

# Practices in SSR

Building Universal Apps

**Why?**

# Pre-rendering

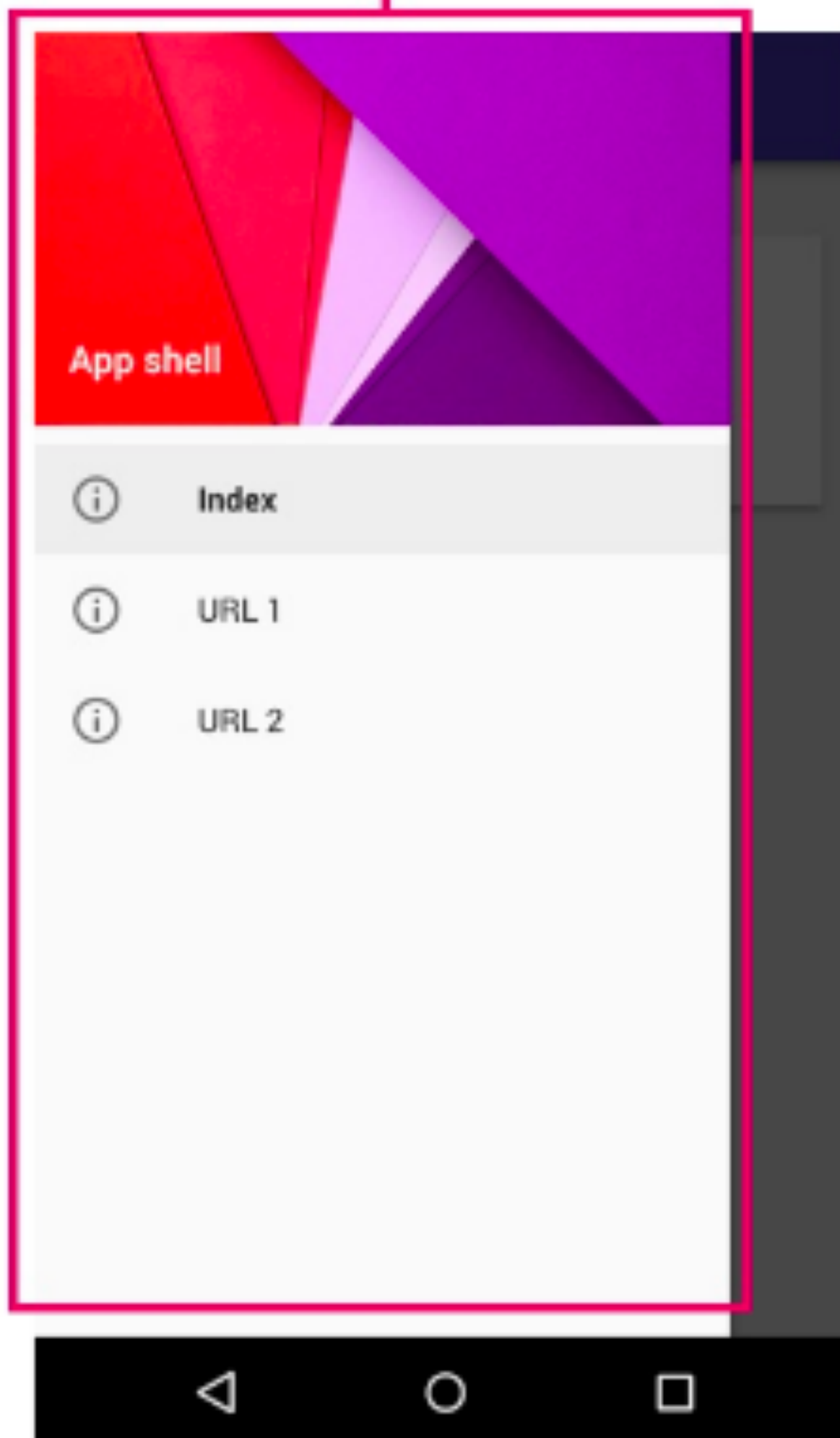
# App Shell Architecture + Code Splitting

# application shell



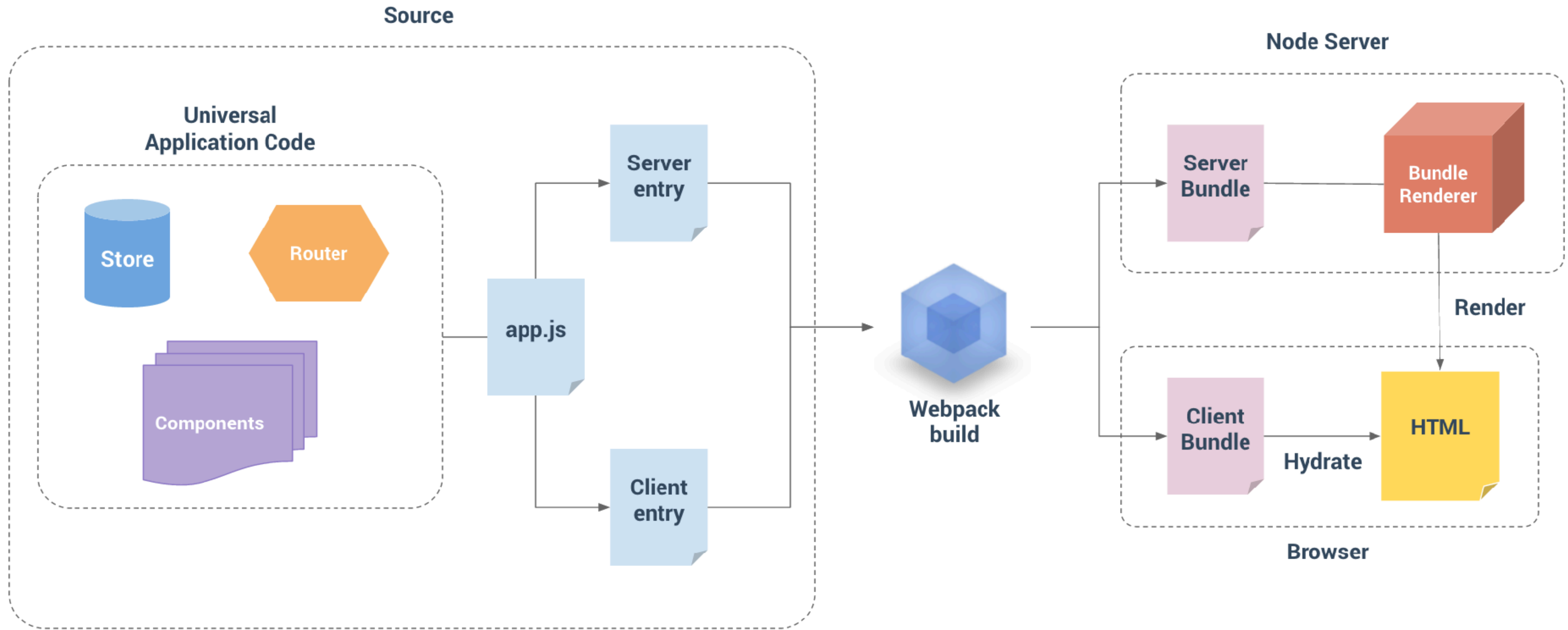
Cached shell loads **instantly** on repeat visits.

# content



Dynamic content then populates the view

# A crash course in SSR





Browser



Renderer



Real App



# Hydration



# Hydration

```
<div id="app" data-server-rendered="true">
```

# Practices

# Inlining Critical CSS




```

<link rel="prefetch" href="/_nuxt/pages/guide/_slug.52c8966fc919085fa6c2.js">
<link rel="prefetch" href="/_nuxt/pages/faq/_slug.5f7b64b7a1289e5beafb.js">
<link rel="prefetch" href="/_nuxt/pages/api/_slug.fb7657c88771bdd30c49.js">
<link rel="prefetch" href="/_nuxt/pages/guide.979c3d23bb67c25de279.js">
<link rel="prefetch" href="/_nuxt/pages/faq.02924d2b7e4370167c7d.js">
<link rel="prefetch" href="/_nuxt/pages/examples.fd85c707f8ad650f66e7.js">
<link rel="prefetch" href="/_nuxt/pages/api.9561014edaf794e84cdd.js">
<link rel="prefetch" href="/_nuxt/pages/guide/release-notes.23cb4e7d94a56ae7b86e.js">
▼ <style data-vue-ssr-id="3f3730c2:0 543ce85a:0 f6b17ac4:0 20dafe74:0 6d6913de:0 05fa800d:0 ff2e28c
7a7d272c:0 441366fe:0 c0094f36:0">
  /*! normalize.css v7.0.0 | MIT License | github.com/necolas/normalize.css */html{line-height:1.
size:2em;margin:.67em 0}figcaption,figure,main{display:block}figure{margin:1em 40px}hr{webkit-
color:transparent;-webkit-text-decoration-skip:objects}abbr[title]{border-bottom:none;text-deco
weight:bolder}code,kbd,samp{font-family:monospace,monospace;font-size:1em}dfn{font-style:italic
align:baseline}sub{bottom:-.25em}sup{top:-.5em}audio,video{display:inline-block}audio:not([cont
size:100%;line-height:1.15;margin:0}button,input{overflow:visible}button,select{text-transform:
[type=submit]::-moz-focus-inner,button::-moz-focus-inner{border-style:none;padding:0}[type=butt
.75em .625em]legend{-webkit-box-sizing:border-box;box-sizing:border-box;color:inherit;display:t
[type=radio]{-webkit-box-sizing:border-box;box-sizing:border-box;padding:0}[type=number]::-webk
webkit-search-cancel-button,[type=search]::-webkit-search-decoration{-webkit-appearance:none}::
[hidden],template{display:none}
.hljs{display:block;overflow-x:auto;padding:.5em;color:#333;background:#f8f8f8}.hljs-comment,.h
.hljs-attr,.hljs-template-variable,.hljs-variable{color:teal}.hljs-doctag,.hljs-string{color:#d
weight:700}.hljs-attribute,.hljs-name,.hljs-tag{color:navy;font-weight:400}.hljs-link,.hljs-reg
deletion{background:#fdd}.hljs-addition{background:#dfd}.hljs-emphasis{font-style:italic}.hljs-

```

# Routing





```
new Promise((resolve, reject) => {
  const { app, router } = createApp()

  // set server-side router's location
  router.push(context.url)

  // wait until router has resolved possible async components and hooks
  router.onReady(() => {
    const matchedComponents = router.getMatchedComponents()
    // no matched routes, reject with 404
    if (!matchedComponents.length) {
      return reject({ code: 404 })
    }

    // the Promise should resolve to the app instance so it can be rendered
    resolve(app)
  }, reject)
})
```

# Data Fetching





```
router.onReady(() => {
  const matchedComponents = router.getMatchedComponents()
  if (!matchedComponents.length) {
    return reject({ code: 404 })
  }

  // call `asyncData()` on all matched route components
  Promise.all(matchedComponents.map(Component => {
    if (Component.asyncData) {
      return Component.asyncData({
        store,
        route: router.currentRoute
      })
    }
  })).then(() => {
    // After all preFetch hooks are resolved, our store is now
    // filled with the state needed to render the app.
    // When we attach the state to the context, and the `template` option
    // is used for the renderer, the state will automatically be
    // serialised and injected into the HTML as `window.__INITIAL_STATE__`.
    context.state = store.state

    resolve(app)
  }).catch(reject)
}, reject)
```

# State Management



```
export default {
  asyncData ({ store, route }) {
    // return the Promise from the action
    return store.dispatch('fetchItem', route.params.id)
  },

  computed: {
    // display the item from store state.
    item () {
      return this.$store.state.items[this.$route.params.id]
    }
  }
}
```

# Head Management



```
created () {  
  this.$ssrContext.title = 'Page Title'  
}
```

```
<html>  
  <head>  
    <title>{{ title }}</title>  
  </head>  
  <body>  
    ...  
  </body>  
</html>
```

# Streaming

# Microcaching

# SSR

## Caveats

- No data reactivity
  - Only beforeCreate & created lifecycle hooks
  - Directives
  - Platform specific API usage
  - Stateful singletons
-



