Miniprojekt – Teil WPF

Neben den Vorlesungen und Übungsaufgaben ist auch ein Miniprojekt Teil von Mobile and GUI Engineering. Das Projekt gliedert sich – wie auch die Vorlesung – in einen Android-Teil und einen WPF-Teil. Ziel des Projekts ist es, das Gelernte selbständig auf ein neues Szenario anwenden zu können.

Miniprojekt bedeutet, dass das Projekt über mehrere Wochen läuft und von Ihnen selbständig, in den Übungslektionen oder im Selbststudium, bearbeitet wird. Sie dürfen das Projekt auch mit einem Partner oder in einer Gruppe von maximal drei Personen durchführen (Teams mit Mitgliedern aus mehreren Studiengängen wären toll). Tragen Sie die Gruppen bitte hier ein: https://goo.gl/0K8i5G#gid=1933443154. (Sheet „Gruppeneinteilung - Teil WPF")

Die beiden Teile des Miniprojekts werden bewertet, aber nicht benotet. Nur wenn Sie beide bestandenen haben sind Sie zur Prüfung zugelassen! Achten Sie also darauf, die Bewertungskriterien einzuhalten und sprechen Sie bei Unklarheiten mit Ihrem Übungsbetreuer.

Bewertungskriterien

Die folgenden Bewertungskriterien dienen als Richtlinien für die Bewertung. Um das Projekt zu bestehen, müssen nicht alle Punkte vollumfänglich erfüllt sein, bei zu grossen Mängeln wird die Prüfungszulassung aber nicht erteilt:

Szenarios	Können die vorgegebenen (oder selbst definierten) Szenarios durchgespielt werden?
Design	Entspricht das User Interface den besprochenen Prinzipien?
Interaktion	Behandelt die App Fehlerfälle und gibt dem Anwender Feedback dazu? Ist die Benutzerführung verständlich?
Code	Ist der Code sauber strukturiert (einheitlicher Stil, keine TODOs, Dokumentation nicht-trivialer Stellen)? Ist der Code in einem Repository eingecheckt? Sind Tests vorhanden?

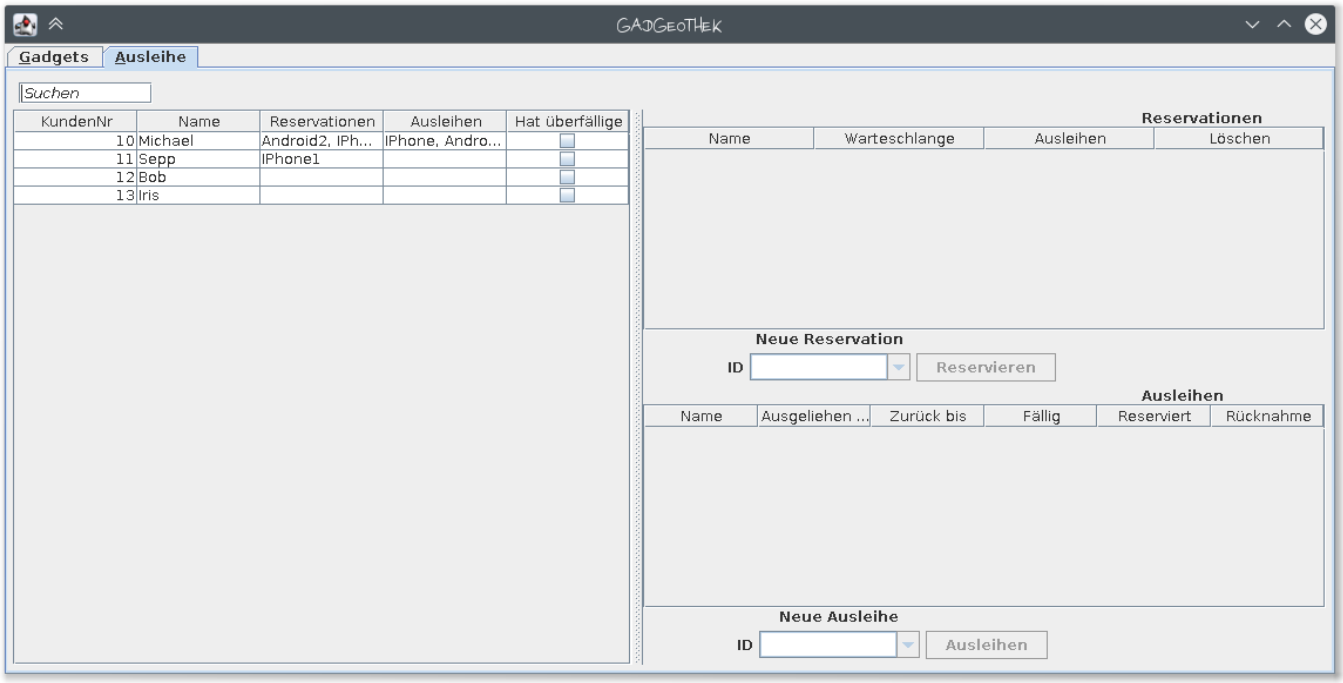
Die Bewertung erfolgt während einer kurzen Demonstration der App in Ihrer Übungslektion in der Semesterwoche 14 (also der letzten WPF-Woche).

Im nächsten Kapitel folgt die Aufgabenstellung für die Gadgeothek-Admin-App die wir bauen wollen.

Aufgabenstellung Gadgeothek-Admin-App

Sie kennen aus dem Android-Teil die Gadgeothek der HSR-Bibliothek. Im WPF-Miniprojekt geht es nun darum, eine Admin-App dazu zu entwickeln.

In den Vorlagen finden Sie das Administrationstool der Gadgeothek, eine Swing-Applikation die im Vorgängermodul User Interfaces 1 entwickelt wurde und uns freundlicherweise von den Autoren Samuel Jost und Philipp Schilter zur Verfügung gestellt wird. Im Miniprojekt soll nun *der Verwaltungsteil* davon mithilfe von WPF neu geschrieben werden.



Eigene Aufgabenstellungen

Alternativ können Sie auch ein eigenes Thema bearbeiten. Schreiben Sie dazu eine eigene Aufgabenstellung, definieren Sie Szenarios die Sie umsetzen werden und senden Sie sie an martin.seelhofer@hsr.ch. Sie werden innert zwei (Arbeits-)Tagen einen go/no go Entscheid erhalten.

In Umfang und Komplexität sollte die eigene Aufgabe in etwa der Gadgeothek-Admin-App entsprechen (Domainmodell mit 3 Klassen, Kommunikation mit einem Server) und muss mit der Windows Presentation Foundation (WPF) in C# und XAML entwickelt werden. Auf Anfrage – und eigenes Risiko – können auch Windows 8 Apps (Modern UI), die mit XAML und C# geschrieben werden, bewilligt werden.

Szenarios/User Stories der Gadgeothek Admin App

Szenarios helfen Ihnen zu verstehen, in welchem Kontext die Software verwendet wird. Die untenstehenden Szenarien beschreiben, was ihre Applikation am Ende des Miniprojektes unterstützen sollte. Verwenden Sie sie zum Testen ihrer Applikation und schauen sie darauf, dass Sie möglichst alle davon optimal unterstützen können.

Szenario/Story Gadgets anzeigen

Als Administrator möchte ich alle verfügbaren Gadgets angezeigt bekommen, um mir einen Überblick über die Gadgeothek zu verschaffen. Ich bin zufrieden, wenn die Gadgets mit allen Eigenschaften in einer Tabellendarstellung angezeigt werden können und ich die Tabelle nach den einzelnen Spalten sortieren kann.

Szenario/Story Gadget hinzufügen

Als Administrator möchte ich ein neues Gadget hinzufügen können, um die Gadgeothek nach Kauf eines neuen Gadgets wieder auf den aktuellsten Stand bringen zu können. Ich bin zufrieden, wenn ich in einem Fenster ein neues Gadget mit all seinen Eigenschaften erfassen kann und dieses anschliessend in die Liste der Gadgets aufgenommen wird.

Szenario/Story Gadgets entfernen

Als Administrator möchte ich ein Gadget entfernen können, um die Gadgeothek nach Verlust eines Gadgets wieder auf den aktuellsten Stand bringen zu können. Ich bin zufrieden, wenn ich ein Gadget in der Tabellendarstellung markieren und dieses nach Bestätigen einer Sicherheitsrückfrage aus der Liste entfernen kann.

Szenario/Story Ausleihevorgänge überwachen

Als Administrator möchte ich die aktuellen Ausleihevorgänge anzeigen können, um den Betrieb der Gadgeothek überwachen zu können. Ich bin zufrieden, wenn ich die Ausleihvorgänge in einer chronologisch sortierten Tabellenform ansehen kann, welche sich selbst in regelmässigen Zeitabständen von einigen Sekunden aktualisiert. Optional kann versucht werden, eine Aktualisierung des User Interfaces in Echtzeit umzusetzen.

Installation

In der Anfangsphase des Projekts wird ein grosser Teil der Domain und der Businesslogik der Gadgeothek-Admin-App zur Verfügung gestellt. Setzen sie bei ihrer Lösung diesen Code ein! Die nachfolgenden Klassendiagramme bietet ihnen eine Übersicht über den vorgebebenen Code. Sollten sie das Bedürfnis haben diesen Code grundlegend zu modifizieren, so dürfen sie dies gerne tun. Der Code steht in Form einer vorbereiteten Visual Studio Solution auf GitHub zur Verfügung.

Der Serverteil ist derselbe wie beim Miniprojekt Android. Er besteht aus einer einfachen Node.js-Anwendung die eine REST-Schnittstelle anbietet. Sie finden den Code dazu unter github.com/HSR-MGE/Miniprojekt-Server. Sie können den Server entweder lokal starten, oder Sie verwenden eine unserer zehn Instanzen mge1.dev.ifs.hsr.ch, mge2 ... mge10.dev.ifs.hsr.ch (Achtung: wegen noch nicht gelöster technischer Probleme liefern diese Instanzen keine Push-Notifications an die Admin-Software).

Um den Server lokal zu starten müssen Sie Node.js installiert haben. Kopieren Sie dann die package.json und server.js Dateien in ein Verzeichnis (bzw. klonen Sie einfach das Repository). In diesem Verzeichnis installieren Sie dann mit npm (das ist der Node-Packagemanager) die benötigten Libraries und starten den Server:

```
$ npm install
...
$ node server.js
```

Der Server hört auf Port 8080 (HTTP) und gibt Debug-Output aus. Alternativ steht auch ein Docker-Container bereit den Sie stattdessen einsetzen können. Zum Beispiel so:

```
$ docker run -t -i -p 8080:8080 misto/miniprojekt-server
```

Die Vorlage der Administrationsoberfläche (Java Swing) github.com/HSR-MGE/Miniprojekt-Admin , welche Sie als Richtlinie für die Umsetzung Ihres User Interfaces nutzen können, starten Sie mit folgendem Befehl und unter der Angabe der Serveradresse und Port:

```
java -Dserver=localhost:8080 -jar gadgeothek-admin.jar
```

Sobald sie später mit dem WPF-Client auf denselben Server verbinden, werden Sie Updates live in der Java-Administrationsoberfläche sehen. Und umgekehrt, falls Sie die Aktualisierung des UI in Echtzeit in Ihrem WPF-Projekt umsetzen (optional).

Vorlagen

Die Kommunikation mit dem Server erfolgt über zwei vorgegebene Service-Klassen (in C#). Diese implementieren zwei einfache Clients für eine REST-Schnittstelle, je einmal zur Nutzung als Admin und als Client. Das Hauptaugenmerk lag bei der Entwicklung auf dem Admin-Teil, da Sie die Funktionen der Client-Implementierung hier voraussichtlich nicht nutzen werden (bereits im Android-Teil umgesetzt).

Die Service-Klassen erwarten im Konstruktor die Web-Adresse (URL), an welcher der Server zu finden ist. Tragen Sie die entsprechende Adresse dazu z.B. als App-Setting in der App.config Ihres WPF-Projekts ein (alternativ können Sie eine ComboBox mit allen verfügbaren Adressen in Ihr User Interface einbauen).

Für den Zugriff auf eine lokal laufende Instanz lautet die entsprechende Zeile im Abschnitt `<appSettings>` somit:

```
<add key="server" value= "http://localhost:8080" />
```

Wenn Sie eine unserer Serverinstanzen verwenden, lautet die entsprechende Zeile:

```
<add key="server" value= "http://mgeX.dev.ifs.hsr.ch/" />
```
(ersetzen Sie das X in «mgeX» jeweils durch die Instanz-Nummer)

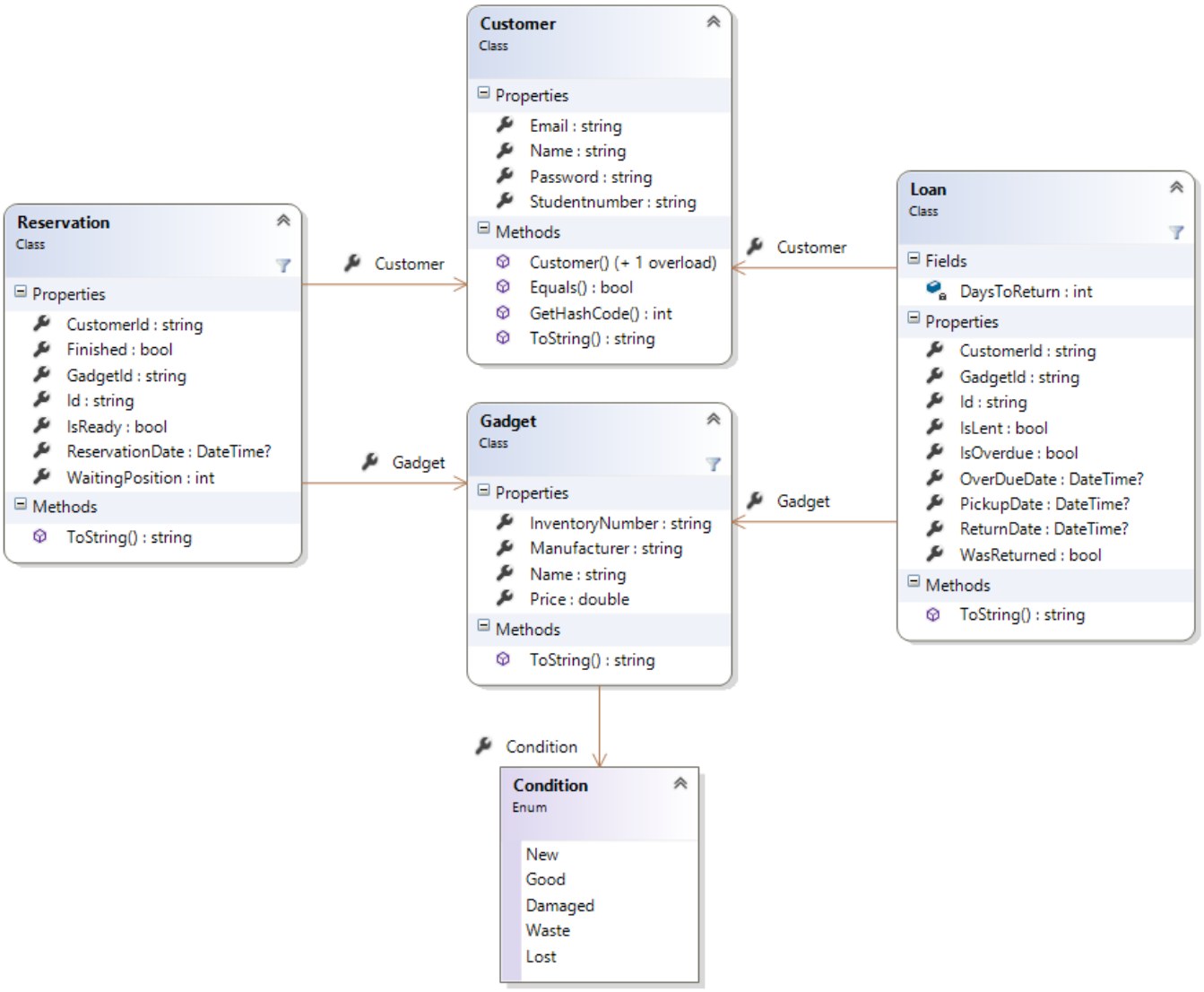
Beachten Sie, dass der Zusatz `/public` hier im Unterschied zum Android-Teil NICHT angegeben werden muss, sondern in der Service-Implementierung als Teil des Pfads zu den REST-Endpunkten automatisch ergänzt wird.

Weitere Hinweise zur Nutzung der Klassen finden Sie in den (vielen) XML-Kommentaren des C#-Quellcodes der Service-Klassen, sowie den beiden Konsolen-Apps, welche in der Projektvorlage zu finden sind.

In der Vorlage finden Sie zudem das Domain-Package, welches die vier Klassen `Gadget` (entspricht einem konkreten Geräte), `Customer` (entspricht einem Kunden), `Loan` (eine Ausleihe eines Gerätes) und `Reservation` enthält. Die Klassen sollten hoffentlich selbsterklärend sein, bei Fragen zögern Sie aber bitte nicht diese in den Übungen zu stellen.

Achtung:
Der Einfachheit halber erfolgt der Aufruf der Service-Klassen synchron, d.h. wenn diese Methoden direkt aus dem User Interface aufgerufen werden, blockiert das User Interface kurz. Normalerweise ist dies kein Problem, da ein Desktop-Rechner i.d.R. über eine schnelle Internetverbindung verfügt. Im Falle von grossen Datenmengen sollte hier aber natürlich darüber nachgedacht werden, die Aufrufe in einem Background-Thread (asynchron) auszuführen, wie Sie das z.B. bei Android gemacht haben.

Klassendiagramm des Domain-Packages



Klassendiagramm des Service-Packages