Konzeption eines Softwaresystems zur Verwaltung von Hochschulsporteinrichtungen

Diplomarbeit von Dirk Beckmann

TECHNISCHE UNIVERSITÄT DARMSTADT

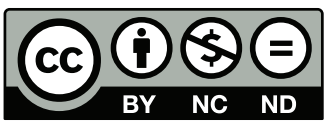
Fachbereich Humanwissenschaft
Institut für Sportwissenschaft

Konzeption eines Softwaresystems zur Verwaltung von Hochschulsporteinrichtungen

Vorgelegte Diplomarbeit von Dirk Beckmann

1. Gutachten: Prof. Dr. Joseph Wiemeyer
2. Gutachten: Dietbert Schöberl

Tag der Einreichung:



Erklärung zur Diplomarbeit

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den ...

(Dirk Beckmann)

Inhaltsverzeichnis

1 Einleitung

2 Analyse der Fachdomäne Hochschulsport

Die Aufgaben und Anforderungen, die an deutsche Hochschulen gestellt werden sind vielfältig und erstrecken sich in viele verschiedene Bereiche. Neben den Kernaufgaben universitäre Lehre und wissenschaftliche Forschung gehört es auch zu den Aufgaben das studentische Leben in sozialen und gesellschaftlichen Belangen zu unterstützen und fördern. Neben Einrichtungen wie dem Studierendenwerk oder Allgemeinen Studierendenausschuss (AStA) ist auch der Hochschulsport eine Einrichtung die unterschiedlichste Aufgaben im Umfeld der Fachhochschulen und Universitäten übernimmt. Dieser Teil soll ein Verständnis für die Domäne Hochschulsport und deren Entwicklung, Organisationsformen, Aufgaben sowie Probleme und Herausforderungen vermitteln.

2.1 Historische Entwicklung des Hochschulsports

Das Sporttreiben an deutschen Hochschulen begann zwar nicht erst nach dem zweiten Weltkrieg, die historische Betrachtung soll sich hier jedoch auf die Zeit zwischen 1948 und 2009 beschränken, um den Rahmen der Arbeit nicht zu sprengen. Da die Organisation von Bildung in die Zuständigkeit der Bundesländer fällt, hat die Entwicklung durchaus unterschiedliche Wege genommen. Ich konzentriere mich in diesem Fall auf den geschichtlichen Rückblick mit dem Schwerpunkt NRW, da hier die umfangreichsten Dokumentationen verfügbar waren. Der 1999 von von 29 Bildungsministern initiierte Bologna-Prozess zur Schaffung eines einheitlichen Hochschulraums (vgl. QUELLE)

2.1.1 Hochschulsport heute

Durch die gesetzliche Verankerung im Hochschlrahmengesetz und die Festschreibung in den einzelnen Landeshochschulgesetzen hat sich eine vorerst gesicherte Grundlage zur Förderung des Sports an Hochschulen manifestiert. Die einzelnen Hochschulsporteinrichtungen besitzen eine gefestigte Position im Hochschulkontext und müssen nicht mehr um ihre grundlegende Daseinsberechtigung kämpfen. Das Grundanliegen des HSP ist und bleibt jedoch die Förderung des Sporttreibens der Studierenden, die sich in speziellen Lebensphasen befinden und durch besondere Umstände gekennzeichnet sind. Das diese nicht generalisiert werden können, lässt sich leicht nachvollziehen, da unterschiedliche Studiengänge, Wohnumstände und Studiumsformen die Lebensphase in sehr heterogener Weise beeinflussen können. Der Einfluss auf den Hochschulsport und im speziellen auf die Akzeptanz und Teilnahme lässt sich an folgenden Kriterien festmachen, die speziell bei der Hochschulsportorganisation explizit berücksichtigt werden müssen:

1. Sportinteresse des Studierenden
2. Ort der Sporteinrichtungen

3. Soziale Kontakte

4. Belastungen und Entbehrungen

Somit ergibt sich für den Hochschulsport heute die besondere Herausforderung den Bedürfnissen der Studierenden gerecht zu werden und ihnen die Gelegenheit zu geben, gemäß ihrer Interessen und Vorstellungen Sport treiben zu können.
Der bereits beschriebene Bologna-Prozess führte allerdings zu zusätzlichen Herausforderungen im Verhältnis Hochschulsport-Studierende. Die Verkürzung der Studienzeit führte zu einem strenger vorgegebenen Studienplan und einer deutlichen Verknappung der eigenen Gestaltungs- und Freizeitmöglichkeiten. Diese Faktoren beeinflussen die Möglichkeiten der Studierenden an der Teilnahme im Hochschulsport erheblich, sodass sich des Hochschulsportprogramm diesen speziellen Anforderungen noch stärker durch mehr Flexibilität und neue Angebotsformen stellen muss.

Neben den Studierenden gehören aber zunehmend auch andere Personenkreise zur Zielgruppe des Hochschulsports. Besonders die Bediensteten der Hochschule werden zunehmend in den Programmen mit speziellen Programmen angesprochen. Speziell im Bereich Gesundheitssport haben sich hier eigene Programme an den Hochschulen oder auch durch übergeordnete Gremien, wie den allgemeinen deutschen Hochschulsportverband, etabliert. Das Angebot ist dabei sehr unterschiedliche und stark abhängig von den verfügbaren Ressourcen, erstreckt sich aber von der Teilnahme am allgemeinen Programm über spezielle Bediensteten Kurse bis hin zur individuellen Betreuung am Arbeitsplatz im Büro. Die Nachfrage nach Gesundheitsfördernden Maßnahmen dieser Art erfreut sich derzeit einer hohen Nachfrage.
Neben den Bediensteten lassen sich aber auch Externe Teilnehmer als weitere Zielgruppe bestimmen. Die Einbindung dieser Personenkreise unterscheidet sich am stärksten in den Unterschiedlichen Einrichtungen und birgt großes Potenzial und Gefahr gleichzeitig. Je nach Organisationsform stehen dem Hochschulsport ganz unterschiedliche Möglichkeiten offen aber auch das regionale Umfeld muss hier genau in Betracht gezogen werden. Mit der Öffnung des Hochschulsports für externe Teilnehmer begibt man sich in direkte Konkurrenz mit kommerziellen Anbietern. Die Integrationsformen variieren von Kooperationen mit Vereinen, Firmen im Rahmen von Betriebssportangeboten, Rehakliniken und Gesundheitszentren bis hin zur freien Öffnung für Jedermann. Allgemein lässt sich jedoch beobachten, dass diese Gruppe vor allem dann angesprochen wird, wenn eine Auslastung durch Bedienstete und Studierende nicht möglich ist. Im Bereich Örganisationsformen"werde ich auf die Unterschiede noch detaillierter eingehen.

2.2 Typische Aufgaben

2.3 Organisationsformen

Die zentralen Einrichtungen
die – unabhängig von einem Fachbereich und direkt dem Senat unterstellt – den Hochschulsport eigenständig organisieren und verwalten

Die in das Aufgabenfeld der Institute für Sportwissenschaft integrierte Organisation des Hochschulsports

(diese enge Verbindung mit der Sportlehrer/innenausbildung kann sich auf das Hochschulsportangebot äußerst positiv auswirken, bringt aber auch durch notwendige „Prioritätensetzungen“ die Gefahr der Benachteiligung mit sich);

Der durch studentische Selbstverwaltung organisierten Hochschulsport
(diese Form hat sich vor allem an kleineren Hochschulen bzw. Fachhochschulen etabliert, wo keine hauptamtlichen Sportlehrer/innen zur Verfügung stehen) (vgl. Radde, 1996)

2.4 Befragung ausgewählter Standorte

2.5 Wichtige Geschäftsprozesse

1. Kunden anlegen
2. Kunden verifizieren
3. Kurse anlegen
4. Kurse finden
5. Kurse darstellen
6. Kurse buchen
7. Geld einziehen
8. Rechnungsstellung
9. Teilnehmer informieren
10. Übungsleiter abrechnen
11. Sportstätten vermieten
12. Verträge verwalten
13. Zutritt überprüfen
14.
15.

2.6 Probleme und Herausforderungen

3 Technische und fachliche Grundlagen

Der dritte Teil dieser Diplomarbeit vermittelt die technischen Grundlagen für einige ausgewählte Bereiche und bereitet die Basis für das Verständnis der gewählten Konzepte in Teil vier. Die Untergliederung in Softwarearchitekturen, System Kommunikation, Datenbanksysteme und Konzepte stellt dabei nur eine grundlegende Einteilung dar und bilden einen Querschnitt durch wichtige Bereiche der Anwendungsarchitektur. Die Aufzählung stellt dabei keinen Anspruch auf Vollständigkeit und die Beschreibungen können nicht bis ins kleinste Detail vollzogen werden, da dies den Rahmen dieser Arbeit sprengen würde. Die Darstellung soll jedoch tiefgründig genug sein, um dem Leser ein fundiertes Verständnis der einzelnen Thematiken zu vermitteln.

3.1 Softwarearchitekturen

Die Definitionen, was Softwarearchitektur ist und wie dieser Term zu definieren ist, unterscheidet sich bei einzelnen Autoren. Bass, Clements und Kazman (2013, S. 4) verwenden für mich eine sehr passende Definition des Begriffes:

> *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*

Vor allem die Zusammenhänge sind hier von Bedeutung, denn auch wenn das Design der Architektur nicht endgültig sein muss und sich im Laufe der Entwicklung noch Veränderungen und Anpassungen ergeben könne, so ist es doch von Anfang an wichtig eine Vorstellung über die geplanten Zusammenhänge im gesamten Team zu entwickeln.
Ich will daher hier drei grundlegende Architekturen darstellen, die sich in den Zusammenhängen unterscheiden. Jede diese Architekturen lässt sich optimieren, anpassen und so besser nutzbar machen und oftmals gibt es auch unterschiedliche Kombinationen die zum gewünschten Erfolg führen.

3.1.1 Monolitisch

Eine der am weitest verbreiteten Software Architekturen ist die monolithische Architektur. Gerade zentrale Server Anwendungen sind häufig nach diesem Prinzip aufgebaut. Die Stärken dieser Architektur liegen in:

- Einfach zu entwickeln - Das Ziel bestehender Entwicklungstools ist es die Erstellung monolithischer Programme zu unterstützen
- Einfach auszuliefern - Nur die Anwendung als Ganzes muss ausgeliefert werden
- Einfach zu skalieren - Hinter einem Lastverteiler können einfach mehrere Kopien der Anwendung laufen

Durch diese Vorteile hat sich dieses Architektur Prinzip weit verbreitet und ist jedem Entwickler bekannt. Gerade für kleine bis mittelgroße Projekte sind die genannten Faktoren entscheidend. Der interne Aufbau der Anwendung kann dabei jedoch sehr unterschiedlich gestaltet

sein. In den meisten Fällen ist allerdings ein Schicht-Aufbau zu erkennen, der unterschiedliche Komponenten intern trennt. Gängige Schichten in einer Webanwendung sind folgende:

- Präsentation - Darstellung und Verwalten von HTTP Anfragen und Ergebnissen in HTML oder JSON/XML Format
- Geschäftslogik - Interne Logik für Geschäftsabläufe
- Datenbank Zugriff - Zuständig für alle Zugriffe auf die Datenbank
- Anwendungsintegration - Nachrichten Transfer, Email etc.

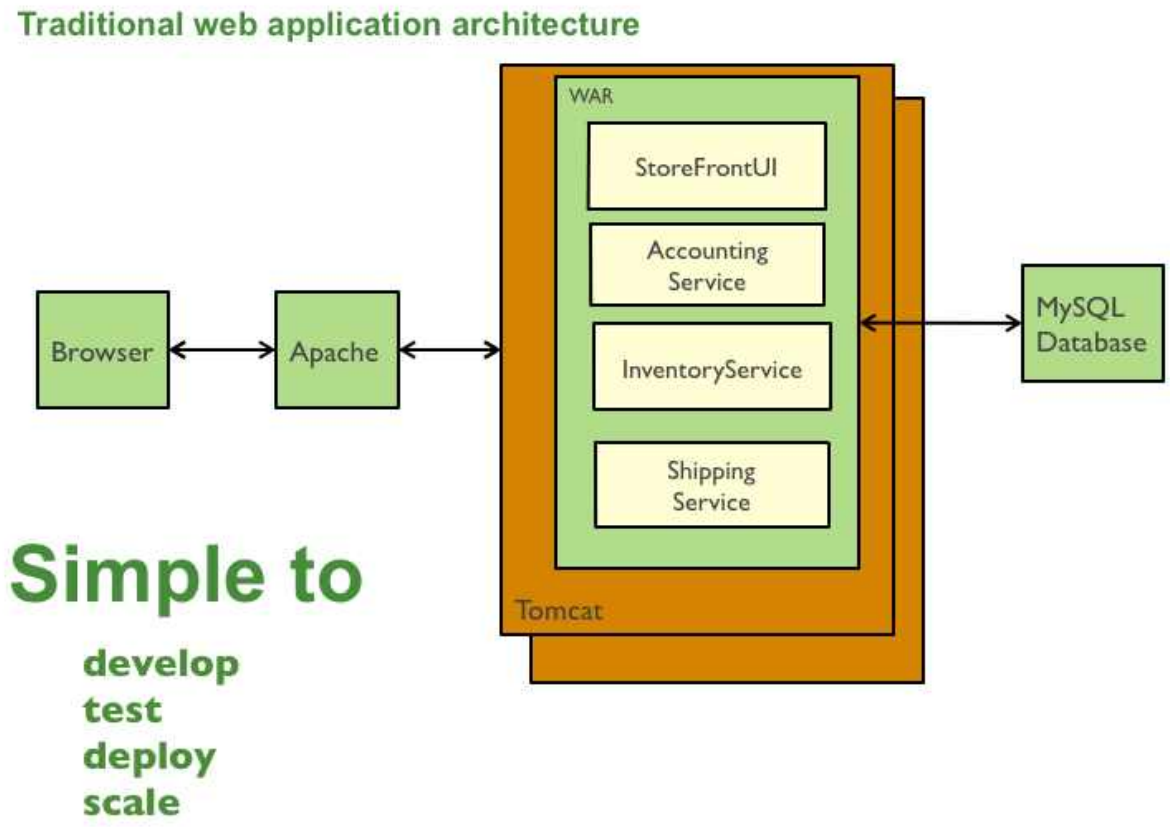


Abbildung 3.1: Traditionelle Webanwendungs Architektur

Richardson (2014) sieht jedoch auch eine Reihe von Nachteilen dieser Architektur, die mit zunehmender Größe der Anwendung und wachsender Teamgröße immer signifikanter werden.

- Die große monolithische Codebasis kann neue Entwickler schnell einschüchtern. Änderungen und Verständnis können schwierig sein und die Entwicklung verlangsamen. Durch fehlende Modulgrenzen geht im laufe der Zeit die Modularität verloren.
- Mit wachsender Größe er Codebasis werden die IDEs (Integrierte Entwicklungsumgeben) langsamer und bremsen die Produktivität
- Anwendungsstart wird zunehmend länger und hat einen großen Effekt auf die Produktivität der Entwickler und die Auslieferung. Wartezeit ist verschwendete Zeit.

- **Continuous deployment is difficult**
- Das Skalieren der Anwendung kann schwierig werden, da es nur in eine Dimension skalieren kann. Mit wachsenden Transaktionen kann das System durch zusätzlich Instanzen skaliert werden. Hingegen kann bei einem wachsenden Datenvolumen diese Architektur nicht gut skalieren. Jede Instanz greift auf alle Daten zurück und führt zu wenig effektivem Caching, höherem Speicherverbrauch sowie zusätzlichem I/O traffic. Außerdem können unterschiedliche Anwendungsbereiche verschiedene Anforderungen z.B. CPU haben, das skalieren von einzelnen Komponenten ist jedoch unmöglich.
- Die monolithische Architektur hat Grenzen wie die Entwicklung skalieren kann. Bei einer entsprechenden Anwendungsgröße werden Teams gebildet, die für spezielle Bereiche verantwortlich sind z.B. UI Team, Buchhaltungs Team etc. Die enge Verzahnung der einzelnen Bereiche erschwert jedoch die unabhängige Arbeit.
- Mit dem Beginn der Entwicklung muss sich auf eine Technologie und ggf. eine Version festgelegt werden, die es sehr schwer macht neue Technologien zu adaptieren. Eine inkrementelle Umstellung gestaltet sich als schwierig, wobei eine vollständige Ersetzung der aktuellen Technologie sehr kostenintensiv ist und das Problem im Kern nicht behebt.

(vgl. Richardson, 2014)

3.1.2 Client/Server-Architekturmodell

Im Gegensatz zur monolitischen Architektur handelt es sich beim Client/Server Modell um ein verteiltes System.
Das Client/Server-Architekturmodel ist ein Systemmodell, das aus einer Anzahl von zugehörigen Diensten und Servern besteht, sowie aus Clients, die diese Dienste nutzen und auf sie zugreifen. Das einfachste Modell ist dabei das zweischichtige Client/Server Modell, die zusammen ein komplettes System bilden mit unterschiedlichen Zuständigkeiten.
Dabei lassen sich drei logische Hauptbestandteile unterschieden.

Server

Der Server ist die zentrale Einheit der Architektur. Sie stellt Services und Daten anderen Einheiten zur Verfügung. Der Server muss nicht zwingend wissen, welche und wie viele Einheiten die von ihm zur Verfügung gestellten Dienste und Daten nutzen.

Client

Clients können in beliebig großer Anzahl existieren. Sie nehmen die Daten oder Services, die vom Server bereitgestellt werden in Anspruch. Um mit den Diensten interagieren zu können, muss ein Client von der Existenz und dessen Adresse wissen. Der Client sendet eine Anfrage an den Server und bekommt ein Resultat zurück geliefert. Sommerville (2007, S. 302) unterscheidet folgende zwei Client-Modelle:

Thin-Client-Modell

Bei diesem Modell werden die gesamten Anwendungsprozesse und Datenhaltung auf dem Server erledigt. Der Client ist nur für die Darstellung verantwortlich. Daraus ergibt sich der Vorteil,

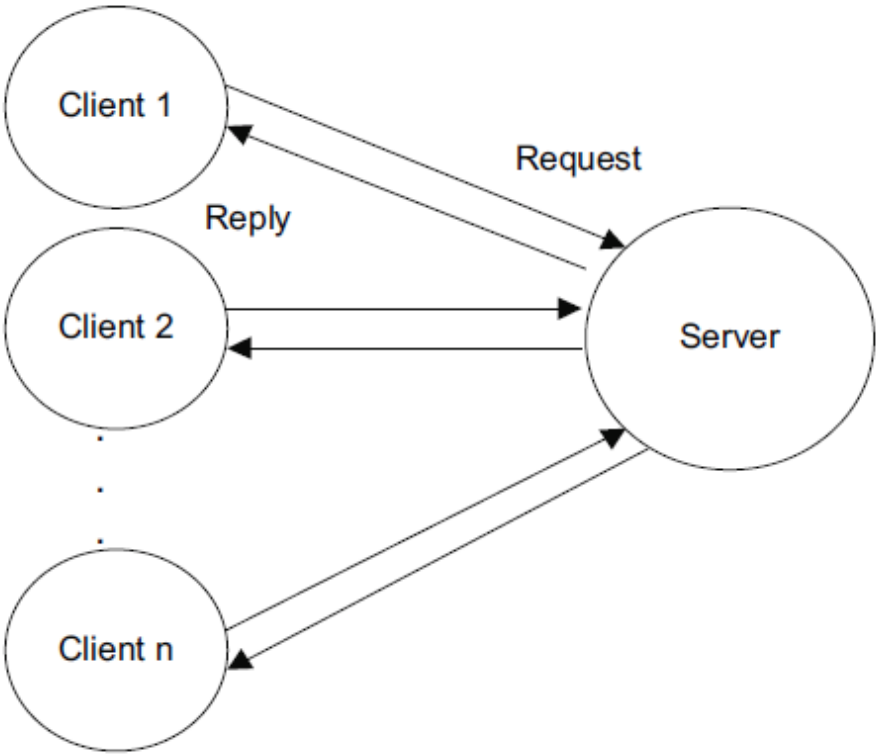


Abbildung 3.2: Client Server Model

dass sehr wenig Ressourcen für den Client benötigt werden und dieser sich auch auf kleinen Systemen (z.B. Terminals) implementieren lässt. Der Nachteil dieses Modelles liegt jedoch darin, dass so sämtliche Arbeit Serverseitig geleistet werden muss. Gerade bei Anwendungen mit großer Client Anzahl oder rechenintensiven Operationen kann so der Server schnell zur kritischen Komponente werden und das System vor Skalierungsprobleme stellen. Zusätzlich führt der Thin-Client zu einer erhöhten Netzwerkbelastung, da alle Daten vom Server zum Client transportiert werden müssen.

Fat-Client-Modell

Dieses Modell verlagert die Anwendungslogik vollständig oder in großen Teilen vom Server direkt in den Client. Dadurch kann der Server deutlich entlastet werden und die Rechenleistung des Client Rechners zusätzlich genutzt werden. Dies kann sich in speziellen Situationen allerdings auch als Nachteile erweisen, wenn die Client Computer die notwendige Rechenleistung nicht selbstständig bereitstellen könne. Der Server fungiert dabei nur noch als Daten Service und die Netzwerklast wird reduziert. Die Verlagerung der Anwendungslogik auf den Client bringt aber auch Nachteile mit sich. Müssen hier Änderungen vorgenommen werden, so müssen alle Clients auf die neue Version aktualisiert werden, was zu einem komplexen Systemmanagement führen kann. Der Aufwand dafür steigt mit zunehmender Zahl der Clients.

Netzwerk (optional)

In der Literatur wird das Netzwerk häufig noch ein dritter Bestandteil aufgeführt, der jedoch nicht zwingend notwendig ist da prinzipiell Server und Client auf einem Computer ausgeführt

werden können und so kein Netzwerk zur Kommunikation benötigen. Da dieses Modell aber in der Praxis nahezu ausschließlich als verteiltes System angewendet wird, ist in diesen Fällen das Netzwerk grundlegende Kommunikationsgrundlage.

Generell hat das zweischichtige Client/Server Modell das Problem, dass die drei logischen Schichten auf zwei unterschiedlichen Computersystemen abgebildet werden müssen.

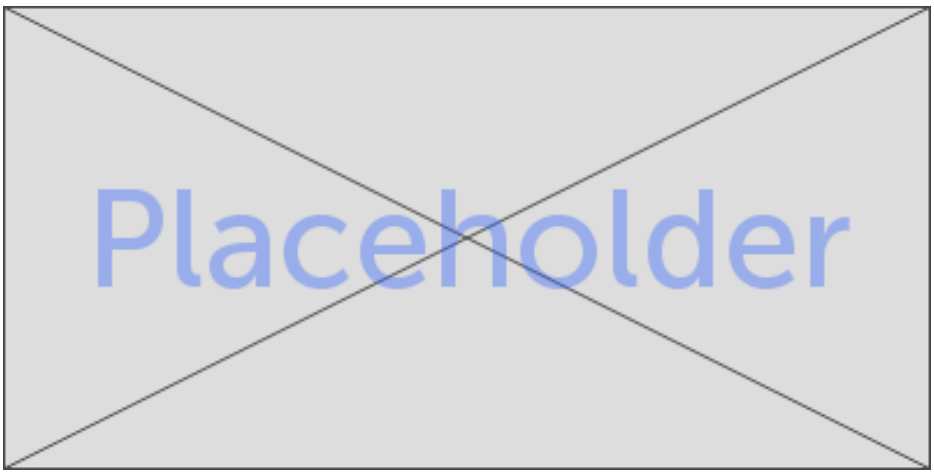


Abbildung 3.3: Logische drei Schichten

Als Alternativansatz hat sich dafür die dreischichtige Architektur Entwickelt. Sie stellt für jede logische Schicht ein separates Computersystem zur Verfügung. Die Vorteile liegen dabei in der besseren Skalierbarkeit, da jedes System individuell skaliert werden kann, geringere Netzwerklast im Vergleich zum Thin-Client-Modell und Anwendungsprozesse sind zentralisiert und lassen sich dort leichter anpassen und aktualisieren.

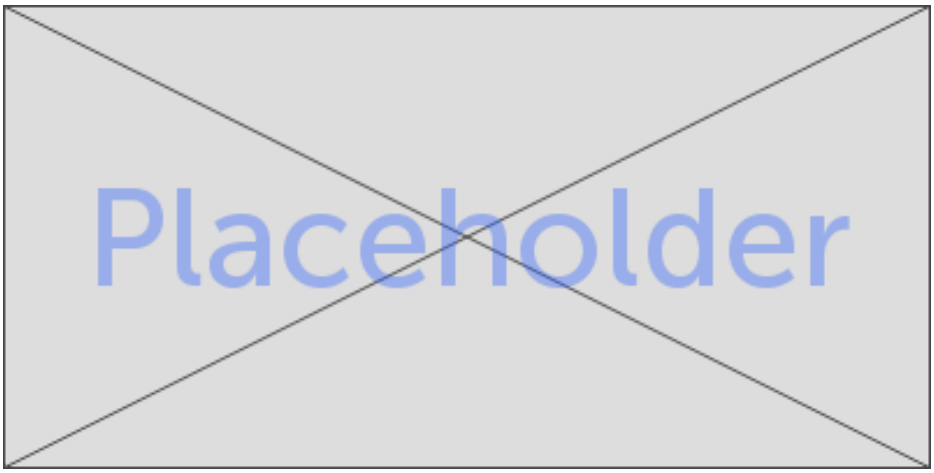


Abbildung 3.4: Drei schichtige Architektur

Sommerville gibt in seinem Buch Software Engineering Beispiele, wann die Anwendung der jeweiligen Architektur sinnvoll ist.

Architektur	Anwendung
Zweischichtige C/S-Architektur mit Thin-Clients	Anwendungen mit Legacy-Systemen, bei denen die Trennung von Anwendungsprozessen und Datenmanagement nicht praktikabel ist Rechenintensive Anwendungen wie zum Beispiel Compiler mit geringem oder keinem Datenmanagement -Datenintensive Anwendungen (Browser oder Abfragesysteme) mit wenig oder keinen Anwendungsprozessen
Zweischichtige C/S-Architektur mit Fat-Clients	Anwendungen, bei denen die Prozesse durch Standardanwendungen (z.b. Microsoft Excel) auf dem Client verarbeitet werden Anwendungen mit rechenintensiver Datenverarbeitung (z.B. Datenvisualisierung) Anwendungen mit relativ stabiler Endbenutzerfunktionalität, die in einer Umgebung mit gut strukturiertem Systemmanagement angeordnet sind
Drei- oder mehrschichtige C/S-Architektur	Anwendungen großen Umfangs mit Hunderten oder Tausenden von Clients Anwendungen, bei denen sowohl die Daten als auch die Anwendung selbst ständigen Veränderungen unterliegen Anwendungen, bei denen Daten aus vielfältigen Quellen verarbeitet werden

3.1.3 Dienstbasiert

Neben den beiden oben genannten Architekturen hat sich eine weitere Form verbreitet. Bei der dienstbasierten Architektur spricht man von einem Architketurmuster aus dem Bereich der verteilten Systeme. Der Schwerpunkt wird dabei auf eigenständige Dienste als primäre Komponente gelegt, auf die über eine Art Remote-Protokoll zugegriffen werden kann. Die Art des Protokolles ist dabei nicht festgelegt oder auf eines beschränkt wie z.B. Representational State Transfer (REST), Simple Object Access Protocol (SOAP), Advanced Message Queuing Protokol (AMQP) oder andere geschehen. Dienstbasierten Architekturen werden signifikante Vorteile im Bezug auf Skalierbarkeit, Entkopplung, Kontrolle über die Entwicklung, Testing und Deployment zugeschrieben. Verteilte Systeme tendieren zudem dazu weniger abhängig und besser modular aufgebaut zu sein. Im Zusammenhang mit dienstabasierten Architekturen bedeutet das, das jeder Dienst eine in sich geschlossen Einheit bildet, sie selbst designed, entwickelt, getestet und ausgeliefert wird. Abhängigkeiten zu anderen Komponenten sollten dabei nicht, oder nur minimal bestehen.
Diese Vorteile bringen jedoch auch einige Nachteile mit sich, so zählen erhöhte Komlexität und höhere initiale Kosten zu den meist genannten Argumenten.

3.2 System Kommunikation

3.2.1 Representalional State Transfer (REST)

REST wurde erstmals von 2000 Roy Thomas Fielding in seiner Dissertation beschrieben. Im Gegensatz zu bekannten Protokollen wie Simple Object Access Protokol (SOAP) oder XML-RPC handelt es sich bei REST um einen Architekturstil, wie Anwendungen basierend auf dem HTTP Protokoll miteinander kommunizieren können. Unterstützt eine Anwendung die REST-Stil, so

bezeichnet man sie als RESTful. (vgl. Melzer, 2010)
REST basiert auf einem Konzept von Ressourcen, über die der Service Kenntnis hat. Der Server erstellt dabei verschiedene Darstellungen, wobei diese entkoppelt ist von der internen Speicherung. HTTP und HTTPS sind die primär benutzten Protokolle bei der Verwendung von REST. HTTP Caching Proxies, Load-Balancers und Monitoring Tools besitzen einen tiefe Unterstützung für HTTP von Haus aus. Den HTTP Verben GET, POST, PUT und DELETE lösen in RESTful Anwendungen spezielle Aktionen aus.

HTTP Verb	REST Aktion
GET	Abfragen einer Ressource ohne Änderungen vorzunehmen
POST	Erstellt eine neue Ressource. Der Link zur neuen Ressource wird zurück gegeben
PUT	Eine Ressource wird angelegt. Wenn die Ressource bereits existiert, wird sie geändert.
PATCH	Ein Teil der Ressource wird geändert
DELETE	Löscht eine Ressource

Der große Vorteil und der Grund für die schnelle Verbreitung von REST Architekturen liegt in der Einfachheit. Methoden für die Nutzung des HTTP Protokoll sind in allen modernen Programmiersprachen enthalten und können einfach genutzt werden. Das Ressourcen Konzept ist leicht anwendbar und bietet umfangreiche Möglichkeiten der eigenen Anpassung.

Die Bandbreite der Möglichkeiten, die mit der REST Architektur umgesetzt werden können zeigt Leonard Richardson in seinem ´´Richardson Maturity Model``indem er vier REST Level beschreibt.

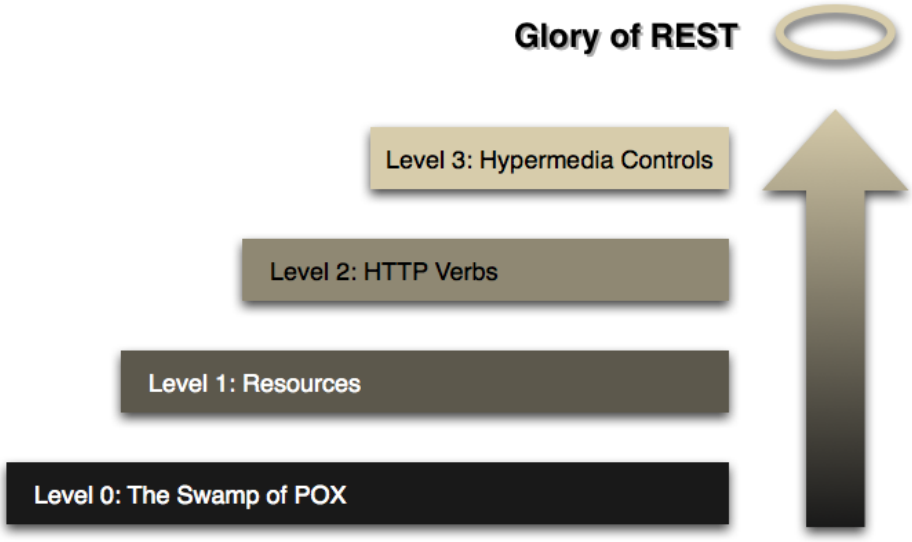


Abbildung 3.5: Richardson Maturity Model

(vgl. Fowler, 2010)

3.2.2 Enterprise Service Bus (ESB)

3.2.3 Message Bus

3.3 Datenbanksysteme

3.3.1 Relationale Datenbanken

3.3.2 NoSQL Datenbanken

3.3.3 Gegenüberstellung

3.4 Konzepte

3.4.1 Separation of Concerns

3.4.2 Domain Driven Design

3.4.3 Data Consistency Primer

3.4.4 Competiting Consumer Pattern

3.4.5 Materialized View Pattern

3.4.6 Queue-Based Lode Leveling Pattern

3.4.7 Api Gateway Pattern

4 System Konzeption

Nachdem in den ersten Kapiteln dieser Arbeit die grundlegenden fachspezifischen Anforderungen der Hochschulsporteinrichtungen an ein Softwaresystem sowie einige wichtige Konzepte der Systementwicklung vorgestellt wurden, soll im vierten Teil ein Konzept erstellt werden, wie eine Umsetzung unter Betrachtung der Anforderungen aussehen kann. Des Weitern sollen in dieses Konzept Anforderungen aus Sicht des Software Erstellers mit berücksichtigt werden, sodass ein einstehendes Konzept theoretisch auch in der Praxis umgesetzt werden kann.
Der Aufbau der Konzeption erfolgt dabei in Anlehnung an Balzert (1996) und beinhaltet die Phasen:

1. Planungsphase
2. Definitionsphase
3. Entwurfsphase

4.1 Planungsphase

- Lastenheft
- Nutzeranforderungen
- Systemanforderungen
- Multi-Tenantcy
- PaaS, SaaS
- Glossar

- WAS und nicht WIE
- Priorisierung?

4.2 Definitionsphase

- Pflichtenheft
-

4.3 Entwurfsphase

- WIE
- Entwerfen einer Software-Architektur
- Zerlegung des definierten Systems in Systemkomponenten
- Strukturierung des Systems durch geeignete Anordnung der Systemkomponenten
- Beschreibung der Beziehungen zwischen den Systemkomponenten
- Festlegung der Schnittstellen, über die die Systemkomponenten miteinander kommunizieren.

Microservice Diagram
https://dzone.com/articles/building-microservices-inter-process-communication-1
https://insidethecpu.com/2015/07/17/microservices-in-c-part-1-building-and-testing

5 Diskussion, Zusammenfassung und Ausblick

5.1 Glossar

5.2 Anhänge

Literaturverzeichnis

Balzert, H. (1996). *Lehrbuch der Software-Technik - Software-Entwicklung*. Heidelberg [u.a.]: Spektrum Akadem. Verl.

Bass, L., Clements, P. & Kazman, R. (2013). *Software architecture in practice* (3rd ed Aufl.). Upper Saddle River and NJ: Addison-Wesley.

Fowler, M. (2010). *Richardson Maturity Model.* Zugriff am 06.07.2016 auf `http://martinfowler.com/articles/richardsonMaturityModel.html`

Melzer, I. (2010). *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis* (4. Aufl.). Heidelberg: Spektrum Akademischer Verlag.

Radde, G. (1996). Hochschulsporteinrichtungen - Eine vergleichende Betrachtung. *dvs-Informationen*, *11* (3), 15–17. Zugriff am 05.07.2016 auf `http://www.sportwissenschaft.de/fileadmin/pdf/dvs-Info/1996/1996_3_radde.pdf`

Richardson, C. (2014). *Pattern: Monolithic Architecture.* Zugriff am 05.07.2016 auf `http://microservices.io/patterns/monolithic.html`

Sommerville, I. (2007). *Software-Engineering* (8. Aufl.). München and Boston [u.a.]: Pearson Studium.