tgm
Die Schule der Technik

JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

Systemtechnik

5HIT 2017/18

Synchronisation bei mobilen Diensten

Laborprotokoll

Marvin Ertl

18. April 2018

	Version:	1.0
Note:	Begonnen:	14.04.2018
Betreuer: Michael Borko	Beendet:	18.04.2018

Inhaltsverzeichnis

1 Einführung

Diese Übung soll die möglichen Synchronisationsmechanismen bei mobilen Applikationen aufzeigen.

1.1 Ziele

Das Ziel dieser Übung ist eine Anbindung einer mobilen Applikation an ein Webservices zur gleichzeitigen Bearbeitung von bereitgestellten Informationen.

1.2 Voraussetzungen

- Grundlagen einer höheren Programmiersprache
- Grundlagen über Synchronisation und Replikation
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von Webservices

1.3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die einen Informationsabgleich von verschiedenen Clients ermöglicht. Dabei ist ein synchronisierter Zugriff zu realisieren. Als Beispielimplementierung soll eine Ëinkaufsliste"gewählt werden. Dabei soll sichergestellt werden, dass die Information auch im Offline-Modus abgerufen werden kann, zum Beispiel durch eine lokale Client-Datenbank.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Wichtig ist dabei die Dokumentation der Vorgehensweise und des Designs. Es empfiehlt sich, die im Unterricht vorgestellten Methoden sowie Argumente (pros/cons) für das Design zu dokumentieren.

1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen "**Grundkompetenz überwiegend erfüllt**"
  - Beschreibung des Synchronisationsansatzes und Design der gewählten Architektur (Interaktion, Datenhaltung)
  - Recherche möglicher Systeme bzw. Frameworks zur Synchronisation und Replikation der Daten
  - Dokumentation der gewählten Schnittstellen
- Anforderungen "**Grundkompetenz zur Gänze erfüllt**"
  - Implementierung der gewählten Umgebung auf lokalem System
  - Überprüfung der funktionalen Anforderungen zur Erstellung und Synchronisation der Datensätze
- Anforderungen **Ërweiterte-Kompetenz überwiegend erfüllt**"
  - CRUD Implementierung

  - Implementierung eines Replikationsansatzes zur Konsistenzwahrung

- Anforderungen **Ërweiterte-Kompetenz zur Gänze erfüllt**"
  - Offline-Verfügbarkeit
  - System global erreichbar

2 Plattform

Um eine Anwendung zu Entwickeln, welche es erlaubt, Daten zu synchronisieren und eine Offline-Verfügbarkeit mittels einer lokalen Client-Datenbank zu Verfügung zu stellen, gibt es einige unterschiedliche Ansätze.

Wie bereits im Unterricht besprochen bietet diff-sync eine einfache Möglichkeit mittels NodeJS Daten zu synchronisieren. Weiters gibt es unzählige Plattformen die solche Features anbieten, unter anderem Googles Firebase. Diese erlaubt es einfach, eine App zu Entwickeln, welche die Daten mit einer Firebase Datenbank synchronisieren. [3]

Schlussendlich wurde Couchbase als Plattform zur Umsetzung dieser Übung gewählt, da es mittels Sync Gateway und Couchbase Lite die umfangreichste und die meisten Funktion bietet. Ebenso unterstützt Couchbase sehr viele unterschiedliche Systeme und Programmiersprachen. Unter anderem auch C# mit UWP, wobei die mit der UWP und .Net Plattform entwickelten Anwendungen mittels Xamarin auch auf Android und IOS laufen können. Da bereits einiges an Erfahrung mittels der Laborübung mit der UWP Plattform gesammelt worden ist, wurde für diese Aufgabe ebenfalls auf diese Plattform gesetzt.

2.1 Couchbase

Zur Umsetzung solch einer Anwendung mittels Couchbase werden 3 Komponenten benötigt.

- **Couchbase Server**: Ein NoSQL Server, welcher zentral alle Daten verwaltet.
- **Sync Gateway**: Dies stellt die Schnittstelle zwischen den Server und den Clients dar. Sync Gateway bietet eine REST-Schnittstelle auf den Server zur Synchronisierung der Datensätze.
- **Couchbase Lite**: Stellt die Client Datenbank zu Verfügung. Kann offline verwendet werden, sobald eine Verbindung hergestellt werden kann, werden die Datensätze mittels Sync Gateway synchronisiert.

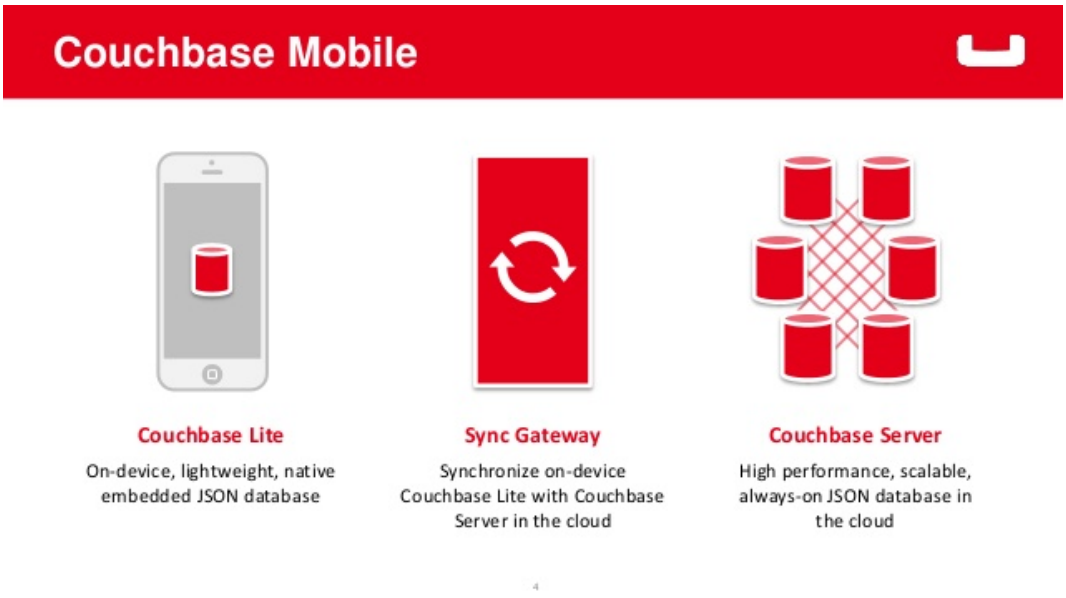

Abbildung 1: Umsetzung mittels Couchbase

Einer der großen Vorteile dieser Plattform ist, dass der Client selber nie mit dem Server im Hintergrund kommuniziert bzw. umgesetzt werden muss. Die App selber greift nur auf die lokale Datenbank zu und nur diese Zugriffe müssen umgesetzt werden, die Synchronisierung übernimmt Sync Gateway und das Abspeichern der Couchbase Server. Außerdem kümmert sich Sync Gateway um die Versionierung der Dokumente sowie um das Lösen von Konflikten bei Dokumenten. [5]

3 Umsetzung

Zum Umsetzen dieser Übung wurde auf den Server mit der IP-Adresse 37.252.185.24 ein Couchbase Server sowie Sync Gateway installiert. Mittels Visual Studio und C# wurde ein UWP-App entwickelt, welche Couchbase Lite als lokale Datenbank verwendet.

3.1 Couchbase Server

Zunächst wurde ein Couchbase Enterprise Server auf einen Server aufgesetzt, verwendet wurde die Version 5.1. Die Enterprise Software darf kostenlos zur Entwicklung und Testen verwendet werden, erst beim normalen Betrieb muss eine Lizenz erworben werden.

Der Server mit der Datenbank ist unter der IP-Adresse 37.252.185.24 und dem Port 8091 erreichbar. Beim Aufsetzen wurde ein neuer Cluster erstellt und die RAM Zuweisung wurde verringert, ansonsten wurden alle Einstellung bei den default Werten belassen.

Über diese Adresse kann nun auf das Admin Dashboard des Couchbase Servers zugegriffen werden. Über dieses wurde ein Benutzer *UserEin* erstellt, welche volle Zugrissrechte auf den Bucket *einkaufsliste* hat. [7]

3.2 Sync Gateway

Wie auch der Couchbase wurde nun Sync Gateway auf dem Server installiert. Auch hier muss kaum etwas konfiguriert werden, nur beim erstmaligem Starten des Services muss eine Konfigurationsdatei angegeben werden.

```
1   {
2       "log": ["*"],
3       "databases": {
4           "db": {
5               "server": "http://localhost:8091",
6               "bucket": "einkaufsliste",
7               "username": "UserEin",
8               "password": "Einkaufsliste",
9               "users": { "UserEin": { "disabled": false, "admin_channels": ["*"], "password":
                ↪  "Einkaufsliste" } },
10              "sync": `function(doc) {
11                  channel("liste");
12              }`
13          }
14      }
15  }
16
```

Diese JSON gibt den Namen der Datenbank an *db*, weiters muss der Couchbase Server angegeben werden, dieser ist lokal aufrufbar. Nun wurde am Server ein Bucket mit dem Namen *einkaufsliste* erstellt, dieser Name muss auch in der Konfiguration angegeben werden. Weiters muss der Benutzer sowie sein Passwort angegeben werden sein, sodass sich der Service mit den Server kommunizieren kann. Abschließend gibt eine *sync* Funktion an, welche Dokumente synchronisiert werden sollen. Hierbei kann mittels Tags gearbeitet werden, sodass ein Client nur bestimmte Dokumente synchronisiert, in diesem Fall wurden alle Dokumente dem Channel *liste* hinzugefügt.

Ob alles richtig funktioniert hat, kann einerseits anhand der Meldung in der Konsole überprüft werden, anderseits muss die Adresse mit dem Port 4984, bei einer erfolgreichem Konfiguration folgendes zurückliefern.

```
1   {"couchdb":"Welcome","vendor":{"name":"Couchbase Sync
    ↪  Gateway","version":"2.0"},"version":"Couchbase Sync Gateway/2.0.0(832;2d8a6c0)"}
```

Wichtig zu beachten ist, sobald Sync Gateway aufgesetzt worden ist, sollte das Admin Dashboard des Couchbase Servers nicht mehr zum Verwalten der Dokumente verwendet werden, dies könnte nämlich zu Problemen bei der Synchronisierung zu den Clients führen. Sync Gateway bietet eine REST-Schnittstelle, über welche alle Dokumente im Bucket verwaltet werden können. [1]

3.3 Couchbase Lite

Wie bereits oben erwähnt wurde bei dieser Übung auf die UWP Plattform zurückgegriffen, da bei der letzten Übung bereits einiges an Erfahrung gesammelt worden ist und die Entwicklung sowie das Testen einfacher und schneller ist, da die Anwendung auch lokal getesten werden kann und nicht in einem Emulator oder auf einem Smartphone getesten werden muss.

Um Couchbase Lite dem Projekt hinzuzufügen, muss das NuGet Package *Couchbase.Lite.Enterprise* installiert werden, dies kann über den NuGet Package Manager in Visual Studio gemacht werden.

Die App bietet eine einfache Benutzeroberfläche, welche alle Gegenstände in einer Liste darstellt, diese können bearbeitet oder gelöscht werden. Über zwei Eingabefelder am unteren Rand, können neue Items zur Liste hinzugefügt werden. Dafür muss nach der Eingabe der Werte auf den *Hinzufügen* Button geklickt werden. Das Item wird sofort in der Liste angezeigt und nach kurzer Zeit, wenn möglich, auch synchronisiert und bei anderen Benutzern angezeigt.

Um einen Eintrag zu bearbeiten, muss dieser aus der Liste ausgewählt werden. Über den [Bearbeiten] Button können nun die Werte geändert werden. Mittels dem *Speichern* Button können diese Änderungen gespeichert werden. Auch bei Änderungen wird der Eintrag sofort am Client angezeigt und nach kurzer Zeit bei anderen Benutzern.

Um Einträge zu löschen, kann ganz einfach der *X* Button bei dem entsprechendem Eintrag getätigt werden. Diese Änderung wird ebenfalls sofort synchronisiert.

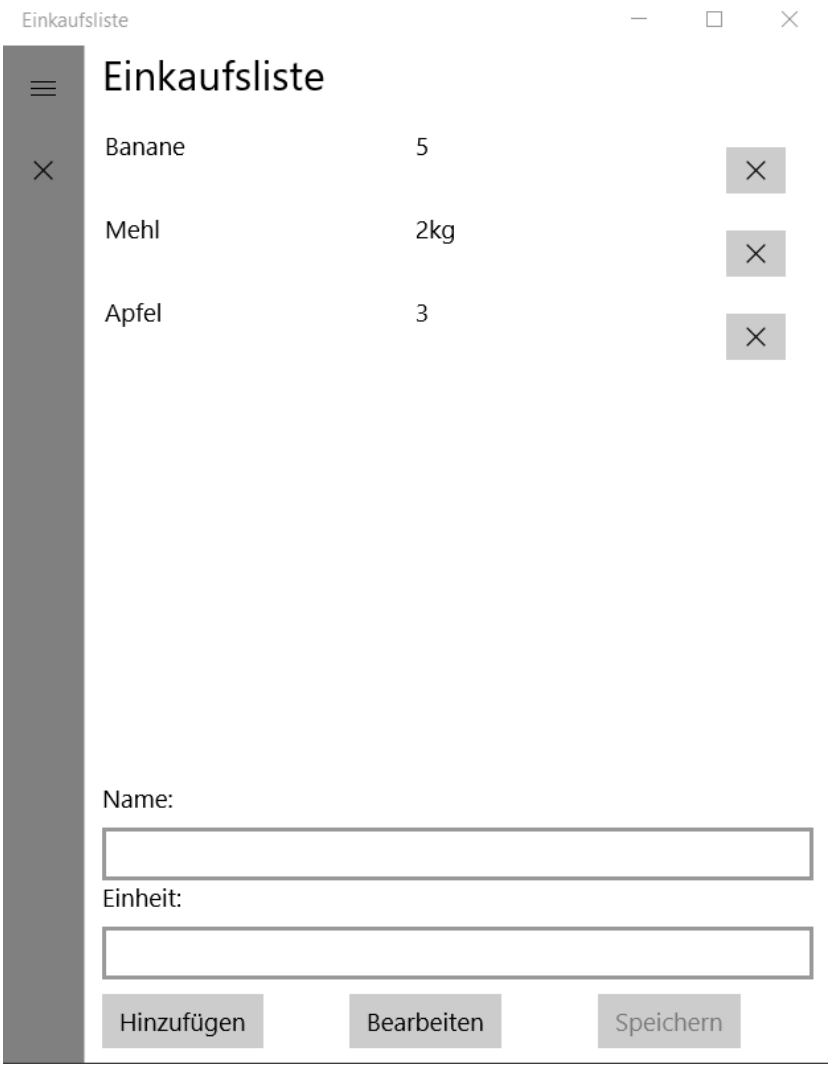

Abbildung 2: Benutzeroberfläche der Einkaufsliste

3.3.1 Benutzeroberfläche

Zur Darstellung der Einträge wurde eine ListView verwendet, diese stellt alle Objekte dar, welche der ObservableCollection hinzugefügt werden. Um dies Umzusetzen, wurde ein Objekt namens *Item* erstellt, dieses beinhaltet alle wichtigen Informationen und wird für die Darstellung in der Liste verwendet. [4]

```
1 <ListView x:Name="MyListView" Grid.Row="1" Grid.Column="0" ItemsSource="{x:Bind ViewModel.Items}"
  ↪ SelectionChanged="ListView_OnSelectionChanged" HorizontalAlignment="Stretch"
  ↪ HorizontalContentAlignment="Stretch">
2     <ListView.ItemTemplate>
3         <DataTemplate x:DataType="Einkaufsliste:Item">
4         <Grid>
5             <TextBlock Grid.Column="0" Text="{x:Bind Name}" Height="50" VerticalAlignment="Center"
              ↪ />
6             <TextBlock Grid.Column="1" Text="{x:Bind Value}" x:Phase="1" Height="50"
              ↪ VerticalAlignment="Center"/>
```

```
            <Button Grid.Column="2" x:Name="delete" FontFamily="Segoe MDL2 Assets"
            ↪  DataContext="{x:Bind ID}" Content="&#xE10A;" Click="Button_Click_Delete" />
        </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Alle Informationen, welche angezeigt werden sollen, werden in den Objekt namens *Item* gespeichert. Dieses Objekt beinhaltet den Namen sowie die Menge von dem Objekt. Zur eindeutigen Identifikation übernimmt das Objekt noch die ID des Couchbase Documents.

```
public class Item
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Value { get; set; }
}
```

Wobei die Klasse ItemViewModel um das Verwalten der Collection von Items kümmert. In dieser Klasse werden beim Start des Programmes alle Objekte ausgelesen, sowie bei der Synchronisierung die Objekteinformationen aktualisiert, sodass die neuen Informationen angezeigt werden. [4]

3.3.2 Lokale Datenbank

In der App Klasse wurde die Methode *StartDBConnection* erstellt, diese baut eine Verbindung zur lokalen Datenbank auf und falls möglich, wird die Synchronisierung mit dem Couchbase Server gestartet. Diese Methode wird beim Start der App aufgerufen und aktiviert als erstes den UWP Support und erstellt bzw. verbindet sich zu der Datenbank *einkaufsliste*.

```
Couchbase.Lite.Support.UWP.Activate();
this.database = new Database("einkaufsliste");
```

Nun muss die Konfiguration für die Replikation erstellt werden und hierfür muss ein Endpoint bestimmt werden. Als Replicator Type wurde Push und Pull gewählt. Weiters muss der Channel angegeben werden, welcher in der Sync Gateway Konfigurationsdatei festgelegt worden ist, angegeben werden. Abschließend muss noch die Authentifikation konfiguriert werden, nun kann die Replikation gestartet werden. [2]

```
var targetEndpoint = new URLEndpoint(new Uri("ws://37.252.185.24:4984/db"));
var replConfig = new ReplicatorConfiguration(this.database, targetEndpoint)
{
    ReplicatorType = ReplicatorType.PushAndPull
};
replConfig.Channels = new List<String>();
replConfig.Channels.Add("liste");
replConfig.Continuous = true;

replConfig.Authenticator = new BasicAuthenticator("UserEin", "Einkaufsliste");

var replicator = new Replicator(replConfig);
replicator.Start();
```

Nun kann in der MainPage die Verbindung zur lokalen Datenbank verwendet werden, um CRUD Operationen durchzuführen.

3.3.3 Synchronisierung

Nun wird die lokale Datenbank mit dem Server synchronisiert, um auf die Änderungen der lokalen Datenbank zu reagieren, welche durch die Synchronisierung durchgeführt werden, muss ein Change Listner auf die Datenbank gelegt werden. Dieser wird immer ausgelöst, wenn eine Änderung an der Datenbank durchgeführt wird. Dieser Listner ruft die Methode *UpdateAll* auf, diese liest alle Dokumente aus der Datenbank aus und aktualisiert die *Items* Objekte für die Darstellung.

```
this.param.App.database.AddChangeListener((sender, args) =>
{
    this.ViewModel.UpdateAll();
});
```

In dieser Methode wird mittels einem QueryBuilder eine Query erstellt, welche alle Dokumente aus der Datenbank ausliest. Die Ergebnisse können zu einer Liste gecastet werden. Mittels dem enthaltenem Dictionary kann auf die Werte der Dokumente zugegriffen werden. [2]

```
using (var query = QueryBuilder.Select(SelectResult.All())
.From(DataSource.Database(this.param.App.database)))
{
    var result = query.Execute();
    var res = result.ToArray();

    var documents = new List<string>();
    foreach (var i in res)
    {
        foreach (var item in Items)
        {
            item.Name = i.GetDictionary(0).GetString("name");
            item.Value = i.GetDictionary(0).GetString("value");
            documents.Add(item.ID);
            found = true;
            break;
        }
    }
}
```

3.3.4 Löschen eines Eintrages

In der ListView hat jeder Eintrag einen Löschen Button, dieser hat im DataContext die ID des Dokumentes gespeichert. Bei einem Click Event wird die Methode *Button_Click_Delete*, diese castet den Sender zu einem Button um und holt sich aus dem DataContext die ID. Mittels dieser ID kann aus der Datenbank das Dokument gelöscht werden. [6]

```
private void Button_Click_Delete(object sender, RoutedEventArgs e)
{
    Windows.UI.Xaml.Controls.Button button = (Windows.UI.Xaml.Controls.Button)sender;
    Document doc = this.param.App.database.GetDocument((string) button.DataContext);
    this.param.App.database.Delete(doc);

}
```

3.3.5 Hinzufügen eines Eintrages

Wird der Button *Hinzufügen* geklickt, werden die Eingaben in die Felder überprüft, falls diese leer sind, wird der Benutzer informiert, dass diese nicht leer sein dürfen. Ist dies nicht der Fall, wird ein neues Dokument erstellt. Hierfür wird ein MitabeDocument erstellt, dieses bekommt automatisch eine ID zugewiesen, das Dokument wird anschließend in die Datenbank gespeichert. [2]

```
1  public void AddItem(string name, String value)
2  {
3      using (var mutableDoc = new MutableDocument())
4      {
5          mutableDoc.SetString("name", name).SetString("value", value).SetString("ID",
           ↪  mutableDoc.Id);
6          this.database.Save(mutableDoc);
7      }
8  }
```

3.3.6 Update eines Eintrages

Wenn der Benutzer einen Eintrag bearbeitet, werden die Werte in die Eingabefelder geschrieben, diese kann der Benutzer bearbeiten. Klickt der Benutzer anschließend auf den *Speichern* Button, wird das Dokument mit der ID aus der Datenbank ausgelesen, die Werte aktualisiert und wieder in die Datenbank abgespeichert. [2]

```
1  using (var doc = this.param.App.database.GetDocument(this.id))
2  using (var mutableDoc = doc.ToMutable())
3  {
4      mutableDoc.SetString("name", this.name.Text);
5      mutableDoc.SetString("value", this.value.Text);
6      this.param.App.database.Save(mutableDoc);
7  }
```

Literaturverzeichnis

[1] *Couchbase*. [Online; accessed 17. Apr. 2018]. 2018. URL: https://developer.couchbase.com/documentation/mobile/2.0/installation/sync-gateway/index.html.

[2] *Couchbase Documentation*. [Online; accessed 18. Apr. 2018]. 2018. URL: https://developer.couchbase.com/documentation/mobile/2.0/couchbase-lite/csharp.html#getting-started.

[3] *Firebase*. [Online; accessed 17. Apr. 2018]. 2018. URL: https://firebase.google.com.

[4] *ListView Class (Windows.UI.Xaml.Controls) - UWP app developer*. [Online; accessed 18. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/uwp/api/Windows.UI.Xaml.Controls.ListView.

[5] *Offline-first apps with Couchbase Sync Gateway | The Couchbase Blog*. [Online; accessed 17. Apr. 2018]. 2018. URL: https://blog.couchbase.com/offline-first-apps-couchbase-sync-gateway.

[6] serenaz. *Buttons - UWP app developer*. [Online; accessed 18. Apr. 2018]. 2018. URL: https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/buttons.

[7] *Why Couchbase?* [Online; accessed 17. Apr. 2018]. 2018. URL: https://developer.couchbase.com/documentation/server/5.1/introduction/intro.html.

Abbildungsverzeichnis