Architektur

Systemarchitektur

Das System soll durch eine Multitierarchitektur realisiert werden. Die Architektur setzt sich aus mehreren Schichten zusammen.

Der Client-Tier ist generell für die Realisierung der Benutzerschnittstellen zuständig. Über diese Schnittstellen sollen die Benutzer Zugriff auf die Systemfunktionen haben. Da die Benutzer aus Sicht der MCI unterschiedliche Merkmale, Aufgaben und Anforderungen besitzen müssen mehrere Schnittstellen implementiert werden. Die Zugriffsrechte sind je nach Benutzertyp eingeschränkt. Dementsprechend können Geschäftsobjekte auch nur zum Teil bearbeitet werden. Die mobilen Clients sollen im Gegensatz zu den lokalen Clients einen Teil der Anwendungslogik beinhalten.

Über die Steuerungsschicht oder auch Middleware genannt werden die Anfragen der Clients sowie die Nachrichten des Anwendungssystems zu den Clients verwaltet. Die Middleware dient der Übermittlung von Daten und asynchroner Nachrichten an die Komponenten des Systems.

Das Anwendungssystem oder auch die Logikschicht ist für die Bearbeitung der Geschäftsobjekte zuständig. Automatische Prozesse werden hier ausgeführt. Dabei greift das System auf Informationen der Datenschicht zu.

Die Datenschicht speichert persistent alle Daten, die für die Medikation relevant und notwendig sind.

Komponenten

Die Systemarchitektur muss nun weiter definiert werden. Dazu ist es nötig, die Systemkomponenten im einzelnen zu betrachten und zu spezifizieren. Dafür müssen verschiedene Technologien abgewägt werden und Entscheidungen mit den bisherigen Ergebnissen aus dem MCI-Vorgehen herangezogen werden.

Server

Der Server kann durch verschiedene Technologien realisiert werden. Da wir im Verlauf des Studiums Erfahrungen mit NodeJS gesammelt haben, soll der Server mit NodeJS realisiert werden. Die Einarbeitung in andere Technologien könnte für das Projekt ein weiteres Risiko darstellen. Außerdem wird NodeJS ständig weiter entwickelt und die Technologie ist weit verbreitet. Dadurch ist die Anbindung verschiedener Middleware- und Datenbank-Systeme möglich. NodeJS schränkt uns daher in keiner Weise ein.

Clients

Die ermittelten Anforderungen (N03) zeigen, dass sowohl mobile als auch lokale Clients entwickelt werden müssen.

Als mobiles Endgerät soll ein Tablet eingesetzt werden. Alternativ könnten auch Smartphones verwendet werden. Allerdings bietet ein Tablet ein größeres Display, welches die Darstellung von vielen Informationen erleichtert.

Die meist verbreitetsten Betriebssysteme für mobile Endgeräte sind Googles Android OS und Apples iOS. Die native Programmiersprache von Android ist Java, wohingegen für iOS in Objective-C programmiert wird. Zusätzlich sind Frameworks vorhanden, die es ermöglichen die Apps in Javascript und Html zu programmieren. Die programmierten Anwendungen können dann durch das Framework für beliebige Betriebssysteme exportiert werden.

Eine wesentliche Rahmenbedingung im Projekt ist es, die Anwendungslogik in Java zu realisieren. Da das Team auch Erfahrungen in der Programmierung mit Java und dem Android SDK hat, werden die mobilen Clients für ein androidbasiertes Endgerät programmiert.

Middleware

Nach den Anforderungen soll entschieden werden, welches Middleware-System für das Projekt verwendet werden soll.

Middlewaresysteme

Grundsätzlich lassen sich die Middleware-Technologien in zwei Arten unterscheiden. Zum einen gibt es kommunikationsorientierte Middleware-Technologien, die sich durch entfernte Aufrufe durch z.B. Webservices oder nachrichtenorientierte Middlewares durch komplexe Warteschlangen klassifizieren. Zum Anderen gibt es anwendungsorientierte Middlware, die sich dadurch auszeichnen, dass sie die kommunikationsorientierte Middleware um eine Laufzeitumgebung erweitern[1].

Aus den Anforderungen (F21) ist ersichtlich, dass vor allem Nachrichten zwischen den Komponenten ausgetauscht werden müssen. Ein wesentlicher Aspekt ist, dass das MDKS asynchrone Nachrichten verteilen muss. Die Nachrichten richten sich dabei an verschiedene Empfänger, die je nach Inhalt der Nachricht adressiert werden sollen. Entfernte Aufrufe oder eine Laufzeitumgebung innerhalb der Middleware sind nicht nötig. Daher soll eine nachrichtenorientierte Middleware gewählt werden, die ein Publish-Subscribe-System unterstützt. Die Nachrichten sollen durch einen Broker verwaltet und an die Message Queues weitergeleitet werden, die wiederum die Nachrichten an die jeweiligen Empfänger weiterleiten. Die Nachrichten werden dann übermittelt, wenn die Empfänger erreichbar sind. Daher gehen die Nachrichten bei Verbindungsabbrüche mit den Clients nicht verloren. Sender und Empfänger können dadurch räumlich wie auch zeitlich entkoppelt werden.

Es gibt eine Reihe von Technologien, die ein Publish-Subscribe ermöglichen. Drei der Technologien sollen im folgenden im Hinblick auf das zu entwickelnde System abgewägt werden.

---

1 https://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCAQFjAAahUKEwj8kbfKi47JAhUGliwKHQHHB0M&url=http%3A%2F%2Fwww4.in.tum.de%2Flehre%2Fseminare%2Fhs%2FWS0506%2Fmvs%2Ffiles%2FAusarbeitung-Krivoborodov.doc&usg=AFQjCNFv-bzMKHJ_YOY4dN99MuntIrCFzg&sig2=RlCoqwiv6p1xnWiwFxCu1w
Sichtung: 2.11.2015, 20:55

Faye vs GCM vs RabbitMQ

Faye ist ein Modul von NodeJS. Nachrichten können durch ein Pub-Sub-System asynchron per http-Protokoll übermittelt werden. Allerdings lassen sich die Message Queues nicht selbst verwalten und insgesamt sind die Möglichkeiten für eine individuelle Einrichtung der Middlware eingeschränkt. Ein weiterer negativer Aspekt ist, dass es keinen offiziellen Client für Java gibt, der die Nachrichten aus der Queue entgegen nimmt. Dies stellt für uns ein Problem dar, da die Clients in Java programmiert werden sollen.

Google Messaging Cloud ist eine weitere Lösung für ein Publish-Subscribe-System. Die Daten werden in einer Message Queue, die sich in einer Cloud im Web befindet, aufbewahrt und von dort an die clients verschickt. Service ist einfach nutzbar und besonders praktisch für Androidapplikationen. Es ist aber zu beachten, dass die Daten aus dem MDKS dadurch an einen Drittanbieter verschickt werden müssen. Da die patientenbezogene Daten einem strengen Datenschutz unterstehen (Anforderungen N05, Domänenrecherche), ist die Weitergabe der Daten an dritte Personen kritisch zu betrachten.

Eine weitere Technologie ist RabbitMQ. RabbitMQ wird als eigenständiger Server auf einem System installiert und fungiert dabei als nachrichtenorientierte Middleware. Die Message Queues können manuell oder über die Clients angelegt und konfiguriert werden. Das Pub-Sub-System wird durch Exchanges und Routingkeys realisiert, wobei die Exchanges die Nachrichten an Queues weiterleiten und die Routingkeys angeben, welche Queue adressiert werden soll. RabbitMQ bietet für sämtliche Programmiersprachen und Clients eine Anbindung. Darunter sind auch Java, Javascript und NodeJS. Die Nachrichten werden über das Protokoll “AMQP” übermittelt.

Aufgrund der beschriebenen Aspekte von, fällt die Wahl auf RabbitMQ. Es ist aber zu beachten, dass keine Erfahrungen mit dieser Technologie im Projektteam vorhanden sind, wodurch ein Risiko entsteht. Daher sollte RabbitMQ und ein

Publish-Subscribe mit dieser Technologie in einem “Proof-of-Concepts” getestet werden.

→ siehe Risiken
→ siehe POC 3 Synchroner Datenaustausch zwischen Android und NodeJS mit Speicherung in MySQL

Datenbank

Eine der Anforderungen ist es, Daten persistent zu speichern, wodurch es notwendig ist ein DBMS zu implementieren. Aus dem angeeigneten Wissen des Studiums sind uns zwei verschiedene Datenbanktypen bekannt. Im folgenden sollte deshalb abgewägt werden welcher Typ für das System im angewandten Nutzungskontext besser geeignet ist.

relational vs dokumentenorientiert

Relationale Datenbanken organisieren die Daten in Tabellen und Feldern, die in Beziehung miteinander stehen. Daher entstehen Referenzen in den Tabellen untereinander. Eine relationale Datenbank folgt einem festgelegten Standard und die Abfragesprache ist einheitlich.

Dokumentenorientierte Datenbanken speichern die Daten in Containerobjekten mit beliebigen Feldern. Dieses Modell fokussiert sich vor allem auf Performance und Verfügbarkeit.

Aus der Domänenrecherche ist bereits ersichtlich, dass im zu entwickelnden System besonders viele Abhängigkeiten in den Datenmodellen bestehen (Datenstrukturen). So setzt sich eine Patientenakte aus Verordnungen, Patientendaten, Indikationen und Untersuchungen zusammen. Diese Abhängigkeiten lassen sich idealerweise durch Beziehungen in relationalen Datenbank realisieren. Aus diesem Grunde fällt die Entscheidung auf ein relationales DBMS. Die Realisierung eignet sich vor allem mit MySQL, da die Implementation einfach ist und Anbindungen in sämtlichen Programmiersprachen durch die weite Verbreitung von MySQL möglich sind.

→ siehe Datenstrukturen

→ siehe ER-Modell

Es gilt zu testen, inwiefern NodeJS eine Implementierung von MySQL ermöglicht. Daher ist es notwendig ein “Proof of Concepts” für diese Technologie einzuplanen.

→ siehe Risiken

→ siehe POC 1 Publish-Subscribe mit RabbitMQ

Webservice

Um Fehler bei der Medikation zu vermeiden, sind zusätzliche Informationen zu Medikamenten notwendig. Es muss festgestellt werden, ob verschriebene Medikamente kompatibel mit den bisherigen gestellten Verordnungen sind (Anforderungen F32). Des Weiteren muss geprüft werden, welche Wirkstoffe oder Speisen die Wirkung der Medikamente aufheben (Anforderungen F12). Diese Informationen werden in der Regel von Fachkräften der Pharmazie auf Grundlage von Nebenwirkungsprüfungen und Studien erarbeitet und sind nicht frei im Internet zugänglich. Deshalb muss ein externer Webservice in das MDKS eingebunden werden, wodurch es möglich ist, Wechselwirkungsdaten und Daten über Risikofaktoren zu bestimmten Medikamenten zu erhalten. Die Daten können dann im NodeJS-Server für eine Medikationskontrolle verwendet werden.

Die Firma “ifap GmbH” bietet einen solchen Webservice an. Allerdings wurde uns ein Testzugang zu diesem verwehrt. Daher ist es notwendig einen Webservice zu simulieren, um die Anwendungslogik der Medikationskontrolle zu realisieren.

Insbesondere ist bei der Anbindung eines externen Webservice der Datenschutz zu beachten. Deshalb müssen die Anfragen auf den Webservice ohne die Weitergabe persönlicher Daten der Patienten realisiert werden und nur Medikamentennamen

enthalten. Dadurch kann der Datenaustausch mit einem Drittanbieter mit Beachtung des Datenschutzes realisiert werden.

Synchrone und asynchrone Nachrichten

Innerhalb des MDKS müssen die Daten asynchron und synchron ausgetauscht werden. Nach den ermittelten Anforderungen müssen Erinnerungen und Benachrichtigungen über Änderungen in Form von Nachrichten an die Empfänger-Clients verschickt werden (Anforderungen F21, F24). Für solche Anwendungsfälle soll ein Publish-Subscribe-System mit RabbitMQ genutzt werden. Die Nachrichten werden dadurch asynchron per AMQP-Protokoll verschickt.

Neben einer asynchronen Kommunikation sollen Daten auch synchron ausgetauscht werden. Ein Anwendungsfall ist z.B. ein Datenabruf von Verordnungen für die Erstellung von Medikationsplänen (Anforderungen F20). Da bereits ein asynchrones Kommunikations-Paradigma mit RabbitMQ gewählt worden ist, soll die synchrone Kommunikation ebenfalls über RabbitMQ und Warteschlangen stattfinden. Daher muss von der Entscheidung den synchronen Datenaustausch mit dem http-Protokoll zu realisieren abgesehen werden. Gründe dafür sind vor allem eine Realisierung einer minimalistischen Architektur. Dafür müssen entsprechende Warteschlangen eingerichtet werden, die die Anfragen an den Server weiterleiten. Der Anwendungsserver soll die Anfragen erhalten und die gewünschten Daten aus der Datenbank holen und an die jeweilige Queue publishen, an die der Client als Consumer verbunden ist. Der Client wartet dabei nach dem Senden der Anfrage auf die Antwort.

Nach weiteren Betrachtungen wurde ersichtlich, dass ein RPC nötig ist, um über RabbitMQ den synchronen Datenaustausch zu realisieren.

→ siehe WBA-Modellierung

Anwendungslogik

Folgend soll die Anwendungslogik innerhalb der Komponenten konkret beschrieben werden. Dabei werden zunächst einzelne Komponenten betrachtet.

Android-Client

Die Primäre Aufgabe des Android-Clients soll es sein, Medikationspläne mithilfe der Daten aus den Patientenakten zu erstellen und eingehende Nachrichten der Middleware sowie ausgehende Nachrichten zur Middleware zu Verwalten und zu verarbeiten.

Für die Erstellung der Medikationspläne sind die Verordnungsdaten aus der Patientenakte notwendig. Der Medikationsplan eines Patienten unterscheidet sich vom Medikationsplan eines Krankenpflegers in der Zusammensetzung der benötigten Daten. Für den einzelnen Patienten sind nur die Daten der jeweiligen Patientenakte notwendig wohingegen ein Krankenpfleger für alle Patienten einer Station zuständig ist und daher sämtliche Verordnungsdaten aller Patienten der Station benötigt, um seinen Aufgaben nachzugehen. Der Medikationsplan muss chronologisch in Abhängigkeit von dem Applikationszeitpunkt der Verordnungen sortiert sein. Außerdem muss es für das Pflegepersonal möglich sein mit dem auf dem Device dargestellten Plan zu interagieren, um Verabreichungen zu dokumentieren (Anforderungen F25). Die Dokumentation der Verabreichungen müssen persistent gespeichert werden.

Die Speicherung findet aber nicht auf dem Device statt, sondern auf der zentralen Datenbank des MDKS (siehe ER-Modell). Dadurch müssen Nachrichten zur Middleware geschickt werden, die dann vom NodeJS-Server abgegriffen werden. Da der Server die Daten in die Datenbank schreibt, müssen die Nachrichten zwingend die notwendigen Daten zur Speicherung der Verabreichung enthalten (siehe Datenstrukturen).

Um Änderungen an Verordnungen für das Personal sowie für den Patienten kenntlich gemacht werden können muss ein Hintergrundservice in der

Clientanwendung integriert werden, der eingehende Nachrichten der Middleware als eine Systembenachrichtigung auf der Statusleiste des Device ausgibt. Abhängig von dem Inhalt der Nachricht müssen die Medikationspläne angepasst werden. Daher soll der Hintergrundservice auch das Aktualisieren des Medikationsplans triggern, damit keine veralteten Daten auf dem Device angezeigt werden. Da die Medikationspläne individuell für den jeweiligen Benutzer erstellt werden müssen auf dem Device individuelle Queues angelegt und an die Middlware angebunden werden.

→ siehe WBA-Modellierung
→ siehe POC 2 Notification-Serviceauf einem Android-Client

NodeJS-Server

Die Aufgabe des Servers ist es zum einen, gestellte Verordnungen auf Wechselwirkungen und Risiken zu prüfen bevor diese in der Datenbank abgelegt werden. Dabei werden die Daten aus dem Webservice verwendet, um sicherzustellen, dass keine Risiken oder Wechselwirkungen innerhalb der Medikation eines Patienten auftreten. Außerdem wird die gestellte Verordnung mit der Medikamentenanamnese des Patienten verglichen. Der Server fungiert an dieser Stelle als überwachende Instanz. Wird also eine Verordnung vom Arzt über den Javascript-Client gestellt, wird zunächst eine Nachricht mit den Verordnungsdaten an den Server versendet, der die Medikationsprüfung vornimmt und anschließend eine Antwort mit dem Status der Verordnung und zusätzlichen Informationen über Risiken und Wechselwirkungen sowie Vorschlägen zu alternativen Medikamenten an den Javascript-Client sendet.

Zum Anderen muss der Server alle Datenanfragen sowie Datenspeicherungen verwalten. Da die synchrone Kommunikation nicht über Http-Get und -Post stattfindet muss eine Anwendungslogik die Anfragen verarbeiten. Die Nachrichten der Middleware müssen daher eine eindeutige Struktur aufweisen und auf verschiedene Queues abgelegt werden. Des Weiteren muss eine Rollenverteilung und die damit

einhergehende Berechtigung auf Datenzugriffe verwaltet werden, um Verletzungen des Datenschutzes zu eliminieren.

Eine weitere Aufgabe des Servers ist es, Benachrichtigungen an die Android-Clients zu verschicken. Dies wird über das Versenden von Nachrichten an die Middleware realisiert. Einerseits sollen Erinnerungen zu Verabreichungen versendet werden. Dafür muss die Systemzeit mit den Applikationszeiten der Verordnungen verglichen werden. Zum Anderen müssen Nachrichten über Änderungen an Verordnungen verschickt werden. Diese Benachrichtigungen werden verschickt, wenn ein Arzt über den Javascript-Client eine Verordnung bearbeitet hat.

→ siehe WBA-Modellierung

Ressourcen

Um ein umfassendes Medikationsystem zu entwickeln, ist es notwendig einige Informationen zur Medikation eines Patienten als Ressource zu modellieren. Im Rahmen des Projektes soll der Fokus allerdings auf die Ressourcen des Patienten und der Verordnungen fallen und den dadurch aggregierten Medikationsplan. Die Ressourcen sollen als Grundlage für die Anwendungslogik und für die Präsentation der Informationen eines Patienten dienen.

→ siehe WBA-Modellierung

Topics (Routing Keys)

Da eine nachrichtenorientierte Middleware im MDKS eingesetzt wird, müssen Queues verwendet werden, die bestimmte Nachrichten an die Consumer weiterleiten. Durch die Topics soll die synchrone und die asynchrone Kommunikation zwischen den Komponenten verwaltet werden. Die Nachrichten auf einen Exchange der RabbitMQ-Middleware werden durch die Angabe eines Topics (Routing-Key) auf die jeweiligen Queues verteilt.

Da ein Krankenhaus auf Stationen aufgeteilt ist, ist es sinnvoll ein Topic zu verwenden, dass sich auf eine bestimmte Station bezieht. Des Weiteren müssen Topics für alle Patienten erstellt werden, damit jeder Patient nur seine persönlichen medikationsbezogenen Nachrichten erhalten kann. Da der Fokus in diesem Projekt auf den Medikationsplan fällt, wird zunächst davon abgesehen weitere Topics wie Verordnungen zu modellieren.

Damit die synchrone Kommunikation ebenfalls mit RabbitMQ umgesetzt werden kann, müssen Topics modelliert werden, die GET- und POST-Requests als Nachrichten verwalten können. Zu dem müssen individuelle Queues für die Clients angelegt werden, die die Daten nach einer Anfrage entgegen nehmen.

→ siehe WBA-Modellierung