Adventskalender

Inhalt

[Ausgangslage 3](#_Toc182224706)

[Frontend: 3](#_Toc182224707)

[Backend 3](#_Toc182224708)

[Datenbank 3](#_Toc182224709)

[Projektaufbauorganisation 4](#_Toc182224710)

[Verwendete Standards und Methoden 4](#_Toc182224711)

[Informieren 5](#_Toc182224712)

[Next.js und Remix 5](#_Toc182224713)

[Datenbanken basierend auf dem Framework 5](#_Toc182224714)

[Planung 6](#_Toc182224715)

[Requierements 6](#_Toc182224716)

[Zeitplan 8](#_Toc182224717)

[Entscheidungen 9](#_Toc182224718)

[Next.js vs. Remix 9](#_Toc182224719)

[Rendering 10](#_Toc182224720)

[Datenbank 10](#_Toc182224721)

[Step by Step Guide (Realisieren) 11](#_Toc182224722)

[Projekt Erstellen 11](#_Toc182224723)

[Hauptseiten erstellen 11](#_Toc182224724)

[Übersicht der Seiten 11](#_Toc182224725)

[Backend Funktionen 11](#_Toc182224726)

[Firebase setup 12](#_Toc182224727)

[Funktionen 12](#_Toc182224728)

[Cashing 12](#_Toc182224729)

[Navigation with User 13](#_Toc182224730)

[Testen (Kontrollieren) 13](#_Toc182224731)

[Zusammenfassung (Auswertung) 14](#_Toc182224732)

[Informationen 14](#_Toc182224733)

[Glossar 14](#_Toc182224734)

[Abbildungsverzeichniss 14](#_Toc182224735)

[Literaturverzeichnis 14](#_Toc182224736)

# Ausgangslage

Das Projekt soll nach IPERKA erarbeitet werden.

Die Webseite soll aus Frontend und Backend in einem Full-Stack React Projekt bestehen.

Die Idee dieses Projektes soll diese Komponente enthalten:

### Frontend:

Für den Adventskalender sollen, falls genügend Zeit vorhanden ist, Animationen eingebaut werden.

Es soll folgende Seiten enthalten:

* Login
  + Mit Email und Passwort
* SignUp
  + Email, Passwort und UserName
* Calendar
  + Namen anzeigen als Titel
  + 24 Törchen -> nur am entsprechenden datum anzeigen

### Backend

Als Backend soll ein Full-Stack React Backend verwendet werden. Dazu kommen Next.Js und Remix in frage und müssen evaluiert werden.

Das Backend soll auch mit Tests abgedeckt werden.

### Datenbank

Für die Datenbank soll eine non-relational Datenbank verwendet werden.

Hier kommt vorallem Firebase in frage, jedoch soll abgeklärt werden, ob es andere gibt, welche besser zu dem gewählten Backend passen.

### Projektaufbauorganisation

Ein Bild, das Text, Visitenkarte, Screenshot enthält.

Automatisch generierte Beschreibung

Da es sich um ein Einzelprojekt handelt, übernehme ich alle Rollen, Aufgaben und Verantwortlichkeiten. Im Detail bedeutet das:

- Projektleiter: Verantwortlich für die Planung, Organisation und Steuerung des Projekts.

- Entwickler: Zuständig für die technische Umsetzung des Projekts, einschliesslich der Implementierung von Frontend und Backend sowie des Designs.

- Tester: Veranwortlich für das Durchführen von Tests zur Qualitätssicherung und Fehlererkennung.

- Dokumentationsverantwortlicher: Dokumentation aller Schritte gemäss IPERKA und Erfüllung der Anforderungen des QV-Leitfadens.

Für die Auswertung sowie für die Überprüfung meiner Arbeit ist meine Aktuelle PA (Linda Krüger) und mein Coach (Frank Grümmert) verantwortlich, welche gleichzeitig auch mein Auftraggeber bilden.

### Verwendete Standards und Methoden

Für die Umsetzung des Projekts wurden folgende Technologien, Standards und Methoden gewählt:

- Arbeitsmethode: Das Projekt orientiert sich am IPERKA-Modell, das in die Schritte Informieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten gegliedert ist.

- Tools: Zur Entwicklungsumgebung zählen VS Code, Firebase Console und GitHub zur Versionskontrolle. Testautomatisierung wird mit Jest für Unit-Tests durchgeführt.

- Dokumentation: Das Projekt wird in Microsoft Word dokumentiert und orientiert sich an den Vorgaben des QV-Leitfadens. Entscheidende Informationen und Quellen werden im Literaturverzeichnis aufgeführt.

# Informieren

## Next.js und Remix

Um eine fundierte Entscheidung zwischen **Next.js** und **Remix** zu treffen, wurden die Vor- und Nachteile beider Frameworks auf der Seitevon **Hygraph** recherchiert. Diese Quelle bietet einen umfassenden Vergleich der Funktionen und Einsatzmöglichkeiten und diente als Grundlage für die Analyse der besten Lösung für das Projekt.

Zusätzlich habe ich mich über die Integration von **Firebase** in beide Frameworks informiert. Dabei konnte ich feststellen, dass Next.js nativ mehrere Optionen zur Datenbereitstellung unterstützt, darunter **Server-Side Rendering (SSR)**, **Static Site Generation (SSG)** und **Incremental Static Regeneration (ISR)**, was eine effiziente Handhabung von dynamischen Daten ermöglicht. Remix legt den Fokus stärker auf client-seitige Datenabfragen und eine interaktive Benutzeroberfläche. Es kann auch serverseitige Datenverarbeitung unterstützen, bietet aber keine native Unterstützung für ISR.,

Diese Recherchen dienen als Grundlage für die spätere Entscheidung und stellen sicher, dass alle relevanten Aspekte von Next.js und Remix, einschliesslich ihrer Interaktion mit Firebase, berücksichtigt wurden.

## Datenbanken basierend auf dem Framework

Eine Quelle, welche die besten Datenbanken für das jeweilige Framework veranschaulicht, habe ich leider nicht gefunden. Dafür gibt es aber einige Quellen, welche Datenbanken in frage kommen. Basierend auf den Unterschieden in den Rendering-Strategien und der Unterstützung für server- und client-seitige Datenabfragen, habe ich die jeweiligen Datenbankoptionen zugeordnet.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Framework | |  | | --- | | Empfohlene Datenbank |  |  | | --- | |  | | Begründung |
| |  | | --- | | **Next.js** |  |  | | --- | |  | | Firebase | Native Unterstützung von SSR und SSG in Next.js ermöglicht effiziente Integration mit Firebase für serverseitige Authentifizierung und Caching. Ideal für dynamische Inhalte. |
|  | |  | | --- | | MongoDB (Atlas) |  |  | | --- | |  | | MongoDB passt gut zu JavaScript-basierten Frameworks und ist ideal für grosse Datenmengen und komplexe Abfragen. API-Routen in Next.js erleichtern den Zugriff. |
| **Remix** | FaunaDB | FaunaDB ist serverlos und optimiert für client-seitige Abfragen, was gut zur client-seitigen Datenabfrage in Remix passt. Unterstützt Echtzeit-Updates und Skalierbarkeit. |
|  | |  | | --- | | Supabase |  |  | | --- | |  | | Open-Source-Alternative zu Firebase, die Echtzeit-Updates und Authentifizierung bietet. Gut für client-seitige Datenbereitstellung, die Remix bevorzugt. |

Grundsätzlich kann mann alle erwähnten Datenbanken nahtlos mit remix und auch mit next.js einbinden. Die richtigi Wahl hängt vorallem von den Anforderungen des Projektes ab. (Lee, 2024)

# Planung

## Requierements

Ein Bild, das Text, Screenshot, Design enthält.

Automatisch generierte Beschreibung

|  |  |
| --- | --- |
| UC01: **Webseite:** Registrierung | |
| Beschreibung | Als neuer Benutzer möchte ich ein Konto erstellen können, damit ich auf die Inhalte des Adventskalenders zugreifen kann.. |
| Estimation: 1h | AC01: Ein Registrierungsformular ist vorhanden, das Felder für E-Mail, Passwort, Vor- und Nachname enthält  AC02: Der Benutzer erhält eine Bestätigung, dass sein Konto erstellt wurde und wird zum Login weitergeleitet |

|  |  |
| --- | --- |
| US02: **Webseite:** Authentifizierung | |
| Beschreibung | Als registrierter Benutzer möchte ich mich mit meinen Anmeldedaten einloggen können, um Zugriff auf meine personalisierten Inhalte zu erhalten. |
| Estimation: 1h | AC01: Ein Login-Formular ist vorhanden mit Eingabefeldern für E-Mail und Passwort.  AC02: Bei erfolgreicher Anmeldung wird der Benutzer zur Adventskalender-Seite weitergeleitet.  AC03 Bei fehlerhaften Anmeldeinformationen wird eine Fehlermeldung angezeigt. |

|  |  |
| --- | --- |
| US03: **Webseite:** Adventskalender-Funktion | |
| Beschreibung | Als Benutzer möchte ich einen Adventskalender sehen, der aus 24 Türchen besteht, damit ich jeden Tag ein neues Türchen öffnen kann. |
| Estimation: 3 h | AC01: 24 Türchen sind sichtbar und nummeriert.  AC02: Jedes Türchen kann nur an seinem spezifischen Tag geöffnet werden.  AC03: Bereits geöffnete Türchen können nicht erneut geöffnet werden |

|  |  |
| --- | --- |
| US04: **Webseite:** Adventskalender-Funktion | |
| Beschreibung | Als Benutzer möchte ich eine personalisierte Nachricht oder ein Geschenk sehen, wenn ich ein Türchen öffne, damit ich jeden Tag eine neue Überraschung erhalte |
| Estimation: 2h | AC01: Nach dem Öffnen eines Türchens wird eine Nachricht oder ein Geschenk angezeigt.  AC02: Die Inhalte sind für jeden Tag individuell gestaltet. |

|  |  |
| --- | --- |
| US05: **Webseite:** Benutzeroberfläche und Design | |
| Beschreibung | Als Benutzer möchte ich eine ansprechende und einfach zu bedienende Benutzeroberfläche, um die Webseite intuitiv nutzen zu können |
| Estimation: 2h | AC01: Die Seiten sind übersichtlich gestaltet mit klaren Buttons und Menüs.  AC02: Animationen werden hinzugefügt, um das Öffnen der Türchen visuell ansprechend zu gestalten. |

|  |  |
| --- | --- |
| US06: **Backend:** Zugriffskontrolle | |
| Beschreibung | Als Benutzer möchte ich sicher sein, dass meine Daten und meine Anmeldung geschützt sind. |
| Estimation: 1.5h | AC01: Alle Anfragen und Datenübertragungen sind über HTTPS gesichert  AC02:Firebase Authentication wird verwendet, um eine sichere Anmeldung zu gewährleisten. |

|  |  |
| --- | --- |
| US07: **Backend:** Datenverwaltung und Caching | |
| Beschreibung | Als Benutzer möchte ich, dass die Seite schnell lädt, damit ich auch bei schlechter Verbindung einfach auf die Inhalte zugreifen kann |
| Estimation: 2h | AC01: Die Adventskalender-Seite ist statisch generiert (SSG) oder durch Caching optimiert (ISR).  AC02: Statische Inhalte sind gecached, sodass sie schnell geladen werden können.s |

## Zeitplan

// TODO: BILD EINFÜGEN -> EXCEL

# Entscheidungen

### Next.js vs. Remix

Next.js wurde für dieses Projekt gewählt, da es wichtige Funktionen bietet, die sich ideal für die Anforderungen der Anwendung eignen. Besonders die Hybrid-Rendering-Funktione von Next.js, einschliesslich Server-Side Rendering (SSR) und Static Site Generation (SSG), bieten eine flexible Lösung zur Optimierung von Ladezeiten und SEO. Diese Optionen ermöglichen es, Inhalte effizient zu generieren und zu cachen, was für Projekte mit häufigen Datenaktualisierungen vorteilhaft ist.

Ein weiterer entscheidender Faktor ist die Unterstützung von Incremental Static Regeneration (ISR) in Next.js, mit der statische Seiten automatisch im Hintergrund aktualisiert werden können. Remix bietet diese Funktion nicht nativ, was Next.js besser geeignet macht, um die Performance zu maximieren und die Inhalte regelmässig zu aktualisieren.

Next.js bietet eine klare Struktur für serverlose Funktionen über API-Routen, was insbesondere mit Version 13 verbessert wurde. Remix unterstützt Serverless-Funktionen, aber Next.js bietet eine optimierte Integration für die serverseitige Verarbeitung.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Beschreibung | Next.js | Gewichtung | Punkte mit Gewichtung | Remix | Gewichtung | Punkte mit Gewichtung |
| |  | | --- | | Server-Side Rendering (SSR) |  |  | | --- | |  | | 5 | 3 | 15 | 5 | 3 | 15 |
| Incremental Static Regeneration (ISR) | 5 | 3 | 15 | 2 | 3 | 6 |
| |  | | --- | | Kompatibilität mit Firebase |  |  | | --- | |  | | 5 | 3 | 15 | 4 | 3 | 12 |
| |  | | --- | | SEO-Optimierung |  |  | | --- | |  | | 5 | 2 | 10 | 5 | 2 | 10 |
| |  | | --- | | Serverless-Funktionen |  |  | | --- | |  | | 4 | 1 | 4 | 4 | 1 | 4 |
|  |  | Total: 59 | |  | Total: 47 | |

**Kriterienbeschreibung**:

* **Server-Side Rendering (SSR)**: Dieses Kriterium bewertet, wie gut das Framework SSR unterstützt, was für SEO und schnelle Ladezeiten entscheidend ist. Next.js und Remix bieten beide SSR, daher erhalten beide hohe Punktzahlen.
* **Incremental Static Regeneration (ISR)**: ISR ist eine einzigartige Funktion von Next.js, die es ermöglicht, statische Inhalte im Hintergrund zu aktualisieren, ohne dass die gesamte Seite neu generiert wird. Dies ist besonders nützlich für dynamische Inhalte, weshalb Next.js hier die volle Punktzahl erhält, während Remix diese Funktion nicht nativ unterstützt.
* **Serverless-Funktionen**: Beide Frameworks unterstützen Serverless-Funktionen, aber Next.js integriert diese nahtlos über API-Routen, was die Entwicklung vereinfacht. Daher erhält Next.js eine etwas höhere Punktzahl.
* **SEO-Optimierung**: Beide Frameworks unterstützen SEO-freundliches Rendering. Durch die flexiblen Rendering-Optionen bietet Next.js jedoch eine etwas bessere SEO-Optimierung, daher eine höhere Punktzahl für Next.js.
* **Kompatibilität mit Firebase**: Dieses Kriterium bewertet, wie gut das Framework mit Firebase arbeitet. Next.js ist aufgrund seiner verschiedenen Rendering-Optionen etwas besser mit Firebase kompatibel, weshalb Next.js eine höhere Punktzahl erhält.

**Begründung der Gewichtung**:

* **SSR** und **ISR** sind entscheidend für das Projekt, da sie die Performance und die Ladezeitoptimierung ermöglichen. Diese Kriterien haben die höchste Gewichtung, da sie für dynamische Inhalte und ein gutes Nutzererlebnis sehr wichtig sind.
* **Serverless-Funktionen** und **SEO-Optimierung** wurden mittelhoch gewichtet, da sie den Entwicklungsprozess und die Sichtbarkeit des Projekts positiv beeinflussen.
* Die **Kompatibilität mit Firebase** hat eine mittlere Gewichtung, da Firebase die Datenbanklösung ist, und eine gute Integration mit dem Framework entscheidend für den Erfolg des Projekts ist.

### Rendering

Basierend auf der Analyse und den Anforderungen des Projekts wurde entschieden, **Server-Side Rendering (SSR)** in **Next.js** zu verwenden, da es optimal zu den dynamischen Inhalten und der Firebase-Integration passt. Die Gründe dafür sind:

1. **Dynamische Inhalte und Authentifizierung**: Da Firebase für Authentifizierung und Datenverwaltung genutzt wird, stellt SSR sicher, dass personalisierte Inhalte bei jedem Seitenaufruf aktuell und sicher ausgeliefert werden.
2. **SEO und Performance**: SSR verbessert SEO und Ladegeschwindigkeit, da Inhalte direkt vom Server bereitgestellt werden.
3. **Caching und Firebase-Integration**: Die Kombination von SSR und Caching-Mechanismen wie Incremental Static Regeneration (ISR) ermöglicht eine optimale Leistung. SSR liefert dynamische Inhalte in Echtzeit, während ISR weniger dynamische Seiten regelmässig, aber nicht ständig aktualisiert.
4. **Serverless-Funktionen und API-Routen**: Next.js API-Routen bieten eine flexible, serverlose Struktur für dynamische Inhalte, was ideal ist, um im Adventskalender Inhalte gezielt bereitzustellen.

(Hidayat, 2023; Tanga, 2024)

### Datenbank

Es gibt ein paar Vor- und Nachteile jenachdem, welche Datenbank und welches Framework verwendet werden. Die Wahl der Datenbank hängt aber hauptsächlich von den Anforderungen des Projektes ab und weniger vom Framework. Da ich mich mit Firebase bereits ein wenig auskenne und es neben der Datenbank auch noch die Authentifizierung, sowie das Hosting bietet, habe ich mich entschieden, Firebase zu verwenden. Wenn man dabei noch beachtet, dass ich Next.js gewählt habe, ist diese Entscheidung sogar sehr gut, da Firebase glücklicherweise gut mit Next.js kompatibel ist.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Beschreibung** | **Gewichtung** | **Firebase** | **Punkte** | **MongoDB (Atlas)** | **Punkte** | **Supabase** | **Punkte** |
| Unterstützung für Realtime-Updates | 3 | 5 | 15 | 2 | 6 | 5 | 15 |
| Serverless-Unterstützung | 2 | 5 | 10 | 4 | 8 | 5 | 10 |
| Integration mit Next.js | 3 | 5 | 15 | 5 | 15 | 4 | 12 |
| Einfachheit der Authentifizierung | 2 | 4 | 8 | 3 | 6 | 5 | 10 |
| Erfahrung | 5 | 2 | 10 | 1 | 5 | 0 | 0 |
|  |  | Total: 58 | | Total: 40 | | Total: 47 | |

**Kriterienbeschreibung**:

* **Unterstützung für Realtime-Updates**: Dieses Kriterium bewertet, wie gut die Datenbank Echtzeit-Updates unterstützt, was wichtig für Anwendungen ist, bei denen Daten ohne Verzögerung aktualisiert werden sollen. Firebase und Supabase bieten starke Echtzeit-Updates und erhalten daher die höchsten Punkte, während MongoDB diese Funktion nicht nativ unterstützt.
* **Serverless-Unterstützung**: Hier wird bewertet, wie gut die Datenbank in serverlosen Umgebungen funktioniert. Dies ist wichtig für die Flexibilität und Skalierbarkeit des Projekts. Firebase, FaunaDB und Supabase sind serverlose Plattformen und erhalten daher die volle Punktzahl, während MongoDB serverlose Funktionen eingeschränkt unterstützt.
* **Integration mit Next.js**: Dieses Kriterium bezieht sich darauf, wie gut die Datenbank mit dem gewählten Framework Next.js integriert werden kann. Eine gute Integration erleichtert die Entwicklung und den Zugriff auf Daten. Alle Datenbanken sind grundsätzlich kompatibel mit Next.js, daher erhalten alle hohe Punktzahlen.
* **Einfachheit der Authentifizierung**: Diese Bewertung bezieht sich auf die Bereitstellung und den Komfort von Authentifizierungsfunktionen. Firebase und Supabase bieten benutzerfreundliche Authentifizierungsoptionen, weshalb sie die höchste Punktzahl erhalten, während MongoDB und FaunaDB in diesem Bereich eingeschränkter sind.
* **Community-Unterstützung**: Die Community-Unterstützung bewertet, wie gross und aktiv die Nutzerbasis und die Entwicklerressourcen für die jeweilige Datenbank sind. Eine grosse Community hilft bei der schnellen Lösung von Problemen und erleichtert den Zugang zu Dokumentation und Ressourcen. MongoDB und Firebase haben sehr grosse Communities, daher erhalten sie hier höhere Punkte.

**Begründung der Gewichtung**:

* Die **Unterstützung für Realtime-Updates** und **Integration mit Next.js** wurden höher gewichtet, da sie entscheidend für die Projektanforderungen sind. Diese Funktionen sind für die schnelle und problemlose Datenverarbeitung besonders wichtig.
* **Serverless-Unterstützung** und **Einfachheit der Authentifizierung** haben eine mittlere Gewichtung, da sie die Flexibilität und den Komfort für Entwickler und Benutzer verbessern.
* **Community-Unterstützung** hat die niedrigste Gewichtung, da sie weniger direkte Auswirkungen auf die Projektfunktionalität hat, jedoch hilfreich für zukünftige Wartung und Problemlösung sein kann.

# Step by Step Guide (Realisieren)

Diese Schritt für Schritt Anleitung, zeigt die einzelnen schritte, welche ich ausgeführt habe, um das Projekt zu realisieren.

## Projekt Erstellen

Basierend auf den gewonnenen Informationen und den Entscheidungen, welche ich getroffen habe, erstelle ich ein Full-Stack React Projekt mit Next.js.

Um dies zu verwirklichen führe ich diesen Befehl im gewünschten verzeichniss aus:

// COMMAND TO CREATE PROJECT

## Hauptseiten erstellen

Um die Seiten zu erstellen, erstelle ich drei Dateien, in welchen ich die Struktur der Seiten, ohne Funktionalität, erstelle. Das bedeutet, dass nur die Komponenten wie: Inputfelder, Buttons und das Layout der Seiten erstellt werden.

### Übersicht der Seiten

|  |  |  |
| --- | --- | --- |
| Login | SignUp | Kalender |
| * Email * Passwort * Login Button * Weiterleitung zu Kalender * Navigation zu SignUp | * Vorname * Nachname * Email * Passwort * SignUp Button * Weiterleitung zu Login | * Begrüssung * 24 Kästchen |

## Backend Funktionen

Um nun die Funktionalität zu integrieren, werden die Funktionen im Backend geschrieben.

Es gibt mehrere Möglichkeiten die verscheidenen Seiten zu generieren. Ich habe mich für // TODO: Server side rendering / client based ? entschieden.

## Firebase setup

Als nächstes installiere ich Firebase in dem Projekt.

Dazu erstelle ich erstmals ein neues Projekt in der Firebase-Console.

// Link Firebase Console

1. Login
2. Klick neu Project.
3. Select Storage
   1. Initialize Storage to store the data
4. Download the json.config file

### Funktionen

User-Objekt:

* Vorname
* Nachname
* Email
* Passwort
* SignUp
  + Die SignUp Funktion erhält als Parameter das User-Objekt.
  + Die Funktion soll den Benutzer in Firebase Registrieren.
* Login
  + Das Login erhält die Email und das gehashte Passwort.
  + Dieses Passwort muss zuerst umgewandelt und anschliessend an Firebase gesendet werden.
  + Bei Gültigen Kredentials soll die Funktion «True» zurückgeben.
* Kalender
  + Diese Funktion erhält die ID vom entsprechenden Törchen.
  + Zuerst überprüft sie, ob dieses Törchen geöffnet werden darf. Basierend auf dem Datum.
  + Danach requestet sie den Inhalt von Firebase und gibt den Inhalt zurück.
  + Zusätzlich ändert sie den Status von geschlossen zu geöffnet.

## Cashing

Die Daten, welche hinter dem Törchen angezeigt werden, sollen nur einmal abgefragt werden und danach gecashed sein.

## Navigation with User

Die Idee von der Navigation ist, dass sichergestellt wird, dass nur Benutzer welche auch eingeloggt sind, zugriff auf die Kalender Seite haben.

Falls genügend Zeit übrig ist, kann dies zu einer Role-Based Navigation erweitert werden, indem es eine Seite gibt, auf der man Kalender für andere Benutzer erstellen kann.

# Testen (Kontrollieren)

Hier werden die wichtigsten Sachen getestet, um die Funktionalität des Codes und der Benutzeroberfläche abzudecken. Darunter:

**Funktionale Tests**

- Login- und Registrierungsfunktion: Überprüfen, ob Benutzer sich erfolgreich registrieren und einloggen können und Fehlermeldungen bei falschen Anmeldedaten angezeigt werden.

- Adventskalender-Interaktion: Sicherstellen, dass die Türchen nur an den entsprechenden Tagen geöffnet werden können und die korrekten Inhalte anzeigen.

**Benutzeroberflächen-Tests**

- Design und Layout: Überprüfen, ob das Design einheitlich ist und die Elemente korrekt dargestellt werden (z. B. Animationen und responsive Darstellung).

- Navigation und Usability: Sicherstellen, dass die Navigation klar und intuitiv ist, insbesondere bei der Interaktion mit den Adventskalender-Türchen.

# Zusammenfassung (Auswertung)

# Informationen

## Glossar

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Abbildungsverzeichniss

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Literaturverzeichnis

Eze, C. (2024, 07 01). *hygraph*. Retrieved from https://hygraph.com/blog/remix-vs-next

Francois, L. (2024, 07 22). *TrackIT*. Retrieved from https://trackit.io/remix-vs-nextjx-react-framework-comparison/

Hidayat, A. (2023, 09 04). *dev.to*. Retrieved from https://dev.to/dj1samsoe/understanding-ssr-csr-isr-and-ssg-a-comprehensive-guide-add

Lee, E. (2024, 09 9). *dev.to*. Retrieved from https://dev.to/ethanleetech/best-databases-for-nextjs-applications-3ef1

Rakesh, U. (2024, 08 12). *dev.to*. Retrieved from https://dev.to/ummalla\_rakesh/remix-vs-nextjs-which-framework-should-you-choose-3a70

Tanga, M. (2024, 08 05). *strapi*. Retrieved from https://strapi.io/blog/ssr-vs-ssg-in-nextjs-differences-advantages-and-use-cases