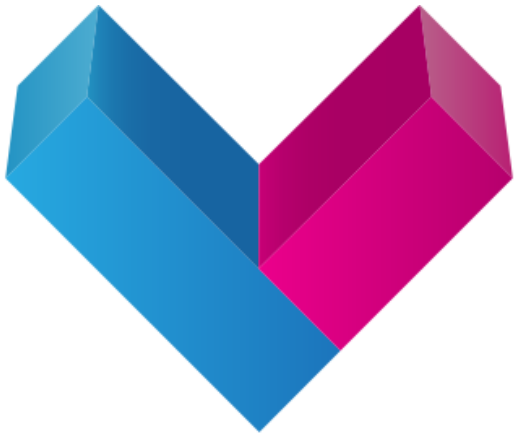
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit Frühjahrssemester 2013



venue

Nightlife Digitalisierung

TECHNISCHER BERICHT

Marco Bachmann

1 INHALTSVERZEICHNIS

1 Inhaltsverzeichnis 2
2 Einleitung 4
2.1 Zweck des Dokuments 4
2.2 Glossar 4
3 Übersicht 5
3.1 Ausgangslage 5
3.2 Aufgabenstellung 5
3.2.1 Aufgabe 5
3.2.2 Vorgaben 5
4 Ergebenisse 8
4.1 Architektur 8
4.1.1 Projekt Struktur 9
4.2 Wekrzeuge 9
4.3 Guest App 9
4.3.1 Technologie 9
4.3.2 UI Design 9
4.3.3 Services 13
4.3.4 Authentication 17
4.3.5 Datenzugriff 19
4.3.6 View Models 26
4.3.7 Views 27
4.3.8 ImageCache 29
4.3.9 PayPal Zahlung 30
4.3.10 Lokalisierung 33
4.3.11 AppBar Commands 34
4.3.12 Live Tiles 35
4.3.13 Push Notifications 35
4.3.14 Windows Phone Wallet 36
4.4 Door App 37
4.4.1 QR Code Scanning 37
4.5 Near Field Communication 38
4.5.1 Technologie 38
4.5.2 Sicherheit 39
4.5.3 Umsetzung 39
4.5.4 App Launch Befehl 43
4.5.5 Geschwindigkeit 44
5 Schlussfolgerungen 46

5.1 Business 46
5.2 Technisch 46
5.2.1 Plattform 46
5.2.2 Near Field Communication 46
6 Referenzen 48
7 Abbildungsverzeichnis 50

2 EINLEITUNG

2.1 ZWECK DES DOKUMENTS

Das Zielpublikum dieses Dokuments sind Software Entwickler, welche sich mit App Entwicklung befassen. Es beschreibt die Umsetzung eines Dienstes für Endkonsumenten auf heterogenen Gerätelandschaften.

2.2 GLOSSAR

CodeBehind	Neben einem deklarativen Datei gibt es eine Datei mit programmatischen Code, der aber zur gleichen Klasse gehört wie die deklarative Datei
Xaml	Xml Application Markup Language, deklarative UI Sprache
MVVM	Model View ViewModel, Software Design Pattern
Windows Phone Toolkit	UI Bibliothek von Microsoft, speziell für Windows Phone
NFC	Near Field Communication
Assembly	Das Resultat eines Kompiliervorgangs in .Net. Ergibt eine DLL oder EXE Datei
Binding Converter	Convertiert den gebundenen Wert bevor er auf das Model geschrieben wird und umgekehrt.
Identity Provider	Ein Identity Provider besitzt Informationen eines Benutzers, welche ihn innerhalb des Identity Provider einzigartig machen.

3 ÜBERSICHT

3.1 AUSGANGSLAGE

Venue ist ein Projekt der Firma triarc laboratories, das die digitale Automatisierung in Nachtklubs, Konzertsälen und Kongresszentren voranbringen soll. Gästelisten, Ressourcenplanung und Türkontrollen sollen mit Venue überwacht werden können. Bisher existieren ein Konzept und ein grundlegendes Design.

Venue besteht aus den Komponenten Door App, Guest App und einem Web Management Client.

3.2 AUFGABENSTELLUNG

3.2.1 AUFGABE

Der Umfang dieser Arbeit besteht aus zwei Teilen, der Guest App und der Door App. Die Guest App soll gemäss Vorgaben implementiert werden. Die Door App soll gemäss Anforderungen entworfen und implementiert werden.

3.2.2 VORGABEN

Beide Apps verbinden sich über das Internet zu einem Server. Der Server bietet eine REST Schnittstelle zur Datenabfrage und Aktualisierung an. Die Authentifizierung erfolgt über Azure Access Control Service.

Für die Guest App steht folgende Design Vorlage bereit:

http://prezi.com/1pcw3frlv3rd/?utm_campaign=share&utm_medium=copy

3.2.2.1 GUEST APP

Die Guest App wird von den Gästen des Clubs verwendet. Grundsätzlich soll die App selbsterklärend und ohne Einführung benutzt werden können. Es sollen viele Bilder verwendet werden, um die App visuell ansprechend zu gestalten. Es soll so wenig Information wie nötig dargestellt werden.
Alle dargestellten Texte (statisch oder dynamisch) werden in der Sprache des Geräts dargestellt.

3.2.2.1.1 PLATTFORM

Es wird auf Windows Phone 8 aufgebaut. Es wird in C# und XAML programmiert. Das UI muss den Anforderungen des Dokuments „DPS Quality Bar (2.0)“ und „DPS UX Bar (5.0)“ gerecht werden.

3.2.2.1.2 FUNKTIONEN

Member Cards:
Der Gast kann seine Member Cards einsehen und diese an der Türe vorweisen. Zur Identifizierung des Members soll entweder NFC verwendet oder ein QR Code angezeigt werden, welcher an der Türe Kontrolliert werden kann.
Weiter kann er für seine Member Cards spezielle Angebote erhalten, welche er in der Member Card Ansicht sieht. Er kann auch weitere Details einsehen, wie die Anzahl Gäste die der Member mitbringen kann oder das Ablaufdatum seiner Member Card.

Special Offers:
Der Gast kann als Member spezielle Angebote erhalten, welche er einlösen kann. Beim Einlösen kann der Kassier dann den Coupon über das Handy einlösen.

Club Ansicht:
In der Club Ansicht werden Informationen über den Club angezeigt. Darunter auch, wie viele Personen diesen Club über Facebook „liken“. Von hier aus kann er direkt zu einer Karten Applikation springen, welche dem Gast helfen soll den schnellsten Weg zum Club zu finden. Die Karten Applikation ist von der Plattform gegeben.

Event Übersicht:
In dieser Ansicht sieht der Gast alle kommenden Events des Clubs. Dabei sieht er, wie viel ihn dieser Event kostet (unter Berücksichtigung der Member Card), wie viele Personen er mitbringen kann, wie viele Likes der Event hat und wie viele seiner Freunde aus Facebook am Event teilnehmen.
Es soll möglich sein zu unterscheiden zwischen öffentlichen und „members only“ Events.

Event Ansicht:
In dieser Ansicht sieht der Gast, wann und wo der Event stattfindet, sowie einen Slogan. Hier kann der Benutzer mitteilen (auch über Facebook), dass er an diesem Event sein wird, er kann ihn „Liken“ oder mit Freunden teilen.

Event Details Ansicht:
In der Details Ansicht sieht er zusätzliche Informationen zum Event, wie Beschreibung und Updates.

Ticket Übersicht:
Hier sieht der Benutzer alle Tickets, die noch in der Zukunft liegen als Übersicht.

Ticket Ansicht:
Das Ticket wird je nach Funktionsumfang als QR Code oder als NFC Tag angeboten. Wenn der Club, sowie das Gerät über NFC verfügen, wird diese Funktion verwendet. Als Fall Back werden in allen anderen Fällen QR Codes dargestellt.

Suche:
Es soll nach folgenden Kriterien nach Events gesucht werden können:

- Typ: Was für Musik an diesem Event gespielt wird
- In welchem Zeitraum der Event stattfindet
- In welcher Stadt der Event ist
- Nach Text im Titel, Club oder Beschreibung (Gewichtung geschieht auf Server)

Es soll nach folgenden Kriterien nach Clubs gesucht werden können:

- Typ: Was für Musik in diesem Club gespielt wird
- In welcher Stadt der Club ist
- Nach Text im Titel, Club oder Beschreibung (Gewichtung geschieht auf Server)

Kaufen:
Tickets können über PayPal gekauft werden.

Resultatskarte:
Suchresultate sollen auf einer Karte angezeigt werden. Dabei wird die Ansicht so eingeschränkt, dass gerade alle Resultate auf der Karte sind. Die Resultate werden als Pins dargestellt.

Resultatsliste:
Suchresultate werden als Liste mit wenigen Details angezeigt.

Profil:
Hier kann der Gast seine persönlichen Daten und Einstellungen einsehen und verändern.

NFC:
Über NFC können Tickets und Member Cards verifiziert werden. Der Gast startet die App und hält das Smartphone beim Eingang an das Tablet/Smartphone des Clubs. Automatisch wird das Ticket des Benutzers übermittelt und über den Door Client des Clubs an den Server übermittelt, welcher das Ticket überprüft.

Notifikationen:
Der Club kann Neuigkeiten über den Club oder seine Events herausgeben. Der Client empfängt diese und zeigt sie dem Benutzer an.

3.2.2.2 DOOR APP

Die Door App wird vom Club Personal bedient und wird am Eingang des Clubs verwendet. Er soll alle nötigen Funktionen zur Verfügung stellen, welche am Eingang verwendet werden. Der Fokus soll auf Funktionalität gelegt werden und nicht auf UI Design, da dieser Teil einen frühen Prototyp darstellen soll.

3.2.2.2.1 PLATTFORM

Es wird ein Windows 8 Tablet (RT/x86) verwendet. Es wird mit C#, XAML und WinRT programmiert.

3.2.2.2.2 FUNKTIONEN

Member Suche:
Der Tür Manager kann nach Gästen auf der Gästeliste per Name suchen. Die Resultate werden vorab angezeigt, dabei wird auch gleich ersichtlich wie viel Kontingent der Gast noch hat.

Ticket/Member Card überprüfen per NFC:
Der Tür Manager überprüft die Identität des Gasts, indem der Gast sein Smartphone an das Eingangs Tablet hält. Dem Tür Manager werden darauf sofort die relevanten Gast Information dargestellt.

Ticket/Member Card überprüfen per QR:
Der Tür Manager überprüft die Identität des Gasts, indem der Gast sein Smartphone mit dem geöffneten Ticket vor die Kamera hält. Das Tür Tablet liest den Code ein und identifiziert das Ticket mit dem Server.

Statistik:
Der Tür Manager liest auf dem Tablet die relevanten Daten des aktuellen Zustands des Clubs ab. Dazu gehören:

- Anzahl Gäste (Männer/Frauen/Total)
  Entwicklung Gäste (Männer/Frauen/Total)

4 ERGEBENISSE

4.1 ARCHITEKTUR

Venue besteht aus einer Server Komponente, welche ein REST Interface zur Datenabfrage bereitstellt. Zur Authentifizierung wird ein Azure Access Control Service verwendet, dazu mehr im Kapitel 4.3.4. Es gibt einen Web Client, welcher zur Administration des Clubs verwendet wird. Er ist nicht Teil dieser Arbeit, deswegen wird er nicht weiter beschrieben.

Der Door Client ist eine Windows 8 Tablet App, welche an der Türe des Clubs verwendet wird. Er besitzt eine konstante Funkverbindung zum Internet. Er spricht direkt mit der REST Schnittstelle des Servers.

Der Guest Client ist eine Windows Phone 8 App, welche vom Gast überall in der Welt verwendet werden kann. Er besitzt in regelmässigen Abständen eine Internetverbindung. Er spricht ebenfalls direkt mit der REST Schnittstelle.

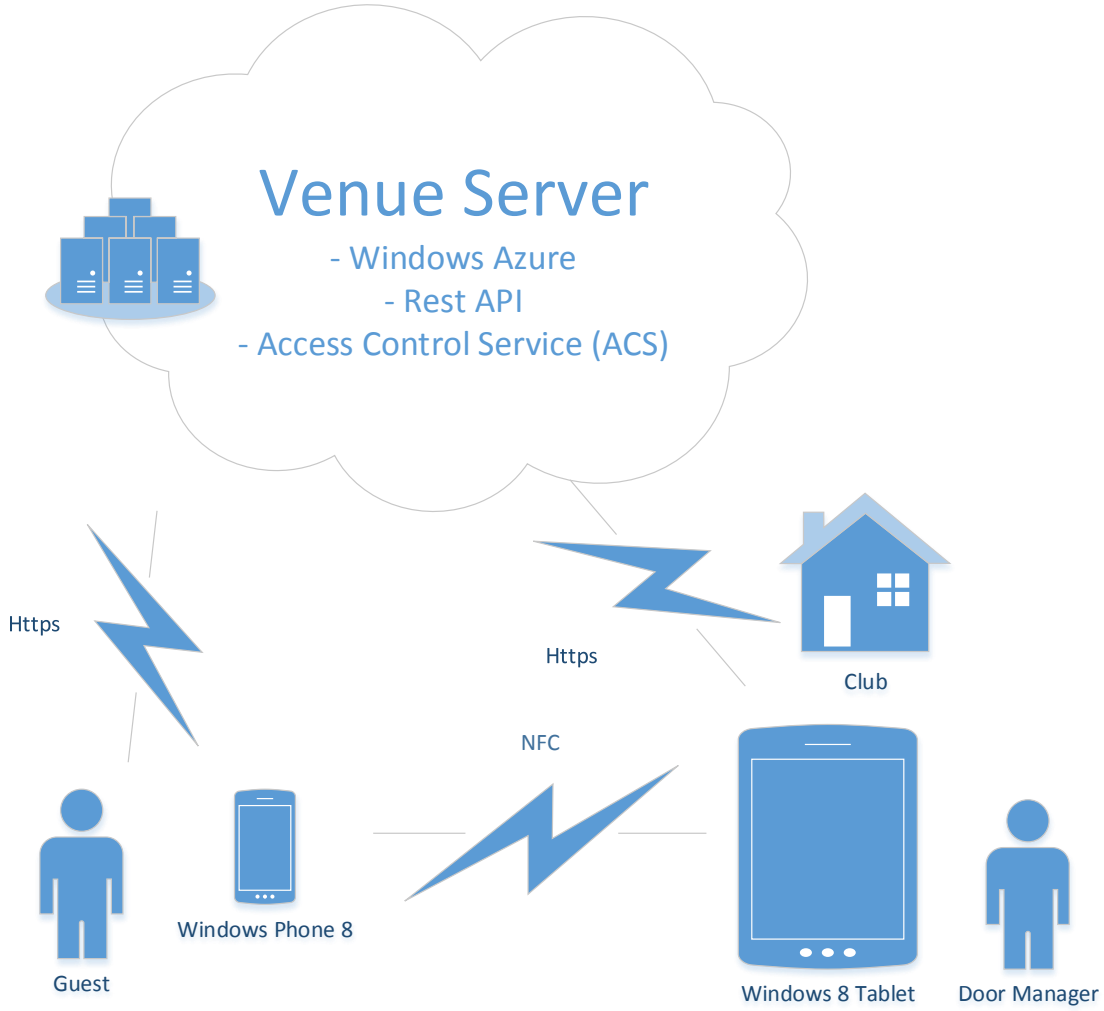


Abbildung 1: Deployment Diagramm

4.1.1 PROJEKT STRUKTUR

Die Projektreferenzen der Venue Clients sind in Abbildung 2 abgebildet. Die oberste Reihe der Hierarchie sind ausführbare Projekte.

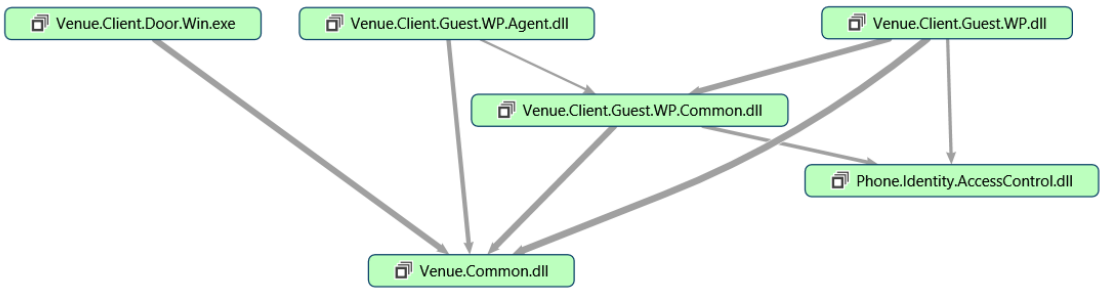


Abbildung 2: Komponenten Diagramm

Venue.Common beinhaltet die abstrahiert Kommunikation zur REST Schnittstelle und die darin verwendeten Datentypen. Es wird von der Windows 8 App und der Windows Phone 8 App verwendet.

Venue.Client.Guest.WP.Common beinhaltet gemeinsam genutzte Dienste für den Windows Phone 8 Client. Es gibt dabei zwei ausführbare Projekte, Venue.Client.Guest.WP.Agent und Venue.Client.Guest.WP. Das Projekt Venue.Client.Door.Win ist die ausführbare Window 8 App.

4.2 WEKRZEUGE

Entwickelt wird auf einem Windows 8 PC mit Visual Studio 2012. In Visual Studio sind die Plugins Resharper 7.1 und T4Toolbox installiert. Es wird das Windows Phone SDK 8.0 verwendet. Als Zeiterfassungssystem wird zu Beginn des Projekts Harvest verwendet, jedoch erfolgte später ein (firmenweiter) Umstieg auf Paymo.Biz. Für Source Control und das Anforderungs-Management wird der Azure Team Foundation Service verwendet, welches eine „Software As A Service“ Lösung für den Team Foundation Server 2012 ist.

Um Abhängigkeiten zu verwalten wurde die Erweiterung NuGet für Visual Studio installiert. Es sorgt dafür, dass alle registrierten Abhängigkeiten heruntergeladen werden, wenn ein Build ausgeführt wird.

4.3 GUEST APP

4.3.1 TECHNOLOGIE

Als Programmiersprache wird C# 5.0 und als Plattform Windows Phone 8.0 verwendet. Das UI wird in Xaml geschrieben.

4.3.2 UI DESIGN

Die in der Ausgangslage beschriebene Vorgabe zum UI Design muss gemäss den Anforderungen überarbeitet werden. Dabei gelten das „DPS UX Bar (5.0)“ und „DPS Quality Bar (2.0)“ Dokument als Richtlinien. Die wichtigste Änderung betrifft den Informationsfluss.

4.3.2.1 INFORMATIONSFLUSS

Der Informationsfluss muss mit der Anforderung der „UX Bar“ übereingehen. Das bedeutet dass Informationen nicht wie in der Vorgabe der Aufgabenstellung dargestellt werden können. Die

Informationshierarchie soll gemäss diesem Dokument so flach als möglich ausfallen. Dafür sind Konzepte wie die Panorama oder Pivot View gedacht, welche die Informationen horizontal auf gleichem Level verteilen. Dabei ist wichtig, dass die Informationen in irgendeiner Art auf die gleiche Stufe gestellt werden können.

Abbildung 3:Beispiel einer Panorama View

Der Informationsfluss sollte nicht tiefer als 5 Stufen werden gemäss „Quality Bar“.

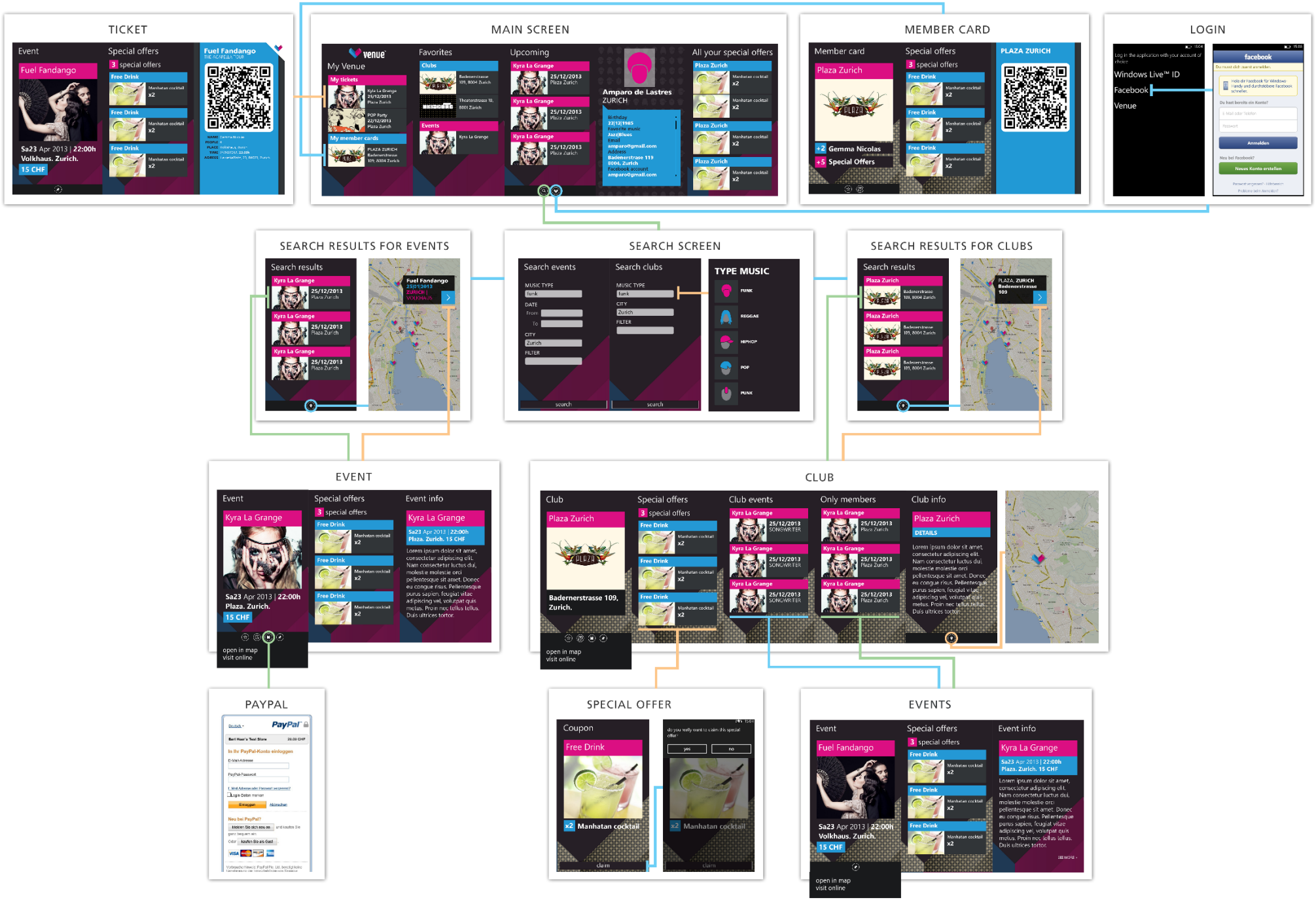


Abbildung 4: Information Map

Auf der Einstiegsstufe sehen in Abbildung 4 wir unterschiedliche Informationen, je nachdem ob der Gast sich bereits angemeldet hat oder nicht. Im nicht angemeldeten Zustand wird er zuerst eine Liste von Events sehen, welche in nächster Zukunft stattfinden werden. Im angemeldeten Zustand wird er zusätzlich seine Tickets und Member Cards, sein Profil sowie seine Favoriten sehen. Die verschiedenen Listen werden nur angezeigt, wenn es auch Items gibt. In der Favoriten Liste kann der Benutzer über einen Long Press direkt Favoriten entfernen (siehe Abbildung 5).

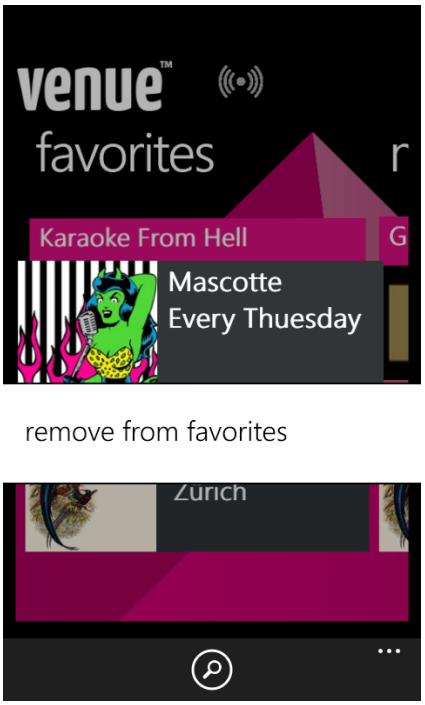


Abbildung 5: Kontext Menu zum entfernen von Favoriten

Das Kontext Menu ist eine Funktion des Windows Phone Toolkit und bietet bereits anschauliche Animationen.

Mit einem Tap auf die verschiedenen Items wird auf die jeweilige Detail Page gesprungen. Zurück gelangt man immer über den von Microsoft vorgeschriebenen Hardware Button. Die weiteren Ansichten zeigen verschiedene Informationen über Clubs und ihre Events.

4.3.2.2 VISUELLES DESIGN

Das visuelle Design ist eine Richtlinie für das UI Design, um wiederkehrende Elemente im gleichen Stile darzustellen. In Abbildung 6 sehen wir, wie verschiedene Elemente dargestellt werden können.

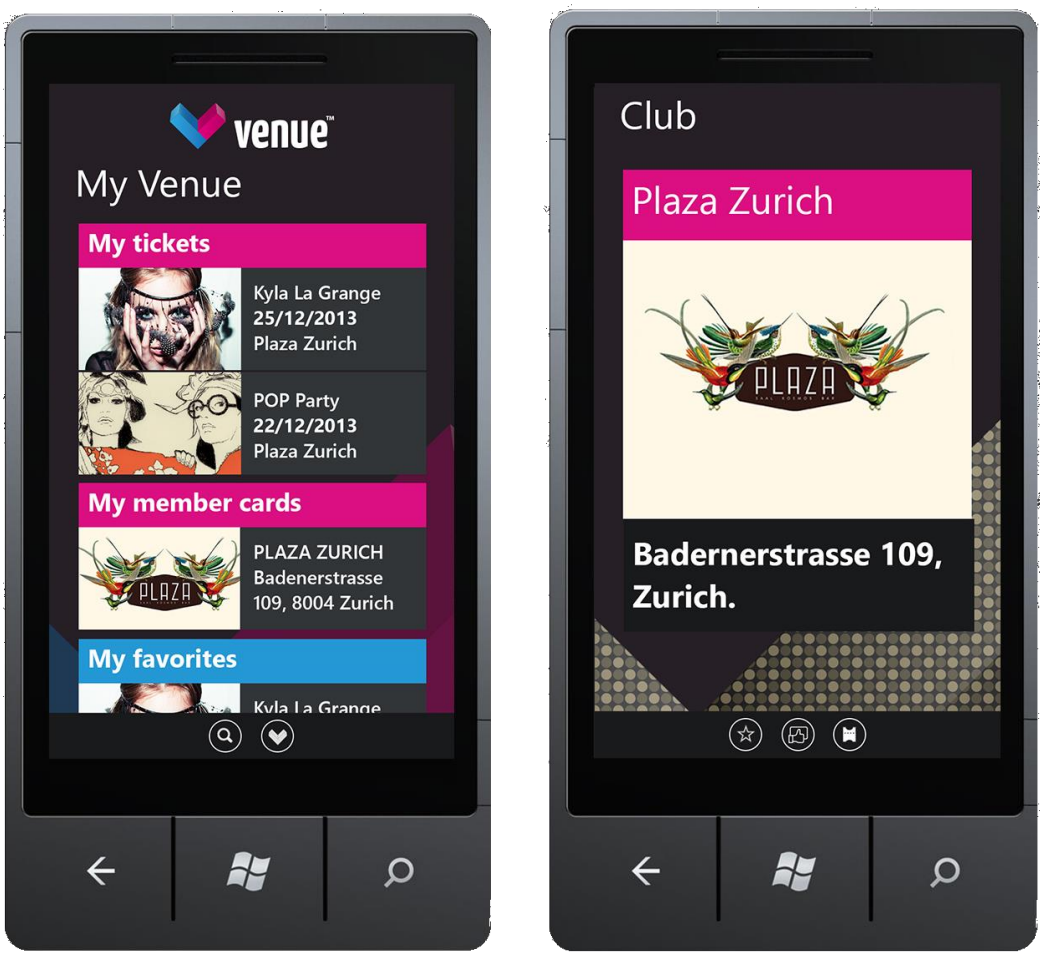


Abbildung 6: Visuelles Design

Ein Kriterium, welches speziell berücksichtig werden musste, ist, dass Buttons wenn möglich immer in die Application Bar (siehe Abbildung 7) angezeigt werden sollen. Die Application Bar soll für den Anwender eines Windows Phone die zentrale Position für Aktionen darstellen. Sie ist jedoch im Vergleich zu bisherigen UI Konzepten eine wenig gewöhnungsbedürftig.

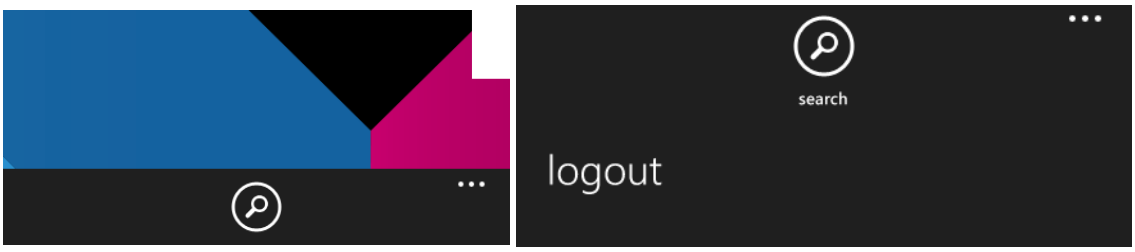


Abbildung 7: ApplicationBar

4.3.3 SERVICES

4.3.3.1 SERVICE LOCATOR

Um Abhängigkeiten von Komponenten untereinander zu verhindern, besonders wenn es um die Thematik von Single Instance Objekten geht, welche oft als Singleton umgesetzt werden, wird in Venue der Service Locator des Patterns & Practices[1] Projekt von Microsoft verwendet. Der Service

[1] Patterns & Practices, (CodePlex 2013)

Locator bietet statische Methoden um einen bestimmten Service anhand eines Interfaces zu finden. Dabei kennt der Aufrufer nicht den implementierenden Typ, sondern nur ein Interface.

Damit der Service Locator jedoch weiss, welchen Typ/Instanz er aufsuchen muss, werden zu Applikationsbeginn die verschiedenen Interfaces mit einer entsprechenden Implementation registriert. Dieser Registrierungsprozess nennt sich Bootstrapper.

Da der ServiceLocator selbst nur eine gemeinsame Schnittstelle definiert, muss noch ein sogenannter ServiceLocatorProvider registriert werden, welcher die Aufgabe hat, alle weiteren Aufrufe des ServiceLocators zu abhandeln. In der .Net Welt gibt es bereits eine Vielzahl von solchen Providern, doch leider kaum welche für Windows Phone. In Venue wird auf den SimpleIoc Provider aus dem MVVM Light[2] Projekt von Laurent Bugnion zurückgegriffen. Leider ist der Provider sehr eingeschärnkt, er kennt zum Beispiel keine verschiedenen Lifecycles Instanzen. Es gibt nur Single Instances, was für dieses Projekt ausreicht.

```
simpleIoc.Register<INavigationManager>(() => new NavigationManager());
```

Abbildung 8: Registrierung bei Bootstrapper

4.3.3.2 ABHÄNGIGKEITEN

Die Services selbst sind wiederum von anderen Services abhängig. Doch da die Services nicht über Dependency Injection[3] instanziert werden, muss jeder Service seine Abhängigkeit selbst beim ServiceLocator anfordern.

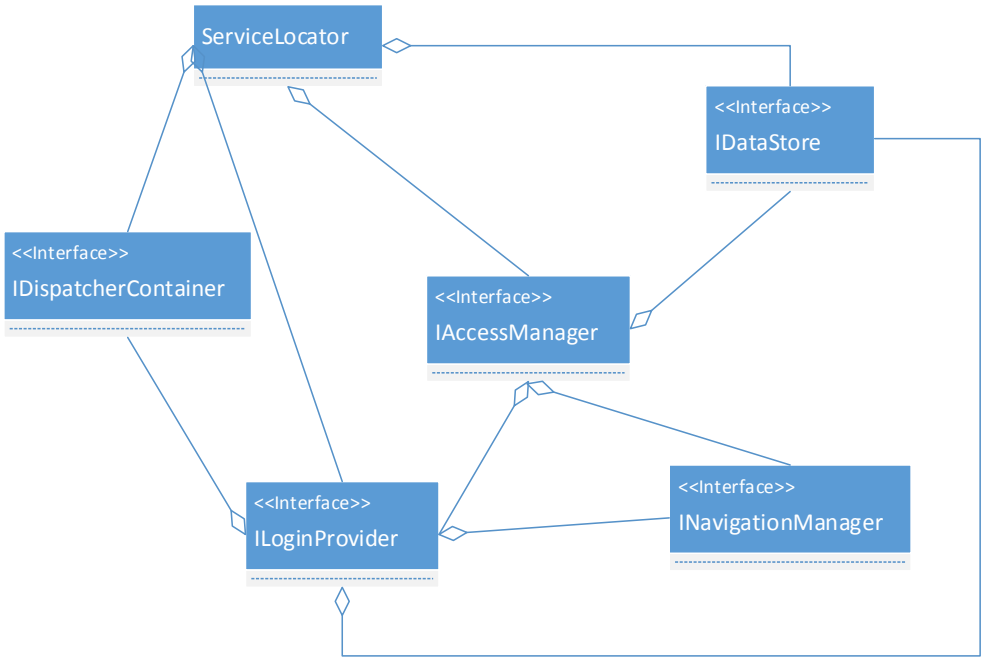


Abbildung 9: Service Abhängigkeiten

[2] MVVM Light, (CodePlex 2013)

[3] Dependency injection, (Wikimedia 2013)

4.3.3.3 NAVIGATIONMANAGER

Der NavigationManager abstrahiert die Navigation zwischen den verschiedenen Seiten. Er ist zustandslos.

```
public interface INavigationManager
{
  void NavigateToClub(Guid clubKey);
  void NavigateToSpecialOffer(Guid specialOfferKey);
  void NavigateToEvent(Guid eventKey);
  void NavigateToMemberCard(Guid memberCardKey);
  void NavigateToTicket(Guid ticketKey);
  void NavigateToPayPal(Guid eventKey);
  void NavigateToTicketUsed(string token);
  void NavigateToTicketFail();
  void NavigateToMainPage();
  void NavigateToLogin();
  void NavigateToMessage(Guid messageKey);
  void GoBack();
}
```

Abbildung 10: INavigationManager Interface

4.3.3.4 GUESTACCESSMANAGER

Der GuestAccessManager abstrahiert die Kommunikation mit dem DoorClient und beinhaltet die Logik zur Einlasskontrolle. Er arbeitet weitgehend Autonom und bietet kaum Funktionalität nach aussen. Er ist zustandsbehaftet.

```
public interface IAccessMananger
{
  bool HasNfc { get; }
}
```

Abbildung 11: IAccessManager Interface

4.3.3.5 DATASTORE

Der DataStore stellt Nutzdaten bereit, bietet die Schnittstelle zur Datenmanipulation und er abstrahiert Caching Funktionen. Er ist zustandsbehaftet.

```
public interface IDataStore
{
  Task<GuestDto> GetCurrentGuest();
  Task<ClubDto> GetClub(Guid clubKey);
  Task<EventDto> GetEvent(Guid eventKey);
  void ClearAll();
  Task<SpecialOfferDto> GetSpecialOffer(Guid parse);
  Task<MemberCardDto> GetMemberCard(Guid parse);
  Task<TicketDto> GetTicket(Guid ticketKey);
  Task<IEnumerable<LocationDto>> GetLocations(string filter);
  Task<IEnumerable<ClubDto>> SearchClubs(string clubName, string location, string
style);
  IEnumerable<ClubDto> CurrentClubSearchResults { get; }
  IEnumerable<EventDto> CurrentEventSearchResults { get; }
  Task<TicketDto> BuyTicket(Guid eventKey);
  Task ClaimSpecialOffer(Guid specialOfferKey);
  Task<IEnumerable<EventDto>> GetNextEvents();
  Task ToggleClubFavorite(Guid clubKey, bool favorite);
  Task ToggleEventFavorite(Guid eventKey, bool favorite);
  Task<IEnumerable<StyleDto>> GetStyles();
  Geoposition GetCurrentLocation();
  MapAddress GetCurrentAddress();
  Task RemoveMessage(Guid message);
  Task<MessageDto> GetMessage(Guid messageKey);
  event EventHandler UpdateRequired;
}
```

Abbildung 12: IDataStore Interface

4.3.3.6 LOGINPROVIDER

Der Login Provider bietet die Abstraktion zur Authentifizierung. Er kann ein Login auslösen und sich ein Authentifizierungscookie speichern. Er ist zustandslos.

```
  public interface ILoginProvider
  {
    bool IsLoggedIn { get; }
    SimpleWebTokenStore WebTokenStore { get; }
    void ForceLogin();
    event EventHandler<EventArgs> Login;
    event EventHandler<EventArgs> Logout;
    void AddAuthentication(HttpWebRequest webRequest);
    void LogOut();
    void FinishLogin();
    bool CheckUrl(Uri responseUri);
  }
```

Abbildung 13: ILoginProvider Interface

4.3.3.7 DISPATCHERCONTAINER

Der DispatcherContainer bietet Zugang zum UI Thread. Damit andere Services direkt Objekte des UI Threads verwenden können muss zuerst über den Dispatcher ein Delegate abgesetzt werden, welcher dann auf dem UI Thread ausgeführt wird. Der DispatcherContainer ist zustandsbehaftet.

```
public interface IDispatcherContainer
{
  Dispatcher Dispatcher { get; }
}
```

Abbildung 14: IDispatcherContainer Interface

4.3.4 AUTHENTICATION

4.3.4.1 AZURE AUTHENTICATION CONTROL SERVICE

Azure Authentication Control Service, kurz ACS, ist ein Authentifizierungssystem, welches als Dienst von Microsoft angeboten wird. Es ist Teil von Windows Azure, dem Cloud Computing Angebot von Microsoft. Das Ziel des ACS ist es, eine Zentrale Authentifizierungsstelle anzubieten, welche verschiedenste Identity Provider[4] (IP) abstrahiert. Dies ermöglicht einer Applikation, auch Relying Party (RP) genannt, verschiedene IP zu unterstützen, ohne dabei für alle diese Anbieter eine eigene Implementation entwickeln zu müssen.

Die RP kennt nur eine Authentifizierungsstelle, den ACS. Er bindet diesen z.B. über den WS-Federations Standard an sich. ACS bietet nur eine passive Authentifizierung an, d. h. es kann kein Login mit Benutzername und Passwort explizit oder programmatisch erfolgen. Auch wird der ACS nie diese Informationen erhalten, denn der ACS ist selbst kein IP. Die passive Authentifizierung erfolgt über HTML Seiten, bei welchen am Schluss ein Security Token in Form eines Cookies ausgestellt wird. Das Format des Tokens ist von der Applikation wählbar. Zur Verfügung stehen:

- Simple Web Token (SWT)
- OAuth Wrap
- OAuth 2.0
- Saml 1.0
- Saml 2.0

Venue verwendet das SWT Token, da es einfacher lesbar ist als z.B. das Saml Token.

ACS kennt vorkonfigurierte IP, darunter Facebook, Windows Live, Google und Yahoo. Es lässt sich auch relativ einfach ein Windows Active Directory einbinden. Wenn die RP einen eigenen IP stellen möchte um den Nutzer nicht zur Registrierung bei den anderen IPs zu zwingen, kann ein eigener IP über WS-Federations oder OpenID beim ACS konfiguriert werden.

Venue besitzt einen eigenen IP, dieser ist jedoch zum Zeitpunkt dieser Arbeit noch nicht fertig implementiert, jedoch bereits beim ACS konfiguriert.

[4] Passive Authentication for ASP.NET with WIF, (Microsoft 2013)

4.3.4.2 ABLAUF DER TOKEN AUSSTELLUNG

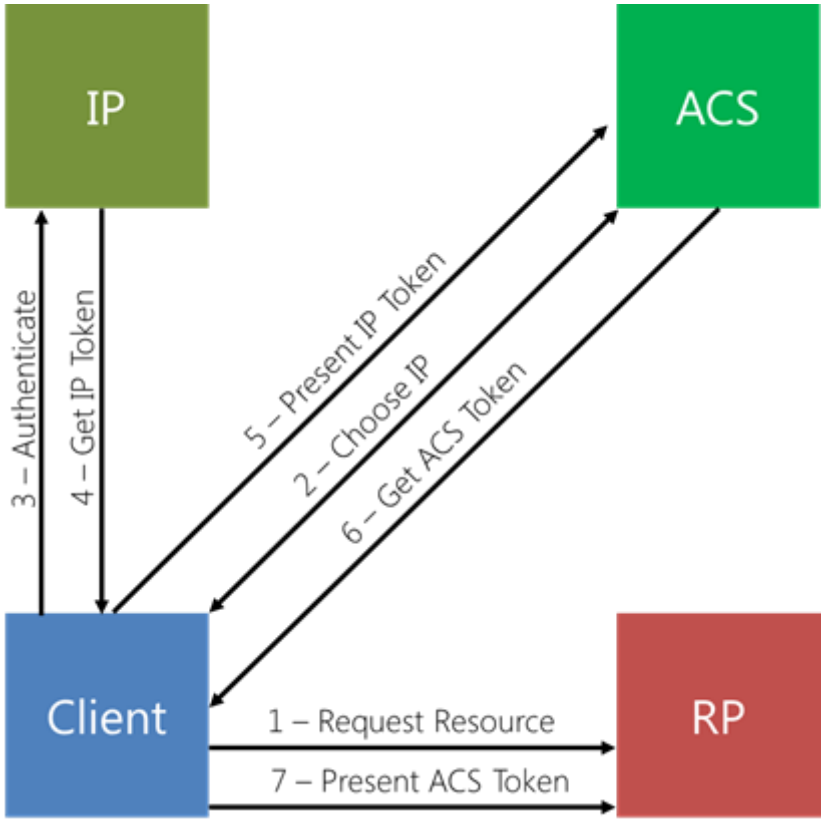


Abbildung 15: Ablauf ACS Authentifizierung, (Microsoft 2013)

Der Ablauf der Token Ausstellung ist folgender:

1. Der Client fragt nach einer Ressource auf dem RP.
2. Da die Anfrage noch nicht authentifiziert ist wird der RP den Client auf den ACS verweisen. Darauf liefert der ACS dem Benutzer eine Auswahl möglicher IPs, worauf der Benutzer einen selektiert.
3. Nun wird der Client auf die Login Seite des IP weitergeleitet. Dort meldet er sich an oder registriert sich neu. Wie das Login aussieht oder funktioniert ist dem IP überlassen. Natürlich werden so auch die gerade aktuellen Zwei-Phasen-Authentifizierungen unterstützt.
4. Nachdem der Benutzer sich angemeldet hat wird ein Token vom IP ausgestellt. Dieses Token genügt jedoch noch nicht um authentifiziert zu sein, denn es wurde nicht vom ACS ausgestellt und somit könnte die RP nichts damit anfangen.
5. Danach wird der Benutzer vom IP an den ACS zurückverwiesen, worauf er das ausgestellte Token mitschickt.
6. Der ACS überprüft dieses Token aufgrund der Konfigurationen und stellt ein neues Token, basierend auf dem Token des IP zusammen. Der ACS kann nun anhand dieses Tokens den Benutzer wieder erkennen.
7. Der ACS leitet den Client auf den RP zurück, wobei der Client nun das Token des ACS mitsendet. Der RP kann nun dieses Token auf dem ACS überprüfen und stellt fest, dass der Client nun authentifiziert ist. Er liefert darauf die gewünschte Ressource.

Im entfernteren Sinne handelt es sich beim ACS um eine SingleSignOn Lösung. Denn wenn man einmal bei einem IP eingeloggt ist, kann, soweit unterstützt, dieses Token für verschiedene Dienste weiterverwendet werden. Wenn man bereits beim IP authentifiziert ist entfällt Schritt 3, welcher für den Benutzer die grösste Interaktion bedeutet.

4.3.4.3 ABLAUF LOGIN PROVIDER

Venue abstrahiert für den Anmeldevorgang einen Service, den LoginProvider. Wenn ein Login erwünscht wird, kann die ForceLogin Methode dieses Services aufgerufen werden. Dieser startet dann die Login.xaml Page. Zuerst wird die Auswahl der IP's geladen und dargestellt danach ein über ein Web View Control das entsprechende Login dargestellt. Nach erfolgreichem Login vermeldet der Login Provider über einen Event, ob das Login erfolgreich war oder nicht. Wenn ein Logout stattgefunden hat wird ebenfalls ein entsprechender Event ausgelöst.

Nachdem ein Login ausgeführt wird, wir das Token im Isolated Storage abgespeichert.

4.3.4.4 IDENTITY PROVIDER SELECTION

Da es sich bei der Gäste App nicht um eine Web Applikation handelt, wird die Authentifizierung nicht mit einer Weiterleitung zum ACS gelingen. Um ein Login durchzuführen muss also ein WebView[5] Element verwendet werden, in welchem man am Schluss das Cookie abgreift und dann alle weiteren Server abfragen mit diesem Cookie befüllt.

Jedoch ist der Selektionsbildschirm bei der Auswahl der IP nicht sehr anschaulich gestaltet und weit von einem nativen Aussehen entfernt. Deswegen wird in Venue die Auswahl der IP explizit abgefragt, worauf man eine strukturierte Liste der IP's erhält. Diese Liste kann dann mit einem nativen Aussehen dargestellt werden. Nachdem der Benutzer dann seinen IP ausgewählt hat, wird schlussendlich doch noch ein WebView Element angezeigt, wo dann der jeweilige Login Screen des IP angezeigt wird.

Microsoft bietet in den Azure Mobile Services[6] bereits ein solches Control an. Da jedoch nur diese kleine Komponente aus den doch grossen Azure Mobile Services gebraucht wird, wurde dieser Source Code aus dem CodePlex Repository kopiert und in der Solution eingefügt.

4.3.5 DATENZUGRIFF

Der Datenzugriff für diese Applikation wird über eine Single Instance (siehe Kapitel 13) bereitgestellt. Es bietet eine höherwertige Abstraktion auf die Daten als es die Proxy Klasse tut, welche nur den direkten Server Aufruf abstrahiert. Die Business Logik kann sich dieser Schnittstelle bedienen ohne sich Gedanken über Lebensdauer, Herkunft und Aufbereitung der Daten machen zu müssen.

4.3.5.1 PORTABLE ASSEMBLY[7]

4.3.5.1.1 ERKLÄRUNG

Von der Microsoft .Net Technologie gibt es schon seit geraumer Zeit verschiedene Ableger wie Silverlight, Windows Phone, XBox oder WinRT. Diese Ableger sind fast identisch zueinander, doch aufgrund von Einschränkungen der verschiedenen Plattformen sind sie nicht ganz identisch. Zum Beispiel gibt es auf Silverlight den Emit Namespace nicht, welcher zur Laufzeit Code Generierung ermöglicht. Somit muss Code für den Desktop nicht zwangsläufig auch für Silverlight kompilieren.

In früheren Tagen musste man dieselben Klassen Dateien in mehreren Projekten referenzieren, um Interoperabilität zu gewährleisten. Später wurde dann das Portable Assembly eingeführt, welche erlaubt, das gleiche Assembly auf mehreren Plattformen zu verwenden. Das Tooling dazu war jedoch immer ein wenig umständlich. In den neuesten Versionen des Visual Studios ist es nun möglich

[5] WebView Control, (Microsoft 2013)

[6] Azure Mobile Services, (Microsoft 2013)

[7] Cross-Platform Development with the .NET Framework (Microsoft 2013)

anzugeben, welche Zielplattformen unterstützt werden sollen und Visual Studio schränkt dann automatisch die Programmierunterstützung für ein gemeinsames Sub Set an Funktionen ein.

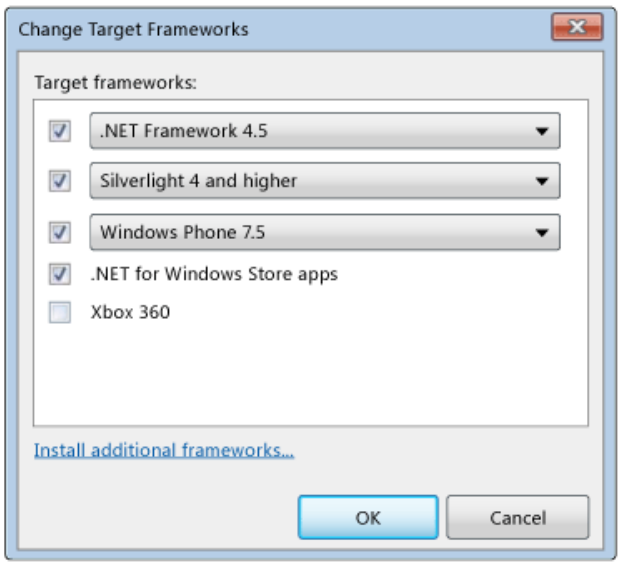


Abbildung 16: Auswahl der Ziel Plattformen für Portable Assemblies

4.3.5.1.2 EINSATZ IN VENUE

Um in Venue Code in der Door und der Guest App zu teilen wird ein Portable Assembly verwendet. Dort können gemeinsame Klassen abgelegt werden wie zum Beispiel der Proxy und seine Datentypen.

4.3.5.2 PROXY GENERATION

T4 Template[8] ist ein Code Generierungssprache, welche in Visual Studio eingebettet ist. Es ermöglicht vor oder nach der Kompilierung, Code oder andere strukturierte Dateien aus anderen Quellen zu erzeugen. Das Sprachkonzept erinnert stark an Asp.Net, in welchem man statischen Text mit dynamischem Code erweitert.

```
<#
foreach(var type in types.Where(d=>d.IsEnum))
{#>
public enum <#=type.Name#>
{
        <#
        foreach(var enumMember in type.GetEnumNames())
        {#>
<#=enumMember#>,
        <#}#>
}
<#}
```

Abbildung 17: T4 Template Beispiel Code

In diesem Projekt wird die T4 Template Sprache dazu verwendet, eine clientseitige typisierte Schnittstelle gegenüber dem Server zu generieren. Ansonsten müsste für jede Operation auf der Server Schnittstelle auf Client Seite eine entsprechende Abstraktion von Hand geschrieben werden (Siehe Abbildung 18).

[8] Code Generation and T4 Text Templates (Microsoft 2013)

```
var urlBuilder = new StringBuilder();
urlBuilder.Append("?eventKey=" + eventKey);
urlBuilder.Append("&gender=" + gender);
var serviceClient = new ServiceClient<Int32?>(_serviceUrl + "Door/Increment" +
urlBuilder.ToString(), "GET");
return serviceClient.GetResult();
```

Abbildung 18:Snippet zur manuellen Proxy Generierung

Jedes Mal, wenn nun die Schnittstelle des Servers angepasst wird, was besonders in der frühen Entwicklungsphase noch des Öfteren geschieht, wird der Fehler erst zur Laufzeit bemerkt. Um die viele Handarbeit zu eliminieren wird hier nun ein T4 Template verwendet. Dieses Template liest aus dem Venue.Web Assembly, dass die REST Schnittstelle definiert, welche Operation mit welchen Parametern verfügbar ist und erstellt aufgrund dessen den Http Aufruf, gemäss Abbildung 18. Die Proxy Datentypen werden ebenfalls aus dem Venue.Web Assembly ausgelesen.

Um möglichst wenig Logik in den Template Dateien zu haben, wurde ein Projekt erstellt, welche eine Hilfsklasse implementiert. Diese übernimmt das Auslesen der Venue.Web dll und stellt die Daten in vereinfachter Form bereit. Die Hilfsklasse sucht im Assembly per Reflection nach WebApi Controllern und liest alle möglichen Aufrufe aus. Dieses Projekt ist so lose gekoppelt, dass man es für andere WebApi Projekte wiederverwenden könnte.

4.3.5.3 CACHING

Die App sollte für den Benutzer so wenig warten müssen als möglich und sie sollte wenn möglich auch offline verfügbar sein, da in der Realität nicht immer von einer Internetverbindung ausgegangen werden kann. Um dies zu erreichen, müssen Daten vom Server lokal und redundant gespeichert werden, sogenanntes Caching.

Für einen Cache kommen verschiedene Zwischenspeicher in Frage. Der schnellste ist der Arbeitsspeicher. Er ist sehr schnell, ist jedoch nicht persistent, d.h. beim nächsten Start der App ist der Speicher verloren. Um einen persistenten Speicher zu haben, bietet sich der Isolated Storage[9] an, welcher die Daten auf dem geräteinternen Flashspeicher ablegt. Dieser ist jedoch deutlich langsamer als der Arbeitsspeicher.

Um nun kurze Wartezeiten und offline Fähigkeit zu verbinden, muss ein zwei stufiges Cache Verfahren verwendet werden, indem der Arbeitsspeicher und der Isolated Storage verwendet wird. Typische Caching Verfahren haben immer denselben Ablauf. Zuerst wird überprüft, ob die gewünschte Information im Cache vorhanden ist. Wenn ja, wird diese sofort zurück geliefert, wenn nein muss die Information geladen und/oder zusammengestellt und danach im Cache gespeichert werden (siehe Abbildung 19).

[9] Quickstart: Working with files and folders in Windows Phone 8 (Microsoft 2013)

Abbildung 19: Klassisches Caching Verfahren

Doch hier handelt es sich um einen mehrstufigen Cache Ansatz, bei welchem mehrere Caches überprüft werden, bevor eine Entität vom Server geladen würde (siehe Abbildung 19).

Abbildung 20: Multi Level Caching

Dieses Verhalten ist wünschenswert bei Daten, die nicht sehr dynamisch sind und nach der Erstellung kaum mehr ändern. Daten, welche jedoch häufiger ändern, sollten immer so aktuell wie möglich gehalten werden. Somit wäre dieses Verhalten nicht mehr wünschenswert, denn es lädt immer die zuvor geladene Entität. Besteht jedoch kein Kontakt zum Server sollte dennoch die zwischengespeicherte Version verwendet werden, um Offline Funktionalität zu gewährleisten. Deswegen wird in dieser App zwischen zwei Cache Modi unterschieden: CacheFirst und OnlineFirst. CacheFirst entspricht dem Verfahren aus Abbildung 20. Für OnlineFirst ändert sich das Verfahren ein wenig. Zuerst wird überprüft, ob eine Internetverbindung vom Gerät zur Verfügung gestellt wird. Wenn ja, lade die Entität und speichere die neue Version der Entität in die Caches. Wenn keine Verbindung vorhanden ist oder das Laden der Daten fehlschlägt, versuche die Entität aus dem Cache zu laden (siehe Abbildung 21).

Abbildung 21: OnlineFirst Cache Verfahren

4.3.6 VIEW MODELS

Eine native Windows Phone 8 Applikation besitzt ein empfohlenes Designpattern, welches sich „Model View ViewModel“[10] nennt, kurz MVVM. Dabei wird versucht, den ganzen Zustand des UI über ein an die View angepasstes Model abzubilden, um die tatsächliche Abbildung von der UI Logik zu trennen. Weitere Informationen zum MVVM können über die angegeben Quellen bezogen werden.

Abbildung 22: MVVM

4.3.6.1 PAGE VIEW MODELS

Jede Windows Phone Page besitzt ein ViewModel, welches nicht direkt eine Entität, vielmehr die Seite selbst beschreibt. Es übernimmt die folgenden Aufgaben:

- Laden der Daten
- Bereitstellen datenbezogener ViewModels
- View-spezifischen Zustand (Sichtbarkeiten etc.)
- Benutzerinteraktionen verarbeiten

Um diese Aufgaben zu lösen wird in Venue eine gemeinsame Klasse PageViewModel verwendet, welche die Grundlagen für all diese Aufgaben besitzt, damit in den jeweilig erbenden Klassen sich so wenig Logik als möglich befindet.

Das PageViewModel wird immer im Xaml erzeugt und direkt dem DataContext Property der Page zugewiesen (siehe Abbildung 23). Dies hat den Vorteil, dass zum Zeitpunkt des Erstellens des UI Codes der Typ im DataContext bekannt und stark strukturiert ist. Dies ermöglicht dem Tooling bei späteren Data Bindings Syntaxvorschläge und Korrekturen einzubringen. Um im CodeBehind der Xaml Dateien nun auf dieses ViewModel zugreifen zu können, wird ein typisiertes Zugangsproperty erstellt (siehe Abbildung 24).

```
<phone:PhoneApplicationPage.DataContext>
    <viewModels:ClubPageVm x:Name="Vm" />
</phone:PhoneApplicationPage.DataContext>
```

Abbildung 23: PageViewModel Erzeugung

```
public ClubPageVm ViewModel
{
    get { return this.DataContext as ClubPageVm; }
}
```

Abbildung 24: PageVIewModel Zugriff

4.3.6.2 DATA VIEW MODELS

Ein Data View Model beschreibt einen Adapter für eine Entität. Dieses ViewModel erweitert eine Entität um View spezifische Eigenschaften. So kann zum Beispiel ein Datum einer Entität auf

[10] Implementing the Model-View-ViewModel pattern in a Windows Phone app (Microsoft 2013)

verschiedene Weisen dargestellt werden. Für jede Art und Weise wird nun ein Property erstellt, welches beim Zugriff aufgrund der Daten der ursprünglichen Entität die gewünschte Ausgabe für die Darstellung ableitet (siehe Beispiel in Abbildung 25).

```
public string DisplayHours
{
   get
   {
     return _eventDto.StartTime.ToShortTimeString() + "h";
   }
}
public string DisplayDate
{
   get
   {
     var first = _eventDto.StartTime.ToString("R").Split(':').First();
     return first.Substring(0, first.Length - 3);
   }
}
```

Abbildung 25: Beispiel von View Properties

4.3.7 VIEWS

Das User Interface in einer nativen Windows Phone 8 App wird in Xaml[11] geschrieben, einer deklarativen UI Sprache. Weitere Information zu Xaml sind in den Quellen zu finden.

In Venue werden die Views mit den Standarderweiterungen „Windows Phone Toolkit[12]“ entwickelt. Es werden keine speziellen Frameworks oder Bibliotheken verwendet und deswegen werden hier nur jene Aspekte der Views beschrieben, die über die Standard Konzepte hinausgehen.

4.3.7.1 NAVIGATION

Windows Phone 8 gliedert sein UI in sogenannte Pages. Es gilt pro Page eine Xaml Datei. Der Projekt-relative Pfad zum Xaml gilt, ähnlich wie im Http, als URL. Und wie im Http ist es auch möglich, aufgrund einer URL von Page zu Page zu navigieren. Um Parameter, einer Page zu übergeben verwendet man die im HTML übliche Schreibweise www.contoso.com?param1=value&param2=value.

In Venue wird die Klasse NavigationManager als Single Instance registriert, welche die URLs und dessen mögliche Parameter in Funktionsaufrufen abstrahiert (siehe Beispiel Abbildung 26).

```
public void NavigateToClub(Guid clubKey)
{
  _rootFrame.Navigate(new Uri("/Views/Club.xaml?clubKey=" + clubKey.ToString(),
UriKind.Relative));
}
```

Abbildung 26: Navigations Methode

Die übergebenen Parameter können dann in der neu gestarteten Page ausgelesen und abgearbeitet werden.

[11] XAML Overview (WPF), (Microsoft 2013)

[12] Windows Phone Toolkit, (Microsoft 2013)

4.3.7.2 SUCHRESULTATSKARTE

Um die Suchresultate auf einer Karte darzustellen wurde das Standard Map Control aus der Windows Phone Library verwendet. Da dieses jedoch noch kein vernünftiges Databinding unterstützt, gibt es im Windows Phone Toolkit eine Erweiterung dafür, damit die Pushpins an eine Liste von Suchresultaten gebunden werden können. Doch selbst dies ist durch ein paar Einschränkungen im MapControl nicht wie gewöhnlich möglich. Das Anhängen der Items geschieht im CodeBehind.

Abbildung 27: Map Ansicht

Die Zentrierung der Karte auf den idealen Ausschnitt geschieht leider nicht automatisch und es gibt leider auch keine Standard Implementation dafür. Hierfür musste eine wenig sphärische Trigonometrie verwendet werden, um einerseits den idealen Zoom Level zu erhalten und andererseits das ideale Zentrum, damit alle PushPins noch sichtbar sind.

$$\text{ZoomLevelHorizontal} = \frac{\log\left(\frac{360}{(256 * ScreenWidth)}\right)}{\log 2}$$

Abbildung 28: Berechnung des optimalen Zoom Levels Horizontal

$$\text{ZoomLevelVertical} = \frac{\log\left(\frac{180}{(256 * ScreenHeight)}\right)}{\log 2}$$

Abbildung 29: Berechnung des optimalen Zoom Levels Vertikal

Schlussendlich muss der kleinere der beiden Werte gewählt werden, da sonst Resultate verloren gehen könnten. Das Zentrum hingegen ist einfacher zu berechnen, da man nur das Mittel zwischen

dem Maximum und dem Minimum nehmen muss, um für eine Dimension den zentralen Wert zu erhalten.

4.3.8 IMAGECACHE

Der eingeführte Daten Cache behandelt nur Entitäten. Da Venue jedoch sehr viele dynamische Bilder anzeigt, muss auch ein entsprechender Cache für das Bildmaterial erstellt werden. Wir haben dabei bei jedem Bild die URL als Voraussetzung. Diese URL lässt sich direkt im Xaml mit DataBinding laden. Dies ist eine sehr komfortable Lösung, da der Bild Download von der UI Komponente übernommen wird. Wenn jedoch keine Internetverbindung besteht kann das Bild nicht geladen werden.

Deswegen wurde hier ein Binding Converter[13] verwendet, welcher für uns das Caching von Bildern übernimmt. Dazu wird ebenfalls der Isolated Storage verwendet. Da wir von den Bildern nicht annehmen, dass sie ändern, werden diese immer vom Cache aufgerufen. Es ist dabei nicht möglich, eine URL auf den Isolated Storage zu setzen, deswegen muss der Converter immer einen BinaryStream öffnen und das Bild anhand des Streams initialisieren. Diese Implementierung sieht auch ein Fallback Bild vor, wenn es nicht möglich ist ein Bild online oder vom Cache zu laden. Dieses DefaultImage kann dem BindingConverter als Parameter im Xaml mitgegeben werden. Der genaue Ablauf ist in Abbildung 30 ersichtlich.

[13] Data binding overview (Microsoft 2013)

Abbildung 30: ImageCache Ablauf

In der Anwendung ist der Converter denkbar einfach, siehe dazu Abbildung 31.

```
<Image Source="{Binding Club.ImageLarge, Converter={StaticResource
CacheImageConverter}, ConverterParameter='/Assets/Images/ClubDefault.png'}" />
```

Abbildung 31: Nutzung des CacheImageConverter

4.3.9 PAYPAL ZAHLUNG

Um dem Gast zu ermöglichen, Tickets zu kaufen, erfordert es folgende Anforderungen:

- Der Gast muss mit seinem PayPal Account bezahlen können.
- Der Club muss das Geld auf seinen PayPal Account erhalten.
- Venue darf das Geld nie in seinem Besitz haben.
- Venue muss jederzeit überprüfen können, wie der Status einer Transaktion ist.
- Venue darf keinen Zugang zum PayPal Account des Clubs oder des Gastes haben

Um diese Anforderungen erfüllen zu können, kann nicht das Direct Payment API[14] von PayPal verwendet werden. Es muss dazu das Adaptive Payment API[15] verwendet werden, welches jedoch einige zusätzliche Schritte benötigt.

[14] Direct Payments API, (PayPal 2013)

[15] Adaptive Payments API, (PayPal 2013)

Abbildung 32: PayPal Adaptive Payments

Der Ablauf lautet wie folgt (ergänzend zu Abbildung 32):

1. Der Gast startet den Kaufvorgang, dabei wird vom Client ein entsprechender Request an den Server geschickt.
2. Der Server errechnet den Preis und startet über einen geschützten Server Kanal bei PayPal eine Zahlung
3. Als Antwort auf diesen Request erhält der Venue Server einen PaymentKey, welcher zur Identifikation der Zahlung dient.
4. Der Venue Server wird ein Ticket erstellen und den PaymentKey darin speichern. Das Ticket wird jedoch noch nicht als bezahlt markiert.
5. Der PaymentKey wird als Antwort auf den ursprünglichen Befehl 1 des Clients gesendet.
6. Um dem Gast die Möglichkeit zu geben, eine Zahlung über PayPal zu tätigen, muss nun eine HTML Seite gestartet werden. Dabei wird der PaymentKey als Parameter übergeben.
7. Auf der HTML Seite wird sobald die Seite geladen ist mit JavaScript ein Post Request ausgelöst, welche den Gast auf die PayPal Seite weiterleitet. Der Post Request enthält den PaymentKey.
8. Nun wird der Gast den gewohnten PayPal Instruktionen folgen, die er von anderen PayPal Seiten kennt.
9. Nachdem der Gast den Zahlungsprozess abgeschlossen oder abgebrochen hat, wird eine bestimmte URL aufgerufen, welche vom nativen Code abgefangen und darauf die HTML Seite geschlossen wird.
10. Nun müssen der Client und das Venue System überprüfen, ob die Zahlung unter diesem PaymentKey erfolgreich war. Deswegen wird ein weiterer Request mit dem PaymentKey an den Venue Server abgesetzt.
11. Der Venue Server fragt für diesen PaymentKey den PayPal Server über den Status dieser Zahlung ab.
12. Der Venue Server erhält darauf einen Status.
13. Wenn der Status erfolgreich ist, markiert er das Ticket als bezahlt und gibt dem Client eine entsprechende Antwort.
14. Der Client weiss nun, ob der Gast ein neues Ticket gekauft hat und reagiert entsprechend.

4.3.10 LOKALISIERUNG

Die Lokalisierung der App erfolgt über den klassischen Weg von Resource Assemblies[16], welches die übliche Methode für .Net Applikation seit Beginn des .Net's darstellt. Hier ist jedoch ein Trick anzumerken, wie das Xaml lokalisiert ist. Hier wird ein Singleton erstellt, der jedoch nur über eine Instanz verfügbar gemacht wird und nicht über einen statischen Zugriff (siehe Abbildung 33). Die Klasse AppResources besitzt alle lokalisierten Eigenschaften als Properties.

```
/// <summary>
/// Provides access to string resources.
/// </summary>
public class LocalizedStrings
{
  private static AppResources _localizedResources = new AppResources();
  public AppResources LocalizedResources { get { return _localizedResources; } }
}
```

Abbildung 33: Falscher Singleton

16

In den globalen Ressourcen (App.xaml) wird nun eine benannte Instanz erstellt (siehe Abbildung 34).

```
<local:LocalizedStrings x:Key="LocalizedStrings" />
```

Abbildung 34: Lokalisierung als globale Ressource

Nun kann überall in der Applikation, statischer Text lokalisiert an einen TextBlock gebunden werden (siehe Abbildung 35). Dadurch, dass die alle Typen zum Kompilierzeitpunkt bekannt sind, kann das Tooling allfällige Fehler (falscher Name etc.) frühzeitig erkennen.

```
<TextBlock Text="{Binding LocalizedResources.Loading, Source={StaticResource
LocalizedStrings}}" />
```

Abbildung 35: Lokalisiertes DataBinding

4.3.11 APPBAR COMMANDS

Windows Phone 7/8 enthält ein einheitliches Konzept um Buttons anzuzeigen, welches bereits in die Page Klasse des UI Frameworks eingebaut ist. Jedoch ist das Property ApplicationBar, welches diese Button beinhält, auf der Page Klasse kein DependencyProperty[17], was den Einsatz von DataBinding oder ein Command Pattern[18] verunmöglicht. Dies wäre vor allem sehr wünschenswert für Sichtbarkeiten von Buttons, denn ist schlechtes UI Design einen Login- und Logout-Button gleichzeitig anzuzeigen.

In Venue wird trotzdem eine Command Pattern Lösung verwendet, welche im PageViewModel die Commands definiert und diese im Sinne von DataBinding automatisch auf die View abbildet.

Ein Command muss die Logik definieren, wann er ausgeführt werden darf, was geschehen soll wenn er ausgeführt wird und wie sein Äusseres sein sollte.

```
public class BuyTicketCommand : ApplicationBarCommand
{
  public EventPageVm EventPageVm { get; private set; }

  public BuyTicketCommand(EventPageVm eventPageVm)
  {
    EventPageVm = eventPageVm;
    this.IconPath = "/Assets/Icons/buy.png";
    this.Name = "buy ticket";
  }
  public override bool CanExecute(object parameter)
  {
    return EventPageVm.Event.EventDto.IsTicketBuyable;
  }

  public override void Execute(object parameter)
  {
    EventPageVm.BuyTicket();
  }
}
```

Abbildung 36: Beispiel einer ApplicationBarCommand

[17] Dependency Property Overview, (Microsoft 2013)

[18] Command Pattern, (Wikimedia 2013)

4.3.12 LIVE TILES

Mit Windows Phone 7 wurde die Metro Oberfläche eingeführt, welche sein graphisches Design stark an New Yorks U-Bahn Schilder anlehnt. Daraus entstanden sind flache rechteckige Kacheln (Tiles), welche von nun an das UI dominieren. Auf Windows Phones ist der gesamte Start Bildschirm mit diesen Kacheln versehen. Microsofts Ziel ist es, dass man nicht nur informationslose Icons auf dem Startbildschirm hat, sondern direkt die wichtigsten, personalisierten Informationen in diesen Kacheln anzeigt.

In Venue ist dieses Prinzip getreu dem Sinne von Microsoft umgesetzt worden. So können innerhalb der App gezielt Inhalte oder direkte Links auf den Startbildschirm angeheftet (pin) werden. In Venue sind folgende Kacheln möglich:

- Club
- Event
- Ticket
- MemberCards
- Special Offers

Der entsprechende Command ist der PinToStartCommand und UnpinFromStartCommand, welche in generischer Form diese Funktionalität dem UI anbieten.

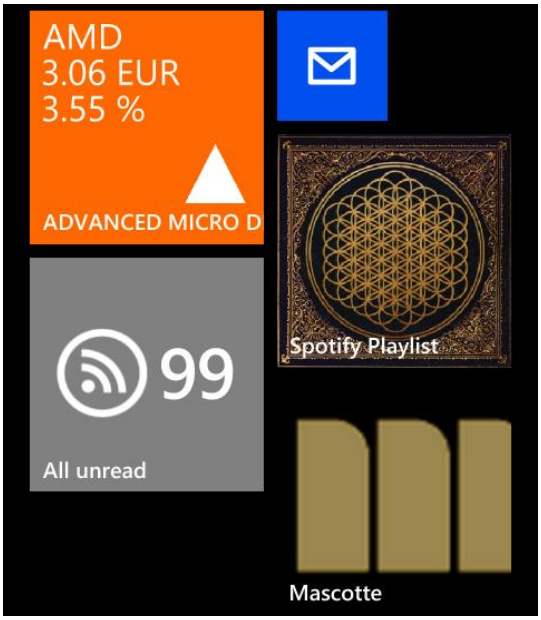


Abbildung 37: Live Tile des Mascotte Clubs

4.3.13 PUSH NOTIFICATIONS

Push Notifications erlauben es, einem Smartphone eine Applikations-Nachricht zu senden. Ohne Push Notifications müsste ein Gerät stetig den Server nach Nachrichten abfragen (sogenanntes Polling), was schlecht für den Akku des Geräts wäre und einen Wildwuchs an verschiedenen Lösungen fördern würde. Push Notifications in Windows Phone 8 sind der einzige Kanal, wie Nachrichten aktiv auf das Gerät gelangen können. Microsoft bietet für diese Kanäle den Dienst Microsoft Push Notification Services (MPNS), welcher direkt in die Plattform integriert ist und für die Zustellung von Push Notifications verantwortlich ist.

Das Prinzip von Push Notifications ist simpel. Ein Gerät erzeugt einen Kanal. Dieser Kanal erhält eine URL. Das Gerät sendet darauf diese URL an den Server, welcher mit einem Aufruf auf diese URL zu

einem späteren Zeitpunkt eine Push Notification auslöst. In Windows Phone 8 gibt es drei Typen von Nachrichten: Toast, Tile oder Raw.

Ein Toast ist eine Nachricht, welche das System in jedem Zustand dem Benutzer anzeigt. Sie erscheint am oberen Bildschirmrand. Wenn der Benutzer darauf drückt, springt er zur angegebenen Ansicht.

Abbildung 38: Toast Nachricht, (Microsoft 2013)

Eine Tile Push Notification ist eine Nachricht, welche es erlaubt, die Tiles auf dem Startbildschirm zu aktualisieren. Dabei kann der Aufrufer der Push Notification URL bestimmen, was auf dem Tile dargestellt werden soll.

Die Raw Push Notification erlaubt es, eine beliebige Datenstruktur zu senden. Diese Nachrichten werden jedoch nur empfangen, wenn die App läuft.

In Venue werden Push Notifications nur für Nachrichten gebraucht. Es werden vom Server Toast Notifications geschickt, wenn der Club eine neue Nachricht sendet. Dabei springt man bei einer Berührung auf die Nachricht direkt zur Nachrichten Ansicht.

4.3.14 WINDOWS PHONE WALLET

Windows Wallet ist eine gemeinsame Brieftasche wo alle Dienste ihre Kreditkarten, Gutscheine oder Zutrittskarten ablegen können. Wallet selbst bietet keine eigene Authentifizierungs- oder Sicherheitsmechanismen. Es verwaltet nur Links von verschiedenen Diensten. Man erstellt dazu sogenannte WalletItems, welche dann dort erscheinen. Ein WalletItem hat bereits viele vorgefertigte Eigenschaften, welche man verwenden kann. Z.B. kann bereits ein 2D Code direkt im WalletItem abgelegt werden. Leider ist es jedoch nicht möglich, direkt NFC Codes zu hinterlegen.

Das einzige was Wallet hier bietet, ist ein sogenanntes Secure Element. Ein Secure Element ist eine Information, welche auf der SIM Karte des Smartphones abgelegt wird. Dies wäre gedacht um z.B. Kreditkarteninformationen dort abzulegen. Das würde bedeuten, dass ein Angreifer zuerst den Pin Code der SmartCard bräuchte, um an diese Informationen zu gelangen.

Prinzipiell wäre das auch interessant für das Venue Ticket, doch Microsoft verlangt für diese Elemente eine spezielle Berechtigung, welche zuerst von Microsoft bewilligt werden muss. Und selbst dann muss noch eine Kooperation mit dem Mobilfunkprovider bestehen, um das Secure Element verwenden zu können.

Ein WalletItem kann eine Navigations-URL beinhalten, welche bei der Auswahl des Items aufgerufen wird. Die URL ist relativ zum Applikationsverzeichnis, wie z.B. „/Views/MemberCard.xaml?membercardKey=1020-12…“

Um WalletItems auf dem neuesten Stand zu halten, muss ein sogenannter WalletAgent implementiert werden. Ein WalletAgent läuft in konfigurierten Abständen im Hintergrund des Systems und kann bei Aktivierung die WalletItems aktualisieren. Venue implementiert einen WalletAgent, welcher alle paar Stunden die Items anpasst, hinzufügt oder ggf. löscht.

4.4 DOOR APP

Da die Door App besitzt nur eine Ansicht (Abbildung 39). Deswegen gibt es auch nur ein ViewModel für diese Ansicht. Jedoch gibt es für die Daten, die es aggregiert noch weitere ViewModel Typen. Um Daten zu aggregieren kann auf das gemeinsame Assembly Venue.Common zugregriffen werden, welches direkte Datenabfragen zum Server ermöglicht.

Auch in der Door App wird ein Bootstrapper und der ServiceLocator verwendet (siehe Kapitel 4.3.3.1). Es gibt jedoch deutlicher weniger Dienste. Einer dieser Dienste ist der DataStore, doch im Gegensatz zu seinem Phone Pendant (Kapitel 4.3.5) gibt es keine Caching Funktionen.

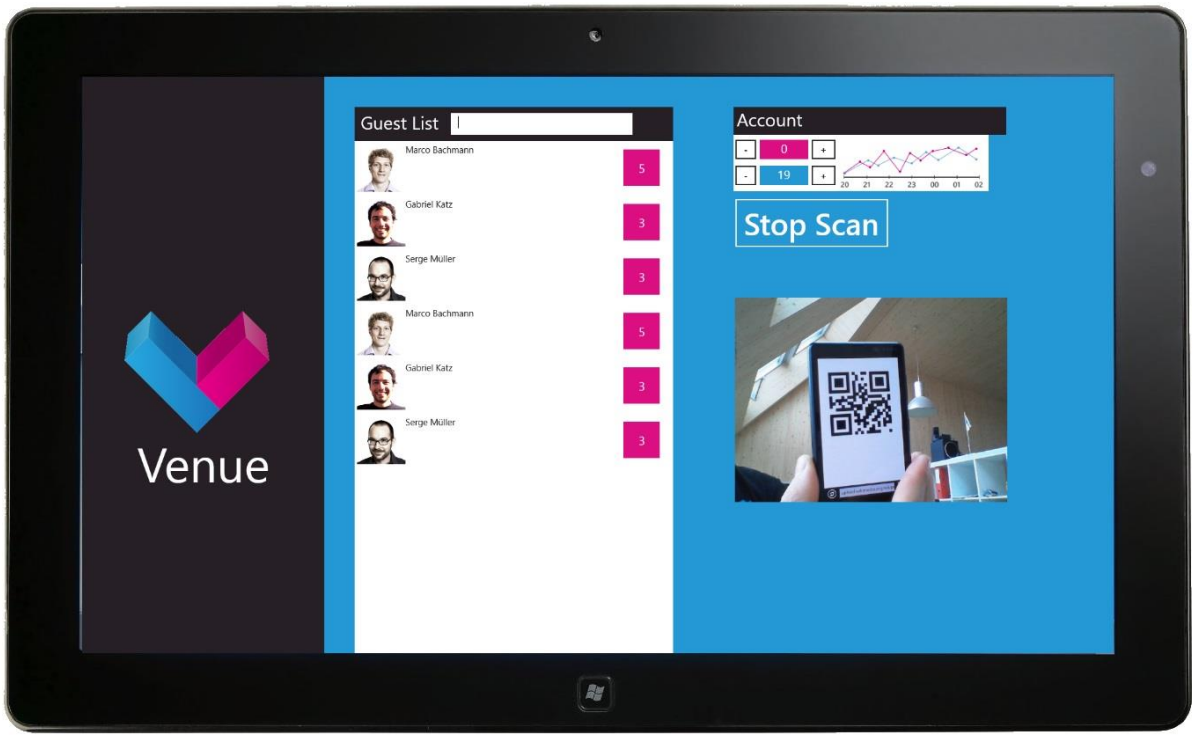


Abbildung 39: Door App Ansicht

4.4.1 QR CODE SCANNING

Das QR Scannen soll als Rückgriff dienen falls der Gast keine NFC Funktion besitzt. Dann hat der Gast die Möglichkeit mit einem ausgedruckten Ticket, einer gedruckten Member Card oder dem Ticket auf dem Handy sich Einlass zu verschaffen.

QR Code Lese-Bibliotheken gibt es einige, bei Windows Phone ist die Auswahl jedoch begrenzt. Doch eine der grössten Open Source Bibliotheken ZXing[19] hat eine .Net Portierung, welche sogar schon für Windows 8 existiert. Doch leider ist die Schnittstelle und Integration nicht so komfortabel. Man kann nicht direkt den Kamera Stream der Bibliothek übergeben, sondern man muss selbst das Bild auslesen und es dann übergeben. Dies verunmöglicht zum Beispiel visuelles Feedback über mögliche

[19] XZing, (CodePlex 2013)

Scann-Probleme, wie sie andere Bibliotheken kennen. Auch die visuelle Darstellung, welcher Code im Bild schlussendlich gescannt wurde, kann nicht zur Verfügung gestellt werden.

In Venue wird der der Kamera Stream initialisiert und ein im UI eingebettet CaptureElement Control zugewiesen. Diese Kombination bietet bestmögliche Visualisierung des Kamera Streams. Doch es ist von Gerät zu Gerät sehr abhängig, wie schnell die Vorschau auf das Kamera Bild reagiert.

Nun wird asynchron dazu immer ein Bild ausgelesen und der ZXing Bibliothek übergeben, welche das Bild auf vorhandene Codes scannt. Immer wenn ein Bild fertig gescannt wurde, wird ein neues erstellt und das Ganze wieder von vorne. Sobald ein Bild gescannt wurde, wird ein Einlass ausgelöst und der Türmanager wird sofort ein Feedback erhalten, ob der Code gültig war.

4.5 NEAR FIELD COMMUNICATION

4.5.1 TECHNOLOGIE

Near Field Communication, kurz NFC, ist eine Kurzstrecken-Übertragungstechnik, welche nun vermehrt in Mobilgeräten eingebaut wird. Spezielle Eigenschaften sind:

- Funkt nur bis zu 10 Zentimetern
- Kleine Datenraten, Brutto bis 400 Kb/s, Netto bei 30 KB/s
- Ohne Kommunikationsverschlüsselung auf Protokollebene
- Verbindungsorientiert
- Nur ein Kanal möglich
- Kann Peer To Peer oder Aktiv/Passiv (mit RFID) verwendet werden

Die Aktiv/Passiv Kommunikation ist schon eine Weile verbreitet. Sogenannte RFID Transponder erlauben es, einen digitalen Code als Etikette zu verwenden. Ein aktives Gerät kann dann diesen digitalen Code per Funk auslesen. Was NFC jetzt jedoch neu bietet, ist eine Peer To Peer Verbindung, d.h. dass zwei Aktive Geräte miteinander darüber Daten austauschen können. Diese Eigenschaften erlauben Dienstleistern ganz neue Anwendungsfälle. Hier sind einige davon:

Kopplung:

Da NFC nicht genügend Bandbreite besitzt um Nutzdaten zu übertragen, kann diese Funktechnik nicht für Audio/Video Übertragungen oder sonstigen Netzwerkdaten genutzt werden. Dafür eignet sich immer noch Bluetooth am besten, welches wenig Anforderungen an die Infrastruktur stellt und sehr stromsparend ist. Doch um Geräte mit Bluetooth koppeln zu können, muss zuerst ein Schlüssel ausgetauscht werden, damit nicht jeder mithören kann oder ein Angreifer nicht Geräte aus grosser Entfernung an sich koppeln kann. Dieser Schlüsselaustausch ist jedoch immer umständlich und fehleranfällig.
Nun kann NFC dazu verwendet werden, um diesen Schlüssel auszutauschen. Der absichernde Faktor dabei ist, dass die beiden Geräte physisch beieinander sein müssen. Wenn man nun davon ausgeht, dass der physische Besitz der zu koppelnden Geräte genug Sicherheit bedeutet, ist diese Methode der Kopplung als sicher zu bezeichnen. Für den Endbenutzer heisst das nun, dass er nur noch die entsprechenden Geräte aneinander halten muss um z.B. seinen Fernseher mit seinem Handy zu verbinden.

Bezahlsysteme:

Wenn ein Smartphone NFC besitzt könnte man sich nun ein Kassensystem vorstellen, welches ebenfalls aktiv mit NFC kommunizieren kann. Wenn nun z.B. ein Kreditkartenanbieter auf dem Smartphone eine App anbietet, bei welcher der Benutzer seine Kreditkarte registriert, kann bei der Kasse diese Kreditkarteninformationen übertragen werden damit die Kasse eine Abbuchung auf diese Karte vornehmen kann. Der Absicherungsmechanismus muss in diesem Fall deutlich höherwertig

sein. Deswegen wird ein gesicherter Kanal aufgebaut, welcher seinen Schlüssel direkt von der SmartCard (SIM Card) des Smartphones bezieht. Dies zieht jedoch nur in Kollaboration mit einem Mobilfunkprovider möglich.

Autorisierungssysteme:

NFC eignet sich sehr gut für Einlasssysteme, denn man benötigt keine zusätzlichen Schlüssel oder SmartCards, man kann alles auf seinem Smartphone haben. Dies vereinfacht die Verwaltung der Zugänge, da man nicht ein Schloss auswechseln muss, sondern dem Benutzer den Zugang verweigert. In Autorisierungssystemen gibt es schon viele solche verschiedenen Lösungen, welche nicht mit NFC als Technologie umgesetzt wurden. Doch gibt es die Möglichkeit eines „Universalschlüssels“ auf dem Smartphone zu haben, welchen man verschiedenen Stellen verwenden kann.

4.5.2 SICHERHEIT

NFC bietet keine eingebaute Übertragungssicherheit. Die Sicherheitsanforderungen sind je nach Anwendungsfall unterschiedlich hoch. So stellt eine Zahlungslösung eine deutlich höhere Anforderung als das Werbeposter.

Bei Peer To Peer Verbindungen kann die Sicherheit auf Applikationsebene umgesetzt werden, zum Beispiel mit einer Public Shared Key (PSK) Verschlüsselung. Diese muss jedoch von jeder Dienstleistung selbst umgesetzt werden, es gibt also keinen Standard.

Mögliche Angriffe:

- Abhören: Aus Distanz könnte mit speziellen Empfangsgeräten die Kommunikation mitgeschnitten werden
- Man in the middle: Zwischen den zwei Kommunikationspartner gibt sich der Angreifer als der jeweils andere aus und kann so die Kommunikation mitschneiden und sogar manipulieren.
- Diebstahl: Bei Diebstahl kann der Angreifer alle Daten aus dem Gerät herauslesen, welche nicht auf einem Secure Element (wie z.B. SIM Karte liegen)

Besonders hierzulande ist auch das allgemeine Misstrauen von Datenmissbrauch stark. Sobald man von Zahlung per Funk spricht, werden viele sofort misstrauisch und fürchten um ihre Habe. Es ist deswegen wichtig, das Vertrauen der Konsumenten in diese Technologie nicht zu zerstören. In Venue wird deswegen von Beginn weg eine starke Verschlüsselung eingesetzt.

4.5.3 UMSETZUNG

Um die Umsetzung einer Zugangskontrolle mit Peer To Peer zu erreichen, muss darauf geachtet werden, dass sehr kleine und nicht zu viele Nachrichten ausgetauscht werden. Denn es würde zu lange dauern, bis eine höherwertige Verbindung (Bluetooth, Wifi) aufgebaut wäre. Es wäre letztendlich auch zu Fehleranfällig einen Protokollwechsel vorzuziehen. Auch kann nicht davon ausgegangen werden, dass der Gast an der Türe Internetempfang hat. Der Door Client hingegen kann auf die Infrastruktur des Clubs zurückgreifen und deswegen kann von einer konstanten (und sicheren) Internetverbindung ausgegangen werden.

4.5.3.1 SICHERHEIT

Das wichtigste beim Thema Sicherheit in der Umsetzung ist, dass nicht Ticketinformationen an eine falsche Person preisgegeben werden. Also muss sichergestellt werden, dass der Door Client auch der ist, für den er sich ausgibt. Hier eignet sich ein Public Shared Key Verfahren, um die Nachrichten zu signieren und somit die Integrität sicher zu stellen. Dabei ist der Door Client im Besitz des Private Keys. Mit diesem Schlüssel wird er seine eigene Nachricht signieren, d.h. einen sicheren HashCode erstellen. Der Gäste Client besitzt den Public Key, welchen er zur offline Verfügbarkeit ebenfalls zum

Ticket speichert. Mit diesem Schlüssel kann er die Signatur in den ursprünglichen HashCode der Nachricht übersetzen, was nur möglich ist, wenn die Signatur mit dem Private Key signiert wurde.

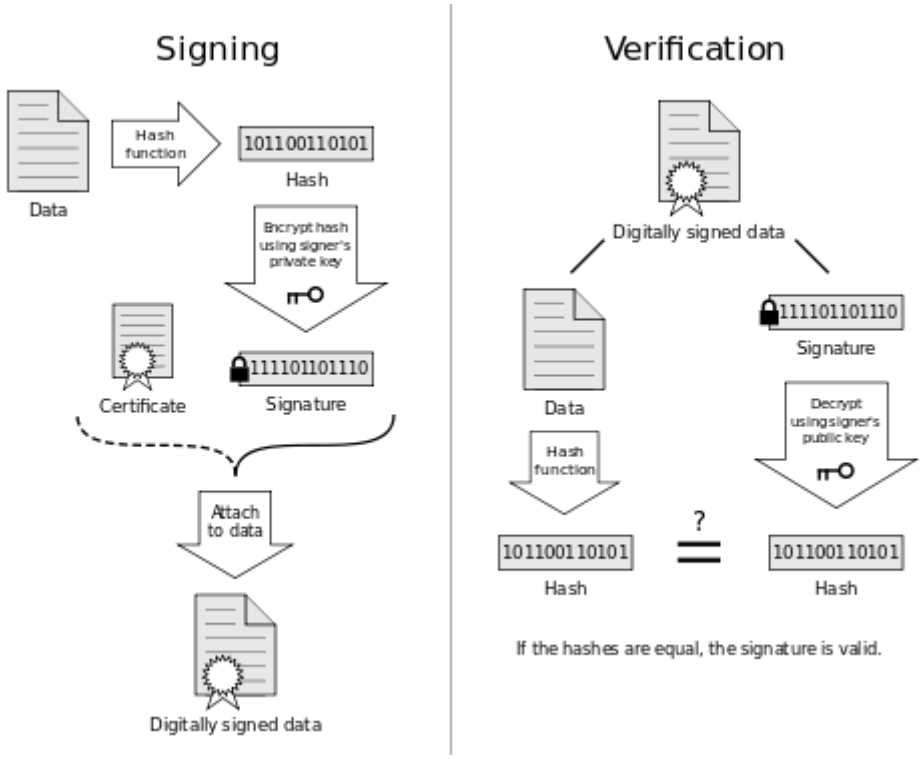


Abbildung 40: Public Shared Key Signing (Acdx 2008)

Damit ein privater Schlüssel bei einem Diebstahl möglichst wenig Schaden anrichten kann, wird pro Event ein neuer Schlüssel erzeugt. So kann der Schaden auf einen Event begrenzt werden.

Als Hashalgorithmus wurde ein SHA256[20] verwendet, welcher eine Implementierung in der .Net Library besitzt. Der Zugriff verläuft über die RSACryptoServiceProvider Klasse. Um den Umgang mit dieser Klasse zu vereinfachen und den Public Key abzulegen wurde die integrierte Xml Serialisierung verwendet. Der Code zur Überprüfung der Signatur ist entsprechend einfach, siehe dazu Abbildung 41.

```
using (var securityProvider = new RSACryptoServiceProvider())
{
  securityProvider.FromXmlString(accessDto.PublicKey);
  return securityProvider.VerifyData(accessParameter.EventKey.ToByteArray(),
(object)"SHA256", accessParameter.Signature);
}
```

Abbildung 41: Verwendung des RSACryptoServiceProvider

[20] SHA-2 (Wikimedia 2013)

4.5.3.2 ABLAUF

Abbildung 42: Ablauf NFC Transaktionen

Wenn die App auf dem Smartphone des Gastes noch nicht gestartet is,t hat die Guest Door App die Möglichkeit, einen sogenannten AppLaunch Befehl zu senden, bei welcher die entsprechende AppID mitgesendet wird. Besitzt das Smartphone die App, wird der Benutzer aufgefordert diese zu starten (siehe Abbildung 43).

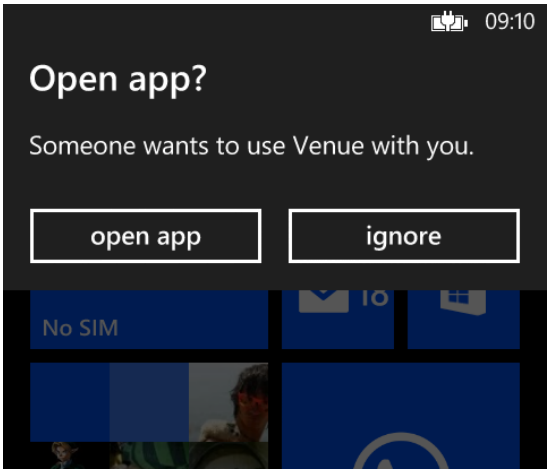


Abbildung 43: AppLauch Bestätigung

Ist die Guest App aber bereits gestartet, wird diese sofort eine „Ready“ Nachricht senden. Erhält die Door App diese Nachricht, wird er keine AppLaunch Nachricht senden. Erhält er aber nach 100 Millisekunden keine Ready Nachricht, kann er davon ausgehen, dass die App nicht gestartet ist. Nun kann der Door Client davon ausgehen, dass er mit der Guest App kommuniziert, somit fragt er für den entsprechenden Event den er kontrolliert, nach einer gültigen Zulassung. Diese Nachricht heisst AccessRequestInfo. Mit dieser Nachricht werden im Json Format Daten gemäss Klasse in der Abbildung 44 übertragen.

```
public class RequestAccessParameter
{
  public Guid EventKey { get; set; }
  public Guid ClubKey { get; set; }
  public byte[] Signature { get; set; }
}
```

Abbildung 44:RequestAccessParameter Nachricht

Diese Nachricht enthält nun eine Signatur welche den gehashten EventKey beinhält. Der Guest Client validiert nun die Nachricht und sucht entsprechend in seinen lokalen Daten um ein Ticket für den Event oder eine MemberCard für den Club. Findet er einen entsprechenden Eintrag, schickt er eine AccessRequest Nachricht. Die AccessRequest Nachricht beinhaltet einen Access Token, welche auf dem Ticket oder der MemberCard ist.

Dieser Token liest der Door Client und sendet ihn zur Überprüfung an den Server, natürlich über eine gesicherte Leitung (SSL). Der Server gibt ihm eine entsprechende Antwort. Daraufhin sendet der Door Client über NFC eine AccessGranted Nachricht welche dazu Daten im Json Format sendet. Die Struktur der Daten entspricht der Klasse in Abbildung 45.

```
public class AccessGrantedParameter
{
  public string Token { get; set; }
  public bool AccessGranted { get; set; }
}
```

Abbildung 45: AccessGrantedParameter Klasse

Nach dieser Nachricht werden auf beiden Clients entsprechende visuelle Feedbacks gegeben. Auf der Door App ist nun der Gast zu sehen (siehe Abbildung 46) und auf der Guest App erscheint ein Erfolgsbildschirm (siehe Abbildung 47).

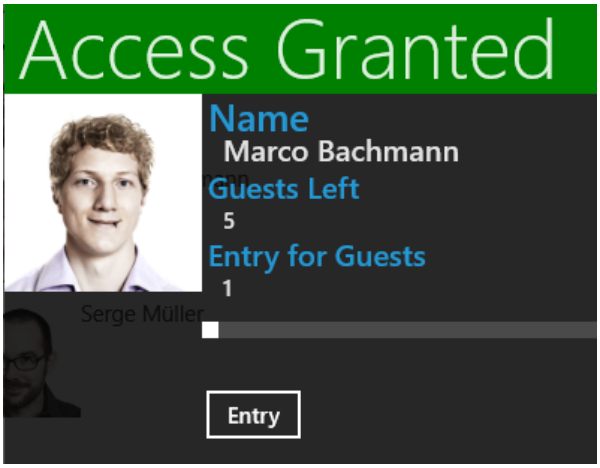


Abbildung 46: Zugang erteilt Meldung der Door App

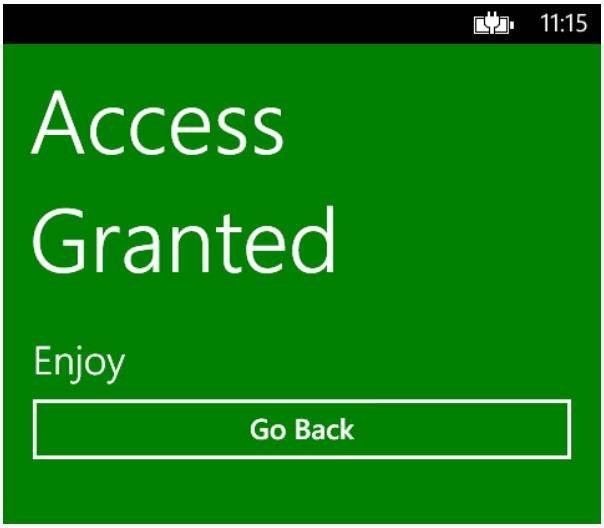


Abbildung 47: Zugang erteilt Meldung der Guest App

4.5.4 APP LAUNCH BEFEHL

Der AppLaunch Befehl ist ein Microsoft spezifischer NFC Befehl. Unterliegend formatiert der Treiber des Geräts den Befehl in eine NDEF[21] formatierte Nachricht mit dem Inhalt „windows.com/LaunchApp“. Doch Microsoft unterstützt hier leider nur den Befehl „LaunchApp:WriteTag“, was so viel bedeutet wie dass ein AppLaunch auf einen RFID Chip geschrieben, jedoch nicht Peer To Peer gesendet werden kann. Wenn man eine AppLaunch Befehl per Peer To Peer senden will, muss man den Befehl als NDEF Befehl absetzen[22]. Um jetzt nicht selbst einen NDEF Befehl formatieren zu müssen gibt es ein CodePlex Projekt „NDEF Library for Proximity APIs“[23]. Diese Bibliothek stellt bereits eine entsprechende AppLaunch Message zur Verfügung.

[21] Understanding NFC Data Exchange Format (NDEF) messages, (Nokia 2013)

[22] How to Create Cross-Platform LaunchApp NFC Tags, (Microsoft 2013)

[23] NDEF Library for Proximity APIs, (Open Source 2013)

```
var ndefLaunchAppRecord = new NdefLaunchAppRecord()
                                    {
                                      Arguments = "t"
                                    };
ndefLaunchAppRecord.AddPlatformAppId("WindowsPhone", AppStoreId);
var ndefMessage = new NdefMessage()
                             {
                               ndefLaunchAppRecord
                             };
_appLaunchPublishId = _proximityDevice.PublishBinaryMessage("NDEF",
ndefMessage.ToByteArray().AsBuffer(), (device, id) =>
device.StopPublishingMessage(id));
```

Abbildung 48: AppLaunch Nachricht senden

Diese NDEF Nachricht kann auch gleich um andere plattformspezifische AppLaunch Befehle erweitert werden. So muss der Aufrufer nicht wissen, welche Plattform sein Gesprächspartner hat, er vereint sie einfach alle in einer Nachricht.

4.5.5 GESCHWINDIGKEIT

Da bei einem Konzert sehr viele Gäste zur selben Zeit kommen ist der Zeitfaktor sehr wichtig. Bisherige Methoden haben alle ihre Nachteile. Ein gedrucktes Ticket mit speziellem Papier (wie bei Ticketcorner) kann nur von autorisierten Stellen ausgestellt werden und muss beim Eingang von jemandem auf Sicherheitsmerkmale überprüft werden. Es ist somit sehr Zeitaufwendig und somit teuer. Ein QR Code wie bei Starticket kann selbst gedruckt werden. Jedoch kommt es an der Türe zu Verzögerungen, denn je nach Qualität des Drucks und der Lichtverhältnisse dauert der Scannvorgang im Schnitt 2 Sekunden, was sich bei 1000 Gästen zur selben Zeit hochrechnet.

Es ist also sehr wichtig, dass der NFC Autorisierungsprozess schneller als QR Codes funktioniert, um einen echten Mehrwert an der Türe darzustellen. Um die Geschwindigkeit des Prozesses zu testen wurde eine Testserie erstellt.

4.5.5.1 AUFBAU

Als Tür Tablet kommt ein Asus Asus Vivo Tab TF810 zum Einsatz. Es verfügt über einen NXP NFC Chip. NXP ist einer der weitverbreitetsten NFC Chiphersteller. Als Gäste Smartphone kommt ein Nokia Lumia 820 zum Einsatz. Es verfügt ebenfalls über einen NFC Controller von NXP. Beide Geräte sind über Wireless mit dem Internet verbunden. Die reine Server Zeit für die Ticketkontrolle ist bei 50 Millisekunden. Der Start der Messung erfolgt, wenn der Gäste Client das NFC DeviceArrived Event auslöst und endet, wenn der Gäste Client die AccessGranted Nachricht erhält. Das Resultat wird auf die Konsole ausgegeben.

4.5.5.2 RESULTATE

In der ersten Testserie befindet sich der Server im lokalen Netz.

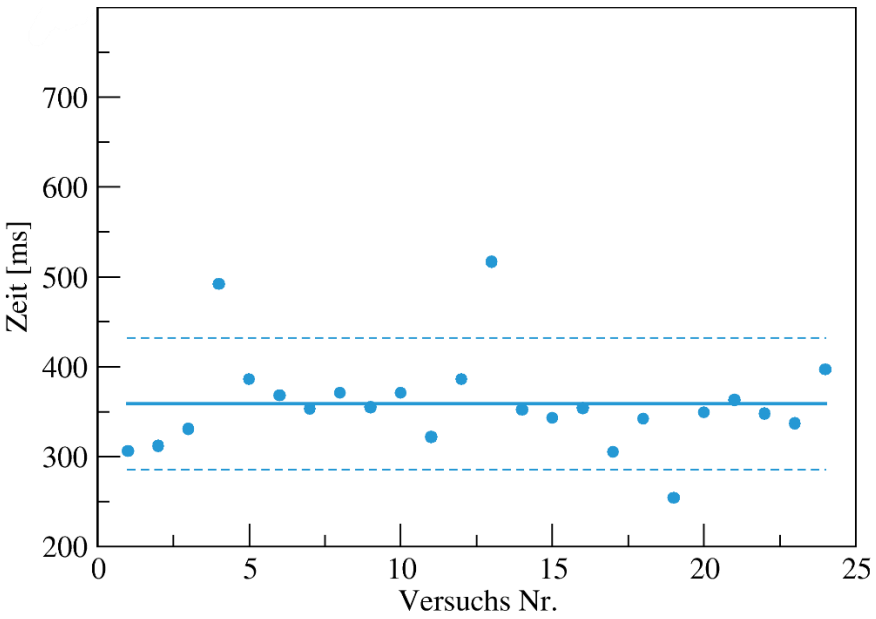


Abbildung 49: Zeitmessung NFC bei lokalem Server

Die Resultate der Abbildung 49 sind relativ Konstant bei 300 bis 400 Millisekunden. Das ist beinahe kaum spürbar bei der Bewegung hin zum NFC Sensor. Da benötigt das Herhalten des Smartphones bereits länger. Da dies bei QR Codes sowieso auch passiert, ist deutlich zu sehen, dass NFC bei Konzerten einen grossen Mehrwert darstellen kann.

Da ein lokaler Server jedoch nicht sehr aussagefähig ist, wurde ein zweiter Versuch erstellt. Es wurden dieselben Geräte verwendet, jedoch steht der Server nun im Microsoft Datenzentrum in Amsterdam.

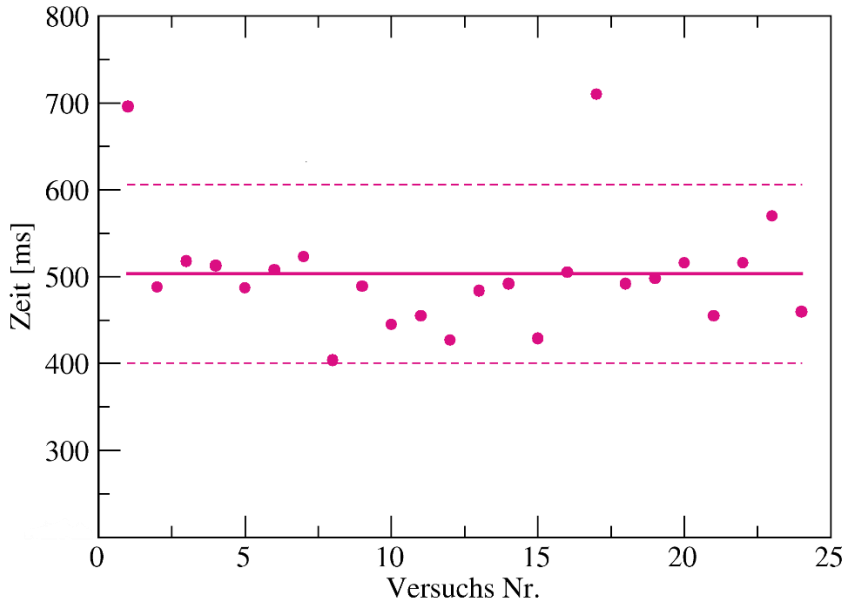


Abbildung 50: Zeitmessung NFC bei Server in Amsterdam

Erfreulicherweise sind die daraus resultierenden Messdaten in Abbildung 50 weiterhin gleich konstant. Doch das allgemeine Mittel hat sich um rund 100 Millisekunden nach oben geschoben, was auf die erhöhte Latenz der Verbindung zurückzuführen ist.

5 SCHLUSSFOLGERUNGEN

5.1 BUSINESS

Venue ist ein Projekt mit vielen Chancen. Technisch gibt es zumindest keine Hürden mehr. Denn von gezieltem Marketing, Suchplattform bis hin zur Türkontrolle kann mit Venue alles abgedeckt werden. Der nächste grosse Schritt wäre wohl noch, dass man mit dem Smartphone an der Bar bezahlen kann, doch weit stehen wir davon auch nicht mehr entfernt.

Das Projekt in Form von dieser Arbeit sollte einen Prototypen sein, welcher die neuen Möglichkeiten für Clubs aufzeigen soll. Dass es keine richtige Serverimplementierung gibt, spielt dabei noch keine grosse Rolle, denn die grossen „Wow“-Effekte entstehen nur vor den neuen Gerätetypen, wie Tablet und Smartphone.

An einem Start dieses Projekt steht jedoch noch ein grosses Hindernis im Weg: die diversen Plattformen. Denn nun haben wir hier eine Lösung, welche zwar zu beindrucken vermag, jedoch nur bei einem sehr kleinen Prozentsatz an Geräten in der Schweiz funktioniert. Um mit einer solchen Lösung Erfolg zu haben, muss jedoch jedes Gerät angesprochen werden, denn nur wenn beinahe jeder Gast erreicht wird macht der Einsatz von Venue Sinn.

5.2 TECHNISCH

5.2.1 PLATTFORM

Microsoft hat mit den Plattformen Windows Phone 8 und WinRT wie gewohnt für Entwickler eine solide Grundlage geschaffen. Denn nicht nur die zugrundeliegende Technik (.Net) sondern auch die Art und Weise wie man UI entwickelt ist seit Einführung des .Net Framework 3.5 im Jahre 2007 dieselbe geblieben.

Die Plattformübergreifende Lösung wird jedoch nur halbherzig angestrebt, und das aus zweierlei Perspektiven. Zum einen haben zwar Windows Phone 8 und WinRT den gleichen System-Kernel und die gleiche Peripherie (Touchscreen, Kamera, Gyroskop etc.), jedoch sind es weiterhin unterschiedliche Programmier-Plattformen. Auf Android oder iOS hingegen, sind dies exakt die gleichen Plattformen. Dort schreibt man eine App, welche auf beiden Geräten funktioniert. Hier ist auch das PortableAssembly nur ein schwacher Trost, denn man kann kein UI damit teilen.

Der andere Aspekt der Plattform übergreifenden Lösung ist, dass Microsoft ebenfalls keinen Schritt auf eine wirkliche übergreifende Lösung unternimmt. Es gibt immer noch keinen Standard zum auf Blackberry OS, iOS, Android oder Windows zu programmieren. Was hier als gemeinsame Grundlage dienen könnte, wäre HTML, doch keine der Plattformen unterstützt dies als „First Citizen“ Programmiersprache. Der Umweg über PhoneGap funktioniert zwar gut genug, doch ist dies nicht das Engagement der Plattform Entwickler, sondern das der App Entwickler, welches darin steckt. Windows 8 unterstützt zwar JavaScript und Html als Programmiersprache, doch programmiert man damit nicht in einem Browser als Plattform, sondern gegen WinRT, was wiederum keine Plattform übergreifende Lösung darstellt.

5.2.2 NEAR FIELD COMMUNICATION

Die Verwendung von NFC stellte sich als denkbar einfach heraus. Nicht zuletzt auch, wegen des sehr einfach gehaltenen Proximity API. Lobend zu erwähnen an dieser Stelle ist, dass Microsoft dieses API auf WinRT wie auf Windows Phone 8 exakt gleich implementiert haben, bzw. die Schnittstellen die gleichen sind.

Dass es bei NFC keinen gemeinsamen Sicherheitsstandard gibt ist ein wenig unverständlich. So wird jeder Dienst seine eigene Sicherheit definieren müssen. Jeder Dienst mit Sicherheitskonzepten wird proprietär und ein geschlossenes System, was besonders mühsam bei Zahlungsvorgängen ist, da für jeden Zahlungsanbieter wieder ein neues Terminal gebraucht wird. Ein Standard wäre hier angemessen.

Die Geschwindigkeit mit der Geräte sich koppeln und austauschen ist überraschend schnell. Jedoch ist eine gewisse Übungszeit notwendig, um herauszufinden, wo sich bei einem Smartphone der NFC Sensor befindet. Wenn sich der Sensor bei jedem Smartphone an einer anderen Stelle befinden würde, wäre dies Fatal, denn man könnte nicht zwei Geräte in derselben Haltung aneinanderhalten, um Kontakt aufzunehmen.

Ob sich NFC überhaupt etablieren wird ist leider zum Ende dieser Arbeit gerade noch in Frage gestellt worden. Denn Apple hat mit der Vorstellung von iOS 7 gezeigt, dass ihre neuen Geräte diesen Herbst kein NFC haben werden. Und so stellt sich doch die Frage, ob Apple überhaupt je auf den NFC Zug springen wird und was dann aus der restlichen NFC Industrie wird.

6 REFERENZEN

Acdx, Wikimedia Author. «File:Digital Signature diagram.svg.» *wikimedia.org.* 28. 11 2008. http://commons.wikimedia.org/wiki/File:Digital_Signature_diagram.svg (Zugriff am 7. 6 2013).

CodePlex. «MVVM Light.» *CodePlex.* 2013. http://mvvmlight.codeplex.com/.

—. «patterns & practices.» *CodePlex.* 2013. http://msdn.microsoft.com/en-us/library/ff921345.aspx.

—. «ZXing.» *ZXing.* 2013. http://zxingnet.codeplex.com/.

Microsoft. «Code Generation and T4 Text Templates.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/vstudio/bb126445.aspx.

—. «Cross-Platform Development with the .NET Framework.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/gg597391.aspx.

—. «Data binding overview.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/windows/apps/hh758320.aspx.

—. «Dependency Property Overview.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/ms752914.aspx.

—. «How to Authenticate Web Users with Windows Azure Active Directory Access Control.» *MSDN.* 2013. http://www.windowsazure.com/en-us/develop/net/how-to-guides/access-control/.

—. «How to Create Cross-Platform LaunchApp NFC Tags.» *TechNet.* 2013. http://social.technet.microsoft.com/wiki/contents/articles/13955.how-to-create-cross-platform-launchapp-nfc-tags.aspx.

—. «Implementing the Model-View-ViewModel pattern in a Windows Phone app.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521153(v=vs.105).aspx.

—. «Microsoft.Phone.SecureElement Namespace.» *MSDN.* 2013. http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.secureelement(v=vs.105).aspx.

—. «Mobile Services.» *Windows Azure.* 2013. http://www.windowsazure.com/en-us/develop/mobile/.

—. «Passive Authentication for ASP.NET with WIF.» *MSDN Magazine.* 2013. http://msdn.microsoft.com/en-us/magazine/ff872350.aspx.

—. «Quickstart: Working with files and folders in Windows Phone 8.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj681698(v=vs.105).aspx#BKMK_FilesandfoldersAPIoverview.

—. «Toasts for Windows Phone.» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662938(v=vs.105).aspx.

—. «WebView Class.» *MSDN.* 2013. http://msdn.microsoft.com/de-ch/library/windows/apps/windows.ui.xaml.controls.webview.

—. «Windows Phone Toolkit.» *CodePlex.* 2013. http://phone.codeplex.com/.

—. «XAML Overview (WPF).» *MSDN.* 2013. http://msdn.microsoft.com/en-us/library/ms752059.aspx.

nearfieldcommunication.com. «NFC Architecture.» *Near Field Communication.* 2013. http://nearfieldcommunication.com/developers/nfc-architecture/.

NFC Forum. «Secure Element Programming.» *NFC Forum.* 2009. http://www.nfc-forum.org/events/oulu_spotlight/2009_09_01_Secure_Element_Programming.pdf.

Nokia. «Understanding NFC Data Exchange Format (NDEF) messages.» *Nokia Developer.* 2013. http://www.developer.nokia.com/Community/Wiki/Understanding_NFC_Data_Exchange_Format_(NDEF)_messages.

Open Source. «NDEF Library for Proximity APIs.» *CodePlex.* 2013. https://ndef.codeplex.com/.

PayPal. *Adaptive Payments API.* 2013. https://www.x.com/de/developers/paypal/products/adaptive-payments.

—. «Direct Payment API Introduction.» *PayPal.com.* 2013. https://www.paypal.com/cgi-bin/webscr?cmd=_dcc_hub-outside.

Smart Card Alliance. «NFC Frequently Asked Questions.» *Smart Card Alliance.* 2013. http://www.smartcardalliance.org/pages/publications-nfc-frequently-asked-questions#5.

Wikimedia. «Command Pattern.» *Wikipedia.* 2013. http://en.wikipedia.org/wiki/Command_pattern.

—. «Dependency injection.» *Wikipedia.* 2013. http://en.wikipedia.org/wiki/Dependency_injection.

—. «SHA-2.» *Wikipedia.* 2013. https://de.wikipedia.org/wiki/SHA-2.

7 ABBILDUNGSVERZEICHNIS

Abbildung 1: Deployment Diagramm 8
Abbildung 2: Komponenten Diagramm 9
Abbildung 3:Beispiel einer Panorama View 10
Abbildung 4: Information Map 11
Abbildung 5: Kontext Menu zum entfernen von Favoriten 12
Abbildung 6: Visuelles Design 13
Abbildung 7: ApplicationBar 13
Abbildung 8: Registrierung bei Bootstrapper 14
Abbildung 9: Service Abhängigkeiten 14
Abbildung 10: INavigationManager Interface 15
Abbildung 11: IAccessManager Interface 15
Abbildung 12: IDataStore Interface 16
Abbildung 13: ILoginProvider Interface 16
Abbildung 14: IDispatcherContainer Interface 17
Abbildung 15: Ablauf ACS Authentifizierung, (Microsoft 2013) 18
Abbildung 16: Auswahl der Ziel Plattformen für Portable Assemblies 20
Abbildung 17: T4 Template Beispiel Code 20
Abbildung 18:Snippet zur manuellen Proxy Generierung 21
Abbildung 19: Klassisches Caching Verfahren 22
Abbildung 20: Multi Level Caching 23
Abbildung 21: OnlineFirst Cache Verfahren 25
Abbildung 22: MVVM 26
Abbildung 23: PageViewModel Erzeugung 26
Abbildung 24: PageVIewModel Zugriff 26
Abbildung 25: Beispiel von View Properties 27
Abbildung 26: Navigations Methode 27
Abbildung 27: Map Ansicht 28
Abbildung 28: Berechnung des optimalen Zoom Levels Horizontal 28
Abbildung 29: Berechnung des optimalen Zoom Levels Vertikal 28
Abbildung 30: ImageCache Ablauf 30
Abbildung 31: Nutzung des CacheImageConverter 30
Abbildung 32: PayPal Adaptive Payments 32
Abbildung 33: Falscher Singleton 33
Abbildung 34: Lokalisierung als globale Ressource 34
Abbildung 35: Lokalisiertes DataBinding 34
Abbildung 36: Beispiel einer ApplicationBarCommand 34
Abbildung 37: Live Tile des Mascotte Clubs 35

Abbildung 38: Toast Nachricht, (Microsoft 2013) 36
Abbildung 39: Door App Ansicht..... 37
Abbildung 40: Public Shared Key Signing (Acdx 2008) 40
Abbildung 41: Verwendung des RSACryptoServiceProvider 40
Abbildung 42: Ablauf NFC Transaktionen 41
Abbildung 43: AppLauch Bestätigung 42
Abbildung 44:RequestAccessParameter Nachricht 42
Abbildung 45: AccessGrantedParameter Klasse 42
Abbildung 46: Zugang erteilt Meldung der Door App 43
Abbildung 47: Zugang erteilt Meldung der Guest App 43
Abbildung 48: AppLaunch Nachricht senden 44
Abbildung 49: Zeitmessung NFC bei lokalem Server 45
Abbildung 50: Zeitmessung NFC bei Server in Amsterdam..... 45