Verteilte Systeme 2 Labor

Winter 2017 — Aufgabenstellung

Die Laboraufgabe besteht darin, einen **"Twitter-Klon"** als **Web-basierte verteilte Anwendung** zu entwickeln. Dabei sind **Anforderungen** an die Funktionen der Web Anwendung, verschiedene nicht-funktionale Eigenschaften, sowie die zu verwendenden Techniken vorgegeben.

Im Zuge der Gesamtaufgabe sind verschiedene **Teilaufgaben** zu erbringen, deren Ziele und Anforderungen im Folgenden beschrieben werden. Die Aufgaben werden jeweils rechtzeitig zu den entsprechenden Projektphasen bekannt gegeben. Hierzu erfolgen regelmäßige **Aktualisierungen** des vorliegenden Dokuments im ILIAS.

Grundsätzlich sind alle Aufgaben mit terminierten **Projektphasen** und einem angestrebten Ergebnis (Deliverable) verbunden. Die Aufgaben können (und sollen) in **Gruppen** bearbeitet werden. Die Einreichung von **Deliverables** erfolgt pro Gruppe zum Ende der entsprechenden Aufgabenphase. Die Form der Einreichung hängt von den Deliverables ab und ist in den Aufgaben beschrieben.

Anforderungen an die Web Anwendung

Bei der Web Anwendung handelt es sich um eine einfache Social Media Anwendung, bei der Nutzer einen persönlichen (Micro)**Blog** aus Kurznachrichten (**Posts**) erstellen können, den Sie mit anderen Nutzern der Community teilen.

Anwendungsfunktionen

Jeder Nutzer muss sich zunächst **registrieren** und dabei zumindest Name und Passwort hinterlegen. Mit diesen Daten können Nutzer sich **per Login authentisieren**. Nach erfolgreicher Authentifizierung durch das System bleiben die Nutzer dort für einen begrenzten Zeitraum **angemeldet**. Angemeldete Nutzer können sich **per Logout abmelden**. Die Abmeldung erfolgt spätestens nach Ablauf des maximalen Anmeldezeitraums.

Angemeldete Nutzer können **Kurznachrichten** mit einer maximalen Länge von **140 Zeichen schreiben und publizieren**. Publizierte Kurznachrichten erscheinen in zeitlich geordneten Listen (**Timelines**). Die Nutzer können sich verschiedene solcher Timelines anzeigen lassen: eine **globale Timeline** enthält alle Posts des Systems, **persönliche Timelines** zeigen nur die eigenen Posts sowie Posts von Nutzern der Community, denen gefolgt wird.

Die **Posts** auf einer Timeline enthalten mindestens den *Nachrichtentext*, den *Autor* der Nachricht und den *Zeitpunkt* der Publikation. Timelines zeigen dabei immer nur eine begrenzte Anzahl von Posts pro Seite an und bieten die Möglichkeit, zu weiteren oder vorangegangenen Seiten zu wechseln (Paging/Pagination). Alternativ kann auch ein kontinuierliches Nachladen von Posts während des rollens der Timeline erfolgen. Wenn neue Kurznachrichten publiziert werden, während ein Nutzer die Timeline einsieht, erhält er einen **unmittelbaren Hinweis** auf der Benutzeroberfläche.

Angemeldete Nutzer können andere Nutzer der Community auf zwei Arten finden: entweder als Autoren von Posts auf der globalen Timeline oder über eine explizite Suchfunktion (optional: durch Erwähnungen innerhalb des Nachrichtentextes). Die **Suchfunktion** erlaubt die Eingabe des Anfangs eines gesuchten Namens (a* Wildcard Suche) und berücksichtigt keine Groß- oder Kleinschreibung. Das Ergebnis wird als **Nutzerliste** angezeigt. Auch Nutzerlisten verwenden zur Übersicht Paging oder Nachladen.

Ist ein Nutzer gefunden, kann dessen **Profil**, bzw. Timeline eingesehen werden. Es ist auch möglich, dem Nutzer zu folgen. Durch Das **Folgen** eines Nutzers werden dessen Posts auf der persönlichen Timeline angezeigt. Die Follower-Beziehung definiert das soziale Netz der Anwendung. Zum öffentlichen Profil eines Nutzers gehören zwei entsprechende **Follower-Listen**: eine Liste enthält solche Nutzer, denen er selbst folgt, die andere Liste solche Nutzer, die ihm folgen.

Nicht-funktionale Eigenschaften

Die Funktionen der Anwendung sollen unabhängig von der möglichen Existenz **replizierter Serverprozesse** funktionieren (Replikationstransparenz). Wenn im Zuge einer hypothetischen Lastverteilung (die nicht implementiert werden muss) ein Benutzer auf verschiedene Webserverreplikate geleitet wird, dann soll seine Anmeldung erhalten bleiben.

Sofortige Hinweise auf neue Posts sollen angemeldeten Nutzern in nahezu Echtzeit erhalten. Insbesondere soll dies unabhängig von dem Aufruf einer neuen Seite durch den Nutzer geschehen. Der Hinweis soll durch ein partielles Update der aktuellen Seite auf Basis von asynchroner Kommunikation realisiert werden. Solche Hinweise erhält ein Nutzer zudem auch dann, wenn zugrundeliegende Posts auf anderen Webserverreplikaten verfasst wurden.

Die Web Anwendung soll sich sowohl auf dem Desktop als **auch auf mobilen Geräten** korrekt darstellen und komfortabel bedienen lassen.

Verwendetet Techniken

Die Web Anwendung soll als 3-Tier-Client-Server-Architektur realisiert werden.

Serverseitig soll die Implementierung in **Java** erfolgen und das **Spring Framework** verwenden. Die Anwendung soll dem MVC-Muster nach Vorgabe von **Spring Web MVC** folgen und die **Thymeleaf** Template Engine nutzen. Die Datenebene soll durch einen **Redis** Key-Value-Store realisiert werden.

Es wird empfohlen, bei der Realisierung der clientseitigen Benutzerschnittstelle **Bootstrap** zu verwenden und ggf. JavaScript mit **jQuery** zu ergänzen.

Asynchrone Kommunikation soll zwischen Browser und Webserver über **Web Sockets** erfolgen. Diese Technik wird in Spring mit dem Messaging Protokoll **STOMP** und einem einfachen Message Broker ergänzt. Der Nachrichtenaustausch zwischen Webserverreplikaten soll über **Redis Publish/Subscribe Channels** erfolgen.

Aufgabe 1 — Konzeption und Gestaltung

Beginn: 12.10.
Abgabe: 9.11.

Analyse und Entwurf stehen im Mittelpunkt der ersten Aufgabe. Neben einer Anforderungsanalyse sollen hier ein konzeptioneller Entwurf der Anwendungsstruktur und erste gestalterische Lösungen für die Benutzerschnittstelle erstellt werden. Auf dieser Basis sollen auch die Datenstrukturen des NoSQL Key-Value-Stores modelliert werden.

Teilaufgaben

Aufgabe 1.1 Strukturieren Sie die Anforderungen der Web Anwendung, und fassen Sie sie in einem Anforderungsdokument kurz zusammen (es kann auch ein Use Case Diagramm erstellt werden).

Aufgabe 1.2 Fertigen Sie einen Entwurf der Seitennavigation für die Web Anwendung als einfaches Zustandsdiagramm (bzw. als endlichen deterministischen Automat) an. Zustände entsprechen dabei generierten Seiten und Transitionen entsprechen Hyperlinks dazwischen.

Aufgabe 1.3 Importieren Sie das Beispielprojekt aus GitHub und untersuchen Sie dessen Funktion (Führen Sie ggf. auch die **Spring Starter Guides** zur *Bereitstellung von Webinhalten* und zur *Verarbeitung von Formulardaten* durch.) In dieser Umgebung sollen Sie ein **Mockup** der Hauptseite(n) Ihrer Web Anwendung erstellen. Das Mockup soll als lauffähige Spring MVC Anwendung realisiert werden, die allerdings keine Anwendungslogik oder dynamische Daten enthält, sonder auf die Gestaltung der Benutzerschnittstelle fokussiert. Das Mockup sollte Aspekte des Layouts, der Navigation, der Informationsdarstellung und der Interaktion beinhalten.

Aufgabe 1.4 Erstellen Sie einen schematischen Entwurf des technischen Datenmodells auf Basis der Redis Datenstrukturen. Berücksichtigen Sie die Entitäten und deren Beziehungen und beschreiben Sie deren Repräsentation mit Redis Datenstrukturen. Beschreiben Sie auch die Muster der verwendeten Schlüssel.

Deliverable

Die Ergebnisse der ersten Aufgabe sind als **Dokument** mit entsprechenden **Abschnitten** für 1. die Anforderungsanalyse, 2. die Seitennavigation, 3. Mockup Screenshot(s) und 4. das Datenmodell einzureichen. Ein **Deckblatt** soll den **Gruppennamen** und die **Gruppenmitglieder** enthalten. Bitte laden Sie eine Kopie des Dokuments im ILIAS hoch.

Die Einreichung erfolgt pro Gruppe a) per **Upload im Ilias** und b) mit einer kurzen **Vorstellung des Ergebnisses**.

Aufgabe 2 — Seitenbasierte Implementierung

Beginn: 16.11.
Abgabe: 14.12.

In der zweiten Aufgabe geht es um die Implementierung der bisherigen Entwürfe als MVC-basierte Web Anwendung mit Spring. Hierzu gehört die Anbindung des Redis Backends mit einer Repository Komponente. Authentifizierung und die Verwaltung von Sitzungen ist ein weiterer wichtiger Bestandteil. Für die geplanten Seiten der Anwendung sollen dann Controller und Templates erstellt werden. Schließlich ist auch noch die clientseitige Programmierung von Layouts und Benutzerinteraktionen zu realisieren.

Teilaufgaben

Aufgabe 2.1 Erstellen Sie eine Schnittstelle mit Operationen zur Verwaltung Ihrer Datenobjekte (z.B. Nutzer, Posts, Sessions etc.). Implementieren Sie die Schnittstelle als Repository Bean und kapseln sie darin den Zugriff auf Redis.

Aufgabe 2.2 Realisieren Sie die Verwaltung von Sessions durch den Server derart, dass ein Nutzer sich durch Login oder Registrierung anmelden kann und danach für einen definierten Zeitraum am System angemeldet bleibt. Speichern Sie Sessions möglichst in der Datenbank, um Replikationstransparenz zu erreichen.

Aufgabe 2.3 Implementieren Sie die von Ihnen in Aufgabe 1 geplanten Funktionen und Seiten der Anwendung mit entsprechenden Spring MVC Controllern und Thymeleaf Templates. Zur besseren Strukturierung des Systems können Sie Anwendungsfunktionen ggf. durch zusätzliche Service

Komponenten implementieren. Die Seiten sollten Registrierung und Login, Anzeige von Nutzerprofilen und Timelines, Erstellung von Posts, Suche von Nutzern sowie Folgen und 'Entflogen' realisieren. Echtzeitbenachrichtigungen können in dieser Aufgabe noch zurückgestellt werden.

Deliverable

Zur Einreichung der zweiten Aufgabe soll das vorhandene Dokument aus Aufgabe 1 um einen Abschnitt erweitert werden, in dem Sie einen kurzen **Überblick der Implementierung** geben. Bitte laden Sie eine Kopie des erweiterten Dokuments im ILIAS hoch.

Die Einreichung erfolgt pro Gruppe a) per **Upload im Ilias** und b) mit einer kurzen **Vorstellung des Prototyps**.

Aufgabe 3 — Asynchrone Erweiterungen

Beginn: 21.12.
Abgabe: 25.1.

In der letzten Aufgabe soll eine Push-Funktion für Benachrichtigungen bei neuen Posts ergänzt werden. Die serverseitige Anwendung schickt dabei neue Posts über eine WebSockets Verbindung an alle Web Clients, die dem Absender folgen. Auf der Client-Seite wird daraufhin jeweils ein Hinweis für den Benutzer gezeigt.

Diese Funktion soll auch bei Verwendung mehrerer replizierter Serverinstanzen funktionieren. Also auch dann, wenn der Absender seinen Post bei einer anderen Serverinstanz einstellt als derjenigen, mit der ein Client per WebSocket verbunden ist. Hierzu müssen die Serverinstanzen untereinander Update-Nachrichten für neue Posts austauschen, wozu eine Nachrichtenwarteschlange (Queue) verwendet werden soll.

Teilaufgaben

Aufgabe 3.1 Implementieren Sie die Push-Funktionalität zwischen Webserver und -client mithilfe von STOMP-over-WebSocket. Wählen Sie dazu zunächst ein passendes Format/Schema für die STOMP Nachrichten. Richten Sie dann eine oder mehrere Warteschlangen als STOMP-Endpunkte mit passenden Topics ein und verbinden Sie damit Webserver und -client. Über diese Verbindung soll der Server neue Posts an den Client senden, der dann Hinweise auf der Web GUI präsentiert.

Aufgabe 3.2 Ergänzen Sie zum Schluss den Austausch von Updatenachrichten zwischen Serverinstanzen über Redis Publish-Subscribe-Messaging. Wählen Sie auch für diesen Fall ein passendes Nachrichtenformat sowie (eine) geeignete Warteschlange(n). Bedenken Sie beim Entwurf das Kommunikationsverhalten bei großer (ggf. wachsender) Anzahl von Nutzern und Posts. Realisieren Sie auf dieser Basis das Versenden und Empfangen von Updates durch Serverinstanzen und koppeln Sie diesen Mechanismus mit der Client-Benachrichtigung aus Aufgabe 3.1.

Deliverable

Zur Einreichung der dritten Aufgabe soll das vorhandene Dokument aus Aufgabe 2 um einen Abschnitt erweitert werden, in dem Sie einen kurzen **Überblick der Implementierung** geben. Bitte laden Sie eine Kopie des erweiterten Dokuments im ILIAS hoch.

Die Einreichung erfolgt pro Gruppe a) per **Upload im Ilias** und b) mit einer kurzen **Vorstellung des Prototyps**.