CIS 658 Web Architectures

App Navigation

Navigate Among Multiple Views

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Last time

Core Concepts of Pinia:

- **Actions** – Define functions to change state and perform async tasks.
- **Getters** – Compute derived state values efficiently.
- **$patch()** – Update multiple state properties at once.
- Subscribe
  - **$onAction** – Listen and react to action calls (e.g., logging or debugging).
  - **$subscribe** – Watch for state changes for persistence or side effects.

Exercise from the previous class

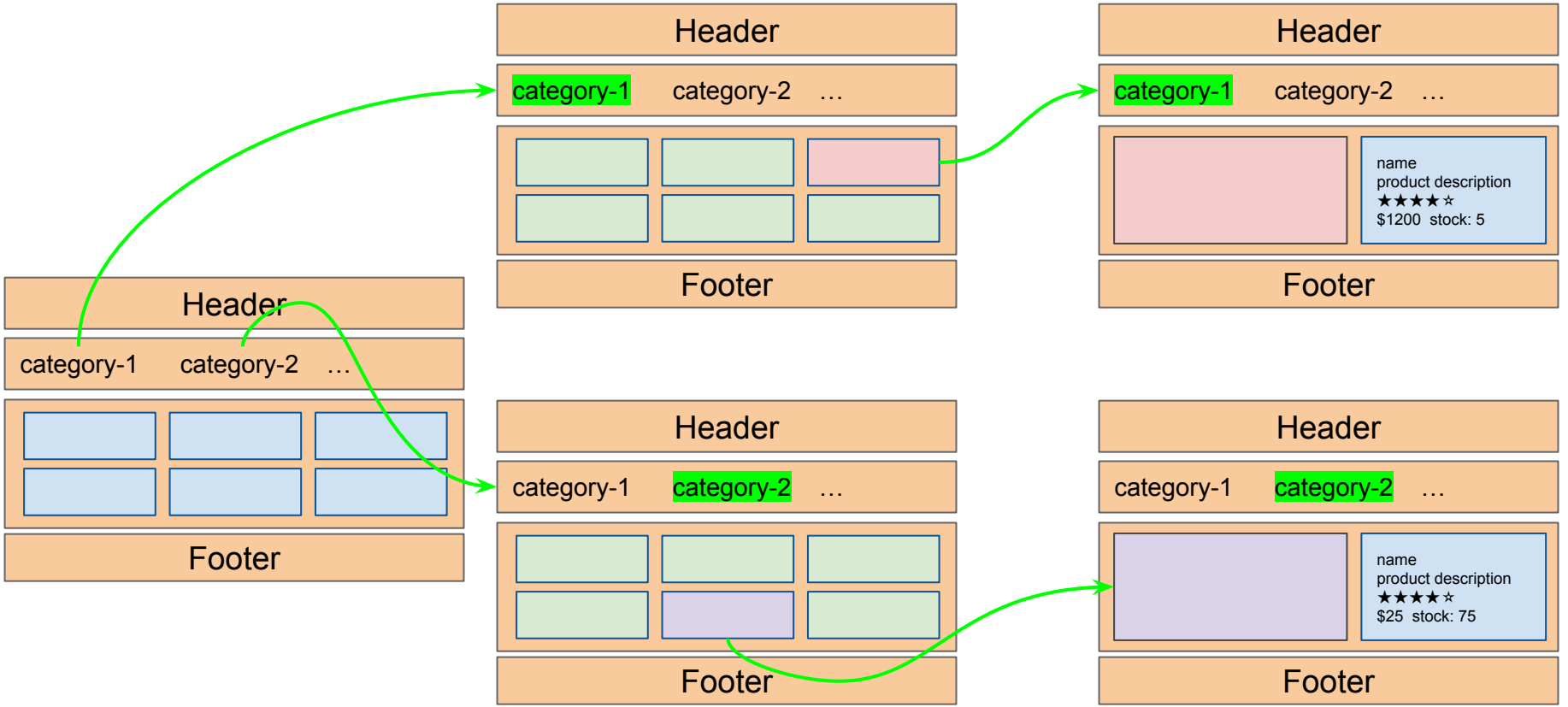
Complete the store-app implementation with the following functionalities:

- Implement a CartStore using Pinia to temporarily store the user's selected items for checkout.
- Utilize the $patch method to add items into cart.
- Displays all items currently in the cart.

Answer

store-app

Routing Web App Navigation



Traditional Web Page Solution

home.html

```
<h1>Local Weather</h1>
<a href="forecast.html">Forecast</a>
<a href="settings.html">Weather Settings</a>
```

```
<h1>Forecast</h1>
<div>
  <!-- content -->
  <a href="home.html">Home</a>
</div>
```

forecast.html

```
<h1>Weather Settings</h1>
<div>
  <!-- content -->
  <a href="home.html">Home</a>
</div>
```

settings.html

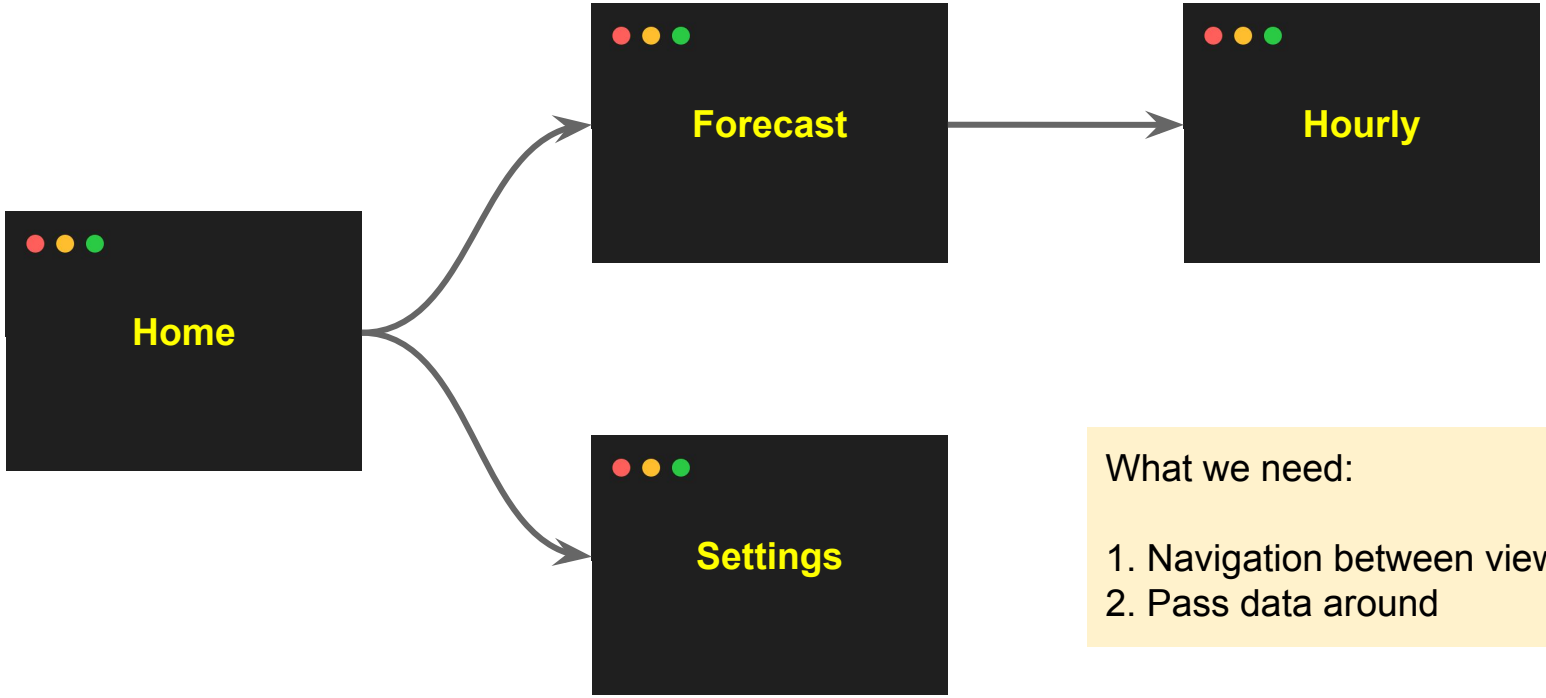
Legacy Web Pages vs. Modern Web Apps

	Legacy Web Sites	Modern Web Apps
Structure	One .html file per page	Single Page Application (SPAs)
Contents Organization	Multiple pages	Multiple web components (“views” / “screens”)
Transitions	Replace the entire page with new .html from the server	Replace only part of the page with new component

Commonality: The browser features for maintaining navigation history

- Back / Forward buttons
- History Stack
- Each page/component is associated with a unique URL path

Weather App



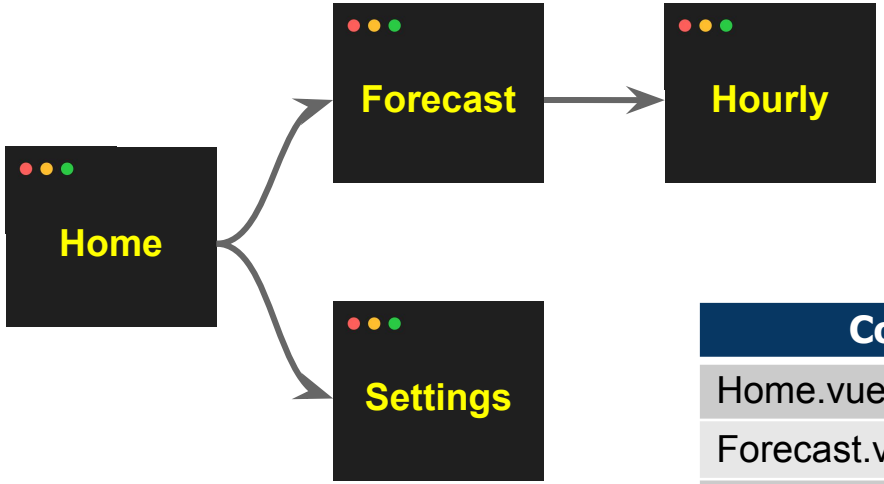
What we need:

1. Navigation between views
2. Pass data around

Vue Router

https://router.vuejs.org

Step 1: Installation & Setup




```
npm install --save vue-router@4
# OR
yarn add vue-router@4
```

Component	URL (internal) Path
Home.vue (landing page)	/
Forecast.vue	/forecast
Hourly.vue	/hourly
Settings.vue	/settings

Online Doc

Step 2: Define Routes (in an array)

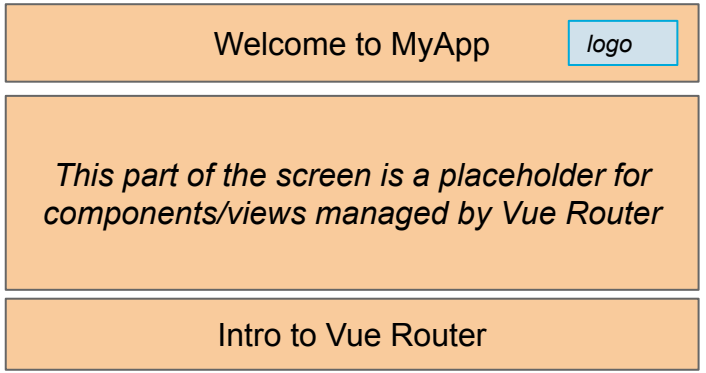
```
import { createApp } from "vue";
import { createRouter, createWebHashHistory } from "vue-router";
import "./style.css";
import App from "./App.vue";
import Home from "./components/Home.vue";
import Forecast from "./components/Forecast.vue";
import Settings from "./components/Settings.vue";
import Hourly from "./components/Hourly.vue";

// Step 2A: Define routes and create router instance
const routes = [
  { path: "/", component: Home },
  { path: "/forecast", component: Forecast },
  { path: "/settings", component: Settings },
  { path: "/hourly", component: Hourly },
];
const router = createRouter({
  history: createWebHashHistory(),
  routes,
});

// Step 2B: Use the router instance in the Vue app
createApp(App).use(router).mount("#app");
```

main.ts

Step 3: Include "View Container" in App.vue

App.vue

```
<template>
  <heaader>Welcome to MyApp</heaader>
  <main>
    <router-view />
  </main>
  <footer>Intro to Vue Router</footer>
</template>
```



Step 4: Use Links in Components

Home.vue

```
<template>
  <h1>Home</h1>
  <a href="#">For your comparison</a> |
  <router-link to="/forecast">Forecast</router-link> |
  <router-link to="/settings">Settings</router-link>
</template>
```

Forecast.vue

```
<template>
  <h1>Forecast</h1>
  <router-link to="/hourly">Hourly</router-link>
</template>
```

Settings.vue

```
<template>
  <h1>Settings</h1>
</template>
```

How To Transition Programmatically

Use Case: Hourly Forecast only for Prime Members

Forecast.vue (before)

```
<template>
  <h1>Forecast</h1>
  <router-link to="/hourly">Hourly</router-link>
</template>
```

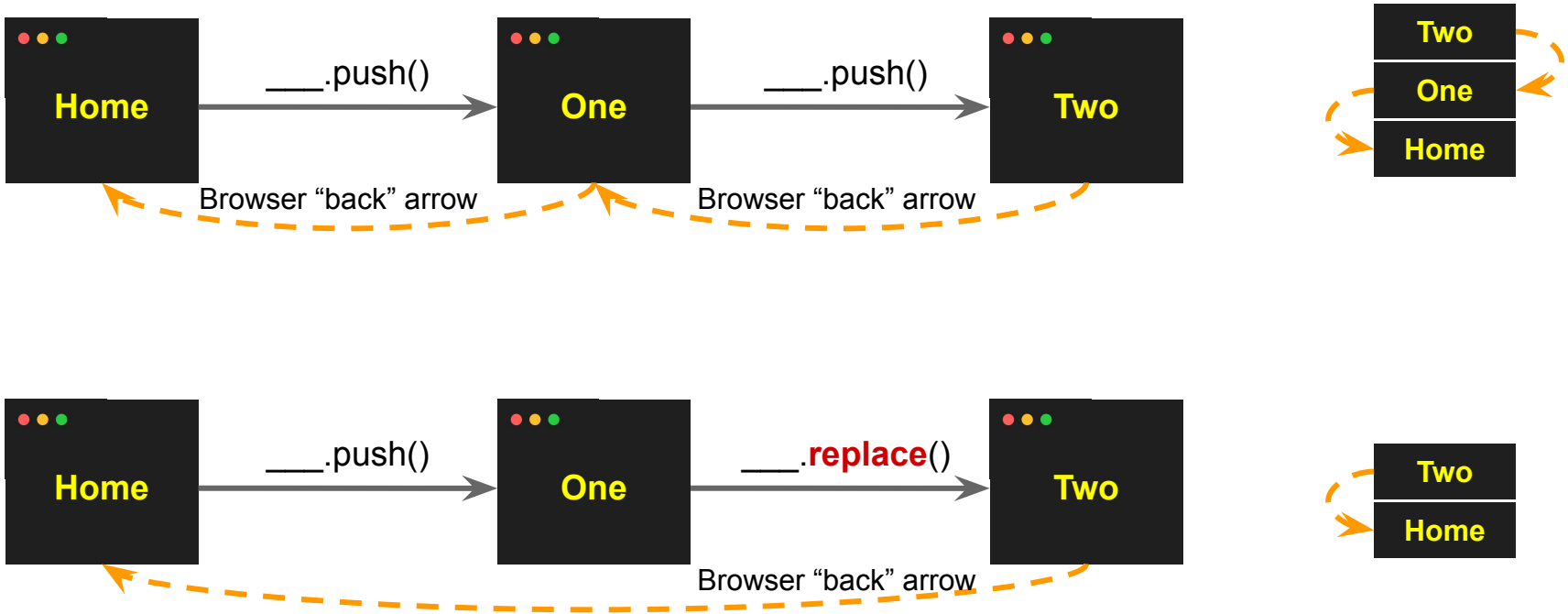
Forecast.vue (after)

```
<template>
  <h1>Forecast</h1>
  <button @click="canIHourly">Prime hourly</button>
</template>
<script setup lang="ts">
import { useRouter } from "vue-router";
const appNav = useRouter();
function canIHourly() {
  if (prime_membership_is_confirmed) {
    appNav.push({ path: "/hourly" });
  }
}
</script>
```

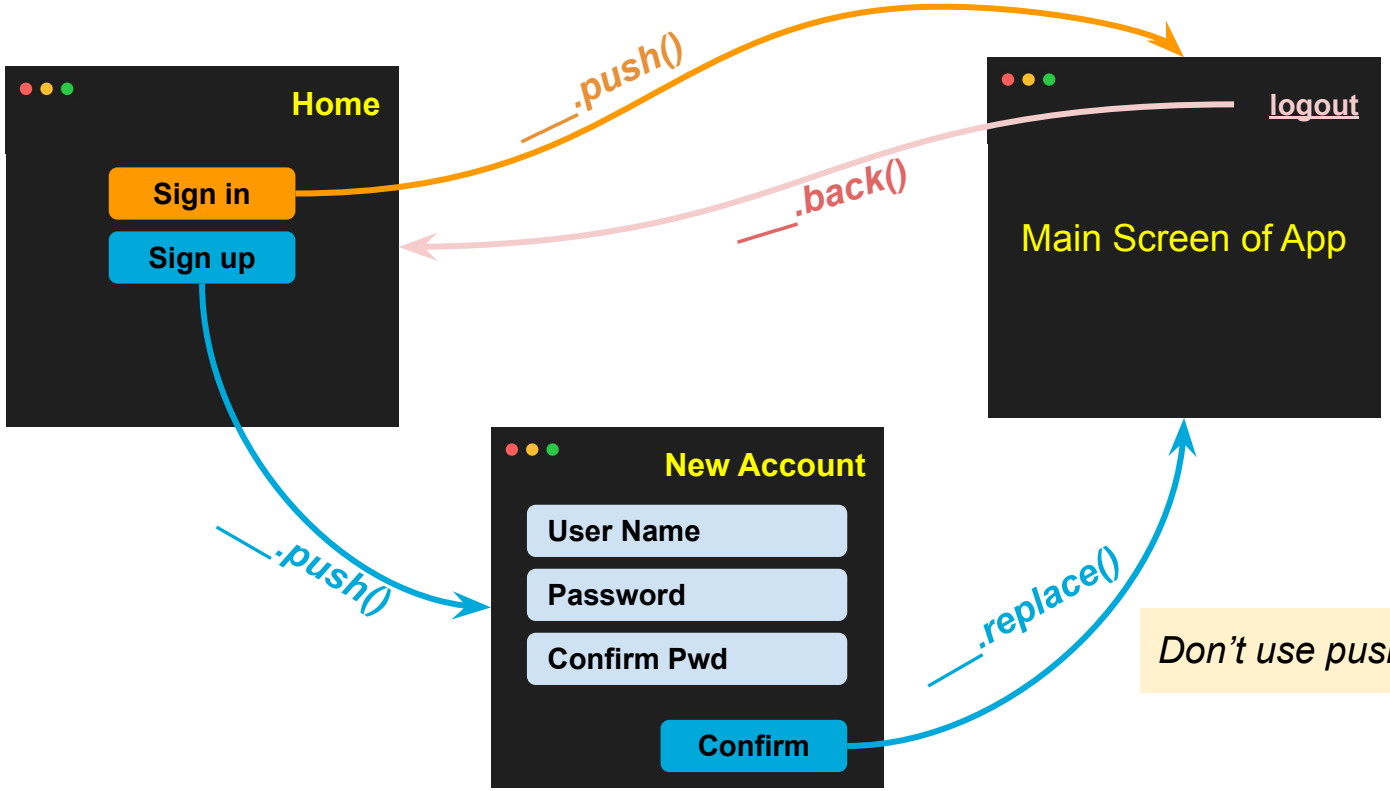
Vue Router Navigation Functions

Function	By Name	Explanation
push()	_____.push({path: “/fooPath”});	Adds an entry to the browser's history and navigates to a different route ‘/fooPath’.
replace()	_____.replace({path: “/fooPath”});	Replaces the current route. This means that pressing the browser's back button won't take you to the previous page but to the one before that. In this example, it replaces the current route with /fooPath
back()	_____.back()	Equivalent to the user clicking the browser's back button. It moves one step backward in the browser's history.
forward()	_____.forward()	Equivalent to the user clicking the browser's forward button. It moves one step forward in the browser's history.
go()	_____.go(-2)	Same as calling $router.back() twice
	_____.go(1)	Same as calling $router.forward()

Browser History Stack: push() vs. replace()



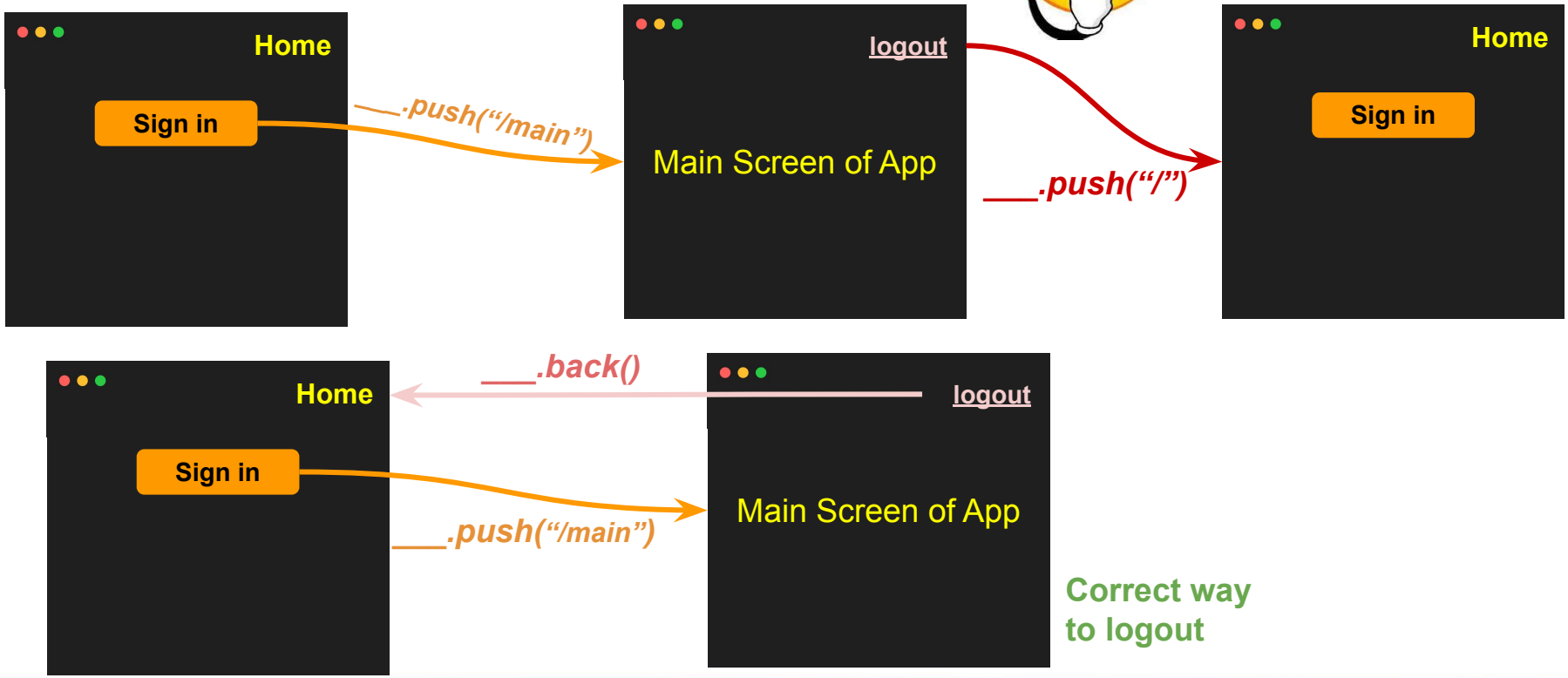
App Screen Flow



Don't use push() here!!

App Screen Flow

Wrong way to logout

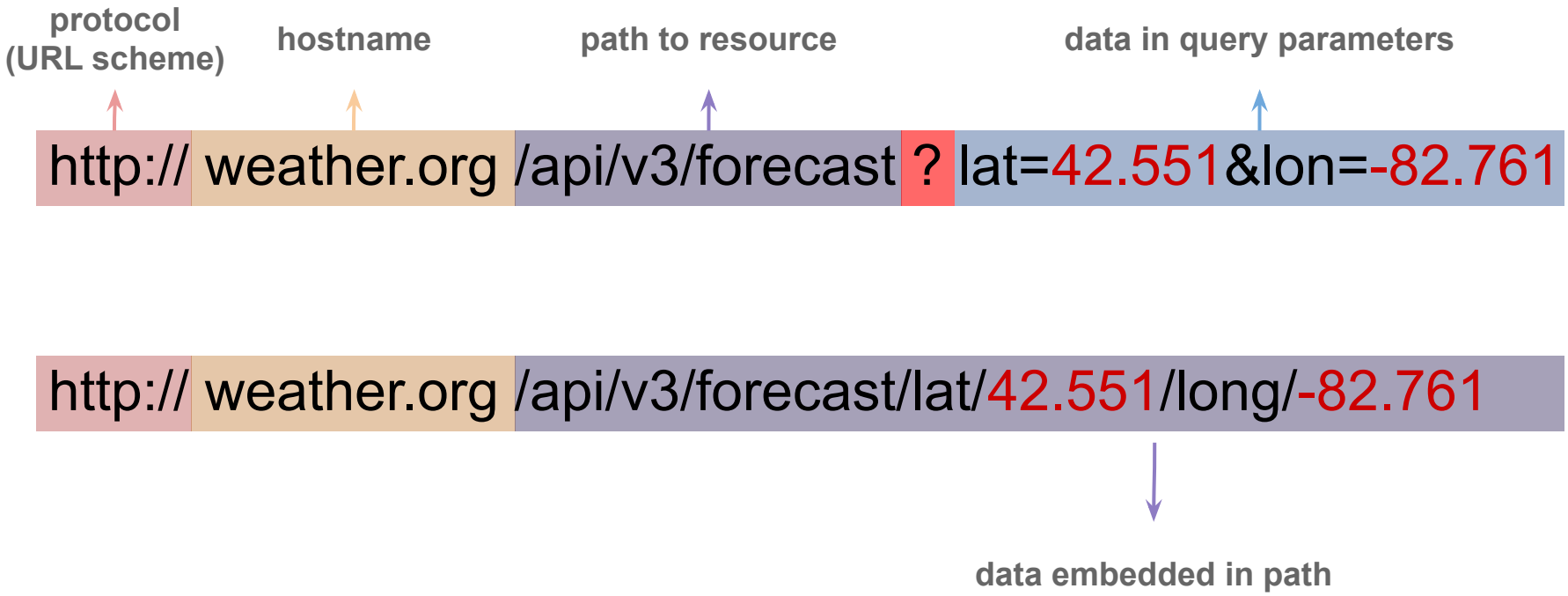


Correct way to logout

Passing Data To Components

Limited to “string-like” data

HTTP paths and query parameters



Passing “String-like” Data To Components

Component Route

*Show **7-day** forecast for ZIP code **49401***

Component	Path	Name	Data Embedded in Path
ForecastByZip.vue	/region	ByZip	/region/49401/7d /region/49401/next/7d

```
import ForecastByZip from "./components/ForecastByZip.vue";
const myComponentRoutes = [
  { path: "/", component: Home },
  { path: "/forecast", component: Forecast },
  { path: "/settings", component: Settings },
  { path: "/hourly", component: Hourly },
  {
    name: "ByZip",
    component: ForecastByZip,
    props: true,
    path: "/region/:zipCode/:numDays",
    // path: "/region/:zipCode/next/:numDays",
  },
];
```

main.ts

Sending & Receiving Props

```
import ForecastByZip from "./components/ForecastByZip.vue";
const myComponentRoutes = [
  // more routes here
  {
    name: "ByZip",
    component: ForecastByZip,
    props: true,
    path: "/region/:zipCode/:numDays",
  },
];
```

main.ts

```
<script setup lang="ts">
type ForecastDetailType = {
  zipCode: string;
  numDays: number;
};
const props = defineProps<ForecastDetailType>();
</script>
```

ForecastByZip.vue (recipient)

```
<script setup lang="ts">
import { useRouter } from "vue-router";
const appNav = useRouter();
function checkAtZip() {
  appNav.push({
    name: "ByZip",
    params: {
      zipCode: "48823",
      numDays: 10,
    },
  });
}
</script>
```

__.vue (sender)

Demo

Advanced:
CSS View Animations/Transitions

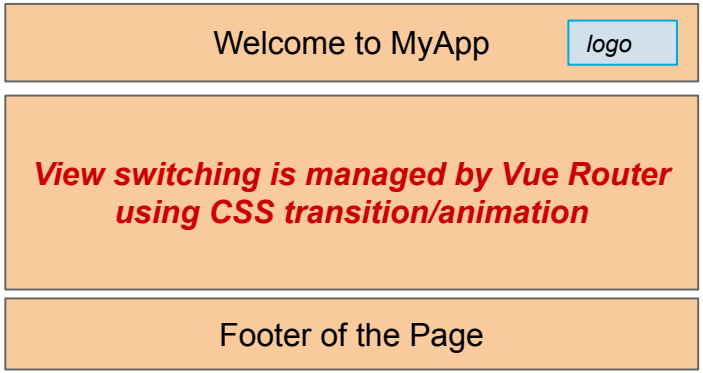
App.vue

No <transition>

```
<template>
  <div>
    <span>Welcome to MyApp</span>
    <img src="./logo.png" alt="MyApp" />
    <router-view v-slot="{ Component }">
      <component :is="Component" />
    </router-view>
    <div>Fotter of the Page</div>
  </div>
</template>
```

With <transition>

```
<template>
  <div>
    <span>Welcome to MyApp</span>
    <img src="./logo.png" alt="MyApp" />
    <router-view v-slot="{ Component }">
      <transition>
        <component :is="Component" />
      </transition>
    </router-view>
    <div>Fotter of the Page</div>
  </div>
</template>
```

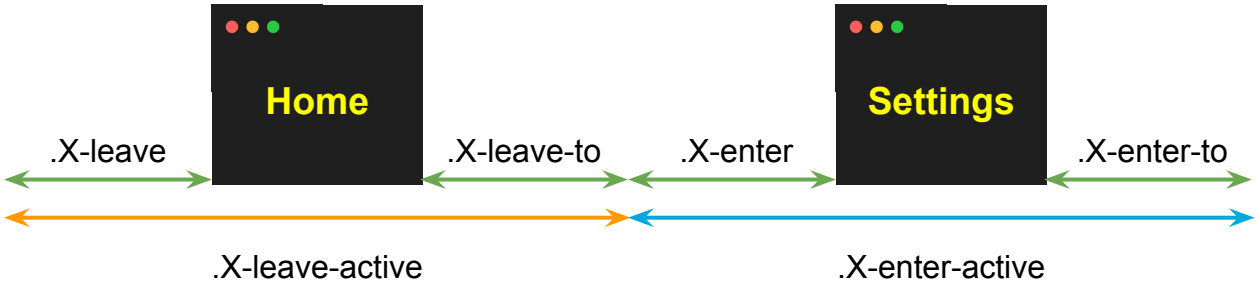


Note: When using transitions with router-view, ensure that every component displayed within has a **single root element** to allow the transition to work correctly.

Transition Classes: Page Transition Animation

Page transition: Home.vue ⇒ Settings.vue

- Leaving Page: Home (removed from router-view)
- Entering Page: Settings (inserted into router-view)
- Controlled by six CSS classes



<transition> & Transition CSS Classes

```
<router-view v-slot="{ Component }">
  <transition name="XYZ">
    <component :is="Component" />
  </transition>
</router-view>
```

CSS for outgoing view

```
.XYZ-leave-active {
  /* General animation/transition control */
}
.XYZ-leave {
  /* CSS properties BEFORE the view appears */
}
.XYZ-leave-to {
  /* CSS properties AFTER the view appears */
}
```

CSS for incoming view

```
.XYZ-enter-active {
  /* General animation/transition control */
}
.XYZ-enter {
  /* CSS properties BEFORE the view appears */
}
.XYZ-enter-to {
  /* CSS properties AFTER the view appears */
}
```

Transition Example

Demo

```
<router-view v-slot="{ Component }">
  <transition name="fade">
    <component :is="Component" />
  </transition>
</router-view>
```

Outgoing view slides RIGHT(from 0% to 100%) in 1000 ms

Incoming view slides DOWN (from 0% to 100%) in 1000ms

```
.fade-leave-active {
  transition-property: all;
  transition-duration: 1000ms;
  transition-timing-function: ease;
}
.fade-leave {
  transform: translateX(0%);
}
.fade-leave-to {
  transform: translateX(100%);
  background: green;
}
```

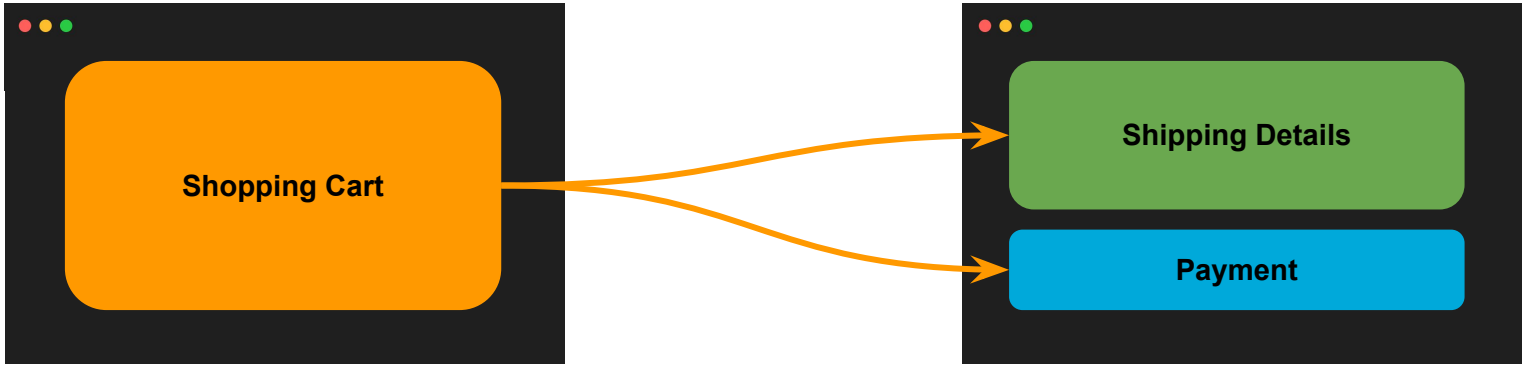
CSS for outgoing view

```
.fade-enter-active {
  transition-property: all;
  transition-duration: 1000ms;
  transition-timing-function: ease;
}
.fade-enter {
  transform: translateY(0%);
}
.fade-enter-to {
  transform: translateY(100%);
  background: rgb(238, 95, 12);
}
```

CSS for incoming view

Graphs of Timing Functions

Navigate into Multi-View Destination?



Navigate into Multi-View Destination

```
<!-- UI content -->
<div>
  <router-view></router-view>
  <router-view name="side"></router-view>
</div>
```

```
/* routing table */
{
  name: "shipAndPay",
  path: "/_____________",
  components: {
    default: ShippingDetails,
    side: Payment,
  },
},
```

Vue router navigation guards (hook functions)

Vue Router Navigation Guards

```ts
const router = createRouter({
  // vue router options go here
});
// "beforeEach" navigation guard
router.beforeEach((to, from, next) => {
  // your code here
  next();
});

// "afterEach" navigation guard
router.afterEach((to, from, failure) => {
  // your code here
});
```

main.ts

Global navigation guards: applied to the entire app

```vue
<script lang="ts">
export default class MyComponent extends Vue {
  beforeRouteEnter(to, from, next) {
    // your code here
  }
  beforeRouteLeave(to, from) {
    // your code here
  }
}
</script>
```

MyComponent.vue

In Component navigation guards: applied only to a specific component

[Navigation Guards]

Navigation Guard: Typical Use Case

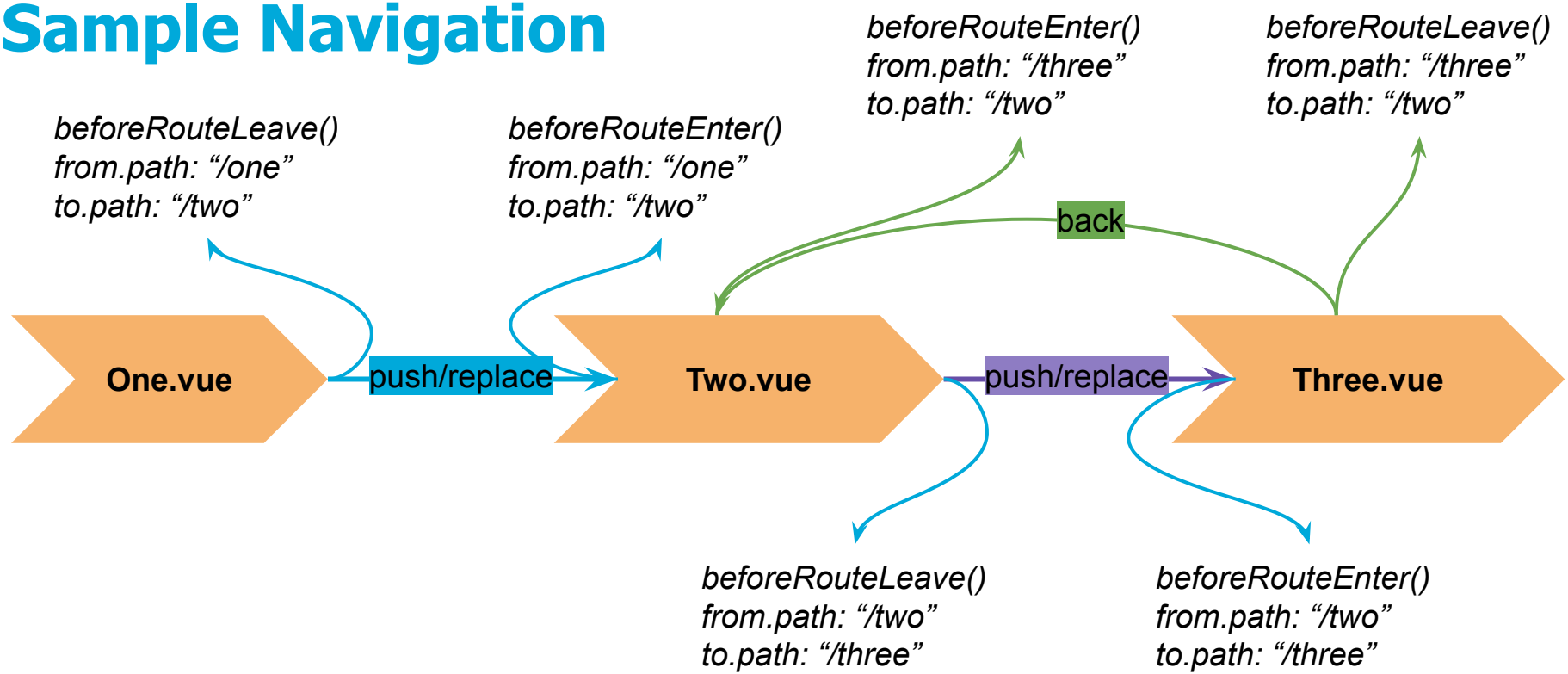
```
// BAD
router.beforeEach((to, from, next) => {
  if (to.name !== "Login" && !isAuthenticated) next({ name: "Login" });
  // if the user is not authenticated, `next` is called twice
  next();
});
```

```
// GOOD
router.beforeEach((to, from, next) => {
  if (to.name !== "Login" && !isAuthenticated) next({ name: "Login" });
  else next();
});
```

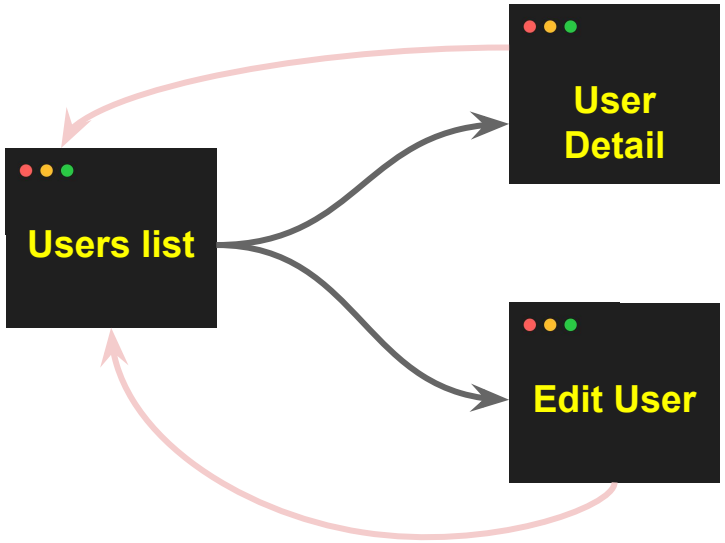
Vue Router Navigation Guards

Navigation Guard	Description	Example of Use Cases
beforeRouteEnter()	Called before Vue Router navigates **into** this component	• Keep statistics of how users enter a specific component • Prevent users from entering a specific component based on some conditions
beforeRouteLeave()	Called before Vue Router navigate **away** from this component	• Warn the user to save data when changes have been made • Undo any actions performed in beforeRouteEnter()

Sample Navigation



Practice: Random User App



Update Home.vue so that:

- Clicking a name opens the related Detail.vue page.
- Clicking Edit opens the related Edit.vue page.