TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Professur für Mediengestaltung

Dokumentation über das OUTPUT-Komplexpraktikum im Sommersemester 2020

Philipp Matthes

Geboren am: 12. März 1997 in Chemnitz
Studiengang: Diplom Informatik (PO 2010)
Matrikelnummer: 4605459
Immatrikulationsjahr: 2016

Projektarbeit

Betreuer
Dipl.-Medieninf. Franziska Hannß

Eingereicht am: 25. Oktober 2020

Inhaltsverzeichnis

1 Einleitung

Die jährlich stattfindende OUTPUT.DD Projektschau an der Technischen Universität Dresden lockt immer wieder zahlreiche Besucher in die Fakultät Informatik und begeistert für aktuelle Forschungs- und Projektthemen. Dabei besuchen nicht nur Schüler die Veranstaltung, sondern auch Studenten und Mitarbeiter der Universität, aber auch viele lokale IT-Unternehmen aus dem „Silicon Saxony“. Damit ist OUTPUT.DD nicht nur eine Projektschau, sondern auch ein Hotspot für die Kommunikation und Diskussion von technischem Fortschritt. Die Veranstaltung wird jährlich von Studenten der Fakultät mitgestaltet und -organisiert. Begleitend zur Veranstaltung wird auch eine App für die Betriebssysteme iOS und Android bereitgestellt, welche sich im Rahmen mehrerer Forschungsprojekte entwickelt hat.

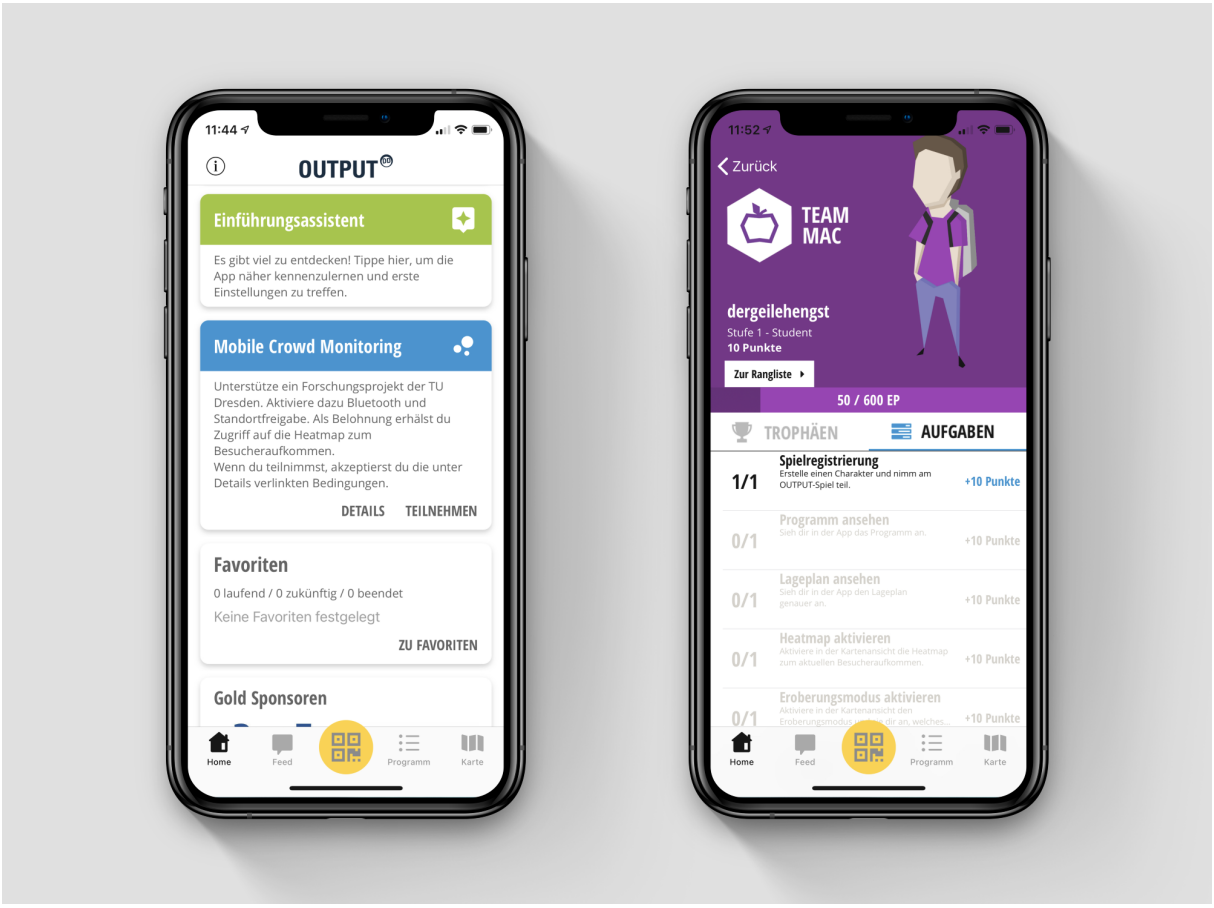

Abbildung 1.1 Die OUTPUT.DD App im Profil.

Die App bietet hierdurch neben der interaktiven Anzeige von Veranstaltungsplänen zusätzlich auch ein eigenes Spiel und komplexere Funktionalitäten, wie die Anzeige des Besucheraufkommens auf einer kartenbasierten Heatmap. Nachdem ich im Wintersemester 2019/2020 vorrangig an der Gestaltung der OUTPUT.DD und der Betreuung des Social Media Auftritts beteiligt war, gründeten wir zu Beginn des Sommersemesters 2020 ein dediziertes App-Team, um die App für eine mögliche Veranstaltung in 2020 vorzubereiten und im Zuge dessen weiter zu verbessern.

1.1 Problemstellung und Motivation

Als grundlegender funktionaler Teil der App dient ein Datenbank-Backend, welches sich um die Synchronisation der relevanten App-Daten kümmert. Hierbei können Daten lokal angelegt, automatisiert an einen Server übermittelt und schließlich an andere App-Nutzer verteilt werden. Die über die jeweiligen App Stores verfügbare Applikation nutzt hierfür den Dienst Realm[1]. Aufgrund der hiermit verbundenen Nutzungskosten wurde jedoch entschieden, dass sich nun eine Migration auf ein anderes Datenbank-Backend anbietet. Im Rahmen der Evaluation verschiedener Dienste wurde hierfür schließlich das Couchbase-Framework [2] ausgewählt, welches auch einen eigenen Synchronisationsdienst inkludiert. Als unsere zentrale Aufgabe resultierte hieraus, die Systemarchitektur der OUTPUT.DD-App zu analysieren, die Integration des Couchbase-Frameworks zu planen und schließlich durchzuführen. Gleichzeitig sollten alle noch bestehenden Abhängigkeiten zu Realm getrennt werden, um schließlich eine vollständige Migration durchzuführen.

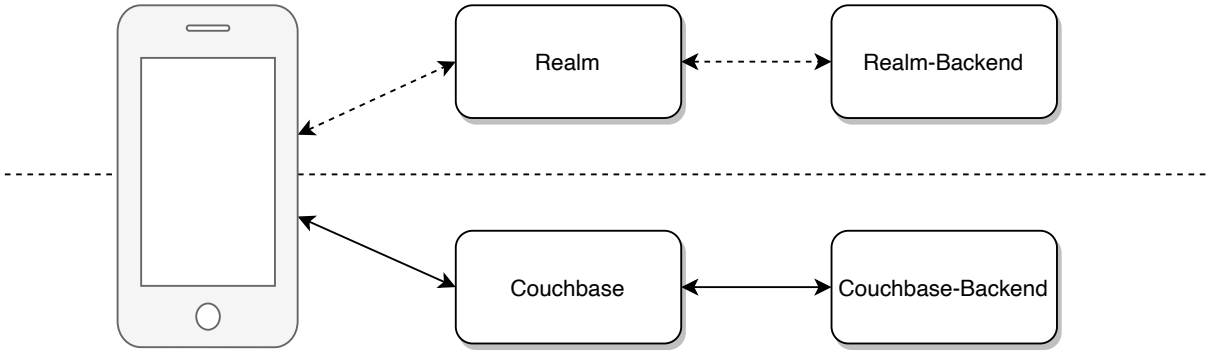

Abbildung 1.2 Das OUTPUT.DD-Realm Backend soll ersetzt werden durch ein Couchbase-Backend.

Aus einer im Rahmen der Projektplanung durchgeführten Analyse der bestehenden Datenbankstruktur gingen außerdem weitere Anforderungen hervor. Neben einer unklaren Trennung des Datenbank-Backends von der sonstigen Logik bestand das Kernproblem darin, dass bestimmte nutzerspezifische und schützenswerte Daten nach außen öffentlich verfügbar waren, obwohl diese Daten nicht benötigt wurden. So konnte beispielsweise nachvollzogen werden, welcher Nutzer zu welchem Zeitpunkt einen bestimmten Ort besucht („QR Code gescannt“) hatte. Aus der mangelhaften Datenbankstruktur offenbarte sich neben diesem Datenschutzproblem auch ein Datensicherheitsproblem in der Manipulierbarkeit des OUTPUT.DD-Spiels. Somit haben wir uns für den zeitlichen Rahmen des Sommersemesters das Ziel gesetzt, eine Migration auf das Couchbase-Backend durchzuführen und hierbei diese Schwachstellen zu eliminieren.

[1] Realm. `https://realm.io/` (Abgerufen am 19.10.2020)
[2] Couchbase. https://www.couchbase.com/ (Abgerufen am 19.10.2020)

1.2 Projektstart und Aufgabenverteilung

Betreut durch M.Sc. Julian Catoni und Dr. Thomas Springer trennten wir unsere Aufgabenbereiche bereits zu Beginn der Projektbearbeitung klar voneinander ab. Somit war es die zentrale Aufgabe von Fritz Windisch, eine Datenbankstruktur zu entwickeln, welche die oben genannten Datenschutz- und Datensicherheitsprobleme mitigiert. Gleichzeitig kümmerten sich Felix Kästner und ich um die Strukturierung und Abgrenzung der Datenbankfunktionalitäten in ein eigenes OUTPUT.DD-Couchbase-Framework sowie die Implementation mithilfe der entsprechenden nativen Couchbase-Bibliotheken, hierbei war meine Aufgabe die Umsetzung der iOS-Funktionalitäten.

2 Erstellung eines Couchbase-Frameworks

2.1 Separierung des Datenflusses

Um die in Abschnitt 1.1 beschriebenen Datenschutz- und Datensicherheitsprobleme zu mitigieren, mussten die bestehenden Datenflüsse zunächst analysiert und kategorisiert werden. Hierbei konnten wir feststellen, dass in der vorliegenden Version der App bestimmte Daten nur von der lokalen Instanz persistiert werden mussten, während andere Daten wiederum zum Server übermittelt werden mussten, um sie dort für weitere Datenverarbeitungsprozesse wie die Aggregation weiterer Daten zu verwenden.

Tabelle 2.1 Die in der Analyse zusammen mit Fritz Windisch analysierte Separierung der Daten in private und öffentliche Daten im alten Datenbankschema der OUTPUT.DD-App.

Private Daten:	Öffentliche Daten:
• Likes und deren Zeitpunkt • Konkrete erfüllte Aufgaben und Zeitpunkte der Erfüllung • Konkrete erhaltene Trophäen und Zeitpunkte des Erhaltens	• Leaderboard (inkl. Score, Nutzername, Team, Avatar, Erfahrungsstufe, Anzahl erfüllter Aufgaben und Trophäen) • Die aus Nutzerdaten aggregierte Heatmap • Statische Inhalte wie Veranstaltungen, Stände, QR-Codes, Aufgaben und Trophäen

Die bisherige Separierung der Daten in private und öffentliche Tabellen ist in Tabelle 2.1 gezeigt. Diese ist jedoch in großen Teilen, wie Fritz Windisch analysierte, nicht mit der Datenschutzerklärung der OUTPUT.DD-App vereinbar. Dies liegt daran, dass in dieser Form unter anderem auch personenbezogene Daten für alle Nutzer öffentlich zur Verfügung gestellt werden, welche jedoch nicht für die Funktionalitäten der App benötigt werden. Auf Grundlage dessen entwickelte Fritz Windisch eine neue Modellstruktur, welche lediglich die benötigten Daten öffentlich zur Verfügung stellt und alle weiteren Daten nach außen verbirgt, so dass nur das Gerät selbst auf die eigenen Informationen zugreifen kann. Hierzu wurden beispielsweise die Spielerdaten aufgespalten in einen öffentlichen und einen privaten Teil. Anschließend stellte uns Fritz Windisch seine Arbeit in Form einer Dokumentation zur Verfügung, auf Grundlage

derer wir die Struktur unseres Datenbank-Interfaces aufbauen konnten.

2.2 Konzeption eines Datenbank-Interfaces

Noch bevor wir die konkrete, von Fritz Windisch vorgeschlagene, Datenbankstruktur umsetzen konnten, befassten wir uns ausgiebig mit der Dokumentation des Couchbase Frameworks, um dessen Besonderheiten und Einschränkungen kennenzulernen. Neben der Synchronisationsfunktionalität des Couchbase Frameworks, hier genannt Replikation, besitzt dieses die weitere Besonderheit, dass Objekte als Dokumente gespeichert werden. Damit unterscheidet sich Couchbase von herkömmlichen SQL-Datenbanken, die Objekte in Tabellen speichern. Der dokumentenorientierte Ansatz von Couchbase wird daher auch als „noSQL" bezeichnet. Gleichzeitig bringt dieser dokumentenorientierte Ansatz eine Reihe von weiteren Problemen für die Migration der Datenbankstruktur mit sich, da die möglichen Datenbankoperationen zwar prinzipiell identisch sind, jedoch die in der Datenbank gespeicherten Objekte anders gehandhabt werden müssen, beispielsweise durch das Ergänzen von zusätzlichen Informationen. Wir erkannten schnell, dass sich diese internen Feinheiten der Datenbank gut durch einen Wrapper[1] abstrahieren, aufbereiten und über Schnittstellen anbieten lässt.

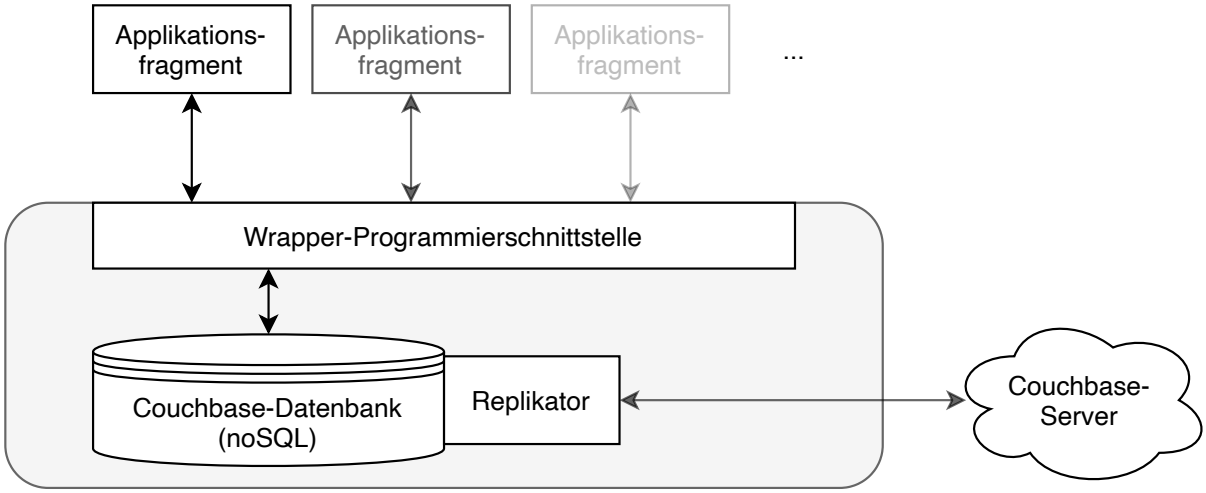

Abbildung 2.1 Um die internen Feinheiten der Datenbank zu abstrahieren, haben wir diese in einem Datenbank-Wrapper gekapselt und durch Programmierschnittstellen verfügbar gemacht. Durch den gekoppelten Replikator transduzierte Änderungen in den synchronisierten Daten können ebenfalls über diese Schnittstellen abonniert werden.

Unsere zentrale Idee für die Umsetzung des Datenbank-Wrappers war die Nutzung des sogenannten Repository-Patterns[2], bei dem die Kapselung der CRUD-Operationen[3] der Datenmodelle im Zentrum steht. Dies wird realisiert, indem die eigentlichen CRUD-Operationen in Funktionen eines Repository mit einem klaren Interface nach außen versteckt werden. Somit werden gezielt bestimmte Operationen, die für die Bereitstellung der Datenbankfunktionalitäten der OUTPUT.DD App benötigt werden, nach außen abgegrenzt.

[1] Wrapper-Pattern: Softwareentwicklungspattern, bei dem die Funktionalität einer Komponente gekapselt wird.
[2] Repository-Pattern: Softwareentwicklungspattern, bei dem die Datenoperationen auf einem Datenmodell gekapselt und gleichzeitig abstrahiert werden.
[3] CRUD-Operationen: Simple Operationen auf einem Datenmodell in einer Datenbank wie **C**reate, **R**ead, **U**pdate und **D**elete.

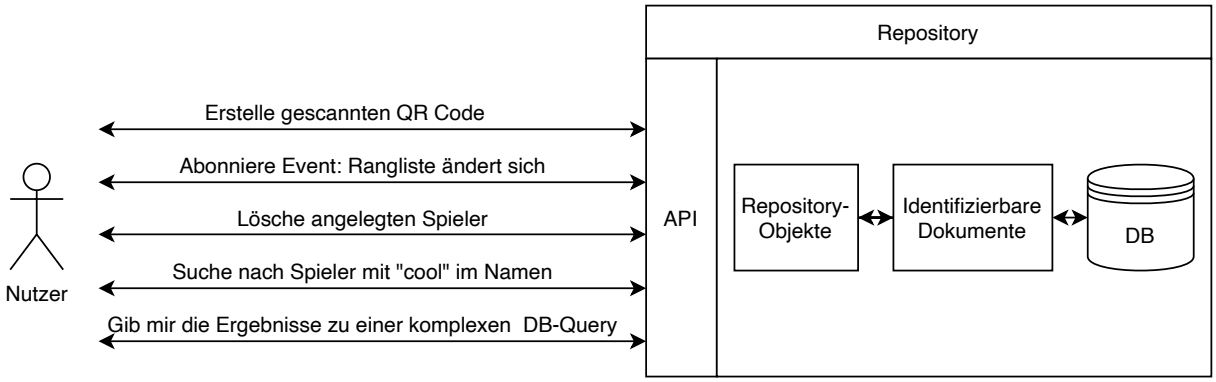

Abbildung 2.2 Nutzeranfragen an den Datenbank-Wrapper sind sehr unterschiedlich und benötigen zum Teil besondere Funktionalitäten (Beispiel Volltextsuche). Wir konnten diese Anforderung erfüllen, indem wir so genannte Repositories nutzen, bei dem deren APIs grundlegende CRUD-Operationen auf Objekten anbieten und diese (eventbasiert) observierbar machen. Diese werden im Repository in eine dokumentenbasierte Repräsentation überführt und entsprechend behandelt.

Bei der konkreten Implementation des Repository-Patterns haben wir uns dafür entschieden, die Objekte jeweils als einzelnes Dokument gemeinsam in eine Datenbank zu speichern. Die Objekte werden hierbei nicht, wie in SQL-Datenbanken, durch deren Tabelle voneinander abgegrenzt, sondern durch einen hinzugefügten `DocumentIdentifier`. Somit ist es trotzdem möglich, Objekte eines konkreten Typs (z.B. eine Liste von OUTPUT.DD Ständen) aus der Datenbank zu erhalten. Die Objekte müssen hierbei zu einem sogenannten `RepositoryObject` Interface konform sein und hierzu neben dem `DocumentIdentifier` auch weitere Informationen, wie deren (zusammengesetzten) Primärschlüssel bereitstellen. Außerdem gibt es hierdurch die Möglichkeit, bestimmte Attribute der Objekte zur Beschleunigung der Datenbankanfragen in einem Datenbankindex zusammenzufassen bzw. bestimmte Attribute eines Modells für die FTS[4]-Funktionalität von Couchbase freizugeben. Die weiteren technischen Details zur Umsetzung dessen sollen der Einfachheit halber hier nicht erwähnt werden.

[4]FTS: **F**ull **T**ext **S**earch zur Suche von Schlagwörtern in Texten.

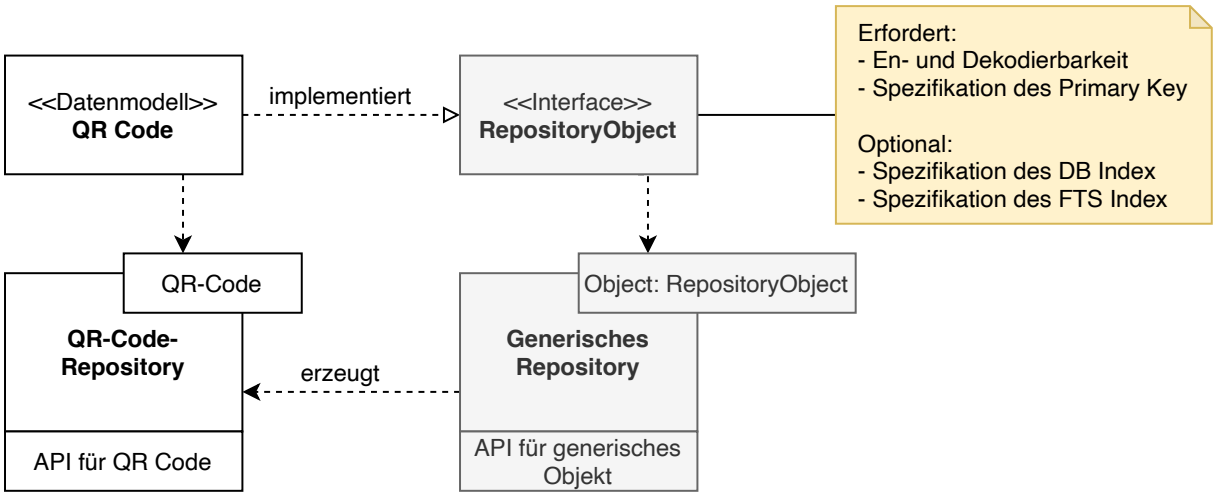

Abbildung 2.3 Zur Nutzung unseres generischen Repositories müssen Objekte lediglich wenige Requirements erfüllen, konkret müssen sie en- und dekodierbar sein und einen Primärschlüssel (wie in Datenbanken üblich) deklarieren. Danach kann vom generischen Repository ein konkretes Repository erzeugt werden, welches sofort alle (auf den Typ des konkreten RepositoryObjects) getypten Funktionen bereitstellt und genutzt werden kann, ohne weiteren Implementationsaufwand. Zusätzlich können Attribute deklariert werden, sollten bestimmte Anfragen an die Datenbank beschleunigt werden (durch das Ergänzen eines DB Index) oder eine Volltextsuche notwendig sein (durch das Ergänzen eines FTS Index).

Die von uns ermittelte Architektur besitzt mehrere zentrale Vorteile. Um die Repository-Architektur für ein konkretes Objektmodell zu verwenden, sind nur wenige Ergänzungen am Objektmodell notwendig, mit denen dieses Objekt in einem eigenen Repository gekapselt werden kann. Die Implementation des Repositories ist hierbei so generisch, dass sie alle notwendigen Operationen selbstständig generiert, die gesamte Dokumenterstellung und das Dokumentmanagement übernimmt und schließlich über standardisierte Schnittstellen nach außen anbietet. Gleichzeitig ist es trotzdem möglich, bei Bedarf konkrete noSQL-Operationen an die Datenbank weiterzugeben und auszuwerten. Somit besteht durch die Kapselung der Couchbase-Datenbank kein Nachteil im Funktionsumfang.

Konkretisierung. Die vorigen Abschnitte haben unser Konzept auf einer hohen Abstraktionsebene erklärt, sodass nun noch einmal konkret darauf eingegangen werden soll, wie wir die von Fritz Windisch vorgeschlagene Trennung in öffentliche und private Datenmodelle umgesetzt haben. Wir sind dabei so vorgegangen, dass wir die Modelle aus Fritz Windischs Architektur übernommen haben und diese in `Personal` (privat) und `Shared` (öffentlich) unterteilt haben. Die Handhabung der Objekte ist hierbei bis auf eine semantische Trennung jedoch prinzipiell gleich - sie werden vom Repository persistiert und synchronisiert, jedoch werden bei privaten Repositories vom Couchbase-Replikator-Service nur die eigenen Objekte synchronisiert, so dass die in Abschnitt 1.1 beschriebene Datenschutzprobleme mitigiert werden. Diese Funktionalität wurde von Fritz Windisch in einem eigenen Couchbase-Replikator-Service[5] implementiert, zusammen mit weiteren Services zur Realisation dieser Funktionalität[6][7], auf deren technische

[5] OUTPUT.DD-Syncserver. `https://bitbucket.org/mapbiquitous/output.dd-syncserver` (Abgerufen am 21.10.2020)
[6] OUTPUT.DD-Couchbase-Update-Notifier. `https://bitbucket.org/mapbiquitous/output.dd-couchbase-update-notifier` (Abgerufen am 21.10.2020)
[7] OUTPUT.DD-Registration-Service. `https://bitbucket.org/mapbiquitous/output.dd-registration-service` (Abgerufen am 21.10.2020)

Details (bspw. Nutzererstellung) im Rahmen der Dokumentation meiner Tätigkeiten nicht weiter eingegangen werden soll. Durch die generische Trennung in `Personal`/`Shared` sowie in `Repository`/`RepositoryObject` konnten wir somit ein in unseren Augen sehr flexibles Framework erstellen, welches sich durch die Verbindung mit dem Couchbase-Replikator-Service durch die erforderliche Synchronisationsfunktionalität erweitern lässt.

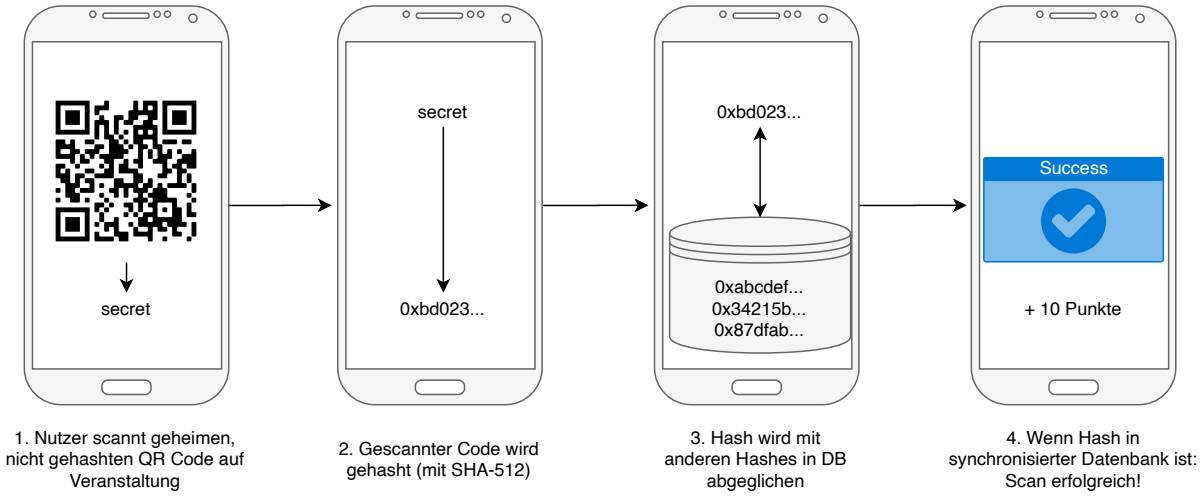

Abbildung 2.4 Die QR Codes auf der Veranstaltung werden zunächst gehasht, bevor sie mit dem Server-Backend abgeglichen werden. Somit würde ein Nutzer mit Zugriff auf die Datenbank zwar alle Hashes sehen, jedoch könnte er sich diese nicht (wie vorher) selbst generieren und damit im Spiel betrügen, da der Scan des Hashes mit der App einen weiteren Hash erzeugen würde, der mit kryptografisch verifizierbarer Sicherheit keinem Hash in der Datenbank entspräche. Somit ist diese Art, im OUTPUT.DD Spiel zu betrügen, nicht mehr möglich.

Verbesserung der Datensicherheit. Mit den von uns durchgeführten Maßnahmen wird nicht nur der Datenschutz von personenbezogenen Daten besser gewahrt, sondern auch die Datensicherheit verbessert. So war es in der alten Version der Datenbankstruktur beispielsweise möglich, die QR Codes aller Veranstaltungen auf OUTPUT.DD auszulesen und mit diesen die dahinter liegenden Errungenschaften im OUTPUT.DD Spiel freizuschalten. Diese Möglichkeit des Exploits wurde von uns behoben, indem die QR Codes vor dem Abgleich mit dem SHA512-Algorithmus gehasht werden. Nutzer haben somit keine Möglichkeit mehr, die QR Codes selbst zu erstellen und als Manipulation des Spiels zu scannen, da dies erfordern würde, dass sie den Hash zurück in den nur auf der Veranstaltung verfügbaren Code überführen müssten. Dies ist jedoch nicht möglich, da der SHA512-Hash als sogenannte „one-way-function“ kryptografisch sicher ist. Somit konnten wir auch die Datensicherheit der OUTPUT.DD App verbessern.

2.3 Umsetzung der Unit- und Integrationstests

Zur Validation und Verifikation des von uns erstellten Couchbase-Frameworks haben wir Unittests und Integrationstests umgesetzt. Während wir uns bei der Umsetzung der Unittests vorrangig auf das Testen einzelner Komponenten konzentrierten, konnten wir mithilfe der Integrationstests die korrekte Interaktion der Komponenten miteinander sicherstellen. Hierbei vereinfachte uns die gute Separation der Komponenten des Frameworks das gezielte Testen

einzelner Teilkomponenten, so dass wir insgesamt eine sehr gute Line Coverage[8] von 88,5 Prozent erreichen konnten.

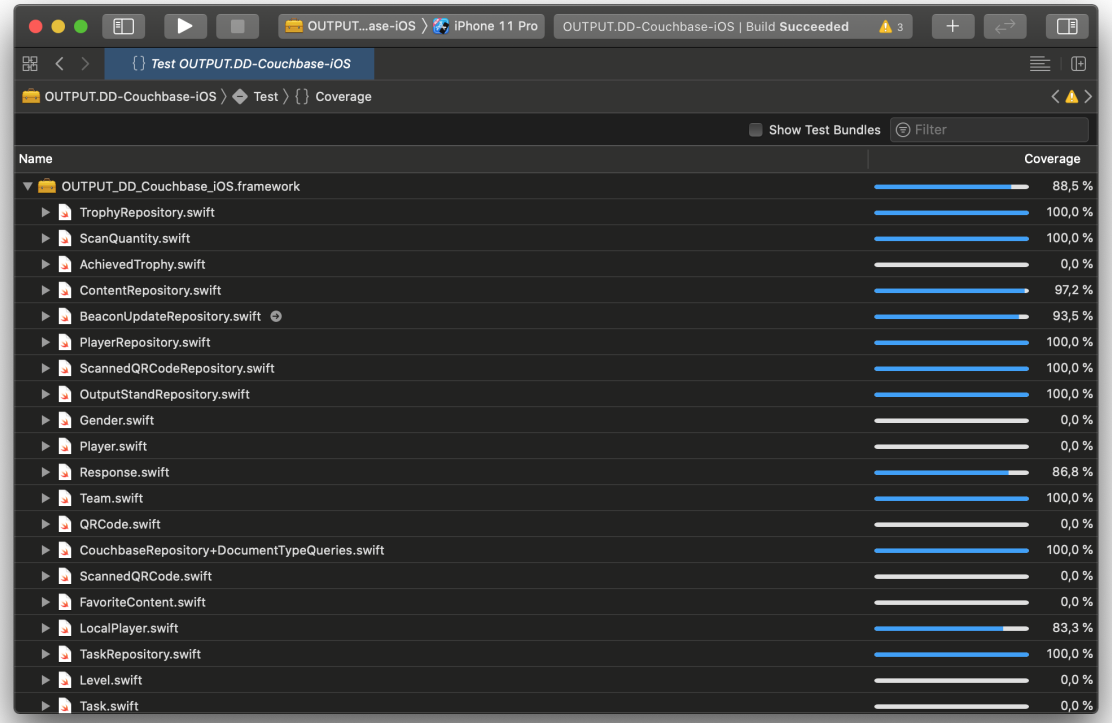

Abbildung 2.5 Eine Übersicht über die Test Coverage unseres iOS-Frameworks. In der jetzigen Form umfasst das iOS-Framework **3238** Codezeilen und **719** Dokumentationszeilen.

Trotz den verbleibenden 11,5 Prozent kann hierbei von einer fast vollständigen Coverage gesprochen werden, weil die verbleibenden Fehlerszenarien jeweils im Verantwortlichkeitsbereich der Tests von Couchbase selbst liegen, beispielsweise beim Ausfall einer Internetverbindung während des Synchronisationsvorgangs. Diese Fehlerszenarien werden vom Framework an geeigneten Stellen nach außen weitergegeben, um eine entsprechende Fehlerbehandlung im breiteren Kontext durch die OUTPUT.DD App zu ermöglichen (zum Beispiel Anzeige eines Alerts), und stellen somit zusammen mit leeren RepositoryObjects den verbleibenden Teil der nicht abgedeckten Line Coverage dar. Somit konnten wir erfolgreich das Ziel erreichen, eine angemessene Testsuite für unser entwickeltes Framework zu erstellen.

2.4 Erstellung und Integration eines containerisierten Docker-Test-Setups

Als Teil unserer Testsuite soll nun noch einmal genauer betrachtet werden, wie wir unsere Integrationstests umgesetzt haben. Um sicherzustellen, dass sich das Framework inklusive dem Couchbase-Replikator später korrekt mit dem Couchbase-Replikator-Service verbindet, mussten wir einen Weg finden, die Replikator-Funktionalitäten zu testen. Hierbei stellte sich jedoch als Problem heraus, dass Couchbase selbst kein integriertes Replikator-Test-Framework

[8] Code Coverage, bei der die getesteten Zeilen der Gesamtanzahl an Zeilen gegenübergestellt wird

anbietet, womit ein Couchbase-Replikator-Service imitiert werden könnte. Somit erstellten wir unseren eigenen Test-Service, welcher einen einfachen Couchbase-Replikator-Service[9] lokal starten und zu Testzwecken eingesetzt werden kann. Unsere Grundidee war hierbei, dass dieser Service lokal vor den entsprechenden Integrationstests gestartet wird und nach diesen wieder gestoppt wird. Um dies reproduzierbar zu machen, integrierten wir den Test-Service in einem eigenen Docker[10]-Image, um von diesem entsprechende Container zu erstellen, welche nach dem jeweiligen Testdurchlauf wieder gelöscht werden können. Durch die Containerisierung entstehen zwei primäre Vorteile. In erster Linie ist die Isolation und Entkopplung des Test-Services vom Host-Betriebssystem durch eine Virtualisierung des Adressraums vorteilhaft, sowie die Möglichkeit, Container dynamisch zu erzeugen und wieder zu eliminieren.

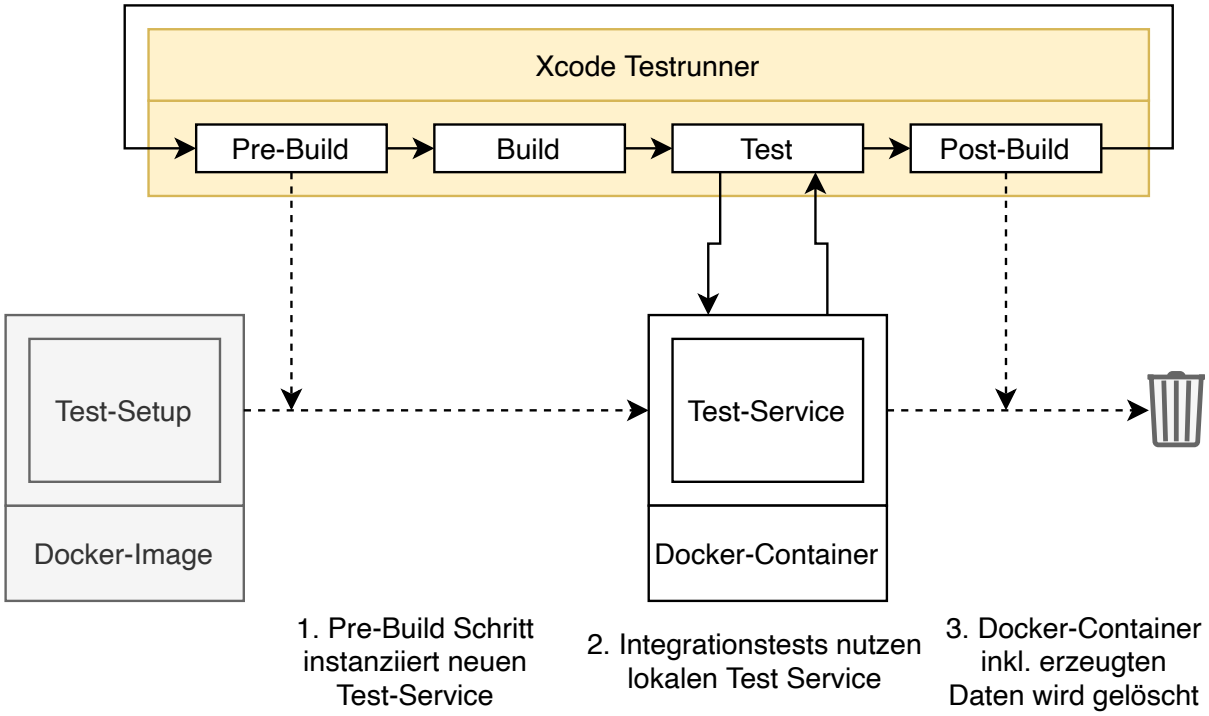

Abbildung 2.6 Um die Synchronisationsfunktionalität zu testen, erstellten wir einen Test-Workflow, bei dem ein Docker-Container zur Instanziierung eines lokalen Test-Services genutzt wird.

Durch Letzteres wurde es für uns möglich, das Docker-Image vor einem Testdurchlauf durch „pre-build“ Schritte in Xcode[11] zu instanziieren und den darin integrierten Service über einen fest definierten lokalen Port zur Verfügung zu stellen, um diesen während des Tests an den Replikator zu binden und schließlich nach dem Test inklusive aller darin persistierten Daten in einem „post-build“ Schritt wieder zu vernichten. Durch das Löschen der persistierten Daten sind aufeinanderfolgende Testläufe nicht voneinander abhängig, also reproduzierbar. Schlussendlich konnten wir so die Synchronisationsfunktionalität unseres Frameworks testen und erfolgreich sicherstellen, dass diese korrekt funktioniert.

[9] OUTPUT.DD-Couchbase-Docker-Test-Setup. `https://bitbucket.org/mapbiquitous/output.dd-couchbase-docker-test-setup` (Abgerufen am 21.10.2020)
[10] Docker. `https://www.docker.com/` (Abgerufen am 21.10.2020)
[11] Xcode. `https://developer.apple.com/xcode/` (Abgerufen am 21.10.2020)

2.5 Export über Cocoapods

Um das nun extensiv getestete Framework später in der OUTPUT.DD App anstelle von Realm zu integrieren, mussten wir jeweils (spezifisch für unsere Plattform) noch an einer Lösung für den Export des Frameworks arbeiten. In meinem Fall (iOS Plattform) hatte ich die Auswahl zwischen drei Paketmanagern:

- **Swift Package Manager** ist das offizielle, von Apple in Xcode integrierte Framework zur Distribution und Integration von Paketen, wird allerdings aktuell noch nicht von ausreichend vielen Paketen unterstützt (erfordert entsprechende Manifestation).
- **Carthage** ist ein weiterer Paketmanager, der jedoch auch nicht von den meisten Bibliotheken unterstützt wird und mit dem ich persönlich noch keine Erfahrung gemacht habe.
- **Cocoapods** wird durch fast alle Frameworks unterstützt und wird bereits in der OUTPUT.DD App verwendet.

Um zu einem späteren Zeitpunkt nicht mehrere Paketmanager gleichzeitig nutzen zu müssen, selektierte ich entsprechend zu den obigen Beschreibungen den Cocoapods Paketmanager.

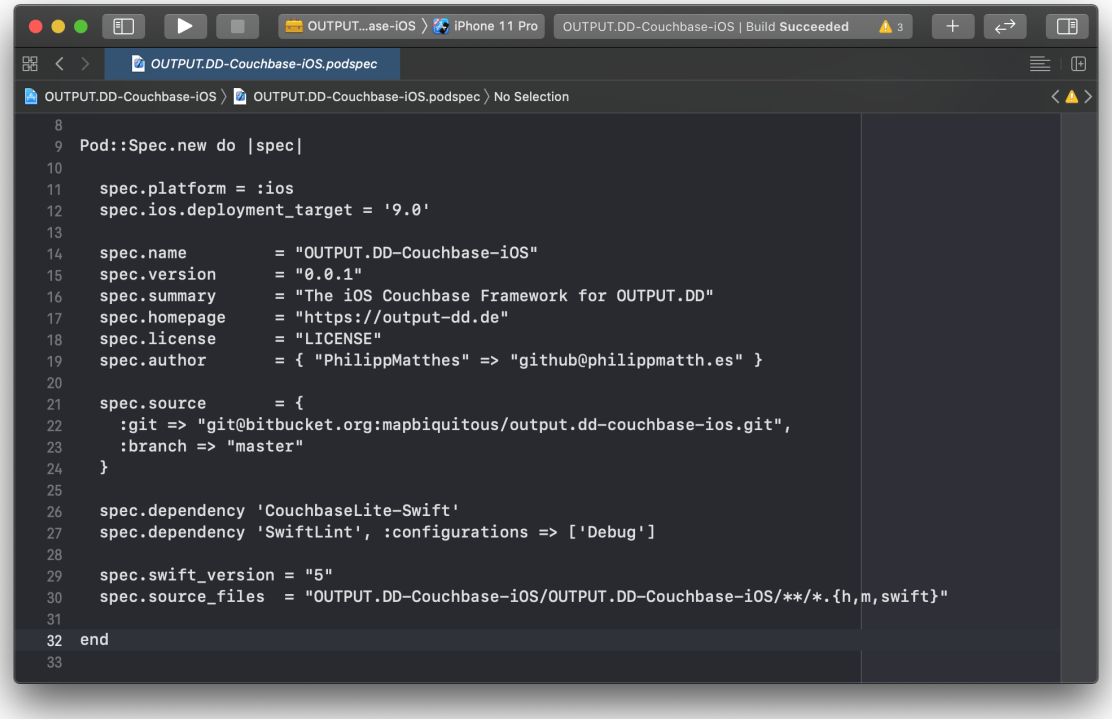

Abbildung 2.7 Die Cocoapods-Spezifikation des Frameworks.

Hierfür legte ich eine Spezifikation an, in der die notwendigen Abhängigkeiten (Couchbase, SwiftLint als Linter[12]) sowie zusätzliche Beschreibungen vermerkt wurden. Das Framework wird hierbei als „privates“ Cocoapods Framework behandelt, also ist nur zugänglich mit einem entsprechenden SSH-Schlüssel, welcher auf das Repository Zugriff hat. Hierdurch wird praktischerweise auch die Authentizität des Zugriffs auf das Framework automatisch gewährleistet. Genauso könnte das Framework jedoch auch öffentlich gemacht werden, da es keine sicherheitsrelevanten Schlüssel enthält. Die Einbindung des Frameworks in der App kann nun

[12] Linter: Programm, welches auf potenzielle Verstöße gegen Codierungsrichtlinien hinweist.

über die Spezifikation eines „Podfiles“ durchgeführt werden. Hierbei wird das Git Repository (gehostet auf Bitbucket[13]) als externer Zugangspunkt genutzt. Ein weiterer Vorteil ist, dass der Referenzpunkt des Pakets auf den `master` Branch des Git Repository gesetzt wurde und somit Aktualisierungen und Änderungen mit einer hohen Flexibilität umgesetzt werden können. Hierbei genügt das Ausführen von `pod update` im Verzeichnis der OUTPUT.DD App und das Paket wird automatisch aktualisiert. Somit konnten wir das Framework nicht nur strukturiert implementieren und testen, sondern stellen auch eine sehr einfache und flexible Möglichkeit der Einbindung zur Verfügung.

2.6 Zusammenfassung

Durch die Erstellung eines Couchbase-Repository-Frameworks konnten wir dem absoluten Ziel unserer Arbeit, dem Ersetzen von Realm durch Couchbase, wesentlich näher kommen. Wir konnten die Datenbankfunktionalitäten erfolgreich kapseln und über standardisierte Schnittstellen zur Verfügung stellen, gleichzeitig den Datenschutz und die Datensicherheit des Datenbank-Backends verbessern und schließlich das erstellte Framework über Cocoapods distribuieren.

[13] Bitbucket. `https://bitbucket.org` (Abgerufen am 21.10.2020)

3 Nutzbarkeit und Mehrwert für die OUTPUT.DD App

3.1 Probleme in der Codequalität der OUTPUT.DD App

Anschließend an die Erstellung des Couchbase-Frameworks hätten wir bereits mit der eigentlichen Migration beginnen können, wobei wir die alten Realm-Schnittstellen sukzessiv durch die neuen Schnittstellen ersetzt hätten. Gleichzeitig konnten wir jedoch bereits zu Beginn des Projektes feststellen, dass der Code der OUTPUT.DD App in sehr schlechtem Zustand ist, sowohl im iOS-Projekt, als auch im Android-Projekt. Als Symptome dafür, dass die App über mehrere Jahr hinweg von verschiedenen Entwicklern mit unterschiedlichen Qualitätsansprüchen und Kenntnisständen weiterentwickelt wurde fanden wir unter anderem so gut wie keine Dokumentation, außerdem auskommentierten und vergessenen Code, hartcodierte Datenpakete und weitere Fehlstrukturierungen. Weiteres Qualitätspotenzial wurde verschenkt, indem die Compiler-Warnungen für externe Bibliotheken deaktiviert und kein Linting-Framework eingebettet wurde.

Issues (1–10 of 10)

Title	T	P	Status	Votes	Assignee	Created	Updated	
#17: ScannedQRCode attribute "identifier" is redundant and has erroneous setter			NEW			2020-04-05	2020-04-05	
#16: Security Hazard: Dependency warnings are explicitly ignored in Podfile			NEW			2020-04-05	2020-04-05	
#10: Add default protocol implementations for RealmData.update if this method is optional			NEW			2020-04-05	2020-04-05	
#15: File TaskRepository.swift contains empty extension			NEW			2020-04-05	2020-04-05	
#14: ScheduleListViewController contains dead commented code			NEW			2020-04-05	2020-04-05	
#13: ScheduleListViewController repeatedly calls getContentDatesFor instead of caching them			NEW			2020-04-05	2020-04-05	
#12: Player attribute "identifier" is not used and erroneous			NEW			2020-04-05	2020-04-05	
#11: Remove redundant ";" symbols in Heatmap Extension			NEW			2020-04-05	2020-04-05	
#9: PlayerViewController.setTrophyIsSelected contains unused variable gameRepository			NEW			2020-04-05	2020-04-05	
#8: Build Phase Compile Sources contains duplicated file "OutputStand.swift"			NEW			2020-04-03	2020-04-03	

Abbildung 3.1 Von mir erstellte Issues zu ubiquitären Problemen in der OUTPUT.DD App.

Die gefundenen Fehler dokumentierte ich weitestgehend in eigenen Issues. Darüberhinaus konnte festgestellt werden, dass noch große Teile des OUTPUT.DD App Codes aus der älteren Objective-C Programmiersprache bestanden. Dies stellt ein Problem für die Wartbarkeit dar, da jede Objective-C Funktionalität noch einmal explizit (über einen sogenannten „Bridging-Header“)

für die neueren Swift-Teile und vice versa zur Verfügung gestellt werden müssen. Daher ist es außerdem wünschenswert, den nicht von der App klar entkoppelten Objective-C Code in aktuellen Swift Code zu übersetzen.

3.2 Beschluss des Neuaufbaus der OUTPUT.DD App

Die oben erläuterten Probleme summieren sich in Form von sehr hohen technischen Schulden[1] auf. Eine direkte Integration des Couchbase-Frameworks wäre in dieser Form zwar möglich gewesen, jedoch würde dies die vorhandenen Antipatterns[2] nicht beseitigen. Unter anderem könnte die oft fehlende Trennung von View[3]- und Model[4]-Code nicht verbessert werden, ohne eine Shotgun Surgery[5] durchzuführen.

Vor diesem Hintergrund bat sich uns eine günstige Situation, als durch SARS-CoV-2 OUTPUT.DD 2020 leider ausfallen musste und wir unseren Projektzeitplan neu evaluieren konnten. Ursprünglich war die Distribution der App für den Zeitraum November 2020 angesetzt, wodurch wir ursprünglich nur die Migration des Datenbank-Backends eingeplant hatten. Nun jedoch, nach erneuter Betrachtung der Situation, haben wir uns gemeinsam in Kooperation mit Dr. Thomas Springer dafür entschieden, diese Opportunität zu nutzen und die OUTPUT.DD App auf Grundlage des State of the Arts und mit hohen Qualitätsansprüchen neu aufzubauen. Konkret bedeutet dies für das konzipierte Framework, dass dieses nun während des Neuaufbaus mit integriert wird und somit seinen vorbestimmten Zweck auch erfüllen kann. Unser Ziel ist es nun, für OUTPUT.DD 2021 eine vollständig neu aufgebaute und modernisierte App zu erstellen, welche gleichzeitig auch einen höheren Qualitätsanspruch stellt.

[1] Technische Schulden: Zeit, die investiert werden müsste, um Komponenten mit schlechter Codequalität zu refaktorisieren.
[2] Antipattern: eine schlechte, kontraproduktive Programmierpraktik.
[3] View: Ansicht mit UI-Komponenten.
[4] Model: Datenobjekt.
[5] Shotgun Surgery: Antipattern, wenn eine Änderung zu weiteren notwendigen Änderungen führt usw. - durch die transitiven Beziehungen nicht voneinander isolierter Codeabschnitte.

4 Zusammenfassung und Ausblick

Im Rahmen unserer Kernaufgabe, die Realm-Datenbank der OUTPUT.DD App durch ein neues Couchbase-Backend zu ersetzen, konzipierten wir eine neue Datenbankstruktur, welche gleichzeitig bestimmte Datenschutz- und Datensicherheitsprobleme mitigiert. Dies haben wir erreicht, indem wir das Datenbank-Backend in zwei Abschnitte teilen: `Personal` und `Shared`. Somit sind bestimmte personenbezogene Daten nicht mehr für alle Nutzer global einsehbar. Außerdem haben wir ein Verfahren implementiert, um die Ausnutzung von QR-Codes im öffentlichen Kompartiment der Datenbank durch kryptographische Verschlüsselung zu verhindern. Für die Abstraktion, Aufbereitung und Bereitstellung der Datenbankfunktionalitäten implementierten wir ein generisches Framework auf Grundlage des Repository-Patterns, welches in seiner jetzigen Form eine effektiv vollständige Testabdeckung durch Unit- und Integrationstests besitzt und rund 3 tsd. Codezeilen umfasst. Zur reproduzierbaren Umsetzung der Integrationstests für die Synchronisationsfunktionalität des Frameworks erstellten und integrierten wir ein containerisiertes Test-Setup in unseren konzipierten Test-Workflow. Im Rahmen der Projektkommunikation legten wir unterdessen bestimmte Protokolle fest, die bestimmte Prozesse (wie die Erstellung eines Nutzers) in einer zentralen Kommunikationsgrundlage vereinheitlichen und dokumentieren. Mithilfe dieser Protokolle sowie dem Export des Frameworks über den Paketmanager Cocoapods wäre eine direkte technische Umsetzung der Migration auf das neue Couchbase Datenbank-Backend möglich gewesen, jedoch haben wir uns vor dem Hintergrund der ausfallenden OUTPUT.DD 2020 und des technischen Zustands der App für einen kompletten Neuaufbau entschieden, mit dem wir im Zeitraum des Wintersemesters 2020/21 bereits begonnen haben. In diesem Rahmen werden wir unser Framework schließlich integrieren, um unsere ursprüngliche Kernaufgabe vollständig abzuschließen und gleichzeitig eine nachhaltige und qualitativ hochwertige App-Grundlage für die weiteren Projektteams zu schaffen.

Persönliches Nachwort. Wie auch bereits im Wintersemester 2019/20 hat es mir sehr viel Spaß gemacht, an OUTPUT.DD mitzuwirken und zu wachsen. Besonders habe ich mich dieses Semester darüber gefreut, dass mich ich zusammen mit Felix Kästner im Rahmen der OUTPUT.DD App in einem meiner zentralen Interessensbereiche (App-Entwicklung) verwirklichen und weiterentwickeln konnte. Zusammen mit dem gesamten App-Team hatten wir stets sehr konstruktive Diskussionen auf hohem technischem Niveau, mit einem sichtbar exzellentem Endprodukt aus neuen Workflow-Konzepten und einem neuen Datenbank-Framework. Auch wenn für den restlichen Teil des OUTPUT.DD Teams unsere Arbeit meist eher unsichtbar und im Hintergrund verlief, denke ich, dass wir schließlich im Rahmen der Neuimplementation der App einen wesentlichen Beitrag mit Nachwirkung auch auf zukünftige Projektteams leisten konnten.