



Globalizer

Detailkonzpet

Koller, Jonas

`jonas.koller@gmx.ch`

Wolfisberg, Donato

`donato.wolfisberg@gmail.com`

18.10.2018





1 Einleitung

Dieses Dokument behandelt das Detailkonzept des Projekts Globalizer. Ziel und Zweck des Dokuments ist es, die fachlichen Anforderungen technisch zu umschreiben, damit diese später implementiert werden können. Zudem soll das Gesamtbild unserer Applikation klar gemacht werden und die Komponenten, sowie der Datenaustausch genau beschrieben werden. Es werden auch all unsere Technologieentscheide vermerkt und begründet. Zusammengefasst soll dieses Dokument also eine klare, wie auch detaillierte Übersicht zu den technischen Aspekten unseres Projekts sein.

2 Allgemeine Informationen

Hier folgt eine kurze Auflistung der Informationen zu diesem Dokument, dem Entwicklungsteam und dem aktuellen Stand.

2.1 Projektmitarbeiter

Name	Vorname	E-Mail	Funktion
Koller	Jonas	jonas.koller@gmx.ch	Projektleiter
Wolfisberg	Donato	donato.wolfisberg@gmail.com	Entwickler
Gian	Ott	gian_ott@sluz.ch	Pr��fer
Manuel	Troxler	manuel_troxler@sluz.ch	Pr��fer

2.2   nderungskontrolle

Version	Datum	Ausf��hrende Stelle	Bemerkung
1	21.09.2018	Projektteam	Erste Version des Dokuments erstellt
2	23.09.2018	Projektteam	Gesamt��berblick erstellt
3	25.09.2018	Projektteam	Zielkatalog erstellt
4	27.09.2018	Projektteam	Abschliessende Arbeiten

2.3 Pr  fung

Version	Datum	Ausf��hrende Stelle
4	27.09.2018	Gian Ott

Inhalt

1	Einleitung	I
2	Allgemeine Informationen	II
2.1	Projektmitarbeiter	II
2.2	Änderungskontrolle	II
2.3	Prüfung	II
3	Systemkomponenten	1
3.1	Frontend	1
3.2	Backend	1
3.3	Datenbank	1
3.4	Zonen-Übersicht	2
4	Analyse des IST-Systems	2
4.1	Zielsetzungen	2
4.1.1	Muss-Ziele	2
4.1.2	Kann-Ziele	3

3 Systemkomponenten

Das Projekt verwendet eine klassische Client - Server Architektur. Somit ergeben sich bei uns drei Hauptkomponenten. Dies ist das Backend, das Frontend und die Datenbank. Wir werden folgend beschreiben, wie diese genau aufgebaut sind.

3.1 Frontend

In der Frontend-Komponente soll die Schnittstelle zwischen Endbenutzer und Backend-System implementiert werden. Wir werden dies mit einer Weboberfl che machen. Diese Komponente soll unabh ngig vom Backend-System auf einem CDN gehostet werden k nnen. Diese Massnahme verringert die Wartezeiten des Endnutzers deutlich.

3.2 Backend

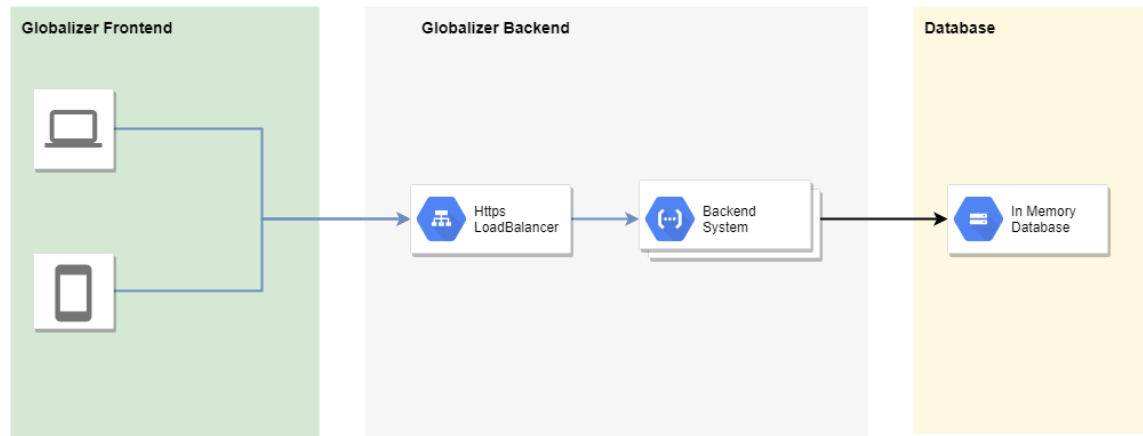
Unser Backend-System soll die Schnittstelle zwischen den Daten und dem Frontend abbilden. Hier werden die rohen Daten von der Datenbank aufbereitet, Berechnungen durchgef hrt und die Security-Aspekte gr sstenteils abgedeckt. Die Authentifizierung, Autorisierung und  berpr fung der  bermittelten Daten soll hier stattfinden. Das Backend soll m glichst klein gehalten werden, damit der Wartungsaufwand minimal bleibt. Es soll so gebaut werden, dass es gut skalierbar verwendet werden kann. Aspekte wie Load-Balancing und Upscaling sollen vom Hoster  bernommen werden.

3.3 Datenbank

Die Datenbank bildet den dritten Teil unseres Systems. Hier soll eine InMemory-Datenbank verwendet werden. Diese braucht weniger Leistung und sorgt f r zus tzliche Geschwindigkeit beim System. Uns ist bewusst, dass die Datenbank ihren aktuellen Stand bei einem Systemabsturz verlieren kann. Da es sich jedoch um einen fl chtigen Chat handelt, welcher nicht zu 100% sicher persistiert werden muss, kann eine InMemory-DB verwendet werden.

3.4 Zonen-Übersicht

Auf der nachfolgenden Grafik kann unsere Architektur im groben ausgelesen werden. Das System ist in die drei "Komponenten-Zonen" unterteilt. Diese sollen wenn möglich unabhängig von den anderen deploybar sein. Dies ermöglicht es uns später auch, unsere Deployment-Zyklen zu vereinfachen. Im Backend-System ist zusätzlich auch der LoadBalancer eingezeichnet, welcher vom Hoster übernommen wird.



4 Design-Pattern

Da weder unser Backend, noch unser Frontend objektorientiert ist, können keine Klassendiagramme erstellt werden. OOP-Eigenschaften wie Vererbungen und Beziehungen werden bei diesem Projekt auch kaum benutzt. Aus diesem Grund verzichten wir auf diese Diagramme. Wir verwenden anstelle dessen aber ein UML-Aktivitätsdiagramm, um unsere Abläufe zu visualisieren.

4.1 Architektur Backend

Das Backend wird in NodeJS erstellt. Es nutzt Websockets zur Kommunikation mit dem Frontend. Das WebSocket-Protokoll ist auf TCP basiert und erlaubt bidirektionale Verbindungen zwischen Server und Client. Das Backend ist offen für neue Verbindungen. Wenn sich ein neuer Benutzer anmeldet, registriert dieser sich beim Backend. Dieses hört danach auf Anfragen des Frontends. Im gegenzug sendet das Backend neue Nachrichten selbst an das Frontend. Durch dieses "Bidirectional Message Pattern" können wir einiges an Bandbreite einsparen, da wir nicht immer wieder Fetch-Anfragen senden müssen um den Client auf dem aktuellen Stand zu





behalten. Wir konnten somit das "Fetch"-Antipattern umgehen, da dieses für eine Chat-Applikation ungeeignet ist. In der folgenden Grafik kann ausgelesen werden, wie die der Ablauf bei Anfragen an den Server aussehen.

5 Analyse des IST-Systems

Im Moment wird die Kommunikation zwischen den Schülern und den Lehrern über diverse Kanäle geführt: Mail, Telefon, Whatsapp und weitere. Die neue Applikation kann diese Kanäle nicht ersetzen, sondern im Bereich des Chats die Austausch von Informationen auf anonymer Basis ermöglichen.

5.1 Zielsetzungen

5.1.1 Muss-Ziele

1.  Globaler Gruppen Chat
2.  Hinterlegen eines Benutzernamens
3.  Benutzer muss den Chat später wieder aufnehmen können. z.B. Cookies oder Session Storage
4.  Die Seite für Mobile geräte optimieren

5.1.2 Kann-Ziele

1.  Private Chats zwischen zwei Personen
2.  Authentifizierung über Google Accounts, aber trotzdem anonym