Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Entwicklung einer Hotelbuchungssoftware

Development of hotel booking software

Alexander Falkenberg, Hasan Alhelal

Teamprojekt

Betreuer: Titel Vorname Nachname

Ort, DD.MM.YYYY

Vorwort

Ein Vorwort ist nicht unbedingt nötig. Falls Sie ein Vorwort schreiben, so ist dies der Platz, um z.B. die Firma vorzustellen, in der diese Arbeit entstanden ist, oder um den Personen zu danken, die in irgendeiner Form positiv zur Entstehung dieser Arbeit beigetragen haben.

Auf keinen Fall sollten Sie im Vorwort die Aufgabenstellung näher erläutern oder vertieft auf technische Sachverhalte eingehen.

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

Abstract

The same in English.

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Einleitung

1.1 Motivation

Diese Arbeit handelt von der Umsetzung eines Buchungssystems für ein Hotel. Ein solches System ist besonders wichtig um den allgemeinen Aufwand beim Verwalten von verfügbaren Zimmern deutlich zu verringern und der Fehleranfälligkeit beim Vergeben von eben diesen vorzubeugen. Außerdem kann diese Software in leicht abgeänderter Form auch problemlos für andere Anwendungen benutzt werden.

1.2 Ziele der Arbeit

Ziel dieser Arbeit ist es ein simples Buchungssystem zu schaffen dass sich ohne großen Arbeitsaufwand verwenden lässt. Es soll den Nutzern des Systems möglich sein ohne die Erstellung eines Nutzerprofils Buchungen tätigen und diese verwalten zu können. Die Informationen über jede Buchung werden den Nutzern anschließend per Email zugesandt.

2

Verwandte Arbeiten

Die Gliederung hängt natürlich vom Thema und von der Lösungsstrategie ab. Als nützliche Anhaltspunkte können die Entwicklungsstufen oder -schritte z.B. der Software-Entwicklung betrachtet werden. Nützliche Gesichtspunkte erhält und erkennt man, wenn man sich

- in die Rolle des Lesers oder
- in die Rolle des Entwicklers, der die Arbeit z.B. fortsetzen, ergänzen oder pflegen soll,

versetzt. In der Regel wird vorausgesetzt, dass die Leser einen fachlichen Hintergrund haben - z.B. Informatik studiert haben. D.h. nur in besonderen, abgesprochenen Fällen schreibt man in populärer Sprache, so dass auch Nicht-Fachleute die Ausarbeitung prinzipiell lesen und verstehen können.

Die äußere Gestaltung der Ausarbeitung hinsichtlich Abschnittformate, Abbildungen, mathematische Formeln usw. wird in Kapitel **??** kurz dargestellt.

3

Grundlagen

3.1 HTML, CSS, JS

3.1.1 HTML

3.1.2 Less

3.1.3 JavaScript

3.2 Node.js

Die serverseitige Grundlage des Systems bildet die Laufzeitumgebung Node.js. Mit dieser ist es möglich JavaScript Programme abseits eines Browsers ausführen zu können. Mithilfe von „npm“, dem Paketmanager von Node.js, werden außerdem externe Pakete für bestimmte Funktionalitäten verwendet um die Implementierung zu vereinfachen.

3.2.1 Express

Das Express-Framework ermöglicht eine unkomplizierte Erstellung einer Webanwendung. So wird beispielsweise die Verarbeitung von HTTP-Anfragen wird mithilfe von spezifischen Methoden deutlich vereinfacht und macht das Erstellen einer API besonders effizient. Innerhalb der Buchungssoftware sorgt Express für die Bereitstellung eines Servers, der alle clientseitigen Dateien zur Verfügung stellt und auf HTTP-Anfragen zur Erzeugung oder zum Abrufen von Buchungen, reagiert.

3.2.2 Nodemailer

Nodemailer ist ein „npm“-Paket, das dafür verwendet wird E-Mails mit Node.js zu versenden. Durch Angabe der Adresse von der die E-Mail verschickt werden soll und verschiedenen Optionen wie z.B. dem Betreff wird das Senden vereinfacht. Um eine E-Mail zu versenden muss im ersten Schritt ein „Transporter“zusammen mit dem verwendeten Mail Dienst und den Anmeldeinformationen der zu sendenen E-Mail Adresse, erzeugt werden. Anschließend werden die Mail-Optionen initialisiert und über die Methode „sendMail“des Transportes, die E-Mail versandt.

3.2.3 Formidable

3.3 MongoDB

Unter den dokumentenorientierten NoSQL-Datenbanken ist MongoDB die meist verwendete. Die wichtigsten Bestandteile einer MongoDB Datenbank sind Dokumente und Collections. Dokumente sind Datenstrukturen die sich aus einem oder mehreren Paaren, bestehend aus Feldern und dazugehörigen Werten, zusammenfügen. In ihrem Aufbau sind diese Dokumente identisch mit JSON Objekten. Collections hingegen sind Sammlungen von verschiedenen Dokumenten und sind vergleichbar mit Tabellen aus relationalen Datenbanken.

Damit eine Node.js-Anwendung mit einer MongoDB kommunizieren kann wird beispielsweise das „npm“-Paket „mongodb“benötigt. Hier können dann auf den einzelnen Collections, Methoden angwendet werden. Die wichtigsten Methoden für das zu erstellende System sind „find“, „insertOne“, „deleteOne“und „deleteMany“. Mithilfe von find werden alle Dokumente einer Collection geliefert, auf die ein in der Methode spezifiziertes Kriterium zutrifft. InsertOne ermöglicht es ein neues Dokument in eine Collection hinzuzufügen und deleteOne bzw. deleteMany ermöglicht das Löschen von Dokumenten aus einer Collection.

4

Konzept

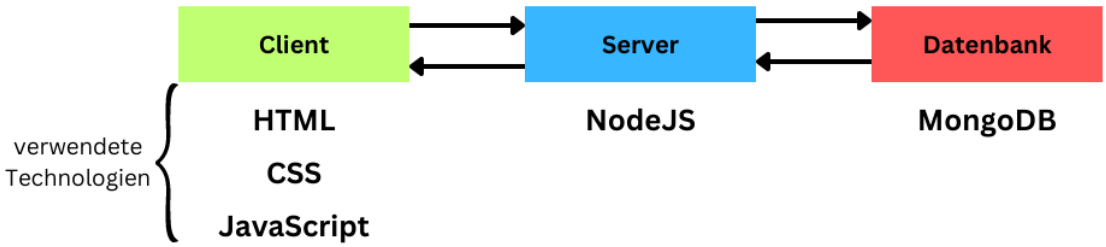

Abbildung 4.1: Grobe Architektur des Systems

5

Realisierung

Die Gliederung hängt natürlich vom Thema und von der Lösungsstrategie ab. Als nützliche Anhaltspunkte können die Entwicklungsstufen oder -schritte z.B. der Software-Entwicklung betrachtet werden. Nützliche Gesichtspunkte erhält und erkennt man, wenn man sich

- in die Rolle des Lesers oder
- in die Rolle des Entwicklers, der die Arbeit z.B. fortsetzen, ergänzen oder pflegen soll,

versetzt. In der Regel wird vorausgesetzt, dass die Leser einen fachlichen Hintergrund haben - z.B. Informatik studiert haben. D.h. nur in besonderen, abgesprochenen Fällen schreibt man in populärer Sprache, so dass auch Nicht-Fachleute die Ausarbeitung prinzipiell lesen und verstehen können.

Die äußere Gestaltung der Ausarbeitung hinsichtlich Abschnittformate, Abbildungen, mathematische Formeln usw. wird in Kapitel **??** kurz dargestellt.

6

Implementierung

7

Beispiele

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

7.1 Warum existieren unterschiedliche Konsistenzmodelle?

Laut [Mal97] sind mit der Replikation von Daten immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz der Daten. Die Form der Konsistenzsicherung bestimmt dabei, inwiefern das eine Kriterium erfüllt und das andere dementsprechend nicht erfüllt ist (Trade-off zwischen Verfügbarkeit und der Konsistenz der Daten). Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien der Daten existieren, dürfen keine Abweichungen auftreten. Die Verfügbarkeit der Daten ist hier jedoch stark eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten, wobei die Konsistenz nur an bestimmten Synchronisationspunkten gewährleistet wird. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen.

Nach [Mos93] kann die Performanzsteigerung der schwächeren Konsistenzmodelle wegen der Optimierung (Pufferung, Code-Scheduling, Pipelines) 10-40 Prozent betragen. Wenn man bedenkt, dass mit der Nutzung der vorhandenen Synchronisierungsmechanismen schwächere Konsistenzmodelle den Anforderungen der strengen Konsistenz genügen, stellt sich der höhere programmiertechnischer Aufwand bei der Implementierung der schwächeren Konsistenzmodelle als ihr einziges Manko dar.

In [Che85] ist beschrieben, wie man sich Formen von DSM vorstellen könnte, für die ein beachtliches Maß an Inkonsistenz akzeptabel wäre. Beispielsweise könnte DSM verwendet werden, um die Auslastung von Computern in einem Netzwerk zu speichern, so dass Clients für die Ausführung ihrer Applikationen die am wenigsten ausgelasteten Computer auswählen können. Weil die Informationen dieser Art innerhalb kürzester Zeit ungenau werden können (und durch die Verwendung der

veralteten Daten keine großen Nachteile entstehen können), wäre es vergebliche Mühe, sie ständig für alle Computer im System konsistent zu halten [CDK02]. Die meisten Applikationen stellen jedoch strengere Konsistenzanforderungen.

7.2 Klassifizierung eines Konsistenzmodells

Die zentrale Frage, die für die Klassifizierung (streng oder schwach) eines Konsistenzmodells von Bedeutung ist [CDK02]: wenn ein Lesezugriff auf eine Speicherposition erfolgt, welche Werte von Schreibzugriffen auf diese Position sollen dann dem Lesevorgang bereitgestellt werden? Die Antwort für das schwächste Konsistenzmodell lautet: von jedem Schreibvorgang, der vor dem Lesen erfolgt ist, oder in der „nahen“ Zukunft, innerhalb des definierten Betrachtungsraums, erfolgten wird. Also irgendein Wert, der vor oder nach dem Lesen geschrieben wurde.

Für das strengste Konsistenzmodell, Linearisierbarkeit (atomic consistency), stehen alle geschriebenen Werte allen Prozessoren sofort zur Verfügung: eine Lese-Operation gibt den aktuellsten Wert zurück, der geschrieben wurde, bevor das Lesen stattfand. Diese Definition ist aber in zweierlei Hinsicht problematisch. Erstens treten weder Schreib- noch Lese-Operationen zu genau einem Zeitpunkt auf, deshalb ist die Bedeutung von „aktuellsten“ nicht immer klar. Zweitens ist es nicht immer möglich, genau festzustellen, ob ein Ereignis vor einem anderen stattgefunden hat, da es Begrenzungen dafür gibt, wie genau Uhren in einem verteilten System synchronisiert werden können.

Nachfolgend werden einige Konsistenzmodelle absteigend nach ihrer Strenge vorgestellt. Zuvor müssen wir allerdings klären, wie die Lese- und Schreibe-Operationen in dieser Ausarbeitung dargestellt werden.

Sei x eine Speicherposition, dann können Instanzen dieser Operationen wie folgt ausgedrückt werden:

- $R(x)a$ - eine Lese-Operation, die den Wert a von der Position x liest.
- $W(x)b$ - eine Schreib-Operation, die den Wert b an der Position x speichert.

7.3 Linearisierbarkeit (atomic consistency)

Die Linearisierbarkeit im Zusammenhang mit DSM kann wie folgt definiert werden:

- Die verzahnte Operationsabfolge findet so statt: wenn $R(x)a$ in der Folge vorkommt, dann ist die letzte Schreib-Operation, die vor ihr in der verzahnten Abfolge auftritt, $W(x)a$, oder es tritt keine Schreib-Operation vor ihr auf und a ist der Anfangswert von x. Das bedeutet, dass eine Variable nur durch eine Schreib-Operation geändert werden kann.
- Die Reihenfolge der Operationen in der Verzahnung ist konsistent zu den Echtzeiten, zu denen die Operationen bei der tatsächlichen Ausführung aufgetreten sind.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$		$W(y)2$	
P_2		$R(x)1$		$R(y)2$

Tabelle 7.1: Linearisierbarkeit ist erfüllt

Die Bedeutung dieser Definition kann an folgendem Beispiel (Tabelle 7.1) nachvollzogen werden. Es sei angenommen, dass alle Werte mit 0 vorinitialisiert sind.

Hier sind beide Bedingungen erfüllt, da die Lese-Operationen den zuletzt geschriebenen Wert zurückliefern. Interessanter ist es, zu sehen, wann die Linearisierbarkeit verletzt ist.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)0$	$R(x)2$

Tabelle 7.2: Linearisierbarkeit ist verletzt, sequentielle Konsistenz ist erfüllt.

In diesem Beispiel (Tabelle 7.2) ist die Echtzeit-Anforderung verletzt, da der Prozess P_2 immer noch den alten Wert liest, obwohl er von Prozess P_1 bereits geändert wurde. Diese Ausführung wäre aber sequentiell konsistent (siehe kommender Abschnitt), da es eine Verzahnung der Operationen gibt, die diese Werte liefern könnte ($R(x)0, W(x)1, W(x)2, R(y)2$). Würde man beide Lese-Operationen des 2. Prozesses vertauschen, wie in der Tabelle 7.3 dargestellt, so wäre keine sinnvolle Verzahnung mehr möglich.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)2$	$R(x)0$

Tabelle 7.3: Linearisierbarkeit und sequentielle Konsistenz sind verletzt.

In diesem Beispiel sind beide Bedingungen verletzt. Selbst wenn die Echtzeit, zu der die Operationen stattgefunden haben, ignoriert wird, gibt es keine Verzahnung einzelner Operationen, die der Definition entsprechen würde.

8

Anwendungsszenarien

9

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

Literaturverzeichnis

CDK02. Coulouris, George, Jean Dollimore und Tim Kindberg: *Verteilte Systeme: Konzepte und Design*. Addison Wesley, München, 3. Auflage, 2002.

Che85. Cheriton, David R.: *Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems*. ACM SIGOPS Operating Systems Review, 19(4):26–33, 1985.

CLR90. Cormen, Thomas H., Charles E. Leiserson und Ronald L. Rivest: *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

GO03. Gottwald, Tim und Rainer Oechsle: *DisASTer (Distributed Algorithms Simulation Terrain): A Platform for the Implementation of Distributed Algorithms*. Diplomarbeit, Fachhochschule Trier, Fachbereich Informatik, 2003.

Mal97. Malte, Peter: *Replikation in Mobile Computing*. Seminar 31/1997, Universität Karlsruhe, Institut für Telematik, 1997.

Mos93. Mosberger, David: *Memory Consistency Models*. Technical Report 93/11, University of Arizona, Department of Computer Science, November 1993.

Nan03. Nannen, Volker: *The Paradox of Overfitting*. Master-Abschlussarbeit, Rijksuniversiteit Groningen, 2003.

Wik. *Wikipedia: Programmausdruck*. `https://de.wikipedia.org/wiki/Programmausdruck`. Abgerufen am 13.10.2021.

A

Selbstständigkeitserklärung

☐ Diese Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

☐ Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Meine eigene Leistung ist:

Datum

Unterschrift der Kandidatin/des Kandidaten