



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	Design-Pattern	2
4.1	Allgemeine Architektur-Pattern	2
4.2	Codestyle und Codequalität	3
4.3	Architektur Backend	3
4.4	Architektur Frontend	4
5	Persistenz	4
6	Detaillierte Komponenten und Schnittstellen	4
6.1	Backend	4
6.2	Funktionale Anforderungen	5
6.3	Schnittstellen	5
6.4	Persistenz	5
6.5	Frontend	5
6.6	Funktionale Anforderungen	6
6.7	Schnittstellen	6
6.7.1	Android frame	6
6.7.2	Desktop Chrome Login	7
6.7.3	Desktop Chrome Chat	8
6.8	Persistenz	8
7		8
7.1	Zielsetzungen	9
7.1.1	Muss-Ziele	9

7.1.2	Kann-Ziele	9
-------	----------------------	---

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 Weboberfläche machen. Diese Komponente soll unabhängig vom Backend-System auf einem CDN gehostet werden können. Diese Massnahme verringert die Wartezeiten des Endnutzers deutlich.

3.2 Backend

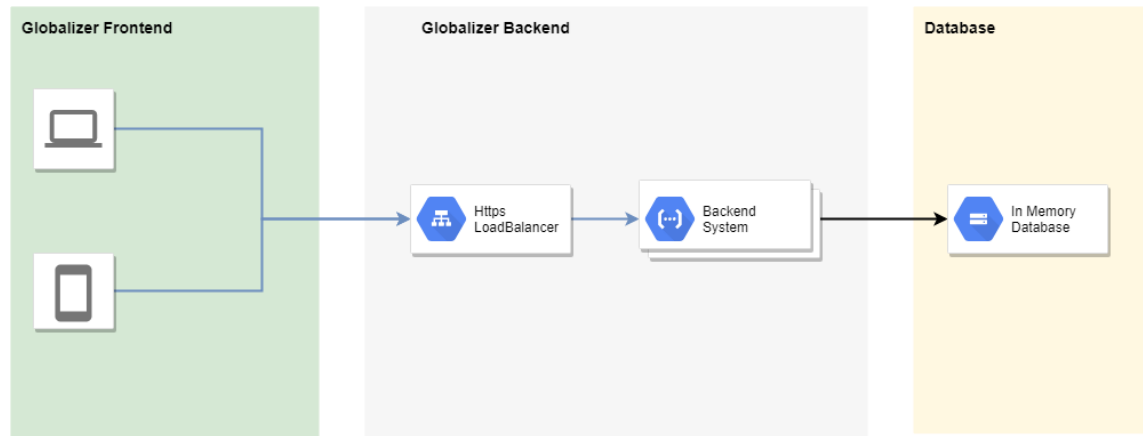
Unser Backend-System soll die Schnittstelle zwischen den Daten und dem Frontend abbilden. Hier werden die rohen Daten von der Datenbank aufbereitet, Berechnungen durchgeführt und die Security-Aspekte grösstenteils abgedeckt. Die Authentifizierung, Autorisierung und Überprüfung der übermittelten Daten soll hier stattfinden. Das Backend soll möglichst 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 übernommen 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 für zusätzliche Geschwindigkeit beim System. Uns ist bewusst, dass die Datenbank ihren aktuellen Stand bei einem Systemabsturz verlieren kann. Da es sich jedoch um einen flüchtigen 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. Wir verwenden in diesem Projekt keine Domain-Driven-Design, Test-Driven-Design oder ähnliches, da unser Projekt zu klein ist, damit dies den Aufwand wert wäre.





4.1 Allgemeine Architektur-Pattern

Wir wollen die Kommunikation zwischen Frontend und Backend möglichst dynamisch und asynchron halten, damit weder das Front- noch das Backend lange Wartezeiten aufweist oder "einfriert". Da die Kommunikation selbst aufgrund der Technologien nur schwer ganz asynchron umsetzbar ist, verwenden wir das in Angular häufig angewandte "pseudo Asynchronität". Die Anfragen an sich verlaufen synchron. Angular lässt aber zu, dass das Frontend währenddessen nicht blockiert wird und

arbeitet bis zur Antwort den restlichen Befehlsstack ab und hört auf Events. Im Backend läuft das auch so. Somit blockieren weder das Back- noch das Frontend und der Nutzer hat ein vielfach besseres Erlebnis beim verwenden der Applikation. Sowohl verwenden auch an beiden Orten das "Observer"-Pattern. Dies wird durch Angular und NodeJS unterstützt, in dem wir die "rxjs"-Library verwenden. Durch die einbindung dieser vereinfacht sich das Behandeln von Events und das reaktive Programmieren.

4.2 Codestyle und Codequalität

Damit unser Projekt auch über lange Zeit verwendbar, Wartbar und Erweiterbar ist, werden wir einige Programmierprinzipien anwenden, welche sicherstellen, dass unser Code langfristig sauber bleibt. Dies sind folgende:

-  Cleancode-Prinzip
-  SOLID-Prinzip
-  Pair-Programming-Prinzip
-  Feature-Banches und Pull-Requests

Falls einer der Begriffe unbekannt sein sollte, kann eine Erklärung zu diesem im Internet gefunden werden.

Wir verwenden Github als Versionskontrolle (mehr dazu bei den Technologieentscheiden). Damit wir die oben genannten Prinzipien einhalten, setzen wir Codefactory ein. Zudem verwenden wir auch TSLint, eine Clientseitige Library zur Überprüfung vom Codestyle.

Desweiteren verwenden wir einen Continuous Build System und ein Continuous Testing System, welches aktiviert wird bei einem "Push" ins Github Repository. Dazu auch später mehr.

4.3 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.

//TODO: Bild Backend aufrufe

4.4 Architektur Frontend

Unser Frontend wird mit Angular (Version 7) implementiert. Wieso genau wir uns für diese Technologie entschieden haben wird später erläutert. Dazu wird das Google Angular Material verwendet, damit wir dank dieser Library nicht alle UI-Komponenten selbst erstellen und designen müssen, sondern auf diese UI-Library zurückgreifen können (diese Library ist ähnlich wie Twitters Bootstrap). Das Frontend sendet auch via Websocket-Protokoll Nachrichten an das Backend.

5 Persistenz

Unsere Applikation verwendet wie bereits oben beschrieben eine InMemory Datenbank. Diese wird eine NoSQL Datenbank sein (NoSQL im Sinne von kein SQL verwendend und nicht darauf basierend). Sie basiert auf einem JSON-Datenmodell. Dies da wir in JavaScript programmieren werden und die Daten verwenden können, ohne diese zuerst parsen zu müssen. Die InMemory-DB wird durch Vanilla-JavaScript ermöglicht und braucht keine weitere Library. Ein konkretes Datenmodell gibt es nicht aufgrund von dem Fakt, dass wir dynamisch Daten speichern und entfernen und das genaue Datenmodell so entsteht.

6 Detaillierte Komponenten und Schnittstellen

Folgend werden alle Komponenten genau beschrieben und deren Schnittstellen definiert.




6.1 Backend

Die Backendkomponente bildet das "Hirn" unserer Applikation. Hier werden sowohl alle wichtigen Daten gespeichert (in der InMemory-DB), als auch alle wichtigen Impulse verwendet (Websockets). Ziel des Backends ist es eine möglichst hohe

flexibilität und Wartbarkeit mit einer möglichst hohen Performance zu kombinieren. Dies wird durch die sehr junge Backend-Sprache NodeJS ermöglicht.

6.2 Funktionale Anforderungen

Das Backend ist betroffen von 3 obligatorischen funktionalen Anforderungen. Diese sind:

-  USER_STANDARD_LOGIN
-  NACHRICHT_SENDEN
-  NACHRICHT_EMPFANGEN

Was diese Anforderungen konkret bedeuten kann im Dokument "Anforderungsspezifikationen" nachgelesen werden. Dort sind diese inklusiv Use-Case-Diagramm dargestellt.

6.3 Schnittstellen

//TODO: Schnittstellen beschreiben

6.4 Persistenz

Das Backend übernimmt den Hauptteil der Persistenz der ganzen Applikation. Hier werden alle Nachrichten und Benutzer in die InMemory-DB gespeichert und der Zustand der ganzen Applikation. Die Entitäten sehen wie gefolgt aus: //TODO: Bilder Entitäten




6.5 Frontend

Die Frontendkomponente bildet die Schnittstelle zwischen Endbenutzer und System. Hier werden die Daten aus dem Backend grafisch dargestellt. Wichtig an dieser Komponente ist, dass das Aussehen dieser modern und stylisch ist, damit der Nutzer ein tolles Erlebnis auf der Seite hat. Auch Geschwindigkeit ist hier ein wichtiges Thema, weshalb Angular 7 eine sehr gut passende Technologie ist. Die Funktionsweise des Frontends kann aus diesen UML-Diagramm ausgelesen werden.

//TODO: UML Flussdiagramm Frontend

6.6 Funktionale Anforderungen

Das Frontend ist betroffen von 3 obligatorischen funktionalen Anforderungen. Diese sind:

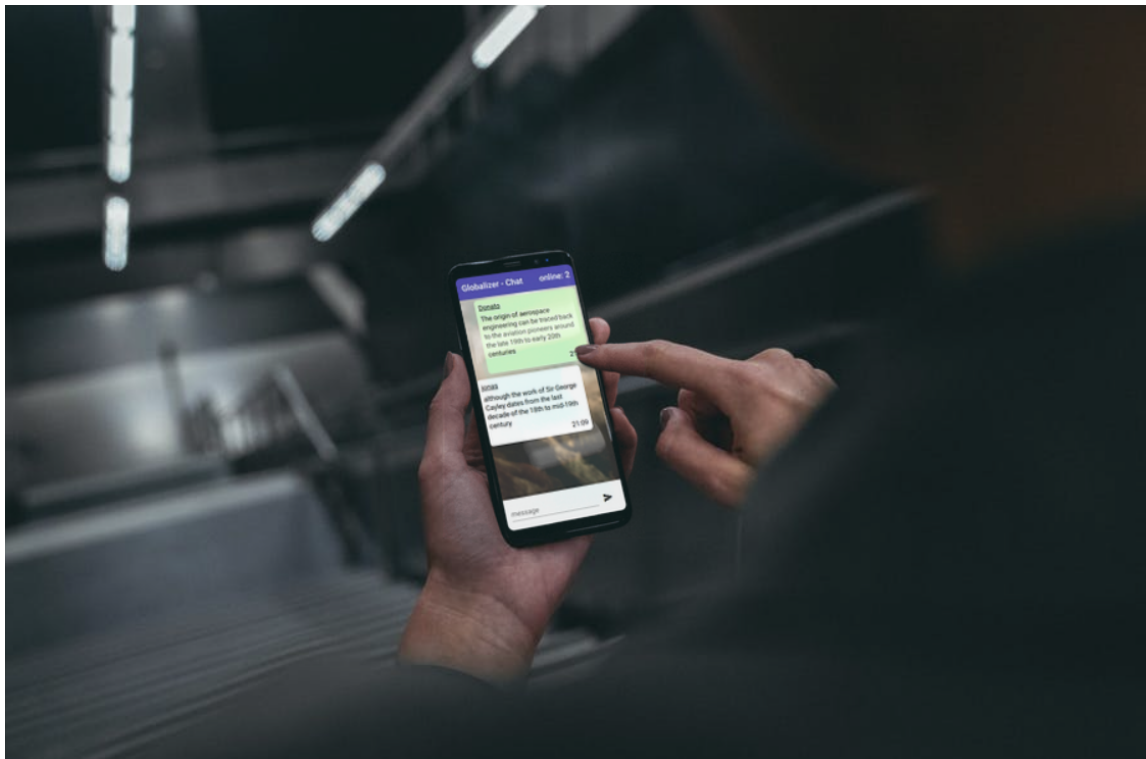
-  USER_STANDARD_LOGIN
-  NACHRICHT_SENDEN
-  NACHRICHT_EMPFANGEN

Was diese Anforderungen konkret bedeuten kann im Dokument "Anforderungsspezifikationen" nachgelesen werden. Dort sind diese inklusiv Use-Case-Diagramm dargestellt.

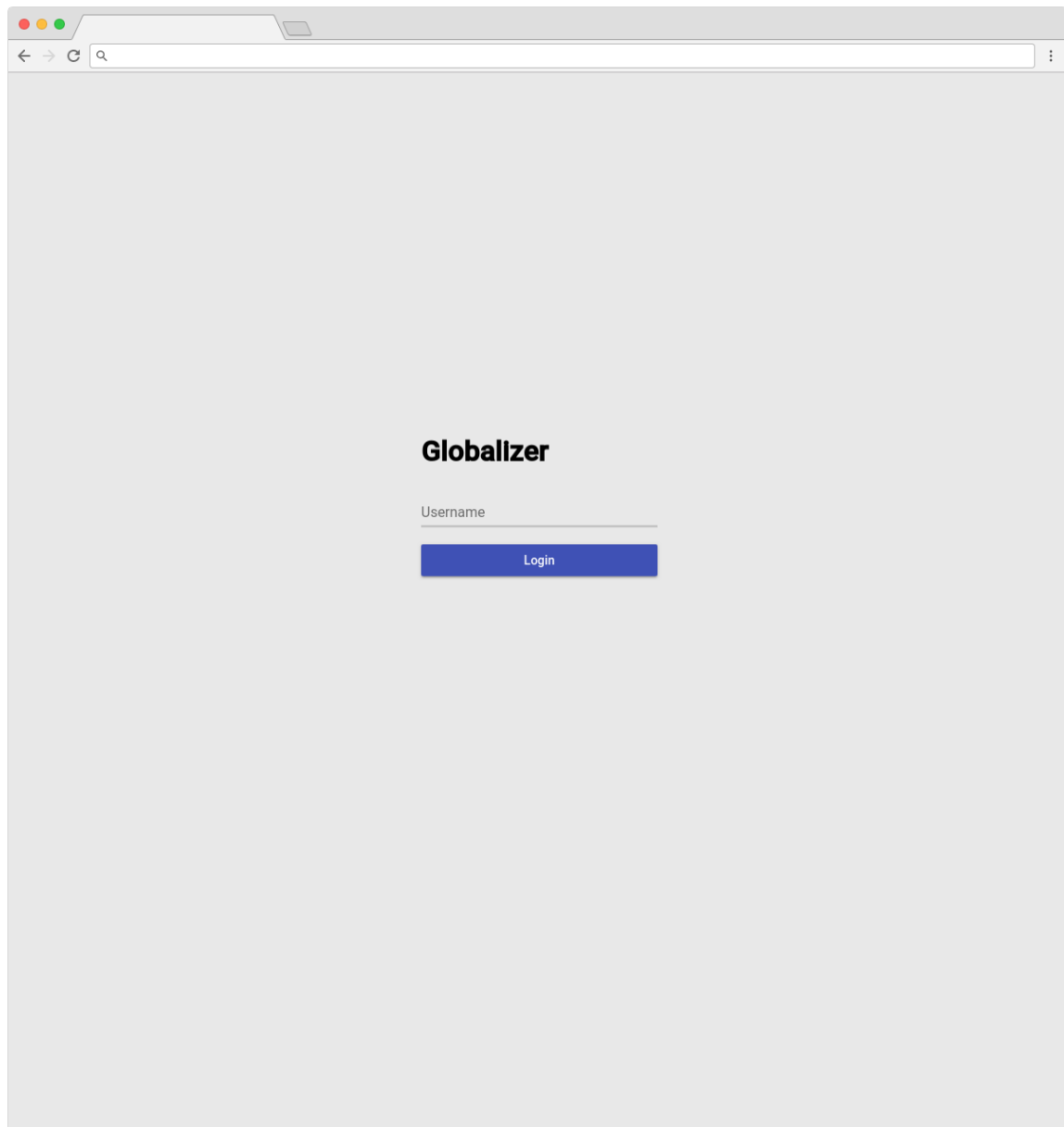
6.7 Schnittstellen

Die Schnittstelle zwischen Benutzer und Frontend ist rein visuell. Diese soll sp  ter so aussehen:

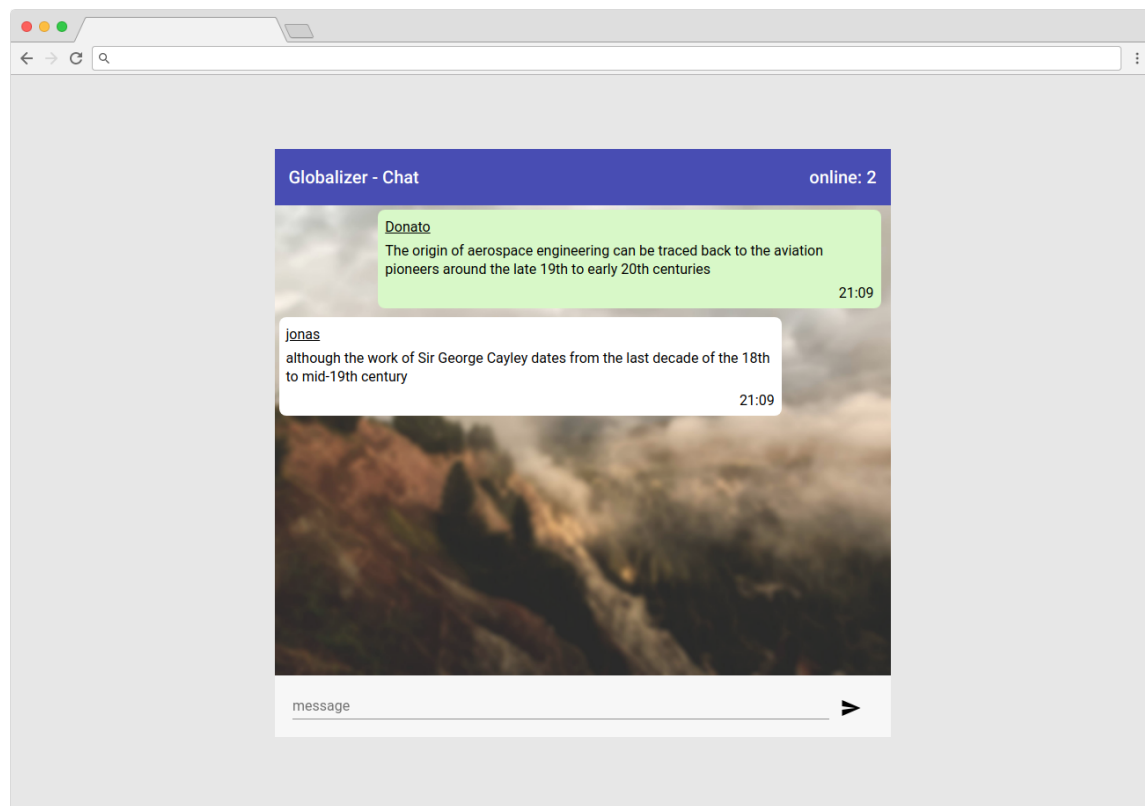
6.7.1 Android frame



6.7.2 Desktop Chrome Login



6.7.3 Desktop Chrome Chat



6.8 Persistenz

Das Frontend hat nur einen ganz minimalen Teil an Persistenz. Dazu wird der durch HTML 5 neu eingeführte LocalStorage eingesetzt. Es wird nach dem Login eines Benutzers dessen ID gesichert, damit später ein Sessionlogin durchgeführt werden kann und der Nutzer sich nicht erneut anmelden muss. Da diese Daten aber auf dem Client selbst gespeichert sind, müssen sie mit vorsicht behandelt werden.





7

Technologieentscheide //TODO: Wieso Github //TODO: Wieso Angular 7 //TODO: Wieso CI? Und Testing beschreiben //TODO: Testting pipeline bild //TODO: Hosting bei Heroku

Deployment //TODO: Deploymentdiagramm bild einfügen

7.1 Zielsetzungen

7.1.1 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

7.1.2 Kann-Ziele

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