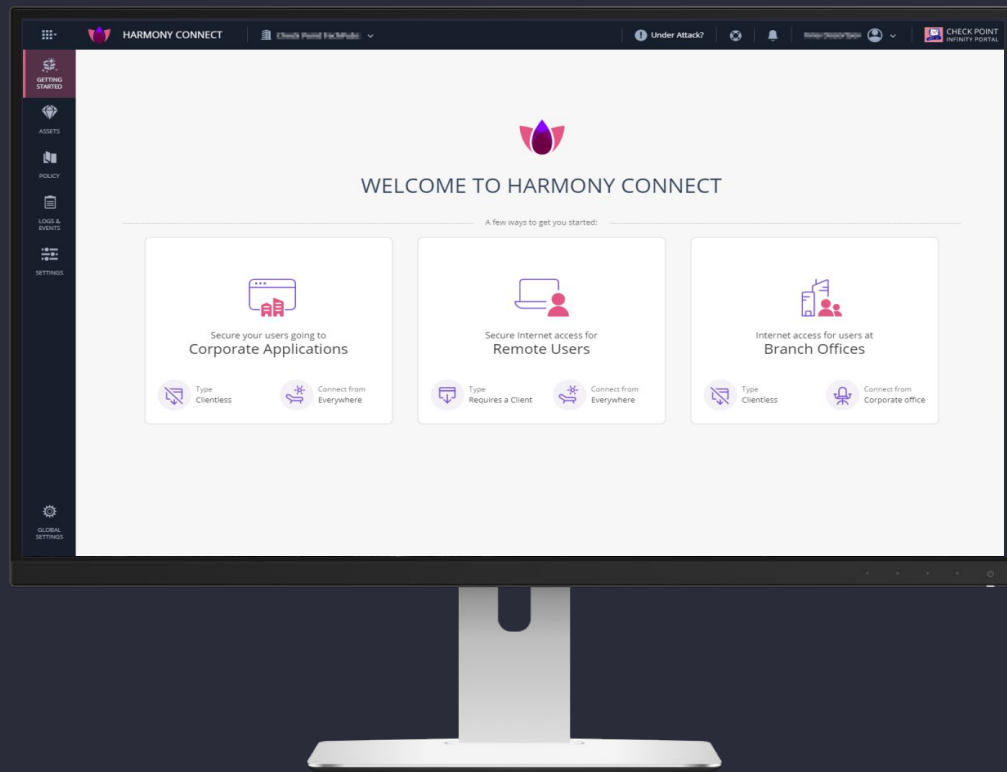


# Rendering Angular modules inside React application

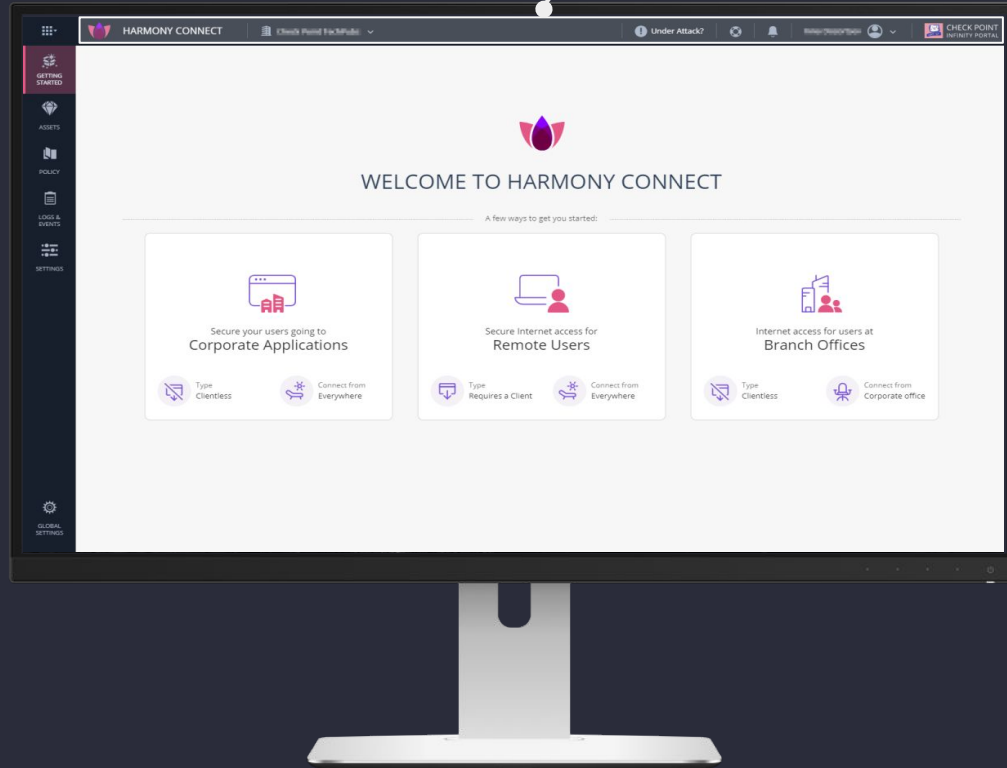


I really, really dislike using multiple frameworks in single application

But...

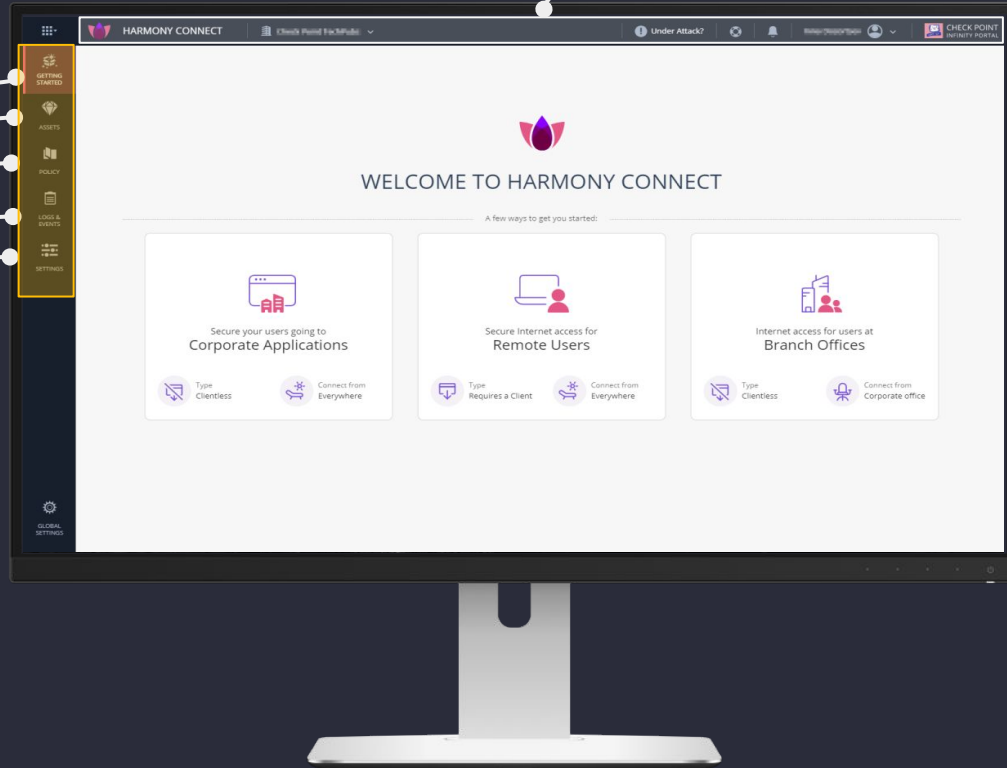


Main App  
Navbar



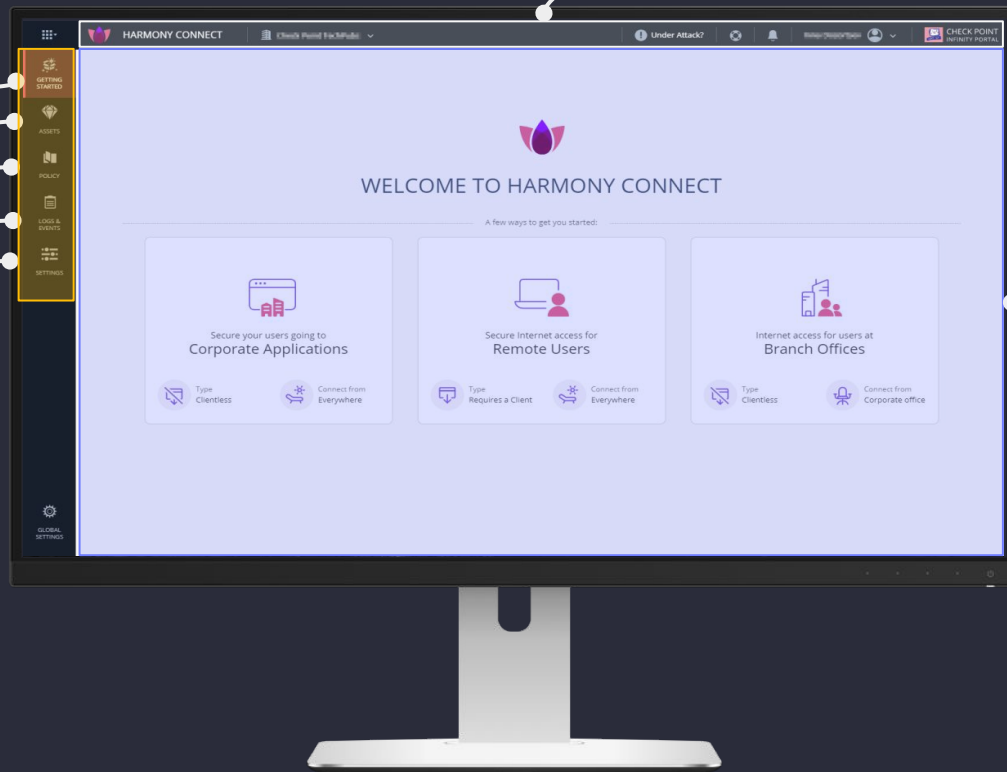
Main App  
Navbar

Mini  
Apps



Main App  
Navbar

Mini  
Apps



App  
Renderer

Rewriting the whole application is seemed Terrifying



**It was time to challenge managers and  
frontend architects**



# About Me



**David Antoon**

@davidantoon

Frontend-Architect at Frontegg 🥚 + ⚛️ A✓

From the north 🇩🇪

Awesome wife with two children 🧑🏻‍🤝‍🧑🏻

Crypto miner and trader ₿

# The struggle with multiple frameworks

- Getting the design just right can be really painful

# The struggle with multiple frameworks

Getting the design just right can be really painful

- Hooks framework routers to play together

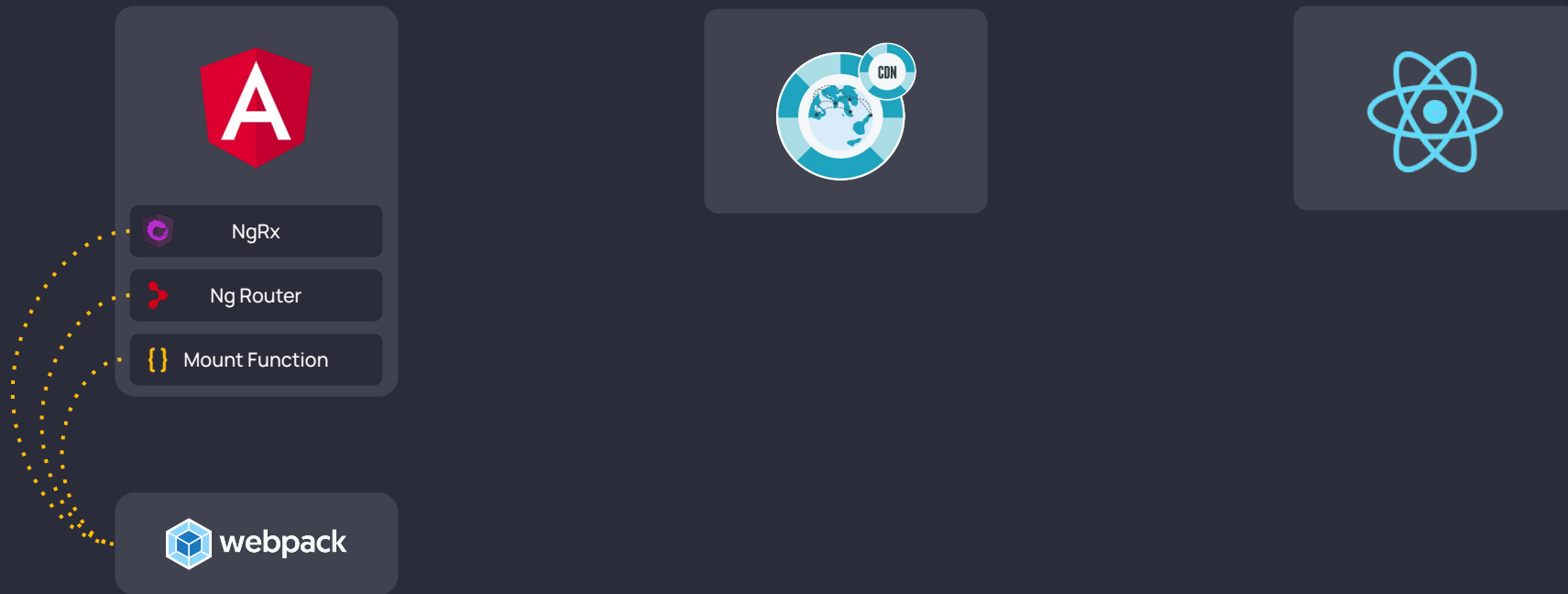
# The struggle with multiple frameworks

Getting the design just right can be really painful

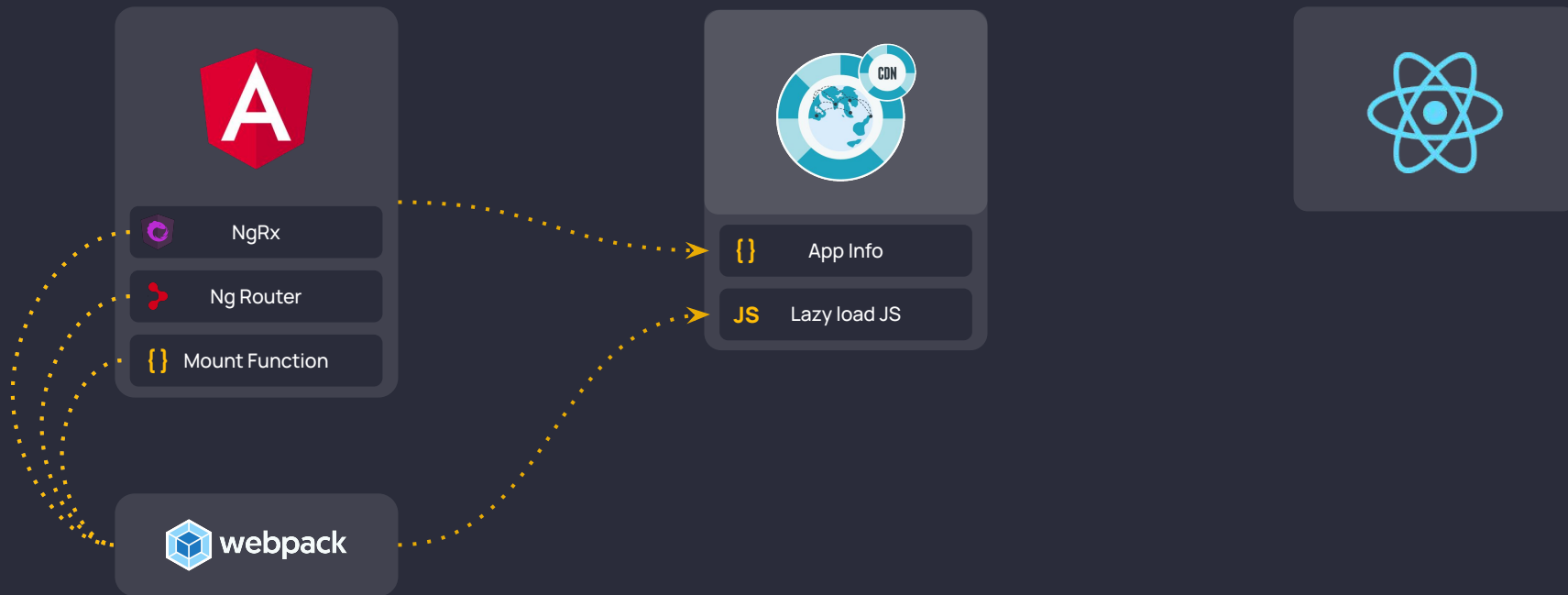
Hooks framework routers to play together

- Syncing shared application state among frameworks

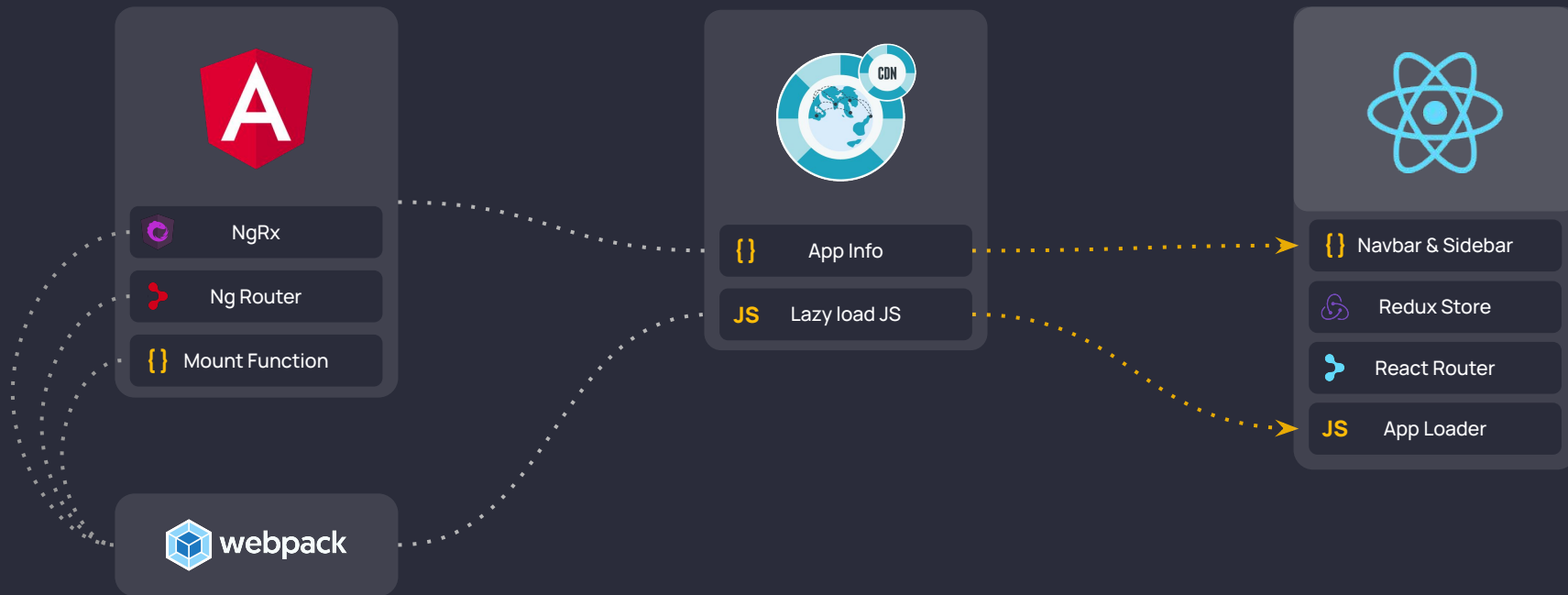
# Vision



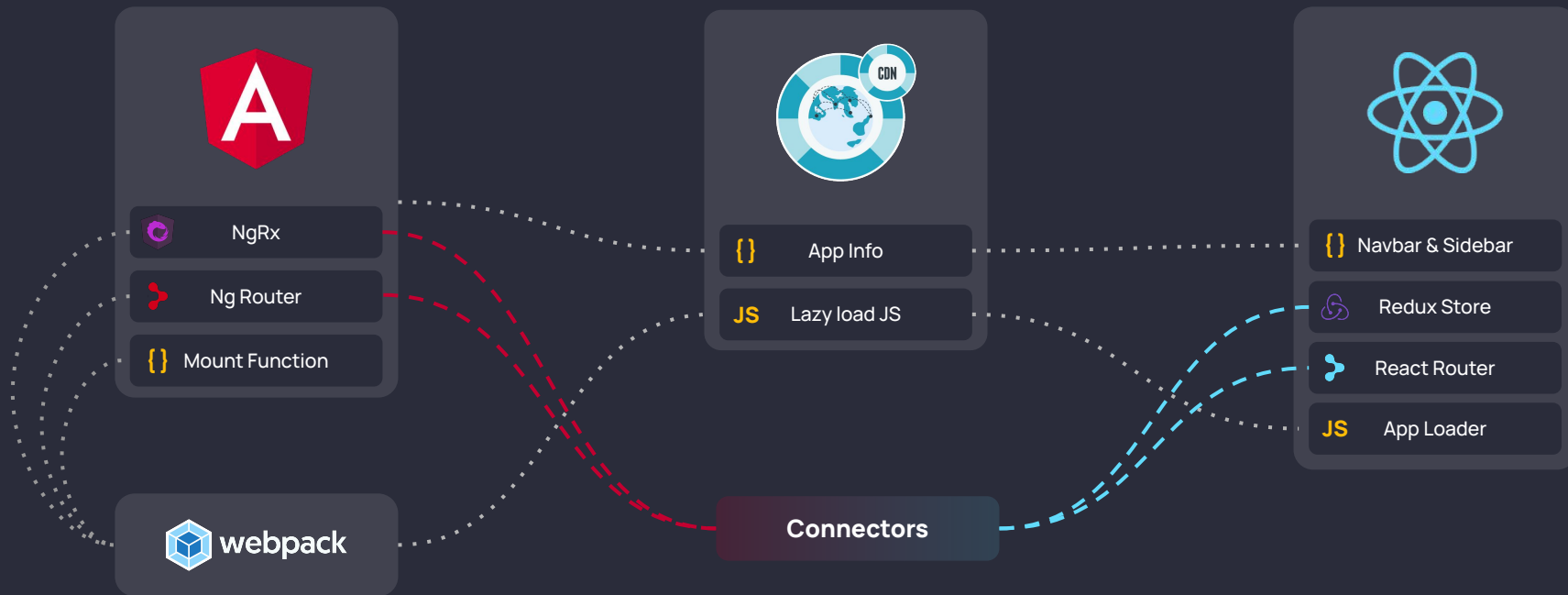
# Vision



# Vision

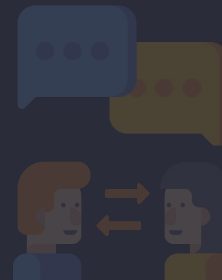


# Vision





# Limitations

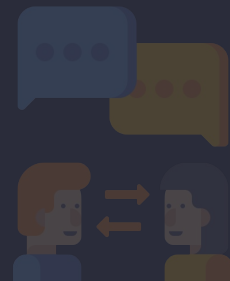


# Limitations



Shadow DOM

Media-Query Factor



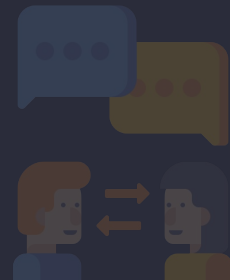
# Limitations



Shadow DOM  
Media-Query Factor



Code-Splitting  
Lazy-Loading



# Limitations



Shadow DOM  
Media-Query Factor



Code-Splitting  
Lazy-Loading



Using Observables  
for async  
communication

# Step 1/5

## Prepare for dynamic render

- Change main application selector to a unique value to identify it from hosted app

```
// file: ./app/app.component.ts
```

```
import { Component, ViewEncapsulation } from '@angular/core';  
import { APPLICATION_ID } from '../helpers';
```

```
@Component({  
  selector: APPLICATION_ID,  
  encapsulation: ViewEncapsulation.ShadowDom,  
  templateUrl: './app.component.html',  
  styleUrls: [ './app.component.css' ]  
})  
export class AppComponent {  
  title = 'hybrid-angular';  
}
```

# Step 1/5

## Prepare for dynamic render

Change main application selector to a unique value to identify it from hosted app

- Encapsulation the application in Shadow Dom

```
// file: ./app/app.component.ts
```

```
import { Component, ViewEncapsulation } from '@angular/core';  
import { APPLICATION_ID } from '../helpers';
```

```
@Component({  
  selector: APPLICATION_ID,  
  encapsulation: ViewEncapsulation.ShadowDom,  
  templateUrl: './app.component.html',  
  styleUrls: [ './app.component.css' ]  
})  
export class AppComponent {  
  title = 'hybrid-angular';  
}
```

# Step 1/5

## Prepare for dynamic render

Change main application selector to a unique value to identify it from hosted app

Encapsulation the application in Shadow Dom

■ Create new helper.ts file

```
// file: ./helpers.ts
```

```
const scriptsFilter = (script: HTMLScriptElement): boolean => {  
  return script.src.indexOf(ApplicationId) !== -1;  
}
```

```
declare let __webpack_public_path__: string;
```

```
export const enableDynamicPublicPath = (): void => {  
  const scriptsArr = Array.from(document.scripts);  
  const src = scriptsArr.find(scriptsFilter)?.src  
  if (src) {  
    const publicPath = src.substring(0, src.indexOf(ApplicationId));  
    __webpack_public_path__ = `${publicPath}/`  
  }  
}
```

```
// file: ./main.ts
```

```
import { enableDynamicPublicPath } from './helpers';
```

```
enableDynamicPublicPath();
```

# Step 1/5

## Prepare for dynamic render

Change main application selector to a unique value to identify it from hosted app

Encapsulation the application in Shadow Dom

Create new helper.ts file

- Override webpack public path at runtime with current script src url

```
// file: ./helpers.ts
```

```
const scriptsFilter = (script: HTMLScriptElement): boolean => {  
  return script.src.indexOf(ApplicationId) !== -1;  
}
```

```
declare let __webpack_public_path__: string;
```

```
export const enableDynamicPublicPath = (): void => {  
  const scriptsArr = Array.from(document.scripts);  
  const src = scriptsArr.find(scriptsFilter)?.src  
  if (src) {  
    const publicPath = src.substring(0, src.indexOf(ApplicationId));  
    __webpack_public_path__ = `${publicPath}/`  
  }  
}
```

```
// file: ./main.ts
```

```
import { enableDynamicPublicPath } from './helpers';
```

```
enableDynamicPublicPath();
```



# Step 2/5

## Make it injectable application

📄 Create AppContext.ts file

```
// file: ./AppContext.ts

interface App {
  id: string;
  mount: (container: Element, router, store: Store) => Promise<void>;
  unmount: () => void;
}

class AppContext {
  /** Holding the mounted application instance */
  private app?: NgModuleRef;
  /** Holding application HTML element reference */
  private element?: Element;
  /** Holding hosted app's store/router references */
  public store?: Store;
  public router?: Router;

  public init(appModule: Type<any>): void;
  private createMountFunction(appModule: Type<any>): void;
  private destroy(): void;
}

export default new AppContext()
```

# Step 2/5

## Make it injectable application

### ■ Create Singleton AppContext

```
// file: ./AppContext.ts
```

```
interface App {  
  id: string;  
  mount: (container: Element, router, store: Store) => Promise<void>;  
  unmount: () => void;  
}
```

```
class AppContext {  
  /** Holding the mounted application instance */  
  private app?: NgModuleRef;  
  /** Holding application HTML element reference */  
  private element?: Element;  
  /** Holding hosted app's store/router references */  
  public store?: Store;  
  public router?: Router;  
  
  public init(appModule: Type<any>): void;  
  private createMountFunction(appModule: Type<any>): void;  
  private destroy(): void;  
}
```

```
export default new AppContext()
```

# Step 2/5

## Make it injectable application

Create Singleton AppContext

Initialize method

```
// file: ./AppContext.ts
```

```
interface App {  
  id: string;  
  mount: (container: Element, router, store: Store) => Promise<void>;  
  unmount: () => void;  
}
```

```
class AppContext {  
  /** Holding the mounted application instance */  
  private app?: NgModuleRef;  
  /** Holding application HTML element reference */  
  private element?: Element;  
  /** Holding hosted app's store/router references */  
  public store?: Store;  
  public router?: Router;  
  
  public init(appModule: Type<any>): void;  
  private createMountFunction(appModule: Type<any>): void;  
  private destroy(): void;  
}
```

```
export default new AppContext()
```

# Step 2/5

## Make it injectable application

Create Singleton AppContext

Initialize method

- ▶ Create Render Element and Mount functionality

```
/**file: */AppContext.ts
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Store } from '@ngrx/store';
import { NgModuleRef } from '@angular/core';
import { createRenderElement } from './createRenderElement';
import { bootstrapModule } from './bootstrapModule';

public init(appModule: Type<any>) {
  window.Apps = {
    ...window.Apps,
    [APPLICATION_ID]: {
      id: APPLICATION_ID,
      mount: this.createMountFunction(appModule),
      unmount: this.destroy
    }
  };
  /** Holding the mounted application instance */
  private app?: NgModuleRef;
  /** Holding application HTML element reference */
  private element?: Element;
  private createMountFunction(appModule: Type<any>){
    return async (container: Element, router: Router, store: Store) => {
      this.element = createRenderElement(container);
      this.router = router;
      this.store = store;
      this.app = await platformBrowserDynamic()
        .bootstrapModule(appModule);
    }
  }
  private destroy(): void {
    // ...
  }
}

export default new AppContext()
```

# Step 2/5

## Make it injectable application

Create Singleton AppContext

Initialize method

Create Render Element and Mount  
functionality

► Use AppContext.init

```
// file: ./main.ts
```

```
import { AppModule } from './app/app.module';  
import AppContext from './AppContext';  
import { enableDynamicPublicPath } from './helpers';
```

```
enableDynamicPublicPath()
```

```
AppContext.init(AppModule)
```

# Step 3/5

## Router Connection

■ Add BASE\_HREF provider

```
// file: ./app/app.module.ts

import { APP_BASE_HREF } from '@angular/common';
import { APPLICATION_ID } from '../helpers';

@NgModule({
  declarations: [ /*...*/ ],
  imports: [ /*...*/ ],
  providers: [ {
    provide: APP_BASE_HREF,
    useValue: `/${APPLICATION_ID}`,
  } ],
  bootstrap: [ /*...*/ ]
})
export class AppModule {}
```

# Step 3/5

## Router Connection

Add BASE\_HREF provider

► AppContext router use-case

```
// file: ./app/app.component.ts

import AppContext from '../AppContext';

@Component({ /* ... */ })
export class AppComponent {

  navigateHostedApp(){
    AppContext.router.push('/other-app-id')
  }

}
```

# Step 4/5

## Store Connection

### ■ Create Injectable Service

```
// file: ./app/app.store-syncer.ts
```

```
@Injectable({ providedIn: 'root' })
export class StoreConnector {
  private stateSubject = new BehaviorSubject<HostedAppState>({});
  private userStateSubject = new BehaviorSubject<HostedAppState['user']>({});

  constructor() {
    AppContext.store.subscribe(() => {
      const state = AppContext.store!.getState();

      this.stateSubject.next(state);
      this.userStateSubject.next(state.user);
    });
  }

  get state$(): Observable<HostedAppState> {
    return this.stateSubject.asObservable();
  }

  get userState$(): Observable<HostedAppState['user']> {
    return this.userStateSubject.asObservable();
  }
}
```



# Step 4/5

## Store Connection

Create Injectable Service

► Add Observables

```
// file: ./app/app.store-syncer.ts
```

```
@Injectable({ providedIn: 'root' })
export class StoreConnector {
  private stateSubject = new BehaviorSubject<HostedAppState>({});
  private userStateSubject = new BehaviorSubject<HostedAppState['user']>({});

  constructor() {
    AppContext.store.subscribe(() => {
      const state = AppContext.store!.getState();

      this.stateSubject.next(state);
      this.userStateSubject.next(state.user);
    });
  }

  get state$(): Observable<HostedAppState> {
    return this.stateSubject.asObservable();
  }

  get userState$(): Observable<HostedAppState['user']> {
    return this.userStateSubject.asObservable();
  }
}
```

# Step 4/5

## Store Connection

Create Injectable Service

Add Observables

▶ Listen to state changes

```
// file: ./app/app.store-syncer.ts
```

```
@Injectable({ providedIn: 'root' })
export class StoreConnector {
  private stateSubject = new BehaviorSubject<HostedAppState>({});
  private userStateSubject = new BehaviorSubject<HostedAppState['user']>({});

  constructor() {
    AppContext.store.subscribe(() => {
      const state = AppContext.store!.getState();

      this.stateSubject.next(state);
      this.userStateSubject.next(state.user);
    });
  }

  get state$(): Observable<HostedAppState> {
    return this.stateSubject.asObservable();
  }

  get userState$(): Observable<HostedAppState['user']> {
    return this.userStateSubject.asObservable();
  }
}
```

# Step 5/5

## Connect with Hosted App

- ▶ Add script to your HTML

```
// file: ./public/index.html

<html>
  <head>
    <!-- head tags -->
  </head>
  <body>
    <script type="application/javascript"
      src="https://{my-cdn-url}/hybrid-angular/index.js"></script>
  </body>
</html>
```

# Step 5/5

## Connect with Hosted App

Add script to your HTML

► Wait for Application loading

```
interface AppLoaderProps {  
  appId: string;  
}  
  
const AppLoader: FC<AppLoaderProps> = ({ appId }) => {  
  const elementRef = useRef<HTMLDivElement>(null);  
  const [ loading, setLoading ] = useState(true);  
  
  useEffect(() => {  
    const interval = setInterval(() => {  
      if (window.Apps?.[appId]) {  
        clearInterval(interval);  
        setLoading(false);  
        window.Apps?.[appId]?.mount(elementRef.current);  
      }  
    });  
    return () => clearInterval(interval);  
  }, [appId])  
  
  return <>  
    <div ref={elementRef}/>  
    {loading && <div>Loading Application...</div>}  
  </>  
}
```

# Step 5/5

## Connect with Hosted App

Add script to your HTML

Wait for Application loading

Mount Application

```
interface AppLoaderProps {
  appId: string;
}

const AppLoader: FC<AppLoaderProps> = ({ appId }) => {
  const elementRef = useRef<HTMLDivElement>(null);
  const [ loading, setLoading ] = useState(true);

  useEffect(() => {
    const interval = setInterval(() => {
      if (window.Apps?.[appId]) {
        clearInterval(interval);
        setLoading(false);
        window.Apps?.[appId]?.mount(elementRef.current);
      }
    });
    return () => clearInterval(interval);
  }, [appId])

  return <>
    <div ref={elementRef}/>
    {loading && <div>Loading Application...</div>}
  </>
}
```



David Antoon

@davidantoon

Download the slides &  
working example

