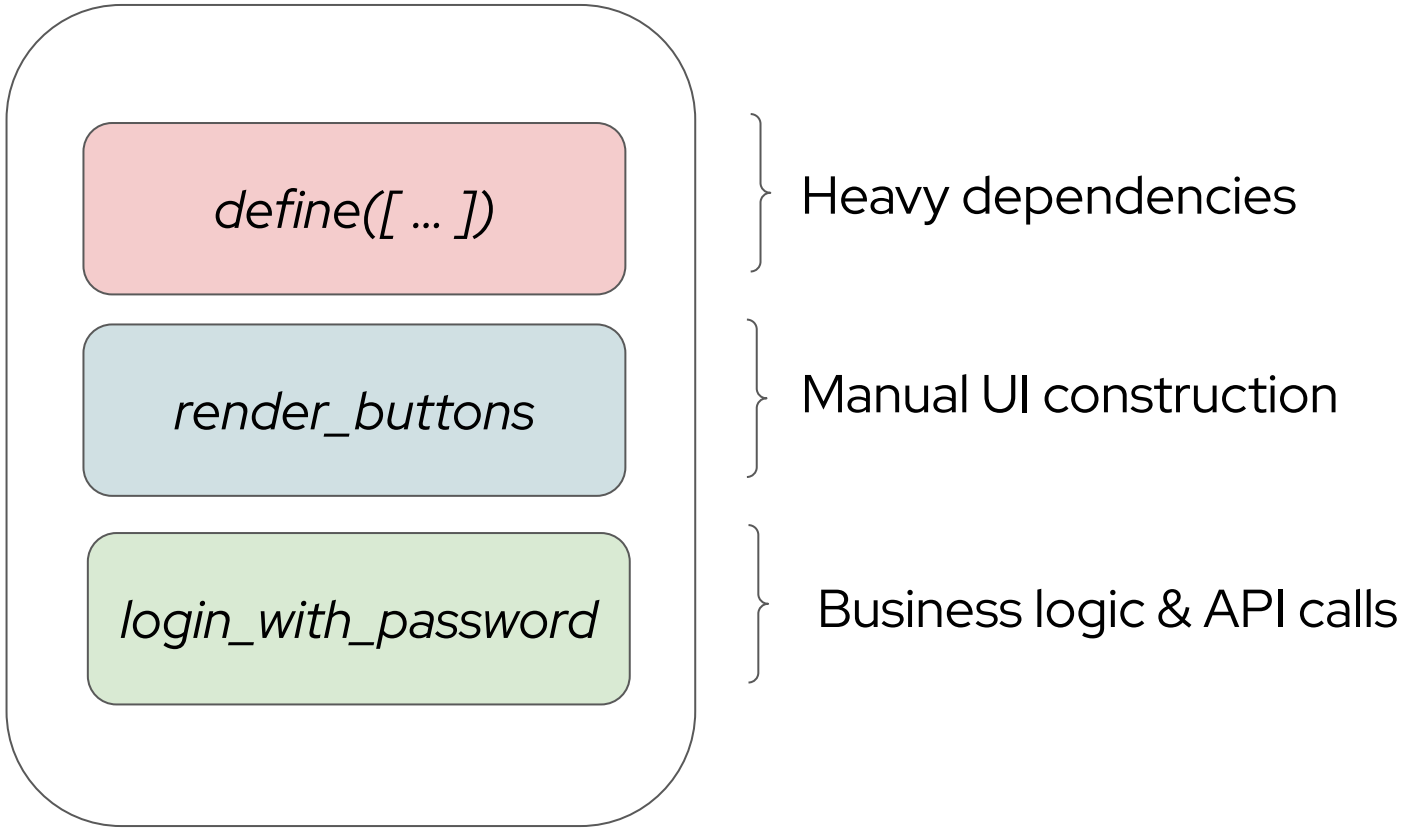
Environment & issues

- JavaScript (Dojo Toolkit) and JQuery
- Monolithic architecture
- Logic is tightly coupled with DOM manipulation
- Imperative methods to create and manage UI elements
- State scattered across widget instances
- Dependency web to load modules too extensive

Before: The “Dojo way”

Example: Monolithic

LoginScreen.js:



Source: install/ui/src/freeipa/widgets/LoginScreen.js

Before: The “Dojo way”

Example: Imperative UI

Old WebUI: Create login button

```
this.login_btn_node = IPA.button({
    name: 'login',
    label: text.get('@i18n:login.login', "Log in"),
    'class': 'btn-primary btn-lg',
    click: this.on_confirm.bind(this)
})[0];
construct.place(this.login_btn_node, container);
construct.place(document.createTextNode(" "), container);
```

- **Imperative code** (“How” instead of “What”)
  - Creates object and places the node
- Necessity of control over the objects and the nodes

“Cut the meat. Light the fire. Put the meat. Cook it. Wait 5 minutes. Put it on the bread. Serve it” (imperative) vs *“I want a hamburger”* (declarative)

Source: install/ui/src/freeipa/widgets/LoginScreen.js

Before: The "Dojo way"

Example: Hard dependencies

```
define(['dojo/_base/declare',
        'dojo/Deferred',
        'dojo/dom-construct',
        'dojo/dom-style',
        'dojo/query',
        'dojo/on',
        'dojo/topic',
        '../ipa',
        '../auth',
        '../config',
        '../reg',
        '../FieldBinder',
        '../text',
        '../util',
        './LoginScreenBase'
       ],
       function(declare, Deferred, construct, dom_style, query, on, topic,
                IPA, auth, config, reg, FieldBinder, text, util,
                LoginScreenBase) {
```

- **Dependency web** → Difficult to maintain and test
- Monolithic load → Browser needs to download and analyze all the modules before showing some UI element
- Fragile → If any dependency fails, the login page won't be shown

"I need ALL these first to make things work" (hard dependency) vs *"I just need X and Y"* (soft dependency)

Source: install/ui/src/freeipa/widgets/LoginScreen.js

The modern approach

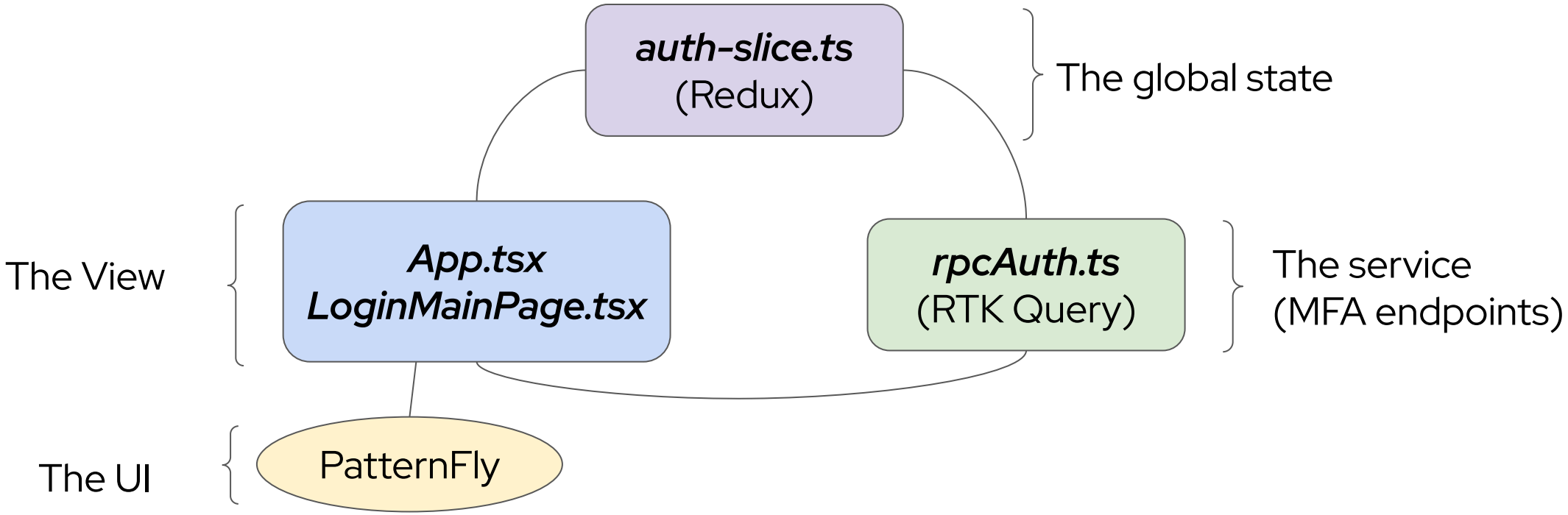
Now: The “React way”

Environment & features

- React + Vite + PatternFly 6
- Declarative UI
- Redux for unified login state management
- API commands logic powered by hooks (RTK Query)
- Strict type safety (TypeScript)

Now: The “React way”

Separations of concerns



Now: The “React way”

Declarative UI

Before: Dojo (Imperative)

```
this.login_btn_node = IPA.button({
    name: 'login',
    label: text.get('@i18n:login.login', "Log in"),
    'class': 'btn-primary btn-lg',
    click: this.on_confirm.bind(this)
})[0];
construct.place(this.login_btn_node, container);
construct.place(document.createTextNode(" "), container);
```

VS

Now: React (Declarative)

```
const loginForm = (
  <LoginForm
    usernameLabel="Username"
    usernameValue={username}
    // We pass HANDLERS, not logic
    onChangeUsername={handleUsernameChange}
    passwordLabel="Password"
    passwordValue={password}
    onChangePassword={handlePasswordChange}
    onLoginButtonClick={onLoginButtonClick}
    loginButtonLabel="Log in"
    // UI State is controlled via props
    isLoginButtonDisabled={authenticating}
  />
);
```

- **Declarative way** → Instead of building the login form, we describe it
- Powered by PatternFly’s components
- Props and handlers vs direct DOM manipulation

Source: freeipa-webui/src/login/LoginMainPage.tsx
https://www.patternfly.org/components/login-page/#loginform
https://www.patternfly.org/components/login-page/#loginpage

Now: The “React way”

Unified global state management

```ts
const initialState: AuthState = {
  isUserLoggedIn: false,
  user: null,
  error: null,
};
```

```ts
const authSlice = createSlice({
  name: "auth",
  initialState,
  reducers: {
    setIsLogin: (state, action: PayloadAction<onLoginPayload>) => {
      state.isUserLoggedIn = true;
      state.user = action.payload.loggedInUser;
      state.error = action.payload.error;
    },
    setIsLogout: (state) => {
      state.isUserLoggedIn = false;
      state.user = null;
      state.error = null;
    },
  },
});

export const { setIsLogin, setIsLogout } = authSlice.actions;
export default authSlice.reducer;
```

- Use of Redux to set **global login state**
  - Tracks: *login status* + *uid* + *error* (if any)
- Helpful to manage **current session** and **routes** (*AppRoutes.tsx*)
- Used in two scenarios:
  - When application initializes (*App.tsx*)
  - Via login mechanism (*LoginMainPage.tsx*)

Source: freeipa-webui/src/store/Global/auth-slice.ts

Now: The “React way”

Communication layer and Hooks

rpcAuth.ts:

```
// 1. Define the Endpoint once
userPasswordLogin: build.mutation<FindRPCResponse, UserPasswordPayload>({
    query: (payload) => ({
        url: "/ipa/session/login_password",
        method: "POST",
        // Type-safe payload access
        body: `user=${payload.username}&password=${payload.password}`,
        headers: { "Content-Type": "application/x-www-form-urlencoded" }
    }),
}),
```

LoginMainPage.tsx:

```
// 1. The Hook
const [onUserPwdLogin] = useUserPasswordLoginMutation();

// 2. The Handler
const onLoginButtonClick = (event) => {
  event.preventDefault();
  setAuthenticating(true);

  onUserPwdLogin({ username, password }).then((response) => {
    if ("error" in response) {
      // Handle Error
    } else {
      // 3. Success: Update Global State
      dispatch(setIsLogin({ loggedInUser: username, error: null }));
      window.location.reload();
    }
    setAuthenticating(false);
  });
};
```

For all authentication methods

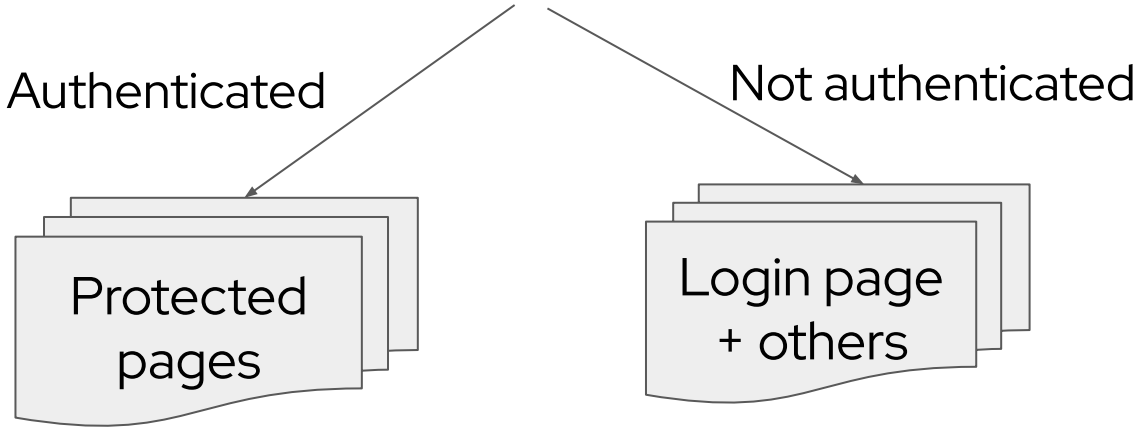
Source: install/ui/src/freeipa/widgets/LoginScreen.js
freeipa-webui/src/service/rpcAuth.ts
freeipa-webui/src/login/LoginMainPage.tsx

Now: The “React way”

Initialization & Session Check

- Modern WebUI loads → **Initial handshake**
  - API calls: *config_show*, *whoami*, *env*, etc.
- If valid session exists → Assigns login state data
  - *setIsLogin()*
- If no valid session → *setIsLogout()*

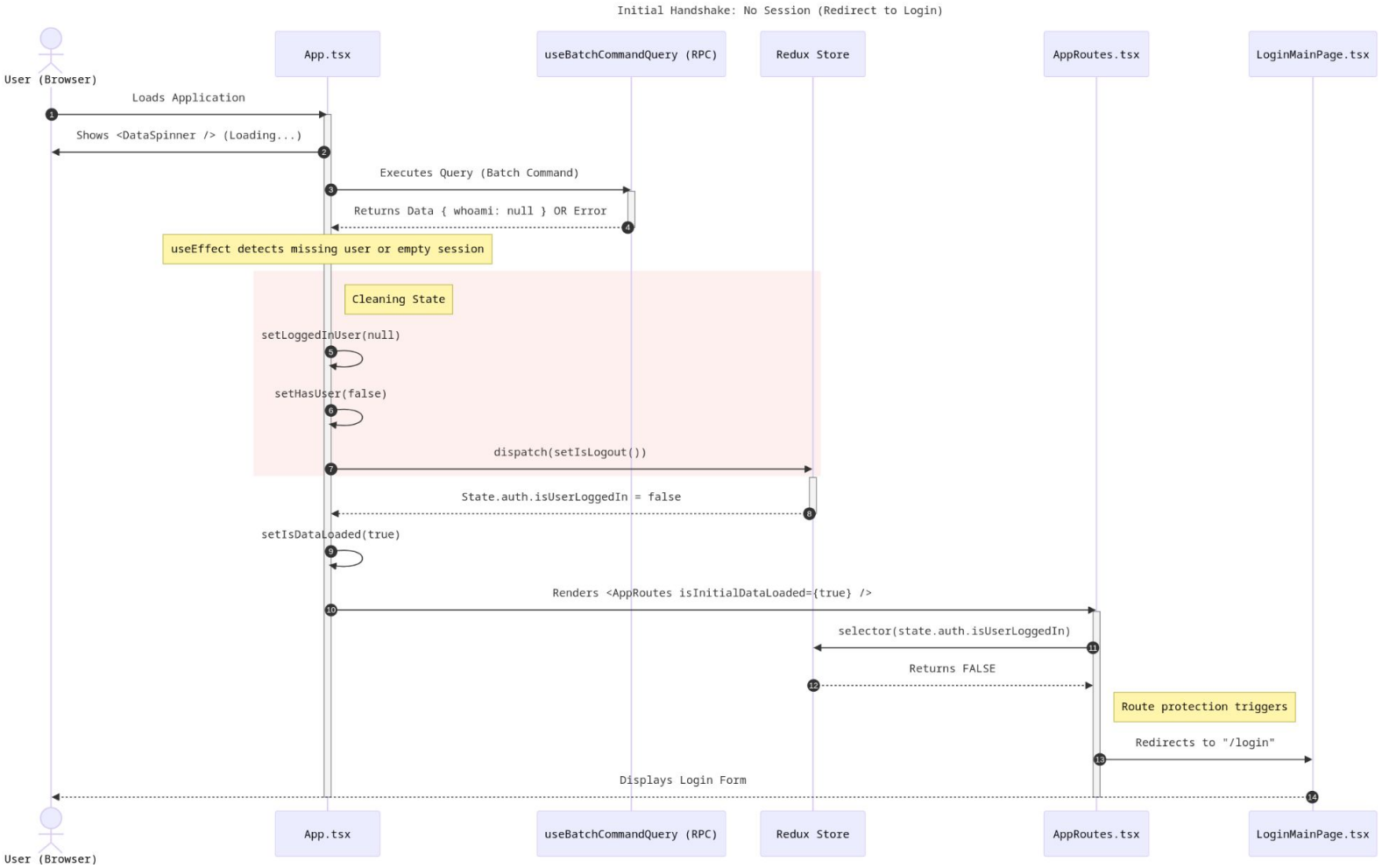
- Acts as the traffic controller
- Based on Redux *auth.**isUserLoggedIn*** state



Source: freeipa-webui/src/App.tsx
freeipa-webui/navigation/AppRoutes.tsx

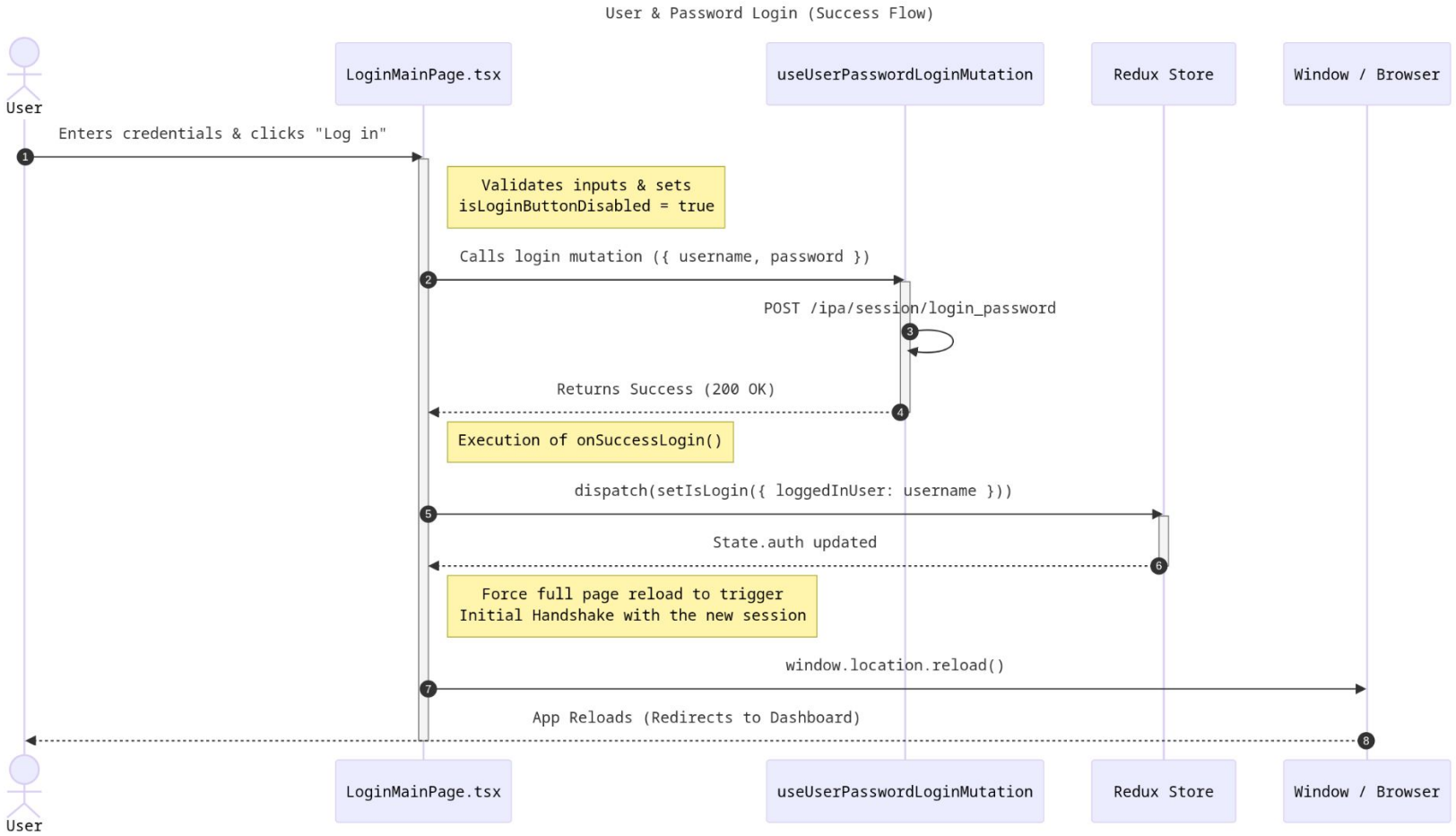
Now: The “React way”

Initialization & Session Check



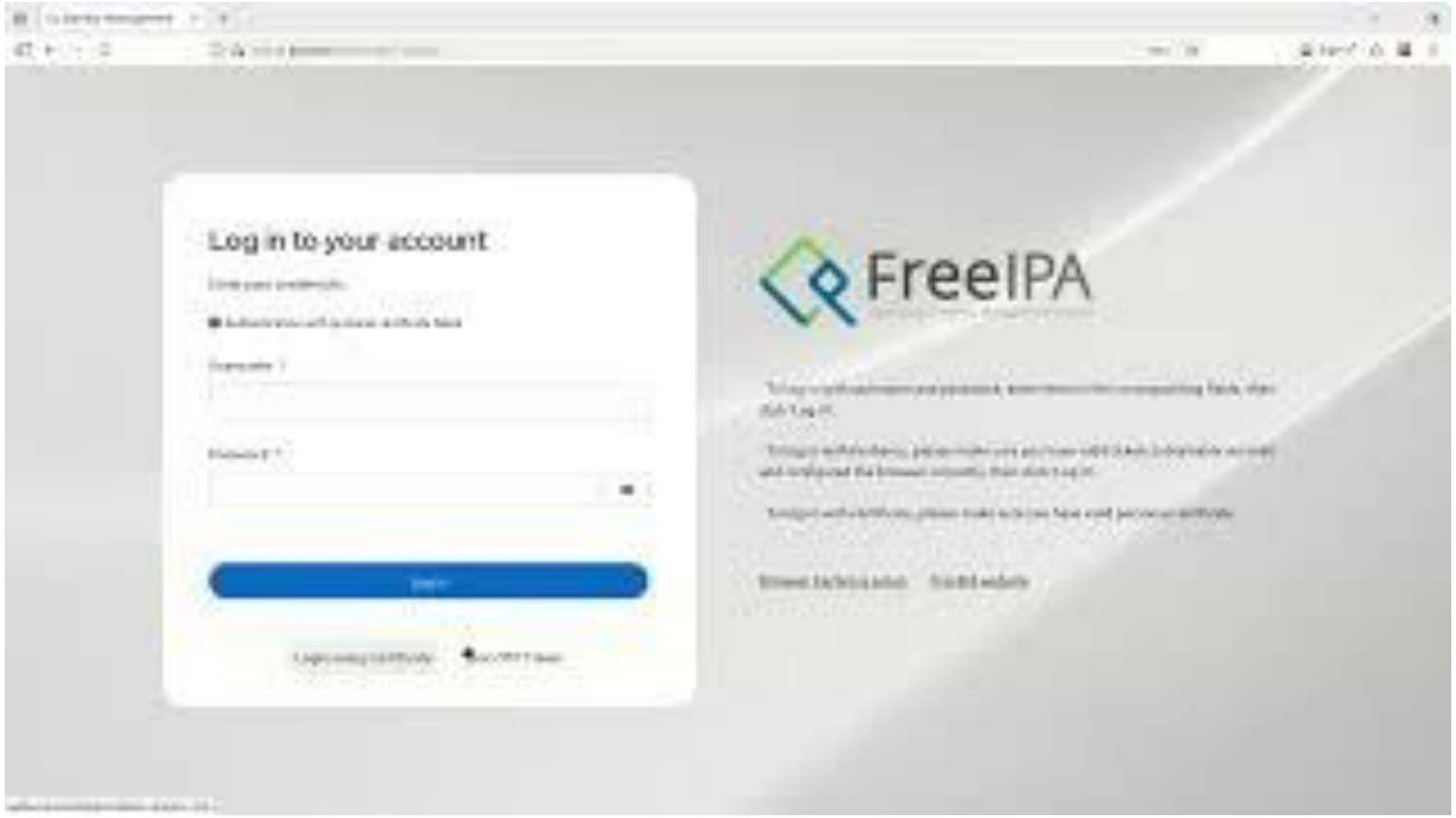
Now: The “React way”

Successful authentication via user + password



Now: The “React way”

Demo



Source: https://youtu.be/Un8BHdz7Z4Y

Conclusions

Summary of the architecture shift

Feature	Legacy (Dojo)	Modern (React)
UI construction	Imperative (*construct.place*)	Declarative (*TSX*)
Global state	Distributed in Widgets	Unified in Redux
Backend logic	Mixed with view	Encapsulated in Hooks
Safety	Implicit (*JavaScript*)	Strict (*TypeScript*)
Bundle	AMD (Asynchronous Module Definition) loader	Vite
Testing	Selenium (*Python*)	Cypress + Gherkin (*TS*)

Authentication - Next steps

- Optimization of some inherited anti-patterns (design choice)
- Web UI login with passwordless methods (OAuth2)
- Enhancement of the Cypress tests
- Internationalization (i18n)

Resources

- FreeIPA modern WebUI - GitHub project
- Design doc: Authentication via Kerberos
- Design doc: Authentication via certificates
- FOSDEM 2024 Talk: Empowering FreeIPA: a dive into the modern WebUI

Q & A