Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung

Mario Kaufmann
Pascal Birnstill
Erik Krempel

Implementierung

Version 1.0

Privacy Crash Cam App für Android

Fabian Wenzel
Giorgio Gross
Christoph Hörtnagl
David Laubenstein
Josh Romanowski

25. Februar 2017

Inhaltsverzeichnis

1 Einleitung **4**

2 Änderungen (Code) **5**
2.1 App . . . 5
2.1.1 AsyncPersistor . . . 5
2.1.2 Encryptor . . . 5
2.1.3 Ringbuffer . . . 5
2.1.4 MemoryManager . . . 6
2.1.5 CameraHandler . . . 6
2.2 Web-Dienst . . . 7
2.2.1 ServerProxy . . . 7
2.2.2 Main . . . 7
2.2.3 AccountManager . . . 7
2.2.4 VideoManager . . . 7
2.2.5 Account . . . 7
2.2.6 Metadata . . . 7
2.2.7 LocationConfig . . . 7
2.2.8 VideoProcessingChain . . . 8
2.2.9 VideoProcessing.Chain . . . 8
2.2.10 OpenCVAnonymizer . . . 8
2.2.11 OpenCVFilter . . . 8
2.3 Interface . . . 8
2.3.1 Login . . . 8
2.3.2 AccountView . . . 8
2.3.3 AccountDataManager . . . 9
2.3.4 Download . . . 9
2.3.5 Account . . . 9
2.3.6 Video . . . 9
2.3.7 MailService . . . 9

3 Änderungen (Projekt) **10**
3.1 App . . . 10
3.1.1 Mp4Parser . . . 10
3.2 Web-Dienst . . . 10
3.2.1 OpenCV . . . 10
3.2.2 Xuggler . . . 10

3.3 Web-Interface 10
3.3.1 Commons-Mail 10
3.4 Allgemein 11
3.4.1 Mocktio 11
3.4.2 JSON 11

4 Implementierungsplan **12**
4.1 Einleitung 12
4.2 Ursprünglicher Implementierungsplan 13
4.3 Tatsächlicher Implementierungsplan 14
4.4 Gründe für die Unterschiede 15

1 Einleitung

Dieses Dokument versucht einen Überblick darüber zu geben, welche Änderungen vorgenommen wurden, um aus unserem Entwurfsdokument für die *PrivacyCrashCam* ein vollständiges, funktionierendes Programm zu erstellen. Unser Ziel ist hierbei, die Gründe für die Änderungen zu erläutern und aufzuzeigen, wie der Entwurf geändert wurde um Probleme zu beheben.

Daher wird zu jedem Modul kurz beschrieben, ob, was (Attribute, Methoden, Parameter) und wieso verändert wurde (2). Zusätzlich zu den Änderungen, die im Code notwendig waren, werden auch Änderungen am Projekt (3), wie z.B. zusätzliche Abhängigkeiten erläutert. Am Schluss des Dokuments befindet sich ein Vergleich zwischen unserem ursprünglichen und tatsächlichen Implementierungsplan (4) und entsprechende Erläuterungen, weshalb sich diese gegebenenfalls unterscheiden (4.4).

2 Änderungen (Code)

2.1 App

2.1.1 AsyncPersistor

Dem Konstruktor muss zusätzlich der Context übergeben werden, damit das Public Key File für das Encryption-Modul geladen werden kann.

2.1.2 Encryptor

Die Methode encrypt() von Encrpytor nimmt nun die Parameter encrypt(File[ ] input, File[ ] output, InputStream publicKey, File encKey). Da nun File Arrays übergeben werden wird erlaubt eine beliebige Anzahl Files mit einem einzigen SecretKey zu verschlüsseln und dadurch eine bessere Erweiterbarkeit erreicht.

Der InputStream ist nötig, da Android nur Klassen mit zugriff auf den Context erlaubt Resourcen zu laden. Dies wirkt sich auch auf IKeyEncryptor aus.

2.1.3 Ringbuffer

Uns war von Beginn an klar, dass der Ringbuffer der App, der die Video-Aufzeichnungen der App vor der Persistierung puffert, eine Herausforderung werden würde. Daher haben wir uns bereits in der Entwurfsphase intensiv mit der Umsetzung auseinander gesetzt und uns dafür entschieden, kurze Video-Stücke in einer Warteschlange zu speichern und diese beim Persistieren zusammenzufügen.

Der MediaRecorder, der die Video-Stücke aufnimmt, kann entsprechend parametrisiert werden, so dass er für ein festes Zeitintervall aufnimmt, und dann die Aufnahme beedet. Die Beendigung der Aufnahme wird dem CameraHandler mitgeteilt, worauf hin dieser mit der Aufnahme des nächsten Video-Stücks starten kann. Bei der Implementierung stießen wir jedoch auf das Problem, dass der MediaRecorder asynchron zum Schreiben des Videos Rückmeldung gibt. Somit ist nicht garantiert, dass das Video-File vollständig geschrieben wurde. Aus diesem Grund mussten wir einen Mechanismus entwickeln, um sicherzustellen, dass die Files fertig geschrieben werden, bevor wir versuchen zu persistieren. Unser erster Versuch war einen Android-FileObserver auf die Files zu setzen und dadurch Rückmeldung zu erhalten, wenn die Files fertig geschrieben sind. Beim Abrufen der Daten muss dann für jedes File in der Warteschlange gewartet werden, bis es geschrieben wurde. Die Funktion getData() wurde in demandData() umbenannt um klarzustellen, dass eventuell Wartezeit auftreten kann. Es

wurden jedoch nicht alle Events empfangen, da manche Files bereits vor dem Einfügen in die Warteschlange vollständig geschreiben wurden.

Der zweite Ansatz wurde auf einem eigenen Branch entwickelt: Die Gundidee war ein FileLock zu verwenden, welches das Video-File *locken* können sollte, wenn der MediaRecorder den Schreibvorgang beendet hatte. Allerdings wurde kein Lock vom MediaRecoder gehalten, weshalb dieser Ansatz unser Problem ebenfalls nicht löste.

Die Problemlösung brachte schließlich user dritter Ansatz: Der FileObserver wurde auf den Ordner mit den Videos gesetzt anstatt auf die Videos selbst. Somit wurde er über alle Änderungen in dem Ordner benachrichtigt, unabhängig davon, wann genau sie gemacht werden.

Da eine generische Impementierung des RingBuffers den spezifischen Anforderungen der File-Beobachtung nicht mehr gerecht werden konnte, entschieden wir uns ihn zu VideoRingbuffer umzubenennen.

Für eine erweiterte Funktionalität wurden die Methoden flushAll() und den Inhalt des Puffers zu leeren und destroy() zum Löschen des Puffers hinzugefügt.

2.1.4 MemoryManager

Die Handhabung temporärer Files wurde erweitert, sodass jede neue Memeory-Manager Instanz einen neuen Ordner als aktuellen Ablageort temporärer Daten erzeugt. Dies erleichtert die Arbeit mit den Files, die innerhalb eines Ordner liegen, da garantiert ist, dass der Ordnerinhalt nicht von anderen Threads modifiziert wird.

Das Allozieren und Löschen der temporären Files brachte in Android jedoch unerwartet Probleme mit sich. Temporäre Files sollten beim beenden der App gelöscht werden, allerdings wartete unsere App meistens vergeblich auf die Benachrichtigung, der Nutzer hätte die App beendet. Die Folge war eine Anhäufung der temporären Files ohne Chance diese löschen zu können. Daher haben wir die Methode deleteCurrentTempData() und deleteAllTempData() hinzugefügt.

2.1.5 CameraHandler

Da die Implementierung des CameraHandler-Interfaces recht schnell komplex und unübersichtlich werden kann, haben wir Bequemlichkeits-Methoden eingefügt, die einen Lifecycle realisieren und von Klienten in der vorgegebenen Reihenfolge aufgerufen werden. Somit haben Klienten eine bessere Kontrolle über den Handler und können ihn mit ihren eigenen Lifecycles, also dem Lifecycle der Fragments, SurfaceViews oder Activities, synchronisieren. Dieses vorgehen bündelt darüber hinaus kritische Operationen, wie den Kamera-Zugriff, und sensible Anweisungen die zu einer bestimmten Zeit geschehen sollten, wie das löschen der temporären Files, in den neu eingefügten Methoden. Das Ergebnis ist ein klarerer und leichter lesbarer Code.

2.2 Web-Dienst

2.2.1 ServerProxy

Der upload()-Methode wurde ein weiterer Parameter hinzugefügt um Informationen über den Video-Namen weitergeben zu können.

getVideosByAccount() wurde zu getVideos() umbenannt, da der Parameter accountData den Suffix redundant macht.

2.2.2 Main

Es wurde eine Funktion restartServer() hinzugefügt, falls man zusätzliche Logik einfügen möchte, die nicht der startServer() und stopServer() Routine entspricht (z.B. zum schnelleren Start bei täglichen Neustarts).

2.2.3 AccountManager

Die Methoden setMail() und setPassword() wurden zu changeAccount() zusammengefasst, um die Schnittstelle zu vereinfachen.

Damit der man der Account Klasse das Salt zum hashen mitgeben kann wurde die Funktion getSalt() hinzugefügt, die das Salt des Accounts aus der Datenbank abfragt.

2.2.4 VideoManager

Da Web-Dienst und Web-Interface zwei vollständig unabhängige Projekte sind ist eine Kommunikation über die Datencontainerklasse VideoInfo nicht direkt möglich. Um dennoch Informationen austauschen zu können wurde die Methode getVideoInfoList() so abgeändert, dass sie nun einen JSON-String erstellt, der versendet und auf dem Interface wieder ausgelesen werden kann.

2.2.5 Account

Das lesen des Hash-Salt aus der Datenbank wird vom AccountManager geregelt, da die Account Klasse keinen Zugriff auf die Datenbank hat. Daher war es notwendig die Methode hashPassword() öffentlich zu machen.

2.2.6 Metadata

Die Typen von date (String -> long) und von gForce (Vector3D -> float[]) wurden angepasst und dem Pendant der App zu entsprechen.

2.2.7 LocationConfig

Der LocationConfig mussten weitere Locations für Resources-Ordner hinzugefügt werden, falls man die Ordnerstruktur verändern möchte.

Da wir uns entschieden haben einen eigenen Ordner für log-Files zu erstellen wurde auch dieser hinzugefügt.

2.2.8 VideoProcessingChain

Um es bei der Fehlerbehandlung zu ermöglichen einzelne Chains aufzuräumen, wurde eine Funktion cleanUp() hinzugefügt. Diese dupliziert zur Zeit die Funktionalität von deleteTmpFiles(), ist aber nützlich falls man die Fehlerbehandlung erweitern möchte.

Es wurden getter für response und videoName hinzugefügt.

2.2.9 VideoProcessing.Chain

Um die VideoProcessingChain besser zu organisieren wurde das Paket in Unterpakete für die komplexen Bearbeitungsschritte (z.B. Anonymisierung) aufgeteilt

2.2.10 OpenCVAnonymizer

Die Klassen Anonymizer, ExampleAnalyzer und ExampleFilter wurden in OpenCVAnonymizer, OpenCVAnalyzer und OpenCVFilter umbenannt, um aussagekräftigere Namen zu geben.

2.2.11 OpenCVFilter

Der Typ von detections in der methode filter() wurde von Rect in RectVector geändert, da eine andere Version von OpenCV verwendet wird, wie zunächst angenommen und diese einen andere Bezeichnung für die Klasse verwendet. Die Funktionalität bleibt gleich.

2.3 Interface

2.3.1 Login

Die LoginView wird bei jedem Start angezeigt noch bevor der Navigator erzeugt wird. Dadurch wird verhindert, dass durch vor und zurückspringen der Login übergangen werden kann. Zusätzlich hat die MyUI Klasse eine Methode login() bekommen, die den Navigator und das Menü erzeugt und dann zur VideoView wechselt.

2.3.2 AccountView

Die AccountView hat eine Referenz auf die UI bekommen. Somit kann nach Ändern oder Löschen eines Accounts der Benutzer ausgeloggt werden.

2.3.3 AccountDataManager

Die Klasse hatte einen Account als Klassenattribut. Dies wäre zum Problem geworden wenn mehrere Benutzer gleichzeitig eingeloggt sind, da das Attribut ständig überschrieben würde. Deshalb wurde dieses Attribut durch ein von Vaadin bereitgestelltes Sessionattribut, das nur während einem Aufruf der Seite gültig ist, ersetzt.

2.3.4 Download

Der VideoDownload über den, von Vaadin bereitgestellten, FileDownloader stellte uns vor einige Herausforderungen, da dieser verlangte, die herunterzuladende Ressource bereits bei der Initialisierung bereit zu stellen. Zudem muss der erstellte FileDownloader auf einen Button angemeldet werden, der dann den Download startet.

Unser erster Versuch war durch einen ersten Button den FileDownloader zu erzeugen und dann durch einen zweiten den Download zu starten. Dabei wurde das Video beim ersten Klick als File geladen. Da der FileDownloader jedoch keine Statusmeldung bereit stellt wussten wir nicht, wann wir das File wieder löschen können.

Darauf hin versuchten wir das Video als Stream bereitzustellen. Allerdings hält der FileDownloader den Stream bis zum tatsächlichen Download offen, was dazu führte dass es zu Timeouts kam, wenn man zu lange wartete.

Schließlich haben wir einen File-Proxy geschrieben, der gegenüber dem FileDownloader als fertige Ressource agiert, das Video allerdings erst anfragt, wenn der Download tatsächlich gestartet wird, sodass keine Timeouts entstehen. Dies hat zudem erlaubt den ersten Button zu entfernen.

2.3.5 Account

Da wie bei dem VideoManager keine Datenobjekte zwischen Interface und Dienst ausgetauscht werden können, wird auch hier nun ein JSON-String erstellt. Dafür wurde die Methode getAsJson() eingeführt.

2.3.6 Video

Beim Erzeugen der Liste der Videos, werden zuerst die Videos in die Liste eingefügt und anschließend die Metadaten hinzugefügt. Deshalb wird ein Konstruktor in der Klasse Video benötigt, mit dem man ein Video ohne Meta-Informationen erzeugen kann.

2.3.7 MailService

Die Klasse hat eine Methode bekommen, mit der sie E-Mail-Adressen auf korrekte Syntax prüfen kann.

3 Änderungen (Projekt)

3.1 App

3.1.1 Mp4Parser

Um die einzelnen kurzen Videostücke bei der Persistierung zusammenzuschneiden verwenden wir die Bibliothek Mp4Parser.

3.2 Web-Dienst

3.2.1 OpenCV

Die Anbindung von OpenCV über Maven erwies sich als schwierig, da OpenCV keine offizielle Maven-Schnittstelle anbietet. Dies führte dazu, dass wir OpenCV selbst in Maven integrieren mussten. Da jedoch die Integration von nativen Bibliotheken über Maven vermehrt zu Fehlern geführt hat, haben wir uns entschieden JavaCV von bytedeco zu verwenden. JavaCV bietet eine eigene mavenfähige Java-Schnittstelle zu OpenCV und verwaltet native Abhängigkeiten automatisch. Zudem wird zusätzlich FFMpeg verwaltet, das wir zur Bearbeitung von Mp4-Videos zusätzlich benötigten.

3.2.2 Xuggler

Um Metadaten zu Mp4-Videos auszulesen haben wir die Bibliothek Xuggler hinzugefügt. Xuggler bietet zudem viele Funktionalitäten die weitere Videomanipulationen ermöglichen, z.B. Resizing.

3.3 Web-Interface

3.3.1 Commons-Mail

Um zu überprüfen, ob ein Nutzer bei der Registrierung eine korrekte E-Mail-Adresse angibt verwenden wir die Bibliothek commons.mail.

3.4 Allgemein

3.4.1 Mocktio

Um das Testen zu vereinfachen verwenden wir zusätzlich zu JUnit auch Mockito und PowerMocktio. Insbesondere in der App ist dies notwendig, da Android verlangt für nicht Instrumentierte Tests alle Android Funktionalität zu mocken.

3.4.2 JSON

Um ein einheitliches Format zur Informationsübertragung zwischen den einzelnen Komponenten sicherzustellen verwenden wir JSON. Für unser Projekt haben wir uns für die JSON-Funktionalität von org.json entschieden.

4 Implementierungsplan

4.1 Einleitung

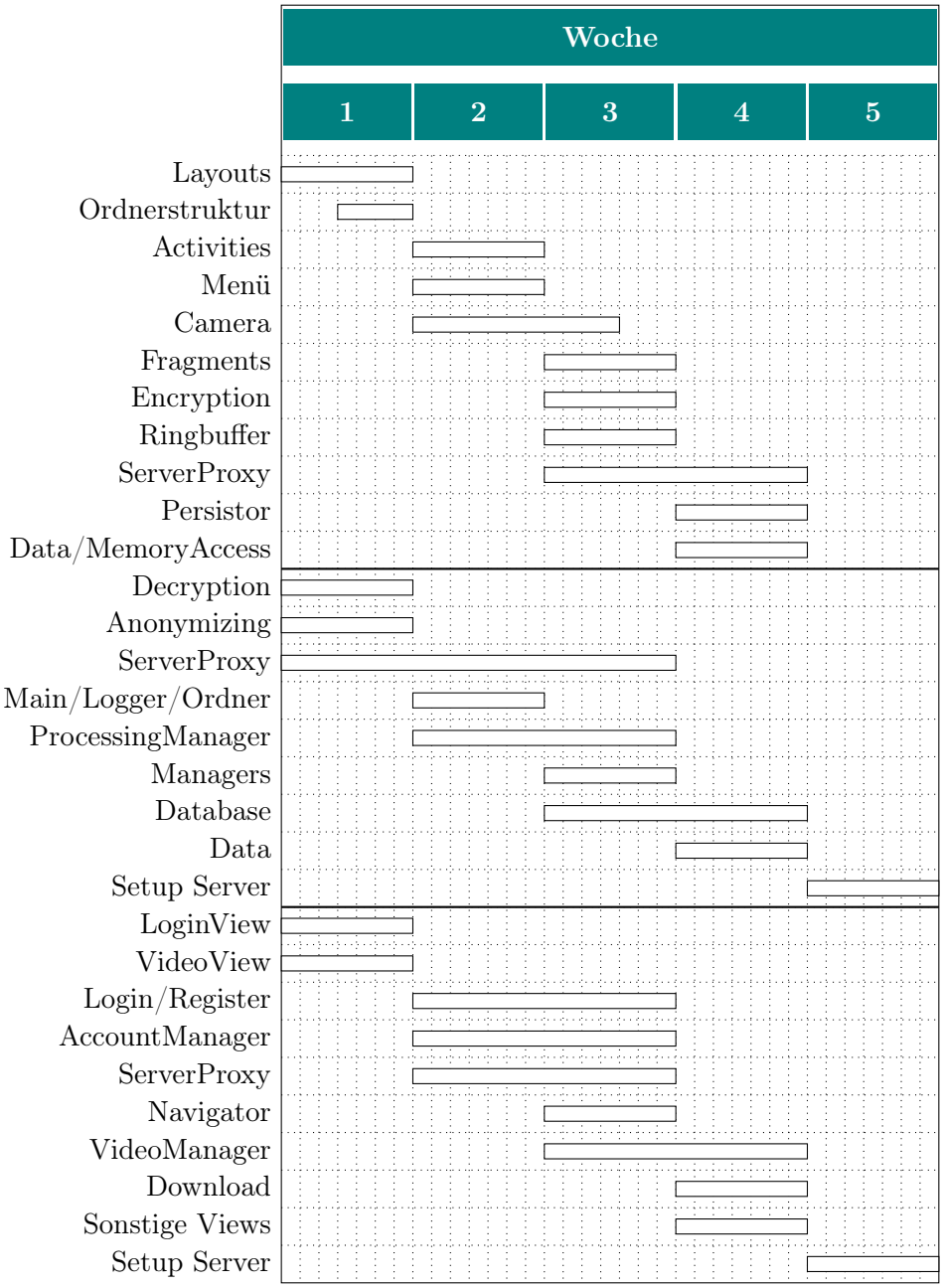
Um eine schnelle und reibungslose Implementierung zu garantieren haben wir einen Plan erstellt, in dem wir festhalten, welches Modul wann implementiert wird.

Bei der Erstellung des Plans haben wir Module und Funktionen priorisiert um eine gestaffelte Implementierung zu realisieren. Das heißt, dass zunächst die Grundfunktionalität implementiert wird und danach schrittweise weitere, immer weniger essentielle Komponenten hinzugefügt werden. Dadurch wird gewährleistet, dass Probleme, die die Umsetzung der *PrivacyCrashCam* als Ganzes gefährden, frühzeitig erkannt und behandelt werden können.

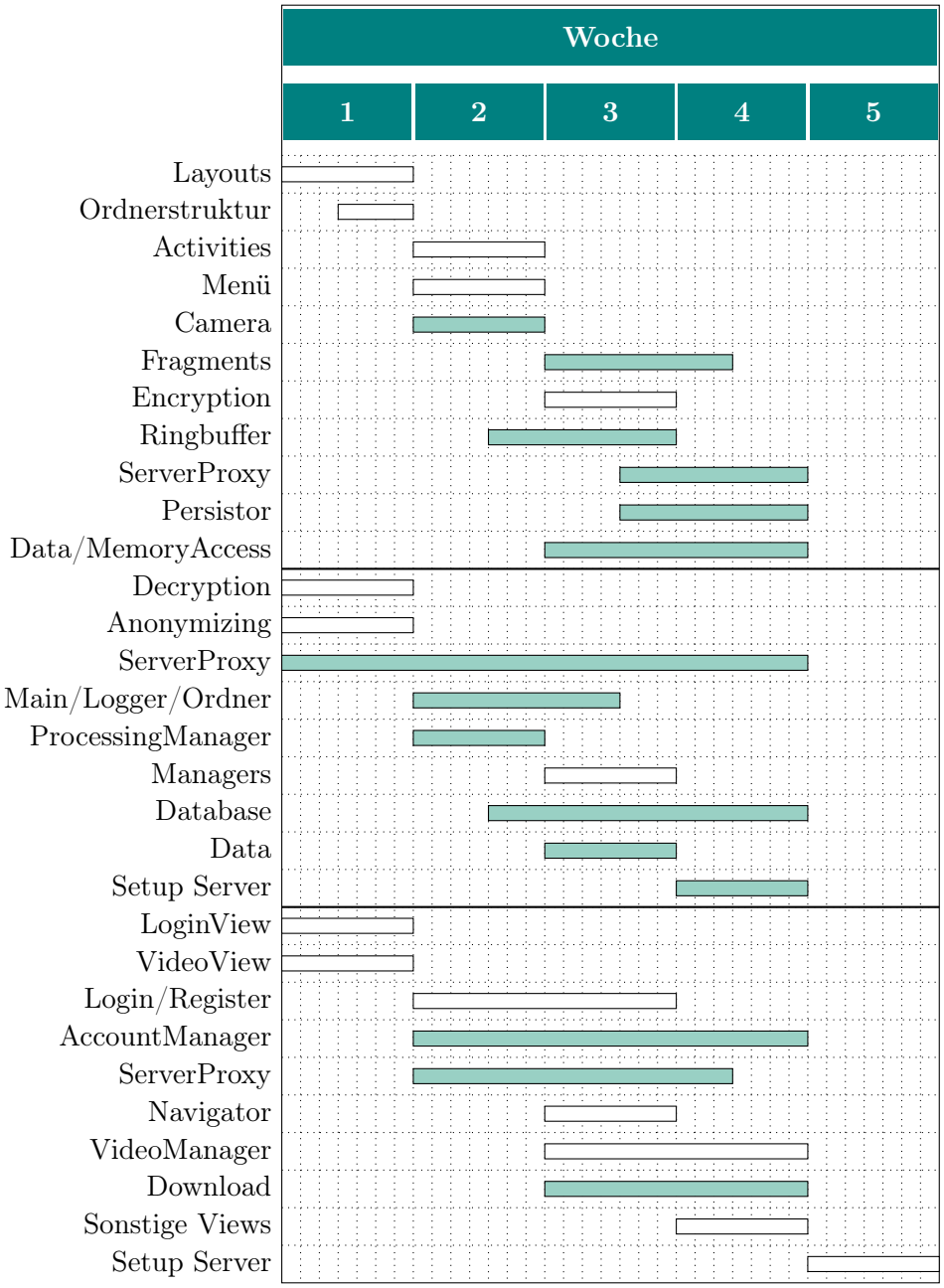
So wurden Funktionen, wie z.B. die VideoProcessingChain in dem Web-Dienst oder die Camera in der App zuerst geschrieben und unwichtigere Elemente wie z.B. die Datenschutzerklärung, oder die Bestätigungsmail beim Anmelden später implementiert. Dies erlaubte uns früh Probleme wie z.B. den Ringpuffer (2.1.3) zu erkennen und zu beseitigen und so eine reibungslose Kernfunktionalität zu gewährleisten.

Ein Implementierungsplan erleichtert zudem die Koordination zwischen den Gruppenmitgliedern, da jeder weiß welche Module schon integriert werden können und bei welchen man beachten muss, dass sie noch implementiert und getestet werden.

4.2 Ursprünglicher Implementierungsplan



4.3 Tatsächlicher Implementierungsplan



4.4 Gründe für die Unterschiede

- **Ringbuffer/Persistor**
  Durch bereits genannte Probleme mit der Umsetzung des Ringbuffers (2.1.3) hat sich die Fertigstellung verzögert. Da der Persistor von den Daten des Ringbuffers abhängt, hat auch er sich verzögert.

- **Data/Memory Access**
  Nicht alle Methoden des MemoryAccess-Moduls wurden direkt benötigt. Daher haben wir die Fertigstellung des Moduls verzögert, um zuerst den Persistor fertig stellen zu können.

- **ServerProxy**
  Da nach und nach die Funktionalität des Web-Interfaces und der App erweitert wurden, stellten sich auch immer wieder neue kleine Probleme mit dem ServerProxy heraus, wodurch immer wieder kleine Änderungen nötig wurden die die Fertigstellung verzögerten.

- **Database**
  Weil die Implementierung des Passwordhashens eine Anpassung des DatabaseManagers erforderte, musste der eigentlich abgeschlossene Manager noch einmal aufgegriffen werden.

- **ServerProxy/AccountManager (Interface)** Da der der ServerProxy des Interfaces mit dem ServerProxy des Web-Dienstes abgeglichen werden musste (4.4) hat sich die Fertigstellung des AccountManagers ebenfalls herausgezögert.

- **Download**
  Wie bereits erläutert erwies sich der Video-Download des Web-Interface als problematisch (2.3.4). Infolge dessen wurde auch hier ein Mehraufwand nötig.

- **Beschleunigungen**
  Durch die Vorarbeit in der Entwurfsphase konnten einige Module schneller abgeschlossen werden als zunächst vermutet. Einzelne Funktionen waren bereits geschrieben und getestet und mussten daher nur noch eingefügt werden. Außerdem hatten wir uns schon vor der Implementierung mit den Technologien auseinandergesetzt, was den Implementierungsprozess beschleunigte.
  - Camera-Modul App
  - VideoProcessingManager Web-Dienst
  - Server Setup Web-Dienst
  - Data-Modul Web-Dienst