Dokumentation EventDoku

[Martin Münch | Stefan Dietrich]

1. Projekt EventDoku

EventDoku ist eine App, die es ermöglicht, Veranstaltungen mit Hilfe eines Smartphones zu dokumentieren. Diese App ist für die *Offene Jugendarbeit Dornbirn* (OJAD) entwicket worden, um die Dokumentation der stattfindenden Veranstaltungen zu vereinfachen.
Mit Hilfe dieser App ist es möglich, dass den Mitarbeiterern, die Veranstaltungen übersichtlich angezeigt werden. Diese Veranstaltungen können nur von der verantwortlichen Person dokumentiert werden. Dabei können besucherspezifische Daten erfasst werden. Es ist ebenfalls möglich, ehrenamtliche HelferInnen und MitarbeiterInnen zu erfassen, die bei dieser Veranstaltung gearbeitet haben.
Die App wurde so aufgebaut, dass die benutzende Person einen Überblick über die noch zu dokumentierenden und die bereits dokumentierten Veranstaltungen erhält. Es wurde darauf geachtet, die App möglichst einfach und benutzerfreundlich zu gestalten. Dabei hat man sich an den Android-Design-Guidlines orientiert.

2. Android App

2.1. Systemanforderung

Betriebssystem

EventDoku wurde so entwickelt, dass Geräte ab dem Android Betriebssystem 4.0 “Ice Cream Sandwich” die App verwenden können. Diese Entscheidung wurde in Absprache mit der OJAD getroffen, da aktuell nur noch zwei Geräte mit einem älteren Betriebssystem in Gebrauch sind. Diese beiden Geräte werden jedoch in absehbarer Zeit ersetzt und müssen daher nicht unterstützt werden. Des Weiteren ist das Design der Anwendung, an die aktuellen Android-Guidelines angelehnt.

Internetverbindung

Beim Start der Anwendung müssen sich die Benutzer mit ihrem Passwort beim Webserver anmelden, dies ist notwendig um aktuelle Daten zu erhalten und die App benutzen zu können. Desweiteren wird für die Verbindung zu dem Webserver die Internet-Permission des Android Gerätes benötigt.. Diese Permission erlaubt es, eine Verbindung sowohl über das WLan als auch über das mobile Internet herzustellen. Die Internetverbindung wird zum späteren Zeitpunkt noch verwendet, um dokumentierte Events zum Webserver zu senden damit dieser die Veranstaltung in der Datenbank ablegen kann.

2.2. Softwarearchitektur

Aufbau

Überblick Schichtenarchitektur

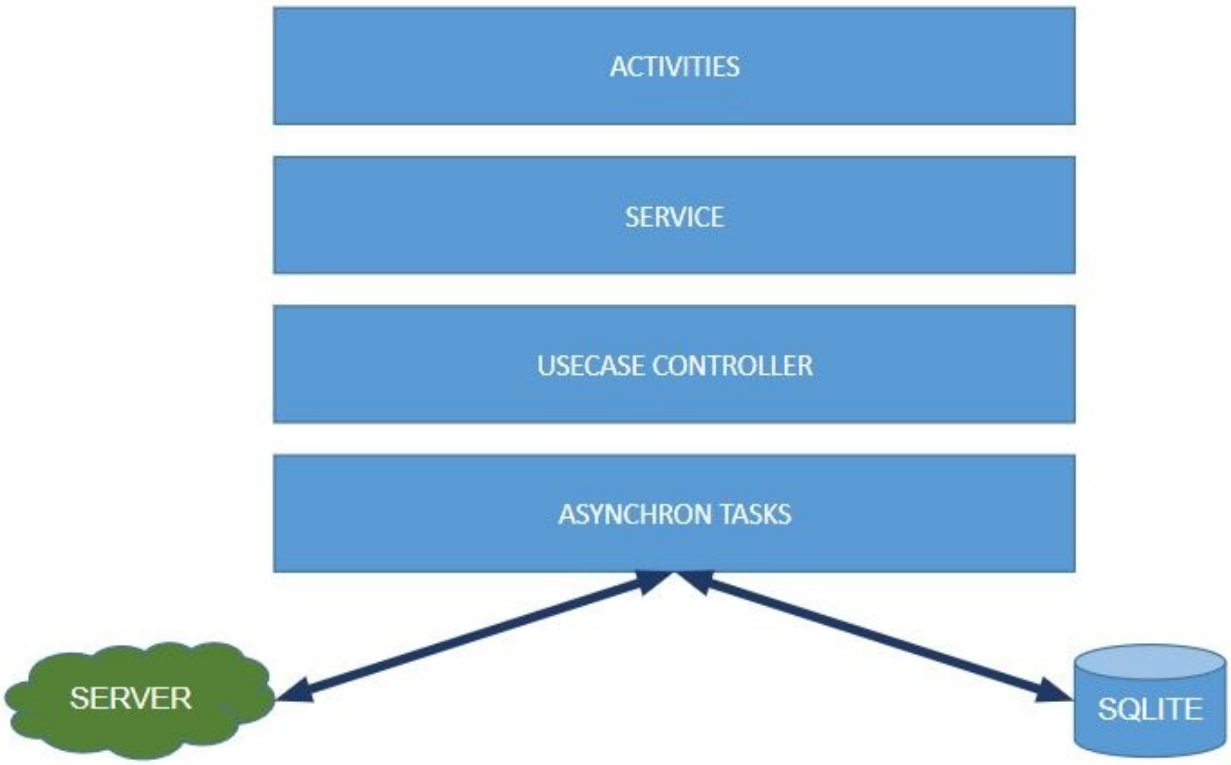


Abbildung1: Schichten-Modell

- **Activities mit Fragmenttabs**

Die Activities und die - als Tabs sichtliche - Fragments bilden die graphische Benutzeroberfläche der Android-Applikation. In dieser Schicht wird die Kommunikation mit den Anwendern realisiert und die verschiedenen Darstellungen implementiert. In der Applikation wird jeder Use Case durch eine Activity abgebildet, welche verschiedenen Fragments beinhalten. Diese Trennung erleichtert eine Erweiterung der App durch neue UseCases und stellt die Informationen den Benutzern übersichtlich da.

- **Service**

In EventDoku gibt es die zentrale Service-Klasse: *EventDokuService*. Dieser Android-Service wird von allen Activities als zentrale Anlaufstelle zur weiteren Bearbeitung von Daten verwendet. Durch diese einzelne Service-Klasse entsteht eine klare Trennung zwischen der Präsentations-Schicht (Activities/Fragments) und der Logikschicht mit den Use Case-Controllern.

- **UseCaseController**

Die im obigen Modell als Use Case-Controller bezeichnete Schicht, stellt die Logikschicht der Android-App dar. Die von der Service-Schicht erhaltenen Daten werden hier weiterverarbeitet. Die einzelnen Use Cases (Controller) werden über die *Controller-Factory* gestartet. Die Use Case-Controller-Schicht ist somit bis auf die Schnittstelle komplett von der oberen Schicht getrennt. Die *Controller-Factory* leitet den Aufruf des Services[1] an den zuständigen Controller weiter und ruft - in diesem - die Methode *startUseCase* auf.
Die einzelnen UseCase-Controller erzeugen und starten selbstständig die zur Abarbeitung notwendigen Asynchronen-Tasks.

- **Asynchrone-Tasks**

Damit die Benutzeroberfläche während zeitintensiven Arbeitsvorgängen nicht blockiert ist, werden diese Arbeitsschritte in Asynchrone-Tasks ausgelagert. Die Initialisierung sowie der Aufruf der Asynchronen-Tasks werden durch die Use Case-Controller definiert.
Zu den Aufgaben der Asynchron-Tasks-Schicht zählen unter anderem die Kommunikation mit dem Webserver als auch das anstoßen von Lese- und Schreiboperationen der SQLite-Datenbank.

- **Server & SQLite**

Die Kommunikation mit dem Webserer und der SQLite Datenbak werden im Punkt *Datenverarbeitung* genauer erläutert.

UseCases

Insgesamt enthält die Android-App *EventDoku* drei UseCases, welche im folgenden kurz erläutert werden.

- Login

Der Login-Use Case ist für die Authentifizierung Anwenders am Webserver zuständig. Hierfür gibt der/die BenutzerIn seinen/ihren Benutzername und sein/ihr Passwort in die Eingabemaske ein und bestätigt den Vorgang mit Login.
Um die sicherheitsrelevanten Daten nicht unverschlüsselt auf den Webserver zu übertragen, wird zuerst eine Zufallszahl von diesem angefordert. Das in die Eingabemaske eingetragene Passwort wird mittels des MD5-Algorithmus gehasht und an diesen Hash die vom Server erhaltene Zufallszahl angehängt. Diese Zeichenkette wird anschließend noch einmal gehasht und mit dem Benutzernamen sowie der - zuvor erhaltenen - Zufallszahl an den Server übertragen. Der Server überprüft diese Daten mit den in der Datenbank enthaltenen Benutzerdaten. Stimmen das Passwort[2] aus der Datenbank[3] mit dem übertragenen Passwort überein, so wird dem Gerät ein Token mit einer Gültigkeit von vier Stunden übergeben. Mit

[1] Dabei handelt es sich um den benötigten Use Case.
[2] In Verbindung mit der zuvor übermittelten Zufallszahl.
[3] Das Passwort wird in gehashtem Zustand in der Datenbank gespeichert.

diesem Token wird der/die BenutzerIn identifiziert und bekommt Zugriff auf die weiteren Methoden des Webservices und somit der Android-Applikation.

- Dashboard

Bei diesem UseCase werden die aktuellen Veranstaltungen sowie für die Benutzung der App relevanten Daten des/der Benutzers/Benutzerin auf das Gerät geladen und in der SQLite-Datenbank aktualisiert. Anschließend wird die Übersichtsansicht (Dashboard) mit den offenen (nicht dokumentierten) - und den dokumentierten Veranstaltungen erstellt. In dieser Ansicht werden den Benutzern, Informationen wie der Name und das Startdatum der Veranstaltung angezeigt.

- Veranstaltung dokumentieren

In diesem UseCase geht es darum eine in der Übersichtsansicht (Dashboard) ausgewählte Veranstaltung zu dokumentieren. Hierfür können in verschiedenen Tabs (Fragments) unterschiedliche Informationen (z.B. Besucherdaten) zu der Veranstaltung hinzugefügt oder abgeändert werden.
Die bearbeite Veranstaltung kann durch einen Klick auf den *absenden*-Button (im Optionsmenü) gespeichert werden. Die geänderten Informationen der Veranstaltungen werden erst nach dieser Interaktion in die SQLite-Datenbank geschrieben und anschließend an den Webserver gesendet. Dabei wird die geänderte Veranstaltung als *dirty* markiert, was bedeutet, dass dieses Event noch nicht auf dem Webserver gespeichert ist. Bei Erhalt der positiven Speicherbestätigung - von Seiten des Servers - wird die *dirty*-Markierung entfernt. Die Markierung - der geänderten Veranstaltung - verhindert das Überschreiben der geänderten Daten durch nachgeladene “alte” Daten aus der Webserver-Datenbank.

Kommunikation innerhalb der App

In der folgenden Abbildung wird die Kommunikation der einzelnenen Komponenten von der Präsentationsschicht bis hin zu den Asynchronen-Tasks anhand des Use Case: “Login" dargestellt.

In dem *Fragment* werden die erforderlichen Daten eingetragen und durch das Klicken auf den Login-Button ein *Event* ausgelöst. Dabei verwendet das *Fragment* eine *Callback*-Methode um die gesammelten Daten an die verantwortliche *Activity* weiterzugeben und damit den Start des Use Case initialisiert.
Darauf hin wird Innerhalb der - in der Activity implementierten - *Callback*-Methode der Android-*Service* aufgerufen der die Anfrage an die *Controller Facade* weiterleitet. Die *ControllerFacade* greift darauf hin auf den Controller (implementierte Logik des Use Cases) zu, welcher die zwei benötigten *Asynchronen-Tasks* initialisiert. Der erste Task dient dazu, die aktuelle Zufallszahl des Webserver mithilfe der Methode: “*GetRandomNumber”* anzufordern. Diese RandomNumber wird zum hashen des Passworts benötigt (Siehe Abschnitt: Use Case Login).

Anschließend wird von dem *LoginController* der zweite Task (*GetTokenTask)* erstellt und ausgeführt.

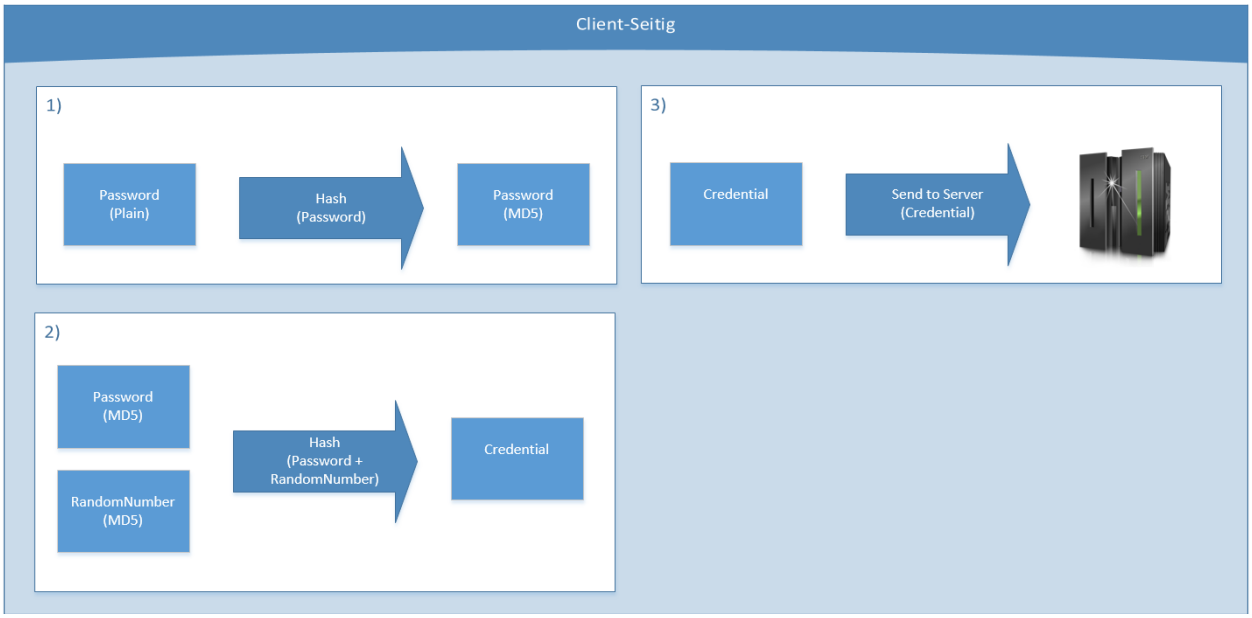


Abbildung2: Use Case: “Login” (Client-Seitig)

Dieser Task sendet die Benutzerdaten an den Webserver und authentisiert sicher bei diesem. Sind diese Daten korrekt, erfolgt die Authentifizierung durch den Webserver und ein Token wird an das Gerät gesendet. (siehe Abb.: 3)

Wenn beide aufgerufenen Tasks erfolgreich ausgeführt wurden, erfolgt der *Callback* an die *Activity.*

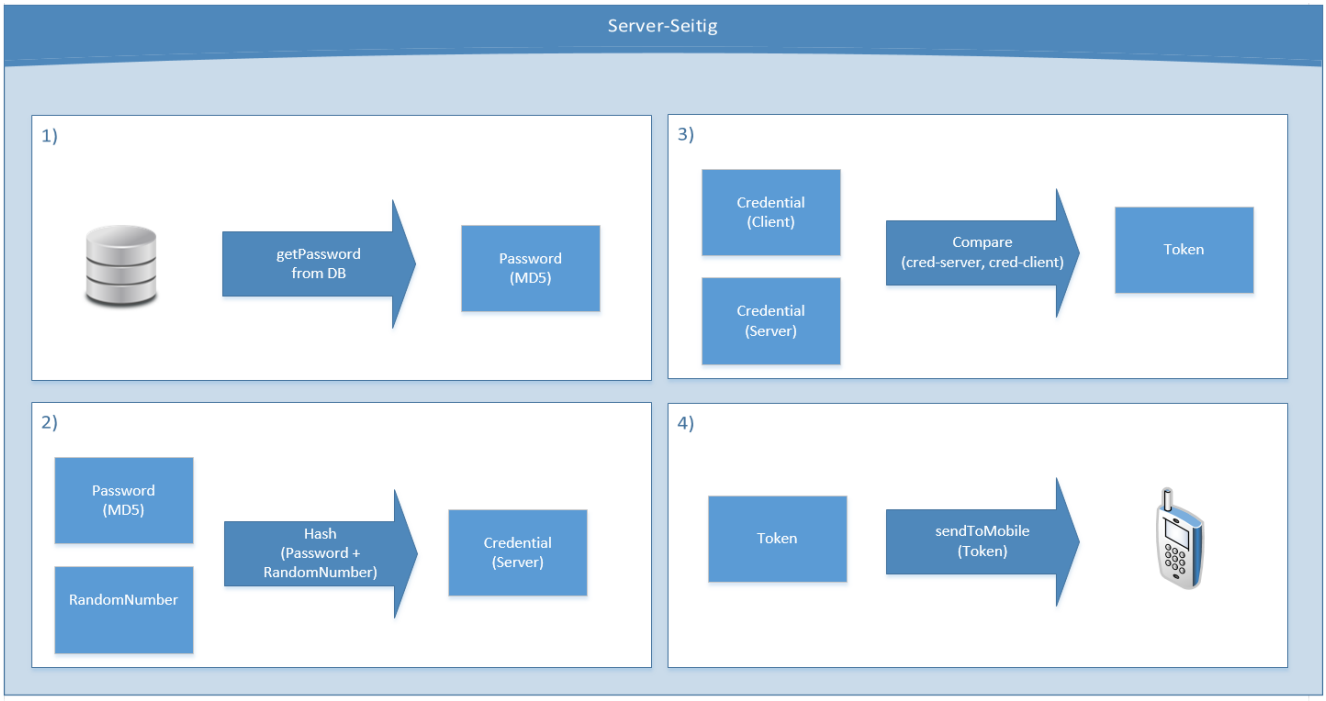


Abbildung3: Use Case: “Login” (Server-Seitig):

Die *LoginActivity* startet nun die nächste *Activity* und ein neuer Use Case beginnt.

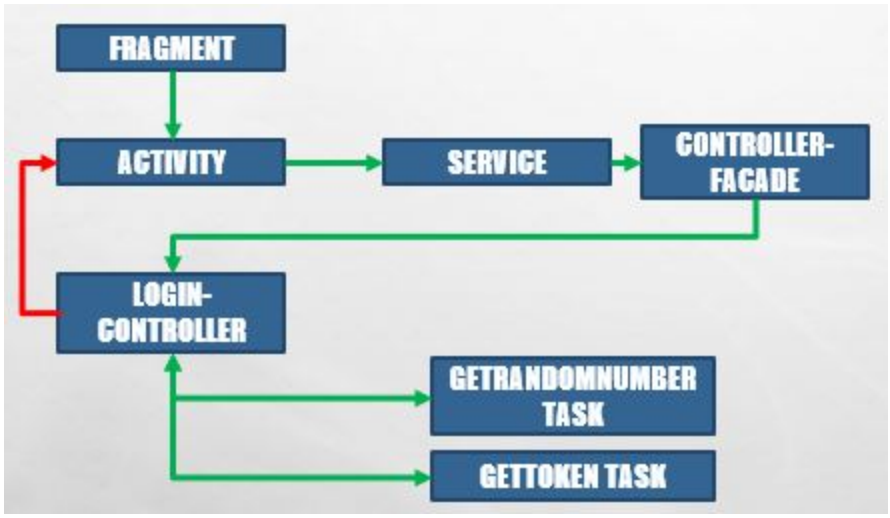


Abbildung4: Use Case-Ablauf

Die in der oberen Abbildung gezeigte graphische Veranschaulichung der Kommunikation zwischen den Komponenten wurde in den restlichen Use Cases nach dem gleichen Prinzip realisiert.

Datenverarbeitung

EventDoku bezieht seine Daten von einem Webserver, des weiteren werden auch Daten auf dem Endgerät gespeichert um lange Ladezeiten während der Anwendung zu vermeiden. Dabei muss sichergestellt werden, dass Daten, die nicht korrekt an den Webserver wurden, gehalten werden. Um das Überschreiben von nicht übermitteltem Daten zu vermeiden, wurde eine SQLite Datenbank implementiert. Im folgenden wird diese beschrieben.

SQLite Datenbank

Die SQLite Datenbank ist für die Persistenz der Daten auf dem verwendeten Gerät zuständig. Außerdem werden beim Start der App alle benötigten Daten vom Webserver abgerufen und in die SQLite Datenbank geschrieben.

Die SQLite Datenbank ist eine vereinfachte Version der Webserver-Datenbak, wobei nicht alle Tabellen übernommen wurden und nur die notwendigsten Informationen in der Datenbank enthalten sind. Die Datenbank wird beim ersten Start der App erstellt und erst bei der Deinstallation der App wieder von dem Gerät gelöscht.

Nach dem erfolgreichen Login der Anwender werden die für die Benutzung der App relevanten Daten vom Webserver angefordert und in die Datenbank gespeichert. Alle vom Webserver erhaltenen Daten, bis auf die Veranstaltungen werden in die SQLite Datenbank übernommen und ersetzen die vorhanden Datensätze. Vor dem einfügen der Veranstaltungen wird überprüft, ob die Informationen in der SQLite-Datenbank aktueller sind als der Stand des Webservers.[4] Ist dies der Fall, wird der Datensatz nicht mit den Daten des Webservers überschrieben.

Der zweite Prozess zum aktualisieren der Daten, erfolgt nach dem speichern einer dokumentierten Veranstaltung. Hierbei werden die aktuellen Daten in die SQLite-Datenbank eingetragen und der Datensatz mit *dirty* markiert. Anschließen werden die Daten der geänderten Veranstaltung an den Webserver gesendet. Bestätigt dieser den erfolgreichen Speichervorgang der Veranstaltungsdaten in der Weberver-Datenbank, wird die Markierung der Veranstaltung in der SQLite-Datenbank entfernt.

Webserver

Die in der App verwendeten Daten sind zentral auf einem Webserver abgelegt, welcher in den nachfolgenden Kapitel genauer erläutert wird. Um auf diesen Webserver zugreifen zu können, wurde in der Klasse *RuntimeCache* die zugehörige IP hinterlegt. Diese IP wurde zentral

[4] Prüft ob die Veranstaltung als *dirty* markiert ist.

abgespeichert, damit bei einem Wechsel der IP-Adresse diese nur an einer Stelle ausgetauscht werden muss.
Die Kommunikation mit dem Webserver erfolgt über Http Post Methoden, welche als Pararmeter einen JSON-String übetragen können. Die daraufhin zurückgesendeten Daten des Webserver werden als JSON-Objekte übertragen und werden in der Klasse *JsonResponseDeserializer* in die erforderlichen Objekte umgewandelt.
Der Webserver wird im Grunde genommen an zwei Zeitpunkten bei der Benutzung der App aufgerufen. Zum einen, werden beim Login die Daten des/der Benutzers/Benutzerin auf den Webserver übertragen und anschließend die Benutzerrelevanten Daten von Veranstaltungen auf das Gerät gesendet. Zum Anderen, wird der Webserver beim speichern von dokumentierten Veranstaltungen aufgerufen und sendet anschließend eine Statusmeldung, ob der Speichervorgang erfolgreich abgeschlossen wurde.

3. Webserver

Bei dem Webserver der *EventDoku* App handelt es sich um einen *Restful* Webservice. Dies ist notwendig, da sich gleichzeitig mehrere Clients am Server anmelden können, dieser aber Zustandslos ist und der Client sich nicht abmelden muss. Dies hat den Vorteil, dass Benutzer sich von verschiedenen Geräten anmelden können, jedoch nur das letzte Gerät den aktuell gültigen Token kennt und dadurch mit dem Webservice kommunizieren kann.
Zur Übertragung von Daten wurde JSON verwendet, da dieses Datenformat simple aufgebaut ist und von allen bei uns verwendeten Geräten unterstützt wird.

3.1. SQL-Datenbank

Der Webserver greift über verschiedene Schichten auf eine SQL-Datenbank zu. Die Daten können in die Kategorien: Stammdaten und Erfassungs-Daten aufgeteilt werden. Die Stammdaten (wie Veranstaltungsorte, Länder, usw.), sind festgelegte nicht änderbare Daten[5], die von den Administratoren in die SQL-Datenbank eingefügt werden. Die in der App Erfassungs-Daten wie die Veranstaltungen, Besucher-Daten, eingeteilte Mitarbeiter usw., sind so ausgelegt, dass diese nicht durch mehrere Anwender gleichtzeitig veränderbar sind. Diese Daten können nur durch den/die verantwortliche/n MitarbeiterIn verändert werden. Dadurch ist sichergestellt das die Daten nicht gleichzeitig von verschiedenen Anwendern überschrieben werden können.

Um die Datenbank auf dem aktuellen Stand zu halten, ist geplant die Veranstaltungs-, Mitarbeiter-, und Ehrenamtlichendaten von einem Microsoft Exchange-Server automatisch in die SQL-Datenbank zu spielen.

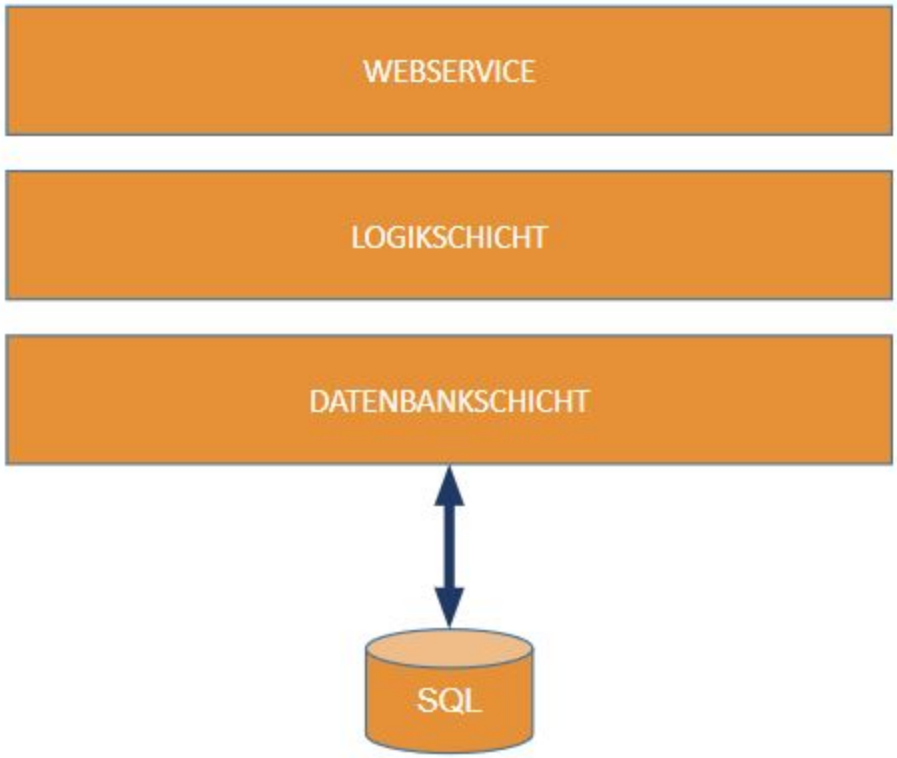
[5] Diese Aussage bezieht sich auf den Webservice und nicht die Datenbank per se.

Die aktuellen Veranstaltungen werden bei der OJAD auf einem Microsoft Exchange Server gespeichert und können auf diesem von anderen Programmen wie Microsoft Outlook aufgerufen und beabeitet werden. Durch die Verbindung der SQL-Datenbank mit dem Microsoft Exchange Server ist die Aktualität der Daten gewährleistet.
Dieser Updatevorgang in die SQL-Datenbank ist zum aktuellen Zeitpunkt noch nicht realisiert.

3.2. Softwarearchitektur Webservice

Schichtenarchitektur EvenDokuBackend

Erläuterung der verschiedenen Schichten



- **Webservice**

Die Webservice-Schicht im *EventDokuBackend*-Projekt besteht aus einem Interface, welches die bereitstehenden Methoden und die benötigten URL's (URL-Pattern) definiert. Über dieses Interface wird der Webservice angesprochen, worauf hin dieser die Methodenaufrufen an die Klasse *SupportedOperations* weiterleitet. Diese Klasse implementiert die Methoden des Interfaces und ist für die Weiterleitung der ankommenden Informationen und das senden von Daten zuständig.
Wie zuvor erwähnt, werden die Daten als JSON-String an den Webservice übermittelt. Um aus dieser Zeichenkette die benötigten Daten für die weiteren Operationen im Backend zu filtern, wird die ankommende Zeichenkette an die Klasse *StringToObjectParser* weitergeleitet. Diese

Klasse wandelt die ankommende Zeichenkette in die gewünschten Objekte um, welche dann an die Logikschicht übergeben werden.
Die ausgehenden Daten werden automatisch mit einem JSON-Parser in JSON-Objekte umgewandelt und in diesem Format an die Empfänger gesendet. Der Vorgang des Parsen wird automatisch vom .NET-Framework übernommen.

- **Logikschicht**

Die Logischicht besteht aus den verschiedenen Controllern, die für die im Backend abzuarbeitenden Use Cases verantwortlich sind. In dieser Schicht werden die ankommenden Daten aus der Webservice Schicht ausgewertet und in den jeweiligen Use Cases bearbeitet. Die fertig bearbeiteten Daten werden anschließend an die Datenbankschicht zur Weiterverarbeitung übergeben.
Eine weitere Aufgabe der Logikschicht ist die Bereitstellung von Daten für die Webservice-Schicht. Hierfür werden verschiedene Leseoperationen in der Datenbankschicht ausgelöst und die daraus gewonnenen Daten für den Webservice gefiltert und aufgearbeitet.

- **Datenbankschicht**

Diese Schicht ist für die Kommunikation des Backends mit der verbundenen SQL-Datenbank verantwortlich. Für die Kommunikation zwischen dem Backend und der SQL-Datenbank wurde MSLinqToSQL verwendet, was als Framework in Visual Studio vorhanden ist. Es mussten die LinqToSQL-Klassen erzeugt werden, welche das Mapping der Datenbanktabellen zu C#-Objekten automatisch übernehmen. Über diese Objekte werden im laufenden Programm Informationen aus der Datenbank ausgelesen bzw. in die Datenbank geschrieben.
Die Logik für die Lese- und Schreibvorgänge des Backends auf die SQL-Datenbak wurden ebenfalls in dieser Schicht implementiert. *EventDokuBackend* ist so aufgebaut, dass unterschiedliche Anwender keine gleichzeitigen Schreiboperationen auf einen Datensätze in der Datenbank vornehmen können. Dies konnte ermöglicht werden, da die Daten nur durch bestimmte einzelne BenutzerInnen geändert werden können.

Kommunikation zwischen den Schichten

Die einzelnen Schichten in *EventDokuBackend* sind so aufgebaut, dass sie von der darüberliegenden Schicht nur über eine speziell definierte Schnittstelle zu erreichen sind. Diese Entscheidung wurde getroffen, damit die einzelnen Schichten voneinander getrennt werden und damit der Austausch von Schichten möglich ist.
Die einzelnen Schichten verfügen über ihre eigenen Objekte, welche die für Operationen benötigte Daten im richtigen Format bereitstellen.
So wurde entschieden, die zu sendenden Daten zu vereinfachen, damit eine möglichst geringe Datenmenge an die Endgeräte gesendet werden muss. Objekte in der Webservice-Schicht enthalten beispielsweise weniger Informationen als Objekte in der Logikschicht.

Um Informationen zwischen den verschiedenen Schichten austauschen zu können, wurden verschiedene Mapper-Klassen erstellt, welche die Umwandlung der einzelnen Objekte übernehmen.

3.3. Features

EventDokuBackend wurde so ausgelegt, dass es nicht nur das Backend der Android-App bildet, sondern auch für andere Applikationen verwendet werden kann. Derzeit wird eine Windows Phone 8 App entwickelt, welche ebenfalls auf *EventDokuBackend* zugreift. Diese App bietet diesselben Möglichkeiten wie die hier beschriebene Android-App. Später soll zusätzlich eine Browserlösung mit Funktionen der Apps realisiert werden, welche ebenfalls auf *EventDokuBackend* zugreift.
Diese Entscheidung für ein einziges Backend wurde auf Grund der einfacheren Wartbarkeit des Systems getroffen. Es wird bei einem ankommenden Request nicht unterschieden, von welchem System die Anfrage stammt, da die Apps auf der gleichen Grundidee aufgebaut sind. So können Fehler leichter gefunden und behoben werden.

4. Entwicklung

4.1. Verwendete Software

Eclipse (Android Developer Tools)

Für die Entwicklung der Android-App wurde die IDE Eclipse (ADT - Android Development Tools) verwendet. Ein Vorteil des ADT-Paketes liegt darin, dass der/die EntwicklerIn (bis auf Java) nichts installieren muss. Alle - für die Entwicklung - benötigten Komponenten sind darin integriert und können sofort eingesetzt werden. Dadurch das Eclipse von Google als IDE für die Android Entwicklung empfohlen wurde, gibt es eine große Community die Hilfe und viele Tutorials über die Android-Entwicklung mit Eclipse bereitstellt.

Visual Studio Ultimate 2013 (Webservice)

Das Backend für unsere Apps (Android und WindowsPhone) wurde mit *Microsoft Visual Studio Ultimate 2013* in der Programmiersprache C# entwickelt. Wie bei der Erläuterung der verschiedenen Schichten von *EventDokuBackend* beschrieben wurde, bietet Visual Studio verschiedene Möglichkeiten eine SQL-Datenbank mit einem Projekt zu verknüpfen.
Bei auftretenden Problemen wurden unter anderen Foren des Microsoft Developer Network und Stackoverflow besucht. Diese Foren sind sehr gut aufgebaut und baten meistens schnelle Hilfe.

Microsoft SQL Server 2014 Management Studio

Zur Erstellung der SQL-Datenbank wurde *Microsoft SQL Server 2014 Management Studio* verwendet. Mit diesem Tool ist es möglich auf SQL Server zuzugreifen und an den darauf

abgelegten Datenbanken zu arbeiten. Das Datenbankskript wurde im Editor dieser Software geschrieben und auf den SQL Server ausgeführt. Zusätzlich wurden neue Datenbanknutzer angelegt und ihnen die benötigten Berechtigungen erteilt.
Mit Hilfe dieser Software wurden die in der App getätigten Updates auf der Datenbank überwacht und die Funktion des Backends getestet.

SourceTree zur Versionsverwaltung

Für die Versionierung der Software wurde Git eingesetzt.
Der Zugriff auf die bei Bitbucket gehosteten Git-Repositories erfollgte über die Software SourceTree von Atlassian. Es wurden verschiedene Entwicklungszweige erstellt, um die App Schritt für Schritt zu erweitern. Für jedes Unterprojekt (Android-App, Windows Phone App, Backend) wurde ein eigenes Repository angelegt.

4.2. Verwendete Hardware zum Testen

Die Android App wurde auf zwei verschiedenen Geräten getestet. Im folgenden werden Testgeräte genauer erläutert und ihre Spezifikationen beschrieben.

Die Internetverbindung für die ausgeführten Tests wurde per Wlan hergestellt, da an den Veranstaltungsorten der OJAD freies Wlan zu Verfügung steht.

Samsung Galaxy S II

Zum einen wurde ein Samsung Galaxy SII I9100 verwendet, auf dem die Android-Version 4.1.2 installiert ist. Es besitzt einen Arm Cortex A9 Dual Core-Prozessor, 1GB RAM, 2GB Anwenderspeicher und ein 3,4-Zoll Display.

Die App läuft flüssig auf diesem Gerät, obwohl verschiedene andere Apps im Hintergrund geöffnet waren. So wurde ein mittels TeamViewer der Screen des Smartphone auf einen PC übertragen, um ein Video von der Benutzung der App zu erstellen.

Samsung Galaxy Note 3

Zusätzlich wurde ein Samsung Galaxy Note 3 N9005 verwendet, auf dem Android in der Version 4.4 installiert ist. Es besitzt einen 2,3 GHz Quad Core-Prozessor, 3 GB RAM und ein 5,7-Zoll Display.

Die App läuft flüssig auf diesem Gerät, obwohl verschiedene andere Apps im Hintergrund geöffnet waren.

5. Zusatzinformationen

Wichtige Information zum Testen der Anwendung:
Zum aktuellen Zeitpunkt ist noch kein Produktivserver installiert. Für Tests wird daher ein Test-System (Server, Webservice und Datenbank) benötigt. Wenn das System gestet werden soll, nehmen Sie bitte Kontakt mit dem Entwicklerteam auf, damit ein Termin vereinbart werden kann. An diesem Termin wird auf dem Laptop der Server gestartet und die jeweilige IP in die App eingefügt, damit ein Test stattfinden kann.

Teile dieser Dokumentation wurden aus der Bachelorarbeit: “Mobile Multiplattform-Entwicklung am Beispiel einer Applikation” von Martin Münch, übernommen.