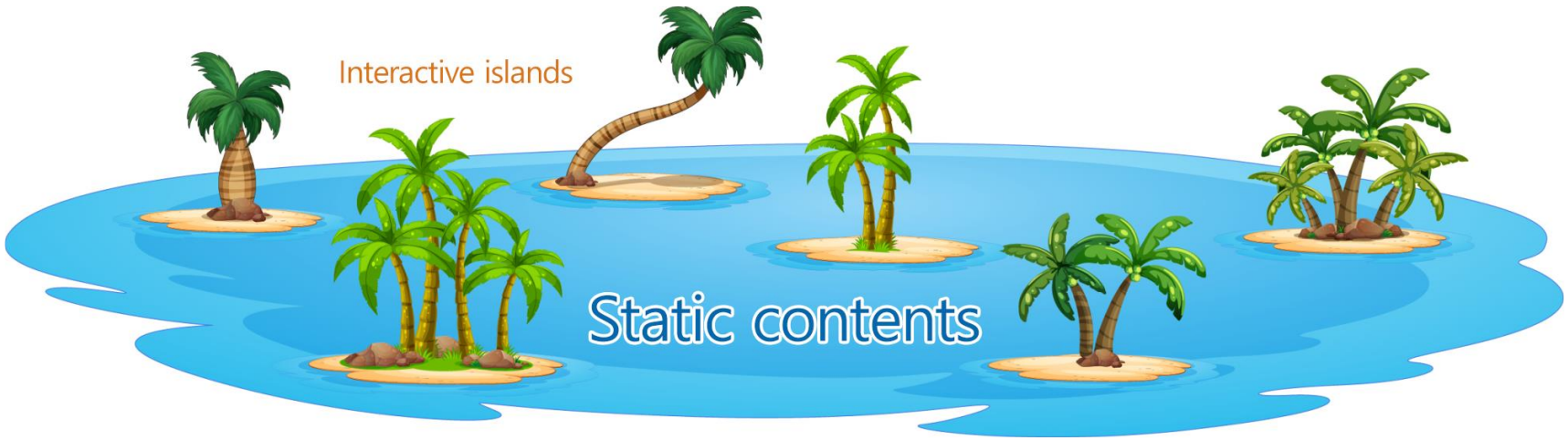
아일랜드 아키텍처 (Islands Architecture)

조새나라

Islands Architecture?

소프트웨어 아키텍처 패턴 중 하나로서
독립적인 모듈들을 섬에 비유하고 이러한 모듈들이 연결되는 인터페이스를 바다로 비유하여

각 모듈들은 서로 독립적으로 개발되고 배포되면서 이를 인터페이스 통신을 통해 상호작용 할 수 있는 환경을 제공하여
전체적인 아키텍처 구조를 갖추는 것이 아일랜드 아키텍처이다.

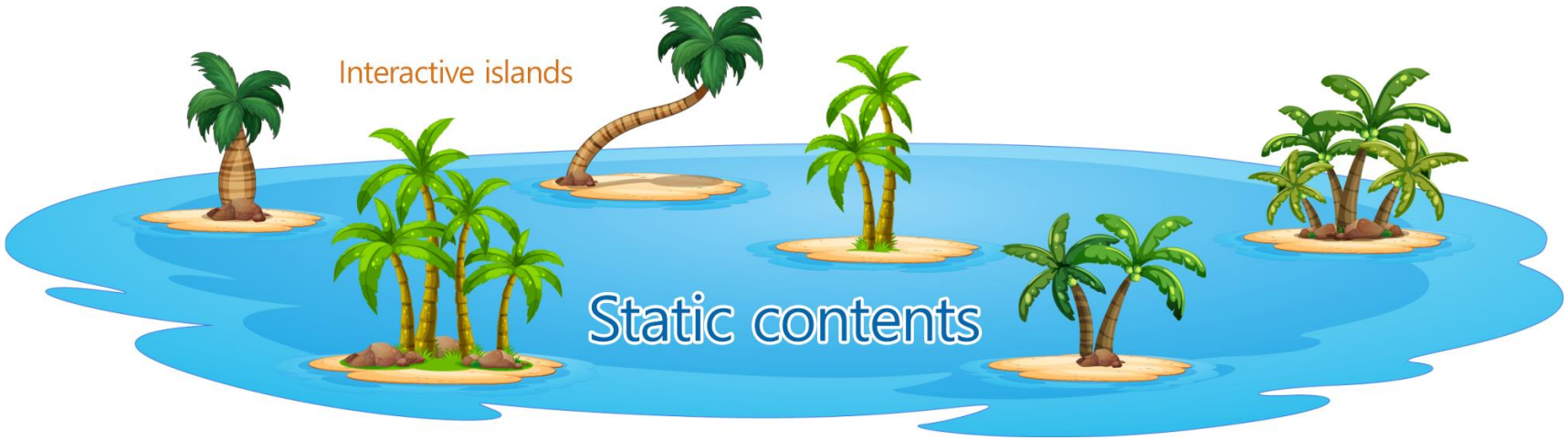


Islands Architecture?

좀 더 쉽게 설명하자면 동적인 독립적인 UI 컴포넌트들이 섬에 해당되며
자체적으로 상태를 관리하고 클라이언트와의 상호작용을 처리한다.

그리고 이러한 동적인 요소를 제외한 모든 정적인 컨텐츠들은 바다 영역이 되며 여기에 정적인 HTML이 해당된다.

이에 알 수 있듯 클라이언트에 전송되는 JS 코드를 최소화하는 것이 아일랜드 아키텍처의 궁극적인 목적이다.



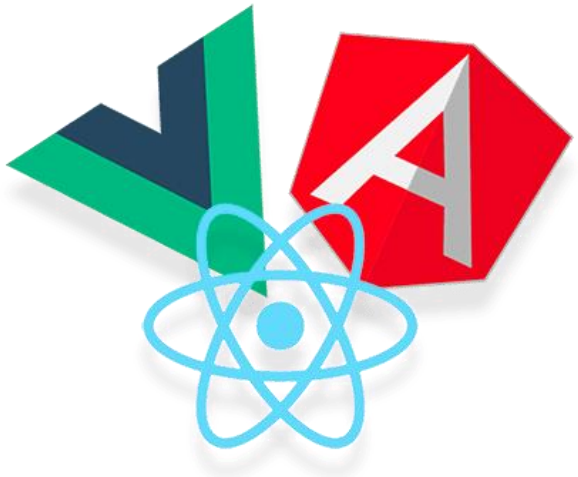
Islands Architecture?

또한 독립적이란 표현에 맞게 아일랜드 아키텍처의 대표적인 구현체인
Astro에서는 섬 영역에 해당하는 각각의 컴포넌트들이 자유로운 프레임워크를
사용할 수 있도록 멀티 프레임워크 환경을 제공하고 있다.

이는 역시 각각의 컴포넌트들이 독립적이기 때문에 상태가 겹치지 않아
여러 프레임워크를 동시에 사용하는데 아무런 제약이 없기 때문이다.

단 아일랜드 아키텍처 = 멀티 프레임워크 지원이라는 명제는 지나친 비약일 수 있다.
아키텍처 상으로 지원하는데 문제가 없다 뿐이지 이를 위해선
Astro와 같이 멀티 프레임워크들을 빌드하여 번들링 해줄 구현체가 필요하다.

따라서 아일랜드 아키텍처 기반의 프레임워크가 이러한 기능을 만약 제공하지 않는다면
본인이 직접 인터페이스를 구현하지 않으면 불가능하다.



Islands Architecture?

아일랜드 아키텍처를 페이지 구조로 치환하여 보면 다음과 같다.

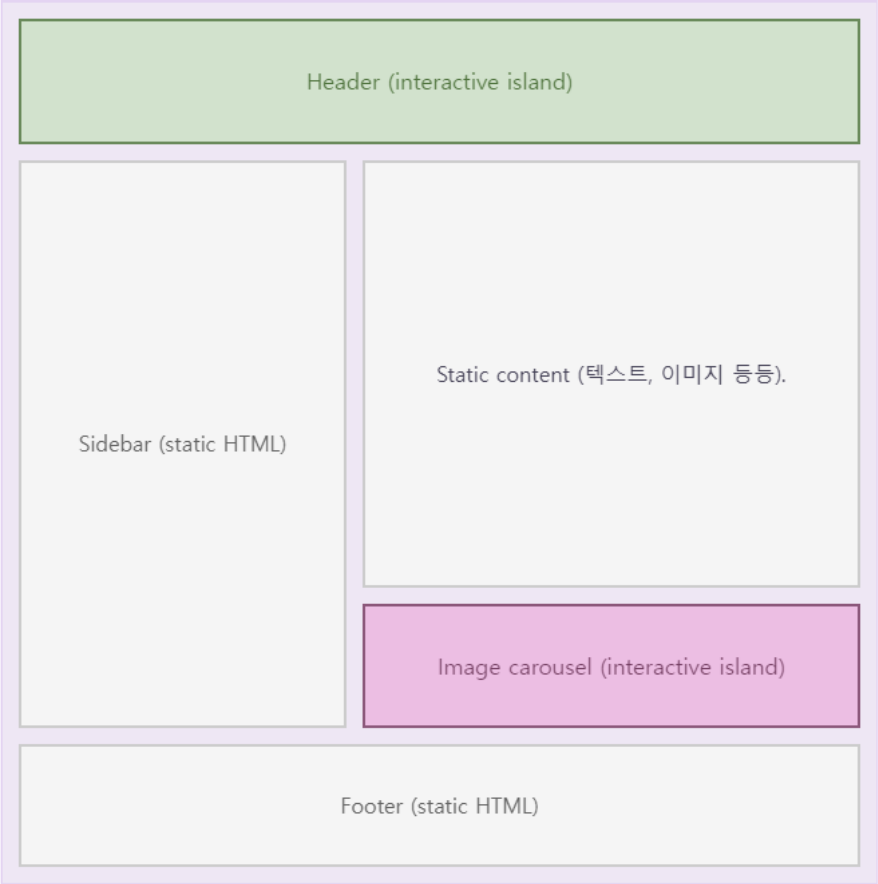
상호작용하는 영역들인 헤더와 이미지 캐러셀은 섬에 해당하며
그 외 나머지 영역들은 바다에 해당하게 된다.

이러한 섬과 바다의 요소들은 CSR과 SSR의 중간 개념인
정적 사이트 생성(Static-Site-Generation, 이하 SSG)의 원리로 작동 하게 된다.

기존의 SSG와 조금 다른 것은 바다에 해당하는 영역들 뿐만 아니라
섬에 해당하는 영역 모두 빌드 타임 시 최대한 정적 HTML으로 변환하게 된다.

이때 섬 영역에서는 최대한 자바 스크립트 코드를 제거하여
클라이언트와 상호작용에 필요한 최소한의 코드만이 남게 될 것이다.

이 덕에 SEO와 페이지 로드 속도가 획기적으로 향상된다.



Islands Architecture?

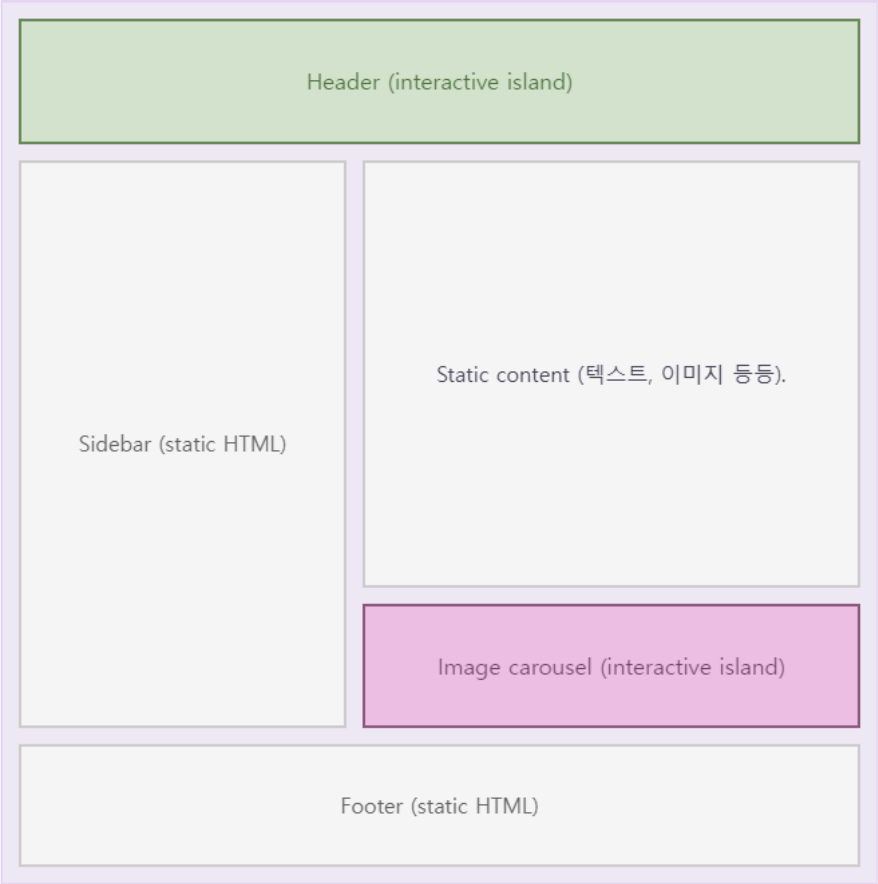
단순히 SSG만이 아일랜드 아키텍처만의 장점이 아니다.

SSG는 Next.js나 Nuxt 같은 여타 프론트엔드 프레임워크의
서드파티 라이브러리들이 가진 공통적인 장점들이다.

아일랜드 아키텍처는 보다 획기적으로 비용을 절약하여
페이지 로드 속도를 향상하는데 초점을 맞추고 있다.

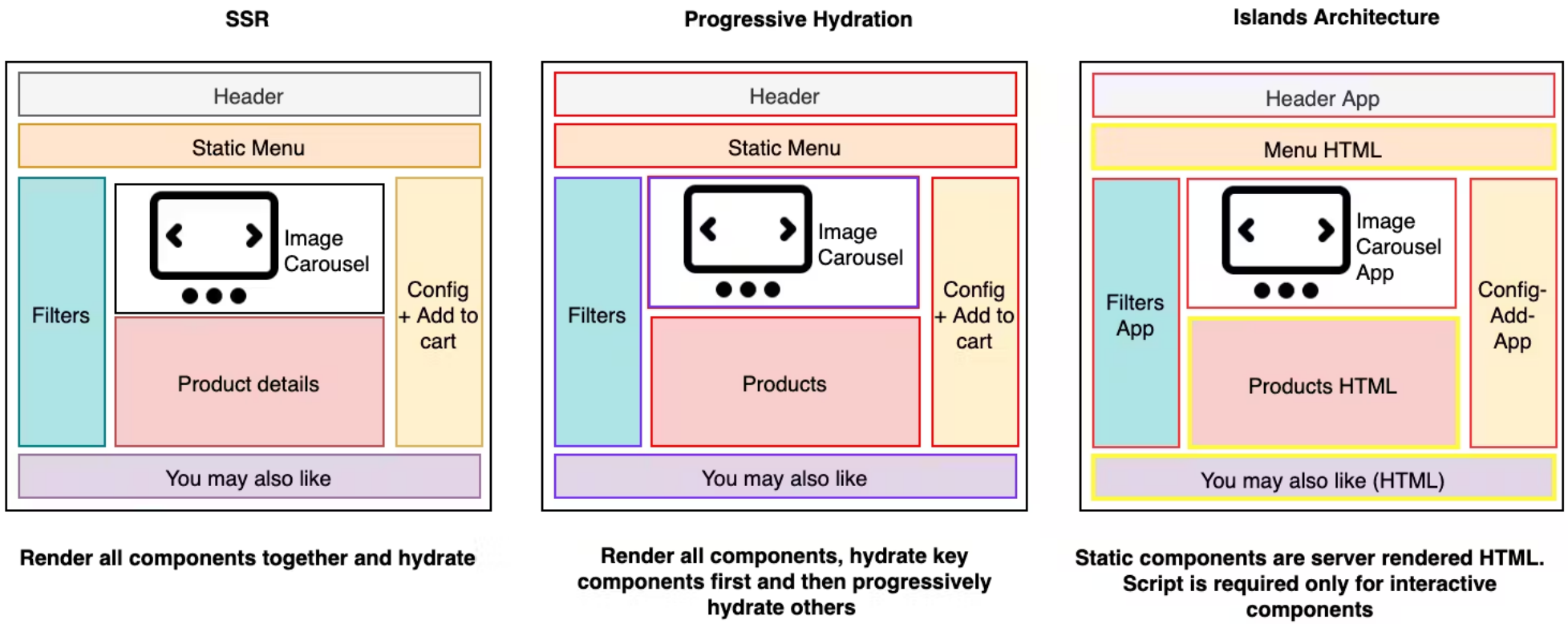
섬이라는 표현과 같이 루트의 개념이 없이 독립적이기에
하향식 렌더링이 필요하지 않으며 각 섬의 성능 문제는 다른 섬에 영향을
끼치지 않는다.

이러한 특징으로 하이드레이션(Hydration)에 비교적 자유로워지며
이는 역시 JS코드의 절약으로 이어지며 성능 향상과 결부된다.



SSR vs Progressive Hydration vs Islands Architecture

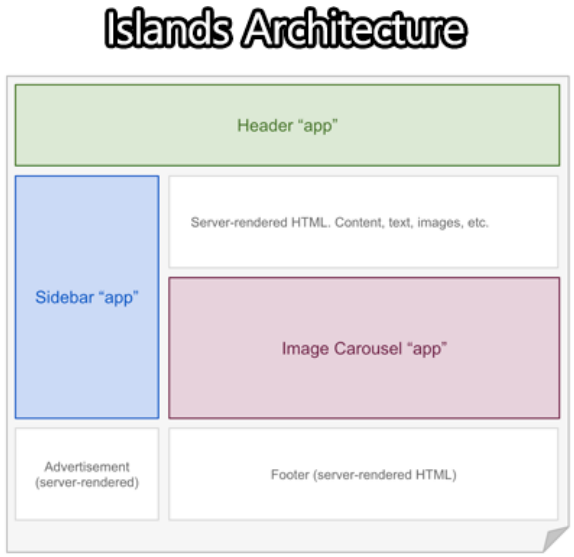
SSR, 점진적 하이드레이션, 아일랜드 아키텍처는 미리 페이지를 렌더링 한다는 공통점이 있지만
자세히 들여다 보면 하이드레이션 방면으로 아래와 같은 유의미한 차이점이 있다.

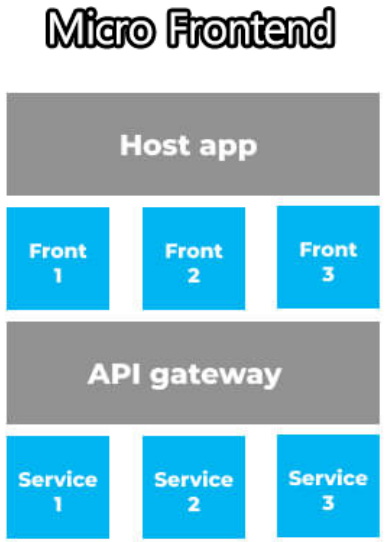


Differences from Micro Frontend

하나를 독립적인 단위로 나누는 것이라는 부분에서
마이크로 프론트엔드 아키텍처와 아일랜드 아키텍처의 큰 골조는 비슷하다 볼 수 있다.

다만 아일랜드 아키텍처는 보다 성능적인 측면에서 접근하여 단위를 HTML로서 구성하는 것으로 좁혀서 보고 있고,
마이크로 프론트엔드는 기능적인 측면에서 접근하여 모노리스한 앱을 기능적 단위의 모듈로 나누는 것으로 보다 넓게 보고 있다.





결론

아일랜드 아키텍처는 빠르게 변화하는 프론트엔드 생태계에서도 거론 되기 시작한 지 그리 오래되지 않은 개념이다.
그럼에도 불구하고 비슷한 개념인 마이크로 프론트엔드처럼 빠르게 발전하고 자리잡고 있는 것으로 보이며

특히나 프론트엔드에서 늘 상 거론되는 SSR과 CSR 사이의 간극에서 발생하는 하이드레이션과 관련된 이슈에서는
주목 받고 있는 방법 중 하나로 거론되고 있으며 이를 기반으로 개발되는 최신 프레임워크들도 늘어나는 추세이다.

물론 여러 단점들도 자리잡고 있다.

상호작용이 복잡해질 수록 하이드레이션을 위한 JS코드가 늘어날 것이고 또한 이를 처리하기 위해 많은 섬들이 생겨날 것이다.
이는 아일랜드 아키텍처의 장점이 퇴색될 수 있다는 얘기이다.

그리고 개발자들로 하여금 가장 민감한 부분인 이를 다루는 개발자 커뮤니티가 협소한 것과 이를 지속적으로 주도하고
관리하는 주체가 불분명한 것이 있다.(FE 오픈 소스에서 막대한 영향력을 끼치는 Vercel이 후원하고는 있음)

그럼에도 불구하고 잘 설계된 아키텍처가 가지는 장점은 프론트 개발자로 하여금 매력을 느끼기엔 충분하다고 볼 수 있다.

감사합니다

- 참조 링크

https://jasonformat.com/islands-architecture/

https://www.patterns.dev/posts/islands-architecture

https://docs.astro.build/ko/

https://dev.to/this-is-learning/why-efficient-hydration-in-javascript-frameworks-is-so-challenging-1ca3

https://velog.io/@composite/%EC%95%84%EC%9D%BC%EB%9E%9C%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B3%90