Webbasierte Multiplayer Schach-App

Bachelorarbeit

vorgelegt von
Jasper Paul Fülle
Matrikelnummer 3367654

Betreut von
Prof. Dr. Thorsten Thormälen

Studiengang Wirtschaftsinformatik

14. April 2023

Fachbereich Mathematik und Informatik
Philipps Universität Marburg

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Schach ist ein traditionsreiches und abwechslungsreiches Brettspiel, deren Ursprung nicht genau bestimmt werden kann. Es wird vermutet, dass das erste schachähnliche Spiel *Tschaturanga* seinen Ursprung in Nordindien um 600 n. Chr. hatte[1]. Im Laufe der Jahrhunderte hat Schach eine bedeutende Rolle in der Kultur und Geschichte gespielt. So wurde beispielsweise die Schach-WM 1972 eine Art Machtkampf im kalten Krieg zwischen der UdSSR, welche den damaligen Schach dominierten, und der USA[2]. Schach bleibt bis heute ein beliebtes Spiel, welches 2020 durch die Netflix Serie *Damengambit* und 2022 durch den Betrugsvorwurf von Magnus Carlsen an seinen 19-jährigen Gegner Hans Niemann[3] eine breitere Aufmerksamkeit erhielt (siehe Abbildung 1). Darüber hinaus hat Schach im digitalen Zeitalter eine neue Popularität erreicht. Online-Schachplattformen wie `chess.com` verzeichnen Milliarden von Live-Partien[4], während Schach Live-Streams auf Plattformen wie `twitch.com` Millionen von Followern anziehen.

Die Entwicklung einer webbasierten Multiplayer-Schach-App bietet eine einzigartige Gelegenheit, ein traditionsreiches und beliebtes Spiel im digitale Zeitalter weiter zu entwickeln. Meine Motivation für diese Arbeit besteht darin, eine App zu entwickeln, die die Grundlagen einer Schach-App enthält und gleichzeitig eine solide Basis für zukünftige Erweiterungen und Verbesserungen bietet. Insbesondere plane ich, innovative Funktionen zu integrieren, die bislang in den gängigen Schach-Apps nicht vorhanden sind, wie z.B. die Möglichkeit, unterschiedliche Schachfiguren und -bretter als Belohnungen freizuschalten oder mit Freunden eine Gruppe zu gründen, welche in einer Liga auf- und absteigen kann. Durch die Entwicklung einer Schach-App mit neuen Funktionalitäten kann ich dazu beitragen, die Popularität von Schach zu steigern und vor allem das Spiel einem breiteren Publikum zugänglich zu machen.

1.2 Zielsetzung

Diese Bachelorarbeit hat das Ziel eine Schach-App zu entwerfen und zu implementieren, die eine intuitive User Experience und ein ansprechendes User Interface mit vielen nützlichen Funktionen beinhaltet.

[1] van der Linde [1874]
[2] Hansen
[3] Sportschau [2022]
[4] chess.com [2018]

User Experience (*UX*) bezieht sich darauf wie ein Nutzer sich auf einer Anwendung bewegt und wie einfach und angenehm es für den Nutzer ist, die Funktionen der Anwendung zu verwenden.

Das User Interface (*UI*) beschäftigt sich mit der visuellen und interaktiven Gestaltung von Benutzeroberflächen. Es umfasst die Gestaltung von Buttons, Formularen und anderen visuellen Komponenten, sowie das Feedback dieser Komponenten, wie zum Beispiel die Rückmeldung eines fehlgeschlagenen Logins. Zusammengefasst beschäftigt sich UX damit, wie man eine Anwendung verwendet und UI damit, wie die Benutzeroberfläche der Anwendung aussieht.[5]

Funktionen der Schach-App sind unter anderem das Registrieren und Einloggen, das Versenden, Annehmen und Ablehnen von Freundschaftsanfragen, das Zuschauen bei laufenden Spielen, das Herausfordern von Freunden zu Schachspielen und natürlich das Spielen von Schachpartien mit einem Chat und verschiedenen Einstellungsmöglichkeiten der Schach Uhren selbst. Dabei wird besonderer Wert auf die Verwendung moderner Web-Technologien wie React, Node.js, Socket.IO, Redis und PostgeSQL gelegt, um eine optimale Benutzererfahrung und Skalierbarkeit zu gewährleisten. Darüber hinaus soll die Arbeit einen Überblick über die technischen Herausforderungen und Lösungen im Zusammenhang mit der Implementierung einer solchen Schach-App bieten.

1.3 Aufbau der Arbeit

Diese Bachelorarbeit gliedert sich in sechs Hauptkapitel, die jeweils unterschiedliche Aspekte der Entwicklung und Implementierung der Schach-App behandeln.

Im ersten Kapitel, der *Einleitung*, werden die Motivation für die Entwicklung der Schach-App, die Zielsetzung der Arbeit und der Aufbau der Arbeit selbst vorgestellt.

Das zweite Kapitel, *Theoretische Grundlagen*, erläutert die Grundlagen von Schach als Spiel sowie die verwendeten Web-Technologien wie Node.js, Express, Socket.io, React und PostgreSQL, die für das Verständnis der nachfolgenden Kapitel wichtig sind.

Im dritten Kapitel, *Systemarchitektur*, wird die Gesamtarchitektur der Schach-App beschrieben, einschließlich der Unterteilung in Frontend und Backend, der Datenbankstruktur und der Kommunikation zwischen den verschiedenen Komponenten.

Das vierte Kapitel, *Implementierung*, geht auf die praktische Umsetzung der Schach-App ein, indem es die Entwicklungsprozesse für das Frontend und das Backend sowie die Integration der Datenbanken erläutert.

Das fünfte Kapitel, *Tests und Evaluation*, behandelt die verschiedenen Tests, die durchgeführt wurden, um die Funktionalität, Usability, Performance und Sicherheit der Schach-App zu bewerten.

Im abschließenden sechsten Kapitel, *Fazit und Ausblick*, werden die Ergebnisse der Arbeit zusammengefasst, eventuelle Limitationen diskutiert und mögliche Erweiterungen und Weiterentwicklungen für die Schach-App vorgeschlagen.

Die Arbeit endet mit dem *Anhang*, der zusätzliche Grafiken und die Liste der verwendeten Literatur enthält.

[5] Robbins [2012]

2. Theoretische Grundlagen

2.1 Schach

Schach ist ein strategisches Brettspiel für zwei Spieler, welches auf einem quadratischen Spielfeld mit 64 Feldern gespielt wird. Jeder Spieler beginnt mit 16 Figuren und das Ziel des Spiels ist es, den König des Gegners schachmatt zu setzen, indem man ihn bedroht, ohne dass der Gegner den Angriff verhindern kann.

Wie sich welche Figuren bewegen und andere Figuren schlagen erkläre ich nicht explizit, lediglich zwei Sonderregeln des Schachs werde ich genauer erklären, da diese bei der Umsetzung des Spiels gesondert gehandhabt werden müssen.

Die erste ist das so genannte *en passant*-Regel. Dabei ist es einem Bauern möglich einen gegnerischen Bauer diagonal zu schlagen, falls dieser zwei Felder gezogen ist und nun auf der gleichen Höhe wie der eigene Bauer steht (siehe Abbildung 2).

Die zweite Zusatzregel ist die *Bauernumwandlung*. Sie besagt, dass falls ein Bauer die gegnerische Grundreihe erreicht, dieser Bauer in eine Dame, einen Springer, einen Turm oder einen Läufer umgewandelt werden kann (siehe Abbildung 3).

2.2 Web-Technologien

2.2.1 Node.js und Express

Node.js und seine Vorteile

Node.js ist eine JavaScript-Laufzeitumgebung, welche erstmals 2009 angekündigt wurde[1] und speziell für die Entwicklung von skalierbaren Netzwerkanwendungen entworfen wurde[2]. Skalierbarkeit bedeutet, dass mit steigender Benutzeranzahl der Ressourcenverbrauch idealerweise linear steigt. Zu den relevanten Ressourcen von Webanwendungen gehören Rechenleistung, Ein-/Ausgabeoperationen (*I/O*) und Arbeitsspeicher, wobei Node.js vor allem die Skalierbarkeit von I/O intensiven Anwendungen verbessert[3]. I/O-Zugriffe sind beispielsweise Zugriffe auf Datenbanken, Webservices oder auf das Dateisystem. Node.js setzt dabei vollständig auf asynchrone Zugriffe. Dabei wartet der Thread nicht auf das Ergebnis eines I/O-Zugriffs, sondern führt andere Aufgaben aus, bis das Ergebnis verfügbar ist. Anschließend wird eine zuvor definierte Callback Funktion (siehe Code Snippet 2.1)

[1] JSConf [2012]
[2] Foundation [2023]
[3] Prediger [2015]

durchgeführt. Bei einem synchronen Zugriff, wie es bei einigen anderen Laufzeitumgebungen der Fall ist, würde der Thread auf das Ergebnis warten und dieses anschließend weiterverarbeiten, wobei jedoch sein Speicherplatz zum Teil belegt bleibt. Die Vorteile hinsichtlich der Skalierbarkeit werden jedoch erst bei einer hohen Anzahl von Zugriffen erkennbar.

```
1 database.query(  "SELECT * FROM user",  function(result) {
2 result...
3 });
```

Code Snippet 2.1: Beispiel einer Callback Funktion **Quelle:** Prediger [2015]

Ein für mich großer und entscheidender Vorteil der Nutzung von Node.js ist, ist die Nutzung der gleichen Programmiersprache für Frontend und Backend, zumal die Syntax von JavaScript mir auch schon ein wenig geläufig war. In einem Team-Projekt kann das natürlich besonders nützlich sein, da Kommunikationsbarrieren durch unterschiedliche Programmiersprachen von Frontend und Backend niedriger sind. Natürlich versteht deshalb der Frontend-Entwickler nicht alles was der Backend-Entwickler macht und umgekehrt, jedoch gibt es eine gemeinsame Grundlage. Neben den Kommunikationsvorteilen ermöglicht die Verwendung von der gleichen Programmiersprache im Frontend und Backend das Teilen von Code. So ist es zum Beispiel möglich Callback Funktionen vom Frontend an das Backend zu senden und dort aufzurufen.

Node.js basiert auf der Verarbeitung von Requests vom Frontend und dem zurück senden von einem Result mittels dem HTTP-Protokoll. Das HTTP-Protokoll verwendet verschiedene Methoden wie GET, POST, PUT und DELETE, um unterschiedliche Aktionen durchzuführen. Zum Beispiel wird GET zum Abrufen von Informationen verwendet, während POST zum Senden von Daten verwendet wird. Die Art und Weise wie ein Request verarbeitet werden soll ist dabei selbst zu definieren (siehe Abbildung 4). Dabei kann der Request sein, eine bestimmte Seite zu laden, womit dann mit der entsprechenden HTML-Datei geantwortet wird, oder es kann als API genutzt werden, um Beispielsweise Daten einer Datenbank zu übermitteln. Um die Verarbeitung dieser Anfragen weniger komplex zu gestalten gibt es die Erweiterung *Express* für Node.js.

Express

Express ist ein leichtgewichtiges und sehr beliebtes Web-Frameworks, welches unter Node.js zur Verfügung steht. Es dient zur Vereinfachung der API von Node.js und stellt hilfreiche Funktionen bereit[4]. Es ermöglicht beispielsweise die Verwendung von *Middleware* und *Routing*.

Middleware ermöglicht, dass eine Anfrage an den Node.js Server nicht ausschließlich von einer Funktion bearbeitet werden muss, welche das Ergebnis zurücksendet, sondern von mehreren Funktionen, die sich um verschiedene Teile der Request kümmern (siehe Abbildung 5). Diese Funktionen heißen *Middleware*. Dabei gibt es eine von uns definierte Reihenfolge der Middlewares. Zum Beispiel können wir definieren, dass zu erst der Request von einer Middleware geloggt werden soll, anschließend soll der Benutzer Authentifiziert werden. Will der Benutzer eine URL aufrufen, für die er keine Berechtigung hat wird eine „not authorized“ Seite zurück gesendet und die nächste Middleware wird nicht aufgerufen.

[4]Hahn [2016]

Ansonsten wird die nächste Middleware der Kette ausgeführt, wie zum Beispiel das senden von Informationen (siehe Abbildung 6). Ein Vorteil der Nutzung von Middlewares ist, dass es bereits viele vordefinierte Middlewares (auch von Dritten) gibt, welche nützliche Funktionalitäten mitbringen. Die Anfrage des Frontends in mehrere kleinere Funktionen aufzuteilen, anstatt eine Funktion zu schreiben, welche sich um all dies kümmert verringert die Komplexität enorm.

Routing hilft dabei zu identifizieren bei welchem Request welche Middleware ausgeführt werden soll (siehe Abbildung 7). Beispielsweise kann eine Anfrage an die URL `/auth` mit den angegeben Login Daten des Benutzers gesendet werden. Unter diesem Pfad können wir dann bestimmte Middlewares verwenden, welche sich mit der Authentifizierung des Benutzers befassen.

2.2.2 Socket.io

Die Kommunikation mit dem HTTP-Protokoll hat den Nachteil, dass für jeden Datenaustausch eine neue Verbindung aufgebaut und wieder geschlossen wird, was zu einer Latenz führt, welche für Echtzeit-Anwendungen ungeeignet ist. Diese Problematik behebt das Framework *socket.io*.

Es ermöglicht eine direkte, bidirektionale Echtzeitübertragung von Daten mittels Websockets und fünf anderen Protokollen zwischen den Clients und dem Server. Diese Echtzeitkommunikation ist für viele Multiplayer-Spiele, wie auch für diese Schach-App, essentiell. So können beim Spielen mit Schachuhr Millisekunden entscheidend sein. Neben der Kommunikation mittels Websockets überprüft socket.io unter anderem Timeouts, Verbindungsabbrüche, stellt Verbindungen automatisch wieder her und sorgt dafür, dass die Events in der richtigen Reihenfolge beim Server und beim Client ankommen.

Die Kommunikation mit Socket.io läuft ausschließlich über Events. So kann man sowohl bei dem Client, als auch bei dem Server Eventlistener definieren, die auf ein bestimmtes Event hören und darauf hin eine Funktion auf den übertragenen Daten anwenden. Diese Eventlistener (definiert mit der Funktion `.on()`) haben als ersten Parameter den Namen des Events als String, auf den dieser Listener hören soll und als zweiten Parameter die auszuführende Callback Funktion, welche mit den Parametern aufgerufen wird, die beim senden des Events übertragen wurden.

Events können basierend auf verschiedenen Aktionen wie zum Beispiel dem Drücken eines Buttons im Frontend oder als Reaktion eines eingegangenen Events auf dem Server gesendet werden (mit der Funktion `.emit()`) (siehe Abbildung 8). Der erste Parameter der `emit`-Funktion ist wieder der Name des Events als String und im Anschluss kann man beliebig viele Parameter übertragen mit denen die Callback-Funktion des Listeners aufgerufen wird.

Ein wichtiges Feature von socket.io sind die Räume[5]. Sockets im Backend können ihnen Beitreten und sie Verlassen. Serverseitig kann man dadurch an alle Sockets, die in einem bestimmten Raum sind etwas senden, ohne es allen einzeln schicken zu müssen. Hier kann man sich entschließen das Event an alle clients im Raum zu versenden (`io.to(...).emit(...)`) oder an alle, außer den sender (`socket.to(...).emit(...)`) (siehe Code Snippet 2.2).

Des weiteren erhält jede socket beim Verbinden eine eigene ID, die ebenfalls als Raum genutzt werden kann. Dementsprechend ist `io.to(socket.id).emit('hello');`

[5]Socket.io

äquivalent zu `socket.emit('hello');`.

```
//Server
io.on("connection", (socket) => {
  socket.join("Chat");
  socket.on("message", (text) => {
    socket.to("Chat").emit("message", text);
  }
  //Empfangen der Nachricht und weiterleiten an alle im Raum,
     ausser Sender.
});

//Frontend
...
socket.emit("message", "hello world"); //Senden
socket.on("message", text => console.log(text)); //Empfangen
...
```

Code Snippet 2.2: Beispiel zum Beitreten Raums und das senden eines Events in diesen Raum

//QUELLE socket.io im nodejsbook

2.2.3 **React**

2.2.4 **PostgreSQL**

3. Systemarchitektur

Die Anwendung ist in zwei Hauptkomponenten unterteilt: das Frontend und das Backend. Das Frontend ist für die Darstellung der Benutzeroberfläche und die Interaktion mit dem Benutzer verantwortlich, während das Backend die Spiellogik, die Verwaltung der Benutzerdaten und die Echtzeit-Kommunikation zwischen den Spielern steuert.

Die Anwendung verwendet moderne Web-Technologien, um eine reaktive und benutzerfreundliche Oberfläche zu schaffen. Das Frontend basiert auf dem React-Framework[1], das es ermöglicht, wiederverwendbare Komponenten zu entwickeln und den Anwendungsstatus effizient zu verwalten. Das User-Interface basiert auf Chakra UI[2], einem modernen und flexiblen Komponenten-Bibliothekssystem, das die Entwicklung von responsiven und zugänglichen Benutzeroberflächen erleichtert. Die Benutzerführung und die Kommunikation zwischen den React-Komponenten sind so gestaltet, dass sie eine nahtlose und intuitive Benutzererfahrung bieten.

Auf der Backend-Seite wird Node.js[3] mit dem Express-Framework verwendet, um einen leistungsstarken und skalierbaren Server bereitzustellen. Die API-Endpunkte und die Kommunikation mittels WebSockets is so konzipiert, dass sie den Anforderungen der verschiedenen Frontend-Komponenten gerecht werden und die Kommunikation zwischen Frontend und Backend erleichtern. Die Echtzeit-Kommunikation zwischen den Spielern wird mit Hilfe von Socket.io[4] ermöglicht, einer Bibliothek, die bidirektionale Kommunikation über WebSockets unterstützt. Für die Speicherung und Verwaltung der Benutzerdaten zum Anmelden wird eine PostgreSQL[5]-Datenbank verwendet, die aufgrund ihrer Leistungsfähigkeit und Flexibilität ausgewählt wurde. Freundeslisten und Daten aktiver Spiele werden in einer Redis[6]-Datenbank gespeichert, die sich durch hohe Leistung und niedrige Latenz auszeichnet, insbesondere bei Lese- und Schreibvorgängen. Redis, eine In-Memory-Datenstruktur, eignet sich ideal für Anwendungen, bei denen schnelle Zugriffszeiten und Skalierbarkeit wichtig sind. Die Kombination von PostgreSQL und Redis ermöglicht eine effiziente Verwaltung sowohl persistenter als auch flüchtiger Daten und fördert eine optimale Benutzererfahrung.

[1] Meta Platforms [2023]
[2] Adebayo [2023]
[3] Foundation [2023]
[4] Socket.IO [2023]
[5] Group [2023]
[6] Ltd. [2023]

3.1 Frontend

3.1.1 React-Komponenten

3.1.2 Benutzerführung

3.1.3 Kommunikation mit dem Backend

3.2 Backend

3.2.1 API-Endpunkte

3.2.2 Datenbankstruktur

PostgreSQL

Redis

3.2.3 Echtzeit-Kommunikation

4. Implementierung

4.1 Frontend-Entwicklung

4.2 Backend-Entwicklung

4.3 Datenbank-Integration

5. Tests und Evaluation

5.1 Funktionalitätstests

5.2 Usability-Tests

5.3 Performancetests

5.4 Sicherheitstests

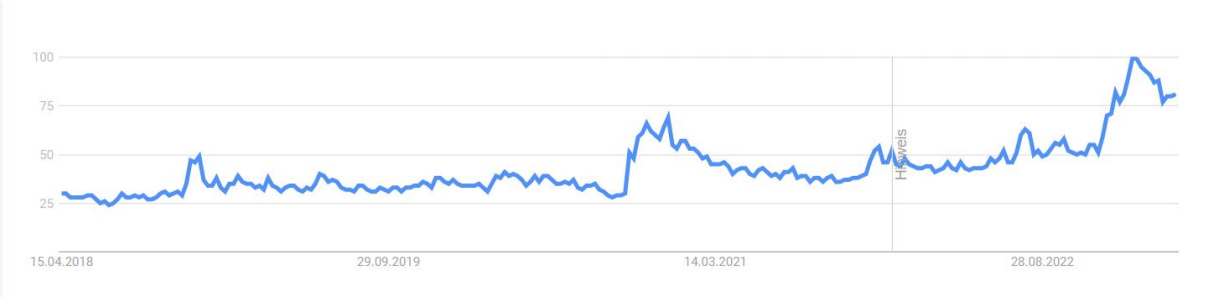
6. Fazit und Ausblick

6.1 Zusammenfassung der Ergebnisse

6.2 Limitationen

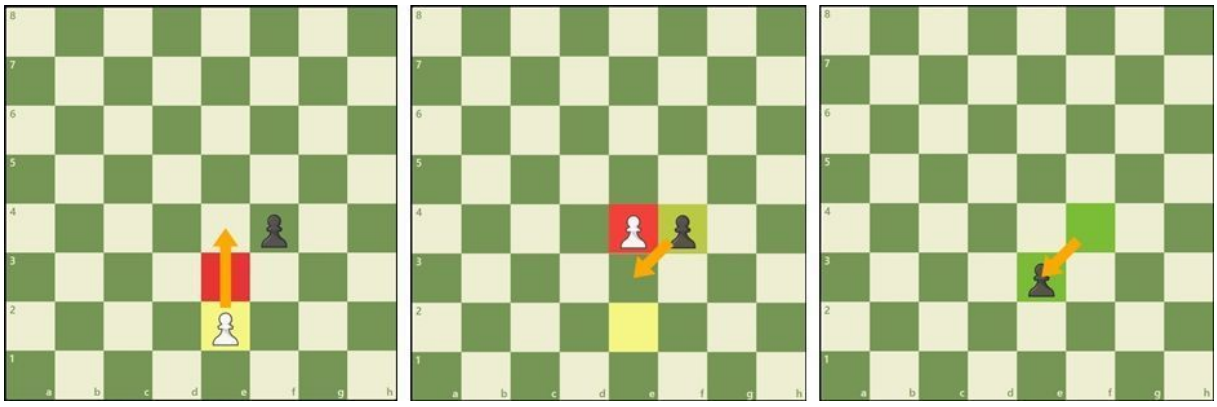
6.3 Potenzielle Erweiterungen und Weiterentwicklung

Abbildungsverzeichnis



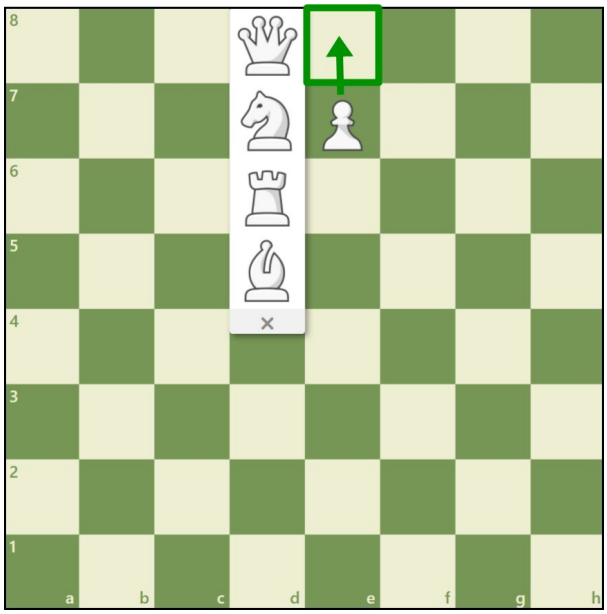
Quelle: `https://trends.google.de/`

Abbildung 1: Relatives Suchinteresse des Wortes *Chess* auf Google in den letzten 5 Jahren.



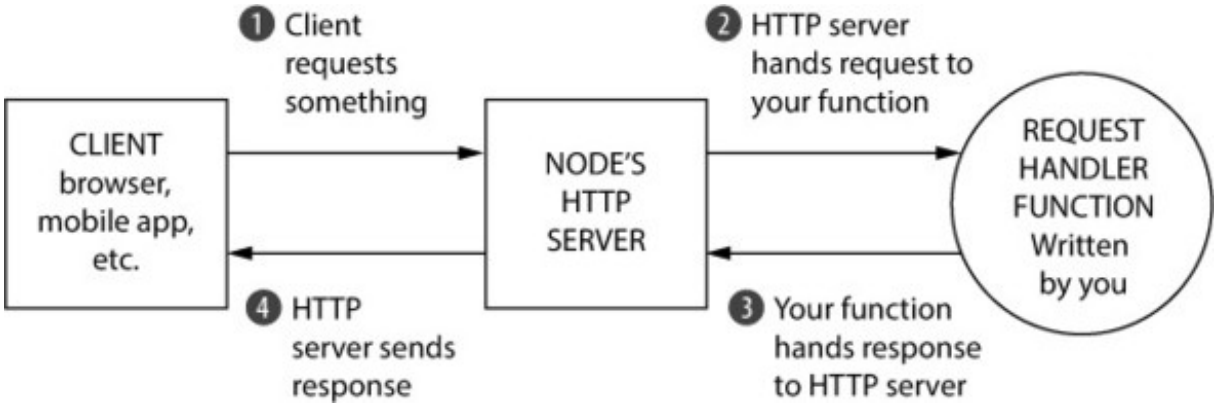
Quelle: `https://www.chess.com/de/schachregeln`

Abbildung 2: Die Zusatzregel *en passant*



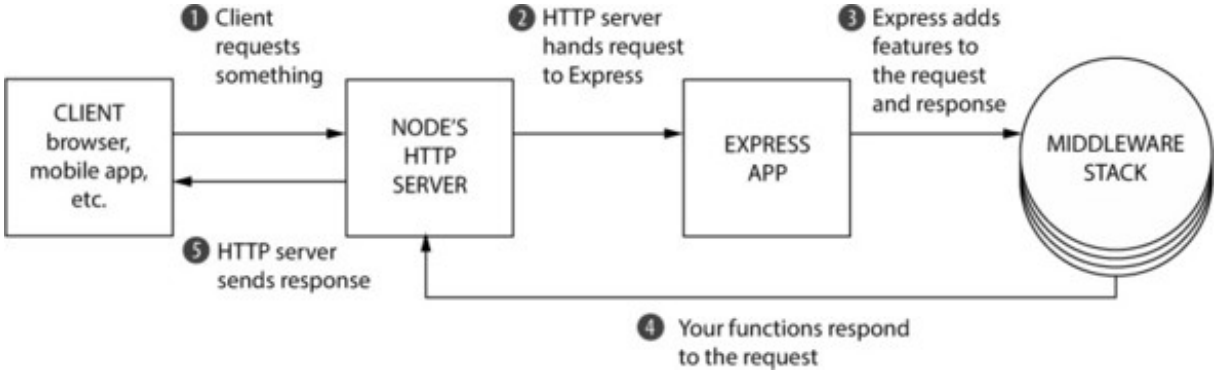
Quelle: `https://www.chess.com/de/schachregeln`

Abbildung 3: Die Zusatzregel *Bauernumwandlung*



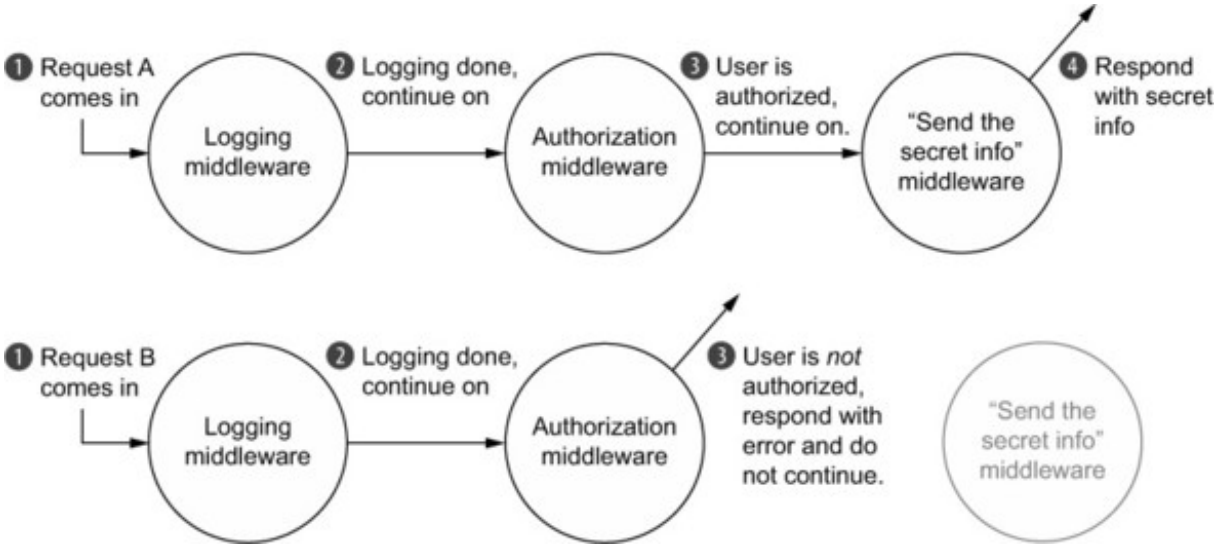
Quelle: Hahn [2016]

Abbildung 4: Ablauf einer Anfrage an einen Node.js Server



Quelle: Hahn [2016]

Abbildung 5: Ablauf einer Anfrage an einen Node.js Server mit Express




```
app.use(function(request, response, next) {
  console.log("In comes a " + request.method + " to " + request.url);
  next();
});
```

The logging middleware, just as before

```
app.use(function(request, response, next) {
  var minute = (new Date()).getMinutes();
  if ((minute % 2) === 0) {
    next();
  } else {
    response.statusCode = 403;
    response.end("Not authorized.");
  }
});
```

If visiting at the first minute of the hour, calls next() to continue on

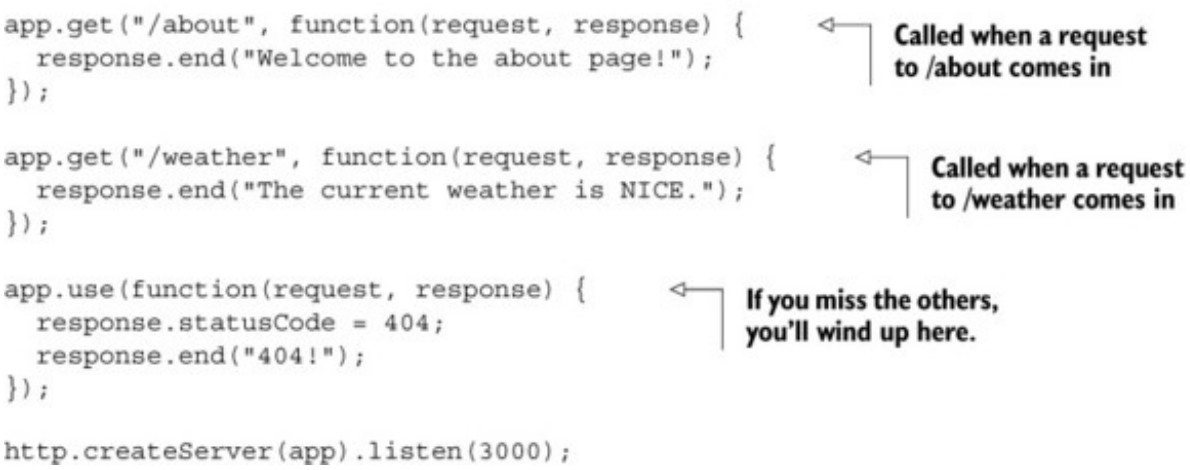
If not authorized, sends a 403 status code and responds

```
app.use(function(request, response) {
  response.end('Secret info: the password is "swordfish"!');
});
```

Sends the secret information

Quelle: Hahn [2016]

Abbildung 6: Beispiel der Nutzung von Middlewares



Quelle: Hahn [2016]

Abbildung 7: Beispiel der Nutzung von Routing

```
import { Server } from "socket.io";

const io = new Server(3000);

io.on("connection", (socket) => {
  // send a message to the client
  socket.emit("hello", "world");

  // receive a message from the client
  socket.on("howdy", (arg) => {
    console.log(arg); // prints "stranger"
  });
});
```

```
import { io } from "socket.io-client";

const socket = io("ws://localhost:3000");

// receive a message from the server
socket.on("hello", (arg) => {
  console.log(arg); // prints "world"
});

// send a message to the server
socket.emit("howdy", "stranger");
```

Quelle:
Socket.IO [2023]

Abbildung 8: Simples Beispiel der Initialisierung einer socket.io Verbindung und das Senden und Empfangen von Events

Literaturverzeichnis

Antonius van der Linde. *Geschichte und Litteratur des Schachspiels, Erster Band.* 1874.

Astrid Hansen. Wie ein schachspiel zum wettstreit der systeme wurde. URL https://www.geo.de/magazine/geo-epoche/19054-rtkl-schachweltmeisterschaft-wie-ein-schachspiel-zum-wettstreit-der.

Sportschau. Nach schach-eklat – ermittlungen gegen niemann und carlsen, 2022. URL https://www.sportschau.de/schach/magnus-carlsen-hans-niemann-ermittlungen-100.html.

chess.com. We've reached 3000000000 live chess games, 2018. URL https://www.chess.com/forum/view/general/weve-reached-3000000000-live-chess-games.

Jennifer Niederst Robbins. *Learning Web Design.* O'REILLY, 2012. ISBN 978-1-449-31927-4.

JSConf. Ryan dahl: Node js, 2012. URL https://www.youtube.com/watch?v=EeYvFl7li9E.

OpenJS Foundation. Node.js - an open-source, cross-platform javascript runtime environment., 2023. URL https://nodejs.org/.

Robert Prediger. *NODE.JS - Professionell hochperformante Software entwickeln.* Carl Hanser Verlag, 2015. ISBN 978-3-446-43722-7.

Evan M. Hahn. *Express in Action - Writing, Builing and Testing Node.js applications.* Manning Publications, 2016. ISBN 978-1-61729-242-2.

Socket.io. Socket.io rooms. URL https://socket.io/docs/v4/rooms/.

Inc. Meta Platforms. React – a javascript library for building user interfaces, 2023. URL https://react.dev/.

Segun Adebayo. Chakra ui - a simple, modular and accessible component library for building react applications, 2023. URL https://chakra-ui.com/.

Socket.IO. Socket.io - bidirectional and low-latency communication for every platform., 2023. URL https://socket.io/.

The PostgreSQL Global Development Group. Postgresql - the world's most advanced open source relational database., 2023. URL https://www.postgresql.org/.

Redis Ltd. Redis - the open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker., 2023. URL https://redis.io/.