Fresh: Das next-gen Framework

Halima Ali
University of Kassel, Software Engineering Research Group,
Department of Computer Science and Electrical Engineering,
Wilhelmshöher Allee 73, 34121 Kassel, Germany
ali.halima@outlook.de

ABSTRACT

Für Webprogrammierer ist es wichtig, das richtige JavaScript -Framework zu wählen, welches nicht nur die Anforderungen ihres aktuellen Webprojekts erfüllt, sondern ihnen auch hilft schnell und einfach skalierbare und sichere Webanwendungen zu erstellen. Das Entwicklerteam hinter "Fresh" behauptet, genau diese Anforderungen zu erfüllen. Es stellt sich nun die Frage, inwiefern dies stimmt. Der Umfang dieser Arbeit besteht darin, eine gründliche Funktions- und Leistungsbewertung des neuen JavaScript-Frameworks bereitzustellen. Dabei wird Fokus auf die besondere Island-Architektur, die wichtigsten Funktionen, wie Routing, und die Performance gelegt. Die Nutzung von Fresh wird ebenfalls anhand einer selbst programmierten Anwendung gezeigt.

Fresh zeigt, dass die Leistungsvorteile und Funktionen es zu einer attraktiven Option für Entwickler auf der Suche nach sicheren und effizienten Webanwendungen machen. Eine generelle Entscheidung für Fresh kann jedoch nicht ausgesprochen werden, da die Ansätze zu anderen Framework unterschiedlich sind. Das Framework bringt Vor- und Nachteile mit, die vor der Entwicklung einer Webanwendung abgewogen werden sollten.

1. EINLEITUNG

In der heutigen digitalen Welt spielt die Entwicklung von Webanwendungen eine immer bedeutendere Rolle. Dabei sind Frameworks zu unverzichtbaren Werkzeugen geworden, um Webseiten schnell und einfach zu realisieren. Gegenstand dieser Arbeit ist es, das Fullstack Framework Fresh genauer zu untersuchen. Im Rahmen dieser Untersuchung wird aufgezeigt, welche Funktionen das Framework besitzt, um die entscheidende Frage zu beantworten, ob Fresh als "next-gen" Framework bezeichnet werden kann.

Im Verlauf dieses Papers wird zunächst ein Überblick über JavaScript Frameworks gegeben. Anschließend wird die Entstehungsgeschichte des Frameworks vorgestellt. Dann wird eingehend die Architektur veranschaulicht und dabei wird insbesondere auf die besondere Island-Architektur Fokus gelegt. Im weiteren Verlauf werden die Funktionen von Fresh, wie das Routing, näher betrachtet und einen detaillierten Einblick in dessen Möglichkeiten geben. Um die Ergebnisse der Untersuchungen praxisnah zu veranschaulichen, wird anschließend eine Beispielanwendung in Form einer Blogwebseite vorgestellt, bei denen die verschiedenen Funktionen von Fresh zum Einsatz kommen. Schließlich wird das Fazit die gewonnenen Erkenntnisse zusammenfassen und die zentrale Forschungsfrage beantworten.

2. EINFÜHRUNG IN JAVASCRIPT FRAMEWORKS

Um die Frage zu beantworten, ob Fresh wirklich das "next-gen" Framework ist, muss erst geklärt werden, was ein Framework ist. Dieses Kapitel dient als grundlegender Einstieg in JavaScript Frameworks. Es werden kurz die Kernkonzepte von HTML, CSS, JavaScript und TypeScript erläutert. Diese grundlegenden Webtechnologien bilden die Basis für das Verständnis der Funktionsweise von Fresh.

2.1 HTML

Die Sprache, welche die Grundbausteine einer jeden Webseite darstellt, ist HTML. HTML steht abgekürzt für Hypertext Markup Language. Die Sprache legt die Struktur und Darstellung einer Webseite fest. Durch die Verwendung von HTML können verschiedene Elemente wie Texte, Bilder, Links und andere Elemente eingefügt werden, um die Webseite nach individuellen Anforderungen zu gestalten [3, 14]. Ein einfaches Beispiel für HTML-Code wäre der folgende:

```
<h1>Hello World!<h1/>
```

Dieser Code würde die erste Überschrift einer Website erstellen. Um die visuelle Gestaltung und Interaktivität der Webseite weiter zu verbessern, können HTML Seiten durch die Anwendung von CSS und JavaScript erweitert werden.

2.2 CSS

CSS, abgekürzt für Cascading Stylesheets, ist eine Sprache zur Gestaltung von Webseiten, die als Stylesheet-Sprache bekannt ist. Sie dient dazu, das Erscheinungsbild von HTML-Elementen auf einer Webseite anzupassen. Mithilfe von CSS können verschiedene Eigenschaften wie Schriftart, Farben, Layout und mehr verändert werden. Dadurch ist CSS ein unverzichtbares Werkzeug, um attraktive und benutzerfreundliche Webseiten zu erstellen [14, 5]. Ein an-

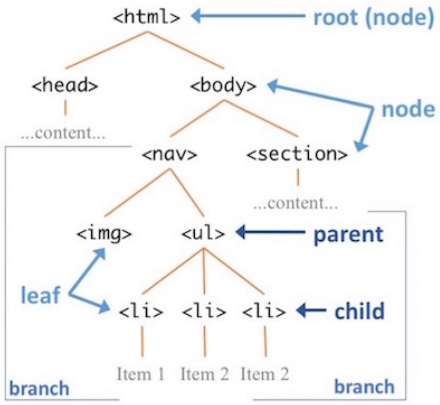

Figure 1: DOM Baum

schauliches Beispiel für CSS ist die Verwendung von CSS Selektoren, um die Hintergrundfarbe oder Schriftart von Überschriften zu ändern.

2.3 JavaScript

JavaScript ist eine sehr wichtige Programmiersprache im Bereich der Webentwicklung [14]. JavaScript wird oft in Verbindung mit HTML genutzt. Vor der Nutzung von JavaScript haben Webseiten, die nur aus reinem HTML bestehen, ein statisches Layout. Die Sprache sorgt für die Interaktivität der Webanwendungen. JavaScript ermöglicht es, dynamische Inhalte, Animationen und andere interaktive Elemente auf einer Webseite zu erstellen, um die Benutzererfahrung zu verbessern [14, 6, 8]. Um zu verstehen, wie JavaScript auf die verschiedenen Elemente der Oberfläche zugreifen kann, wird im Folgenden das Document Object Model (DOM) vorgestellt.

DOM ist eine baumartige Darstellung des Inhalts einer Webseite – ein Baum aus „Knoten“ mit unterschiedlichen Beziehungen, je nachdem, wie diese im HTML Dokument angeordnet sind [7]. Solch ein Baum ist in Figure 1[1]zusehen. Um die Knoten „ansprechen“ zu können, werden Selektors genutzt, d.h. die Knoten haben einen Ansprechnamen. Jeder Knoten ist ein Objekt, welches verschiedene Attribute und Methoden hat. Diese Attribute und Methoden sind die primären Werkzeuge, um mit JavaScript die Webseite zu manipulieren [2, 7].

2.4 Fullstack Frameworks

Ein Fullstack-Framework ist eine Art von Software-Plattform, die es Entwicklern ermöglicht, sowohl den Teil einer Anwendung zu erstellen, den die Benutzer sehen und mit dem sie interagieren (Frontend), als auch den unsichtbaren Teil, der im Hintergrund arbeitet und Daten verarbeitet (Backend)[13]. Das bedeutet, dass man mit einem Fullstack-Framework die komplette Anwendung bzw. Webseite von Anfang bis Ende bauen kann, ohne auf verschiedene separate Tools für unterschiedliche Aufgaben zugreifen zu müssen[13].

[1]Bildquelle: https://info340.github.io/dom.html

3. ENTSTEHUNG UND ZWECK VON DENO FRESH

Im Rahmen seines Vortrags “10 Things I Regret About Node.js” hat Ryan Dahl, der Schöpfer von Node.js, die Entwicklung von Deno angekündigt [17]. Deno ist eine Alternative zu Node.js und eine Laufzeitumgebung für JavaScript und TypeScript. Bei Deno handelt es sich um eine völlig neue Implementierung, die auf modernen JavaScript Funktionen basiert und alle vom Ersteller erwähnten Fehler behebt [36]. Angesichts der Unterschiede zwischen Deno und Node stellt sich heraus, dass ein spezialisiertes Framework benötigt wird. An dieser Stelle tritt Fresh in Erscheinung, ein Framework, das speziell für diese Bedürfnisse entwickelt wurde. Am 28. Juni 2022 wurde die Veröffentlichung der ersten Version, Fresh 1.0, angekündigt [4]. Diese erste Version legte den Grundstein für die weitere Entwicklung des Frameworks. Am 18. Juli 2023, erfolgte die Veröffentlichung der neusten Version 1.3 [12].

4. ARCHITEKTUR

Im kommenden Kapitel steht die gründliche Analyse der Architektur von Fresh im Mittelpunkt. Es werden die zentralen Konzepte wie das Rendering-Verfahren, die Inselarchitektur, JSX und TypeScript eingehend erläutert.

4.1 Just-in-Time-Renderung und -Builds

Fresh verfolgt den Ansatz, Webseiten so schnell wie möglich dem Benutzer zur Verfügung zu stellen[4]. Dazu macht das Framework von dem Konzept ”Just in Time” (JIT) Gebrauch.

Serverseitiges Rendering ist ein Konzept in der Webentwicklung, bei dem die Anwendung die HTML-Seiten auf dem Server rendert. Das bedeutet, dass der Großteil der Webseite bereits auf dem Server gerendert wird und als statischer HTML-Code an den Client geschickt wird [15, 22]. Der Prozess läuft folgendermaßen ab: Der Webbrowser stellt eine Anfrage an den Server und dieser antwortet sofort mit der fertig gerenderten Seite. Dieses Verfahren ist auch als JIT-Rendering bekannt und bietet eine Technik, bei der das serverseitige Rendern einer Webseite nur dann erfolgt, wenn der Client danach fragt[16]. Dadurch wird die Ladezeit weiter reduziert, da der Server nur das rendert, was für die aktuelle Anforderung benötigt wird[16].

Die Entwickler von Fresh haben bewusst auf separate Build Schritte wie Bundling verzichtet, stattdessen setzt Fresh auf JIT-Builds[16]. Bundling ist ein Prozess, bei dem ein sogenannter ”Bundler” eine Liste der Abhängigkeiten für einen Einstiegspunkt im Code erstellt. Von diesem Punkt aus arbeitet der Bundler rückwärts, um alle Abhängigkeiten zu finden. Anschließend werden all diese Abhängigkeiten zu einer einzigen Ausgabedatei zusammengefügt, die dann an den Browser übertragen werden kann. Bei größeren Projekten kann dieser Prozess länger dauern, sodass es zu Verzögerungen im Laden der Webseite kommen kann[16]. Das Rendern einer Webseite mit Fresh hingegen ähnelt dem Laden einer einfachen Internetseite. Bei Just-in-Time-Builds werden URLs für das Laden der Abhängigkeiten genutzt, d.h. für alle Importe ruft die gerenderte Seite diese Adressen auf, um den notwendigen Code zu laden[16].

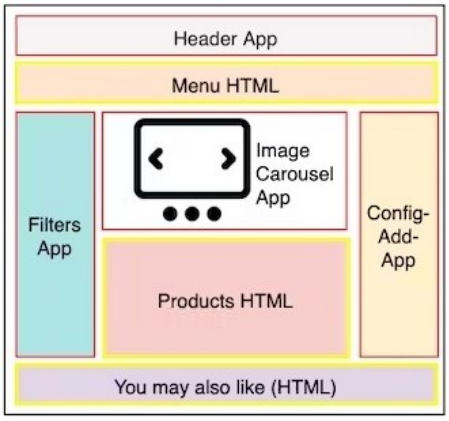

Figure 2: Island Architectur

4.2 Islands Architektur

Fresh nutzt grundlegend das Islands Architecture Muster [4]. Jason Miller beschrieb dieses Muster bereits 2020 in seinem Blogbeitrag zur Island-Architektur [19].

Im Allgemeinem besteht die Island-Architektur aus zwei Hauptkomponenten: den statischen und den dynamischen Bereichen, auch Inseln genannt. Das Figure 2[2] zeigt eine grafische Darstellung der Island-Architektur. Es verdeutlicht die klare Differenzierung der genannten Insel-Typen. Die statischen Bereiche einer Webseite bestehen aus reinem, nicht-interaktivem HTML-Code und erfordern keine besondere Behandlung [20]. Sie können als das Grundgerüst der Seite betrachtet werden, welches sich nicht verändert und keine Interaktion erfordert. In der Abbildung sind die statischen Bereiche, das Menü, die Produktdarstellung und die Fußleiste der Webseite [20]. Im Gegensatz dazu bestehen die dynamischen Bereiche einer Webseite aus einer Kombination von HTML- und JavaScript-Code. Diese Inseln sind Bereiche der Seite, die eine höhere Interaktivität erfordern, beispielsweise Benutzereingaben oder das Bild-Karousel in Figure 2[20].

Bei Fresh wird die Island-Architektur folgendermaßen eingesetzt: Fresh Webseiten liefern standardmäßig nur reines HTML an den Client aus. Dieses statische HTML-Dokument enthält mehrere separate eingebettete Anwendungen, nämlich die dynamischen Inseln. Jede dieser Inseln stellt ein Preact Component dar[22]. Preact ist eine Alternative zu React, die nur 3kb groß ist. Durch die Nutzung von Preact will Fresh dafür sorgen, dass nur die minimalste Menge an JavaScript genutzt wird [18].

Anstatt die gesamte Seite mit JavaScript neu zu rendern, kann der Client selektiv nur die Inseln aktualisieren, dieser Prozess wird auch Rehydration genannt [22]. Jede Komponente verfügt über ihr eigenes Hydratisierungsskript, das asynchron und unabhängig von anderen Skripten auf der Seite ausgeführt wird. Dies bedeutet, dass jegliche Vorgänge in einer Komponente keinen Einfluss auf andere Bereiche der Webseite haben. Dieser Aspekt erweist sich als nützlich, insbesondere, wenn in einer Komponente Performanceprobleme auftreten sollten. Der Vorteil dieser Herangehensweise liegt darin, dass der Client nur für die Darstellung und Aktualisierung derjenigen Teile der Seite verantwortlich ist, die eine Interaktion erfordern. Inhalte, die rein statisch sind und keine clientseitige Skript-Logik erfordern, sind hingegen sehr leichtgewichtig. Da für diese statischen Inhalte keine zusätzlichen Script Ressourcen heruntergeladen oder ausgeführt werden müssen, kann die Leistung der Anwendung verbessert und die Ladezeit reduziert werden.

[2]Bildquelle: https://www.patterns.dev/posts/islands-architecture

4.3 JSX und TypeScript

JSX (JavaScript XML) spielt eine entscheidende Rolle im Ansatz von Fresh zur Erstellung deklarativer UI-Komponenten[4]. JSX ist eine Erweiterung von JavaScript, die erlaubt HTML-ähnliche Syntax direkt in den JavaScript-Dateien zu schreiben. Das Ziel ist es, die “Rendering Logik und Markup zusammen am selben Ort zu haben”, um die Komplexität und das Debuggen des Codes zu vereinfachen [33].

Fresh nutzt neben JavaScript auch TypeScript [28]. TypeScript ist eine Erweiterung von JavaScript, die statische Typisierung und zusätzliche Funktionen wie Klassen und Schnittstellen unterstützt. Es wurde von Microsoft entwickelt, um die Entwicklung großer Anwendungen in JavaScript zu erleichtern. TypeScript bietet die Möglichkeit, Anwendungen zu strukturieren und Fehler frühzeitig zu erkennen. Durch die Verwendung von der Sprache wird der Code robust und leichter zu warten [1].

5. FUNKTIONEN

Dieses Kapitel gibt Einblick in die Umsetzung von Routing, Middleware, Datenverwaltung und Deployment in Fresh.

5.1 Routing

Bevor jedoch weiter auf das Routing System in Fresh eingegangen wird, ist es zunächst erforderlich, die Konzepte der Routes vorzustellen.

5.1.1 Routes und Handlers

In Fresh beschreiben Routes, wie eine Anfrage bei einem bestimmten Pfad verarbeitet werden soll [30]. Dabei gibt es zwei Hauptkomponenten: den Handler und die Komponente. Der Handler ist ein generischer Begriff, der Code bezeichnet, der nach einer Anfrage an eine spezifische URL sucht, beispielsweise ”/login”, und oft auch nach einem bestimmten HTTP-Verb, wie z.B. POST. Der Handler sucht nach spezifischem Code, um diese präzise URL und das Verb zu verarbeiten[35, 30]. Jede Route hat einen zugehörigen Handler, der eine Funktion ist und bei jeder Anfrage an die Route aufgerufen wird. Der Handler muss eine Antwort zurückgeben, die dann an den Client gesendet wird [30]. Handler können auch verwendet werden, um API-Routen zu erstellen. Es gibt zwei Formen von Handlern: Entweder deckt der Handler alle HTTP Methoden ab oder einen Handler pro HTTP Methode. Standardmäßig wird, wenn keine Custom Handler definiert sind, ein Standard Handler verwendet, der einfach die Seitenkomponente rendert. Die Seitenkomponente ist die Vorlage für eine Seite in Fresh. Sie wird als JSX-Element auf dem Server gerendert. Bei der Rendierung erhält die Seitenkomponente übergebene Props,

Table 1: Routing Patterns

	Dateinamen	Route Muster	Passender Pfad
1	index.ts	/	/
2	blog/index.ts	/blog	/blog
3	about.ts	/about	/about
4	blog/bar.ts	/blog/bar	/blog/bar
5	blog/[foo].ts	/blog/:foo	/blog/hello
6	blog/[foo]/bar.ts	/blog/:foo/bar	/blog/hello/bar

die von ihr verwendet werden können, um genau zu bestimmen, was gerendert werden soll[30].

5.1.2 File-Based-Routing-System

In diesem Kontext bezieht Routing sich auf den Prozess der Weiterleitung eingehender HTTP-Anfragen an die entsprechenden Routes bzw. an den Code, der die Anfrage bearbeiten kann [31, 10]. In Fresh erfolgt diese Weiterleitung basierend auf dem URL-Pfad. Ein zentrales Konzept ist das "File-Based-Routing-System", bei dem Dateinamen verwendet werden, um zu bestimmen, über welche Route eine bestimmte Anfrage bearbeitet werden soll [31]. In der Regel werden diese Dateien im "routes/"-Ordner in dem Projekt abgelegt. Die Dateinamen in diesem Ordner folgen bestimmten Routenmustern, die es ermöglichen, den gewünschten Code zur Bearbeitung der Anfrage zu identifizieren [31]. Dieses Muster wird in Table 1 gezeigt. In den ersten beiden Beispielen wird dargstellt, dass Dateien mit dem Namen *index.ts* zum Wurzelverzeichnis des Ordners weitergeleitet werden. Das dritte und vierte Beispiel zeigen, dass der Pfad von verschachtelten Datein in Abhängigkeit von dem Verzeichnispfad erstellt wird. Fresh ermöglicht auch die Nutzung von Seiten mit dynamischen Routen. In der Tabelle, in Zeile fünf, ist die Datei *../blog/[foo].ts* unter den URLs *blog/hello*, *blog/goodbye*, *blog/1* usw. erreichbar. In dem letzten Beispiel wird auch die Möglichkeit der Kombination von verschachtelten Dateien und dynamischen Routen gezeigt.

5.2 Middleware

Middleware im Routing ist eine entscheidende Zwischenschicht innerhalb eines Routing-Frameworks, die zwischen dem Eingang einer Anfrage und ihrer Verarbeitung positioniert wird. Es fungiert als eine Art Filter, das spezifische Anweisungen ausführt, bevor die Anfrage an die eigentliche Zielroute weitergeleitet wird[32]. Fresh macht großen Gebrauch von diesem Prinzip[29].

Einer der Hauptfunktionen von Middleware besteht darin, zusätzliche Funktionen in den Prozess der Anfrageverarbeitung einzufügen. Das bedeutet, dass Middleware verschiedene Aufgaben erledigen kann, wie das Aktualisieren von Sitzungsdaten oder sogar das Umleiten von Anfragen an bestimmte Endpunkte. Dadurch erhält der Entwickler die Flexibilität, den Anfragenverlauf anzupassen [32].

Ein weiterer wichtiger Aspekt betrifft die Authentifizierung und Autorisierung von Benutzern. Durch Middleware kann eine Anwendung Benutzer authentifizieren und ihre Berechtigungen überprüfen, bevor ihnen der Zugriff auf bestimmte Routen gewährt wird. Dies gewährleistet, dass nur autorisierte Benutzer auf geschützte Bereiche der Webseite zugreifen können und stellt somit die Sicherheit und Integrität sicher [32].

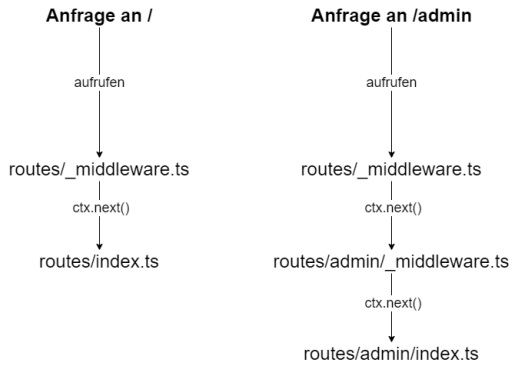

Figure 3: least specific first

Im Fall von Fresh werden Middlewares in der Datei "routes/_middleware.ts" definiert. Jede Middleware erhält eine "next"-Funktion im "context"-Argument, die verwendet wird, um nachgeordnete Handler auszulösen. Der "ctx" (Kontext) enthält auch eine "state"-Eigenschaft, die genutzt werden kann, um beliebige Daten an nachgelagerte oder vorgelagerte Handler zu übergeben [29].

Ein weiterer Vorteil von Middleware in Fresh ist die Möglichkeit, diese zu stapeln. Das bedeutet, dass mehrere Middlewares nacheinander angewendet werden können, um verschiedene Aufgaben und Operationen in der Reihenfolge ihrer Stapelung auszuführen[29]. In Figure 3 wird gezeigt, in welcher Reihenfolge die Middlewares aufgerufen werden. Wenn die URL "/admin" aufgerufen wird, erfolgt zunächst der Aufruf der Middleware in der obersten Schicht, in unserem Fall routes/_middleware.ts. Diese Middleware ruft dann wiederum die Middleware in der nächsttieferen Schicht, routes/admin/_middleware.ts, auf. Fresh bezeichnet dieses Prinzip als "am wenigsten spezifisch zuerst" (Englisch: "least specific first"), umschrieben wird damit, dass die weniger spezifische Middleware zuerst angewendet wird.

5.3 Datenverwaltung

In realen Projekten ist oft notwendig, dynamische Daten aus verschiedenen Quellen abzurufen, wie beispielsweise aus Dateien auf der Festplatte oder über APIs von externen Servern.

Diese Datenabfragen erfolgen nicht direkt während des Renderns, sondern werden durch die zuvor beschriebenen asynchronen Handler Funktionen realisiert[23]. Die Daten werden in den Handler-Funktionen der entsprechenden Routen geladen und anschließend an die Seitenkomponenten übergeben. Die Seitenkomponente kann dann auf diese Daten über das data-Feld in den props zugreifen[26]. Die Seite wird also erst gerendert, wenn alle benötigten Daten vorliegen.

5.4 Deployment

Deployment bezieht sich auf den Prozess, bei dem eine entwickelte Anwendung auf einem Server oder einer Plattform bereitgestellt wird, um sie für Benutzer zugänglich

zu machen. Es umfasst alle notwendigen Schritte, um die Anwendung von der Entwicklungsumgebung auf einen Live-Server oder eine Produktionsumgebung zu übertragen[34].

Fresh bietet den Vorteil, dass es auf jeder Plattform deployt werden kann, die Deno basierende Web Server ausführen können, wie Deno Deploy[25]. Deno Deploy ist eine Plattform, die eine weltweit verteilte Edge-Laufzeitumgebung nutzt, um die Anwendungen zu hosten. Dabei wird die Demo Site hochgeladen und von Deno Deploy auf globalen Edge-Knotenpunkten bereitgestellt [24]. Diese Knotenpunkte befinden sich an verschiedenen Standorten weltweit, um eine optimale Performance und Zuverlässigkeit für die Benutzer zu gewährleisten. Die Plattform ermöglicht eine einfache Bereitstellung von Fresh Webseiten auf der Plattform, ohne dass komplexe Konfigurationen erforderlich sind. Durch die Nutzung der weltweit verteilten Edge Knotenpunkte wird die Latenz minimiert und eine optimale Performance erreicht. Die Edge-Knotenpunkte verwalten den Datenverkehr effizient und verteilen ihn auf mehrere Server, um eine hohe Skalierbarkeit und Ausfallsicherheit zu gewährleisten

6. BEISPIEL AN EINER ANWENDUNG

In dieser Beispielanwendung wird ein Authentifizierungsprozess mit Fresh präsentiert. Die Anwendung dient als unkomplizierte Blogseite, bei der sich der Benutzer erstmal registrieren muss, um Zugriff auf den Blog zu erlangen.

6.1 Eingesetzte Technologien

Für die Umsetzung dieses Projekts wurden verschiedene Technologien eingesetzt. Zum Kern der Anwendung gehören Fresh bzw. Deno, TypeScript und JSX. Als Entwicklungsumgebung wurde Visual Studio Code verwendet.

Zur Gestaltung des visuellen Erscheinungsbildes wurde Tailwind CSS verwendet. Dieses moderne CSS-Framework ermöglicht es, auf einfache Weise Benutzeroberflächen zu gestalten, indem es eine Vielzahl von vorgefertigten Styles und Komponenten bereitstellt[11]. Fresh verfügt über eine integrierte Unterstützung für das Styling mit Tailwind CSS. Die Nutzung von dem CSS-Framework muss lediglich bei der Projekterstellung angegeben werden.

Damit sich die Nutzer anmelden bzw. registrieren können, wurde SupaBase genutzt. SupaBase ist eine Open-Source-Plattform für Backend-as-a-Service, die PostgreSQL als Datenbank nutzt[37]. In diesem Projekt wurde SupaBase für den Authentifizierungsprozess verwendet, um Benutzer anzumelden und dadurch zu autorisieren. Die Plattform ist ebenfalls komplett mit Deno kompatibel[21].

Die Webseite wurde mittels Deno Deploy online gestellt. Diese ist unter folgendem Link erreichbar:

`https://fresh-demo-blog.deno.dev`

6.2 Vorbereitung

Um Fresh nutzen zu können, muss zunächst Deno installiert werden. Hierfür wird folgender Befehl genutzt.

```
curl -fsSL https://deno.land/install.sh | sh
```

Um ein neues Fresh Projekt zu erstellen, muss der folgende Befehl ausgeführt werden, wobei `blog` der Name des Projektordners ist.

```
deno run -A -r https://fresh.deno.dev blog
```

Dieser Befehl sorgt dafür, dass ein minimales Projektgerüst im aktuellen Verzeichnis erstellt wird. Auf oberster Ebene befinden sich die Dateien `deno.json`, `dev.ts`, `main.ts`, `fresh.gen.ts` und `README.md`. Weiterhin gibt es die Verzeichnisse `routes`, `islands`, `static` und `components`.

In der `deno.json` Datei wird die Import Map definiert, die genutzt wird, um die Abhängigkeiten zu verwalten. Zudem sind hier Aufgabenbefehle definiert, wie beispielsweise der `run`-Befehl, um die Webseite auf dem Localhost zu starten. Der `dev.ts` Datei kommt eine besondere Bedeutung zu, da sie der Einstiegspunkt für das Projekt im Entwicklungsmodus ist. Auf der anderen Seite dient die `main.ts` Datei als Einstiegspunkt für das Projekt im Produktionsmodus. Wenn das Projekt auf Deno Deploy bereitgestellt wird, erfolgt die Verknüpfung mit dieser Datei. Die `fresh.gen.ts` Datei enthält das Manifest des Projekts und wichtige Informationen zu den verschiedenen Routen und Islands unserer Anwendung. Diese Datei wird automatisch von Fresh generiert und verwaltet.

Die Verzeichnisse `routes` und `islands` sind entscheidend für die Strukturierung des Projekts, da sie die Routen und die Islands enthalten. Im Verzeichnis `static` befinden sich statische Dateien, die ohne spezielle Verarbeitung bereitgestellt werden, wie beispielsweise Bilder. Im Ordner `components` werden alle Komponenten gespeichert, die explizit nur auf dem Server gerendert werden sollen.

6.3 Blog

In den folgenden Abschnitten werden einzelne Aspekte der Anwendung vorgestellt.

6.3.1 Inhalt der Webseite

Die Webseite besteht aus den folgenden Kernkomponenten: Die Home-Seite dient als Startseite der Anwendung und enthält einen Hinweis an die Benutzer, sich einzuloggen, um die volle Funktionalität nutzen zu können. Für die Anmeldung und Registrierung sind separate Login und Sign-up Seiten verfügbar. Die Login-Page ermöglicht es registrierten Benutzern, sich mit ihren Zugangsdaten einzuloggen, während die Sign-up Seite es neuen Benutzern ermöglicht, ein neues Konto zu erstellen. Nachdem die Benutzer sich erfolgreich angemeldet haben, erhalten sie Zugriff auf den Blog Bereich. Dieser ist einfach gehalten und enthält lediglich ein Bild-Karousel, welches verschiedene Bilder anzeigt, einen Counter und zwei Sektions über Fresh und Deno. Der User kann sich hier mit der Logout Funktion ausloggen und wird automatisch zur Startseite weitergeleitet.

6.3.2 Layout

Das Grundgerüst von jeder Seite besteht aus einem Header und einem Footer. Die beiden Komponenten sind in dem `components` Ordner angelegt. Eine Abbildung dieser Struktur befindet sich in Figure 5, in der eine Layout-Komponente mit dem Header und Footer erstellt wurde. Diese Layout-Komponente ist auf jeder Seite der Anwendung verwen-

```tsx
export default function Counter(count) {
  return (
    <div>
      <Button {/*TS Code um Zahl um 1 verringern*/}>-1</Button>
      <p>"Aktuelle Zahl"</p>
      <Button {/*TS Code um Zahl um 1 erhöhen*/}>+1</Button>
    </div>
  );
}
```

```tsx
export default function Blog() {
  const count = {/*aktuelle Zahl mit Preact setzen*/};
  return (
      <Counter count={count}> </Counter>
  );
}
```

Figure 4: Counter Island: ***links: Ausschnitt aus Counter.tsx; rechts-oben: Einsatz in blog.tsx; rechts-unten: Website Ansicht***

```tsx
<Header />
      <div>
        "Enter Children here"
      </div>
<Footer />
```

Figure 5: Layout Komponente

det, indem es als HTML-Markup verwendet wird: `<Layout>...<Layout/>`. In das Layout kann dann der individuelle Inhalt der Seite eingefügt werden. Dadurch wird eine konsistente Darstellung des Headers und Footers auf allen Seiten gewährleistet, ohne dass es zu Code Duplikaten kommt.

6.3.3 *Islands*

In der Anwendung sind drei verschiedene Inseln implementiert worden, die als interaktive Bereiche der Webseite dienen. Die erste Insel beinhaltet einen Button, der seine Farbe ändert, wenn der Mauszeiger darüber schwebt. Die zweite Insel enthält ein Bild-Karousel mit insgesamt vier unterschiedlichen Bildern. Die Benutzer können durch Klicken auf die Buttons links und rechts durch die Bilder navigieren. Alternativ können auch die vier Punkte unten am Objekt verwenden werden, um zwischen den Bildern zu wechseln. Zusätzlich ist das Bild-Karousel automatisiert, was bedeutet, dass die Bilder automatisch wechseln. Die letzte Insel ist ein einfacher Counter, der auf Knopfdruck hoch- bzw. herunterzählt. In Figure 4 ist der Code für den Counter abgebildet. Man kann erkennen, dass die Insel in der `blog.tsx` Datei wie ein reguläres Preact-Components genutzt wurde. Fresh kümmert sich automatisch darum, dass die Zahl bei Veränderung neu gerendert wird.

6.3.4 *Routes*

Auf der Top-Level-Schicht befinden sich die Dateien `index.ts`, `login.tsx`, `logout.tsx` und `signup.tsx`. Die Datei `index.ts` generiert die Vorlage für die Startseite der Anwendung. Die Datei `login.tsx` besteht aus einer Kombination aus Handler und Komponente. Die Komponente nutzt das HTML-Element `<forms>`, mit dem ein Login Button und die Eingabefelder für die E-Mail-Adresse und das Passwort erstellt werden. Die Komponente lässt ebenfalls eine Fehlermeldung anzeigen, wenn ein Error auftritt. Wenn der Benutzer auf den Login Button klickt, wird ein POST-Request an den Server gesendet, um den Benutzer einzuloggen. Dieser wird über den Handler verarbeitet[3].

Die Datei `signup.tsx` kombiniert ebenfalls Handler und Komponente. Ähnlich wie beim Login wird ein Post-Request gesendet, um den User anzumelden, der vom Handler verarbeitet wird. Die Datei `logout.tsx` enthält nur einen Handler, der dafür sorgt, dass der Benutzer ausgeloggt wird und zur Startseite weitergeleitet wird.

Im Unterordner `auth` befindet sich die Datei `blog.tsx`, die den Blog generiert. Die Seite enthält hauptsächlich statisches HTML. Nur das Bild-Karousel und der Counter stellen die einzigen interaktiven Komponenten dar.

6.3.5 *Authentifikation*

Der Authentifizierungsprozess setzt sich für neue Benutzer aus zwei Schritten zusammen. Der erste Schritt stellt die Registrierung dar. In Figure 6 wird der Post-Handler in `signUp.tsx` in vereinfachter Form dargestellt. Der definierte Handler liest zunächst die E-Mail-Adresse und das Passwort aus den Eingabefeldern aus. Anschließend wird der Benutzer mittels von SupaBase registriert[4]. Wenn der Registrierungsvorgang abgeschlossen ist, wird der Benutzer zur Login-Seite weitergeleitet. Dies wird mit der Anpassung des `headers` erreicht.

Der zweite Schritt setzt sich aus dem Login zusammen. Auch hier werden zunächst die Benutzerinformationen aus den Eingabefeldern gelesen, um den Nutzer dann mittels SupaBase anzumelden. Sobald dieser Vorgang erfolgreich war, wird ein Cookie namens ”superBase” gesetzt, der ein sogenanntes Token enthält.

In diesem Authentifizierungsprozess spielen Tokens eine entscheidende Rolle. Der Token ist eine Zeichenkette, die über SupaBase erhältlich ist, sobald der Login-Prozess erfolgreich war. In dem Projekt wird der Token verwendet, um zu überprüfen, ob ein Benutzer eingeloggt ist. Dieser wird in den Cookies des Browsers gespeichert und kann von den Route Handlern gelesen werden.

Beim Logout wird das Cookie mit dem Token gelöscht, um die aktive Sitzung zu beenden. Nach dem erfolgreichen Ausloggen wird der Benutzer zur Startseite weitergeleitet.

6.3.6 *Einsatz von Middleware*

[3] siehe Authentifikation

[4] Mehr zu Authentifikation mit SupaBase hier:https://supabase.com/docs/guides/auth

```
export const handler: Handlers = {
  async POST(req, _ctx) {
    //get form data from req
    const form = await req.formData();
    const email = form.get("email") as string;
    const password = form.get("password") as string;

    //sign-in using Supabase

     //redirect user to homepage
    const headers = new Headers();
    headers.set("location", "auth/blog");
    return /*response*/;
  },
};
```

Figure 6: vereinfachter Sign-up Handler

```
export function handler(_req, ctx) {
  //check if user is logged in, if not redirect to login-page
  if (!ctx.state.token) {
    const headers = new Headers();
    headers.set("location", "/login");
    return /*response*/
  }

  //redirect to blog
  return ctx.next();
}
```

Figure 7: vereinfachte Middleware

Um den Sicherheitsaspekt von Fresh zu verdeutlichen, kommen Middlewares zum Einsatz.

Die `/auth/_middleware.tsx`-Datei wird aufgerufen, wenn auf `auth/blog.tsx` zugegriffen werden soll. In Figure 7 ist zu sehen, dass mittels eines Handlers geprüft wird, ob der Benutzer eingeloggt ist, indem die Cookies auf das Vorhandensein eines Tokens geprüft werden. Falls kein Token vorhanden ist, wird dieser zur Login-Seite umgeleitet. Falls ein Token vorhanden ist, wird `ctx.next()` aufgerufen, welches den Benutzer zur Blogseite weiterleitet. Dies verhindert, dass Benutzer das Login umgehen können, indem diese einfach die URL `auth/blog` verwenden. Dadurch wird sichergestellt, dass nur autorisierte Nutzer Zugriff auf den Blog haben.

7. FAZIT

Die Analyse von Fresh und seiner Architektur sowie Funktionsweise zeigt, dass das Framework durchaus einige Merkmale eines ”next-gen” Frameworks aufweist.

Die Verwendung der Island-Architektur ist ein vielversprechender Ansatz. Die klare Trennung von statischen und dynamischen Bereichen ermöglicht eine gezielte Optimierung und Interaktivität der Webseiten, was wiederum dazu führt, dass die Webseite schneller geladen wird und eine verbesserte Benutzererfahrung bietet. Durch die Verwendung des File-Based-Routing-Systems ermöglicht Fresh eine übersichtliche Verwaltung der Routes und erleichtert die Wartbarkeit von Webanwendungen aus der Entwicklersicht. Das Konzept der Middleware in Fresh ermöglicht es, zusätzliche Funktionen in den Anfrageverarbeitungsprozess einzubinden, welches die Effizienz und Sicherheit der Anwendung erhöht. An der Beispielanwendung ist dies vor allem verdeutlicht worden, indem Middlewares dafür gesorgt haben, dass nur autorisierte Benutzer Zugang zu gesicherten Pfaden haben.

In der Vergangenheit gab es Unsicherheiten darüber, ob das Deno Team sich vollständig dazu verpflichtet hat, Fresh aktiv zu pflegen[27]. Doch mit der Veröffentlichung von Version 1.2 gab das Team ein klares Signal, dass es sich nun der kontinuierlichen Wartung von Fresh widmet. Hierfür ist Marvin Hagemeister zum neuen Teamleader ernannt worden[27]. Obwohl Fresh derzeit noch eine relativ kleine Community hat, zeichnet sich ein stetiges Wachstum ab. Entwickler und Entwicklerinnen entdecken zunehmend das Potenzial und die Vorzüge dieses Frameworks. Die Tatsache, dass Fresh zu den am meisten ”gestarred” Frontend-Projekten auf GitHub im Jahr 2022 zählt, ist ein klares Indiz für die wachsende Beliebtheit und Anerkennung innerhalb der Entwicklergemeinschaft[9]. Die Unterstützung und Weiterentwicklung von Fresh durch die Entwicklergemeinschaft könnte entscheidend sein, um das Framework kontinuierlich zu verbessern, um den Anforderungen eines ”next-gen” Frameworks gerecht zu werden.

Alles in allem zeigt Fresh das Potenzial, ein ”next-gen” Framework zu sein. Es bietet Entwicklern die Werkzeuge, um moderne und leistungsstarke Webanwendungen zu entwickeln. Ob Fresh letztendlich das ”next-gen” Framework wird, hängt von der weiteren Entwicklung, der Akzeptanz in der Entwicklergemeinschaft und der Fähigkeit, mit den sich verändernden Technologien mitzuhalten ab.

8. REFERENCES

[1] Typescript is javascript with syntax for types. https://www.typescriptlang.org. zuletzt besucht: 2023-05-27.

[2] Dom. https://dom.spec.whatwg.org/what, 2023. zuletzt besucht: 2023-05-26.

[3] O. Avci, R. Trittmann, and W. Mellis. *XHTML und HTML*, pages 86–138. Vieweg+Teubner Verlag, Wiesbaden, 2003.

[4] L. Casonato. Fresh 1.0. https://deno.com/blog/fresh-is-stable, 2022. zuletzt besucht: 2023-06-05.

[5] M. contributors. Css: Cascading style sheets. https://developer.mozilla.org/en-US/docs/Web/CSS, 2023. zuletzt besucht: 2023-05-25.

[6] M. contributors. What is javascript? https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript, 2023. zuletzt besucht: 2023-05-25.

[7] W. contributors. Document object model. https://en.wikipedia.org/wiki/Document_Object_Model, 2023. zuletzt besucht: 2023-05-26.

[8] W. contributors. Javascript. https://en.wikipedia.org/wiki/JavaScript, 2023. zuletzt besucht: 2023-05-24.

[9] Deno. Fresh is here to stay, 2023. zuletzt besucht: 2023-07-30.

[10] DIVPUSHER.COM. What is: Routing.

https://divpusher.com/glossary/routing/: :text=Routing 2023. zuletzt besucht: 2023-06-25.

[11] A. Fitzgerald. Tailwind css: What it is, why use it examples. https://blog.hubspot.com/website/what-is-tailwind-css, 2022. zuletzt besucht: 2023-07-27.

[12] M. Hagemeister. Fresh 1.3 – simplified route components and more. https://deno.com/blog/fresh-1.3, 2023. zuletzt besucht: 2023-07-20.

[13] A. Ivanovs. The 10 best full-stack web frameworks. https://stackdiary.com/full-stack-web-frameworks/: :text=Full-Stack%20Platformtext=js%20offers%20a2023. zuletzt besucht: 2023-06-29.

[14] D. Jackson. Html vs. css vs. javascript: What's the difference? https://brytdesigns.com/html-css-javascript-whats-the-difference, 2019. zuletzt besucht: 2023-05-25.

[15] A. Jiang. A gentle introduction to islands. https://deno.com/blog/intro-to-islands, 2023. zuletzt besucht: 2023-06-19.

[16] A. Jiang. You don't need a build step. https://deno.com/blog/you-dont-need-a-build-step, 2023. zuletzt besucht: 2023-07-16.

[17] JSConf. 10 things i regret about node.js - ryan dahl - jsconf eu. https://www.youtube.com/watch?v=M3BM9TB-8yAab_channel=JSConf, 2018. zuletzt besucht: 2023-06-05.

[18] Mikołaj. What is preact and when should you consider using it? https://www.merixstudio.com/blog/what-preact-and-when-should-you-consider-using-it/, 2020. zuletzt besucht: 2023-06-11.

[19] J. Miller. Islands architecture. https://jasonformat.com/islands-architecture/, 2020. zuletzt besucht: 2023-06-11.

[20] Patterns.dev. Islands architecture. https://www.patterns.dev/posts/islands-architecture. zuletzt besucht: 2023-06-11.

[21] D. team. supabase-js in deno. https://deno.land/x/supabase@v2.24.0, 2023. zuletzt besucht: 2023-07-27.

[22] the fresh authors. Architecture. https://fresh.deno.dev/docs/concepts/architecture, 2023. zuletzt besucht: 2023-06-19.

[23] the Fresh authors. Data fetching. https://fresh.deno.dev/docs/concepts/data-fetching, 2023. zuletzt besucht: 2023-07-10.

[24] the fresh authors. Deploy to production. https://fresh.deno.dev/docs/getting-started/deploy-to-production, 2023. zuletzt besucht: 2023-07-01.

[25] the fresh authors. Deployment. https://fresh.deno.dev/docs/concepts/deployment, 2023. zuletzt besucht: 2023-07-01.

[26] the Fresh authors. Fetching data. https://fresh.deno.dev/docs/getting-started/fetching-data, 2023. zuletzt besucht: 2023-07-10.

[27] the Fresh authors. Fresh is here to stay. https://fresh.deno.dev/docs/getting-started/fetching-data, 2023. zuletzt besucht: 2023-07-10.

[28] the fresh authors. Introduction, "https://fresh.deno.dev/docs/introduction". https://fresh.deno.dev/docs/introduction, 2023. zuletzt besucht: 2023-05-27.

[29] the fresh authors. Middleware. https://fresh.deno.dev/docs/concepts/middleware, 2023. zuletzt besucht: 2023-06-30.

[30] the fresh authors. Routes. https://fresh.deno.dev/docs/concepts/routes, 2023. zuletzt besucht: 2023-06-25.

[31] the fresh authors. Routing. https://fresh.deno.dev/docs/concepts/routing, 2023. zuletzt besucht: 2023-06-25.

[32] the Laravel authors. Middleware. https://laravel.com/docs/10.x/middleware, 2023. zuletzt besucht: 2023-06-30.

[33] the React authors. Writing markup with jsx. https://react.dev/learn/writing-markup-with-jsx, 2023. zuletzt besucht: 2023-05-27.

[34] the umbraco authors. What is deployment? https://umbraco.com/knowledge-base/deployment/. zuletzt besucht: 2023-07-01.

[35] the Vercel authors. Route handlers. https://nextjs.org/docs/app/building-your-application/routing/router-handlers, 2023. zuletzt besucht: 2023-06-26.

[36] M. Tyszkiewicz. Fresh: an extremely fast and simple web framework from deno. https://blog.graphqleditor.com/fresh-js, 2022. zuletzt besucht: 2023-06-05.

[37] U. G. Woke. Firebase and supabase: Key differences you need to knows. https://www.red-gate.com/simple-talk/development/other-development/firebase-and-supabase-key-differences/: :text=Supabase%20is%20a%20real-time,and%20managing%20your%20authentication%20server., 2023. zuletzt besucht: 2023-07-27.