Script

(Duration: 47 minutes)

Part 1: Set up Nuxt boilerplate [3 minutes]

A) Run:

```
$ npx nuxi init demo
$ cd demo
$ yarn add vue@3 vuex@4 @vue/reactivity @vue/shared postcss
```

B) Inspect code:

```
$ code .
```

C) Start dev server:

```
$ yarn dev -o
```

Part 2: Integrate Element Plus [4 minutes]

A) Run:

```
$ yarn add element-plus@1.1.0-beta.20 --exact
```

B) Add `plugins/element-plus.client.ts`:

```
import { defineNuxtPlugin } from '#app'
import ElementPlus from 'element-plus'
import 'element-plus/theme-chalk/index.css'

export default defineNuxtPlugin(ctx => {
  ctx.vueApp.use(ElementPlus)
})
```

C) Edit `nuxt.config.ts` to disable Server Side Rendering, as Element Plus is not yet compatible with it:

```
export default defineNuxtConfig({
  ssr: false
})
```

D) Edit `app.vue` to add:

```
<el-button type="primary">Hello</el-button>
```

Part 3: Create UI skeleton [22 minutes]

A) Edit `app.vue` to add:

Markup:

```
<template>
  <div>
    <div v-if="status === 'disconnected'">
      <h2>Disconnected</h2>
      <p>Reason: {{ error.message }}</p>
      <el-button type="primary" @click="connect">Connect</el-button>
    </div>
    <div v-else v-loading="status === 'connecting'" element-loading-text="Connecting…">
      <el-table :data="metrics" stripe>
        <el-table-column prop="label" label="Label" width="180" />
        <el-table-column prop="value" label="Value" />
      </el-table>
      <el-divider />
      <el-button type="danger" @click="attack">Attack</el-button>
    </div>
  </div>
</template>
```

Logic:

```
<script lang="ts">
import { defineComponent } from 'vue'
import { mapState, mapActions } from 'vuex'
import { useStore } from './store'

export default defineComponent({
  name: 'App',
  computed: mapState([
```

```
    'status',
    'metrics',
    'error',
  ]),
  methods: mapActions([
    'connect',
    'attack',
  ]),
  setup() {
    const store = useStore()
    store.dispatch('connect')
  }
})
</script>
```

B) Add `store.ts` with mocked guts:

```
import { InjectionKey } from 'vue'
import {
  Store,
  useStore as baseUseStore,
  createStore,
} from 'vuex'

export const key: InjectionKey<Store<State>> = Symbol()

export function useStore() {
  return baseUseStore(key)
}

export interface State {
  status: 'disconnected' | 'connecting' | 'connected'
  metrics: Metric[]
  error: Error | null
}

export interface Metric {
  label: string
  value: string
}

const SET_DISCONNECTED = 'Disconnected'
const SET_CONNECTING = 'Connecting'
const SET_CONNECTED = 'Connected'
const SET_METRICS = 'Set Metrics'

export const store = createStore<State>({
  state: {
    status: 'disconnected',
    metrics: [],
    error: null,
  },
  actions: {
    async connect({ state, commit }): Promise<void> {
      commit(SET_CONNECTING)
```

```
39
40      await sleep(1000)
41      commit(SET_CONNECTED)
42
43      await sleep(1000)
44      commit(SET_METRICS, {
45        metrics: [
46          {
47            label: 'Apples',
48            value: 5
49          },
50          {
51            label: 'Bananas',
52            value: 30
53          },
54        ]
55      })
56    },
57    async attack({ state, commit }): Promise<void> {
58      const metrics = state.metrics
59        .map(({ label, value }) => {
60          return {
61            label,
62            value: (parseInt(value) - 1).toString()
63          }
64        })
65        .filter(({ value }) => parseInt(value) > 0)
66      commit(SET_METRICS, { metrics })
67
68      if (metrics.length < 2) {
69        await sleep(2000)
70        commit(SET_DISCONNECTED, { error: new Error('connection closed') })
71      }
72    }
73  },
74  mutations: {
75    [SET_DISCONNECTED](state, { error }) {
76      state.status = 'disconnected'
77      state.metrics = []
78      state.error = error
79    },
80    [SET_CONNECTING](state) {
81      state.status = 'connecting'
82      state.error = null
83    },
84    [SET_CONNECTED](state) {
85      state.status = 'connected'
86    },
87    [SET_METRICS](state, { metrics }) {
88      state.metrics = metrics
89    },
90  }
91})
92
93function sleep(duration: number): Promise<void> {
94  return new Promise(resolve => {
```

```
    setTimeout(() => { resolve() }, duration)
  })
}

declare module '@vue/runtime-core' {
  interface ComponentCustomProperties {
    $store: Store<State>
  }
}
```

C) Add `plugins/vuex.client.ts`:

```
import { defineNuxtPlugin } from '#app'
import { store, key } from '@/store'

export default defineNuxtPlugin(ctx => {
  ctx.vueApp.use(store, key)
})
```

Let's take it for a spin! ...Hmm, that doesn't look too great, so let's:

D) Edit `app.vue` to add styling:

Add CSS class to root div:

```
 <template>
-  <div>
+  <div class="app-content">
     <div v-if="status === 'disconnected'">
       <h2>Disconnected</h2>
       <p>Reason: {{ error.message }}</p>
```

Wrap root div in a new root div so we can center it relative to it:

```
<template>
  <div class="app-container">
    <div class="app-content">
```

Add CSS class to the `disconnected` view:

```
 <template>
   <div class="app-container">
     <div class="app-content">
-      <div v-if="status === 'disconnected'">
+      <div v-if="status === 'disconnected'" class="disconnected-view">
         <h2>Disconnected</h2>
         <p>Reason: {{ error.message }}</p>
```

```
    <el-button type="primary" @click="connect">Connect</el-button>
```

Add CSS rules:

```
<style scoped>
.app-container {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
}

.app-content {
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 25px;
  box-shadow: var(--el-box-shadow-base);
}

.disconnected-view h2 {
  margin: 5px 250px 15px 5px;
  font-weight: bold;
}

.disconnected-view p {
  margin: 0 5px 40px 5px;
}

.disconnected-view button {
  float: right;
}
</style>
```

Looks better now! Time to make this app actually do something useful:

Part 4: Implement instrumentation [18 minutes]

A) Run:

```
$ yarn add @frida/web-client @frida/web-shims
```

B) Add `vite.config.ts` to insert web shims:

```
import { defineConfig } from 'vite'
import shims from '@frida/web-shims'

export default defineConfig({
```

```
  resolve: {
    alias: shims(import.meta.url)
  }
})
```

C) Edit store.ts to wire up the control logic:

Starting with the client library and Frida agent to be injected:

```
  useStore as baseUseStore,
  createStore,
} from 'vuex'
+import * as frida from '@frida/web-client'
+import agent from 'assets/agent.js?raw'
```

We'll also need some extra state:

```
export interface State {
  status: 'disconnected' | 'connecting' | 'connected'
+  client: frida.Client
+  script: frida.Script | null
  metrics: Metric[]
  error: Error | null
}
```

That we need to initialize:

```
export const store = createStore<State>({
  state: {
    status: 'disconnected',
+    client: new frida.Client('127.0.0.1:27042'),
+    script: null,
    metrics: [],
    error: null,
  },
```

Let's replace the meat of the connect() implementation after SET_CONNECTING with:

```
    try {
      let quake: frida.Process | undefined
      do {
        const processes = await state.client.enumerateProcesses()

        quake = processes.find(({ name }) => name === 'QuakeSpasm')
        if (quake === undefined) {
          await sleep(1000)
          continue
        }
      } while (quake === undefined)
```

```
12
13      const session = await state.client.attach(quake.pid)
14      session.detached.connect(reason => {
15        if (state.status === 'connected') {
16          let message = ''
17          for (let c of frida.SessionDetachReason[reason]) {
18            if (message.length > 0 && c === c.toUpperCase()) {
19              message += ' '
20            }
21            message += c.toLowerCase()
22          }
23          commit(SET_DISCONNECTED, { error: new Error(message) })
24        }
25      })
26
27      const script = await session.createScript(agent)
28      script.message.connect(message => {
29        let handled = false
30
31        if (message.type === frida.MessageType.Send) {
32          const { payload } = message
33
34          if (payload.type === 'metrics') {
35            commit(SET_METRICS, { metrics: payload.metrics })
36            handled = true
37          }
38        }
39
40        if (!handled) {
41          console.warn('Unhandled message:', message)
42        }
43      })
44      await script.load()
45
46      commit(SET_CONNECTED, { script })
47    } catch (error) {
48      commit(SET_DISCONNECTED, { error })
49    }
```

And replace the stub code in attack() with:

```
1 state.script!.exports.attack()
```

We'll also need to update the mutations to handle the new state:

```
1    mutations: {
2      [SET_DISCONNECTED](state, { error }) {
3        state.status = 'disconnected'
4  +     state.script = null
5        state.metrics = []
6        state.error = error
7      },
8  @@ -90,8 +116,9 @@ export const store = createStore<State>({
9        state.status = 'connecting'
```

```
    state.error = null
  },
-   [SET_CONNECTED](state) {
+   [SET_CONNECTED](state, { script }) {
    state.status = 'connected'
+     state.script = script
  },
  [SET_METRICS](state, { metrics }) {
    state.metrics = metrics
```

D) Add `assets/agent.js` and `assets/agent.d.ts`:

Starting with the easy one, `assets/agent.d.ts`:

```
declare module 'assets/agent.js?raw' {
  const content: string
  export default content
}
```

Now we're ready to implement the agent. Let's start with attack():

```
rpc.exports = {
  async attack() {
    await perform(() => attackDown())
    await sleep(50)
    await perform(() => attackUp())
  }
}
```

And import `attackDown()` + `attackUp()` from Quake's executable:

```
const attackDown = new NativeFunction(importSymbol('IN_AttackDown'), 'void', [])
const attackUp = new NativeFunction(importSymbol('IN_AttackUp'), 'void', [])

...

function importSymbol(name) {
  return Module.getExportByName('QuakeSpasm-SDL2', name)
}
```

We'll also need `sleep()`:

```
function sleep(duration) {
  return new Promise(resolve => {
    setTimeout(() => { resolve() }, duration)
  })
}
```

And to make our function calls happen on the UI thread instead of Frida's JS thread, we'll need to implement perform():

```
const pendingUIThreadOps = []

...

function perform(f) {
  return new Promise((resolve, reject) => {
    pendingUIThreadOps.push(() => {
      try {
        const result = f()
        resolve(result)
      } catch (e) {
        reject(e)
      }
    })
  })
}

Interceptor.attach(importSymbol('IN_SendKeyEvents'), {
  onEnter() {
    while (pendingUIThreadOps.length > 0) {
      const f = pendingUIThreadOps.shift()
      f()
    }
  }
})
```

But wait, we also want to show some metrics in our UI. Let's do this after rpc.exports has been assigned:

```
pushMetrics()
setInterval(pushMetrics, 1000)
```

And right after attack(), stats may have changed, so let's also push the metrics at that point as well:

```
  async attack() {
    ...

    await pushMetrics()
```

Ok, we'll need another import:

```
const clientState = importSymbol('cl')
```

This is the struct in which Quake stores its (you guessed it) client state.

We'll also need a couple of constants:

```
const METRIC_HEALTH = 0
const METRIC_SHELLS = 6
```

These represent the indices into an array of game stats.

So going top down, right after the setInterval():

```
async function pushMetrics() {
  const metrics = await readMetrics()
  send({
    type: 'metrics',
    metrics
  })
}

function readMetrics() {
  return perform(() => {
    return [
      {
        label: 'Health',
        value: readMetric(METRIC_HEALTH)
      },
      {
        label: 'Shells',
        value: readMetric(METRIC_SHELLS)
      },
    ]
  })
}

function readMetric(metric) {
  return clientState.add(28 + metric * 4).readInt()
}
```

Let's take it for a spin!

MorphWiz Demo

Start the Portal:

```
$ cd ~/Downloads
$ ./frida-portal-15.1.6-macos-arm64 --cluster-endpoint=172.20.10.5 --control-endpoint=172.20.10.5
```

Connect iPhone through USB, start the MorphWiz app, and run:

```
$ frida-join -U MorphWiz 172.20.10.5
```

Get somebody in the audience to join the hotspot and navigate to http://172.20.10.5:3000