Layouts.

We'll create a layout component called `DefaultLayout` and two views, `Home` and `About`, which use this layout to display their content. This example will help you understand how to structure your files and use slots for content projection.

Step-by-Step Example

1. Create the Layout Component

First, create a `DefaultLayout.vue` component that will act as the layout for your views. This layout will have a header, a footer, and a main content area where the view-specific content will be inserted.

src/layouts/DefaultLayout.vue:

```vue
<!-- layouts/DefaultLayout.vue -->

<template>
  <div>
    <header>
      <h1>Site Header</h1>
    </header>
    <main>
      <!-- Content inserted via slot will appear here -->
      0 references
      <slot></slot>
    </main>
    <footer>
      <p>Site Footer</p>
    </footer>
  </div>
</template>
```

```vue
<script>
export default {
  name: 'DefaultLayout',
}
</script>
```

```vue
<style>
/* Your styles here */
header {
  background-color: #333;
  color: white;
  padding: 10px;
  text-align: center;
}

footer {
  background-color: #333;
  color: white;
  padding: 10px;
  text-align: center;
}

main {
  padding: 20px;
}
</style>
```

2. Create the Home View

Next, create a `Home.vue` component that will use the `DefaultLayout` component and insert its content into the layout's slot.

src/views/Home.vue:

```vue
<!-- views/Home.vue -->

<template>
  <DefaultLayout>
    <h2>Home Page Content</h2>
    <p>Welcome to the home page!</p>
  </DefaultLayout>
</template>

<script>
import DefaultLayout from '@/layouts/DefaultLayout.vue';

export default {
  name: 'Home',
  components: {
    DefaultLayout,
  },
}
</script>

<style>
/* Your styles here */
</style>
```

3. Create the About View

Similarly, create an `About.vue` component that uses the `DefaultLayout` component.

src/views/About.vue:

```vue
<!-- views/About.vue -->

<template>
  <DefaultLayout>
    <h2>About Page Content</h2>
    <p>Welcome to the about page!</p>
  </DefaultLayout>
</template>

<script>
import DefaultLayout from '@/layouts/DefaultLayout.vue';

export default {
  name: 'About',
  components: {
    DefaultLayout,
  },
}
</script>

<style>
/* Your styles here */
</style>
```

4. Set Up the Router

Ensure that your router is set up to navigate between the `Home` and `About` views.

src/router/index.js:

```
// router/index.js

import { createRouter, createWebHistory } from 'vue-router';
import Home from '@/views/Home.vue';
import About from '@/views/About.vue';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home,
  },
  {
    path: '/about',
    name: 'About',
    component: About,
  },

];

const router = createRouter({
    history: createWebHistory(),
    routes
  });

export default router;
```

5. Main Application Entry Point

Ensure your main application entry point includes the router.

src/main.js:

```javascript
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App).use(router).mount('#app');
```

6. Root Component

Finally, your root component `App.vue` should have a `router-view` to render the current route's component.

src/App.vue:

```vue
<template>
  <div id="app">
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: 'App',
}
</script>

<style>
/* Global styles */
#app {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
</style>
```

Explanation

- **DefaultLayout.vue**: This is your layout component with a header, a main content area (`<slot></slot>`), and a footer.
- **Home.vue** and **About.vue**: These are your view components. They use the `DefaultLayout` component and insert their specific content into the layout's slot.
- **Router**: The router configuration maps routes to the respective view components.
- **App.vue**: The root component renders the current route's component using `router-view`.

By following this structure, you create a reusable layout that can be used across different views in your application, allowing you to maintain a consistent layout while changing only the main content based on the route.

Summary

- **Named Slot**: Use `<slot name="slotName"></slot>` to define where content should be projected.
- **Default Slot**: Use `<slot></slot>` for content projection without naming.

By using slots, you can create flexible and reusable layout components that can display different content based on the parent component.