HAW HAMBURG

BACHELORTHESIS
Torsten Huhn

Konzeption und Implementierung eines Prototyps einer Webanwendung unter besonderer Berücksichtigung von Thread-Pools

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Torsten Huhn

Konzeption und Implementierung eines Prototyps einer Webanwendung unter besonderer Berücksichtigung von Thread-Pools

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Schäfers
Zweitgutachter: Prof. Dr. Axel Schmolitzky

Eingereicht am: 09. November 2021

Torsten Huhn

Thema der Arbeit

Konzeption und Implementierung eines Prototyps einer Webanwendung unter besonderer Berücksichtigung von Thread-Pools

Stichworte

Java, Thread-Pools, Webanwendung, Spring

Kurzzusammenfassung

Im Rahmen dieser Bachelorarbeit wird untersucht, wie eine in Java geschriebene Webanwendung entwickelt werden kann, sodass diese auch bei hoher Auslastung stabil bleibt. Insbesondere werden die Möglichkeiten der Thread-Pools genauer betrachtet. Zu diesem Zweck wird eine Webanwendung entwickelt, die einen typischen Anwendungsfall im Studium der Technischen Informatik an der Hochschule für Angewandte Wissenschaften abbildet. Die Studierenden müssen im Laufe des Studiums eine Vielzahl von Programmen entwickeln. Damit der Lehrende ein besseren Einblick über den Verlauf der Entwicklung bekommt, geschieht dies über das verteilte Versionsverwaltungssystem GIT. Die Software bietet dem Studierenden die Möglichkeit GIT-Repositories zu melden und dem Lehrenden eine zentrale Stelle für die Verwaltung der Repositories. Es wird eine Variante in reinem Java fast ohne zusätzliche Bibliotheken implementiert, um den Aspekt der Thread-Pools genauer beleuchten zu können und eine weitere mithilfe des Spring-Frameworks, um im Anschluss die Performance der beiden Implementierungen vergleichen zu können.

Torsten Huhn

Title of Thesis

Conception and implementation of a prototype of a web application with special consideration of thread pools

Keywords

Java, Thread Pools, Webapplikation, Spring

Abstract

In the context of this bachelor thesis, it was investigated how a web application written in Java can be developed so that it remains stable even under high load. In particular, the possibilities of thread pools were examined in more detail. For this purpose, a web application was developed that represents a typical use case in the study of computer engineering at the University of Applied Sciences. Students have to develop a large number of programs during their studies. In order to provide the teacher with a better insight into the development, this is done via the distributed version management system GIT. The software offers the student the possibility to report GIT repositories and the teacher a central place for the administration of the repositories. A variant was implemented in pure Java, almost without additional libraries, in order to be able to examine the aspect of thread pools more closely, and another was implemented using the Spring Framework in order to be able to compare the performance of the two implementations afterwards.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die generelle Relevanz von Webanwendungen hat in den letzten Jahren sehr zugenommen. Zu Beginn des World Wide Web beschränkte es sich ausschließlich auf statische Seiten, die nur informeller Natur waren. Im Laufe der Zeit kamen immer mehr Interaktionsmöglichkeiten hinzu und auch immer mehr klassische Desktopanwendungen, wie zum Beispiel Textverarbeitungsprogramme, fanden den Weg in das Internet. Laut dem Statistischen Bundesamt ist die Anzahl der Personen, die das Internet nutzen, in der letzten Dekade von 75 % auf 90 % gewachsen. Es sind also inzwischen neun von zehn Leuten online. Aus diesem Grund ist es wichtig, dass Webanwendungen und Clouddienste einer Vielzahl von parallelen Zugriffen standhalten können. Hierfür wird nicht nur eine entsprechend starke Infrastruktur benötigt, auch die Webanwendungen sollten so entwickelt werden, dass sie trotz hoher Auslastung noch ihren Dienst erfüllen können. Neue Prozessoren haben nicht unbedingt immer mehr Leistung pro Prozessorkern, sondern zeichnen sich häufig dadurch aus, dass immer mehr Kerne in einem Prozessor verbaut sind. Dementsprechend ist es wichtig, dass die Software dafür ausgelegt ist. Dies hat mich dazu bewogen, im Rahmen der vorliegenden Bachelorthesis Fragestellungen im Zusammenhang mit Thread-Pools aufzustellen und praxisbezogen anhand einer selbst entwickelten Software zu analysieren.[25]

1.1 Ziel

Das Ziel dieser Arbeit ist es, eine Webanwendung zu entwickeln und dabei insbesondere das Thema Thread-Pools im Bezug auf Konfigurationsmöglichkeiten und Performance zu beleuchten. Dafür wurde eine Webanwendung in reinem Java geschrieben, um den Aspekt der Thread-Pools besser untersuchen zu können und eine weitere Implementierung mit den gleichen Funktionen mit Unterstützung des Spring Frameworks. Im Folgenden wurden die beiden Implementierungen durch einen Lasttest im Bezug auf Stabilität verglichen.

1.2 Aufbau der Arbeit

Im ersten Kapitel werden die Grundlagen von Webanwendungen und Thread-Pools erläutert. Die Anforderungsanalyse untersucht funktionale und nicht-funktionale Anforderungen der Webanwendung. Im dritten Kapitel werden Konzepte der Implementierung dargestellt. Um die beiden Implementierungen besser vergleichen zu können, wurde ein Lasttest durchgeführt. Der Testaufbau und die Ergebnisse werden in dem Kapitel „5 Test“ veranschaulicht. In den letzten beiden Kapiteln wird die Webanwendung bewertet und zum Schluss ein Fazit gezogen.

Anmerkung: Im Verlauf der vorliegenden Bachelorthesis sind stets Personen männlichen und weiblichen Geschlechts gleichermaßen gemeint; aus Gründen der einfacheren Lesbarkeit wird im Folgenden nur die männliche Form verwendet.

2 Grundlagen

In diesem Kapitel werden der grundlegende Aufbau einer Webanwendung und die gängigen Technologien in diesem Zusammenhang erläutert. Des Weiteren werden die Grundlagen von Nebenläufigkeit in der Softwareentwicklung beschrieben.

2.1 Verteilte Systeme

Ein verteiltes Computer System ist eine Sammlung von unabhängigen Systemen, die Nachrichten über ein Netzwerk austauschen und gemeinsam eine Aufgabe lösen. Die Kooperation der Systeme findet dementsprechend über Rechnergrenzen hinweg statt und die Systeme können auch örtlich von einander getrennt sein. Der Benutzer bekommt den Eindruck, dass es sich um ein zusammenhängendes System handelt. Ihm wird auch nicht deutlich, wo sich das System örtlich befindet oder ob es den Ort wechselt. Auch wenn viele andere Benutzer auf dieselbe Ressource zugreifen, sollte der Benutzer davon nichts merken. Treten Fehler auf, sollte das System in der Lage sein, dieses möglichst unsichtbar und zeitnah zu beheben. Der Benutzer bekommt den Eindruck, dass es sich um ein einzelnes System handelt und nicht um viele verschiedene. Diese Tatsache wird als „Transparenz“ bezeichnet. Andrew S. Tanenbaum definiert ein verteiltes System folgendermaßen. [29]

> “A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.”
>
> Andrew S. Tanenbaum

Verteilte Systeme lösen viele Aufgaben, bringen aber auch neue Herausforderungen mit sich. Es entstehen Abhängigkeiten vom Netzwerk und von Teilkomponenten. Das System hat keinen gemeinsamen Zustand bzw. keine gemeinsame Uhr und somit müssen Synchronisations- und Koordinationsprobleme gelöst werden.

Verteilte Systeme bilden die Grundlage für die in Abschnitt 2.4.1 beschriebene n-Tier Architektur. Die unterschiedlichen Formen der Transparenz sind in Tabelle 2.1 aufgelistet. In dem Fall der entwickelten Webanwendung ist es dem Benutzer nicht ersichtlich, dass die Anwendung auf mehrere Systeme verteilt ist (2.1 - Zugriff) und an welchem Ort sie sich befindet (2.1 - Ort). Die Tatsache, wie viele Benutzer gleichzeitig Zugreifen, bleibt im verborgenen (2.1 - Nebenläufigkeit). Die anderen Formen der Transparenz spielen in der Webanwendung keine wesentliche Rolle und werden nur der Vollständigkeit halber aufgelistet.

Tabelle 2.1: Formen der Transparenz

Transparenz	Beschreibung
Zugriff	Verbirgt die Art und Weise, wie auf Daten zugegriffen wird.
Ort	Verbirgt, wo sich eine Ressource befindet.
Migration	Verbirgt, dass sich eine Ressource an einen anderen Ort verschoben hat.
Relokation	Verbirgt, dass sich eine Ressource an einen anderen Ort verschoben hat, während sie genutzt wird.
Replikation	Verbirgt, dass eine Ressource repliziert ist.
Nebenläufigkeit	Verbirgt, dass eine Ressource von mehreren konkurrierenden Benutzern gleichzeitig genutzt werden kann.
Fehler	Verbirgt den Ausfall und die Wiederherstellung einer Ressource.

2.2 Web-Technologien

Die im Rahmen dieser Bachelorarbeit entwickelte Webanwendung benutzt neben Java auch Hypertext Markup Language (HTML) und JavaScript. Diese sowie weitere Technologien der Webentwicklung werden in dem folgenden Abschnitt erklärt.

2.2.1 HTTP

Das Hypertext Transfer Protocol (HTTP) definiert, wie Client und Server miteinander kommunizieren können. Zur Adressierung von Ressourcen gibt es die Uniform Resource Locator (URL). In der folgenden Liste sind die wichtigsten Request Arten beschrieben, die das HTTP-Protokoll bietet.[15]

- **GET** - fordert anhand einer Uniform Resource Identifier (URI) eine Ressource an
- **POST** - schickt zB. Formulardaten an den Server und erstellt eine neue Ressource
- **PUT** - die Daten einer bestehenden Ressource werden mit den Daten aus dem Request Body aktualisiert
- **DELETE** - löscht die Daten einer bestimmten Ressource

2.2.2 HTML

HTML ist eine textbasierte Auszeichnungssprache für Internetseiten. Durch diverse Auszeichnungselemente, den sogenannten Tags, die vom Internetbrowser interpretiert werden, lässt sich eine Seite strukturieren. In der fünften Version bietet HTML unter anderem eine Application Programming Interface (API) für „Geolocation“, um den eigenen Standort zu bestimmen, Optimierungen für mobile Endgeräte, Offline-Nutzung und bessere Ergebnisse für Suchmaschinen. Da HTML aber eine statische Sprache bleibt, lassen sich diese API's häufig nur verwenden indem zusätzlich JavaScript verwendet wird.[14]

2.2.3 JavaScript

JavaScript wurde 1995 von Netscape veröffentlicht und dient der clientseitigen, dynamischen Veränderung von Internetseiten. Es können Benutzerinteraktionen ausgewertet, Inhalte verändert oder nachträglich geladen werden. Inzwischen findet JavaScript auch auf der Serverseite Verwendung. Populär wurde dies durch die Veröffentlichung von Node.Js im Jahr 2009. Es bietet eine JavaScript-Laufzeitumgebung und ist gut geeignet für Multi-User und Echtzeit-Web-Anwendungen. Java unterscheidet sich eindeutig von JavaScript. Java ist eine objektorientierte Programmiersprache, die zu Bytecode kompiliert wird und in einer Java Virtual Machine (JVM) ausgeführt wird. Im Gegensatz dazu ist JavaScript eine objektorientierte Skriptsprache, die in der Regel von einem Browser interpretiert wird. [11]

2.2.4 REST

Representational State Transfer (REST) ist weder Protokoll noch Standard, hat sich aber im Bereich der Maschine-zu-Maschine-Kommunikation, zum Beispiel wenn Backend-Server untereinander Daten austauschen, als Quasi-Standard etabliert. REST-Endpunkte werden eindeutig durch eine URI identifiziert und benutzen die oben genannten HTTP-Standardmethoden, um mit Ressourcen zu interagieren. Zum Datenaustausch wird häufig das JavaScript Object Notation (JSON) - Format verwendet. [9]

2.2.5 DOM

Um das dynamische Verändern einer Seite mit JavaScript zu vereinheitlichen wurde das Document Object Model (DOM) etabliert. Dies ist eine standardisierte Programmierschnittstelle für die Strukturierung von HTML in einer Baumstruktur. Dadurch ist das Hinzufügen, Verändern oder Löschen bestimmter Knoten bzw. HTML-Tags möglich und somit lässt sich clientseitig die Seite im Prinzip beliebig verändern, ohne dass mit dem Server kommuniziert werden muss.[13]

2.3 Aufbau einer Webanwendung

Eine Webanwendung ist eine Software die auf einem Webserver im Internet ausgeführt wird. Im Gegensatz zu herkömmlichen Desktop-Anwendungen benötigt der Benutzer immer eine aktive Internetverbindung. Die meisten Webanwendungen basieren auf dem Client-Server-Prinzip. Abbildung 2.1 zeigt einen Client der eine Anfrage (Request) an einen Server schickt. Dieser verarbeitet die Anfrage entsprechend und schickt die Antwort (Response) zurück an den Client.

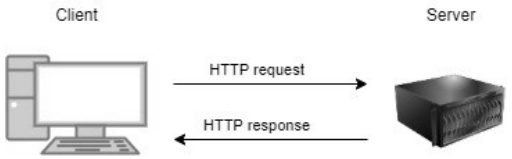

Abbildung 2.1: Client-Server

2.3.1 Webserver

Ein Webserver nimmt hauptsächlich die HTTP-Anfragen eines Clients entgegen und beantwortet diese entsprechend. Hierbei handelt es sich um statischen Inhalt wie zum Beispiel HTML-Seiten, Bilder, Videos oder Dateien. Die erste Technologie, die dynamische Internetseiten möglich machte, war das Common Gateway Interface (CGI). Dieser Standard erlaubt es dem Webserver mit einer externen Software zu kommunizieren und somit dynamische Inhalte zu erstellen. Am Anfang wurden diese Programme in der Programmiersprache Perl geschrieben. Es sind aber im Prinzip alle Programmiersprachen denkbar, nicht nur Skriptsprachen wie zum Beispiel Ruby oder Python, sondern auch zu Maschinencode kompilierbare Sprachen wie C oder C++. Nachteile sind die Plattformabhängigkeit und Limitierungen im Bezug auf Effizienz und Skalierbarkeit, da jede Anfrage einen neuen Prozess erstellt. [18]

2.3.2 Anwendungsserver

Die Grenze zwischen Web- und Anwendungsserver kann mittlerweile nicht mehr exakt gezogen werden, da ein Anwendungsserver häufig HTTP als primäres Protokoll verwendet und Webserver durch Skriptsprachen auch dynamische Inhalte ermöglichen. Hauptsächlich ist der Anwendungsserver dafür zuständig, Interaktionen zwischen der clientseitigen und der serverseitigen Anwendung, auch Geschäftslogik genannt, zu ermöglichen. Der Anwendungsserver bietet häufig eine Sammlung von Diensten, die von der Anwendung benutzt werden kann. Auf einem Anwendungsserver können mehrere Anwendungen gleichzeitig bereitgestellt werden. Grundlegend besteht ein Jakarta Enterprise Edition ehemals JAVA Enterprise Edition (Jakarta EE) Anwendungsserver aus zwei Containern. Der Web Container (siehe Abbildung 2.2) nimmt HTTP-Anfragen entgegen. Anhand der enthaltenen Informationen aus der Anfrage wird ein Java-Objekte erstellt. Das Objekt wird an das entsprechende Servlet, ein Java-Objekt, das Methoden zur Bearbeitung der Anfrage bereitstellt, weitergeleitet. Jakarta Server Pages (JSP) sind HTML-Seiten mit eingebettetem Java-Code, die zu einem Servlet kompiliert werden.

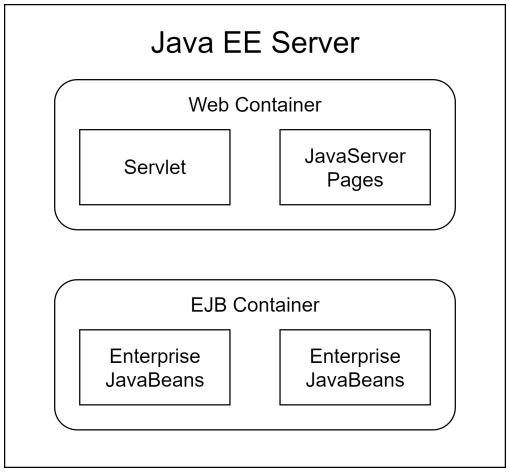

Abbildung 2.2: Anwendungsserver

Im zweiten Container werden die Jakarta Enterprise Beans früher Enterprise JavaBeans (EJB) (siehe Abbildung 2.2) verwaltet. EJB sind einfache Java-Klassen, auch Plain Old Java Object (POJO) genannt und bilden die Geschäftslogik einer Anwendung ab. Sie sind wiederverwendbare Software-Komponenten und müssen folgende Kriterien erfüllen.[16]

- Standard-Konstruktor - public und ohne Parameter
- Zugriffsmethoden - getter und setter
- Implementiert das Serializable-Interface

2.3.3 Backend

Das Backend ist eine Anwendung oder eine Sammlung von Anwendungen, die auf einem Server platziert ist. Seine Hauptaufgabe ist es Anfragen anzunehmen und zu beantworten. Grundlegend besteht es aus einem Web- bzw. Anwendungsserver. Dieser ist dafür zuständig die Anfragen anzunehmen und an die Anwendung weiterzuleiten. Die Anwendung verarbeitet die Anfrage weiter und führt die Geschäftslogik aus. Sie arbeitet meistens mit einer Datenbank zusammen, um die Daten zu speichern.

2.3.4 Frontend

Das Frontend ist die Präsentationsebene einer Anwendung, die der Benutzer sehen kann und über die Aktionen im Backend ausgelöst werden. Auf dem Server zählen die Servlets und das Erstellen der HTML-Seite mit zum Frontend. Die erstellte Internetseite wird bei dem Anwender im Webbrowser dargestellt. Sie wird durch HTML-Tags strukturiert. Das Design wird mittels Cascading Style Sheets (CSS) definiert. Durch das mitliefern von JavaScript kann der Browser auch selbstständig Funktionen ausführen und ist somit nicht zwangsläufig auf die Kommunikation mit dem Server angewiesen, um die dargestellte Seite zu verändern. Die Abbildung 2.3 zeigt wie die ausgelieferte Seite das DOM clientseitig mit Javascript verändert und somit eine veränderte Benutzeroberfläche erzeugt. [4]

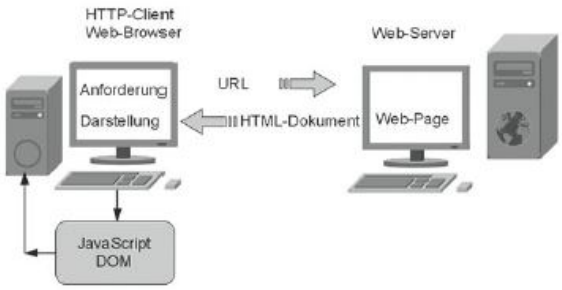

Abbildung 2.3: Clientseitige Dynamik [18]

2.4 Architektur

Der folgende Abschnitt beschreibt die wichtigsten Softwarearchitekturen, die bei der Entwicklung der Webanwendung verwendet wurden.

2.4.1 n-Tier-Architektur

Die n-Tier-Architektur besagt, dass sich eine Anwendung vertikal in beliebig viele (n) physikalisch voneinander getrennten Systeme (*Tier*) aufteilen lässt. In einer Webanwendung sind das meistens die folgenden drei Teilaufgaben. Die Speicherung der Daten, die Geschäftslogik, die Methoden zur Verarbeitung der Daten bietet und die Präsentation, die dem Benutzer die Anwendung zeigt und mit der er interagieren kann.

2-Tier Architektur

Die 2-Tier Architektur basiert auf dem klassischen Client-Server-Modell. Wobei der Client rein für die Darstellung der Anwendung zuständig ist und der Server alle Funktionen der Geschäftslogik abbildet und die Datenbank bereitstellt.

3-Tier Architektur

Die 3-Tier Architektur besteht wie die 2-Tier Architektur aus einem Client und einem Server. Zusätzlich ist die Datenbank auf einem weiteren physisch getrennten System untergebracht (siehe Abbildung 2.4). Diese Aufteilung ist insbesondere bei hoher Auslastung der Webanwendung sinnvoll.

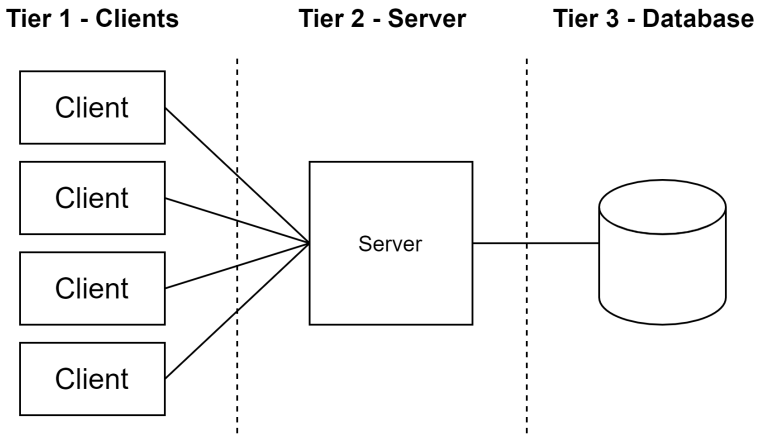

Abbildung 2.4: 3-Tier Architektur

Multi-Tier (4-Tier und mehr)

Die 4-Tier Architektur unterscheidet sich im Wesentlichen nicht sehr von der 3-Tier Architektur. Der Unterschied ist, dass bestimmte Funktionen auf andere Server ausgelagert werden. [30]

2.4.2 Schichtenarchitektur

Der Server (Abbildung 2.4) lässt sich horizontal in mehrere Schichten (Layer) unterteilen. Hierbei werden Komponenten zum Beispiel von der Geschäftslogik gruppiert und

die Kommunikation findet im Anschluss über definierte Schnittstellen statt. Hierbei sollte darauf geachtet werden, dass nur mit Schichten interagiert wird, die direkt darüber oder darunter liegen. Diese Aufteilung erlaubt es dem Entwickler ganze Schichten auszutauschen, ohne dass dies größere Änderungen bei den umliegenden Schichten bedeutet. [30]

2.4.3 Model View Controller

Das Model View Controller (MVC) Entwurfsmuster findet häufig in der Softwareentwicklung respektive Webentwicklung Verwendung. MVC trennt das Datenmodell von der Benutzeroberfläche, mit dem Ziel die Anwendung wartbarer zu machen. So können sich Designer um die Gestaltung der Oberfläche kümmern und Programmierer um die Geschäftslogik, ohne dass es zwingend notwendig ist, sich jeweils mit dem anderen Teil gut auszukennen. Diese Trennung der Zuständigkeiten (separation of concerns) ist ein Designprinzip, das in der Softwareentwicklung immer eingesetzt werden sollte. Die lose Koppelung ermöglicht es auch, die Benutzeroberfläche auszutauschen, ohne dass die Geschäftslogik betroffen ist. [6]

Model
Das Modell (Model) enthält die Daten und ist für die Geschäftslogik verantwortlich. In der Ursprungsform wird die Präsentation automatisch benachrichtigt, wenn sich die Daten im Modell ändern. Dies ist bei Webanwendungen nicht direkt möglich, weil sich die Präsentation bzw. die Benutzeroberfläche beim Client im Browser befindet und im Normalfall der Server nur auf Anfragen des Clients reagiert und nicht umgekehrt unaufgefordert benachrichtigt.

View
Die Präsentation (View) ist zuständig für die Darstellung der Daten des Modells und für die Realisierung der Benutzer Interaktionen. Bei einer Webanwendung ist die Präsentation in Client- und Serverseite aufgeteilt. Wobei der Server die HTML-Seite erzeugt und der Browser auf der Clientseite für das Anzeigen der Seite und das Verarbeiten von Benutzeraktionen verantwortlich ist.

Controller
Die Steuerung (Controller) nimmt HTTP-Anfragen entgegen und dient als Vermittler zwischen Modell und Präsentation.[19]

Der Ablauf einer HTTP-Anfrage wird in der Abbildung 2.5 beschrieben. Der Client schickt eine HTTP-Anfrage (1) an den Server. Der zu der Anfrage passende Controller interagiert mit der Datenlogik bzw. dem Modell (2). Im Anschluss delegiert der Controller das Erstellen der Benutzeroberfläche (3). Als letztes wird die erstelle HTML-Seite zurück an den Client geschickt(4).

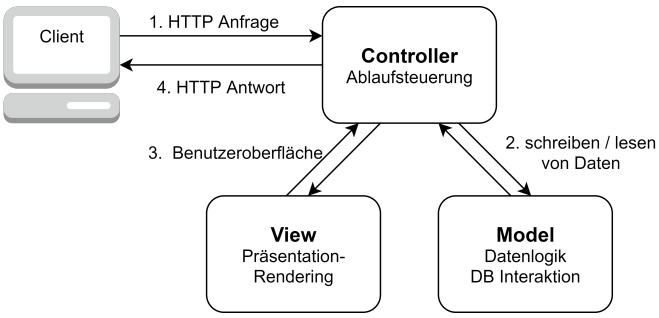

Abbildung 2.5: Model View Controller

Im Java Umfeld heißt diese Annäherung an das MVC Entwurfsmuster „Model 2“. Ein Servlet übernimmt die Rolle des Controllers. Er erzeugt und befüllt die JavaBeans, die in diesem Fall die Aufgaben des Modells übernehmen. Im Anschluss delegiert der Controller das Rendern der View an eine JSP. In ihr befindet sich der HTML-Code mit Platzhaltern für die Daten aus dem Modell.

2.5 Thread

In der Softwareentwicklung werden Threads verwendet, um mehrere Aufgaben gleichzeitig erledigen zu können. Sie werden auch leichtgewichtiger Prozess genannt und sind ein aktiver Ausführungsstrang bei der Abarbeitung eines Programms. Threads werden in einem passiven Container respektive Prozess verwaltet. Das Betriebssystem erstellt für jede Java Anwendung, die gestartet wird, einen Prozess. Jede Java-Anwendung hat mindestens einen Thread, den sogenannten „Main-Thread“. Es können aber beliebig viele weitere Threads im Programmablauf erzeugt werden. Ein Prozessor hat nur eine begrenzte Zahl an Prozessorkernen und kann somit auch nur entsprechend dieser Anzahl echt parallel Aufgaben erledigen. Den Unterschied zwischen scheinbar und echt parallel veranschaulicht die Abbildung 2.6. Das System in der oberen Hälfte hat nur einen Prozessorkern. Damit die Aufgaben (Tasks) eins und zwei scheinbar parallel ausgeführt werden, wechselt

der Scheduler, also der Programmteil der Anwendung oder des Betriebssystems, der für die Verteilung der Rechenzeit zwischen den Threads verantwortlich ist, zwischen den beiden Aufgaben hin und her. In der unteren Hälfte ist ein System, das zwei Prozessorkerne hat und somit können beide Aufgaben echt parallel abgearbeitet werden.

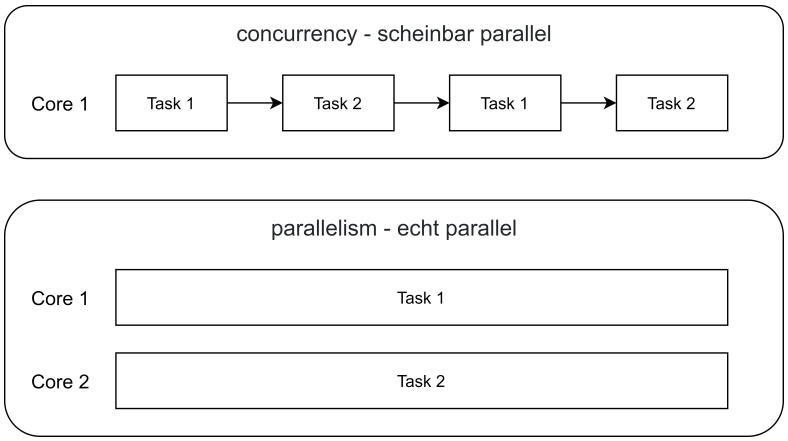

Abbildung 2.6: Concurrency vs Parallelism

Insbesondere bei rechenintensiven Aufgaben sollte darauf geachtet werden, dass nicht zu viele Threads erstellt werden, die dann alle um die Ressource Prozessorkern konkurrieren. Der Scheduler müsste sonst häufig unnötigerweise den Kontext wechseln. Bei weniger rechenintensiven Aufgaben ist der häufige Kontextwechsel kein Problem und führt zu mehr scheinbarer echter Parallelität, obwohl die Rechenzeit tatsächlich zwischen allen Threads aufgeteilt wird.

Ist eine Klasse dafür vorgesehen von einem Thread ausgeführt zu werden, sollte diese zunächst das Runnable-Interface implementieren. Es beschränkt sich auf das Definieren der Run-Methode, die den auszuführenden Code enthält, ohne Rückgabewert und Parameter. Alternativ kann das Callable-Interface implementiert werden. Es definiert die Call-Methode, hat auch keine Parameter, liefert jedoch ein Generic als Ergebnis zurück. Die Feature-Klasse nimmt das Ergebnis bei dem Aufrufen der Call-Methode entgegen und bietet Methoden zum Prüfen ob das Ergebnis schon vorliegt. Die Java-Klasse-Thread implementiert das Runnable-Interface. Eine Möglichkeit ist es, dass eine Klasse von der Thread-Klasse erbt und die Run-Methode überschreibt. Eine andere Möglichkeit ist, dass eine Klasse das Runnable-Interface implementiert, die Run-Methode überschreibt und dieses Objekt dem Konstruktor des Thread-Objekts als Parameter mitgibt. Wobei die Run-Methode niemals direkt aufgerufen werden darf. Dies würde dazu führen, dass

kein neuer Thread gestartet wird, sondern der aufrufende Thread der Methode den Code selber ausführt. Ein Thread wird durch die Start-Methode gestartet. Runnable wird von der Thread-Klasse und vom ExecutorService (siehe 2.6.1) unterstützt. Wohingegen ein Callable nur an ein ExecutorService übergeben werden kann.

Wenn mehrere Threads an derselben Stelle im Speicher arbeiten, kann es passieren, dass das Ergebnis fehlerhaft ist und je nach Reihenfolge der Zugriffe variiert. Dieses Problem wird auch „Race Condition“ genannt. Es lässt sich vermeiden, indem der Zugriff auf die gemeinsame Ressource Speicher in einen möglichst kleinen Code-Block zusammengefasst wird, der so genannten „Critical Section“. Durch das Anwenden von geeigneten Synchronisationsmechanismen wird sichergestellt, dass nur ein Thread zur gleichen Zeit den kritischen Abschnitt betritt. Ist eine Methode oder ein Abschnitt vor gleichzeitigem Zugriff geschützt, wird diese als „Thread-Safe“ bezeichnet.

2.6 Thread-Pools

Muss die Anwendung eine sehr hohe Anzahl gleichförmiger Aufgaben parallel bewältigen, würde ohne Regulierung für jede Aufgabe ein Thread gestartet. Das kostet nicht nur Zeit und Speicherplatz, vielmehr leiden alle Threads durch den ständigen Kontextwechsel. Abhilfe schafft hier ein Thread-Pool. Dieser ermöglicht es, nur eine vorher definierte Anzahl an Threads bereitzustellen. Trifft eine neue Aufgabe ein, wird ein Thread im Pool aktiv und arbeitet diese ab. Sind alle Threads beschäftigt, werden die Aufgaben, die gerade nicht von einem Thread bearbeitet werden können, in einer Warteschlange abgelegt.

2.6.1 ExecutorService

In der Java Version 1.5 wurde durch das Paket „java.util.concurrent“ die Grundlage für Thread-Pools geschaffen. Es bietet die Möglichkeit von der manuellen Thread-Erstellung zu abstrahieren und ist in der Lage asynchron Aufgaben auszuführen und wiederverwendbare Threads in einem Pool zu verwalten. Das Executor-Interface entkoppelt die Aufgabenerstellung von der Ausführung. Der Funktionsumfang ist mit einer Methode zum Ausführen sehr gering. Das ExecutorService-Interface erweitert das Executor Interface um einige Methoden, wie zum Beispiel das Ausführen von Callables oder das Beenden des Thread-Pools.

2.6.2 ThreadPoolExecutor

Der ThreadPoolExecutor implementiert das ExecutorService-Interface und bietet Konfigurationsmöglichkeiten für einen Thread Pool. Die Anzahl der immer verfügbaren Threads (corePoolSize) sowie die maximale Anzahl der Threads (maxPoolSize) lassen sich einstellen. Standardmäßig werden bei dem Erstellen des Thread-Pools nur Threads gestartet, wenn neue Aufgaben eintreffen. Dies betrifft auch die Core-Pool-Threads. Ist die aktuelle Anzahl der Threads im Pool niedriger als die „corePoolSize“ und eine neue Aufgabe trifft ein, dann wird ein neuer Thread gestartet selbst wenn ein anderer Thread im Pool gerade keine Aufgabe hat. Wenn die Anzahl der Threads im Pool höher ist als die „corePoolSize“ aber niedriger als die „maxPoolSize“, dann werden nur neue Threads gestartet wenn die Warteschlange voll ist. Reduziert sich das Aufkommen der Aufgaben, werden nach einer bestimmten Zeit (keepAliveTime) die neuen Threads nach und nach gelöscht, bis die „corePoolSize“ wieder erreicht ist. Dies ist in Abbildung 2.7 veranschaulicht. [7]

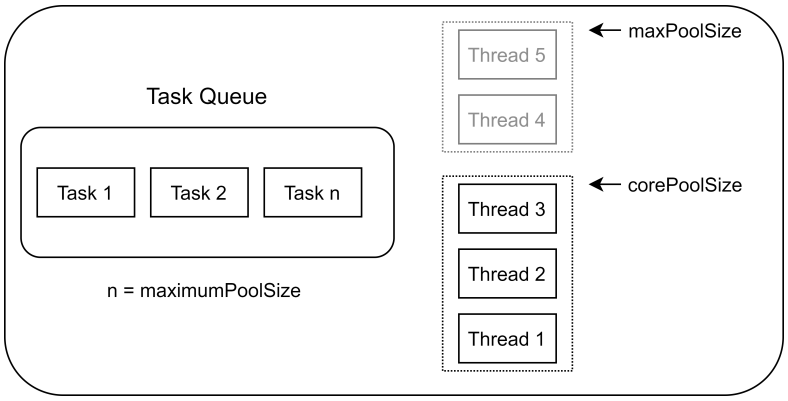

Abbildung 2.7: ThreadPoolExecutor

Die BlockingQueue in Java ist Thread-Safe und wird als Warteschlange verwendet. Die Abbildung 2.8 zeigt, wie Aufgaben (Tasks) vom Thread-Pool bearbeitet werden. Es gibt drei unterschiedliche Arten, wie sich eine Warteschlange verhält. Die SynchronousQueue ist eine Implementierung der BlockingQueue und ist streng genommen keine Warteschlange. Trifft eine neue Aufgabe ein, muss ein Thread bereit sein, diese anzunehmen. Anderenfalls wird die Aufgabe abgelehnt. Um dies zu verhindern, sollte der Thread-Pool eine hohe „maxPoolSize“ haben, damit für die Aufgabe ein neuer Thread erstellt werden kann. Eine weitere Art ist die unbegrenzte Warteschlange wie zum Beispiel die LinkedBlockingQueue, deren Größe ist Integer.MAX_VALUE wenn kein Parameter angegeben

ist. In diesem Fall hat die „maxPoolSize“ keine Bedeutung, da die neuen Aufgaben bei voller Auslastung der Threads immer in die Warteschlange gelegt werden. Die dritte Möglichkeit ist eine begrenzte Warteschlange wie zum Beispiel die ArrayBlockingQueue.

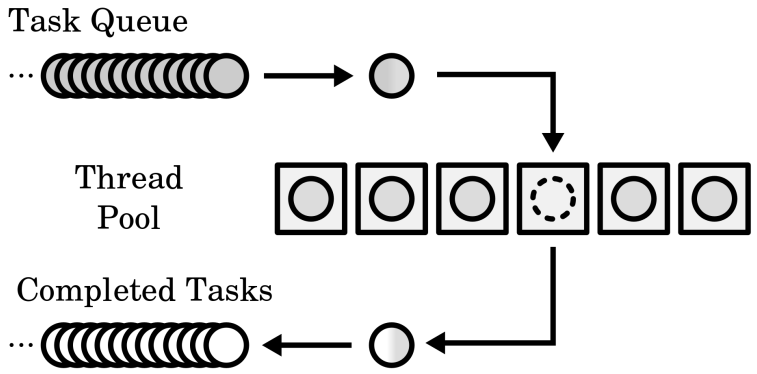

Abbildung 2.8: BlockingQueue [10]

Optional kann dem Konstruktor des ThreadPoolExecutor eine ThreadFactory übergeben werden, um auf Parameter wie Thread-Priorität und -Namen Einfluss zunehmen. Die RejectPolicy greift, wenn der Thread-Pool keine Aufgaben annehmen kann, weil die Warteschlange voll ist oder der Pool heruntergefahren wurde. Standardmäßig wird in dem Fall eine RejectedExecutionException geworfen. Die Aufgabe kann an den aufrufenden Thread zurückgegeben oder verworfen werden. Es besteht auch die Möglichkeit, dass sie in die Warteschlange aufgenommen wird und dafür die Aufgabe, die schon am längsten wartet, verworfen wird.

2.6.3 Executors

Die Hilfsklasse Executors bietet Fabrikmethoden, um ThreadPoolExecutor bzw. ScheduledExecutorService erstellen zu können. Dies sind Thread-Pools mit bestimmten Eigenschaften, die in der folgenden Auflistung genauer beschrieben werden.

- **CachedThreadPool** erstellt so viele Threads wie benötigt, aber verwendet auch schon erstellte wieder. Da eine SynchronousQueue benutzt wird, kann immer nur ein Task angenommen werden, wenn ein Thread frei ist. Andernfalls wird ein neuer Thread erstellt.

- **FixedThreadPool** erstellt eine feste Anzahl an Threads und legt Aufgaben, die nicht angenommen werden können, in einer unbegrenzten Warteschlange ab.
- **ScheduledThreadPool** erstellt einen Thread-Pool, der zu einer bestimmten Zeit Aufgaben ausführen kann oder diese in einem definierten Intervall wiederholt.
- **SingleThreadExecutor** erstellt genau einen Thread, der alle Aufgaben ausführt. Neue Aufgaben werden in einer unbegrenzten Warteschlange abgelegt.
- **WorkStealingPool** erstellt einen WorkStealingPool (siehe 2.6.3 - ForkJoinPool), der als Ziel des Parallelisierung-Levels die Anzahl der Prozessorkerne hat.

ForkJoinPool

Lässt sich eine Aufgabe nach dem „Divide and Conquer“ Prinzip sinnvoll in gleichgeartete Teilaufgaben unterteilen, kann die Verwendung eines ForkJoinPools von Vorteil sein. Der ThreadPoolExecutor hat nur eine Warteschlange für Aufgaben, dessen Synchronisation Zeit kostet. Der ForkJoinPool versucht dies zu verhindern, indem jeder Arbeiter-Thread zusätzlich seine eigene Warteschlange hat, die das Einfügen und Herausnehmen an beiden Enden erlaubt. Dies wird in Java auch „Deque - double ended queue“ genannt. Erstellte Teilaufgaben legt der Thread in seiner eigenen Warteschlange ab. Er arbeitet immer die neuesten Aufgaben zuerst ab (LIFO - Last In First Out). Hat ein Thread in der eigenen Warteschlange keine Aufgaben mehr, nimmt er sich aus der Warteschlange eines anderen Threads die Aufgabe, die am längsten wartet (FIFO - First In First Out). [5]

3 Anforderungsanalyse

In diesem Kapitel wird beschrieben, welche Funktionen und Anforderungen die Webanwendung erfüllen soll.

3.1 Problemstellung

Das Fallbeispiel und die dazugehörige exemplarische Problemstellung wird im Folgenden beschrieben. An der Hochschule für angewandte Wissenschaften Hamburg müssen die Studierenden in verschiedenen Veranstaltungen praktische Programmieraufgaben, üblicherweise in einem Team mit zwei Mitgliedern, lösen. Damit der Lehrende einen möglichst transparenten Einblick in die Entwicklung hat, werden die jeweiligen Aufgaben im Internet mit entsprechenden Zugriffsrechten gespeichert. Hierfür wird in der Regel das Versionsverwaltungswerkzeug GIT benutzt. Der Speicherort für den Quellcode wird GIT-Repository genannt. Die Webanwendung soll es den Studierenden ermöglichen, sich in Gruppen zu organisieren und die entsprechenden Links zu den Repositories dem Lehrenden zur Verfügung zu stellen. Zusätzlich bietet die Anwendung, unabhängig von der ersten Problemstellung dem Studierenden die Möglichkeit, dem Lehrenden Dokumente durch das Hochladen von Dateien mit bestimmter Dateiendung zur Verfügung zu stellen.

3.2 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben bestimmte Anwendungsszenarien und Funktionalitäten, die das System erfüllen muss. Im Folgenden werden zwei wichtige genauer beschrieben. Alle anderen Anwendungsszenarien befinden sich im Anhang B.

Tabelle 3.1: Team registrieren

Abschnitt	Inhalt
Bezeichner	UC-001
Name	Team Registrieren
Ergebnis	Der Benutzer hat ein Team registriert.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist gestartet auf einem Server. Der Benutzer ist angemeldet.
Nachbedingung	Der Benutzer hat ein Team registriert und kann nun Aufgaben melden.
Hauptszenario	1. Der Benutzer wählt ein Semester und eine Veranstaltung. 2. Der Benutzer gibt das Beitrittspasswort der jeweiligen Veranstaltung an. 3. Der Benutzer bestimmt ein Passwort, das ein Teammitglied benötigt, um beizutreten. 4. Das Formular wird an die Webanwendung geschickt. 5. Die Webanwendung verifiziert alle Daten und speichert das Team in der Datenbank.
Ausnahmeszenario	5a. Eine oder mehrere Daten sind nicht korrekt. Dem Benutzer wird eine entsprechende Meldung angezeigt.

Tabelle 3.2: GIT-Repository melden

Abschnitt	Inhalt
Bezeichner	UC-002
Name	GIT-Repository melden
Ergebnis	Der Benutzer hat eine GIT-Repository gemeldet.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist gestartet auf einem Server. Der Benutzer ist angemeldet und hat ein Team für eine Veranstaltung registriert.
Hauptszenario	1. Der Benutzer navigiert zu der Übersicht seiner Teams. 2. Der Benutzer wählt bei einem Team „Add Repository“. 3. Der Benutzer wählt eine Aufgabe und trägt das Repository in das Eingabefeld ein. 4. Der Benutzer schickt das Formular durch das Klicken auf den Button „Repository Hinzufügen“ an die Webanwendung. 5. Die Webanwendung überprüft, ob der Link zu dem Repository syntaktisch korrekt aufgebaut ist. 6. Der Link zu dem Repository wird in der Datenbank gespeichert.
Ausnahmeszenario	5a - 6a. Die Syntax des Links ist nicht korrekt. Der Link wird nicht gespeichert und der Benutzer entsprechend benachrichtigt.

Der Lehrende hat die Möglichkeit, Fächer und Semester zu speichern und durch eine Kombination der beiden Elemente eine Veranstaltung zu erstellen, bei der sich die Studierenden registrieren können.

3.3 Nicht Funktionale Anforderungen

Die nicht Funktionalen Anforderungen beschreiben keine konkrete Funktion, die von der Anwendung erbracht werden muss. Viel mehr handelt es sich um Leistungs- oder Qualitätsmerkmale der Software. Sie müssen messbar, vergleichbar und reproduzierbar sein.

3.3.1 Performance

Die Webanwendung soll bei moderater Auslastung in der Lage sein, Anfragen in weniger als 200 Millisekunden zu beantworten.

3.3.2 Zuverlässigkeit

Die Zeit, bis es zu einer Fehlfunktion kommt, soll höher sein als 240 Stunden, damit die Nichtverfügbarkeit des Dienstes so gering wie möglich gehalten werden kann.

3.3.3 Bedienbarkeit

Die Webanwendung soll eine Benutzeroberfläche bieten, die intuitiv bedient werden kann, sodass bei jedem Vorgang ersichtlich ist, wo sich die Funktionen befinden und welche Daten für eine erfolgreiche Durchführung benötigt werden. Ein durchschnittlicher Benutzer sollte nach einer Einarbeitungszeit von fünf Minuten alle Funktionen benutzen können.

4 Konzeption und Implementierung

In diesem Kapitel werden die Design- und Architekturentscheidungen der Webanwendung aufgezeigt.

4.1 Dynamische Webanwendung vs Single Page Application

Dynamische Webseiten werden auf dem Server anhand der angeforderten Daten aus der Anfrage generiert. Dies geschieht meistens mit Hilfe einer „Template-Engine“. Sie fügt die Daten aus dem Modell in eine Vorlage (Template) der angeforderten Seite ein. In diesem Fall wird bei jeder Anfrage des Benutzers die ganze HTML-Seite erneut übertragen. Bei einer Single Page Application (SPA) wird die Seite in Form von JavaScript und das Layout nur einmal an den Client geschickt. Wechselt der User die Ansicht, werden die dafür benötigten Daten von einem Webdienst üblicherweise im JSON Format angefordert. Die SPA kann dann die reinen Daten verarbeiten und das DOM selbständig anpassen. Im Rahmen dieser Arbeit wurde eine dynamische Webseite erstellt, da diese hauptsächlich in Java geschrieben werden kann und eine SPA umfassende Kenntnisse in Javascript voraussetzt.[26]

4.2 Monolithisch vs Microservice

Eine sehr grundlegende Architekturentscheidung bei der Entwicklung einer Webanwendung ist, ob eine einzige Anwendung die Funktion abbilden soll. Dabei ist kein unstrukturiertes Programm gemeint. Es kann sich zum Beispiel auch um eine Schichtenarchitektur handeln. In diesem Zusammenhang spricht man auch von einem Monolith. Eine Microservice-Architektur unterteilt die Webanwendung pro Funktion in eigenständige

Services. Hierbei kann jeder Service unabhängig bestehen und die Komponenten kommunizieren über eine REST-API miteinander.

4.2.1 Microservice

Die Grundidee ist es, die Anwendung in eine Menge lose gekoppelter Dienste zu unterteilen, wobei jeder Dienst eine entsprechende Funktion der Anwendung abbildet. Jeder Microservice hat seine eigene Geschäftslogik und eine eigene Datenbank. Jeder Service speichert nur die Daten, die für seine Aufgabe notwendig sind. Dies kann insgesamt zu einer doppelten Speicherung von Daten führen, wenn es Überschneidungen bei den Funktionen gibt. Weitere Benutzeroberflächen, wie zum Beispiel Mobile-Apps, kommunizieren meistens nicht direkt mit den einzelnen Microservices im Backend sondern benutzen dafür ein API-Gateway als Vermittler. Die Abbildung 4.1 zeigt eine Beispielanwendung, die in die drei Microservices Account, Inventory und Shipping unterteilt ist, wobei jeder Service seine eigene Datenbank hat. Der Browser kommuniziert mit dem Teil der Anwendung, der die Benutzeroberfläche ausliefert. Das API-Gateway stellt Daten über eine REST-API für die Mobile-App bereit.[2]

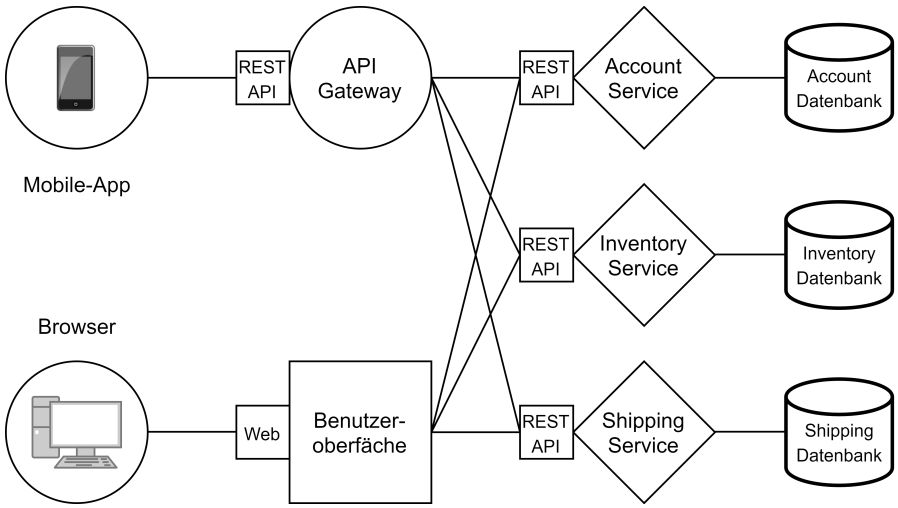

Abbildung 4.1: Microservice Architecture [2]

Vorteile von Microservices

- Zerlegt große Anwendungen in überschaubare eigenständige Anwendungen, die für sich genommen einfacher zu verstehen und zu warten sind.
- Jeder Microservice kann in einem großen Unternehmen von einem separaten Team entwickelt werden.
- Nach Bedarf kann jeder Service für sich skaliert werden.
- Da jeder Microservice seine eigene Datenbank benutzt, kann jeder Service die für ihn am besten passende Datenbank aussuchen.

Nachteile von Microservices

- Das Bereitstellen einer Microservice Architektur ist aufwändiger.
- Änderungen, die mehrere Services betreffen sind im Vergleich zu einer monolithischen Anwendung schwieriger umzusetzen und erfordern eine gewisse Planung in der Umsetzung.
- Eine Mircroservicearchitektur ist auch immer ein verteiltes System und somit muss eine Form der Kommunikation etabliert werden.
- Die geteilte Datenbankstruktur hat den Nachteil, dass Microservice übergreifende Transaktionen schwer umsetzbar sind und dann meistens der Ansatz von „eventual consistency“ gewählt werden muss.

4.2.2 Monolithisch

Unter einer monolithischen Softwarearchitektur versteht man eine Anwendung, die alle Funktionen in einer Anwendung vereint. Der Vorteil ist, dass sie einfacher zu entwickeln, testen und zu deployen ist. Sie lässt sich zwar nicht partiell skalieren, dafür ist ein horizontales Skalieren von vielen Instanzen hinter einem Lastenverteiler möglich. Die Performance kann durch den gemeinsamen Speicher besser sein, da es keine Notwendigkeit für Interprozesskommunikation gibt. [3]

4.2.3 Architekturentscheidung

Für die Webanwendung, die im Rahmen dieser Bachelorthesis entwickelt wurde, ist ein monolithischer Ansatz gewählt worden. Da die Anwendung ein kleines Funktionsspektrum hat, wäre eine weitere Unterteilung in kleinere Dienste nicht sinnvoll. Es würde zu unnötigem Aufwand durch die Etablierung von Interprozesskommunikation und geteilten Datenbanken kommen. Für einen kleinen Prototyp ist der monolithische Ansatz besser, insbesondere weil im weiteren Verlauf die Performance der Webanwendung bezüglich des Thread-Pools getestet wird und dies lässt sich bei einem System besser testen als bei vielen verteilten.

4.3 Sondierung möglicher Frameworks

Neben der Implementierung in reinem Java wurde eine weitere Variante zum Vergleich mithilfe eines Frameworks entwickelt. Im folgenden werden Frameworks verglichen, die potenziell für die Webanwendung geeignet sind.

4.3.1 Jakarta Server Faces

Java EE, welches von der Eclipse Foundation übernommen wurde und seitdem Jakarta EE heißt, ist eine Sammlung von Spezifikationen, die es erleichtert Java Enterprice Anwendung respektive Webanwendungen zu entwickeln. Jakarta Server Faces (JSF) ist Teil dieser Spezifikation und ein Model 2 MVC Framework. Es basiert auf Servlets, die in einem Webcontainer ausgeführt werden. Die Abbildung 4.2 zeigt, dass Anfragen vom Client von dem zentralen Faces Servlet angenommen werden. Es erzeugt bzw. befüllt die JavaBeans mit Daten und delegiert entsprechend der Anfrage an das passende Facelet. Dies liest Daten aus den JavaBeans und erstellt die Benutzeroberfläche auf Basis von Extensible Hypertext Markup Language (XHTML), einer Auszeichnungssprache ähnlich der HTML nur mit einer strikteren Syntax. Als letzter Schritt wird die HTML-Seite an den Client geschickt. Zu jeder JSF Seite gehört eine „Backing Bean“, die von einem Container verwaltet wird. Sie enthält die Methoden, die bei der Interaktion mit der Benutzeroberfläche ausgeführt werden. Die Webanwendung benötigt sowohl bei der Entwicklung als auch im späteren Betrieb einen Anwendungsserver. [8]

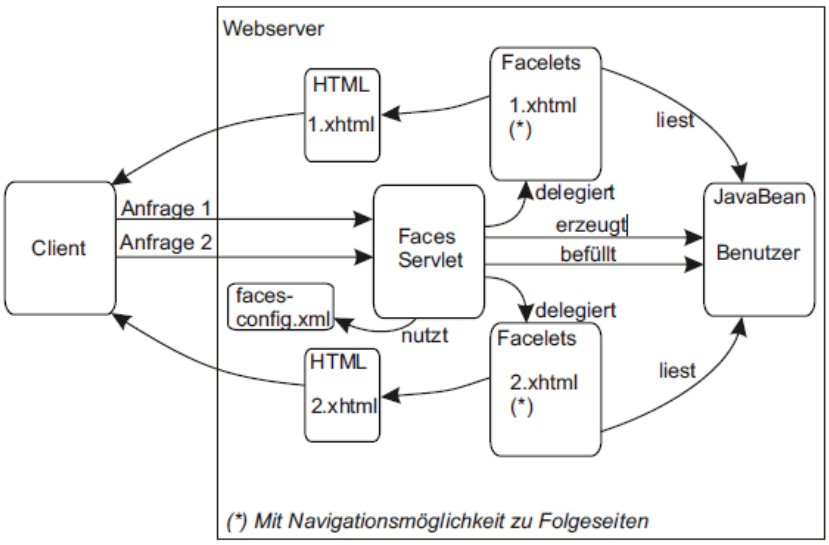

Abbildung 4.2: Jakarta Server Faces [8]

4.3.2 Apache Struts 2

Ein weiteres auf Java basierendes Model 2 MVC Framework ist Apache Struts 2. Eine intuitive Konfiguration ist durch Annotationen möglich. Ähnlich wie bei JSF gibt es ein zentrales Servlet, das eingehende Anfragen verarbeitet. Aktion-Klassen erweitern den Controller und bilden eine Brücke zwischen der Clientseite und der Geschäftslogik. Sie bestehen aus Java POJOs und erleichtern somit das Testen. Die Aktionklasse kann, bevor die Geschäftslogik ausgeführt wird, auch andere Aufgaben übernehmen wie zum Beispiel Authentifizierung, Logging oder Session Validierung. Aktion-Klassen werden nur einmal instanziiert, genauso wie Servlets. Dies muss in einer Anwendung mit mehreren Threads entsprechend berücksichtigt werden. Templates für die Präsentation können mit Java Server Pages, Apache's Freemarker oder anderen Template-Engines erzeugt werden. [1]

4.3.3 Java Spring Boot

Grundlagen

Spring und insbesondere Spring Boot ist ein Framework, dass sich sehr vielfältig einsetzen lässt. Häufig werden mit ihm Webanwendungen nach dem Model-View-Controller-Pattern umgesetzt. Spring Boot bietet hierfür eine sehr elegante Möglichkeit den Webserver in die Anwendung einzubetten und somit eine selbstständige Anwendung zu erschaffen. Spring verfolgt das Prinzip „Convention over configuration“ was Spring wie folgt definiert.

> For a lot of projects, sticking to established conventions and having reasonable defaults is just what they (the projects) need... this theme of convention-over-configuration now has explicit support in Spring Web MVC. What this means is that if you establish a set of naming conventions and suchlike, you can substantially cut down on the amount of configuration that is required to set up handler mappings, view resolvers, ModelAndView instances, etc. This is a great boon with regards to rapid prototyping, and can also lend a degree of (always good-to-have) consistency across a codebase should you choose to move forward with it into production.
>
> Spring Dokumentation [24]

Eine weitere wichtige Grundlage für Spring ist „Dependency Injection“. Das Ziel ist es auf einer höheren Ebene eine Verbindung zwischen Objekten zu schaffen und gleichzeitig eine lose Kopplung zu ermöglichen. Spring verwendet dafür den IoC (Inversion of Control) Container.

Ein weiterer Vorteil ist der modulare Aufbau. Dieser wird in Abbildung (4.3) veranschaulicht. Im Folgenden werden die wichtigsten Module näher erläutert. [22]

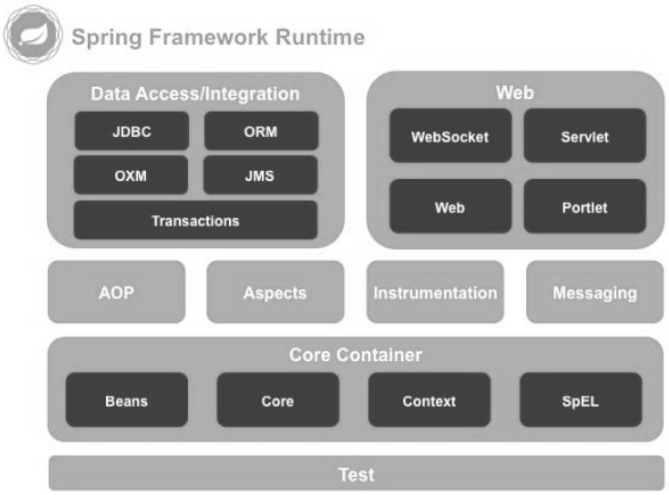

Abbildung 4.3: Spring Modules [22]

- **Core** Der Core Container enthält die fundamentalen Komponenten des Frameworks. Hierbei handelt es sich im Wesentlichen um Spring eigene Beans die über Kontexte verwaltet und über Annotationen verwendet werden können. Hierzu zählt unter anderem das nach dem Fabrikmuster umgesetzte Erstellen der Beans.
- **Data Access/Integration** Das Daten Modul vereinfacht den Zugriff auf Datenbanken, vom einfachen Datenbank Treiber bis hin zu Object Relational Mapping, also der Möglichkeit Daten zwischen sonst inkompatiblen Systemen auszutauschen.
- **Web** Schnittstellen für Webanwendungen. Dabei kann es sich um Servlets oder WebSockets handeln. Es bietet eine Implementierung des Model-View-Controller Patterns.
- **AOP** Aspect Oriented Programming (AOP) ermöglicht es wiederkehrende Aufgaben, wie zum Beispiel Logging und Authentifizierung, separat zu definieren und dann automatisch an beliebig vielen Stellen im Programmablauf einzubauen.

Spring MVC

Spring MVC ist ein Request getriebenes Framework. Wie in Abbildung 4.4 zu sehen ist, werden eingehende Anfragen von dem Front Controller, auch Request Dispatcher genannt, angenommen und von ihm an den für diesen bestimmten Request verantwortlichen

Controller delegiert. Dieser erstellt ein Modell mit den benötigten Daten und liefert diese mit dem Namen der View zurück an den Front-Controller. Im letzten Schritt wird mittels einer Template-Engine aus dem Modell und dem Template eine fertige HTML-Seite erzeugt, die an den Benutzer ausgeliefert wird. [23]

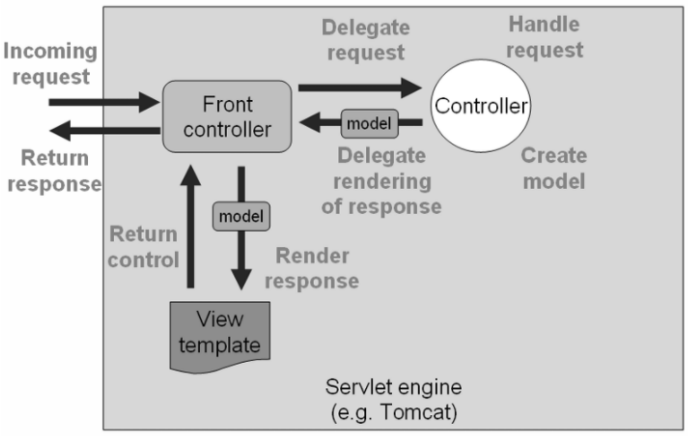

Abbildung 4.4: Spring MVC [23]

4.3.4 Entscheidung

Der Verfasser hat sich bei der Entwicklung der Webanwendung dieser Bachelorthesis für Spring entschieden. Struts 2 schied aus, weil es kaum Verwendung findet und somit der Community Support dementsprechend gering ist. Jakarta EE bietet theoretisch ähnliche Funktionen wie Spring, ist aber für die Erstellung eines Prototyps nicht gut geeignet, da es eine Einarbeitung in Anwendungsserver erfordert, wohingegen Spring mit einem integrierten Webserver arbeiten kann. Die Anwendung kann also auf einem beliebigen Server ohne weitere Installation und Konfiguration eines Anwendungsservers laufen. Spring bietet auch einfachere Möglichkeiten des Testens. Die Integration neuer Features in das Framework ist bei Spring schneller, weil es nicht wie Jakarta EE einen langwierigen Spezifizierungsprozess durchlaufen muss.

4.4 Implementierung

Der folgende Abschnitt beschreibt den Aufbau und die Unterschiede der beiden Implementierungen.

4.4.1 Umsetzung

Die Programme wurden in der Programmiersprache Java mit Hilfe der Eclipse Integrated Development Environment (IDE) entwickelt. Der Quellcode wurde mit dem freien Versionskontrollsystem GIT verwaltet. Zum einen, weil so jederzeit ein alter Speicherstand oder einzelne Abschnitte im Quellcode wiederhergestellt werden können und zum anderen, weil das Speichern des Quellcodes in einem entfernten Repository indirekt auch als Backup der Dateien dient. Beide Implementierungen benutzen das Build-Management-Tool Maven der Apache Software Foundation. Es basiert auf einer Extensible Markup Language (XML) Konfigurationsdatei und vereinfacht das Hinzufügen und Verwalten von externen Bibliotheken. Der Build-Prozess kann in der Konfigurationsdatei genau definiert werden. Unter anderem wird hier festgelegt ob die Anwendung als Web Application Archive (WAR) verpackt wird und somit für den Einsatz auf einem Anwendungsserver wie zum Beispiel Tomcat geeignet ist oder ob die Anwendung in einem Java Archive (JAR) mit integriertem Tomcat verpackt werden soll. Die JAR Datei ließe sich so auf einem beliebigen Server mit installierter JVM ausführen.

4.4.2 Clean Code

Damit der Code übersichtlich und wartbar bleibt, sollten bei der Entwicklung die folgenden Grundsätze beachtet werden.

DRY - Don't Repeat Yourself

Das Designprinzip DRY bedeutet, dass sich Teile des Quellcodes, die identisch oder sehr ähnlich sind, nicht im Quellcode wiederholen sollten. Vielmehr sollte eine Lösung für ein Problem nur an einer Stelle im Quellcode definiert werden. Daraus folgt eine wesentlich bessere Wartbarkeit des Quellcodes, da bei einer Überarbeitung immer an genau einer Stelle für ein gegebenes Problem Änderungen notwendig werden.[12]

KISS - Keep It Simple, Stupid

Dieser Grundsatz sagt aus, dass eine Lösung für ein Problem immer so einfach und klar sein sollte wie möglich. Jede Methode sollte nur eine bestimmte Aufgabe lösen und nicht

mehrere. Wenn die Methode zu lang und unübersichtlich ist, sollte diese nach Möglichkeit weiter zerteilt werden.

YAGNI - You Aren't Gonna Need It

Es sollten nur Funktionen dem Programm hinzugefügt werden, die tatsächlich gebraucht werden und nicht solche die nur möglicherweise in der Zukunft benötigt werden. Dies reduziert die Entwicklungszeit und macht den Code übersichtlicher.

4.4.3 Schichtenarchitektur

Die Schichtenarchitektur ist hauptsächlich aus der Spring Perspektive beschrieben. Die Java Implementierung verfolgt den gleichen Ansatz.

Data Access Layer

Der Data Access Layer wurde mit Jakarta Persistence API (JPA) erstellt. Als JPA-Implementierung wurde Hibernate verwendet. Da dies der Standard JPA-Provider von Spring-Boot ist, wird automatisch die benötigte EntityManagerFactoryBean erzeugt. Die Factory ist so konfiguriert, dass entsprechend annotierte Entitäten automatisch als solche erkannt werden. Die Repository-Klassen sind mit @Repository Annotiert, dadurch werden sie von Spring als Komponente erkannt und zusätzlich werden JPA spezifische Exceptions umgewandelt in Spring's DataAccessException. Die Repository-Klassen erweitern die JpaRepository-Klasse und bieten somit unter anderem CRUD (Create, Read, Update, Delete) Operationen, ohne dass viel ähnlicher Quellcode für diese Operationen für jedes Repository erstellt werden muss. Die Java Implementierung verwendet auch Hibernate und hat dies mit Data Access Objects (DAO) umgesetzt. Sie entkoppeln die Service-Klassen von der Datenbank.[20]

Business Layer

Der Business Layer enthält die gleichnamig annotierten Service-Klassen und implementiert die Geschäftslogik (siehe Abbildung 4.5). Dadurch kann der Data Access Layer sehr überschaubar sein und führt somit zu einer Trennung der Zuständigkeiten. Diese lose

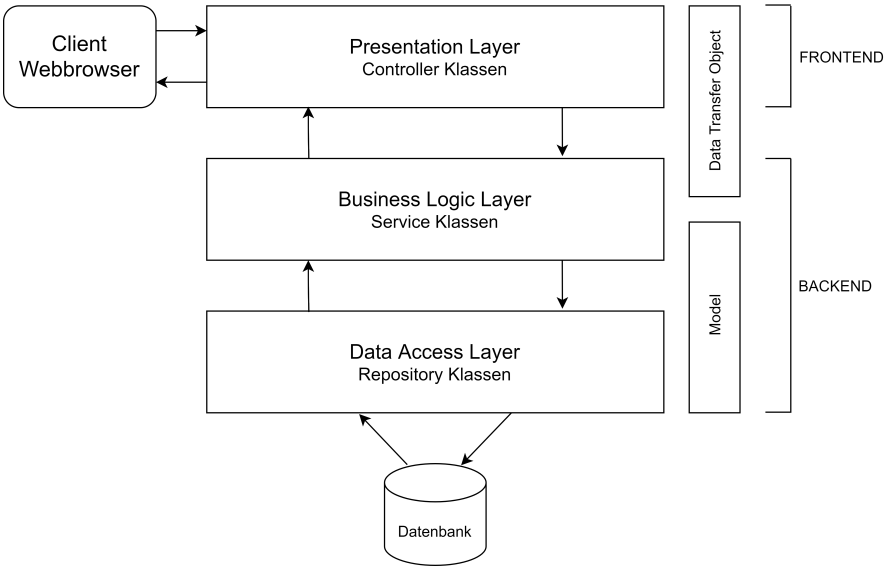

Abbildung 4.5: Layering

Koppelung ermöglicht ein einfaches Austauschen der Datenbank. Aber auch aus Gründen der Sicherheit ist dieser Layer wichtig, da somit die Präsentationsschicht nicht direkt auf die Datenbank zugreifen kann.

Presentation Layer

Die Klassen, die als Controller annotiert sind, werden von Spring als Komponente erkannt, die Anfragen aus dem Web verarbeiten. Durch das definieren eines URL-Patterns auf Klassen- oder Methodenebene wird die passende Methode zu einer Webanfrage ausgeführt. Der Controller ruft Methoden der Service-Klassen auf, um Businesslogik auszuführen. Im Falle einer Leseoperation legt der Controller die Daten in dem Modell ab. Beim Rendern der Seite, die zurück an den Client geschickt wird, werden die Daten aus dem Modell von der Template-Engine an die vorher definierten Platzhalter im Template eingefügt. [17]

4.4.4 Struktur der Implementierungen

Die Java-Implementierung besteht aus 3004 Zeilen Code und die Spring Version aus 2943 (siehe Tabelle 4.1). Beide haben die Packages „controller“, „service“, „model“ und „dao“ bzw. „repository“, deren Funktion im vorherigen Abschnitt Schichtenarchitektur 4.4.3

beschrieben wurde. Das, was bei Spring implizit ist bzw. der Anwendungsserver übernimmt, wie zum Beispiel das Bereitstellen eines Thread-Pools und das Annehmen von Anfragen, wird in der Java Implementierung im Package „application“ umgesetzt. Spring hat zusätzlich Data Transfer Objects (DTO) um die Daten aus einer Anfrage auf das entsprechende Modell mappen zu können. Die Klassen zum Passwort Hashing, also dem verschlüsselten Speichern der Passwörter und Passwort Validierung sowie Konfigurationsdateien befinden sich bei der Spring Implementierung in den Packages validation und config.

Tabelle 4.1: Paketstruktur der Java und Spring Implementierungen

Package	Java Lines of Code	Spring Lines of Code
application	863	-
controller	505	615
service	629	678
model	662	1025
dao | repository	345	112
dto	-	198
valivation and config	-	315
total	3004	2943

4.4.5 Frontend

Die folgende Abbildung 4.6 zeigt exemplarisch die Benutzeroberfläche am Beispiel der Funktion „Team beitreten“. Wird eine Veranstaltung gewählt, werden nur die Teams angezeigt, die zu dieser Veranstaltung gehören und noch nicht voll sind. Die Java Implementierung überträgt dazu alle Veranstaltungsdaten in den lokalen Speicher des Browsers und filtert die entsprechenden Teams mit Javascript. Die Spring Implementierung setzt dies eleganter mittels einer Fetch-API um. Wählt der Benutzer eine Veranstaltung, wird serverseitig eine Liste der gültigen Teams erstellt und im JSON-Format an den Browser übermittelt.

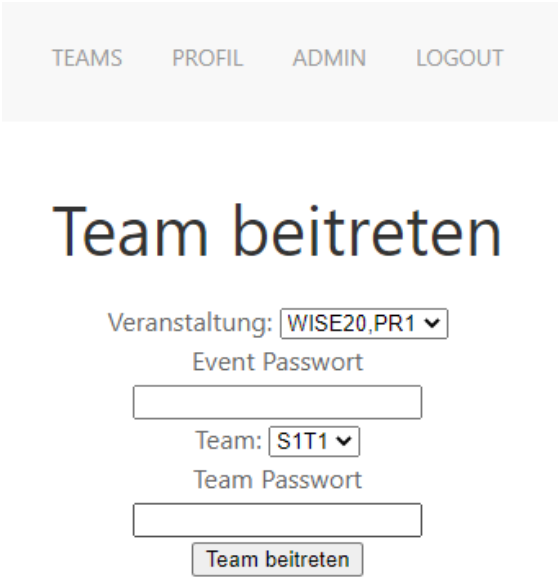

Abbildung 4.6: Benutzeroberfläche

Template-Engine

Zum serverseitigen Erstellen der HTML-Seiten wurde im Falle der Spring Implementierung die Template-Engine Thymeleaf benutzt. Sie fügt sich sehr gut in die HTML-Dateien ein. Eine Template-Datei kann, bevor sie gerendert wird, von einem Browser geöffnet werden und wird trotzdem korrekt angezeigt. Der HTML-Seite können Tags oder Attribute hinzugefügt werden, die es erlauben Daten einzufügen. Es können auch Bedingungen definiert und Schleifen verwendet werden. Die Java Implementierung verwendet als Template-Engine Apache FreeMarker. [27]

UIkit

Für die Gestaltung der Benutzeroberfläche wurde das freie Frontend Framework UIkit verwendet. Es basiert auf CSS bzw. Javascript und bietet für alle gängigen HTML-Tags Klassen um die Optik der Webanwendung zu verbessern und ein einheitliches Design zu erstellen. [28]

4.4.6 Datenbank

Für die Persistierung der Daten wurde das weit verbreitete relationale Datenbankverwaltungssystem MySQL verwendet. Java enthält die Datenbankschnittstelle Java Database Connectivity (JDBC) um mit der Datenbank auf Basis von MySQL Befehlen zu interagieren. Um dies zu vereinfachen wurde das Persistenz- und Object-Relational Mapping (ORM)-Framework Hibernate, das die Java Persistence API implementiert, verwendet. ORM ist eine Technik bei der Java-Objekte auf die Strukturen von relationalen Datenbanken abgebildet werden. Eine Abbildung des Datenbankmodells, das für die Implementierungen verwendet wurde, befindet sich im Anhang C.1.

4.5 Thread-Pools

Es folgt die Berechnung der Thread-Pool-Größe für das Testsystem und die Thread-Pool Implementierung bzw. Konfiguration der beiden erstellten Anwendungen.

4.5.1 Thread-Pool-Größe

Laut Brian Goetz, dem Autor von „Java Concurrency in Practice“, ist die Ermittlung der Größe des Thread-Pools keine exakte Wissenschaft. Jedoch gilt, dass ein zu großer Thread-Pool dazu führt, dass Threads um die zentrale Recheneinheit (engl. CPU) und Speicher Ressourcen ringen und ein zu kleiner Thread-Pool den Durchsatz negativ beeinflussen kann. Der offensichtlichste Faktor ist die Anzahl der zur Verfügung stehenden Prozessorkerne. Da bei der Entwicklung nicht immer klar ist, wie viele Prozessorkerne der Server hat, auf dem die Anwendung nach der Entwicklung laufen wird bzw. es wahrscheinlich ist, dass im Laufe der Zeit die Anwendung auf einem anderen Server umzieht, sollte die Thread-Pool-Größe nicht fest definiert werden, sondern sich möglichst dynamisch an das System anpassen. In Java lässt sich die Anzahl der Kerne mit dem folgenden Aufruf ermitteln.

```
int cores = Runtime.getRuntime().availableProcessors();
```

Bei rechenintensiven Aufgaben kann angenommen werden, dass die Anzahl der Threads gleich der Anzahl der Prozessorkerne im System plus eins ist. Der eine Thread mehr ist

sinnvoll, weil es trotz theoretischer Vollauslastung des Systems passieren kann, dass die Anwendung auf einen Speicherbereich zugreift, der sich gerade nicht im Hauptspeicher befindet oder andere I/O-Operationen die Ausführung zwischenzeitlich verhindern. Bei der Bestimmung einer geeigneten Thread-Pool-Größe sollte auch immer das Gesamtsystem betrachtet werden. So macht es zum Beispiel keinen Sinn sehr viele Threads für das Abarbeiten der HTTP-Anfragen bereitzuhalten, wenn fast jede Anfrage auf die Datenbank zugreift und der Datenbank-Verbindungspool viel kleiner ist. In dem Fall würden die vielen Threads, die Anfragen annehmen, ständig auf die Threads der Datenbank warten. [7]

Parameter

- N_{cpu}= Anzahl der Prozessorkerne
- $N_{threads}$= Anzahl der Threads
- U_{cpu} = Geplante Auslastung der CPU, $0 \leq U_{cpu} \leq 1$
- $\frac{W}{T}$ = Verhältnis zwischen Warte- und Ausführungszeit

Daraus ergibt sich die Formel zur Berechnung der Thread-Pool-Größe:

$$N_{threads} = N_{cpu} * U_{cpu} * (1 + \frac{W}{T})$$

Das Testsystem hat zwei Prozessorkerne (N_{cpu}) und teilt sich das System mit der Datenbank. Es wird angenommen, dass die Anwendung bei zwei von drei Anfragen auf die Datenbank zugreift. Deswegen wurde ein Faktor von 0,65 für die geplante Auslastung (U_{cpu}) gewählt. Aufgrund der Messung des Verfassers braucht die Webanwendung für eine Anfrage ohne Datenbankzugriff durchschnittlich 20 ms und mit Datenbankzugriff 110 ms. Die Wartezeit (W) ergibt sich aus der Anfrage mit Datenbankzugriff (110 ms) minus der Anfrage ohne Datenbankzugriff (20 ms). Daraus ergibt sich eine Wartezeit von 90 ms. Die Ausführungszeit (T) entspricht der Anfrage ohne Datenbankzugriff und beträgt 20 ms.

$$N_{threads} = 2 * 0,65 * (1 + \frac{90}{20}) = 7,15$$

4.5.2 Java-Implementierung

Neben der Entwicklung einer Anwendung wurde auch untersucht, wie die Webanwendung mehrere Anfragen gleichzeitig bearbeiten kann. Dafür existiert ein Thread, der auf einem Socket horcht und eingehende Anfragen in einer Warteschlange platziert. Ein Thread aus dem Pool, der gerade keine Aufgabe hat, übernimmt die neue Anfrage. Sollten alle Threads beschäftigt sein, bleibt die Anfrage solange in der Warteschlange, bis ein Thread die Aufgabe übernehmen kann. Der folgende Codeausschnitt 4.1 zeigt den in der Java Implementierung verwendeten Thread-Pool. In Zeile 2 und 3 wird der ThreadPoolExecutor instanziiert. Die „corePoolSize“ beträgt in diesem Fall 8, die „maxPoolSize“ 32, die „keepAliveTime“ 8 und die LinkedBlockingQueue umfasst 10000 Plätze. In Zeile 14 werden die Anfragen an den Thread-Pool übergeben.

Listing 4.1: Thread-Pool-Code

```
1  public ConnectionHandler() {
2      this.executor = new ThreadPoolExecutor(8, 32, 8,
3      TimeUnit.SECONDS,  new LinkedBlockingQueue<Runnable>(10000)));
4      server = getServerSocket();
5  }
6
7  @Override
8  public void run() {
9      while (ConfigurationStms.running) {
10         System.out.println("waiting for new connection...");
11         Webserver w;
12         try {
13             w = new Webserver(server.accept());
14             executor.execute(w);
15             System.out.println("connected!");
16         } catch (IOException e) {
17             e.printStackTrace();
18         }
19     }
20 }
```

Ein ThreadPoolExecutor arbeitet mit einer begrenzten LinkedBlockingQueue, die dafür ausgelegt ist mit parallelen Zugriffen umzugehen und verhindert so, dass zwei Threads im gleichen Moment den Task annehmen können. Bei einer unbegrenzten Warteschlange könnte der Webanwendung bei hoher Last der Speicherplatz für die Warteschlange ausge-

hen. Dies lässt sich bei dem ThreadPoolExecutor durch das Begrenzen der Warteschlange verhindern.

4.5.3 Spring-Implementierung

Die Spring Implementierung benutzt den freien Anwendungsserver Tomcat der Apache Software Foundation. Eine Spring Anwendung kann auf einem Separation Tomcat-Server gestartet werden oder einen integrierten Tomcat-Server verwenden. Dieser hat den Vorteil, dass für die Entwicklung und auch für den späteren Betrieb nicht unbedingt ein Tomcat-Server installiert und konfiguriert werden muss. Der eingebettete Tomcat-Server verwendet als TaskQueue eine LinkedBlockingQueue. Im folgenden sind die wichtigsten Konfigurationsmöglichkeiten aufgelistet.

- **server.tomcat.threads.max**
  Maximale Anzahl an Threads, die Anfragen bearbeiten.
- **server.tomcat.threads.min-spare**
  Minimale Anzahl an Threads, die Anfragen bearbeiten.
- **server.tomcat.accept-count**
  Maximale Länge der Warteschlange für den Fall, dass alle Threads beschäftigt sind.
- **server.tomcat.max-connections**
  Maximale Anzahl an Verbindungen, die ein Server zur gleichen Zeit verarbeiten kann.

[21]

5 Test

Softwaretests im Allgemeinen dienen dazu, dass die Anwendung, bevor sie ausgeliefert wird, auf definierte Anforderungen bezüglich Qualität, Geschwindigkeit, Skalierbarkeit, Stabilität und Bedienbarkeit getestet wird. Werden in einem Softwaretest keine Fehler gefunden, heißt dies nicht, dass generell keine Fehler mehr vorhanden sind. Es heißt lediglich, dass bestimmte Testfälle keine Fehler mehr enthalten. Eine komplette Testabdeckung durch das Testen aller möglichen Eingabeparameter ist häufig nur bei kleineren Anwendungen möglich.

Im Folgenden wird die zwei durchgeführten Lasttests beschrieben. Diese Tests simulieren reale Lastsituationen. Im Fall der Webanwendung ist das der parallele Zugriff mehrerer Benutzer auf die Seite zur gleichen Zeit. Dadurch kann ermittelt werden, wie sich die Antwortzeiten bei hoher Last verhalten und ab wann ein System möglicherweise nicht mehr reagiert. In Bezug auf Webanwendungen ist es wichtig, dass die Seite innerhalb weniger Sekunden reagiert.

5.1 Aufbau

Für den Lasttest wurde das freie und in Java geschriebene Tool JMeter der Apache Software Foundation verwendet. Es wird häufig für Leistungstests eingesetzt und kann diverse Anwendungsszenarien abdecken, wie zum Beispiel das Testen von Webanwendungen, Webservices, Dateiserver, Datenbanken und Emailserver. In der Benutzeroberfläche von JMeter kann ein Testplan, der den Ablauf des Testes definiert, manuell erstellt werden. Es besteht aber auch die Möglichkeit über einen integrierten Proxy den Ablauf eines Seitenbesuches aufzuzeichnen und den Testplan generieren zu lassen. Der eigentliche Test wird dann anhand des Testplans in der Kommandozeile ausgeführt.

Insbesondere für das Simulieren des Anmeldevorgangs ist es wichtig, dass sich der Test möglichst genau so verhält, wie es ein Webbrowser machen würde. Dafür bietet JMeter

eine Reihe von Modulen, die sich um das Verwalten von Header Informationen, Cookies und Authentifizierung kümmern. Ein wesentliches Element des Testplans ist die „Thread-Group“. Sie definiert die Anzahl der Benutzer bzw. Threads, die den Test parallel durchführen, die Anzahl der Wiederholungen pro Thread und eine Zeit-Periode in der sukzessive Threads gestartet werden bis das Maximum erreicht ist. Innerhalb der „Thread-Group“ werden verschiedene HTTP GET und POST Sample definiert, die den Besuch der Internetseite simulieren. Der Testaufbau besteht aus einem System mit zwei Prozessorkernen, auf dem die Webanwendung läuft und einem System mit vier Prozessorkernen auf dem JMeter läuft.

5.2 Abruf der Anmeldeseite

Der erste Test beschränkt sich auf das wiederholte Abrufen der Anmeldeseite, damit die Threads des Systems, auf dem die Anwendung läuft, nicht mit den Threads der Datenbank für benutzerspezifische Daten konkurrieren müssen. Für den Test wurden 500 Benutzer bzw. Threads gestartet, die jeweils den Aufruf der Anmeldeseite 200-mal wiederholten. Es wurden also insgesamt 100.000 Samples (siehe Tabelle 5.1) pro Implementierung erstellt.

Tabelle 5.1: Anmeldeseite

Titel	Samples	Durchschn. Antwortzeit in ms	Transaktionen pro Sekunde	Fehler in %
Spring	100.000	4063.54	116.06	0
Java SE	100.000	7477.94	62.21	20.39

Der durchgeführte Test zeigt, dass die durchschnittliche Antwortzeit bei der Spring Implementierung nur halb so lang wie bei der JAVA SE Implementierung ist. Dies führt dazu, dass die 100.000 Samples der Spring Implementierung in 13 Minuten abgearbeitet sind und die gleiche Anzahl an Samplen mit der Java SE Implementierung 27 Minuten brauchen. Des Weiteren werden bei der Spring Implementierung alle Anfragen beantwortet und bei der JAVE SE Variante kommt es zu einer Fehlerquote von 20 Prozent. Die Abbildung 5.1 veranschaulicht, dass die Spring Implementierung mit durchschnittlich 116 Transaktionen pro Sekunde einen höheren Durchsatz hat als die Java Implementierung. Diese kommt durchschnittlich nur auf 62 Transaktionen pro Sekunde.

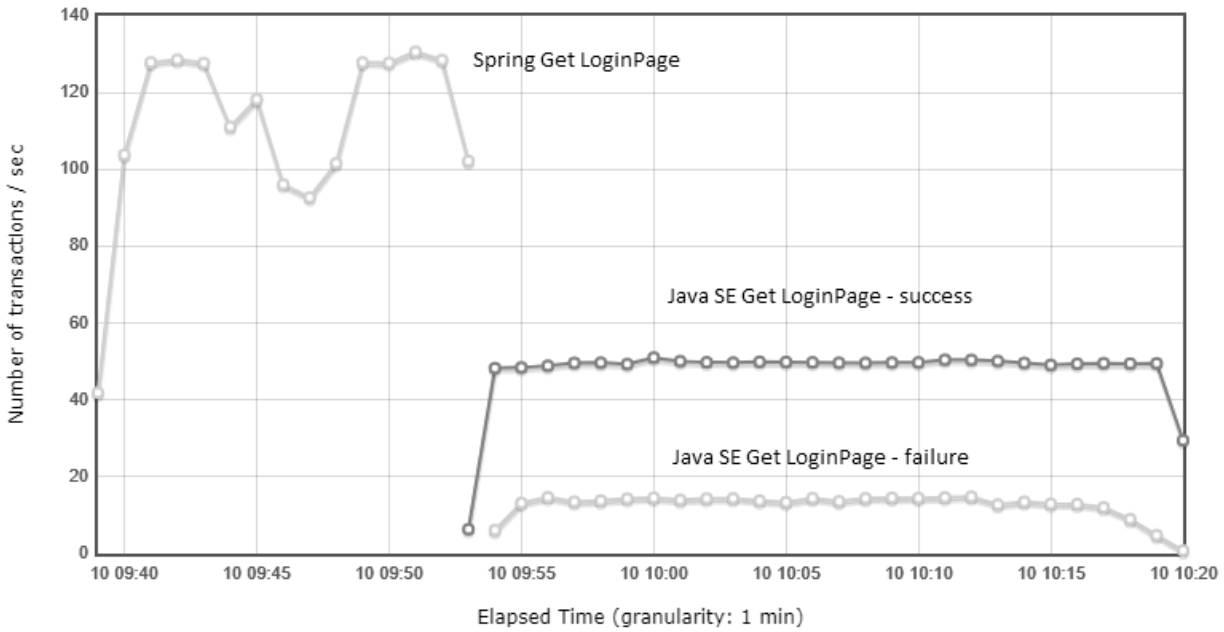

Abbildung 5.1: JMeter: Test ohne Datenbankzugriff

5.3 Login Simulation

Der zweite Test simuliert einen Anmeldevorgang. Zunächst wurde wieder die Anmeldeseite abgerufen, dann wurden die Zugangsdaten an die Webanwendung geschickt und im Anschluss zwei unterschiedliche interne Seiten aufgerufen. Der Test wurde erst einmal auch mit 100.000 Sampeln (500 Threads * 50 Wiederholungen * 4 Anzahl der Anfragen pro Wiederholung) durchgeführt, dies führte dazu, dass die Anwendung nicht mehr reagierte. Deswegen wurde die Anzahl schrittweise reduziert, bis der Test wieder durchführbar war. Der durchgeführte Test lief mit 20.000 Samplen pro Implementierung (100 Threads * 50 Wiederholungen * 4 Anzahl der Anfragen pro Wiederholung). Der Test lief mit der gleichen Anzahl an Samplen nicht stabil. Grund dafür ist, dass für den Test zusätzlich Datenbankabfragen notwendig waren. Folglich kam das System auf dem die Anwendung lief mit der Last nicht mehr zurecht. Bei dem zweiten Test wurde nicht, wie im vorherigen die Abbildung der Transaktionen pro Sekunde gewählt, sondern die durchschnittliche Antwortzeit. An der Abbildung 5.2 ist gut erkennbar, dass bei der Java-Implementierung alle Anfragen ungefähr die gleiche Antwortzeit von durchschnittlich 4,3 Sekunden haben. Bei der Spring Implementierung haben GET Login, Team, Profile eine niedrigere durchschnittliche Antwortzeit von ca. 2,7 Sekunden. Jedoch dauert das Ein-

loggen (Spring Login) mit 11,7 Sekunden deutlich länger. Dies führt insgesamt dazu, dass die durchschnittliche Antwortzeit der Spring Implementierung mit 4,9 Sekunden etwas höher ist als die der anderen Variante mit 4,3 Sekunden.

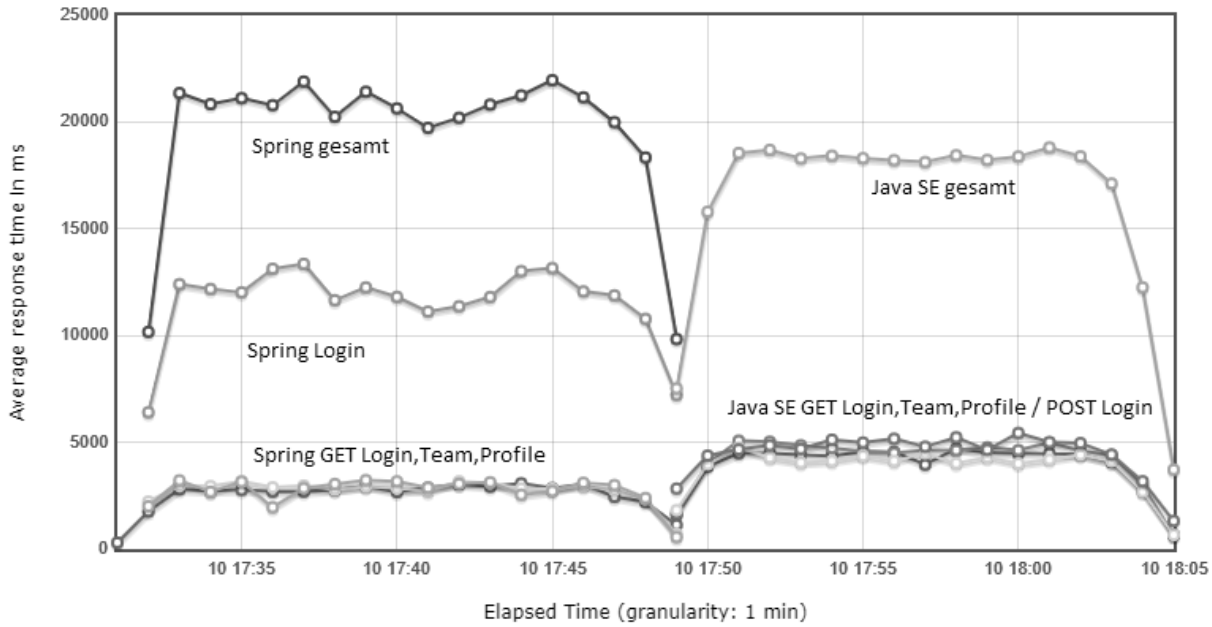

Abbildung 5.2: JMeter: Logintest mit Datenbankzugriff

6 Evaluation

6.1 JAVA SE

Die in reinem Java geschriebene Implementierung der Webanwendung erfüllt zwar bis auf das Hochladen von Dateien (UC-007) alle funktionalen Anforderungen. Die vielen Implementierungsdetails, die beachtet werden mussten, die bei der herkömmlichen Webentwicklung impliziert vorhanden sind und der damit verbundene Aufwand stand letztendlich nicht im Verhältnis zu dem erzielten Umfang der Webanwendung. Bei dem Entwickeln einer Webanwendung wird in der Regel ein Technologie-Stack verwendet, der dem Entwickler alle grundlegenden Funktionen zur Verfügung stellt. Diese solide Grundlage spart nicht nur Zeit, sondern ist durch die potenzielle weite Verbreitung sehr gut getestet und enthält somit wesentlich weniger Fehler und infolgedessen auch weniger Sicherheitslücken. Externe Bibliotheken wurden nur für die Datenbankverbindung bzw. das Mappen der Objekte auf die Datenbanktabellen, für das Rendern der HTML-Dateien und für das Passwort-Hashing verwendet. Alles andere, wie zum Beispiel das Socket-Handling, das Erstellen und Verarbeiten der HTTP-Anfragen, das Verwalten der angemeldeten Benutzer und vieles mehr, wurde vom Verfasser der Arbeit selber implementiert. Dies bringt jedoch viel Potenzial für Fehler und Sicherheitslücken. Die gemessene Fehlerquote von 20 Prozent bei dem Abruf der Anmeldeseite ist definitiv zu hoch. Im Rahmen dieser Bachelorarbeit blieb keine Zeit für die Ursachenforschung. Für die genauere Betrachtung der Thread-Pools im Zusammenhang mit einer Webanwendung war dieser Ansatz akzeptabel. Der selber implementierte Thread-Pool der Java SE Implementierung konnte weniger Anfragen beantworten als die Spring Implementierung, obwohl beide die gleiche Thread-Pool-Konfiguration und eine LinkedBlockingQueue verwendet haben. Der Grund dafür ist vermutlich weniger der Thread-Pool, sondern viel mehr die Java SE Implementierung. In der relativ kurzen Zeit ist es nicht möglich eine Webanwendung zu einwickeln, die mit Spring auf dem selben Niveau ist. Spring wurde bereits 2002 veröffentlicht und wird seitdem kontinuierlich verbessert. Die Entwicklung einer Webanwendung in reinem

Java ohne Framework ist im Rahmen einer Bachelorarbeit nicht unbedingt sinnvoll. Auch bei der Entwicklung für den produktiven Betrieb sollte nach Möglichkeit immer ein entsprechendes Framework benutzt werden.

6.2 Spring

Die Annahme, dass es mit Spring für den Entwickler möglich ist, sich ohne viel Konfiguration mit der Entwicklung der Geschäftslogik zu beschäftigen, hat sich bestätigt. Der Grundsatz von Spring „Convention over Configuration“ traf zu. Ursprünglich wurde angenommen, dass sich die Klassen des Data Access Layers und des Service Layers mit wenig Aufwand in die Spring Implementierung integrieren. Im Falle der Entity-Klassen war dies bis auf leichte Anpassungen der Benutzerklasse für Spring Security möglich. Die Anpassung des Service Layers war hingehen etwas aufwendiger, da die Art und Weise, wie Spring Daten validiert und speichert, unterschiedlich war. Die Annahme, dass sich mit Spring Boot in kurzer Zeit eine solide, übersichtliche und sichere Anwendung erstellen lässt, hat sich bestätigt. Die funktionalen Anforderungen werden von der Anwendung erfüllt. Bei den nicht funktionalen Anforderungen ist die Anwendung bezüglich der Performance bei den meisten HTTP-Anfragen schneller, lediglich bei dem Anmeldevorgang (UC-004) war die Java SE Implementierung im Vorteil.

6.3 Retrospektive

Der erste Ansatz war, eine REST-API mit entsprechendem Frontend-Framework, wie zum Beispiel Angular, zu entwickeln. Dies wurde verworfen, weil es viel JavaScript voraussetzt und die implementierte Alternative fast ohne JavaScript auskommt. Rückblickend betrachtet wäre dieser Ansatz möglicherweise die bessere Wahl gewesen, da sich in diesem Fall das Backend auf das Ausliefern von JSON-Daten beschränkt hätte und somit kompakter und dadurch auch vergleichbarer gewesen wäre.

7 Fazit

Es hat sich gezeigt, dass das Entwickeln einer dynamischen Webanwendung in Java ohne zusätzliches Framework weniger sinnvoll ist. Es müssen sehr viele Details beachtet und implementiert werden. Dieser hohe Aufwand steht nicht im Verhältnis zu dem erzielten Ergebnis. Insbesondere in dem Zeitraum einer Bachelorarbeit ist es nicht möglich, eine Webanwendung zu erstellen, die in den Punkten Performance und Sicherheit überzeugen kann. Die beiden Implementierungen ließen sich in ihrer Gesamtheit gut testen. Allerdings war es schwer zu separieren, inwiefern die Leistung der Thread-Pools hier einen Einfluss hatte. Dafür waren die beiden Implementierungen am Ende zu unterschiedlich. Bei der Entwicklung einer Webanwendung sollte im Idealfall ein Framework wie zum Beispiel Spring verwendet werden. Es ist nur wenig Konfiguration notwendig und bietet solide Grundlage um Webanwendungen zu entwickeln.

Literaturverzeichnis

[1] Cavaness, Chuck: *Programming - Jakarta Struts // Programming Jakarta Struts*. 2nd ed. Beijing : O'Reilly, op. 2004. – ISBN 0596006519

[2] Chris Richardson: *Pattern: Microservice Architecture*. – URL https://microservices.io/patterns/microservices.html. – Zugriffsdatum: 2021-09-13

[3] Chris Richardson: *Pattern: Monolithic Architecture*. – URL https://microservices.io/patterns/monolithic.html. – Zugriffsdatum: 2021-09-13

[4] Cody Lindley: Front-end Developer Handbook 2019. . – URL https://frontendmasters.com/books/front-end-handbook/2019/. – Zugriffsdatum: 2021-09-13

[5] Doug Lea ; Gannon, Dennis (Hrsg.) ; Mehrotra, Piyush (Hrsg.): *A Java Fork/-Join Framework*

[6] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns - Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp, 2015. – ISBN 978-3-8266-9700-5

[7] Goetz, Brian: *Java concurrency in practice*. Upper Saddle River, NJ and Boston and Indianapolis : Addison-Wesley, 2006. – ISBN 0321349601

[8] Goll, Michael: *JavaServer Faces*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020. – ISBN 978-3-658-31802-4

[9] Golubski, Wolfgang: *Entwicklung verteilter Anwendungen*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019. – ISBN 978-3-658-26813-8

[10] HowToDoInJava: *Java Thread Pool – ThreadPoolExecutor*. – URL https://howtodoinjava.com/java/multi-threading/java-thread-pool-executor-example/. – Zugriffsdatum: 2021-09-13

[11] Kommer, Isolde: *HTML5, CSS3 und JavaScript Webseiten entwickeln*. 2. Ausgabe. Bodenheim : HERDT, September 2015 (HERDT Classics). – ISBN 9783862494415

[12] Martin, Robert C.: *Clean code: A handbook of agile software craftsmanship*. Upper Saddle River NJ : Prentice Hall, 2009. – ISBN 9780132350884

[13] Mozilla: *Document Object Model*. – URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model. – Zugriffsdatum: 2021-09-13

[14] Mozilla: *HTML5*. – URL https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. – Zugriffsdatum: 2021-09-13

[15] Mozilla: *HTTP*. – URL https://developer.mozilla.org/en-US/docs/Web/HTTP. – Zugriffsdatum: 2021-09-13

[16] Müller-Hofmann, Frank ; Hiller, Martin ; Wanner, Gerhard: *Programmierung von verteilten Systemen und Webanwendungen mit Java EE*. Wiesbaden : Springer Fachmedien Wiesbaden, 2015. – ISBN 978-3-658-10511-2

[17] Patel, Nilang: *Spring 5.0 Projects*. [Place of publication not identified] : Packt Publishing, 2019. – ISBN 978-1-78839-041-5

[18] Pomaska, Günter: *Webseiten-Programmierung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2012. – ISBN 978-3-8348-2484-4

[19] Praful Todkar: *MVC - desktop application vs web application*. – URL https://computellect.com/2011/12/20/mvc-desktop-application-vs-web-application/. – Zugriffsdatum: 2021-09-13

[20] Simons, Michael: *Spring Boot 2: Moderne Softwareentwicklung mit Spring 5*. 1. Auflage. Heidelberg : dpunkt.verlag, 2018. – ISBN 3864905257

[21] Spring: *Common Application Properties*. – URL https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html. – Zugriffsdatum: 2021-09-13

[22] Spring: *Introduction to the Spring Framework.* – URL https://docs.spring.io/spring-framework/docs/4.3.17.RELEASE/spring-framework-reference/html/overview.html. – Zugriffsdatum: 2021-09-13

[23] Spring: *Web MVC framework.* – URL https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html. – Zugriffsdatum: 2021-09-13

[24] Spring: *Web MVC framework.* – URL https://docs.spring.io/spring-framework/docs/3.0.0.M3/reference/html/ch16s10.html. – Zugriffsdatum: 2021-09-13

[25] Statistisches Bundesamt: *Internetnutzung.* – URL https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Einkommen-Konsum-Lebensbedingungen/IT-Nutzung/Tabellen/zeitvergleich-computernutzung-ikt.html. – Zugriffsdatum: 2021-09-13

[26] Steyer, Manfred ; Softic, Vildan: *AngularJS: Moderne Webanwendungen und Single Page Applications mit JavaScript.* Köln : O'Reilly, 2015. – ISBN 3955619508

[27] Thymeleaf: *Documentation.* – URL https://www.thymeleaf.org/documentation.html. – Zugriffsdatum: 2021-09-13

[28] UIKit: *Introduction.* – URL https://getuikit.com/docs/introduction. – Zugriffsdatum: 2021-09-13

[29] van Steen, Maarten ; Tanenbaum, Andrew S.: *Distributed Systems.* Third edition (Version 3.01 (2017)). London : Pearson Education, February 2017. – ISBN 9781543057386

[30] Vogel, Oliver: *Software-Architektur: Grundlagen - Konzepte - Praxis.* 2. Aufl. Heidelberg : Spektrum Akad. Verl., 2009. – ISBN 9783827419330

Abkürzungen

API Application Programming Interface.

CGI Common Gateway Interface.

CSS Cascading Style Sheets.

DAO Data Access Objects.

DOM Document Object Model.

DTO Data Transfer Objects.

EJB Jakarta Enterprise Beans früher Enterprise JavaBeans.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

Jakarta EE Jakarta Enterprise Edition ehemals JAVA Enterprise Edition.

JAR Java Archive.

JDBC Java Database Connectivity.

JPA Jakarta Persistence API.

JSF Jakarta Server Faces.

JSON JavaScript Object Notation.

JSP Jakarta Server Pages.

JVM Java Virtual Machine.

MVC Model View Controller.

ORM Object-Relational Mapping.

POJO Plain Old Java Object.

REST Representational State Transfer.

SPA Single Page Application.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

WAR Web Application Archive.

XML Extensible Markup Language.

A CD Inhaltsverzeichnis

Auf der CD befinden sich im Verzeichnis „Implementierungen“ die beiden entwickelten Anwendungen Java SE und Spring. Die verwendeten Testpläne liegen in dem Ordner „Testplan“ und die Kopien der flüchtigen Weblinks in dem Ordner „Internetquellen“. Die Bachelorthesis(.pdf) befindet sich im Root-Verzeichnis.

Implementierungen
- Java SE
- Spring

Internetquellen
Testplan
Bachelorthesis.pdf

B Funktionale Anforderungen

Tabelle B.1: Registrierung

Abschnitt	Inhalt
Bezeichner	UC-003
Name	Registrierung
Auslösendes Ereignis	Der Benutzer möchte sich registrieren.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist auf einem Server gestartet. Der Benutzer ist mit dem Internet verbunden.
Nachbedingung	Der Benutzer hat sich im System registriert.
Ergebnis	Der Benutzer ist registriert.
Hauptszenario	1. Der Benutzer navigiert im Browser zum Registrierungsformular. 2. Der Benutzer füllt alle benötigten Felder mit gültigen Werten aus. 3. Der Benutzer schickt das Formular an die Webanwendung. 4. Die Webanwendung prüft, ob der Benutzer schon im System registriert ist und validiert alle weiteren Angaben. 5. Der neue Benutzer wird in der Datenbank gespeichert. 6. Der Benutzer wird in den geschützten Bereich weitergeleitet.
Ausnahmeszenario	4a. Der Benutzer ist schon im System registriert.

Tabelle B.2: Login

Abschnitt	Inhalt
Bezeichner	UC-004
Name	Login
Auslösendes Ereignis	Der Benutzer möchte sich anmelden.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist auf einem Server gestartet. Der Benutzer ist mit dem Internet verbunden.
Nachbedingung	Der Benutzer ist angemeldet.
Ergebnis	Der Benutzer ist angemeldet und kann die Webanwendung benutzen.
Hauptszenario	1. Der Benutzer navigiert im Browser zum Anmeldeformular. 2. Der Benutzer füllt alle benötigten Felder mit gültigen Werten aus. 3. Der Benutzer schickt das Formular an die Webanwendung. 4. Die Webanwendung prüft, ob der Benutzer im System vorhanden ist und überprüft die Zugangsdaten. 5. Der Benutzer wird in den geschützten Bereich weitergeleitet.
Ausnahmeszenario	4a. Der Benutzername oder das Passwort sind nicht korrekt.

Tabelle B.3: Logout

Abschnitt	Inhalt
Bezeichner	UC-005
Name	Logout
Auslösendes Ereignis	Der Benutzer möchte sich abmelden.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist auf einem Server gestartet. Der Benutzer ist angemeldet.
Nachbedingung	Der Benutzer ist abgemeldet.
Ergebnis	Der Benutzer ist abgemeldet und befindet sich wieder auf der Anmeldeseite.
Hauptszenario	1. Der Benutzer klickt auf die Schaltfläche zum Abmelden. 2. Browser und Webanwendung löschen die Daten zur Authentifizierung. 3. Der Benutzer wird zur Anmeldeseite weitergeleitet.

Tabelle B.4: Team beitreten

Abschnitt	Inhalt
Bezeichner	UC-006
Name	Team Beitreten
Auslösendes Ereignis	Der Benutzer möchte einem Team beitreten.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist auf einem Server gestartet. Der Benutzer ist eingeloggt.
Nachbedingung	Der Benutzer hat sich bei einem Team angemeldet.
Ergebnis	Der Benutzer hat sich bei einem Team angemeldet und kann Repositories melden.
Hauptszenario	1. Der Benutzer wählt ein Team, das zu einer Veranstaltung gehört. 2. Der Benutzer gibt das Teampasswort ein. 3. Das Formular wird an die Webanwendung geschickt. 4. Die Webanwendung verifiziert alle Daten und speichert das neue Teammitglied in der Datenbank.
Ausnahmeszenario	4a. Das Teampasswort ist nicht korrekt. Dem Benutzer wird eine entsprechende Meldung angezeigt.

Tabelle B.5: Datei hochladen

Abschnitt	Inhalt
Bezeichner	UC-007
Name	Datei hochladen
Auslösendes Ereignis	Der Benutzer möchte eine Datei hochladen.
Akteure	Benutzer
Vorbedingung	Die Webanwendung ist auf einem Server gestartet. Der Benutzer ist angemeldet.
Nachbedingung	Der Benutzer hat eine Datei hochgeladen.
Ergebnis	Eine Datei wurde hochgeladen.
Hauptszenario	1. Der Benutzer navigiert zum Dateiupload. 2. Der Benutzer wählt eine Datei von seinem Dateisystem. 3. Das Formular wird an die Webanwendung geschickt. 4. Die Webanwendung verifiziert die Datei und speichert diese im Dateisystem.
Ausnahmeszenario	4a. Die Webanwendung kann die Datei nicht speichern, weil das Format nicht stimmt.

C Datenbankmodell

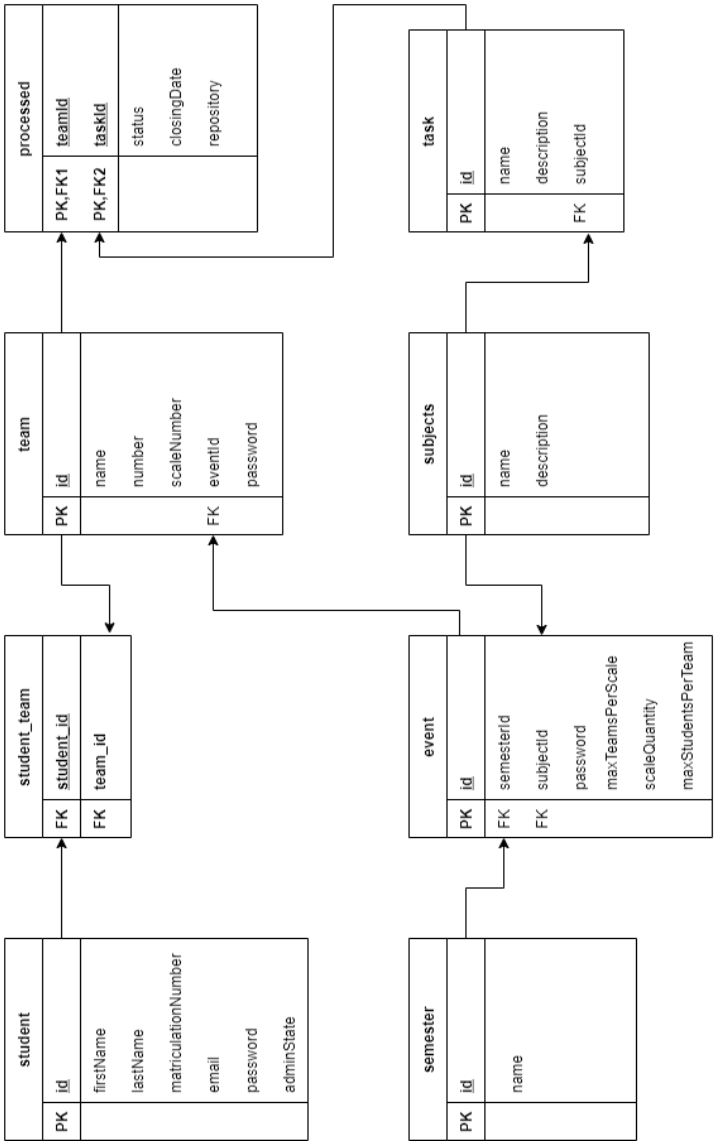

Abbildung C.1: Datenbankmodell

D Installationsanleitung

Um die Anwendung starten zu können wird eine MySQL Datenbank benötigt. Mit dem „MySQL Installer“ für Windows lässt sich der MySQL Server und die MySQL Workbench installieren. Die meisten gängigen Linux Distributionen bieten entsprechende MySQL Pakete über die Paketverwaltung an.

https://dev.mysql.com/downloads/installer/

In der MySQL Workbench, einem Tool zum Erstellen und Verwalten von MySQL Datenbanken, lässt sich das Script zu der Erstellung der Datenbank öffnen und durch die Ausführung wird die Datenbank erstellt.

Spring: CD:\sqlScripts\createDBSpring.sql
Java SE CD:\sqlScripts\createDB.sql

Um mit der Java Anwendung eine Verbindung zu der Datenbank herstellen zu können, muss die Adresse, der Benutzername und das Passwort an die MySQL Datenbank angepasst werden. In der Spring Konfiguration muss zusätzlich der Pfad für das Speichern der hochgeladenen Dateien definiert werden.

Spring: CD:\stmsSpring\src\main\resources\application.properties
JAVA SE: CD:\Dev\eclipse\stms\src\main\resources\hibernate.cfg.xml

Die Anwendung kann mit Apache Maven erstellt und in einer Jar-Datei verpackt werden. „Clean“ löscht den vorherigen Build und „Package“ impliziert das Kompilieren des Programms und verpackt es. Mit dem Parameter „-DskipTests“ kann das Programm auch ohne gültige Datenbankverbindung gebaut werden. Die ausführbare Java-Datei befindet sich nach dem Erstellen in dem Unterverzeichnis „target“.

CD:\stmsSpring\mvnw.cmd clean package -DskipTests

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original