# Haupterkenntnisse

## Island Architecture

Die Island Architecture sorgt im Vergleich zu aktuell etablierten Frameworks wie z. B. Next.js dafür, dass eine Website eine deutlich geringere Menge an JavaScript-Code enthält. Next.js gehört zu den Meta-Frameworks, d. h. es ist ein Framework, das andere Frameworks miteinander kombiniert und um Funktionalitäten ergänzt (<https://prismic.io/blog/javascript-meta-frameworks-ecosystem>). Im Fall von Next.js baut dieses auf dem Frontend Framework bzw. der Frontend Bibliothek React auf und ergänzt es um Funktionalitäten wie u. a. Server Side Rendering. Websites, die mit Next.js implementiert werden, können sowohl Client als auch Server Komponenten nutzen. Während der Inhalt von Server Komponenten statisch ist und bleibt, können Client Komponenten interaktives Verhalten bereitstellen. Ein Beispiel einer Client Komponente ist eine Schaltfläche, bei dessen Anklicken das entsprechende Produkt zum Warenkorb hinzugefügt wird.

Bei der Verwendung von Server Side Rendering rendern Meta-Frameworks wie Next.js sämtliche Komponenten auf dem Server, wobei dieser statisches bzw. nicht-interaktives HTML erzeugt. Im Browser rendert das Frontend Framework die Client Komponente anschliessend ein zweites Mal. Dabei fügt sie der nicht-interaktiven Client Komponente Event Listeners hinzu und initialisiert den Client State. Dieser Vorgang nennt sich Hydration und macht aus einer statischen eine interaktive Komponente. Daten, die das Framework für die Hydration im Browser benötigt, bezieht sie aus der HTML-Seite. Beim Server-seitigen Rendering hat der Server die Daten für sämtliche Client Komponenten im JSON-Format in die HTML-Seite eingefügt. Meta-Frameworks wie Next.js fügen Daten sowohl für Server als auch Client Komponenten der HTML-Seite im JSON-Format hinzu. Diese gleiche Behandlung der Komponenten-Typen hat folgende Bedeutung:

* Daten, die in einer Client Komponente angezeigt werden, sind in der HTML-Seite doppelt enthalten: Als Resultat des Server-seitigen Rendering enthält die Seite die Daten der Client Komponente in den entsprechenden HTML-Elementen. Ein zweites Mal sind die Daten im JSON-Format in der HTML-Seite enthalten. Das Frontend Framework benötigt diese für die Hydration der Client Komponente.
* Bei Websites mit vielen statischen bzw. Server Komponenten wächst die HTML-Seite unnötig an. Die Menge des unnötigen Wachstums beläuft sich auf die Summe der in die HTML-Seite gerenderten Daten, die für das Server-seitige Rendering von Server Komponenten verwendet worden sind, im Browser jedoch kein interaktives Verhalten erhalten.
* Für Websites, die mit Next.js umgesetzt sind, führt der Framework-Code im Browser die Hydration aller Komponenten beim Laden der Seite durch. Der Framework-Code unterscheidet nicht, ob sich eine Komponente im sichtbaren Bereich der Seite befindet oder der Anwender erst später z. B. durch Scrollen, sie in den sichtbaren Bereich bewegt.

Frameworks, die den Island-Architecture-Ansatz umsetzen, verringern gegenüber Websites, die mit dem etablierten und populären Meta-Framework Next.js umgesetzt sind, die Menge an JavaScript Code. Diese Reduktion erreichen Frameworks, die den Island-Architecture-Ansatz anwenden, weil sie die Hydration von Inseln - die Bereiche einer Website, die interaktives Verhalten bereitstellen - unabhängig voneinander durchführen können. So kann etwa das Framework Astro die Hydration einer Insel verzögern, bis sie in den sichtbaren Bereich gelangt. Erst wenn eine Komponente im sichtbaren Bereich erscheint, fordert der Browser vom Server den Code der Komponente an.

Ein zweiter Vorteil ist die geringere Grösse von HTML-Seiten, weil sie nur die Daten für Client Komponenten im JSON-Format enthalten, die während der Hydration im Browser interaktives Verhalten erhalten. Frameworks wie Astro und Marko erreichen diese Reduktion mit ähnlichen Mitteln: Während bei Astro Inseln explizit mittels Client-Direktive als solche definiert werden, prüft der Marko Compiler zur Build-Zeit, bei welchen Komponenten es sich um Client Komponenten handelt. Nur für Komponenten, die bei Astro explizit als Inseln definiert sind bzw. die der Marko Compiler als Client Komponenten erkennt, rendert der Server die notwendigen Daten im JSON-Format in die HTML-Seite.

## Web Components

Web Components ist eine Sammlung von aktuell vier Technologien, die die Implementierung von wiederverwendbaren Frontend Komponenten ermöglichen. Zu den vier Technologien gehören die HTML Custom Elemente, das Shadow Document Object Model eines jeden Web Components, JavaScript-Module nach dem aktuellen Standard sowie HTML Templates. Seit Januar 2020 werden laut dem Mozilla Developer Network alle vier Technologien von sämtlichen gängigen Browsern unterstützt (<https://developer.mozilla.org/en-US/docs/Web/API/Web_components#browser_compatibility>). Web Components sind Teil der Web APIs. Im Vergleich zu Komponenten, die mit Frontend Frameworks wie React und Vue umgesetzt sind, benötigen Web Components keine zusätzlichen Bibliotheken und Frameworks. Sie sind Framework-agnostisch und können in zahlreichen Frontend Frameworks eingesetzt werden. Verglichen mit React- oder Vue-Komponenten bedeutet das, dass eine Website, die ausschliesslich Web Components nutzt, um die Grösse des Codes des Frontend Frameworks leichter wird. Web Components tragen mit ihrer Browser-weiten Unterstützung dazu bei, die Menge an JavaScript Code, die eine Website benötigt, zu verringern. Sie erreichen damit mit einem anderen Mittel dasselbe Ziel wie Frameworks, die den Island-Architecture-Ansatz anwenden.

Mit React- und Vue-Komponenten teilen Web Components Eigenschaften wie den Lebenszyklus, begonnen bei deren Initialisierung, über den Zeitpunkt, zu dem die Komponente dem Document Object Model hinzugefügt bzw. wieder entfernt wird und sich Werte von Eingabe-Eigenschaften ändern.

## Micro Frontend

Unter dem Begriff Micro Frontend ist ein Architektur-Muster zu verstehen, das die Idee von Micro Services, wonach ein Monolith in separate Komponenten aufgeteilt wird, auf die Frontend-Welt anwendet. Die Aufteilung eines Frontend-Monolithen in mehrere kleinere Frontends ermöglicht u. a., dass Entwicklungsteams diese weitgehend unabhängig voneinander entwickeln und deployen können. Tätigkeiten wie der allfällige Datenaustausch sowie das Einbinden von mehreren Micro Frontends in einer Website erfordern allerdings die Abstimmung unter den jeweiligen Teams.

# Aspekte im Fokus im Abschlussbericht

Der Abschlussbericht wird sich auf folgende Aspekte fokussieren:

* Wie es zu der aktuellen Situation gekommen ist, in welcher etablierte und populäre Frameworks wie Next.js mehr Code an den Client sendet, als für die Bereitstellung von interaktivem Verhalten notwendig ist.
* Wie Frameworks, die den Island-Architecture-Ansatz anwenden, die Menge an JavaScript Code senken. Anhand zweier Prototyen, konkret Astro und Marko, die den Island-Architecture-Ansatz anwenden, können die Unterschiede im HTML-Markup einer Seite sowie der Menge an JavaScript Code, die der Browser lädt, beziffert werden. Der Vergleich erfolgt mit einem Prototypen, der mit Next.js umgesetzt ist.
* Die Verwendung von Web Components, die mit der JavaScript-Bibliothek namens Lit umgesetzt sind, zeigen im mit Astro umgesetzen Prototypen auf, wie wiederverwendbare Frontend-Komponenten, die nicht mit einem Frontend Framework umgesetzt sind, die Menge an JavaScript Code weiter senken.
* Wie das Framework Astro, das es ermöglicht Frontend Komponenten unterschiedlicher Frontend Frameworks zu integrieren, Ähnlichkeiten hat mit Micro Frontends.