Audit 3

Projekt an der TH Köln

Wintersemester 22/23

Entwicklungsprojekt – Perspektive – Social Computing

Von Frederik Hausen, Philipp Zimmer, Sebastian Koch

Bei Mirjam Blümm, Uwe Müsse, Simon Schulte

User-Profiles

Heimatforscher (jung)	
Merkmal	**Merkmalsausprägung**
Alter	18-30
Verfügbarkeit	Freizeit, gering bis hoch
Rollen	Consumer
Sprachkompetenz	Deutsch, English & weitere möglich
Kulturelle Kompetenz	Deutsch
Computer Literacy	Sehr Hoch
Selbstständigkeit	Hoch bis sehr hoch
Erreichbarkeit	Telefonisch, Email
Technische Ausstattung	Laptop, Handy, PC, Tablet
Wohnort	Großstadt

Neuer Bewohner (Ausländer)	
Merkmal	**Merkmalsausprägung**
Alter	25-40
Verfügbarkeit	Freizeit
Rollen	Consumer
Sprachkompetenz	Deutsch (möglich gering), Englisch & Muttersprache
Kulturelle Kompetenz	Deutsch (möglich gering), Heimatkultur
Computer Literacy	Hoch
Selbstständigkeit	Mittel
Erreichbarkeit	Telefonisch, Email
Technische Ausstattung	Laptop, Handy, Tablet
Wohnort	Kleinstadt

Zu User-Profiles

2

Anhand von Feedback zu den User Profiles aus dem letzten Audit wurden diese mit weiteren Profilen angereichert. Die damit definierte Zielgruppe war sehr eng bezüglich der inludierten Stakeholder und in Teilen schlecht definiert, so war beim Alter nur von „Senioren“ und „Durchschnitt“ die Rede anstatt klare Jahresangaben zu benutzen. Vor allen Dingen wurden jüngere Menschen nicht mit in das System einbezogen. Es ergaben sich daraus nur sehr begrenzte Use Szenarien. Die hier vorgestellten Profile sind Beispielhaft für die erfolgte Erweiterung. Anhand dieser beiden wurden zudem auch noch neue Personas erstellt.
Personas:
https://github.com/sebastiankoch10/EPWS2223HausenKochZimmer/blob/639b4999fcf3b82715c466e36d7729fe9b0d42e7/Modellierungen/Personas.docx

Der Einfluss dieser Iteration war bisher recht gering, da noch keine UI Modellierungen angefertigt wurden und somit die User Profiles und Personas nur bei der Modellierung der Use Cases zum Vorschein kamen.

Use-Cases

Name	Subscribe
Kurzbeschreibung	Der Benutzer abonniert eine Stadt
Beteiligte	• Akteur Benutzer • System
Auslöser	Der Benutzer drückt den „Abonnieren“-Button der jeweiligen Stadt
Vorbedingung	Der Benutzer muss eingeloggt sein.
Normalablauf	Der Benutzer drückt den „Abonnieren“-Button der jeweiligen Stadt Abgleich, ob Benutzer eingeloggt ist. Abgleich, ob Benutzer Stadt schon abonniert hat.
Alternativer Ablauf (Erweiterungen)	1a. Drückt der Benutzer auf Abonnieren und ist noch nicht eingeloggt, so gelangt er auf eine Seite, um sich einzuloggen oder zu registrieren. 2a. Drückt der Benutzer auf Abonnieren und hat die jeweilige Stadt schon abonniert, so bekommt er die Frage, ob er das Abonnement löschen möchte.
Ablauf mit Fehlern	3a. Funktion zum Abonnement konnte nicht aufgerufen werden.
Ergebnis	Benutzer hat die Stadt mit seinem Benutzerprofil abonniert.
Nachbedingung	Benutzer bekommt Benachrichtigungen der abonnierten Stadt.

Name	Upload eines Bildes
Kurzbeschreibung	Der Benutzer lädt ein Bild hoch
Beteiligte	• Akteur Benutzer • System
Auslöser	Der Benutzer drückt den Button der Upload-Funktion
Vorbedingung	Der Benutzer muss eingeloggt sein.
Normalablauf	Der Benutzer drückt den „Hochladen“-Button. Benutzer wählt Stadt aus. Der Benutzer wählt ein Bild von seinem lokalen Gerät aus. Abgleich, ob Bild richtiges Datenformat hat. Abfragen der Metadaten des Bildes. Erzeugen eines Bildobjekts mit Bild und Metadaten Abspeichern des Bildobjekts Ansicht des Bildobjekts erzeugen
Alternativer Ablauf (Erweiterungen)	1a. Drückt der Benutzer auf „Hochladen“ und ist noch nicht eingeloggt, so gelangt er auf eine Seite, um sich einzuloggen oder zu registrieren. 2a. Benutzer befindet sich auf einem Städteprofil und drückt den „Hochladen“-Button, dann muss er keine Stadt auswählen.
Ablauf mit Fehlern	3a. Bild hat ein fehlerhaftest Datenformat und Benutzer bekommt Fehlermeldung. 4a. Bild hat fehlerhafte Metadaten und Benutzer bekommt Fehlermeldung.
Ergebnis	Benutzer hat ein Bild zu einer Stadt gepostet.
Nachbedingung	Benutzer bekommt Benachrichtigungen, wenn mit seinem Bild interagiert wird.
Hinweise	
Name	Login

3

Wir haben uns für diese Use-Cases entschieden, da wir diese als die Grundfunktionalitäten für unseren Prototypen identifiziert haben. Nach letztem Feedback werden wir den Use-Case „Login“ mit dem Use-Case „Publish“ ersetzen, da dieser für unser System relevanter ist. Aus Zeitgründen werden wir diesen Use-Case jedoch erst für Audit 4 modellieren, da dieser für unser Weiterarbeiten vorerst nicht wichtig ist und wir ihn daher gering priorisieren.

Use-Cases

Name	Login
Kurzbeschreibung	Der Benutzer loggt sich ein.
Beteiligte	• Akteur Benutzer • System
Auslöser	Der Benutzer drückt den Button der Einlog-Funktion
Vorbedingung	Keine
Normalablauf	Der Benutzer drückt den Button der Einlog-Funktion Username/E-Mail und Passwort Dataset abgefragt. Registrys werden verglichen. Bei gefundener Übereinstimmung wird der Benutzer eingeloggt und auf die Startseite weitergeleitet.
Alternativer Ablauf (Erweiterungen)	1a. Besteht kein Userprofil, wird er auf die Registrierungsseite weitergeleitet 2a. Hat er das Passwort falsch eingegeben, wird die „Passwort vergessen?"-Funktion aufgerufen.
Ablauf mit Fehlern	3a. Bei fehlender Übereinstimmung wird der Benutzer nicht eingeloggt und bekommt Fehlermeldung.
Ergebnis	Benutzer wird eingeloggt.
Nachbedingung	Benutzer bleibt eingeloggt bis es sich ausloggt.

4

Wir haben uns für diese Use-Cases entschieden, da wir diese als die Grundfunktionalitäten für unseren Prototypen identifiziert haben. Nach letztem Feedback werden wir den Use-Case „Login" mit dem Use-Case „Publish" ersetzen, da dieser für unser System relevanter ist. Aus Zeitgründen werden wir diesen Use-Case jedoch erst für Audit 4 modellieren, da dieser für unser Weiterarbeiten vorerst nicht wichtig ist und wir ihn daher gering priorisieren.

POCs

Priority

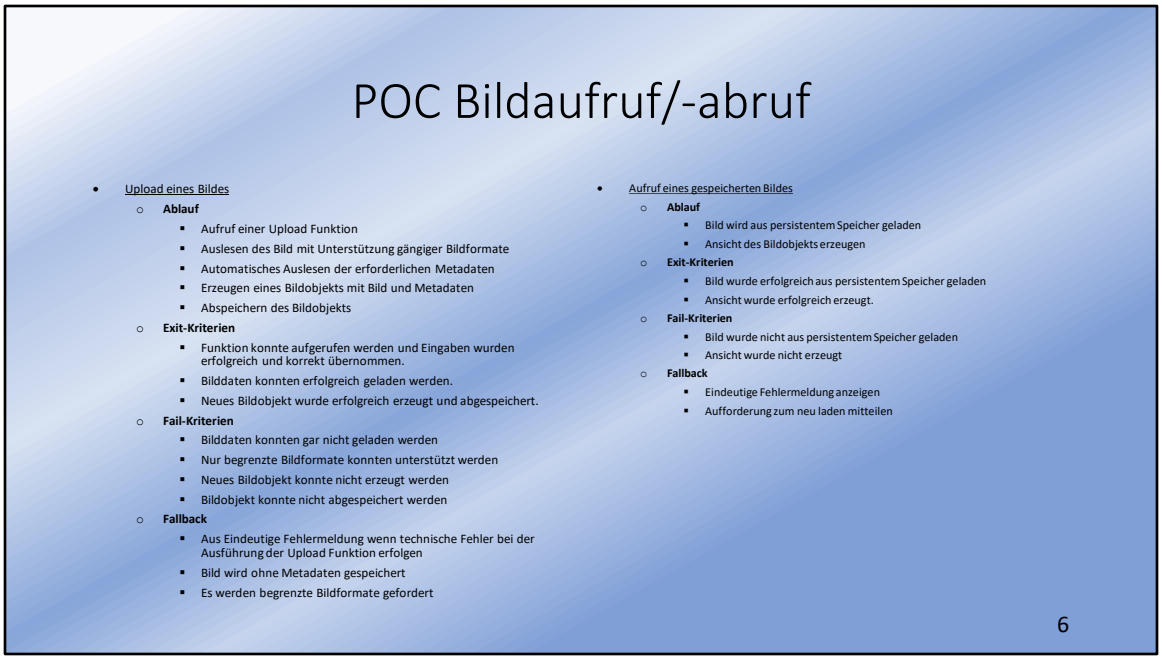
- Upload eines Bildes
- Aufruf eines gespeicherten Bildes
- Pub/Sub Funktionalität – subscribe
- Pub/Sub Funktionalität – publish
- Login

Keine Priority

- Registrierung
- Verschlüsselung
- „Befreunden“ mit anderem User
- Annahme „Befreundung“
- Chat Funktion
- Zugriff auf Datenbank

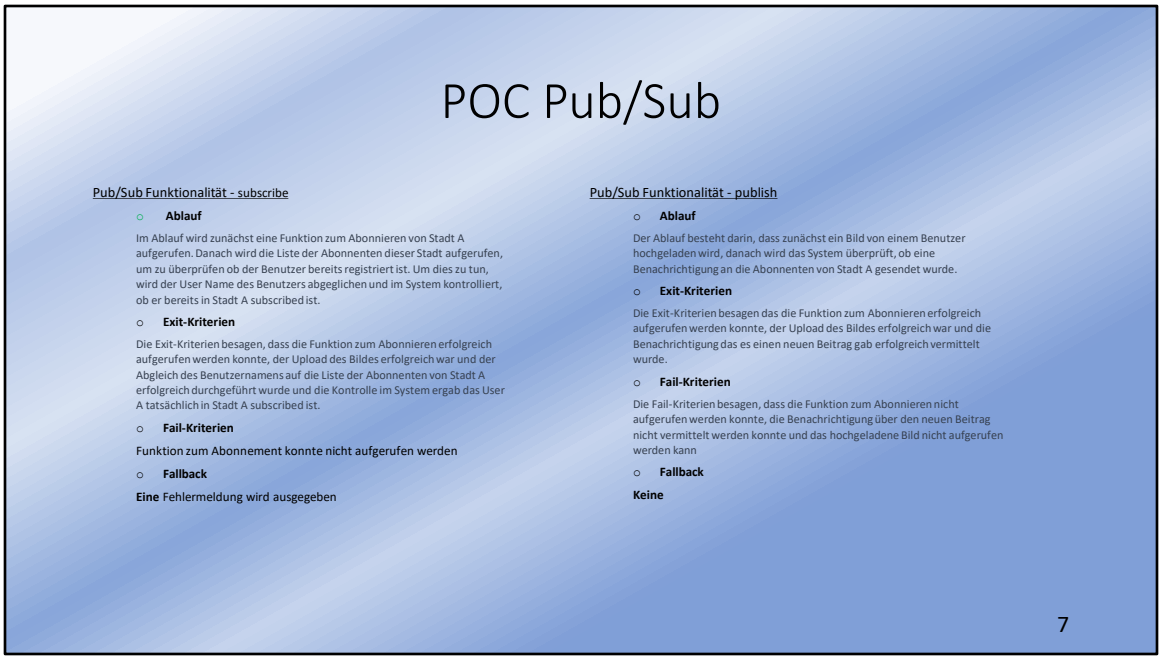
5

Die POC mit Priority decken die kern Funktionalitäten unseres Systems ab und wurden demensprechen eingeteilt. Die POCs decken sich ebenfalls mit unseren Use Cases.

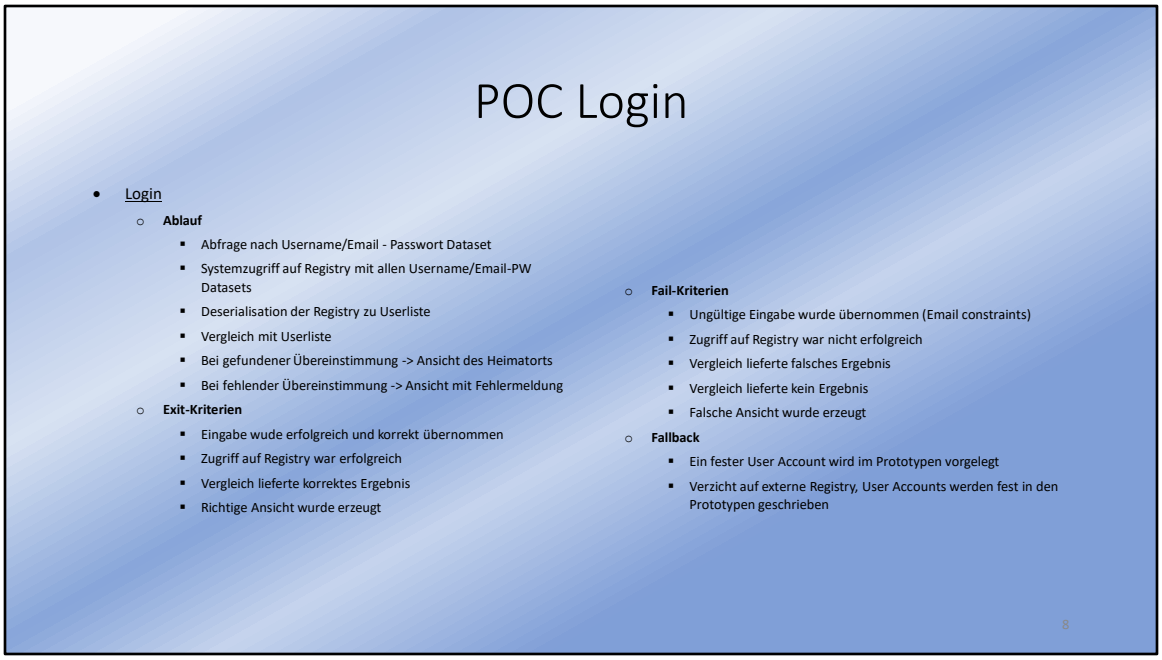


Upload eines Bildes
Ablauf: Beim diesem POC ist geplant, dass er neben der Verarbeitung der gängigen Bildatei ebenso einige der für das System relevanten Metadaten der digitalisierten Bildatei verarbeitet. Als gängige Bildateien sind uns png, jpeg sowie webp geläufig und bei Recherchen zu Bildverarbeitung durch Kotlin, haben wir diese Formate als standart verarbeitbare Formate Identifiziert. Als relevante Metadaten haben wir die Auflösung, Bildformat, Uploader identifiziert, diese dienen vor allem zur Qualitätskontrolle, zum Verarbeiten und zum Zuordnen.
Exit-Kriterien, Fail- Kriterien: Ergeben sich aus dem Ablauf des POCs
Fallback: Der Fallback des Bilderuplads per Fehlermeldung soll es ermöglichen das der User immer sein Upload aus seiner Sicht erledigen kann oder wenn nötig weiß das er dies z.B. später ohne Probleme erledigen kann. Das System kann bei Störungen nicht auf alternative Funktionalitäten zurückgreifen, da der Upload als Kernfunktionalität nicht angemessen genug ersetzt werden könnte.
Ein Fehler beim verarbeiten der Metadaten der Bilder darf den wesentlich wichtigeren Ablauf des POC nicht stören, da diese wenn nötig auch zu einem späteren Zeitpunkt ergänzt werden können.
Die Begrenzung der Uploadbaren Bilformate auf z.B. nur jpq ist eine Option um die Funktionalität zusichern, würde aber auch die User Experimente negativ einfließen.

Gelaufener POC: Der Úpload des Bilds hat geklappt, jedoch ist aus Zeitgründen die Speicherung und die gängigen Bildformate wie im Fallback definiert nicht enthalten.
Aufruf eines gespeicherten Bildes
Bei diesem POC soll sichergestellt werden das die User des User-Contents eine entsprechende Anzeige erhalten, um eine Irritation zu verhindern.
Gelaufener POC: Durch Probleme bei Zugriffsrechten unter Android und Fehler beim schreiben in der realtime Datenbank, wird das Bild aus dem programm Speicher zur Laufzeit geladen.

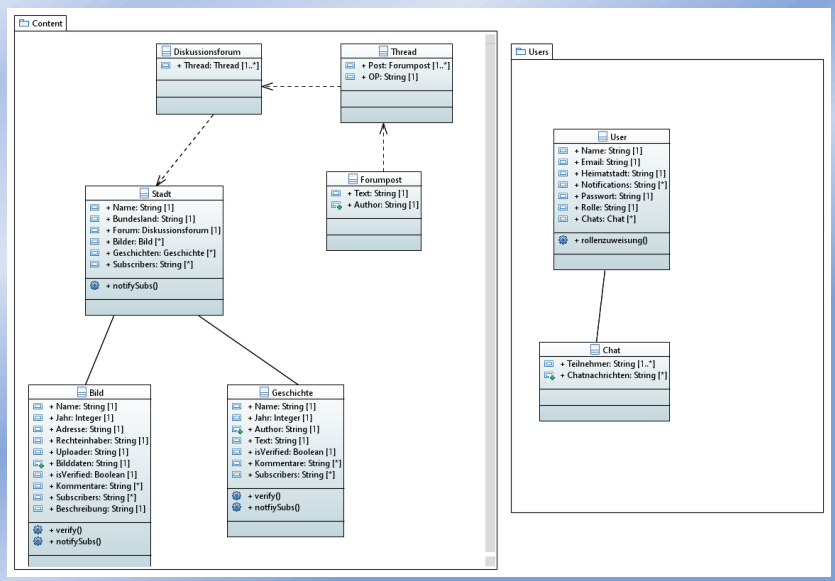


Dieser Code implementiert eine Pub/Sub-Funktionalität für eine Android-App. Es gibt zwei Schaltflächen in der Benutzeroberfläche, eine zum Abonnieren (subscribe) und eine zum Veröffentlichen (publish). Beim Klicken auf die Schaltfläche „Subscribe" wird geprüft, ob der Benutzername bereits in der Liste der Abonnenten enthalten ist. Wenn nicht, wird der Benutzername zur Liste hinzugefügt. Wenn der Benutzername bereits in der Liste vorhanden ist, wird er aus der Liste entfernt und eine Meldung „Unsubscribed" angezeigt. Beim Klicken auf die Schaltfläche "Veröffentlichen" soll ein Abonnent die Möglichkeit haben, ein Bild zu einer Stadt hochzuladen und anschließend werden alle Abonnenten in der Liste benachrichtigt. Wenn ein User nicht subscribed ist, hat er nicht die Möglichkeit, ein Bild zu einer Stadt hochzuladen.



Im Rahmen des Login PoCs stand im Raum das Speichern und den Vergleich von Passwörtern mittels hash-codes umzusetzen. Selbstverständlich dürfen Passwörter nicht in Klartext gespeichert werden, allerdings fehlte us die Zeit sich mit dem Thema auseinanderzusetzen. Da wir aber auschlieslich mit Dummy-Usern und demnach Dummy-Passwörtern arbeiten, ist das Speichern dieser in Klartext zu verzeihen. Der PoC diente dann daraufhin die Deserialisation einer JSON Datei mit den Usernamen und Passwörten zu einer Liste mit Userobjekten zu erproben. Dies war erfolgreich und diese Funktionalität wird auch im Prototypen wiedeverwendet. Wäre dies nicht erfolgreich gewesen hätte einen einzelnen User fest erzeugt und den Login übersprungen oder eine Userliste zur Laufzeit fest erzeugt und damit den Login simuliert.

Klassen-diagramm



Zum Klassen-Diagramm

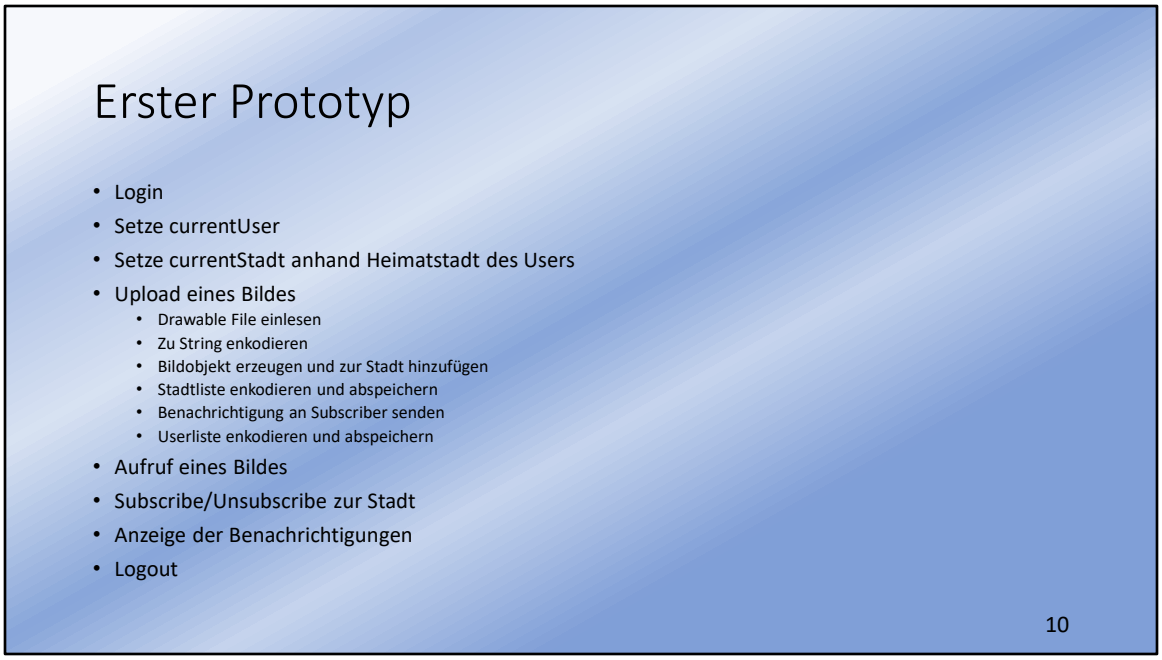
9

Im Rahmen der Modellierungen für das System wurde ein Klassendiagramm angefertigt, um damit festzulegen welche Informationen und Daten abgespeichert werden. Außerdem erfolgt hierdurch eine klare Struktur des Systems, es wird ersichtlich, dass zur Speicherung der Daten zwei Listen ausreichen. Eine Liste aller Städte, die jeweils die Bilder, Geschichten und ihr Forum beinhalten und eine Liste aller User.

Im Zuge der Programmierung der PoCs und des Prototypen durchlief das Diagramm einige Iterationen. So wurde zunächst vorgesehen, dass die unterschiedlichen Rollen der User als Erben implementiert werden. Da aber sich keine einzigartigen Attribute für diese Rollen ergeben haben, wurde darauf verzichtet und die Rolle wird als einfacher String gespeichert. Beim Aufruf einer Funktion, die eine bestimmte Rolle benötigt, wird dieses Attribut des aktuellen Users geprüft. Die vorgesehenen Rollen sind regulärer User, blockierter User, Experte (Verifikation), Moderator (Löschen von Beiträgen) und Administrator (Rollenzuweisung).
Außerdem war noch bis zur Programmierung unklar wie die Pub/Sub Funktionalität implementiert wird. Die dabei entwickelten Funktionen und Attribute wurden daraufhin zur Vollständigkeit im Klassendiagramm integriert.
Schlieslich ist noch aus der Programmierung erfolgt, dass vorher in den

Subscriberlisten, Autoren, Uploader, usw. User als Typ vorgesehen waren. Allerdings würde dies dazu führen, dass die User mehrfach abgespeichert werden in diesen jeweiligen Attributen und der allgemeinen Liste aller User. Es ist nun vorgesehen nur die Namen zu speichern und bei Aufruf mit der Userliste zu vergleichen.

Die Klassen zur Implementierung eines Forums und Chatfunktionen sind noch grob gehalten, da unklar ist, ob wir diese Funktionen in dem Prototypen implementieren. Sollte dies der Fall sein ist aber damit schon mal eine Grundlage geschaffen, die dann erweitert werden kann.



Im ersten Prototypen wurde versucht die Kernfunktionalitäten des Systems umzusetzen, der Upload von Bildern sowie damit einhergehende Pub/Sub Funktionalität. Die Grundlage für die Lösungsansätze wurden durch die PoCs, die Use Case Spezifikation sowie das Klassendiagramm gegeben. Hier ist anzumerken, dass eine detailliertere Interaktionsmodellierung die Arbeit hätte erleichtern können. Der Login erfolgt nach dem im dazugehörigen PoC getestetem Prinzip. Der damit eingeloggte User wird bis zum Logout zum currentUser. Daraufhin wird eine Liste aller Stadtobjekte geladen und anhand der Heimatstadt des Users die currentStadt gesetzt. Dem User werden daraufhin 4 Buttons angeboten (Upload, Aufruf, Sub/Unsub, Logout). Beim Upload sowie beim Aufruf eines Bildes wird aktuell noch ein hardcoded Bild verwendet. Beim Upload wird der Filepath zu diesem Bild fest verwendet und der Aufruf greift stets auf das erste Bild in der Bilderliste der Stadt zu. Dies gilt es in der weiteren Entwicklung so zu erweitern, dass ein beliebiges Bild hochgeladen und aufgerufen werden kann. Die Pub/Sub Funktionalität ist dahingegen im Grunde vollständig implementiert. Zu bedenken wäre nur ob die Verwaltung der eingegangenen Benachrichtigungen noch zu erweitern ist.

Ein wichtiger Punkt bei diesem Prototypen ist, dass die Datenverwaltung auschlieslich über JSON erfolgt. So werden (1) die Bildaten zu einem String enkodiert, (2) diese in

einem Bildobjekt gespeichert, (3) was Teil eines Stadtobjekts ist, (4) welches Teil einer Liste aus Stadtobjekten ist, (5) die wiederrum zum einem JSON-String enkodiert wird und (6) dieser String wird in eine entsprechende Datei geschrieben. Diese Datei, diese Liste wird zum Beginn der Activity geladen. Dies bedeutet, dass der komplette Inhalt der Datenhaltung in den Speicher geladen wird. Ab einer bestimmten Größe ist dies nicht mehr angemessen. Dieser Problematik sind wir uns bewusst aber wegen des Umfangs unseres Prototypen wird dieser Zustand in Kauf genommen. Die Alternative wäre ein Datenbanksystem einzubinden, darauf wurde aus Zeitgründen verzichtet.

Audit 4 Deliverables

- Use Cases iterieren (KW03)
- funktionalen Prototyps (KW07)
- Fazit und kritisch reflektiertes Prozessassessment des gesamten Projektes anhand der ursprünglichen Zielsetzung (KW08)