Abschlussprüfung Winter 2025/2026 Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung eine „Interaktive UML-Lern-App mit Quiz, Notizen, Lernpfaden (Levels) und Kursverwaltung“

Eingereicht von:

Josiane Kanouo Maneyo
Umschülerin zur Fachinformatikerin für Anwendungsentwicklung

IAD GmbH Marburg

Neue Kasseler Straße 62e

35039 Marburg

Betreuer: Marcus Brauer

3. Oktober 2025

Abgabedatum: 28.11.2015

Marburg, 2025

Inhaltsverzeichnis

Litetaturverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

Abkürzungsliste

Abkürzung	**Bedeutung**
API	Application Programming Interface (Programmierschnittstelle)
App	Application (Anwendung, Programm für Mobilgeräte)
CRUD	Create, Read, Update, Delete (Datenbank-Operationen)
DB	Datenbank
DSGVO	Datenschutz-Grundverordnung
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
IDE	Integrated Development environnment
IAD	Informationsverarbeitung und angewandte Datentechnik
JSON	JavaScript Object Notation
MVVM	Model-View-ViewModel
SQL	Structured Query Language
UI	User Interface
UID	Unique Modeling Language
ULM	Unified Modeling Language
UX	User Experience
SDK	Software Development Kit

1 Einleitung

Dieses Kapitel beschreibt die Ausgangssituation, die Motivation und die Zielsetzung des Projekts. Es erläutert, welches Problem durch die Anwendung gelöst werden soll, welchen Nutzen sie bietet und wie das Projekt in den Gesamtkontext der Umschulung eingebettet ist. Darüber hinaus werden die Projektumgebung, die Beteiligten und die Rahmenbedingungen vorgestellt, um einen Überblick über den Hintergrund des Projekts zu geben.

1.1 Projektumfeld

Das Projekt wird im Rahmen der Umschulung zur Fachinformatikerin für Anwendungsentwicklung an der IAD- Bildungszentrum Marburg GmbH durchgeführt. Die IAD zählt zu den größten privaten Bildungsträgern Deutschlands im Bereich IT und Management. Am Standort Marburg werden praxisnahe Umschulungen und Weiterbildungen im IT-Bereich angeboten. Im Ausbildungsgang Fachinformatiker/-in für Anwendungsentwicklung liegt der Schwerpunkt auf der Softwareentwicklung mit Java und C#, ergänzt durch Themen wie Datenbanken, Projektmanagement und Softwaredesign. Die Projektarbeit erfolgt als Einzelprojekt. Die Verantwortung umfasst Analyse, Entwurf, Implementierung, Test und Dokumentation. Fachliche Betreuung erfolgt durch Marcus Brauer, Dozent an der IAD, der die technische Umsetzung unterstützt und die Einhaltung der IHK-Vorgaben überwacht. Zielgruppe der Anwendung sind Lernende und Lehrkräfte, die eine digitale Plattform zur Kursauswahl, Bearbeitung von Quizfragen, Verwaltung von Fehlerlisten und Erstellung persönlicher Notizen nutzen können. Die Anwendung adressiert den Übergang von statischen Papierübungen zu interaktiven Tools, die ortsunabhängiges Training mit direktem Feedback und zentraler Auswertung ermöglichen. Neben der bestehenden Lernumgebung wird mit der App eine moderne, mobile Erweiterung geschaffen, die neuen Lernenden den Einstieg durch ein interaktives Tutorial erleichtert und Rückmeldungen direkt in die Weiterentwicklung einfließen lässt.

1.2 Projektziel

Ziel des Projekts ist die Entwicklung einer interaktiven Android-Anwendung, die Lernenden und Lehrkräften des IAD Bildungszentrums ein Werkzeug zum Üben, Wiederholen und Verwalten von Lerninhalten bietet. Die Anwendung ersetzt traditionelle, statische Methoden und schafft eine strukturierte, motivierende Lernumgebung für UML-Diagramme Die App soll den Lernprozess digitalisieren und Lernenden ermöglichen, selbstständig, interaktiv und

ortsunabhängig zu üben.Durch automatisierte Auswertung, Feedback und Fortschrittskontrolle wird das Lernen effizienter gestaltet und Lehrkräfte werden administrativ entlastet.

1.3 Projektbegründung

Der Unterricht an der IAD basiert derzeit überwiegend auf Papierübungen, PDF und Word-Dokumenten für UML-Diagramme. Diese Methode ist unflexibel, bietet kein direktes Feedback und erlaubt keine zentrale Verwaltung oder Auswertung. Lehrkräfte haben dadurch keinen einfachen Überblick über den Lernstand, und Ergebnisse lassen sich nicht systematisch dokumentieren oder vergleichen. Die geplante Anwendung digitalisiert diese Prozesse, erhöht Transparenz und steigert die Motivation. Lernende können gezielt Fehler wiederholen, ihren Fortschritt verfolgen und individuelle Notizen anlegen.

Eine Eigenentwicklung wurde gewählt, da bestehende Lernplattformen wie Moodle oder Quizlet keine Kombination aus Fehlerwiederholung, Notizen und Levelsystem bieten. Die Lösung ist somit speziell auf die Bedürfnisse der Umschulungsteilnehmenden zugeschnitten und unterstützt die Digitalisierung des Unterrichts, wodurch der Lernprozess effizienter, praxisnäher und motivierender gestaltet wird als mit herkömmlichen Papierübungen. Durch das Onboarding-Tutorial wird der Einstieg in die Anwendung auch für technisch weniger erfahrene Lernende deutlich einfacher.

1.4 Projektschnittstellen

Die Anwendung kommuniziert mit Firebase-Diensten (Authentication, Cloud Firestore, Storage, Security Rules) über sichere HTTPS-Verbindungen. Offline-Nutzung wird durch lokale Datenspeicherung ermöglicht. Beteiligte Personen sind Marcus Brauer als Auftraggeber, Josiane Kanouo Maneyo als Entwicklerin und Testpersonen aus der IAD. Externe Systeme oder APIs werden nicht integriert. Zusätzlich wird die Feedback-Sammlung in Firebase zur Speicherung eingehender Rückmeldungen genutzt.

1.5 Projektabgrenzung

Das Projekt beschränkt sich auf die Entwicklung der Android-Anwendung mit den im Antrag festgelegten Kernfunktionen. Ausgeschlossen sind eine Webversion, KI-Funktionen, komplexe Statistiken oder die Veröffentlichung im Google Play Store. Das Projekt endet mit einem funktionsfähigen Prototyp, der die geforderten Anforderungen erfüllt. Es wird keine UML-Diagramm-Erstellung integriert; der Fokus liegt auf quizbasiertem Lernen und Verwaltung.

2 Projektplanung

Im folgenden Kapitel wird der zeitliche und organisatorische Ablauf des Projekts dargestellt. Hierzu werden die einzelnen Projektphasen beschrieben, der Zeitaufwand dokumentiert und die eingesetzten Methoden und Werkzeuge erläutert.

2.1 Projektphasen

Das Projekt umfasst einen Zeitrahmen von 80 Stunden und gliedert sich in acht aufeinanderfolgende Phasen: Analyse, Konzept, Entwurf, Implementierung, Test und Dokumentation. Die Analysephase diente der Anforderungsermittlung, gefolgt von Entwurf und Programmierung sowie Tests zur Sicherstellung der Funktionalität. Die Soll-Phase und Implementierung umfassen zusätzlich die Entwicklung eines Onboarding-Tutorials sowie einer In-App-Feedback-Funktion, um die Benutzerfreundlichkeit zu erhöhen. Eine detaillierte Übersicht der Stundenverteilung ist in der *Tabelle 1* im Anhang A1 enthalten.

2.2 Abweichungen vom Antrag

Gegenüber dem ursprünglichen Projektantrag wurde der Funktionsumfang gezielt erweitert. Ergänzt wurden ein Onboarding-Tutorial und eine In-App-Feedback-Funktion, um die Benutzerfreundlichkeit und die Interaktivität der App zu verbessern. Das Onboarding erleichtert neuen Nutzenden den Einstieg in die Anwendung, während die Feedback-Funktion eine direkte Rückmeldung ermöglicht und so zur kontinuierlichen Verbesserung beiträgt. Der Gesamtzeitrahmen von 80 Stunden blieb unverändert; der Mehraufwand wurde durch optimierte Arbeitsschritte und eine angepasste Zeitverteilung ausgeglichen.

2.3 Ressourcenplanung

Zur Umsetzung stehen ein Windows-Laptop, ein Android-Smartphone sowie die Software Android Studio, Kotlin und Firebase zur Verfügung. Versionierung und Datensicherung erfolgen über GitHub. Zusätzliche Tools wie Draw.io, Microsoft Word und PlanUML unterstützen die Erstellung von Diagrammen und der Dokumentation.

An Personalressourcen wurden acht Stunden für einen fachlichen Betreuer einkalkuliert, der bei fachlichen Themen unterstützen konnte und Teile der Analyse- sowie Planungsphase begleitete.

2.4 Vorgehensmodell

Für die Umsetzung wurde ein phasenorientiertes Vorgehensmodell auf Basis des V-Modells angewendet. Die Struktur gliederte sich in klar definierte Phasen von der Anforderungsanalyse über Entwurf, Implementierung und Test bis hin zur Abnahme. Das V-Modell bot durch die enge Verknüpfung von Entwicklungs- und Testphasen eine systematische Vorgehensweise und erhöhte Nachvollziehbarkeit. Ergänzend wurden kurze Feedbackzyklen nach jeder umgesetzten Funktion (z. B. Login, Quiz, Feedback, Tutorial) integriert, um frühzeitig Qualitätssicherung und Benutzerfreundlichkeit zu gewährleisten. So konnte trotz fester Phasenstruktur flexibel auf Designänderungen und Testergebnisse reagiert werden.

3 Analysephase

In der Analysephase werden die Anforderungen an die zu entwickelnde Anwendung systematisch erfasst und präzisiert. Dabei werden sowohl funktionale als auch nicht-funktionale Anforderungen beschrieben, um sicherzustellen, dass alle relevanten Aspekte berücksichtigt werden. Zusätzlich wird das bestehende Problem analysiert, und es werden die Ziele sowie der Nutzen der geplanten Lösung klar abgegrenzt. Diese Phase bildet die inhaltliche Grundlage für den späteren Entwurf und die technische Umsetzung.

3.1 Ist-Analyse

Der Unterricht an der IAD-Bildungszentrum Marburg GmbH basiert derzeit auf klassischen Lehrmethoden, bei denen die Lernenden UML-Diagramme in Papier- oder PDF-Form üben. Eine unmittelbare Rückmeldung zu den Ergebnissen erfolgt nicht; Korrekturen werden von Lehrkräften manuell durchgeführt und können nicht zentral dokumentiert werden. Auch der individuelle Lernfortschritt ist nicht transparent einsehbar. Vorhandene Tools wie draw.io oder PlantUML ermöglichen zwar das Zeichnen von Diagrammen, unterstützen jedoch keinen interaktiven Lernprozess mit Feedback-, Notiz- oder Fortschrittsfunktionen. Das Fehlen einer digitalen, benutzerfreundlichen Lösung führt zu hohem Zeitaufwand und geringerer Motivation bei den Lernenden. Eine moderne, mobile App-Lösung kann diese Probleme beseitigen, indem sie Lernfortschritte automatisch erfasst, Fehler dokumentiert und Lerninhalte flexibel zur Verfügung stellt.

3.2 Funktionale Anforderungen

Ziel des Projekts ist die Entwicklung einer interaktiven Android-App, die den Lernprozess für UML Diagramme digitalisiert und Lernende beim eigenständigen Üben unterstützt. Die Anwendung bietet einen rollenbasierten Zugang über Firebase Authentication, bei dem zwischen *User* (Lernende) und *Admin* (Lehrkräfte) unterschieden wird. Lernende können sich

registrieren, anmelden, Kurse auswählen, Quizze durchführen, Notizen anlegen und Feedback abgeben. Administratoren verwalten Kurse, Lerneinheiten und Fragen direkt in der App. Alle Benutzerdaten und Lernfortschritte werden in Firebase Firestore gespeichert und beim Login automatisch der jeweiligen Rolle zugeordnet. Die App umfasst folgende Hauptfunktionen: Ein Kurs- und Lerneinheitensystem strukturiert die Inhalte in Themenbereiche (*Courses* und *Units*), die jeweils Quizfragen mit Antwortoptionen, Lösung und Punktewert enthalten. Das Quizmodul lädt die Fragen dynamisch aus Firestore, zeigt sie nacheinander an und berechnet nach Abschluss Punktestand, Erfolgsquote und Level. Eine Fehlerliste speichert falsch beantwortete Fragen für gezieltes Wiederholen, während ein integrierter Timer die Bearbeitungszeit begrenzt. Über die Notizenfunktion können Lernende eigene Gedanken oder Erklärungen speichern, die sowohl offline verfügbar als auch automatisch synchronisiert werden. Das Level- und Punktesystem motiviert durch Fortschrittsanzeigen und Levelaufstiege, die im Profil dargestellt werden. Zur Unterstützung des Lernprozesses enthält die App ein kurzes Onboarding-Tutorial, das neue Nutzer durch die wichtigsten Bereiche führt, sowie eine Feedback-Funktion, über die Anregungen oder Fehlermeldungen direkt an die Entwicklerin gesendet werden können. Ein separater Administrationsbereich erlaubt Lehrkräften das Erstellen und Bearbeiten von Kursen und Fragen in Echtzeit. Die gesamte Kommunikation erfolgt verschlüsselt, und die Firestore Regeln sichern die Daten gegen unbefugten Zugriff. Dank Offline-Caching bleibt die Anwendung auch ohne Internetverbindung voll funktionsfähig. Damit erfüllt die UML-Lern App sämtliche im Projektantrag definierten Anforderungen. Eine detaillierte tabellarische Funktionsübersicht mit technischen Zuordnungen ist in der *Tabelle 2* im Anhang A2 enthalten.

3.3 Wirtschaftlichkeitsbetrachtung

Die Wirtschaftlichkeit des Projekts wurde anhand der Entwicklungskosten und des erwarteten Nutzens für die IAD untersucht. Da die Entwicklerin als Umschülerin kein Gehalt erhält, entfallen Personalkosten. Stattdessen wurde der Zeitaufwand der betreuenden Fachkraft als kalkulatorischer Aufwand berücksichtigt. Ziel der Betrachtung war es, den Nutzen für die IAD darzustellen, insbesondere die Zeitersparnis für Dozenten und die Reduktion des Verwaltungsaufwands durch die App.

3.3.1 Make-or-Buy-Entscheidung

Zu Beginn wurde geprüft, ob eine bestehende Lernplattform wie Moodle, OpenOLAT oder Udemy Business eingesetzt werden könnte. Diese Systeme bieten zwar umfangreiche Funktionen, sind jedoch serverbasiert, kostenpflichtig und nicht auf Offline-Nutzung ausgelegt.

Zudem wäre eine Anpassung an die spezifischen Anforderungen der IAD insbesondere das rollenbasierte Rechtesystem (Lehrkraftr/Lernende) und die Integration von Firebase mit zusätzlichem Aufwand verbunden.

Die Eigenentwicklung der UML-Lern-App in Kotlin mit Firebase Firestore bot dagegen folgende Vorteile: Keine Lizenz- oder WartungskostenL; Offline-Fähigkeit; volle Kontrolle über Datenspeicherung (DSGVO-konform, Firebase Cloud in EU-Region); Erweiterbarkeit für künftige; Module oder Kurse. Auf Basis dieser Argumente fiel die Entscheidung auf die Eigenentwicklung („Make“).

3.3.2 Projektkosten

Da keine Lohnkosten für die Entwicklerin anfallen, beschränken sich die Projektkosten auf den Betreuungsaufwand, Arbeitsmittel und Infrastruktur. Die Kalkulation basiert auf einem Ansatz von 8 Stunden für den Betreuer (technische Begleitung, Feedback, Abnahme) sowie den pauschalen Nutzungskosten der Hardware und Firebase-Umgebung.

Kostenart	Anzahl/ Zeit	Satz	Kosten (€)	Bemerkung
Betreueraufwand (IAD)	8Stunden	40 €/Std.	320 €	Fachliche Betreuung, Abnahme
Arbeitsplatz & Internet	80 Stunden	5 € / Std.	400 €	Strom, PC, Verbindung
Firebase-Ressourcen	3 Monate	10 €/Monat	30€	Cloud-Nutzung (Speicher, Authentifizierung)
Gesamtkosten	-	-	≈750€	--

Tabelle 3 Projektkosten

Da der Firebase Spark Plan (kostenlose Stufe) genutzt wurde, fallen keine laufenden Kosten an.Eventuelle Erweiterungen (z. B. Analytics oder Cloud Functions) könnten in der Zukunft geringe Zusatzkosten verursachen.

3.3.3 Nutzen und Einsparungen

Die App reduziert den administrativen Aufwand erheblich. Vor der Einführung mussten Trainer Quizfragen ausdrucken, manuell korrigieren und Ergebnisse übertragen. Pro Kurs mit 20 Lernenden dauerte dies durchschnittlich 10 Minuten pro Lernenden und Woche. Durch die

automatische Auswertung in der App entfallen diese Aufgaben fast vollständig. Bei 20 Lernenden über 40 Unterrichtswochen ergibt sich:

20 Lernende × 10 Minuten × 40 Wochen = 8 000 Minuten = 133 Stunden pro Jahr

Bei einem Dozentenstundensatz von 30 € entspricht das einer jährlichen Einsparung von:

133 Stunden × 30 € = ≈ 4 000 €

Weitere qualitative Vorteile:

Trainer können die gewonnene Zeit für inhaltliche Betreuung oder neue Kurse nutzen. Lernende erhalten sofortiges Feedback, was den Lernerfolg verbessert. Papierverbrauch und Druckkosten sinken. Ergebnisse sind digital archiviert und jederzeit auswertbar. Das Image der IAD als moderne Bildungseinrichtung wird gestärkt.

3.3.4 Amortisation und Fazit

Die Gesamtkosten der Entwicklung belaufen sich auf rund 750 €.

Bei einer jährlichen Zeitersparnis von etwa 4 000 € amortisiert sich die App bereits nach:

750 € / 4 000 € ≈ 0,19 Jahre = ca. 2,3 Monate

Damit ist das Projekt nicht nur wirtschaftlich sinnvoll, sondern auch langfristig profitabel. Die Anwendung spart Dozenten dauerhaft Zeit, senkt interne Aufwände und verbessert die Unterrichtsqualität. Auch wenn keine direkten Einnahmen entstehen, führt der indirekte Nutzen zu einer messbaren Effizienzsteigerung für die IAD. Das Projekt trägt somit unmittelbar zur wirtschaftlichen und organisatorischen Optimierung der Schulungsprozesse bei.

3.4 Qualitätsanforderungen

Neben den fachlichen Funktionen wurden auch qualitative Anforderungen definiert, die Leistung, Sicherheit und Benutzerfreundlichkeit der Anwendung sicherstellen. Die App soll stabil, intuitiv und plattformgerecht funktionieren und dabei den Datenschutzbestimmungen entsprechen. Die Benutzerfreundlichkeit steht im Vordergrund: Eine klare, übersichtliche Oberfläche nach den Richtlinien des Material Design sorgt für eine intuitive Bedienung. Alle Texte und Icons sind für mobile Geräte optimiert, und Animationen unterstützen den Lernfluss, ohne abzulenken. Zur Leistung zählt, dass das Quiz, auch bei mehreren Datenabfragen, schnell reagiert und Ergebnisse in Echtzeit anzeigt. Durch lokale Zwischenspeicherung (Offline-Cache) bleibt die App auch bei instabiler Verbindung funktionsfähig, wodurch eine hohe Verfügbarkeit erreicht wird. Hinsichtlich Sicherheit werden alle Benutzerdaten ausschließlich über verschlüsselte HTTPS-Verbindungen übertragen. Die Authentifizierung erfolgt über Firebase, sodass keine sensiblen Passwörter in der App gespeichert werden. Über

Firestore Security Rules wird der Zugriff auf Daten strikt nach Benutzerrolle kontrolliert. Die App erfüllt zudem grundlegende Anforderungen an Datenschutz und Integrität, indem nur notwendige personenbezogene Daten (z. B. Benutzername, UID) gespeichert werden.Ein weiteres Ziel war die Wartbarkeit und Erweiterbarkeit. Durch den modularen Aufbau im MVVM-Architekturmuster (Model–View–ViewModel) können neue Funktionen wie zusätzliche Kurse, Auswertungen oder Lernstatistiken ohne größere Codeanpassungen ergänzt werden. Die Qualitäten sind im Pflichtenheft A3 dokumentiert.

3.5 Anwendungsfälle

Das System unterstützt zwei Akteure: Lernende und Administratoren. Lernende können nach dem Login Kurse auswählen, Fragen beantworten und Ergebnisse direkt einsehen. Fehler werden gespeichert und können wiederholt geübt werden. Administratoren verwenden die App, um Kurse und Fragen zu pflegen und Inhalte laufend zu aktualisieren. Eine grafische Darstellung der Anwendungsfälle befindet sich im Use case Diagramm siehe Anhang A4.

3.6 Verweis auf das Lastenheft

Alle funktionalen und nicht-funktionalen Anforderungen sind im Anhang A5 Lastenheft detailliert beschrieben. Das Lastenheft definiert Muss- und Kann-Kriterien sowie Abgrenzungen des Projekts und bildet die Grundlage für die Abnahme.

4 Entwurfsphase

Dieses Kapitel beschreibt die technische und gestalterische Planung der UML-Lern-App. Hier werden die grundlegende Architektur, die Struktur der Datenbank, das Design der Benutzeroberfläche sowie der Aufbau der App erläutert. Die Entwurfsphase bildet somit den Übergang zwischen der theoretischen Planung und der praktischen Umsetzung.

4.1 Zielplattform

Für die Umsetzung der UML-Lern-App wurde die Android-Plattform als Zielumgebung gewählt. Ausschlaggebend dafür waren die weite Verbreitung von Android-Geräten, die einfache Verfügbarkeit von Entwicklungswerkzeugen sowie die Möglichkeit, moderne Technologien wie Firebase direkt zu integrieren. Als Programmiersprache kommt Kotlin zum Einsatz, da sie von Google offiziell unterstützt wird, eine klare Syntax besitzt und sich besonders gut für mobile Anwendungen eignet. Die Datenhaltung erfolgt in der Cloud Firestore-Datenbank von Firebase, die eine dokumentenbasierte Struktur bietet und sowohl Online- als auch Offline-Betrieb ermöglicht. Diese Architektur erlaubt eine flexible, sichere und leicht erweiterbare

Speicherung aller Benutzerdaten, Kurse und Quiz-Ergebnisse. Die App folgt dem Client-Server-Prinzip: Der Client (die Android-App) kommuniziert mit dem Server (Firebase-Dienste) über eine gesicherte HTTPS-Verbindung. Dadurch können Benutzer weltweit auf ihre persönlichen Daten zugreifen, ohne dass lokale Server notwendig sind. Die Anwendung wurde auf Emulatoren (Pixel 6, Android 14) und auf realen Geräten wie dem Samsung A56 getestet. Die Entwicklungsumgebung war über ein GitHub-Repository versioniert. Damit wird sichergestellt, dass die Anwendung flüssig läuft und gleichzeitig auch auf älteren Geräten nutzbar bleibt. Die Kombination aus Kotlin, Firebase und Android Studio bietet somit eine moderne, performante und zukunftssichere Grundlage für die Umsetzung der Lern-App. Ausschnitt aus der Datei *build.gradle* (Projektabhängigkeiten) ist im Anhang A6.

4.2 Architektur und Designprinzip

Die Architektur der UML-Lern-App basiert auf dem Model-View-ViewModel-Muster (MVVM), einem bewährten Architekturkonzept für Android-Anwendungen. Dieses Muster trennt die Datenhaltung, die Geschäftslogik und die Benutzeroberfläche klar voneinander. Dadurch bleibt die Anwendung modular aufgebaut, leicht wartbar und flexibel erweiterbar.

Abbildung 2 MVVM- Prinzip

Das Model repräsentiert die Datenebene. Hier befinden sich alle Datenklassen, die direkt der Struktur der Firestore-Datenbank entsprechen beispielsweise *User*, *Admin*, *Course*, *Unit*, *Question*. Diese Klassen enthalten keine Logik, sondern dienen ausschließlich der Datenhaltung und -übertragung.

Das ViewModel bildet die Vermittlungsschicht zwischen Daten und Benutzeroberfläche. Es ruft Daten aus dem Repository ab, validiert Eingaben, verarbeitet sie und bereitet sie für die Anzeige in der View vor. Außerdem überwacht das ViewModel Änderungen und aktualisiert die Oberfläche automatisch über LiveData, sobald sich Daten in Firestore ändern. Die View umfasst die Benutzeroberfläche also Activities, Fragments und die zugehörigen XML-Layouts. Sie ist für die Darstellung und Interaktion zuständig, enthält jedoch keine Geschäftslogik. Über Data Binding kommuniziert sie direkt mit dem ViewModel, wodurch eine klare Trennung

zwischen Darstellung und Logik gewährleistet ist. Zwischen ViewModel und Datenebene vermittelt ein Repository, dass alle Firebase-Operationen (Create, Read, Update, Delete) kapselt. Das Repository stellt eine einheitliche Schnittstelle für den Datenzugriff bereit und sorgt dafür, dass Änderungen an der Datenquelle (z. B. Firestore) keine Anpassungen in der restlichen Anwendung erfordern. Dadurch bleibt der Code übersichtlich, testbar und langfristig erweiterbar.

Die UML-Lern-App gliedert sich in fünf zentrale Funktionsmodule:

1. Benutzermanagement Registrierung, Login und Rollenprüfung
2. Kurs- und Unitverwaltung Laden und Anzeigen der Lerneinheiten
3. Quizsystem dynamische Anzeige von Fragen, Bewertung und Ergebnisdarstellung
4. Adminbereich Verwaltung von Kursen und Fragen
5. Notizenfunktion (persönliche Lernunterstützung durch eigene Einträge)

Das grafische Design orientiert sich am Material Design von Google. Ziel war eine übersichtliche, barrierearme und responsive Oberfläche mit klarer Farbstruktur, moderner Typografie und konsistenten Interaktionselementen. Durch die Verwendung von ConstraintLayouts und Material Komponenten (z. B. Buttons, FloatingActionButtons und Cards) entstand ein modernes, benutzerfreundliches Erscheinungsbild, das auch auf unterschiedlichen Bildschirmgrößen stabil funktioniert.

4.3 Datenmodell und Firestore-Struktur

Die Datenspeicherung der UML-Lern-App erfolgt über Firebase Cloud Firestore, eine dokumentenorientierte NoSQL-Datenbank. Firestore organisiert Daten in Collections (Sammlungen) und Documents (Dokumenten) und ist somit besonders flexibel und leistungsfähig für mobile Anwendungen. Dieses Modell ermöglicht eine hierarchische, leicht erweiterbare Datenhaltung, die speziell für Anwendungen mit wechselnder Internetverbindung geeignet ist. Die App greift in Echtzeit auf Firestore zu, wobei alle Daten zentral gespeichert und automatisch synchronisiert werden

4.3.1 Zentrale Entitäten

Die Firestore-Datenbank bildet die Grundlage aller Anwendungsfunktionen. Im Mittelpunkt steht die Sammlung users, in der alle Benutzerprofile gespeichert werden. Jedes Profil enthält Informationen wie Benutzername, E-Mail-Adresse, Rolle (user oder admin), Punktestand und

Lernfortschritt. Die Datenspeicherung erfolgt in einer klar strukturierten Firestore-Organisation. Die Sammlung courses enthält alle Lernkurse mit Titel, Beschreibung und Schwierigkeitsgrad. Innerhalb jedes Kurses sind die zugehörigen units (Lerneinheiten) angelegt, die wiederum mehrere questions (Fragen) enthalten. Diese hierarchische Struktur bildet den Lernpfad der App ab. Die Sammlung attempts dokumentiert die Quizversuche der Lernenden mit Punktzahl, Dauer und Abschlusszeit. Notes sind als persönliche Subcollection innerhalb des Benutzer Dokuments angelegt und ermöglichen das Speichern individueller Notizen. In der Sammlung feedback werden Rückmeldungen und Bewertungen der Nutzenden abgelegt, die von Administratoren ausgewertet werden können. Durch diese Organisation entsteht ein nachvollziehbares und konsistentes Datenmodell, das die zentralen Funktionen der App vollständig unterstützt. Alle Entitäten sind als Kotlin-Data-Classes implementiert, wodurch eine direkte Verbindung zwischen Datenbankstruktur und Quellcode besteht.

4.3.2 Beziehungen zwischen den Entitäten

Die Beziehungen zwischen den Datenelementen sind logisch und übersichtlich aufgebaut. Ein Benutzer kann mehrere Quizversuche, Notizen und Feedback-Einträge besitzen, wodurch eine 1-n-Beziehung zwischen der Entität User und ihren abhängigen Daten entsteht. Ein Kurs besteht aus mehreren Einheiten, und jede Einheit wiederum aus mehreren Fragen –diese Beziehungen sind als Komposition umgesetzt, da untergeordnete Elemente ohne ihren übergeordneten Kontext nicht existieren. Das vollständige Klassendiagramm mit allen Entitäten, Beziehungen und Kardinalitäten befindet sich im Anhang A7.

4.3.3 Sicherheit und Rollenmodell

Ein zentraler Bestandteil der Architektur ist das Sicherheits- und Rollenmodell, das den Zugriff auf Daten präzise steuert. Die UML-Lern-App nutzt Firebase Authentication zur Authentifizierung und Firestore Security Rules für den Zugriffschutz (siehe Rollen Regeln im Anhang A8 Zugriffsrollen) Jeder Benutzer wird eindeutig durch eine UID identifiziert und besitzt in der Sammlung Users ein Feld Rolle, das die jeweilige Rolle definiert. Es werden zwei Rollen unterschieden: User (Lernende) und Admin. User dürfen Quizze absolvieren, eigene Notizen erstellen und Feedback geben, haben jedoch keine administrativen Rechte. Admins dürfen zusätzlich Kurse und Fragen erstellen, bearbeiten und löschen. Beim Anmelden wird die Rolle automatisch aus Firestore geladen, damit die Oberfläche (z. B. Adminbereich) dynamisch angepasst werden kann. Dadurch ist gewährleistet, dass Lernende keine administrativen Bereiche betreten können.

```
    // USERS
    match /users/{userId} {
      allow read: if true; // für Username/Email-Lookup
      allow create: if signedIn() && userId == me();
      allow update, delete: if signedIn() && (userId == me() || isAdmin());
    }

    // COURSES (+ nested)
    match /courses/{courseId} {
      allow read: if signedIn();
      allow create, update, delete: if isAdmin();

      match /questions/{qId} {
        allow read: if signedIn();
        allow create, update, delete: if isAdmin();
      }
```

Abbildung 2 Auszug Sicherheitsregeln

Diese Regeln stellen sicher, dass Benutzer ausschließlich ihrer eigenen Daten lesen oder schreiben können und nur Administratoren Änderungen an Kursen oder Fragen vornehmen dürfen. Alle Verbindungen zwischen App und Firestore erfolgen verschlüsselt über HTTPS (TLS 1.2). Zusätzlich werden alle Benutzereingaben in der App validiert, um fehlerhafte oder unerlaubte Datenübertragungen zu verhindern. Damit ist ein robustes, rollenbasiertes Sicherheitssystem gewährleistet, das Datenschutz, Integrität und Stabilität sicherstellt.

4.3.4 Offline-Struktur und Synchronisation

Firestore unterstützt die lokale Datenspeicherung, wodurch alle Daten auch offline verfügbar sind. Wenn das Gerät keine Internetverbindung hat, werden Änderungen zunächst lokal zwischengespeichert. Sobald wieder eine Verbindung besteht, synchronisiert Firestore diese automatisch mit der Cloud. Dies betrifft insbesondere Notizen, Quizversuche und Feedback-Einträge, die Lernende jederzeit erfassen können. Die hierarchische Struktur folgt dabei festen Pfaden, z. B. /users/uid123/notes/note01 oder /courses/course01/units/unit01/questions/q01. Diese klare Organisation ermöglicht eine effiziente Abfrage, Synchronisation und spätere Auswertung der Daten. Sie trägt entscheidend zur Stabilität und Benutzerfreundlichkeit der Anwendung bei, insbesondere in schulischen Netzwerken mit eingeschränktem Internetzugang.

4.4 Benutzeroberfläche

Die grafische Gestaltung orientiert sich am Material Design von Google und folgt dem IAD-Farbschema mit einem kräftigen Rot (#C62828) als Akzentfarbe und Weiß als Hintergrund. Das Erscheinungsbild ist bewusst schlicht und klar gehalten, mit großzügigen Abständen, klarer Typografie und einem hohen Kontrast. Als Schriftart kommt Roboto zum Einsatz.Die wichtigsten Ansichten der App sind: die Login- und Registrierungsseite, die Kursübersicht, die

Quiz-Ansicht, die Ergebnisanzeige, die Fehlerliste, die Notizverwaltung und die Feedback-Seite. Für Administratorinnen und Administratoren existiert eine eigene Verwaltungsoberfläche, über die neue Kurse oder Fragen angelegt, geändert und gelöscht werden können. Alle Ansichten wurden zunächst als Mockups entworfen und anschließend als XML-Layouts umgesetzt. Symbole stammen aus den „Outlined Material Icons“, und sämtliche Oberflächen sind responsiv. Die Entwürfe sind im Anhang A9 zu sehen.

4.5 Technischer Aufbau

Die App besteht aus mehreren Activities und Fragmenten, die gemeinsam die Hauptmodule bilden. Beim Start prüft die Splash-Activity, ob ein Benutzer angemeldet ist, und leitet je nach Status zur Login-Activity oder zur Hauptansicht weiter. Die Main-Activity enthält eine Navigationsleiste, über die auf Kurse, Notizen, Fehlerliste und Profil zugegriffen werden kann. Die Quiz-Funktion bildet den zentralen Bestandteil der Anwendung. Vor Beginn kann die Anzahl der Fragen gewählt werden; die Aufgaben erscheinen in zufälliger Reihenfolge, um Lerneffekte zu vermeiden. Bei Multiple-Choice-Fragen werden Teilpunkte vergeben: richtige Antworten erhöhen, falsche reduzieren die Punktzahl, jedoch nie unter null. Die App bietet zwei Modi – einen Übungsmodus ohne Zeitlimit und einen Prüfungsmodus mit Timer, der das Quiz automatisch beendet. Nach Abschluss werden Ergebnis und Lernfortschritt berechnet; falsch beantwortete Fragen werden in einer Fehlerliste gespeichert. Dank Firestore-Offline-Caching funktioniert die App auch ohne Internetverbindung, wobei Daten nachträglich synchronisiert werden. Ein separater Administrationsbereich ermöglicht autorisierten Benutzerinnen und Benutzern das Erstellen und Bearbeiten von Kursen und Fragen. Der Ablauf ist im Aktivitätsdiagramm (Anhang A10) dargestellt.

5.Implementierungsphase

Die Implementierungsphase umfasst die technische Realisierung der in der Entwurfsphase konzipierten Funktionen. Schwerpunkt lag auf der Umsetzung der Benutzerverwaltung, der Quizlogik und der Firestore-Anbindung, um eine lauffähige Android-App zu schaffen.Alle Module wurden modular aufgebaut, getestet und anschließend zu einer funktionierenden Gesamtanwendung zusammengeführt.

5.1 Implementierung der Datenstrukturen

Zu Beginn der Implementierungsphase wurden alle notwendigen Collections (Sammlungen) und Subcollections angelegt. Die wichtigsten.

Jede dieser Sammlungen wurde durch eine entsprechende Kotlin-Datenklasse im Package data.model abgebildet. Beispiele sind User.kt mit Feldern für UID, Benutzername, E-Mail, Rolle, Punkte und Level, Course.kt mit Titel, Beschreibung und Levelanforderung sowie Question.kt mit Fragetext, Antwortmöglichkeiten und korrekter Lösung. Diese Struktur ermöglicht eine klare Trennung zwischen Anwendungslogik und Datenspeicherung. Über Repository-Klassen (z. B. UserRepository, QuizRepository) werden CRUD-Operationen gekapselt, damit ViewModels unabhängig von Firebase arbeiten können. Die Synchronisierung zwischen Firestore und App erfolgt über LiveData und Coroutines, wodurch Datenänderungen in Echtzeit übernommen werden. Zur Sicherung der Daten wurden Firestore Security Rules implementiert, die Zugriffe nach Benutzerrolle einschränken. Nur Admins dürfen Kurse und Fragen ändern, während Lernende ausschließlich ihre eigenen Dokumente lesen und schreiben dürfen. Die Datenstruktur wurde so gestaltet, dass Offline-Caching und automatische Synchronisierung unterstützt werden. Die Struktur der Datenspeicherung kann im Anhang A11 nachvollzogen werden.

5.2 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wurde mit XML-Layouts in Android Studio entwickelt und folgt den Material-Design-Richtlinien von Google. Zur Gewährleistung einer einheitlichen Struktur kommen ConstraintLayout und LinearLayout zum Einsatz. Die Navigation zwischen den Hauptbereichen der App (Login, Kursübersicht, Quiz, Profil, Notizen) erfolgt über Activities und Buttons. Die Oberflächen wurden activity_login.xml, activity_quiz.xml, activity_profile.xml, activity_admin.xml und activity_notes.xml. Icons und Farben wurden mit der Android-Resource-API verwaltet. Die Einbindung von Animations-Effekten so entworfen, dass sie übersichtlich und barrierearm sind: hohe Kontraste, große Touch-Flächen und responsive Anordnung für verschiedene Displaygrößen. Das Farbschema orientiert sich am Corporate Design der IAD (Mischung aus Rot, Weiß und Grautönen). Zentrale Views sind (z. B. Fade-In bei Activity-Wechseln) verbessert die Benutzererfahrung. Screenshots der fertigen Ansichten befinden sich im Anhang A12.

5.3 Implementierung der Geschäftslogik

Die Geschäftslogik wurde gemäß dem MVVM-Muster umgesetzt. Die Verarbeitung von Benutzereingaben, Datenabrufen und Rückmeldungen geschieht über ViewModels mit Repository-Anbindung. Codeausschnitt aus der MainActivity mit Firebase-Verbindung Anhang A13 Mainactivity) Das LoginViewModel steuert Registrierung, Authentifizierung und Rollenprüfung über FirebaseAuth. (Siehe Anhang A14). Das QuizViewModel lädt die Fragen, prüft Antworten, berechnet den Punktestand und speichert das Ergebnis in attempts. (Siehe

Anhang A15)Das NotesViewModel verwaltet (Anhang A16) CRUD-Operationen auf den Subcollections users/{uid}/notes. Die Anwendung nutzt Kotlin-Coroutines für asynchrone Operationen und stellt so eine flüssige Nutzererfahrung sicher. Die Fehlerbehandlung wird über try-catch-Blöcke und Firebase-Callbacks realisiert, wobei Fehlermeldungen als Toast-Benachrichtigungen angezeigt werden. Das gesamte System ist so konzipiert, dass es bei Ausfällen oder Offline-Phasen keine Datenverluste gibt. Lokale Änderungen werden im Cache gespeichert und sobald möglich synchronisiert. Für die Admins wurde eine separate Logik implementiert, die CRUD-Operationen in den Kurs- und Fragensammlungen erlaubt. (Siehe Anhang A17) Damit wurde die in Kapitel 4 beschriebene Architektur vollständig technisch umgesetzt. Die App ist nach Abschluss dieser Phase funktionsfähig, stabil und bereit für die Test- und Abnahmephase.

6 Testphase

Ziel der Testphase war die Überprüfung der Funktionsfähigkeit, Stabilität und Datensicherheit der UML-Lern-App. Alle Hauptmodule wurden mithilfe von Black-Box-Tests geprüft, bei denen ausschließlich das Verhalten der Anwendung aus Benutzersicht bewertet wurde. Die Tests wurden von mehreren Umschülerinnen und Umschülern der IAD Marburg sowie der Projektbetreuerin durchgeführt. Getestet wurde auf verschiedenen Android Geräten (Versionen 10–13) und im Emulator, um Kompatibilität und Leistung sicherzustellen. Im Fokus standen die Kernfunktionen der App: Die Benutzerverwaltung über Firebase Authentication ermöglichte eine fehlerfreie Registrierung, Anmeldung und Rollentrennung zwischen Lernenden und Administratoren. Das Quizmodul zeigte Fragen korrekt an, berechnete die Punktzahl zuverlässig und speicherte die Ergebnisse in Firestore. Mehrere Testläufe bestätigten, dass der Timer das Quiz korrekt beendete und alle Resultate im Profil angezeigt wurden. Das Notizen-Modul funktionierte stabil; Einträge konnten offline erstellt und nach Wiederverbindung automatisch synchronisiert werden. Auch die Feedback Funktion und das Onboarding-Tutorial wurden erfolgreich getestet – beide arbeiteten fehlerfrei. Prüfungen der Firestore Security Rules zeigten, dass ausschließlich autorisierte Benutzer auf ihre eigenen Daten zugreifen konnten. Die Anwendung zeigte insgesamt eine hohe Stabilität, kurze Ladezeiten und eine zuverlässige Synchronisierung mit Firebase. Kleinere Layoutanpassungen (z. B. Schriftgröße, Button-Abstand) wurden nach Rückmeldung der Testgruppe optimiert. Alle definierten Testfälle verliefen erfolgreich, und die App erfüllt die im Projektantrag festgelegten Anforderungen vollständig. Eine Übersicht der einzelnen Testfälle befindet sich im Anhang A18.

7 Abnahmephase

Nach Abschluss der Testphase wurde die Anwendung der Projektbetreuerin vorgestellt und auf einem Android-Gerät demonstriert. Ziel der Abnahme war die Überprüfung, ob alle im Projektantrag definierten Anforderungen vollständig umgesetzt und die App stabil lauffähig ist. Während der Präsentation wurden die zentralen Funktionen – Benutzerregistrierung, Login, Kursauswahl, Quiz, Notizen, Feedback sowie die administrative Kursverwaltung – erfolgreich vorgeführt. Alle Abläufe liefen fehlerfrei, und die Synchronisierung mit Firebase funktionierte in Echtzeit. Die Betreuerin bestätigte die vollständige Umsetzung der Anforderungen. Besonders positiv bewertet wurden die übersichtliche Benutzeroberfläche, das Offline-Caching und die klare Rollentrennung zwischen Lernenden und Administratoren. Kleinere Anpassungen an Farbgestaltung und Schriftgröße wurden anschließend vorgenommen, um die Lesbarkeit zu optimieren. Das Projekt gilt damit als erfolgreich abgenommen. Ein Abgleich der umgesetzten Funktionen findet sich im Ist-Soll-Vergleich (Anhang A19).

8 Projektdokumentation

Alle Projektschritte – von der Analyse über die Implementierung bis zur Test- und Abnahmephase – wurden systematisch geplant, umgesetzt und dokumentiert. Die technische Umsetzung erfolgte in Android Studio mit Kotlin unter Verwendung von Firebase für Authentifizierung, Datenspeicherung und Sicherheit. Struktur, Datenmodelle und Architekturentscheidungen sind in dieser Dokumentation beschrieben und durch Diagramme, Codebeispiele und Testergebnisse im Anhang nachvollziehbar dargestellt. Das Projektziel, den Lernprozess für UML-Diagramme zu digitalisieren, wurde erreicht. Die Anwendung überzeugt durch Stabilität, Benutzerfreundlichkeit und eine sichere, rollenbasierte Zugriffskontrolle.

9.Fazit und Ausblick

9.1 Lessons Learned

Im Verlauf der Entwicklung der UML-Lern-App konnten umfangreiche praktische Erfahrungen in der Planung, Umsetzung und Qualitätssicherung eines Softwareprojekts gesammelt werden. Das Projekt bot die Möglichkeit, das theoretische Wissen aus der Umschulung in einem realen Anwendungsszenario anzuwenden und zu vertiefen.

Ein wesentlicher Erkenntnisgewinn ergab sich durch die Arbeit mit der MVVM-Architektur, die eine klare Trennung von Daten, Logik und Präsentation ermöglicht. Dabei wurde deutlich, wie wichtig eine strukturierte Codeorganisation und ein sauberes Datenmodell für Wartbarkeit, Erweiterbarkeit und Fehlersuche sind. Die konsequente Anwendung dieses Musters trug wesentlich zur Stabilität und Übersichtlichkeit der Anwendung bei.

Ein weiterer zentraler Lernaspekt war der Einsatz von Firebase als Cloud-Backend. Die Implementierung sicherer Firestore-Regeln, die Datensynchronisation zwischen Offline- und Online Betrieb sowie die Nutzung von Firebase Authentication führten zu einem vertieften Verständnis moderner Cloud-Technologien. Diese Themenbereiche waren im regulären Unterricht bislang nicht enthalten und wurden durch eigenständige Recherche praxisnah erarbeitet.

Auch die Entwicklung in der Programmiersprache Kotlin stellte einen erheblichen Wissenszuwachs dar. Da Kotlin nicht Teil des Umschulungsplans war, bot das Projekt die Gelegenheit, eine moderne, für Android optimierte Sprache eigenständig und sicher anzuwenden. Darüber hinaus förderte die Arbeit mit GitHub die strukturierte Versionskontrolle und ein methodisches Vorgehen in der Entwicklung. Die frühzeitige Planung und Durchführung von Testfällen verdeutlichte den Zusammenhang zwischen Softwarequalität, Code-Struktur und Teststrategie. Abschließend lässt sich feststellen, dass das Projekt das Verständnis für Softwarearchitektur, Cloud-Integration, App-Design und agile Entwicklungsprozesse wesentlich vertieft hat.

Es stärkte sowohl das technische als auch das methodische Denken und zeigte die Bedeutung von Planung, Dokumentation und kontinuierlicher Qualitätskontrolle für den erfolgreichen Abschluss eines IT-Projekts.

9-2 Ausblick

Für die Weiterentwicklung der UML-Lern-App sind mehrere funktionale Erweiterungen geplant. Eine Statistikfunktion soll künftig den Lernfortschritt visuell darstellen, und eine Exportoption ermöglicht die Ausgabe von Ergebnissen im PDF- oder Excel-Format. Ein Zertifikatsmodul wird automatisch Bestätigungen für erfolgreich absolvierte Einheiten erstellen. Darüber hinaus ist eine Funktion vorgesehen, mit der Lernende UML-Diagramme direkt in der App skizzieren können, um ihr Wissen praktisch anzuwenden. Langfristig ist eine Portierung auf iOS sowie eine Erweiterung um weitere Lernbereiche wie SQL oder Software-Testing denkbar

A Anhang

A1 Zeitplan / Stundenverteilung

Zurück zum Kapitel 2.1

Anhang A1 Projektphasenplan

Der folgende Projektphasenplan zeigt die zeitliche Struktur und den Umfang der einzelnen Arbeitsschritte des Projekts „Interaktive Lern-App mit Quiz, Notizen, Lernpfaden und Kursverwaltung“. Die Phasen entsprechen dem Projektantrag der IHK.

Nr.	Projektphase	Beschreibung	Zeit in Stunden
1	IST-Analyse	Sichtung vorhandener Lernmaterialien, Analyse der Anforderungen.	3
2	Soll-Konzept	Definition der Zielsetzung, Funktionsumfang (Module, Quiz, Fortschrittsanzeige).	3
3	Planungsphase	Erstellung des Projektplans, Risikoanalyse, Auswahl der Firebase-Struktur.	3
4	Datenmodellierung	Entwurf der Firestore-Struktur (Sammlungen, Dokumente, Felder).	4
5	Oberflächenentwurf	Erstellung von Layouts (XML) für Login, Kursübersicht, Quiz, Profil, Notizen, Fehler-Wiederholung.	4
6	Implementierung	Firebase-Anbindung, Lernkontrolle-Logik, Benutzerprofil, DB-Zugriff und alle Funktionen.	40
7	Testing	Funktionstests, UI-Tests, Offline-Tests, Fehlerbehebung.	7
8	Dokumentation	Projektbericht, Diagramme, Screenshots, Quellcode-Auszüge.	16
	Gesamt		80

A2 Funktionsübersicht

Zurück zum Kapitel 3.2

Funktionsübersicht UML-Lern-App mit Firebase-Anbindung

Nr.	Funktion	Kurzbeschreibung
1	Registrierung & Login	Nutzer können ein Konto anlegen, sich anmelden und abmelden. Authentifizierung erfolgt über Firebase Authentication.
2	Rollen & Rechte	Zwei Rollen: Lernender und Administrator. Nur Admins dürfen Kurse und Fragen verwalten.
3	Kursauswahl	Lernende wählen einen Kurs und erhalten passende Lerninhalte und Quizfragen.
4	Quiz durchführen	Fragen beantworten, Feedback erhalten, Punkte sammeln.
5	Automatische Auswertung	Nach jeder Frage oder am Ende wird die Punktzahl automatisch berechnet.
6	Fehlerliste	Falsch beantwortete Fragen werden gespeichert und können wiederholt werden.
7	Notizen	Persönliche Notizen pro Frage speichern, bearbeiten und löschen.
8	Levelsystem	Automatischer Aufstieg ins nächste Level nach Erreichen einer Punktzahl.
9	Onboarding-Tutorial	Kurze Einführung für neue Nutzer zu den wichtigsten App-Funktionen.
10	In-App-Feedback	Nutzer können Rückmeldungen und Vorschläge direkt in der App absenden.
11	Datensicherheit	Datenzugriff ist durch Firebase Security Rules geschützt.
12	Timer-Funktion	Prüfungsmodus mit begrenzter Zeit, z. B. 20 Minuten für 10 Fragen.
13	Zufällige Reihenfolge	Fragen und Antworten werden zufällig angezeigt, um Auswendiglernen zu vermeiden.
14	Frageanzahl wählbar	Nutzer können vor dem Quiz entscheiden, wie viele Fragen sie beantworten möchten.

Interaktive UML-Lern-App mit Firebase-Anbindung

15	Offline-Funktion	Mit dem Firebase-Offline-Cache sind Inhalte auch ohne Internet verfügbar.

A3 Pflichtenheft (Auszug)

Zurück zum Kapitel 3.4

- **Ist-Zustand:** Papierbasierte UML-Übungen ohne Feedback.
- **Soll-Zustand:** Digitale App mit interaktivem Quiz, Notizen, Levels und Verwaltung.
- **Qualitätsanforderungen:** Zuverlässigkeit, Benutzerfreundlichkeit, Datensicherheit.

4. Systemarchitektur

- **Zielplattform:** Android (Kotlin in Android Studio).
- **Backend:** Firebase (Authentifizierung, Firestore für Daten, Storage für Diagramme).
- **Sicherheit:** Rollenbasierte Zugriffsrechte via Security Rules.

5. Funktionale Anforderungen

- **Authentifizierung:** Registrierung/Login mit Firebase Auth.
- **Rollenmanagement:** User lernen, Admins verwalten (Kotlin-Logik).
- **Kurs- und Quiz-Logik:** Dynamische Lade- und Auswertungsfunktionen (Firestore).
- **Levelsystem:** Automatische Freischaltung bei 80% Punkten.
- **Offline-Funktion:** Firebase-Cache für Inhalte.

1

Pflichtenheft: UML-Lern-App Josiane Kanouo Maneyo

6. Nicht-funktionale Anforderungen

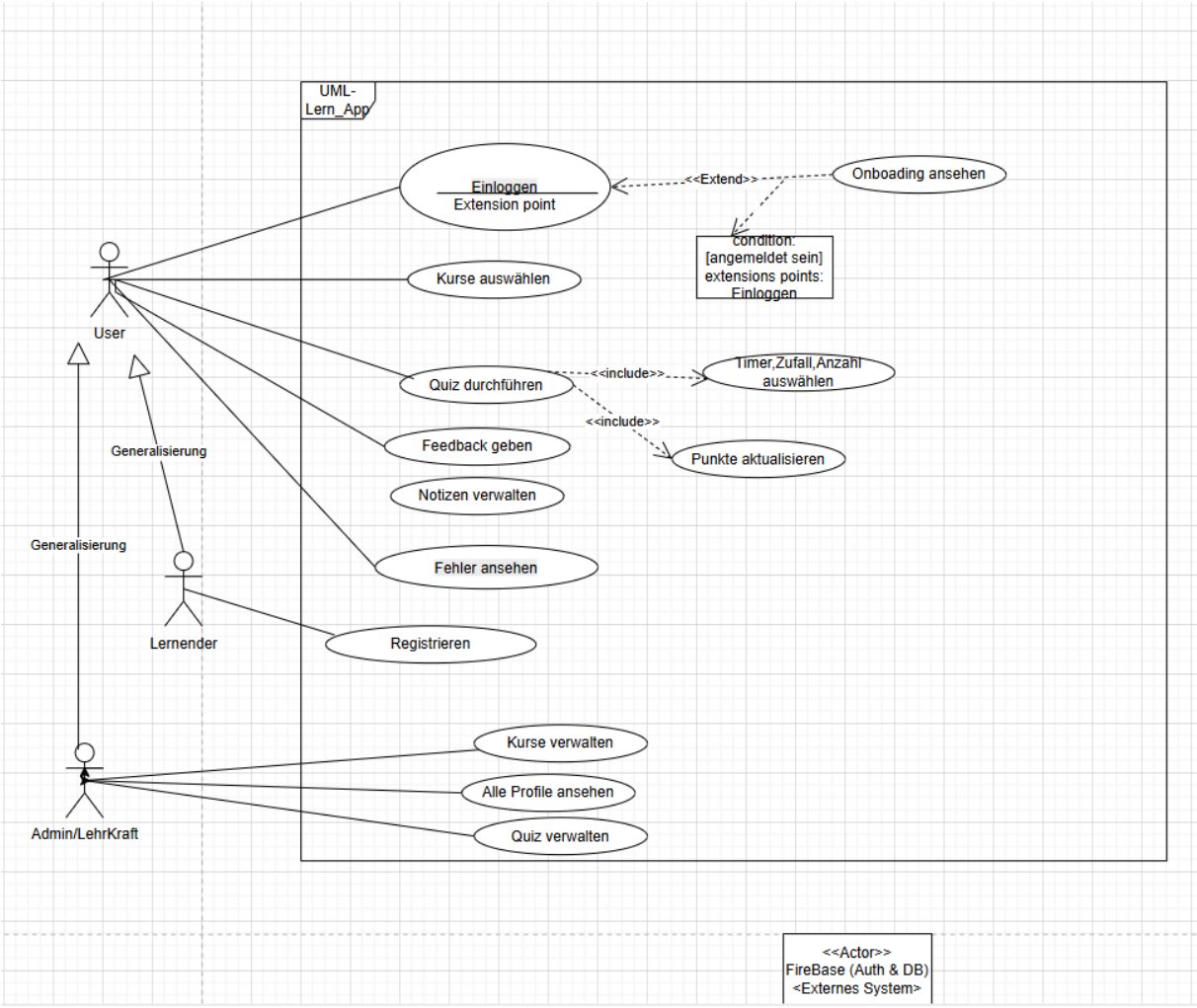
- **Leistung:** Antworten in <2 Sekunden auswerten.
- **Skalierbarkeit:** Bis zu 500 Nutzer gleichzeitig.
- **Sicherheit:** Verschlüsselte Datenübertragung (HTTPS).

7. Test- und Akzeptanzkriterien

- Erfolgreiche Anmeldung und Kurszugriff.
- Korrektes Quiz-Ergebnis (Punkte, Fehlerliste).
- Admin kann Kurse hinzufügen und Fortschritte sehen.

A4 Use case Diagramm

Zurück zum Kapitel 3.5



A5 Lastenheft (Auszug)

Zurück zum Kapitel 3.6

4. Abgrenzung zu Kauflösungen

Die App unterscheidet sich durch:

- Eigene Notizen: Nutzer können Notizen speichern und bearbeiten
- Klare Lernpfade (Levels): Strukturierter Lernweg von Grundlagen zu schwierigen Aufgaben
- Einfache Verwaltung für Admins: Kurse und Fragen anlegen/ändern

1

5. Funktionen (Anforderungen)

- **Registrierung & Login:** Nutzer legen ein Konto an und melden sich an.
- **Rollen & Rechte:** User lernen, Admins verwalten Inhalte.
- **Kursauswahl:** Nutzer wählen Kurse mit passenden Fragen.
- **Quiz durchführen:** Fragen beantworten, Feedback erhalten.
- **Automatische Auswertung:** Punkte und Erfolgsquote berechnen.
- **Fehlerliste:** Falsche Fragen speichern und wiederholen.
- **Notizen:** Persönliche Notizen pro Frage speichern.
- **Levelsystem:** Automatischer Aufstieg bei genügend Punkten.
- **Timer-Funktion:** Prüfungsmodus mit Zeitbegrenzung.
- **Zufällige Reihenfolge:** Fragen und Antworten mischen.
- **Frageanzahl wählbar:** Nutzer wählen die Anzahl der Fragen.
- **Offline-Funktion:** Inhalte über Firebase-Cache verfügbar.
- **Datensicherheit:** Zugriff durch Firebase Security Rules geschützt.

6. Wirtschaftlicher Nutzen

Die App reduziert Kosten für Papierübungen, ermöglicht zentrale Auswertung und steigert die Lernmotivation durch interaktive Elemente.

A6: Build Gradle & dependencies

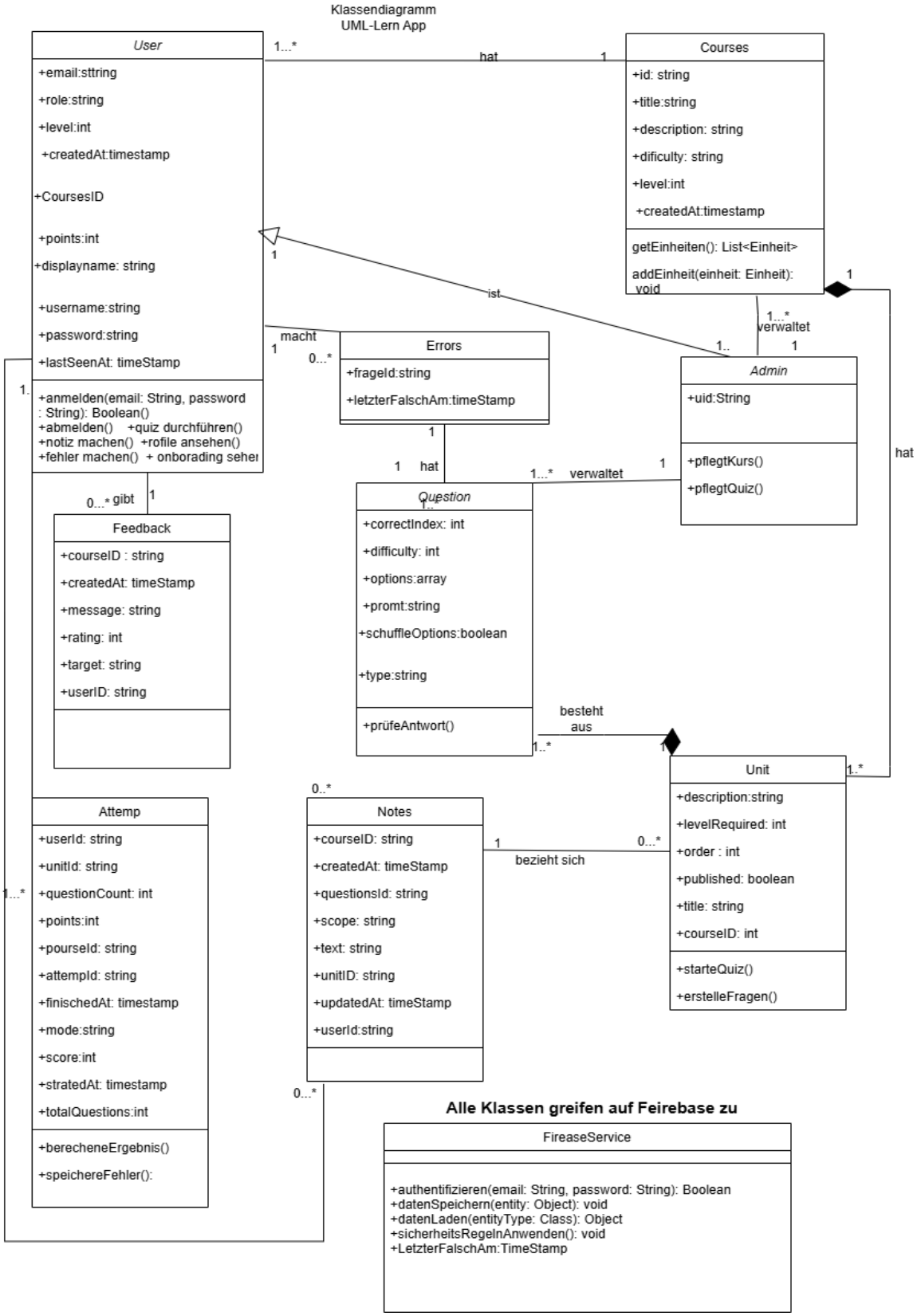
Zurück zum Kapitel 4.1

build.gradle.kts (:app)

```
android {
    buildTypes {
        release {
            ...files = getDefaultProguardFile( name = "proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    kotlinOptions {
        jvmTarget = "11"
    }
}

dependencies {
    // Firebase Plattform (steuert Versionen)
    implementation(platform( notation = "com.google.firebase:firebase-bom:33.6.0"))

    // • Benötigt für FirebaseAuth
    implementation("com.google.firebase:firebase-auth-ktx")

    // • Firestore (KTX - enthält u. a. .toObject<T>())
    implementation("com.google.firebase:firebase-firestore-ktx")
    implementation("com.google.android.material:material:1.11.0")


    // Deine bisherigen Android-Dependencies
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.appcompat)
    implementation(libs.material)
    implementation(libs.androidx.activity)
    implementation(libs.androidx.constraintlayout)

    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
}
```

A7: Klassendiagramm

Zurück zum Kapitel 4.3.2



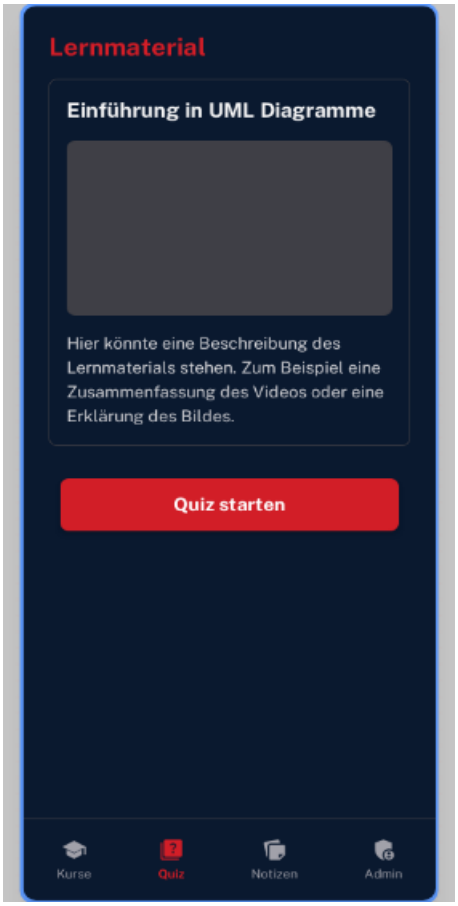
A8:Firebase Rules

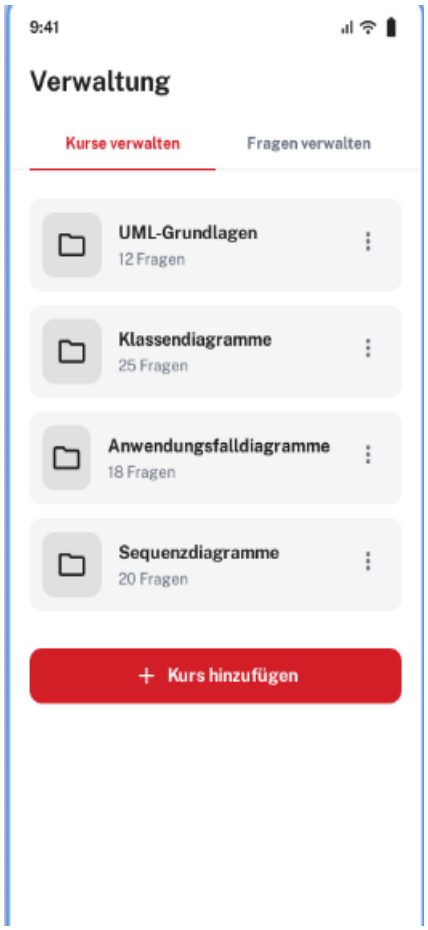
Zurück zum Kapitel 4.3.3

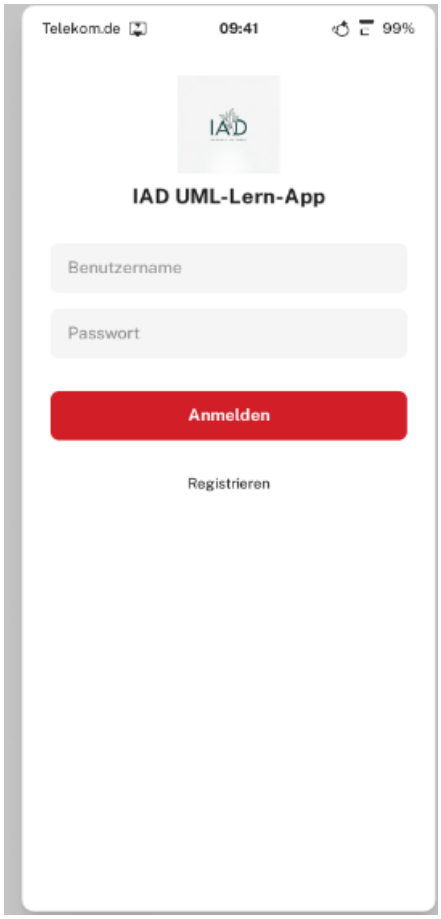
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    function signedIn() { return request.auth != null; }
    function me() { return request.auth.uid; }
    function isAdmin() {
      return signedIn()
        && exists(/databases/$(database)/documents/users/$(me()))
        && get(/databases/$(database)/documents/users/$(me())).data.role == "admin";
    }

    // USERS
    match /users/{userId} {
      allow read: if true; // für Username/Email-Lookup
      allow create: if signedIn() && userId == me();
      allow update, delete: if signedIn() && (userId == me() || isAdmin());
    }

    // COURSES (+ nested)
    match /courses/{courseId} {
      allow read: if signedIn();
      allow create, update, delete: if isAdmin();

      match /questions/{qId} {
        allow read: if signedIn();
        allow create, update, delete: if isAdmin();
      }

      match /units/{unitId} {
        allow read: if signedIn();
        allow create, update, delete: if isAdmin();
      }
    }
```

A9: Mock ups

Zurück zum Kapitel 4.4

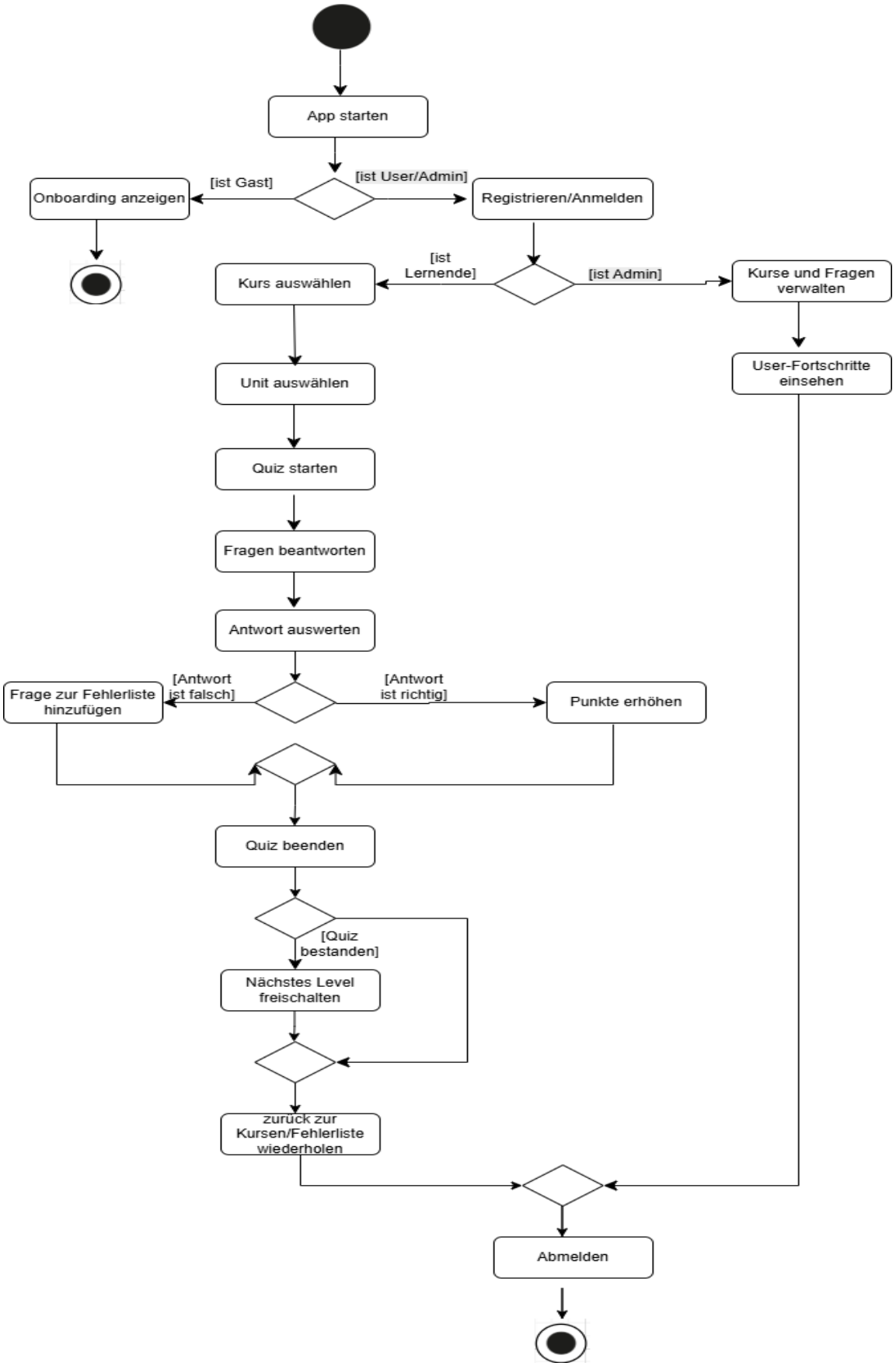





Telekom.de
09:41
99%
IAD UML-Lern-App
Benutzername
Passwort
Anmelden
Registrieren

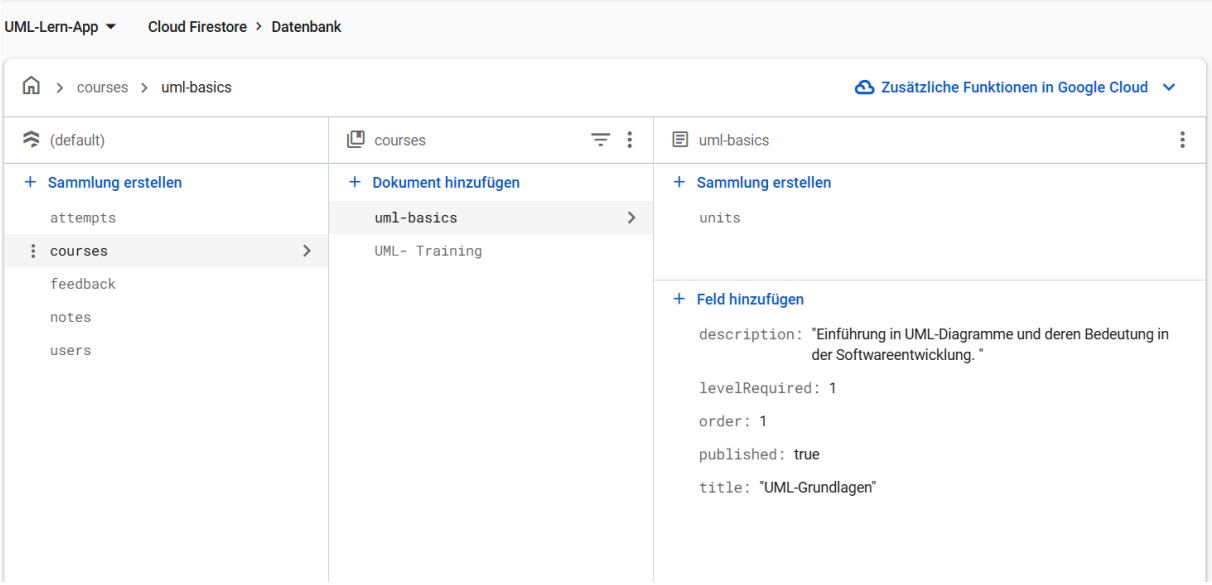
A10:Aktivitätsdiagramm

Zurück zum Kapitel 4.5



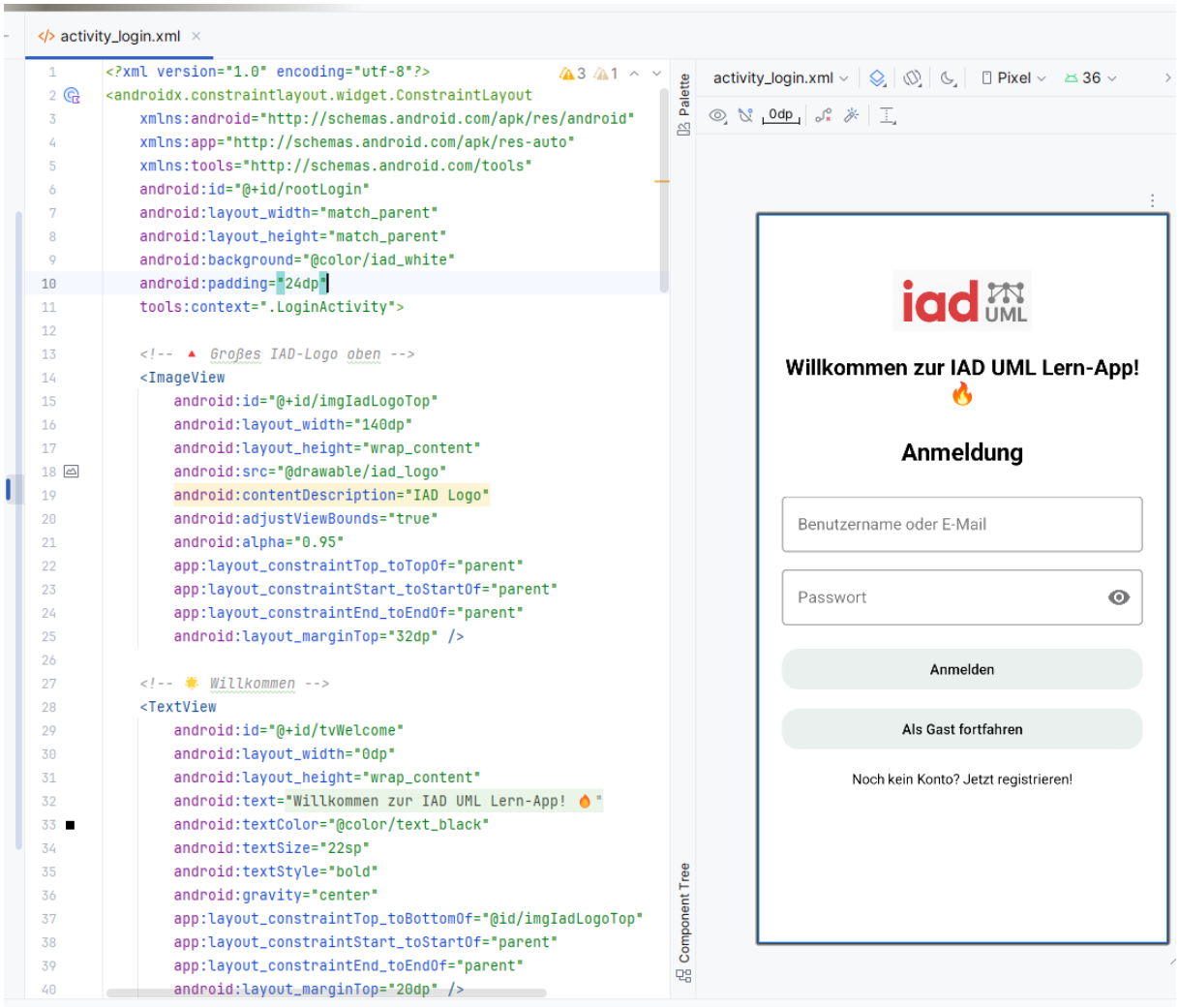
A11 Firebase Ansicht

Zurück zum Kapitel 5.1



A12 Screenshort GUI Login

Zurück zum Kapitel 5.2



A13: Screenshort Codeauszug Main

Zurück zum Kapitel 5.3

MainActivity.kt

```kotlin
package com.example.uml_lern_app

import ...

2 Usages
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_register)

        // ---------------------------------------------------------------
        // Verbindung zur Firestore-Datenbank wird hergestellt.
        // Diese Instanz ermöglicht Lese- und Schreibzugriffe auf Collections.
        // ---------------------------------------------------------------
        val db = FirebaseFirestore.getInstance()

        // ---------------------------------------------------------------
        // Lese-Test: Abfrage aller Dokumente aus der Collection "courses".
        // Die Ergebnisse werden nach dem Zahlenfeld "order" sortiert.
        // Bei Erfolg werden alle Kurse im Log ausgegeben.
        // ---------------------------------------------------------------
        db.collection( collectionPath = "courses")
            .orderBy( field = "order")
            .get()
            .addOnSuccessListener { daten ->
                if (daten.isEmpty) {
                    Log.d( tag = "COURSE",  msg = "Keine Kurse gefunden.")
                } else {
                    for (doc in daten.documents) {
                        val titel = doc.getString( field = "title")
                        Log.d( tag = "COURSE",  msg = "id=${doc.id} title=$titel")
                    }
                }
            }
            .addOnFailureListener { fehler ->
                Log.e( tag = "COURSE",  msg = "Fehler beim Lesen",  tr = fehler)
            }

    }
```

A14 : Codeauszug (Login)

Zurück zum Kapitel 5.3

LoginActivity.kt

```
8 Usages
class LoginActivity : AppCompatActivity() {

    // ViewBinding: stellt typsichere Referenzen auf die Views bereit
    12 Usages
    private lateinit var binding: ActivityLoginBinding

    // Firebase: Auth + Firestore
    2 Usages
    private val auth: FirebaseAuth by lazy { FirebaseAuth.getInstance() }
    private val db by lazy { Firebase.firestore }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // 1) Layout setzen (activity_login.xml)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // 2) LOGIN-Button: username/email → email auflösen → signIn
        binding.btnLogin.setOnClickListener {
            val usernameOrEmail = binding.etUsername.text?.toString()?.trim().orEmpty()
            val password        = binding.etPassword.text?.toString().orEmpty()

            if (usernameOrEmail.isEmpty() || password.isEmpty()) {
                toast( msg = "Bitte Benutzername/E-Mail und Passwort eingeben.")
                return@setOnClickListener
            }

            setLoading(true) // UI sperren + Progress zeigen
```

A15 : Codeauszug (Quiz)

Zurück zum Kapitel 5.3

QuizActivity.kt

```
18      class QuizActivity : AppCompatActivity() {
290         private fun loadWrongSet(courseId: String): Set<String> {
291             val key = KEY_WRONG_PREFIX + courseId
292             val json = prefs().getString(key, defValue = "[]") ?: "[]"
293             val arr = JSONArray(json)
294             val set = mutableSetOf<String>()
295             for (i in 0 ≤ until < arr.length()) set += arr.optString( index = i)
296             return set
297         }
298
            1 Usage
299         private fun saveWrong(questionText: String, courseId: String) {
300             val key = KEY_WRONG_PREFIX + courseId
301             val set = loadWrongSet(courseId).toMutableSet()
302             set += questionText
303             val arr = JSONArray()
304             set.forEach { arr.put( value = it) }
305             prefs().edit().putString(key, arr.toString()).apply()
306         }
307
            1 Usage
308         private fun removeWrong(questionText: String, courseId: String) {
309             val key = KEY_WRONG_PREFIX + courseId
310             val set = loadWrongSet(courseId).toMutableSet()
311             if (set.remove( element = questionText)) {
312                 val arr = JSONArray()
313                 set.forEach { arr.put( value = it) }
314                 prefs().edit().putString(key, arr.toString()).apply()
315             }
316         }
317
            1 Usage
318         private fun addProfilePoints(delta: Int) {
319             val current = prefs().getInt( key = KEY_PROFILE_POINTS, defValue = 0)
320             prefs().edit().putInt(KEY_PROFILE_POINTS, current + delta).apply()
321         }
322
            1 Usage
323         private fun setUnitPassed(unitId: String, passed: Boolean) {
324             prefs().edit().putBoolean(KEY_PASSED_PREFIX + unitId, passed).apply()
```

A16 : Codeauszug (Notes)

Zurück zum Kapitel 5.3

NotesActivity.kt

```
27      class NotesActivity : AppCompatActivity() {
54          override fun onCreate(savedInstanceState: Bundle?) {
76              unitId = intent.getStringExtra( name = "unitId")
77              questionId = intent.getStringExtra( name = "questionId")
78
79              // Liste einrichten
80              adapter = NotesAdapter(
81                  items = notes,
82                  onEdit = { note -> startEdit(note) },
83                  onDelete = { note -> deleteNote( id = note.id) }
84              )
85              rvNotes.layoutManager = LinearLayoutManager( context = this)
86              rvNotes.adapter = adapter
87
88              // Filterwechsel → neu laden
89              spFilter.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
90                  override fun onItemSelected(parent: AdapterView<*>, v: View?, pos: Int, id: Long) = loadNotes()
91                  override fun onNothingSelected(parent: AdapterView<*>) {}
92              }
93
94              // Neu → Editor öffnen
95              btnAdd.setOnClickListener {
96                  editingNoteId = null
97                  etNote.setText("")
98                  etNote.visibility = View.VISIBLE
99                  btnSave.visibility = View.VISIBLE
100                 etNote.requestFocus()
101             }
102
103             // Speichern → create oder update
104             btnSave.setOnClickListener {
105                 val text = etNote.text.toString().trim()
106                 if (text.isEmpty()) {
107                     Toast.makeText( context = this,  text = "Bitte Text eingeben",  duration = Toast.LENGTH_SHORT).show()
108                     return@setOnClickListener
109                 }
110                 if (editingNoteId == null) createNote(text) else updateNote( id = editingNoteId!!, text)
111             }
st) 112
```

A17: Codeauszug (Admin)

Zurück zum Kapitel 5.3

AdminActivity.kt

```
26      class AdminActivity : AppCompatActivity() {
41          override fun onCreate(savedInstanceState: Bundle?) {
64              subscribeCourses()
65          }
66
67          override fun onStart() {
68              super.onStart()
69              guardAdmin()   // ← NEU: Zugriff prüfen
70          }
71
72          // --- Admin-Guard (offline: Cache, dann Server) ---
            1 Usage
73          private fun guardAdmin() {
74              val u = auth.currentUser
75              if (u == null) {
76                  Toast.makeText( context = this,  text = "Bitte zuerst einloggen.",  duration = Toast.LENGTH_SHORT).show()
77                  finish()
78                  return
79              }
80              val doc = db.collection( collectionPath = "users").document( documentPath = u.uid)
81
82              // 1) Cache (offline)
83              doc.get(Source.CACHE)
84                  .addOnSuccessListener { d ->
85                      if ((d.getString( field = "role") ?: "") == "admin") isAdmin = true
86                  }
87                  .addOnCompleteListener {
88                      // 2) Server (online)
89                      doc.get(Source.SERVER)
90                          .addOnSuccessListener { fresh ->
91                              isAdmin = (fresh.getString( field = "role") == "admin")
92                              if (!isAdmin) {
93                                  Toast.makeText( context = this,  text = "Kein Admin-Zugriff.",  duration = Toast.LENGTH_LONG).show()
94                                  finish()
95                              }
96                          }
```

A18 Testfälle

Zurück zum Kapitel 6

A19 : Ist/ Soll-Vergleich

Zurück zum Kapitel 7

Soll & Ist Nr	Ziel(soll)	Ergebnis
1	Benutzer kann sich mit Benutzername + Passwort anmelden	Anmeldung funktioniert über LoginActivity.kt mit Firebase Authentication; Fehlermeldungen werden korrekt angezeigt
2	Rollenverwaltung (Lernender / Admin)	Firestore-Datenfeld *role* steuert Schreibrechte; Lernende haben Leserechte, Admins vollen Zugriff
3	Kursauswahl und Laden der Fragen aus Firestore	CourseSelectionActivity lädt Kurse per collection("courses"); Auswahl über Spinner funktioniert stabil
4	Quiz mit Punktebewertung und Auswertung	QuizActivity prüft Antworten, zählt Punkte, speichert Ergebnis in *attempts*
5	Speicherung falscher Antworten (Fehlerliste)	Fragen-IDs werden in users/{uid}/mistakes gespeichert und abrufbar angezeigt
6	Notizfunktion für eigene Lernhinweise	users/{uid}/notes Subcollection implementiert; Einträge bleiben dauerhaft gespeichert
7	Offline-Modus	Firestore-Cache aktiviert; Tests ohne Internet erfolgreich
8	Timer im Prüfungsmodus	CountdownTimer mit Sekunden-Anzeige integriert; Zeitlimit stoppt Quiz automatisch
9	Adminbereich zur Datenpflege	AdminActivity erstellt, bearbeitet und löscht Kurs- und Fragen-Daten zuverlässig
10	Feedback-Funktion	Formular in FeedbackActivity sendet Text an *feedback*-Collection
11	Level-System / Fortschritt	Punkte und Level werden in *users*-Collection gespeichert und angezeigt
12	Statistik / Zertifikat	Datenbasis vorbereitet, Umsetzung für nächste Version geplant
13	Design / Usability	Einheitliche Farben (IAD-Rot), responsives Layout für verschiedene Displays
14	Dokumentation & Abnahme	Entwickler-, Benutzer-, Projektdokumentation erstellt und signiert abgenommen