Abschlussprojekt in der Informatikausbildung GymInf

"Einmal um die Schweiz"

Eine Webplattform für Sportliche Challenges in und ausserhalb der Schule.

Inhaltsverzeichnis

# 1. Einleitung

## 1.1 Hintergrund des Projekts

In der heutigen Arbeitswelt sind Herausforderungen wie jene, die durch Plattformen wie "bike-to-work" geboten werden, zu einer festen Grösse geworden. Obwohl es mit "bike-to-school" eine ähnliche Aktion für Schüler gibt, ist diese weniger in den Schulalltag integriert und findet vor allem auf gymnasialer Stufe kaum Beachtung. Beide Aktionen legen den Schwerpunkt auf das blosse Mitmachen. Mein Projekt hingegen betont stärker den Aspekt der Herausforderung und des Wettbewerbs, wie er insbesondere im Sport üblich ist.

Aus diesem Bedürfnis heraus entstand die Idee zu diesem Projekt. Die Grundidee stammt von Patrick Senn, Sportlehrer am Gymnasium Neufeld, der solche sportlichen Herausforderungen umsetzen wollte, aber an der fehlenden geeigneten Webplattform scheiterte. Dies führte zur Entwicklung einer neuen Webanwendung, die nicht nur sportliche Aktivitäten fördert, sondern auch technische Kompetenzen vermittelt.

Ein zentrales Anliegen meines Projekts ist es, Schülern aufzuzeigen, wie eine Webplattform von Grund auf entwickelt wird. Dabei werden die einzelnen Schritte des Entwicklungsprozesses speziell hervorgehoben und verständlich erklärt, sodass Schüler den "Wie"-Aspekt nachvollziehen können. Dies bietet eine einmalige Gelegenheit, technisches Wissen praktisch zu vermitteln und gleichzeitig das Interesse an Informatik und Webentwicklung zu fördern.

Ein besonderer Schwerpunkt liegt dabei auf dem Datenbankdesign. Ein robustes und gut strukturiertes Datenbankdesign ist entscheidend für die Funktionalität und Skalierbarkeit der Webanwendung. In diesem Projekt wird detailliert aufgezeigt, wie Datenbanken entworfen werden, um Daten effizient zu speichern und zu verwalten, wobei die Erklärungen speziell auf Schüler zugeschnitten sind.

## 1.2 Zielsetzung und Motivation

Das Hauptziel dieses Projekts ist die Entwicklung einer Plattform, die speziell auf den schulischen Kontext zugeschnitten ist. Dabei verfolge ich zwei zentrale Ziele:

Erstens möchte ich demonstrieren, wie eine solche Webplattform von Grund auf aufgebaut wird, indem ich die einzelnen Entwicklungsschritte klar und verständlich für Schüler hervorhebe. Dadurch sollen Schüler die Möglichkeit erhalten, den Prozess der Webentwicklung aktiv nachzuvollziehen und ein tieferes Verständnis für die technischen Hintergründe zu entwickeln.

Zweitens soll die eigentliche Plattform entstehen, die sportliche Aktivitäten fördert und nahtlos in den Schulalltag integriert wird. Der Fokus liegt dabei weniger auf dem äusseren Erscheinungsbild und mehr auf der Funktionalität. Durch die Betonung von Herausforderung und Wettbewerb möchte ich die Motivation der Teilnehmer steigern und langfristig eine tiefere Einbindung in den Schulkontext erreichen. So soll die Freude am sportlichen Engagement auf gymnasialer Stufe verstärkt werden.

Meine Motivation für dieses Projekt speist sich aus der Begeisterung für Sport und Wettbewerbe, die nicht nur die körperliche Fitness fördern, sondern auch den Teamgeist stärken. Gleichzeitig bietet das Projekt eine hervorragende Gelegenheit, praktische Kenntnisse in der Webentwicklung zu vermitteln. Besonders spannend ist es, Schülern zu zeigen, wie eine Webanwendung von Grund auf entwickelt wird—von der Planung über das Datenbankdesign bis hin zur Umsetzung—und dabei die technischen Aspekte verständlich zu erklären.

Ein weiterer Motivationsfaktor ist die intensive Beschäftigung mit der technischen Gestaltung eines robusten Datenbankdesigns. Da eine gut durchdachte Datenbankstruktur das Herzstück jeder erfolgreichen Webanwendung bildet, möchte ich diesen Bereich besonders hervorheben und Schülern die Grundlagen des Datenbankentwurfs näherbringen. Mit diesem Projekt will ich nicht nur sportliche Herausforderungen zugänglicher machen, sondern auch eine technische Lösung präsentieren, die als Lernplattform für weitere ähnliche Projekte dienen kann.

Insgesamt verbindet das Projekt meine Leidenschaft für Sport mit meinem Interesse an Informatik und der Freude daran, praktische Lösungen zu entwickeln. Durch die spezielle Hervorhebung der technischen Erklärungen möchte ich sowohl einen pädagogischen als auch sportlichen Mehrwert für die Schüler bieten.

# 2. Projektplanung und Umfang

Die Entwicklung einer Webanwendung erfordert eine sorgfältige Planung und eine klare Definition des Projektumfangs. Bei der Realisierung einer solchen Anwendung sind verschiedene Phasen und Aspekte zu berücksichtigen, um ein erfolgreiches Ergebnis zu gewährleisten. In diesem Abschnitt gebe ich einen Überblick darüber, **was bei der Planung und dem Umfang eines Webapplikationsprojekts alles dazugehört**, und erkläre, **wie ich diese Elemente in meinem Projekt "Einmal um die Schweiz" umsetzen werde**.

## 2. 1. Allgemeine Aspekte der Projektplanung

Bei der Projektplanung einer Webanwendung sind folgende Schritte und Überlegungen essentiell:

* **Anforderungsanalyse**: Identifizierung der Bedürfnisse und Erwartungen der zukünftigen Nutzer sowie der Stakeholder. Dies umfasst funktionale Anforderungen (was die Anwendung tun soll) und nicht-funktionale Anforderungen (wie die Anwendung sein soll, z. B. in Bezug auf Performance oder Sicherheit).
* **Konzeption und Architektur**: Entwicklung eines Gesamtkonzepts für die Anwendung, inklusive der technischen Architektur. Hier werden Entscheidungen über die verwendeten Technologien, Frameworks und Plattformen getroffen.
* **Zeit- und Ressourcenplanung**: Festlegung des Projektzeitraums, Meilensteine und des benötigten Ressourcenbedarfs, sowohl personell als auch materiell.
* **Risikoanalyse**: Identifikation potenzieller Risiken und Herausforderungen, die den Projektverlauf beeinflussen könnten, sowie die Planung von Massnahmen zur Risikominimierung.
* **Qualitätssicherung**: Definition von Standards und Methoden zur Sicherstellung der Qualität des Endprodukts, einschliesslich Teststrategien und -verfahren.

## 2. 2. Allgemeiner Projektumfang einer Webanwendung

Der Projektumfang definiert den Rahmen dessen, was das Projekt liefern wird. Dies beinhaltet:

* **Funktionalitäten**: Detaillierte Beschreibung der Features und Funktionen, die die Anwendung bieten wird.
* **Benutzeroberfläche und User Experience**: Gestaltung der Interaktion zwischen Nutzer und Anwendung, um eine intuitive und ansprechende Nutzung zu gewährleisten.
* **Datenmanagement**: Planung, **wie** Daten erfasst, gespeichert, verarbeitet und geschützt werden.
* **Sicherheit**: Massnahmen zum Schutz der Anwendung und der Nutzerdaten vor unbefugtem Zugriff und Datenverlust.
* **Skalierbarkeit und Wartbarkeit**: Berücksichtigung der zukünftigen Erweiterbarkeit der Anwendung und der Einfachheit der Wartung.

2.3 Detaillierte Anforderungsanalyse

Anforderungsanalyse:

In meinem Projekt beginne ich mit einer gründlichen Sammlung und Analyse der Anforderungen. Durch direkte Gespräche und Interviews mit Lehrern und Schülern ermittle ich die spezifischen Bedürfnisse und Erwartungen an die Plattform. Dabei geht es nicht nur darum, welche Funktionen gewünscht sind, sondern auch darum, wie diese Funktionen die Nutzer unterstützen und den Schulalltag bereichern können.

Um die Anforderungen systematisch zu erfassen und zu dokumentieren, erstelle ich ein Pflichtenheft. Das Pflichtenheft ist ein zentrales Dokument in der Softwareentwicklung, das alle funktionalen und nicht-funktionalen Anforderungen detailliert festhält. Es dient als verbindliche Vereinbarung zwischen Auftraggeber und Entwickler und bildet die Grundlage für die weitere Projektarbeit. Den Schülern erkläre ich, wie ein Pflichtenheft aufgebaut ist und welche Elemente es enthält, wie zum Beispiel Zielsetzungen, Produktfunktionen, Benutzerschnittstellen und Qualitätsanforderungen.

Pflichtenheft für das Projekt "Einmal um die Schweiz"

Hinweis: Da es für dieses Projekt keinen offiziellen Auftraggeber gibt, fungiere ich selbst als Initiator und Entwickler. Dieses Pflichtenheft dient dazu, die Anforderungen und Ziele des Projekts klar zu definieren und als Leitfaden für die Umsetzung zu dienen.

1. Einleitung

1.1 Projektübersicht

"Einmal um die Schweiz" ist eine Webplattform, die sportliche Herausforderungen innerhalb und ausserhalb der Schule fördert. Die Plattform ermöglicht es Schülern und Lehrern, an verschiedenen sportlichen Challenges teilzunehmen, ihre Leistungen zu erfassen und sich in einem freundlichen Wettbewerb miteinander zu messen.

1.2 Ziele des Projekts

Entwicklung einer benutzerfreundlichen Webplattform zur Förderung sportlicher Aktivitäten im schulischen Kontext.

Vermittlung technischer Kompetenzen an Schüler durch transparente Darstellung des Entwicklungsprozesses.

Schaffung einer nachhaltigen Lösung, die langfristig in den Schulalltag integriert werden kann.

2. Ausgangssituation und Motivation

Es existiert derzeit keine Plattform, die spezifisch auf die Bedürfnisse von Schulen und Schülern zugeschnitten ist, um sportliche Herausforderungen zu organisieren und zu verfolgen. Die Motivation für dieses Projekt liegt sowohl in der Förderung von Sport und Gemeinschaft als auch in der praktischen Anwendung von Kenntnissen in der Webentwicklung.

3. Geltungsbereich

Die Plattform richtet sich primär an:

Schüler und Lehrer des Gymnasiums Neufeld.

Potenziell auch an andere Bildungseinrichtungen und Sportgruppen mit ähnlichen Bedürfnissen.

4. Funktionale Anforderungen

4.1 Benutzerverwaltung

Registrierung und Anmeldung

Benutzer können sich registrieren und ein persönliches Profil erstellen.

Sicherer Login-Prozess mit Passwortverschlüsselung.

Rollenmanagement

Unterscheidung zwischen Schülern, Lehrern und Administratoren.

Unterschiedliche Berechtigungen je nach Rolle.

4.2 Challenge-Management

Erstellen von Challenges

Berechtigte Benutzer können neue Challenges erstellen.

Definition von Challenge-Details wie Titel, Beschreibung, Zeitraum und Zielsetzung.

Verwalten von Challenges

Bearbeiten und Löschen bestehender Challenges.

Verwaltung der Teilnehmer.

4.3 Teilnahme und Leistungserfassung

Teilnahme an Challenges

Benutzer können sich für Challenges anmelden.

Erfassung von Aktivitäten

Benutzer können ihre sportlichen Aktivitäten (z. B. gelaufene Kilometer) eintragen.

Möglichkeit, Aktivitäten zu bearbeiten oder zu löschen.

4.4 Auswertung und Ranglisten

Statistiken

Persönliche Leistungsübersicht für jeden Benutzer.

Ranglisten

Anzeige von Rankings nach individuellen und gruppenbezogenen Kriterien.

4.5 Benachrichtigungssystem

E-Mail-Benachrichtigungen

Bestätigung bei Registrierung und Teilnahme an Challenges.

In-App-Benachrichtigungen

Hinweise auf neue Challenges und Aktivitäten.

5. Nicht-funktionale Anforderungen

5.1 Benutzerfreundlichkeit

Intuitive Navigation und klare Benutzeroberfläche.

Responsives Design für verschiedene Endgeräte.

5.2 Sicherheit

Schutz personenbezogener Daten durch SSL-Verschlüsselung.

Sichere Speicherung von Passwörtern (Hashing, Salting).

5.3 Performance

Schnelle Ladezeiten und effiziente Datenbankabfragen.

Skalierbarkeit für steigende Nutzerzahlen.

5.4 Wartbarkeit

Gut strukturierter und dokumentierter Code.

Verwendung von gängigen Frameworks und Standards.

6. Technische Anforderungen

6.1 Frontend

HTML5, CSS3 und JavaScript.

Nutzung eines Frontend-Frameworks (z. B. React oder Vue.js).

6.2 Backend

Serverseitige Sprache: Python (Django) oder Node.js (Express).

RESTful API für die Kommunikation zwischen Frontend und Backend.

6.3 Datenbank

Relationale Datenbank (MySQL oder PostgreSQL).

Datenbankdesign gemäss Normalisierungsregeln.

6.4 Hosting

Deployment auf einem Cloud-Dienst (z. B. Heroku, AWS).

Automatisierte Backups und Updates.

Use Cases

# Use Cases für das interkantonale Schulprojekt "Around Switzerland"

## Use Case 1: Sportlehrperson registriert sich auf der Plattform

\*\*Akteur:\*\* Sportlehrperson

\*\*Ziel:\*\* Die Sportlehrperson registriert sich, um Zugang zur Webplattform zu erhalten.

\*\*Vorbedingungen:\*\* Keine

\*\*Nachbedingungen:\*\* Die Sportlehrperson ist registriert und kann sich anmelden.

\*\*Schritte:\*\*

1. Die Sportlehrperson navigiert zur Registrierungsseite.

2. Die Sportlehrperson gibt die notwendigen Informationen ein (z.B. Name, Schule, E-Mail-Adresse).

3. Die Sportlehrperson bestätigt die Eingabe und sendet das Formular ab.

4. Das System prüft die eingegebenen Daten auf Vollständigkeit und Korrektheit.

5. Das System erstellt ein Benutzerkonto für die Sportlehrperson und sendet eine Bestätigungsmail.

6. Die Sportlehrperson klickt auf den Bestätigungslink in der E-Mail, um die Registrierung abzuschliessen.

## Use Case 2: Die Sportlehrperson registriert eine Klasse

\*\*Akteur:\*\* Die Sportlehrperson33

\*\*Ziel:\*\* Eine neue Klasse für die Teilnahme am Projekt registrieren.

\*\*Vorbedingungen:\*\* Die Sportlehrperson hat sich erfolgreich auf der Webplattform angemeldet.

\*\*Nachbedingungen:\*\* Die Klasse ist registriert und kann an dem Projekt teilnehmen.

\*\*Schritte:\*\*

1. Die Sportlehrperson navigiert zur Seite "Klasse registrieren".

2. Die Sportlehrperson gibt die notwendigen Informationen ein (z.B. Klassenbezeichnung, Schule).

3. Die Sportlehrperson bestätigt die Eingabe und sendet das Formular ab.

4. Das System prüft die eingegebenen Daten auf Vollständigkeit und Korrektheit.

5. Das System speichert die Klasse in der Datenbank und zeigt eine Erfolgsmeldung an.

## Use Case 3: Die Sportlehrperson trägt Laufaktivitäten von Schülern ein

\*\*Akteur:\*\* Die Sportlehrperson

\*\*Ziel:\*\* Laufaktivitäten von Schülern erfassen, um den Fortschritt der Klasse im Projekt zu verfolgen.

\*\*Vorbedingungen:\*\* Die Sportlehrperson hat sich erfolgreich auf der Webplattform angemeldet und die Klasse ist bereits registriert.

\*\*Nachbedingungen:\*\* Die Laufaktivitäten der Schüler sind erfasst und im Fortschritt der Klasse berücksichtigt.

\*\*Schritte:\*\*

1. Die Sportlehrperson navigiert zur Seite "Laufaktivitäten eintragen".

2. Die Sportlehrperson wählt die betreffende Klasse und den Schüler aus.

3. Die Sportlehrperson gibt die Distanz und das Datum der Laufaktivität ein.

4. Die Sportlehrperson bestätigt die Eingabe und sendet das Formular ab.

5. Das System prüft die eingegebenen Daten auf Vollständigkeit und Korrektheit.

6. Das System speichert die Laufaktivität in der Datenbank und zeigt eine Erfolgsmeldung an.

## Use Case 4: Klassen fordern sich gegenseitig heraus

\*\*Akteur:\*\* Die Sportlehrperson

\*\*Ziel:\*\* Eine Herausforderung zwischen zwei Klassen initiieren, um die Motivation und den Wettkampfgeist zu fördern.

\*\*Vorbedingungen:\*\* Die Sportlehrperson hat sich erfolgreich auf der Webplattform angemeldet und beide beteiligten Klassen sind registriert.

\*\*Nachbedingungen:\*\* Die Herausforderung ist eingeleitet und die Sportlehrperson der herausgeforderten Klasse erhält eine Benachrichtigung.

\*\*Schritte:\*\*

1. Die Sportlehrperson navigiert zur Seite "Herausforderung erstellen".

2. Die Sportlehrperson wählt die eigene Klasse als Herausforderer und die gegnerische Klasse als Herausgeforderte aus.

3. Die Sportlehrperson legt das Start- und Enddatum für die Herausforderung fest.

4. Die Sportlehrperson bestätigt die Eingabe und sendet das Formular ab.

5. Das System prüft die eingegebenen Daten auf Vollständigkeit und Korrektheit.

6. Das System speichert die Herausforderung in der Datenbank und sendet eine Benachrichtigung an die Sportlehrperson der herausgeforderten Klasse.

## Use Case 5: Die Sportlehrperson überprüft den Fortschritt der Klasse

\*\*Akteur:\*\* Sportlehrperson

\*\*Ziel:\*\* Die Sportlehrperson möchte den Fortschritt seiner Klasse in den verschiedenen Challenges überwachen.

\*\*Vorbedingungen:\*\* Die Sportlehrperson hat sich erfolgreich auf der Webplattform angemeldet und die Klasse hat an Challenges teilgenommen.

\*\*Nachbedingungen:\*\* Die Sportlehrperson erhält einen Überblick über den Fortschritt und die Leistungen der Klasse.

\*\*Schritte:\*\*

1. Die Sportlehrperson navigiert zur Seite "Klassenfortschritt".

2. Die Sportlehrperson wählt die Klasse aus, deren Fortschritt er überprüfen möchte.

3. Das System zeigt eine Übersicht der Teilnahme und Leistungen der Schüler in den verschiedenen Challenges an.

4. Die Sportlehrperson kann die Daten nach verschiedenen Kriterien (z.B. Challenge, Datum) filtern und sortieren.

5. Die Sportlehrperson kann die Übersicht als Bericht exportieren oder drucken.

8. Qualitätssicherung

Testverfahren:

Unit-Tests für einzelne Komponenten.

Integrationstests für das Zusammenspiel verschiedener Module.

Benutzerakzeptanztests mit einer Testgruppe von Schülern und Lehrern.

Code Review:

Regelmässige Überprüfung des Codes auf Qualität und Effizienz.

Versionskontrolle:

Nutzung von Git für die Nachverfolgung von Änderungen.

9. Projektrisiken und Annahmen

Risiken:

Zeitliche Verzögerungen durch unerwartete technische Probleme.

Sicherheitsrisiken bei der Datenübertragung und -speicherung.

Annahmen:

Benutzer verfügen über Grundkenntnisse im Umgang mit Webanwendungen.

Internetzugang und moderne Browser sind verfügbar.

10. Zeitplan und Meilensteine

Phase 1 (2 Wochen): Anforderungsanalyse und Erstellung des Pflichtenhefts.

Phase 2 (3 Wochen): Design der Benutzeroberfläche und Datenbankmodellierung.

Phase 3 (6 Wochen): Implementierung von Frontend und Backend.

Phase 4 (3 Wochen): Testen und Fehlerbehebung.

Phase 5 (2 Wochen): Deployment und Einführung der Plattform.

Phase 6 (laufend): Wartung und Weiterentwicklung basierend auf Nutzerfeedback.

11. Dokumentation

Entwicklungsdokumentation:

Technische Spezifikationen.

Architekturdiagramme.

Benutzerdokumentation:

Benutzerhandbuch.

FAQs und Hilfeseiten.

Schulungsmaterialien:

Präsentationen für Workshops mit Schülern und Lehrern.

12. Schlussbemerkung

Dieses Pflichtenheft legt die Grundlage für die Entwicklung der Webplattform "Einmal um die Schweiz". Obwohl es keinen externen Auftraggeber gibt, ist es wichtig, die Anforderungen klar zu definieren, um ein zielgerichtetes und erfolgreiches Projekt zu gewährleisten. Durch die Einbindung von Use Cases und die Erstellung eines Pflichtenhefts möchte ich den Schülern zeigen, wie professionelle Softwareentwicklung geplant und umgesetzt wird.

2.4 Datenbankdesign

## 2. Datenbankdesign

Ein zentrales Element bei der Entwicklung einer Webanwendung ist das \*\*Datenbankdesign\*\*. Die Datenbank bildet das Rückgrat der Anwendung, da sie alle notwendigen Informationen speichert und verwaltet. Ein gut durchdachtes Datenbankdesign gewährleistet nicht nur die effiziente Speicherung und Abfrage von Daten, sondern auch die Skalierbarkeit und Wartbarkeit der Anwendung. In diesem Abschnitt werde ich detailliert erläutern, \*\*wie\*\* ich das Datenbankdesign für das Projekt "Einmal um die Schweiz" entwickelt habe, und dabei die einzelnen Schritte und Überlegungen erklären. Ziel ist es, den Schülern einen praxisnahen Einblick in den Prozess des Datenbankentwurfs zu geben.

### 2.1 Anforderungen an die Datenbank

Bevor mit dem eigentlichen Design begonnen wird, ist es wichtig, die Anforderungen an die Datenbank zu definieren. Dies umfasst die Identifikation aller Daten, die gespeichert werden müssen, sowie deren Beziehungen zueinander. Für das Projekt ergeben sich folgende Hauptanforderungen:

- \*\*Benutzerverwaltung\*\*: Speicherung von Benutzerinformationen wie Namen, E-Mail-Adressen, Passwörtern (verschlüsselt), Rollen (Schüler, Lehrer, Administrator) und Profilbildern.

- \*\*Challenges\*\*: Speicherung von Informationen zu den sportlichen Herausforderungen, einschliesslich Titel, Beschreibung, Start- und Enddatum, Ersteller und Teilnehmer.

- \*\*Teilnahmen\*\*: Erfassung der Beziehung zwischen Benutzern und Challenges, d. h., welche Benutzer an welchen Challenges teilnehmen.

- \*\*Aktivitäten\*\*: Speicherung der von den Benutzern erfassten sportlichen Aktivitäten, z. B. Datum, Art der Aktivität, zurückgelegte Distanz oder Dauer.

- \*\*Ranglisten und Statistiken\*\*: Berechnung und Speicherung von Punkten, Ranglistenpositionen und anderen statistischen Daten.

### 2.2 Konzeptuelles Datenmodell (ER-Modell)

Nachdem die Anforderungen definiert sind, erstelle ich ein \*\*Entity-Relationship-Diagramm (ER-Diagramm)\*\*, um die Datenstruktur und die Beziehungen zwischen den verschiedenen Entitäten (Tabellen) zu visualisieren. Das ER-Modell dient als Grundlage für das logische und physische Datenbankdesign.

\*\*Hauptentitäten und Attribute:\*\*

1. \*\*Benutzer\*\*

- BenutzerID (Primärschlüssel)

- Vorname

- Nachname

- E-Mail (eindeutig)

- Passwort (verschlüsselt)

- Rolle (Schüler, Lehrer, Administrator)

- Registrierungsdatum

- Profilbild

2. \*\*Challenge\*\*

- ChallengeID (Primärschlüssel)

- Titel

- Beschreibung

- Startdatum

- Enddatum

- ErstellerID (Fremdschlüssel zu Benutzer)

- Erstellungsdatum

3. \*\*Teilnahme\*\*

- TeilnahmeID (Primärschlüssel)

- BenutzerID (Fremdschlüssel zu Benutzer)

- ChallengeID (Fremdschlüssel zu Challenge)

- Anmeldedatum

4. \*\*Aktivität\*\*

- AktivitätID (Primärschlüssel)

- TeilnahmeID (Fremdschlüssel zu Teilnahme)

- Datum

- Aktivitätstyp (z. B. Laufen, Radfahren)

- Wert (z. B. zurückgelegte Distanz)

- Einheit (z. B. Kilometer)

5. \*\*Punkte\*\*

- PunkteID (Primärschlüssel)

- TeilnahmeID (Fremdschlüssel zu Teilnahme)

- Gesamtpunkte

- Aktualisierungsdatum

\*\*Beziehungen zwischen den Entitäten:\*\*

- Ein \*\*Benutzer\*\* kann mehrere \*\*Challenges\*\* erstellen (1:n Beziehung zwischen Benutzer und Challenge).

- Ein \*\*Benutzer\*\* kann an mehreren \*\*Challenges\*\* teilnehmen, und eine \*\*Challenge\*\* kann von mehreren \*\*Benutzern\*\* absolviert werden (n:m Beziehung zwischen Benutzer und Challenge, realisiert durch die \*\*Teilnahme\*\*-Entität).

- Eine \*\*Teilnahme\*\* kann mehrere \*\*Aktivitäten\*\* haben (1:n Beziehung zwischen Teilnahme und Aktivität).

- Eine \*\*Teilnahme\*\* hat genau einen \*\*Punkte\*\*-Eintrag (1:1 Beziehung zwischen Teilnahme und Punkte).

### 2.3 Normalisierung der Datenbank

Die Normalisierung ist ein Prozess zur Organisation der Datenbankstrukturen, um Redundanzen zu minimieren und Dateninkonsistenzen zu vermeiden. In diesem Projekt habe ich die Datenbank bis zur \*\*dritten Normalform (3NF)\*\* normalisiert.

\*\*Erste Normalform (1NF):\*\*

- Sicherstellen, dass in jeder Tabelle die Attribute atomar sind, d. h., keine mehrfachen Werte in einem Attribut.

- Beispiel: Das Attribut "Vollname" wird in "Vorname" und "Nachname" aufgeteilt.

\*\*Zweite Normalform (2NF):\*\*

- Alle Nicht-Schlüsselattribute müssen vom gesamten Primärschlüssel abhängen.

- Da wir bei zusammengesetzten Primärschlüsseln arbeiten (z. B. in der \*\*Teilnahme\*\*-Tabelle), stellen wir sicher, dass alle Attribute vollständig vom Primärschlüssel abhängen.

\*\*Dritte Normalform (3NF):\*\*

- Alle Attribute sind nur vom Primärschlüssel abhängig und nicht von anderen Nicht-Schlüsselattributen.

- Beispiel: In der \*\*Aktivität\*\*-Tabelle gibt es keine Attribute, die von anderen Nicht-Schlüsselattributen abhängen.

### 2.4 Physisches Datenmodell und Datenbankschema

Nach der Erstellung des konzeptuellen Modells und der Normalisierung wird das physische Datenbankschema erstellt. Dies beinhaltet die Definition der Tabellen, Spalten, Datentypen, Schlüssel und Indexe in der gewählten Datenbankmanagementsystem (DBMS), z. B. MySQL oder PostgreSQL.

\*\*Beispiel für das Datenbankschema:\*\*

```sql

-- Tabelle für Benutzer

CREATE TABLE Benutzer (

BenutzerID SERIAL PRIMARY KEY,

Vorname VARCHAR(50) NOT NULL,

Nachname VARCHAR(50) NOT NULL,

Email VARCHAR(100) NOT NULL UNIQUE,

Passwort VARCHAR(255) NOT NULL,

Rolle VARCHAR(20) NOT NULL,

Registrierungsdatum TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Profilbild VARCHAR(255)

);

-- Tabelle für Challenges

CREATE TABLE Challenge (

ChallengeID SERIAL PRIMARY KEY,

Titel VARCHAR(100) NOT NULL,

Beschreibung TEXT,

Startdatum DATE NOT NULL,

Enddatum DATE NOT NULL,

ErstellerID INT REFERENCES Benutzer(BenutzerID),

Erstellungsdatum TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Tabelle für Teilnahme

CREATE TABLE Teilnahme (

TeilnahmeID SERIAL PRIMARY KEY,

BenutzerID INT REFERENCES Benutzer(BenutzerID),

ChallengeID INT REFERENCES Challenge(ChallengeID),

Anmeldedatum TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE (BenutzerID, ChallengeID)

);

-- Tabelle für Aktivitäten

CREATE TABLE Aktivitaet (

AktivitaetID SERIAL PRIMARY KEY,

TeilnahmeID INT REFERENCES Teilnahme(TeilnahmeID),

Datum DATE NOT NULL,

Aktivitaetstyp VARCHAR(50) NOT NULL,

Wert DECIMAL(10,2) NOT NULL,

Einheit VARCHAR(20) NOT NULL

);

-- Tabelle für Punkte

CREATE TABLE Punkte (

PunkteID SERIAL PRIMARY KEY,

TeilnahmeID INT REFERENCES Teilnahme(TeilnahmeID),

Gesamtpunkte DECIMAL(10,2) NOT NULL,

Aktualisierungsdatum TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE (TeilnahmeID)

);

```

\*\*Erklärungen zu den Datenbankbefehlen:\*\*

- \*\*SERIAL PRIMARY KEY\*\*: Definiert eine automatisch inkrementierende Primärschlüsselspalte.

- \*\*VARCHAR(n)\*\*: Zeichenkette mit maximaler Länge \*n\*.

- \*\*REFERENCES\*\*: Definiert einen Fremdschlüssel zu einer anderen Tabelle.

- \*\*TIMESTAMP DEFAULT CURRENT\_TIMESTAMP\*\*: Speichert das aktuelle Datum und die Uhrzeit, wenn ein neuer Datensatz erstellt wird.

- \*\*UNIQUE\*\*: Erzwingt die Eindeutigkeit der angegebenen Spalte(n).

### 2.5 Umsetzung der Beziehungen und Constraints

Die Implementierung von Beziehungen und Constraints (Einschränkungen) ist entscheidend für die Datenintegrität. Durch die Verwendung von Fremdschlüsseln werden referenzielle Integrität gewährleistet, d. h., dass Verweise zwischen Tabellen gültig sind.

\*\*Beispiele:\*\*

- In der \*\*Teilnahme\*\*-Tabelle wird sichergestellt, dass ein Benutzer nicht mehrfach an derselben Challenge teilnehmen kann, durch die \*\*UNIQUE (BenutzerID, ChallengeID)\*\*-Constraint.

- Die \*\*Aktivität\*\*-Tabelle referenziert die \*\*Teilnahme\*\*-Tabelle, um zu wissen, zu welcher Teilnahme eine Aktivität gehört.

- Durch die \*\*ON DELETE CASCADE\*\*-Option kann festgelegt werden, dass beim Löschen eines Benutzers alle zugehörigen Teilnahmen, Aktivitäten und Punkte ebenfalls gelöscht werden.

\*\*Beispiel mit ON DELETE CASCADE:\*\*

```sql

-- Anpassung der Teilnahme-Tabelle

CREATE TABLE Teilnahme (

TeilnahmeID SERIAL PRIMARY KEY,

BenutzerID INT REFERENCES Benutzer(BenutzerID) ON DELETE CASCADE,

ChallengeID INT REFERENCES Challenge(ChallengeID) ON DELETE CASCADE,

Anmeldedatum TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE (BenutzerID, ChallengeID)

);

```

### 2.6 Indizierung und Performance-Optimierung

Um die Abfragegeschwindigkeit der Datenbank zu erhöhen, setze ich Indexe auf häufig verwendete Spalten.

\*\*Beispiele für Indexe:\*\*

- \*\*E-Mail-Adresse\*\* in der \*\*Benutzer\*\*-Tabelle für schnelle Login-Abfragen.

- \*\*ChallengeID\*\* in der \*\*Teilnahme\*\*-Tabelle, um schnell alle Teilnehmer einer Challenge zu finden.

- \*\*TeilnahmeID\*\* in der \*\*Aktivität\*\*-Tabelle für schnelle Zugriffe auf die Aktivitäten eines Teilnehmers.

```sql

CREATE INDEX idx\_benutzer\_email ON Benutzer (Email);

CREATE INDEX idx\_teilnahme\_challenge ON Teilnahme (ChallengeID);

CREATE INDEX idx\_aktivitaet\_teilnahme ON Aktivitaet (TeilnahmeID);

```

### 2.7 Sicherheit und Datenschutz

Der Schutz der Benutzerdaten ist von grösster Bedeutung. Hier einige Massnahmen, die ich implementiere:

- \*\*Passwortverschlüsselung\*\*: Verwendung von sicheren Hashing-Algorithmen wie bcrypt zur Speicherung von Passwörtern.

- \*\*Zugriffsrechte\*\*: Einschränkung der Datenbankzugriffe durch definierte Benutzerrollen (z. B. Leser, Schreiber).

- \*\*Prepared Statements\*\*: Vermeidung von SQL-Injections durch die Verwendung vorbereiteter Anweisungen in der Anwendungsschicht.

- \*\*SSL/TLS-Verschlüsselung\*\*: Sicherstellung der verschlüsselten Datenübertragung zwischen Anwendung und Datenbankserver.

### 2.8 Backup und Wiederherstellung

Um Datenverlust zu vermeiden, plane ich regelmässige Backups der Datenbank:

- \*\*Automatisierte tägliche Backups\*\*: Skripte, die die Datenbank in regelmässigen Abständen sichern.

- \*\*Offsite-Backups\*\*: Speicherung der Backups an einem externen Ort.

- \*\*Wiederherstellungsplan\*\*: Dokumentation der Schritte zur Wiederherstellung der Datenbank im Notfall.

### 2.9 Dokumentation des Datenbankdesigns

Eine ausführliche Dokumentation erleichtert die Wartung und Weiterentwicklung der Datenbank:

- \*\*Datenbankdiagramme\*\*: Visuelle Darstellung der Tabellen und ihrer Beziehungen.

- \*\*Beschreibung der Tabellen und Spalten\*\*: Zweck und Datentypen erläutern.

- \*\*Kommentar im Code\*\*: Nutzung von Kommentaren in den SQL-Skripten zur Erklärung komplexer Strukturen.

\*\*Beispiel für eine Tabellenbeschreibung:\*\*

- \*\*Tabelle:\*\* Benutzer

- \*\*Zweck:\*\* Speicherung der grundlegenden Informationen aller Benutzer der Plattform.

- \*\*Wichtige Felder:\*\*

- \*\*Email:\*\* Eindeutige E-Mail-Adresse zur Identifizierung und Anmeldung.

- \*\*Passwort:\*\* Verschlüsseltes Passwort zur Authentifizierung.

- \*\*Rolle:\*\* Bestimmt die Berechtigungen innerhalb der Plattform.

### 2.10 Zusammenfassung und Ausblick

Das Datenbankdesign ist ein wesentlicher Bestandteil des Projekts "Einmal um die Schweiz". Durch die sorgfältige Planung und Umsetzung der Datenbankstrukturen stelle ich sicher, dass die Anwendung effizient, sicher und skalierbar ist. Die gewählten Entitäten und Beziehungen ermöglichen eine flexible Erweiterung der Plattform, beispielsweise durch zusätzliche Funktionen wie Gruppen-Challenges oder erweiterte Statistikfunktionen.

\*\*Für die Schüler bedeutet dieser Abschnitt:\*\*

- \*\*Verständnis des Datenbankentwurfsprozesses\*\*: Von der Anforderungsanalyse über das ER-Modell bis hin zur Implementierung.

- \*\*Anwendung von Normalisierungsprinzipien\*\*: Um eine effiziente und konsistente Datenstruktur zu gewährleisten.

- \*\*Kenntnis von Sicherheitsmassnahmen\*\*: Zum Schutz sensibler Benutzerdaten.

- \*\*Bedeutung der Dokumentation\*\*: Für die Wartbarkeit und Verständlichkeit des Projekts.

In der weiteren Entwicklung der Plattform können die Schüler selbstständig Erweiterungen am Datenbankschema vornehmen und lernen, \*\*wie\*\* Änderungen sorgfältig geplant und implementiert werden müssen, um die Integrität der Daten zu bewahren.

### Anhang: Glossar wichtiger Begriffe

- \*\*Entität\*\*: Ein Objekt oder Konzept in der realen Welt, das in der Datenbank dargestellt wird (z. B. Benutzer, Challenge).

- \*\*Attribut\*\*: Eine Eigenschaft oder Merkmal einer Entität (z. B. Vorname, Nachname).

- \*\*Primärschlüssel\*\*: Ein Attribut oder eine Kombination von Attributen, die jeden Datensatz eindeutig identifiziert.

- \*\*Fremdschlüssel\*\*: Ein Attribut in einer Tabelle, das auf den Primärschlüssel einer anderen Tabelle verweist und so eine Beziehung zwischen den Tabellen herstellt.

- \*\*Normalisierung\*\*: Der Prozess der Organisation von Daten in einer Datenbank, um Redundanzen zu minimieren und die Datenintegrität zu gewährleisten.

- \*\*Index\*\*: Eine Datenstruktur, die den schnellen Zugriff auf Daten in einer Tabelle ermöglicht.

---

Mit diesem ausführlichen Datenbankdesign lege ich den Grundstein für eine stabile und effiziente Webanwendung. Durch die detaillierte Erläuterung der einzelnen Schritte und Entscheidungen möchte ich den Schülern vermitteln, \*\*wie\*\* wichtig ein durchdachtes Datenbankdesign für den Erfolg eines Softwareprojekts ist und \*\*wie\*\* man dabei methodisch vorgeht.

2.5 Technologiestack und Infrastruktur

3. Grundstruktur der Web-App

Ein Bild, das Text, Screenshot, Diagramm, Design enthält.

Automatisch generierte Beschreibung

Frontend

# Backend

In diesem Projekt wurde mit Node.js und Express gearbeitet. Ich habe mich bewusst für diese Technologie entschieden, da die Lernkurve dadurch etwas steiler ist und vorhandene Kenntnisse in JavaScript sinnvoll genutzt werden konnten.

Bevor ich die Entwicklung meiner PERN-Anwendung (Postgres, Express, React und Node) detailliert darstelle, möchte ich kurz erklären, wie die einzelnen Komponenten zusammenarbeiten.

## Was ist Node.js?

Node.js ist eine Laufzeitumgebung, die es ermöglicht, JavaScript auch ausserhalb des Browsers auszuführen. Sie wurde 2009 entwickelt und basiert auf der V8 JavaScript-Engine von Google Chrome. Jeder Browser verfügt über eine Engine, die den JavaScript-Code in Maschinencode übersetzt. Im Falle von Chrome wird dafür die V8-Engine genutzt.

Ryan Dahl, der Erfinder von Node.js, hat die V8-Engine extrahiert und darauf Node.js aufgebaut.[[1]](#footnote-1) Seitdem hat sich Node.js kontinuierlich weiterentwickelt. Besonders geschätzt wird die grosse Community, die die Entwicklung neuer Features stark vorantreibt. Ein weiterer Vorteil von Node.js liegt in der Möglichkeit, Full-Stack-Anwendungen einfacher zu entwickeln, da sowohl Frontend als auch Backend mit der gleichen Sprache – JavaScript – programmiert werden können.

Unterschied zu browserbasierten Anwendungen

Im Gegensatz zu browserbasierten Anwendungen, die interaktiv sind, wie zum Beispiel das Klicken auf Schaltflächen oder das Umschalten einer Navigation, handelt es sich bei Node.js-Anwendungen um serverseitige Anwendungen. Diese bestehen ausschliesslich aus Logik, ohne eine grafische Benutzeroberfläche.

In browserbasierten Anwendungen können zahlreiche Funktionen genutzt werden, die direkt auf der Benutzeroberfläche verfügbar sind. In Node.js hingegen liegt der Fokus auf der Verarbeitung von Anfragen und der Ausführung von Logik, um Daten bereitzustellen oder zu verarbeiten, da Node.js serverseitig arbeitet und keine direkte Benutzerschnittstelle bietet.

## CLI und Projekteinrichtung

Um ein Projekt mit Node.js zu entwickeln, muss Node.js natürlich zunächst installiert werden. Auf die Details der Installation werde ich hier nicht weiter eingehen, entsprechende Anleitungen finden sich leicht im Internet, beispielsweise auf der offiziellen Seite nodejs.org.

Eine Möglichkeit, JavaScript-Code in Node.js auszuführen, ist die REPL (Read-Eval-Print Loop). Diese eignet sich allerdings eher für Experimente und Tests. Für die eigentliche Entwicklung werde ich das CLI (Command Line Interface) verwenden, um den Code unter Node.js auszuführen.

Zu Beginn erstelle ich einen Projektordner, in dem ich die Datei app.js anlege. Diese Datei enthält den Code der Anwendung und kann mit dem Befehl node app.js über das Terminal ausgeführt werden. Da die Webanwendung schnell an Komplexität gewinnen wird, empfiehlt es sich, einen geeigneten Code-Editor wie Visual Studio Code (VSC) zu nutzen. Grundlegende Kenntnisse über die Funktionsweise eines Editors setze ich hier voraus.

Um ein einfaches Beispiel zu demonstrieren, kann die Datei app.js in Visual Studio Code erstellt und dann mit node app.js ausgeführt werden. Dies ermöglicht die direkte Ausführung des Codes auf der Kommandozeile.

## Wichtige globale Objekte und Module in Node.js

Node.js stellt einige wichtige globale Objekte zur Verfügung, die ich im Verlauf der Entwicklung immer wieder nutzen werde:

\_\_dirname: Gibt das Verzeichnis der aktuellen Datei zurück.

\_\_filename: Gibt den vollständigen Pfad der aktuellen Datei zurück.

require: Wird verwendet, um Module zu importieren.

module (CommonJS): Stellt die aktuelle Modulinstanz dar und ermöglicht das Exportieren von Funktionen oder Objekten.

process: Enthält Informationen zum aktuellen Node.js-Prozess, zum Beispiel Umgebungsvariablen (process.env).

Diese globalen Objekte sind entscheidend für die Struktur unserer Anwendung. Zum Beispiel kann ich mit require ein Modul importieren, das ich in meiner Anwendung verwenden möchte. Node.js folgt dem CommonJS-Modulsystem, das es mir ermöglicht, Code in wiederverwendbare Teile zu zerlegen. Ein einfaches Beispiel dafür wäre das Erstellen eines eigenen Moduls:

In der Datei greeting.js erstelle ich eine Funktion:



Diese Funktion kann dann in meiner app.js genutzt werden:



Hier importiere ich das Modul greeting mithilfe von require und kann dann die Funktion greet nutzen, um eine Begrüssung auszugeben. Dies ist ein einfaches Beispiel dafür, wie Module in Node.js verwendet werden, um den Code modular und übersichtlich zu halten. Im weiteren Verlauf werde ich auf diese globalen Objekte und Konzepte immer wieder zurückkommen, um eine saubere und gut strukturierte Anwendung zu entwickeln.

## Eingebaute Module in Node.js

Node.js bietet eine Vielzahl an eingebauten Modulen, die ich in meiner Anwendung frei nutzen kann. Diese Module erleichtern die Entwicklung und decken viele grundlegende Anforderungen ab. Im Folgenden werde ich einige wichtige Module vorstellen, die häufig verwendet werden:

* **OS-Modul** (os): Dieses Modul ermöglicht es mir, Informationen über das Betriebssystem zu erhalten, zum Beispiel verfügbare Speicherressourcen oder die CPU-Architektur. Ein einfaches Beispiel wäre das Abrufen von Systeminformationen:



In diesem Beispiel wird das os-Modul verwendet, um grundlegende Informationen über das Betriebssystem abzurufen. Ich kann sehen, welches Betriebssystem verwendet wird (os.platform()), die CPU-Architektur (os.arch()) sowie die Informationen über den verfügbaren und den gesamten Arbeitsspeicher (os.freemem() und os.totalmem()). Dieses Modul ist besonders hilfreich, wenn ich systemabhängige Entscheidungen in meiner Anwendung treffen muss.

Weitere nützliche Module sind:

* **Path-Modul** (path): Das path-Modul hilft dabei, Dateipfade zu bearbeiten, zu kombinieren oder zu normalisieren. Es ist besonders nützlich, um sicherzustellen, dass Dateipfade unabhängig vom Betriebssystem korrekt funktionieren.
* **File System Modul** (fs): Das fs-Modul ermöglicht den Zugriff auf das Dateisystem. Ich kann damit Dateien lesen, schreiben, löschen oder auch Ordner erstellen. Es ist unverzichtbar für den Umgang mit Dateien in Node.js.
* **HTTP-Modul** (http): Das http-Modul wird verwendet, um einen HTTP-Server zu erstellen. Damit kann ich Anfragen empfangen und darauf reagieren, was die Grundlage für eine Webanwendung bildet.

Diese Module werden alle als eingebaute Module bezeichnet, die ohne zusätzliche Installation direkt verwendet werden können. Sie helfen dabei, häufige Anforderungen abzudecken und bieten eine solide Grundlage für die Entwicklung serverseitiger Anwendungen. Weitere Informationen zu den spezifischen Methoden und Eigenschaften der Module finden sich in der offiziellen Dokumentation unter nodejs.org.

## Einführung in npm und externe Module

Neben den eingebauten Modulen spielt npm (Node Package Manager) eine zentrale Rolle in der Entwicklung von Node.js-Anwendungen. npm ist ein Paketmanager, der automatisch zusammen mit Node.js installiert wird und es mir ermöglicht, externe Lösungen – sogenannte **Pakete** – in meine Projekte zu integrieren. Es ist wichtig, den Unterschied zwischen einem **Modul** und einem **Paket** zu verstehen:

* **Modul**: Ein Modul ist eine Einheit von JavaScript-Code, die in Node.js importiert und wiederverwendet werden kann. Jedes Modul hat eine spezifische Funktionalität, wie zum Beispiel das fs-Modul für den Dateisystemzugriff. Module können entweder eingebaut sein (wie os, http etc.) oder von mir selbst erstellt werden.
* **Paket**: Ein Paket hingegen ist eine Sammlung von Modulen oder JavaScript-Dateien, die in einem Ordner gebündelt sind und zusätzliche Informationen in einer package.json enthalten. Pakete können viele Module umfassen und über npm installiert werden. Pakete bieten eine umfassendere Funktionalität und helfen bei der Erweiterung von Anwendungen durch die Integration von externen Lösungen.

Mit npm kann ich unter anderem:

* **Externe Pakete integrieren**: Es gibt zahlreiche Pakete, die von anderen Entwicklerinnen und Entwicklern erstellt wurden und spezifische Probleme lösen oder Funktionalitäten bereitstellen. Ich kann diese Pakete einfach installieren und in meinem Projekt verwenden.
* **Eigene Pakete erstellen und teilen**: Ich kann meinen eigenen Code als Paket strukturieren und über npm teilen, sodass auch andere Entwickler darauf zugreifen und ihn nutzen können.
* **Abhängigkeiten verwalten**: npm ermöglicht es, die Abhängigkeiten einer Anwendung zentral zu verwalten. Das bedeutet, alle Pakete, die ich in meiner Anwendung verwende, können über eine einzige Datei (package.json) dokumentiert und verwaltet werden.

**Was ist die package.json?**

Die Datei package.json ist eine zentrale Datei in jedem Node.js-Projekt, die wichtige Informationen über das Projekt und seine Abhängigkeiten enthält. Sie dient als Konfigurationsdatei und enthält Metadaten über das Projekt, wie den Namen, die Version und die verwendeten Skripte. Besonders wichtig ist, dass sie die Abhängigkeiten (also die verwendeten Pakete) dokumentiert, sodass jeder, der das Projekt verwendet, die gleiche Umgebung nachstellen kann.[[2]](#footnote-2)

Ein einfaches Beispiel für eine package.json:

{

"name": "mein-projekt",

"version": "1.0.0",

"description": "Ein Beispielprojekt",

"main": "app.js",

"scripts": {

"start": "node app.js"

},

"dependencies": {

"express": "^4.17.1"

}

}

In diesem Beispiel beschreibt die package.json die grundlegenden Informationen des Projekts. Die wichtigsten Teile sind:

* **name** und **version**: Der Name und die Version des Projekts.
* **scripts**: Hier kann ich Skripte definieren, die ich über npm ausführen kann. Zum Beispiel startet das Skript start meine Anwendung mit dem Befehl node app.js.
* **dependencies**: Hier werden die Abhängigkeiten (Pakete) definiert, die für das Projekt erforderlich sind. In diesem Fall ist das Paket express in Version 4.17.1 oder höher erforderlich.

Die package.json ermöglicht es anderen Entwicklern, das Projekt einfach zu klonen und alle benötigten Abhängigkeiten durch den Befehl npm install automatisch zu installieren. Dadurch wird die Entwicklung deutlich vereinfacht, insbesondere bei grösseren Projekten mit vielen Abhängigkeiten.

Ein typisches Beispiel: Angenommen, ich möchte in meiner Anwendung einen Slider einbauen. Hier kann ich entweder den Slider von Grund auf neu entwickeln oder auf ein bereits vorhandenes Paket zugreifen, das von der Community entwickelt und über npm bereitgestellt wurde. Die Wahrscheinlichkeit ist hoch, dass jemand dieses Problem bereits gelöst hat und seine Lösung als Paket veröffentlicht hat. Ich kann das Paket dann einfach in mein Projekt integrieren, indem ich den entsprechenden Befehl ausführe, beispielsweise:

npm install slider-library

Danach kann ich das Modul aus diesem Paket in meiner app.js nutzen:

const slider = require('slider-library');

// Verwende das Slider-Modul in deiner Anwendung

Das npm-Ökosystem ist unter [npmjs.com](https://www.npmjs.com) gehostet, wo sich alles von nützlichen Utility-Funktionen bis hin zu vollständigen Frameworks und Bibliotheken findet. Ein Beispiel für ein weit verbreitetes Paket ist create-react-app, das für die Entwicklung von React-Anwendungen verwendet wird.

Es ist wichtig zu erwähnen, dass es im npm-Registry keine zentrale Qualitätskontrolle gibt – jeder kann ein Paket hochladen. Daher sollte darauf geachtet werden, nur populäre und gut getestete Pakete zu verwenden. Ein guter Indikator für die Qualität eines Pakets ist die Anzahl der wöchentlichen Downloads: Wenn ein Paket oft heruntergeladen wird, ist die Wahrscheinlichkeit hoch, dass es sich um eine bewährte und stabile Lösung handelt.

**Einrichten eines Node.js-Projekts mit npm init -y**

Um ein neues Node.js-Projekt aufzusetzen, wird oft der Befehl npm init verwendet. Dieser Befehl erstellt eine package.json-Datei, die die Basisinformationen über das Projekt enthält. Wenn ich jedoch schnell eine package.json mit den Standardeinstellungen erstellen möchte, kann ich den Befehl npm init -y verwenden. Dieser Befehl setzt alle Fragen, die normalerweise während der Initialisierung gestellt werden, automatisch auf die Standardwerte.

Mit npm init -y wird eine package.json erstellt, die automatisch Felder wie name, version und main enthält. Diese Standardeinstellungen können später jederzeit angepasst werden.

Ein typisches Beispiel für die Nutzung von npm init -y:

mkdir mein-projekt

cd mein-projekt

npm init -y

Nach Ausführung dieses Befehls befindet sich eine package.json-Datei im Projektordner, die die Basis für die Installation weiterer Abhängigkeiten bildet. Dadurch ist es möglich, direkt mit der Entwicklung zu beginnen und benötigte Pakete hinzuzufügen, beispielsweise:

npm install express

Nun kann ich express in meiner Anwendung nutzen, indem ich es in meiner app.js importiere und eine einfache Funktion ausführe:

const express = require('express');

console.log('Express wurde erfolgreich importiert.');

Der Befehl npm init -y ist eine grossartige Möglichkeit, schnell eine package.json zu erstellen, ohne jeden Parameter manuell eingeben zu müssen. Dies ist besonders nützlich, wenn ich schnell starten möchte und später die Feinabstimmung vornehmen kann.

Der Befehl npm init -y ist eine grossartige Möglichkeit, schnell eine package.json zu erstellen, ohne jeden Parameter manuell eingeben zu müssen. Dies ist besonders nützlich, wenn ich schnell starten möchte und später die Feinabstimmung vornehmen kae

In meiner Entwicklung werde ich sowohl auf eingebaute Module als auch auf externe npm-Pakete zurückgreifen, um die bestmögliche Funktionalität und Effizienz zu erreichen. Das Zusammenspiel dieser Module und Pakete ermöglicht es mir, meine Anwendung modular, flexibel und wartbar zu gestalten.

**Nutzung von Nodemon für die Entwicklung**

Während der Entwicklung möchte ich nicht jedes Mal, wenn ich Änderungen an meiner Anwendung vornehme, den Node-Prozess manuell neu starten. Hier kommt nodemon ins Spiel. Nodemon ist ein Tool, das meine Anwendung automatisch neu startet, wenn ich Änderungen an den Quelldateien vornehme. Dadurch spare ich Zeit und vereinfache den Entwicklungsprozess erheblich.

Um nodemon zu installieren, kann ich es als sogenannte **Dev-Dependency** hinzufügen. Dev-Dependencies sind Pakete, die nur während der Entwicklung benötigt werden und nicht in der Produktionsumgebung verwendet werden müssen. Um nodemon als Dev-Dependency hinzuzufügen, verwende ich folgenden Befehl:

npm install nodemon --save-dev

Nach der Installation kann ich meine package.json so anpassen, dass nodemon verwendet wird, um meine Anwendung zu starten. Dazu füge ich ein Skript unter scripts hinzu:

"scripts": {

"start": "node app.js",

"dev": "nodemon app.js"

}

Jetzt kann ich die Anwendung im Entwicklungsmodus mit dem Befehl npm run dev starten. Nodemon wird die Anwendung automatisch neu starten, wann immer ich Änderungen an den Dateien vornehme. Das spart eine Menge Zeit, da ich nicht jedes Mal den Befehl node app.js manuell ausführen muss.

**Exkurs: Der Event Loop in Node.js**

Um das Verhalten von JavaScript in Node.js besser zu verstehen, ist es hilfreich, sich mit dem Konzept des **Event Loops** auseinanderzusetzen. Der Event Loop ist ein Mechanismus, der es JavaScript ermöglicht, nicht-blockierende I/O-Operationen auszuführen, obwohl JavaScript von Natur aus eine synchrone und single-threaded Sprache ist.

**Was bedeutet synchron und single-threaded?**

JavaScript ist eine **synchrone, single-threaded** Sprache, was bedeutet, dass der Code zeilenweise von oben nach unten ausgeführt wird. Stellen wir uns vor, wir haben folgenden Code:

console.log('Erster Task');

for (let i = 0; i < 1e9; i++) {

// Eine zeitaufwendige Schleife, die einige Sekunden dauert

}

console.log('Nächster Task');

In diesem Beispiel wird die erste console.log-Anweisung ausgeführt, danach läuft die zeitaufwendige Schleife, und erst wenn diese beendet ist, wird die nächste console.log-Anweisung ausgeführt. Das bedeutet, dass während der Schleife der gesamte Code blockiert wird, und nichts anderes kann ausgeführt werden. JavaScript führt also den Code **synchron** aus, bis die aktuelle Aufgabe beendet ist.

**Der Event Loop und asynchrone Aufgaben**

Wenn wir jedoch asynchrone Aufgaben ausführen wollen, wie zum Beispiel einen Netzwerkaufruf, kommt der **Event Loop** ins Spiel. Der Event Loop ermöglicht es, zeitaufwendige Aufgaben "auszulagern", sodass andere Teile des Codes weiterlaufen können, ohne dass alles blockiert wird.

Ein klassisches Beispiel ist die Verwendung von setTimeout:

console.log('Erster Task');

setTimeout(() => {

console.log('Timeout Task');

}, 0);

console.log('Nächster Task');

Obwohl setTimeout auf 0 gesetzt ist und man erwarten könnte, dass die Callback-Funktion sofort ausgeführt wird, passiert dies erst nach der Ausführung des gesamten synchronen Codes. Das bedeutet, dass erst 'Erster Task' und 'Nächster Task' geloggt werden, bevor der Timeout Task erscheint. Der Event Loop stellt sicher, dass die Callback-Funktion erst ausgeführt wird, nachdem der aktuelle Stapel abgearbeitet wurde.

**Event Loop in Node.js**

In Node.js funktioniert der Event Loop ähnlich wie im Browser. Der Event Loop ist dafür verantwortlich, dass zeitaufwendige Aufgaben, wie z. B. Datenbankabfragen oder Dateizugriffe, die Hauptausführung nicht blockieren. Anstatt die gesamte Anwendung zu blockieren, registriert der Event Loop Callbacks, die ausgeführt werden, sobald die zeitaufwendige Operation abgeschlossen ist.

Stellen wir uns vor, wir haben eine Anwendung, bei der mehrere Benutzer gleichzeitig Anfragen stellen. Der Event Loop sorgt dafür, dass keine Anfrage die anderen blockiert, indem zeitaufwendige Aufgaben ausgelagert und die zugehörigen Callbacks registriert werden. Sobald die Aufgabe abgeschlossen ist, wird der Callback ausgeführt, jedoch erst, wenn der Event Loop Zeit dafür hat – das bedeutet, nachdem alle anderen sofort ausführbaren Aufgaben erledigt sind.

**Beispiel: fs.readFile und der Event Loop**

Ein gutes Beispiel für den Event Loop in Node.js ist das Dateisystemmodul (fs) und die Methode fs.readFile. Angenommen, wir möchten den Inhalt einer Datei lesen, ohne den gesamten Prozess zu blockieren, während auf das Lesen der Datei gewartet wird. Hier kommt der Event Loop ins Spiel, um eine asynchrone Bearbeitung zu ermöglichen.

Schauen wir uns folgendes Beispiel an:

const fs = require('fs');

console.log('Starte das Lesen der Datei');

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) {

console.error('Fehler beim Lesen der Datei:', err);

return;

}

console.log('Dateiinhalt:', data);

});

console.log('Nach dem Start des Lesevorgangs');

In diesem Beispiel passiert Folgendes:

1. Der Befehl console.log('Starte das Lesen der Datei') wird ausgeführt.
2. Die Funktion fs.readFile wird aufgerufen, um die Datei example.txt zu lesen. Da diese Methode asynchron ist, wird der Lesevorgang "ausgelagert" und der Callback registriert, der ausgeführt wird, sobald der Lesevorgang abgeschlossen ist.
3. Während der Dateiinhalt gelesen wird, läuft der restliche Code weiter. Der Befehl console.log('Nach dem Start des Lesevorgangs') wird ausgeführt, ohne auf das Ergebnis von fs.readFile zu warten.
4. Sobald der Lesevorgang abgeschlossen ist, wird der Callback ausgeführt und der Inhalt der Datei (oder ein Fehler) wird ausgegeben.

Der Event Loop sorgt dafür, dass der Code nicht blockiert wird, während die Datei gelesen wird. Stattdessen wird der Callback erst dann ausgeführt, wenn der Lesevorgang beendet ist und der Event Loop Zeit hat, ihn zu bearbeiten. Dies macht Node.js besonders effizient, wenn es um I/O-lastige Anwendungen geht, da zeitaufwendige Aufgaben wie das Lesen von Dateien oder Netzwerkaufrufe die restliche Programmausführung nicht blockieren.

Einrichten eines einfachen HTTP-Servers mit dem http-Modul

Bevor wir zu Express übergehen, möchte ich noch kurz erklären, wie wir mithilfe des eingebauten http-Moduls einen grundlegenden Webserver aufsetzen können. Das http-Modul ist ein integriertes Modul in Node.js, das es uns ermöglicht, Webserver zu erstellen. Es bietet die Basis für serverseitige Funktionalitäten und ist damit die Grundlage für viele Frameworks, einschliesslich Express.

Um einen einfachen HTTP-Server zu erstellen, verwenden wir das http-Modul folgendermassen:

const http = require('http');

const server = http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.write('Willkommen auf unserer Homepage');

res.end();

});

server.listen(5000, () => {

console.log('Server läuft auf http://localhost:5000');

});

In diesem Beispiel:

* Importiere ich das http-Modul mit require('http').
* Verwende ich die Methode http.createServer(), um einen neuen Server zu erstellen. Diese Methode nimmt eine Callback-Funktion an, die auf jede eingehende Anfrage (req) und die zu sendende Antwort (res) reagiert.
* In der Callback-Funktion sende ich eine einfache Textantwort zurück (Willkommen auf unserer Homepage).
* Starte ich den Server, der auf Port 5000 auf Anfragen wartet. Sobald der Server gestartet ist, erscheint eine Meldung in der Konsole.

Das http-Modul bietet eine gute Möglichkeit, die Grundlagen des Servers in Node.js zu verstehen. Express baut auf diesen grundlegenden Konzepten auf und vereinfacht sie erheblich, was wir im nächsten Abschnitt sehen werden.

Nach der grundlegenden Einrichtung des Node.js-Projekts möchte ich nun **Express** vorstellen. Express ist ein minimalistisches und flexibles Framework für Node.js, das das Erstellen von Webanwendungen und APIs erheblich erleichtert. Es hat sich über die Jahre fast als Standard etabliert, wenn es um die Entwicklung serverseitiger Anwendungen mit Node.js geht.

**Warum Express?**

Die Gründe, warum Express so populär geworden ist und als Standard in der Node.js-Entwicklung gilt, sind vielfältig:

1. **Einfachheit und Flexibilität**: Express bietet eine sehr einfache und leicht verständliche API, die das Erstellen von Servern unkompliziert macht. Im Vergleich zum Arbeiten mit dem reinen http-Modul von Node.js reduziert Express den Boilerplate-Code erheblich, was bedeutet, dass die Entwicklung schneller und übersichtlicher wird.
2. **Minimalistisches Design**: Express ist minimalistisch und modular aufgebaut. Das bedeutet, dass nur grundlegende Funktionen bereitgestellt werden, die durch eine Vielzahl von **Middleware**-Komponenten erweitert werden können. So kann ich genau die Features hinzufügen, die ich brauche, ohne unnötige Komplexität in meinem Code zu haben.
3. **Grosse Community und Unterstützung**: Die grosse Popularität von Express bedeutet, dass es eine grosse Community und viele Tutorials, Beispiele und Unterstützung gibt. Dies erleichtert das Finden von Lösungen, wenn man auf ein Problem stösst.
4. **Middleware-Unterstützung**: Express unterstützt Middleware, kleine Funktionen, die zwischen dem Eingang einer Anfrage und der Antwort ausgeführt werden können. Dies ermöglicht es, wiederverwendbare Komponenten wie Authentifizierung, Logging oder Fehlerbehandlung einfach in den Anwendungsablauf einzubinden.

**Ein einfaches Beispiel mit Express**

Um Express zu nutzen, muss es zuerst installiert werden. Das kann ich mit folgendem Befehl tun:

npm install express

Nachdem Express installiert ist, kann ich es nutzen, um einen einfachen Webserver zu erstellen. Ein Beispiel sieht folgendermassen aus:

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hallo Welt!');

});

app.listen(3000, () => {

console.log('Server läuft auf http://localhost:3000');

});

In diesem Beispiel:

* Importiere ich Express und erstelle eine Instanz der Express-Anwendung (app).
* Definiere ich eine Route (/), die auf HTTP GET-Anfragen reagiert und eine einfache Antwort ('Hallo Welt!') zurücksendet.
* Starte ich den Server auf Port 3000, und sobald der Server läuft, wird eine Meldung in der Konsole angezeigt.

Dieses Beispiel zeigt, wie einfach es ist, mit Express einen Server aufzusetzen. Mit wenigen Zeilen Code kann eine funktionierende Webanwendung erstellt werden, die später durch weitere Routen und Middleware erweitert werden kann.

Durch die Nutzung von Express wird die Entwicklung von Servern in Node.js deutlich effizienter und flexibler. Express bietet alle grundlegenden Werkzeuge, die man braucht, und ermöglicht durch seine Modularität eine sehr präzise Anpassung an die jeweiligen Anforderungen.

Datenbankanbindung

4.Integration und Tests

Testpläne und -verfahren

Ergebnisse der Tests

Feedback und Optimierung

5. Einholung von Feedback

Feedbackmethoden

Auswertung des Feedbacks

6.2 Iterative Verbesserungen

Identifizierte Probleme und Lösungen

Implementierte Verbesserungen

Deployment und Wartung

7.1 Deployment-Prozess

Vorbereitung und Durchführung

Rollout-Plan

7.2 Wartungsplan

Regelmässige Wartungsarbeiten

Langfristige Wartungsstrategie

Fazit und Ausblick

8.1 Zusammenfassung der Ergebnisse

8.2 Herausforderungen und Lösungsansätze

8.3 Zukunftsperspektiven und Weiterentwicklun

Anhang

9.1 Glossar

Begriffe und Definitionen

9.2 Literaturverzeichnis

Quellen und Referenzen

9.3 Abbildungsverzeichnis

Übersicht der verwendeten Abbildungen

9.4 Tabellenverzeichnis

Übersicht der verwendeten Tabellen

9.5 Codebeispiele

Wichtige Codeausschnitte und Erklärungen

1. Siehe dazu Ryan Dahl's Original Node.js Talk: Ein Vortrag von Ryan Dahl, dem Schöpfer von Node.js, in dem er die Ursprünge und die Motivation hinter der Entwicklung von Node.js erläutert. https://www.youtube.com/watch?v=ztspvPYybIY. Letzter Zugriff: 15. Oktober 2024. [↑](#footnote-ref-1)
2. Die Ausführungen beruhen auf der folgenden Quelle von Tierney Cyren, der hier weitere Nützliche Informationen in seinem Blog preisgibt: Cyren, Tierney; <https://nodesource.com/blog/the-basics-of-package-json-in-node-js-and-npm/> Letzter Zugriff am 16.10.2024 [↑](#footnote-ref-2)