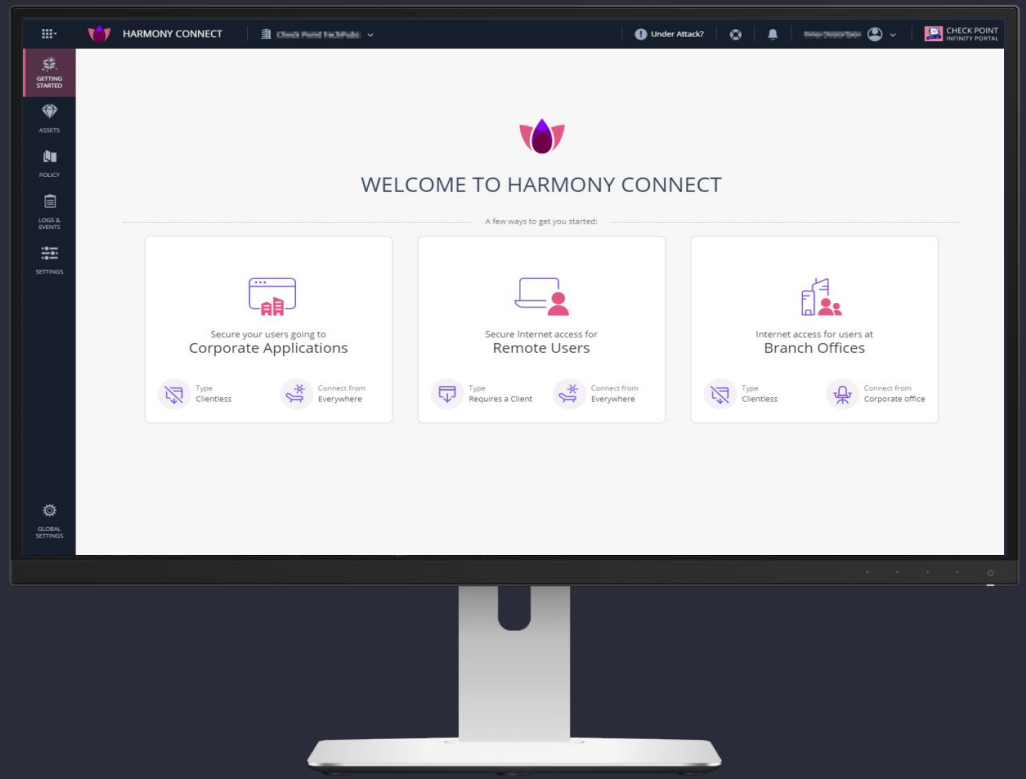
frontegg

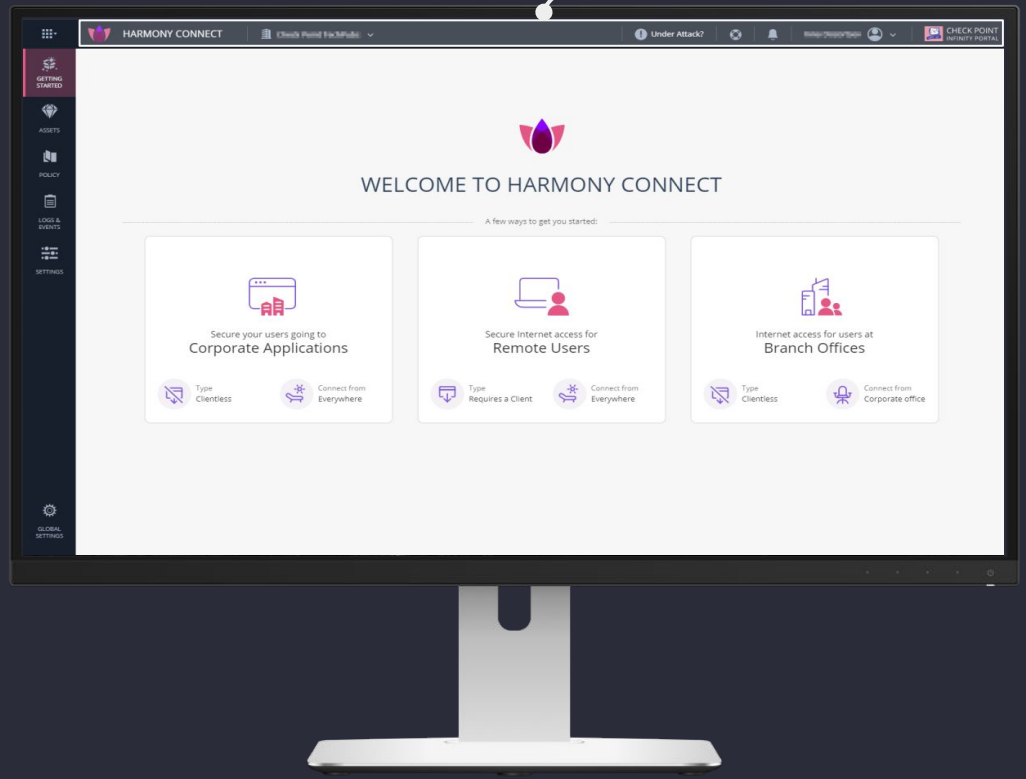
Rendering Angular modules inside React application

I really, really dislike using multiple frameworks in single application

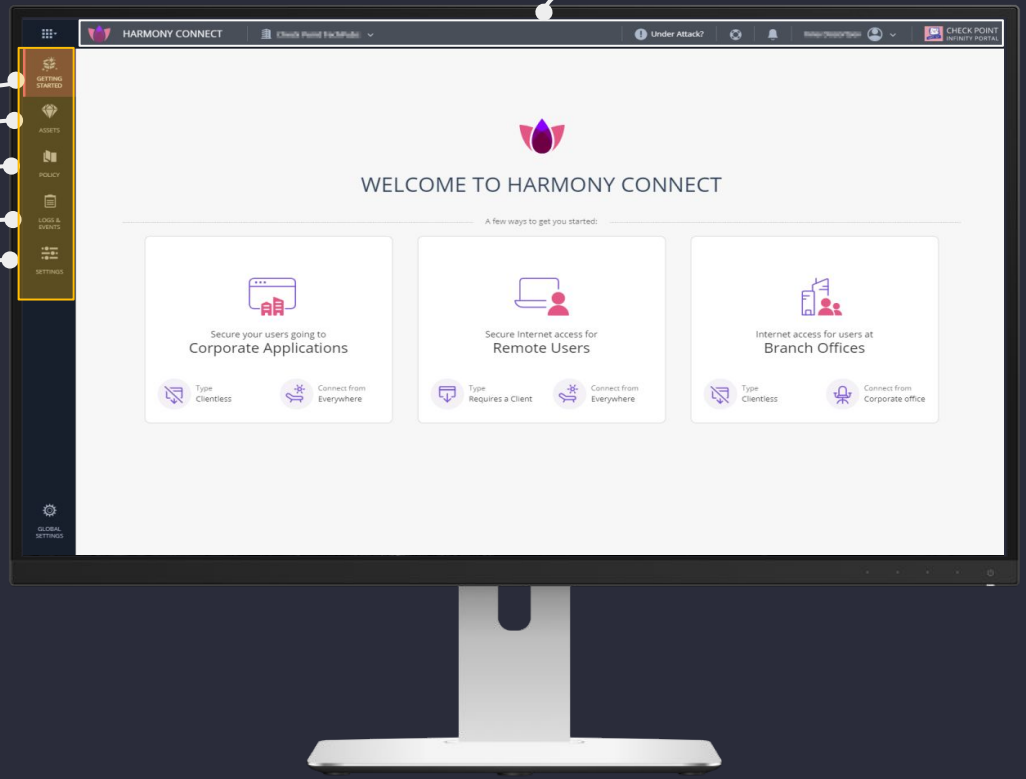
But...


HARMONY CONNECT
Under Attack?
CHECK POINT
INFINITY PORTAL
GETTING STARTED
ASSETS
POLICY
LOGS & EVENTS
SETTINGS
GLOBAL SETTINGS
WELCOME TO HARMONY CONNECT
A few ways to get you started:
Secure your users going to
Corporate Applications
Type
Clientless
Connect from
Everywhere
Secure Internet access for
Remote Users
Type
Requires a Client
Connect from
Everywhere
Internet access for users at
Branch Offices
Type
Clientless
Connect from
Corporate office

Main App
Navbar

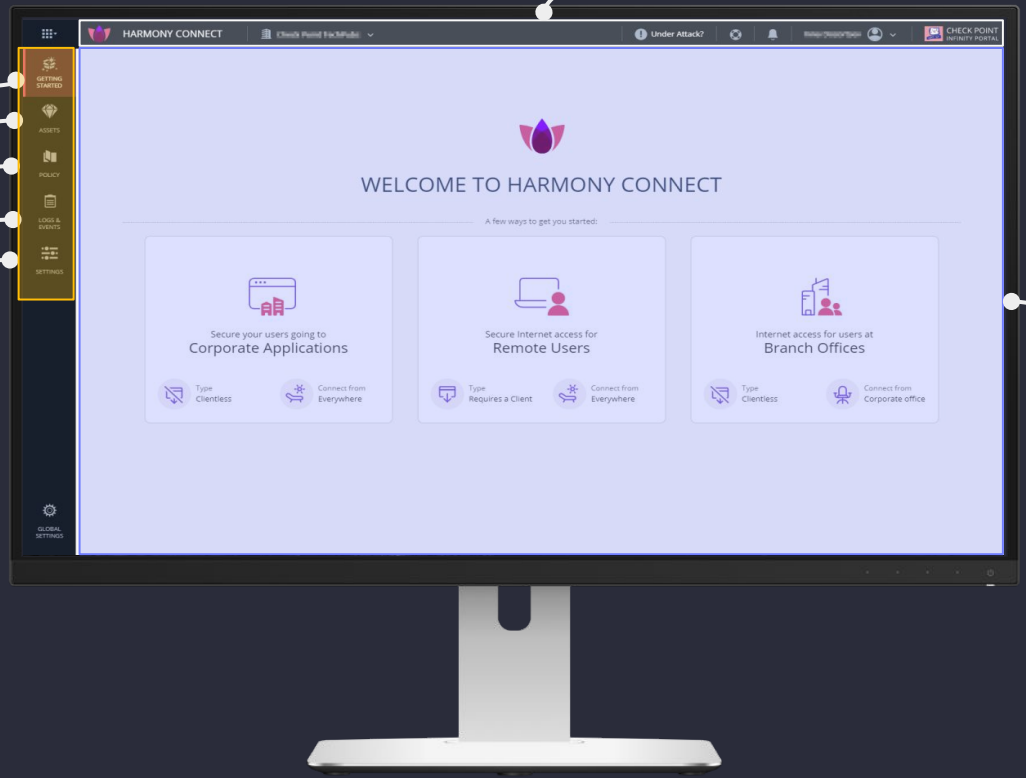


Main App Navbar



Mini Apps

Main App Navbar



Mini Apps

App Renderer

Rewriting the whole application is seemed Terrifying

frontegg

It was time to challenge managers and frontend architects

frontegg

About Me

David Antoon

@davidantoon

- Frontend-Architect at Frontegg + 
- From the north 
- Awesome wife with two children 
- Sport Cars 
- 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

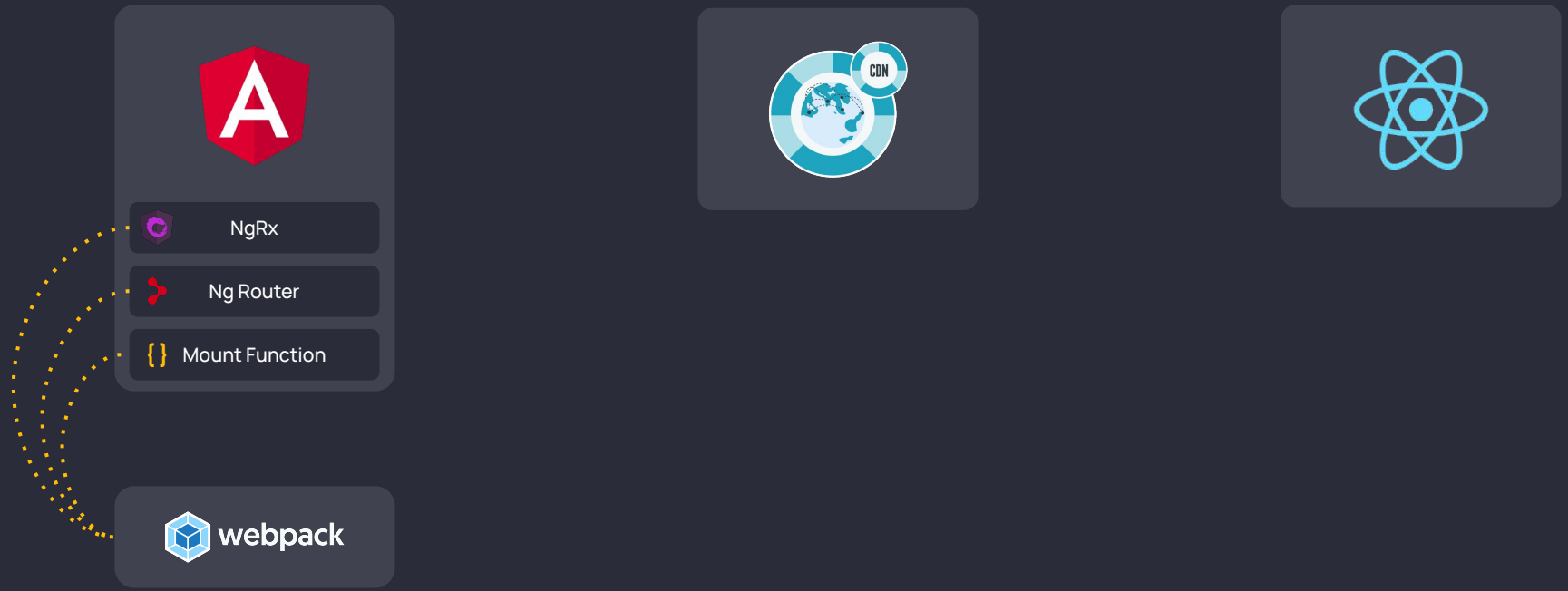
Getting the design just right can be really painful

Hooks framework routers to play together

- Syncing shared application state among frameworks

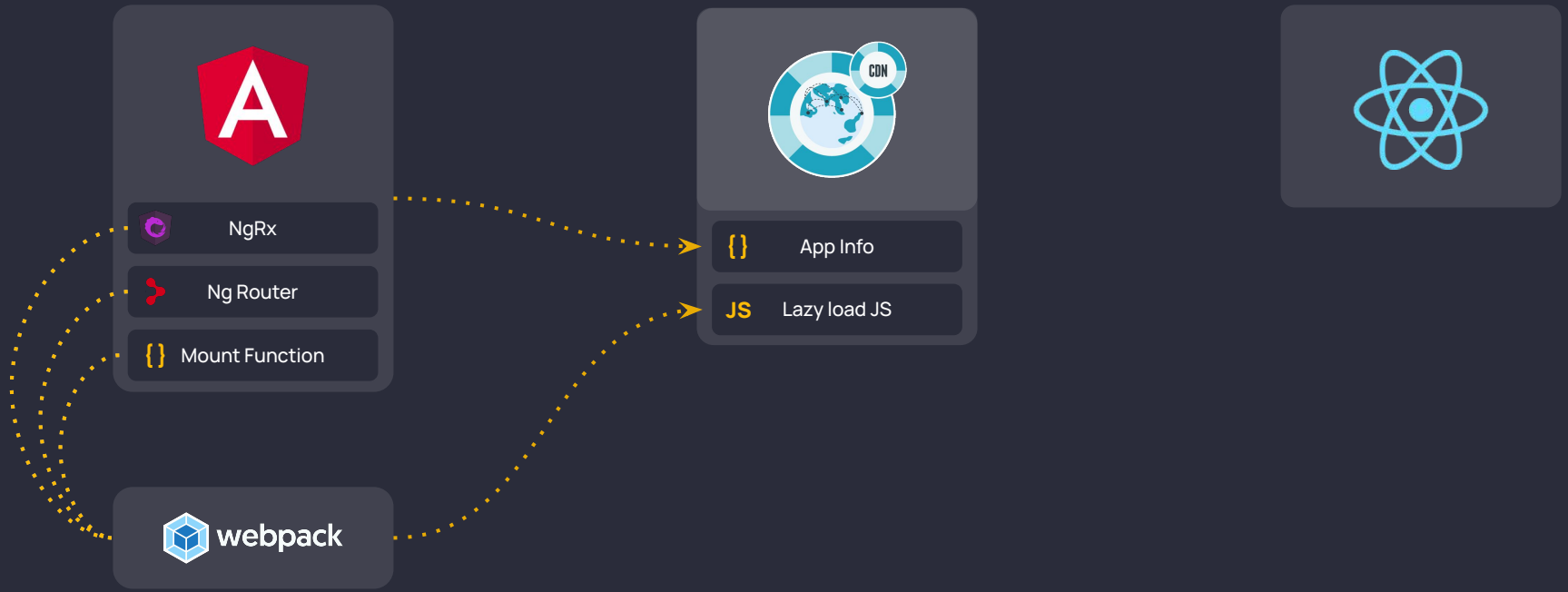
frontegg

Vision



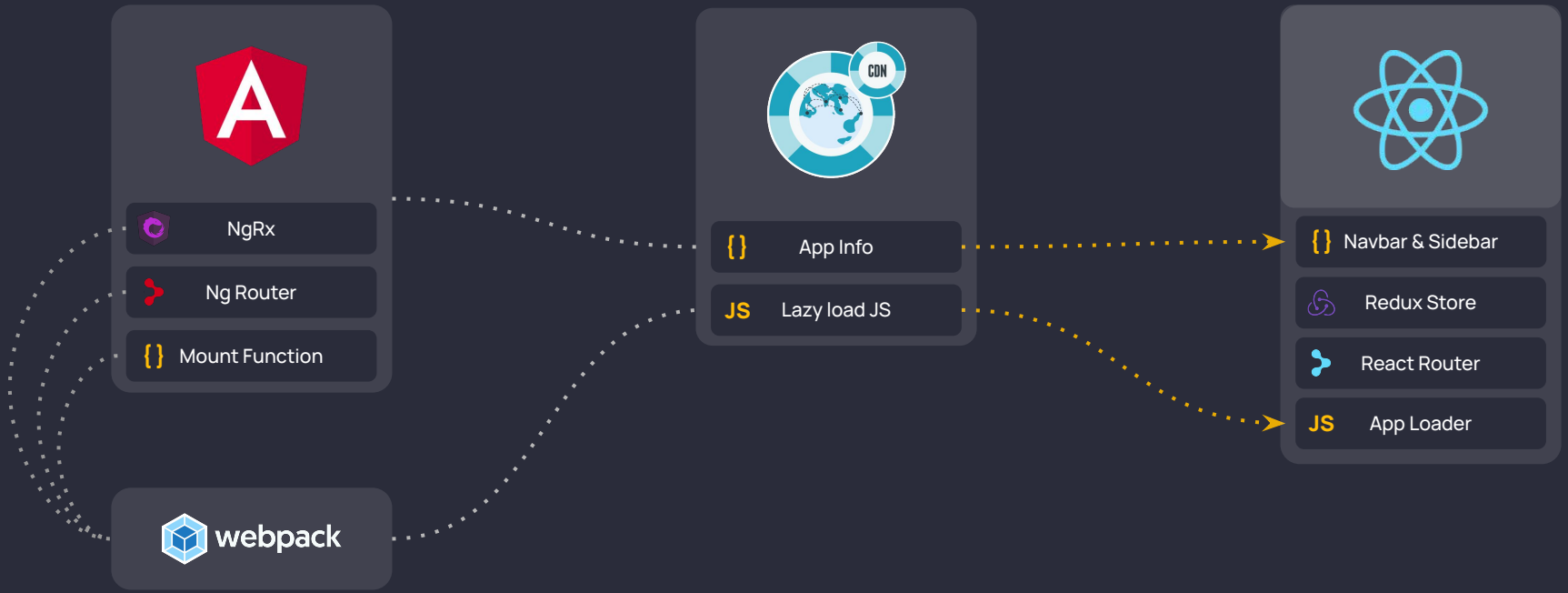
frontegg

Vision



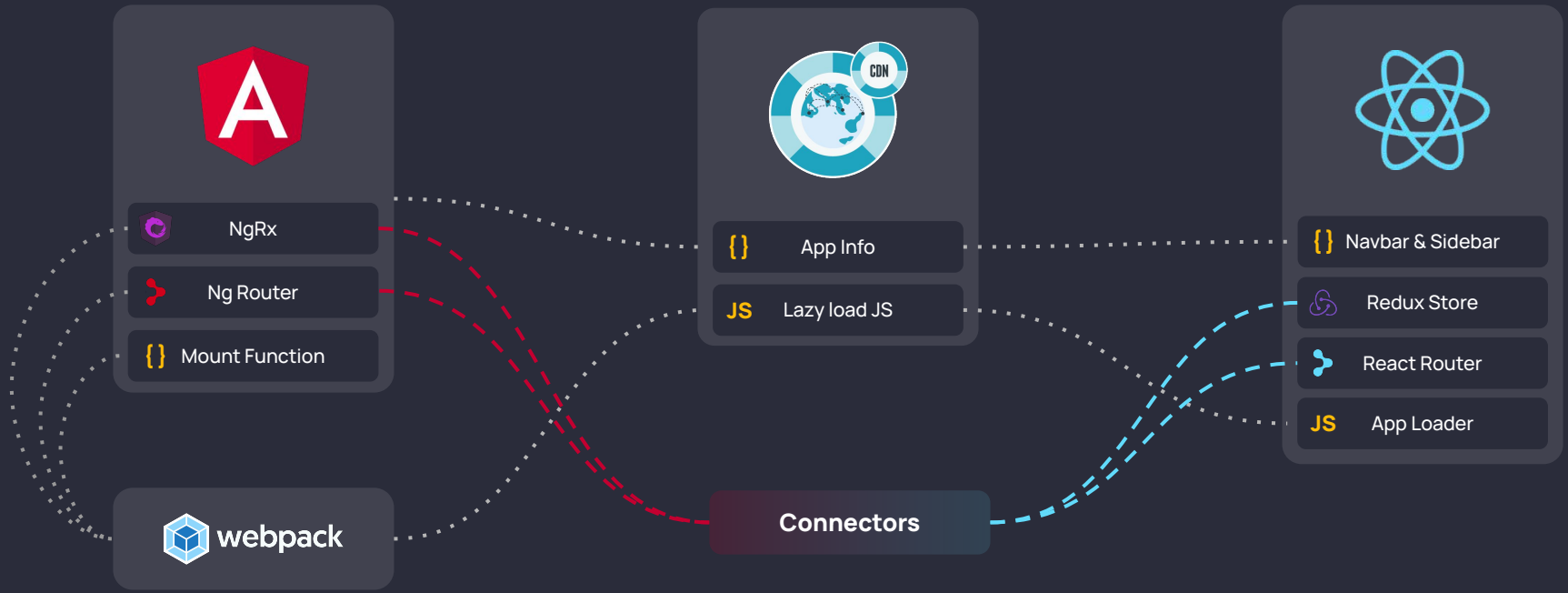
frontegg

Vision



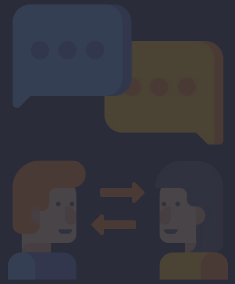
frontegg

Vision



frontegg

Limitations

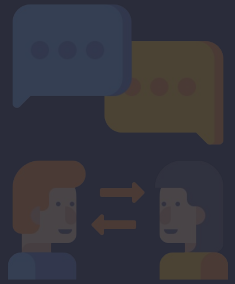


frontegg

Limitations

Shadow DOM

Media-Query Factor



frontegg

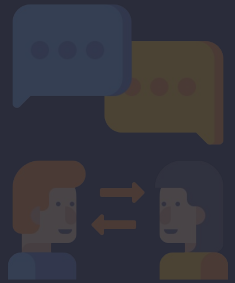
Limitations

Shadow DOM

Media-Query Factor

Code-Splitting

Lazy-Loading



frontegg

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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

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

public init(appModule: Type<any>) {
  window.Apps = {
    ...window.Apps,
    [APPLICATION_ID]: {
      id: APPLICATION_ID,
      mount: this.createMountFunction(appModule),
      unmount: this.destroy
    }
  };
}
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);
  }
}

/** ... */
```

Step 2/5

Make it injectable application

- Create Singleton AppContext
- Initialize method
- Create Render Element and Mount functionality
- Use AppContext.init

frontegg

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

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

/** Replace this */
// platformBrowserDynamic().bootstrapModule(appModule)

/** With: */
AppContext.init(AppModule)
```

Step 3/5

Router Connection

- Add BASE_HREF provider

frontegg

```
// 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

frontegg

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

import AppContext from '../AppContext';

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


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

 ngOnInit(){
   AppContext.router.subscribe( this.routeListener )
 }

}
```

Step 3/5

Router Connection

- Add BASE_HREF provider
- AppContext router use-case

frontegg

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

import AppContext from '../AppContext';

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


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

 ngOnInit(){
   AppContext.router.subscribe( this.routeListener )
 }

}
```

Step 4/5

Store Connection

- Create Injectable Service

frontegg

```
// file: ./app/app.store-connector.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

frontegg

```
// 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

frontegg

```
// 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

frontegg

```
// 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

frontegg

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

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

frontegg

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

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

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

Demo

David Antoon

@davidantoon

Download the slides & working example