

1.1.2025

Fahrschul-Manager

Abschlussprojekt für den Techniker in Informatik



Christophe Paleyron, Luis Schulte

Eidesstattliche Erklärung:

Hiermit versichern wir, dass wir die vorliegende Projektarbeit selbstständig verfasst und keine anderen als angegebenen Quellen und Hilfsmittel benutzt haben. Alle Ausführungen, die anderen veröffentlichten oder nicht veröffentlichten Schriften wörtlich oder sinngemäß entnommen wurden, haben wir kenntlich gemacht.

Die Projektarbeit war in gleicher oder ähnlicher Fassung noch kein Bestandteil einer anderen Prüfungsleistung.

Nürnberg, den 15.04.2025

Christophe Paleyron

Luis Schulte

1 Inhaltsverzeichnis

2	Vorwort.....	3
3	Einleitung.....	3
3.1	Projektbeschreibung	3
3.2	Projektziel.....	4
3.3	Motivation.....	4
3.4	Projektvereinbarung	Fehler! Textmarke nicht definiert.
4	Anforderungen und Planung.....	5
4.1	Funktionale Anforderungen	5
4.2	Nicht-Funktionale Anforderungen	5
4.3	Technische Anforderungen	6
4.4	Entity-Relationship-Modell	7
4.5	Logisches Modell.....	9
4.6	Designschema	11
4.7	Zeitplan	13
5	Architektur und Design	14
5.1	Systemarchitektur	14
5.2	Datenbankdesign	15
5.3	API-Spezifikationen	Fehler! Textmarke nicht definiert.
5.4	Benutzeroberfläche.....	16
6	Implementierung	23
6.1	Verwendete Technologien	23
6.2	Code-Struktur.....	25
6.3	Besondere Herausforderungen	27
7	Test und Qualitätssicherung	29
7.1	Teststrategie	29
7.2	Testfälle	Fehler! Textmarke nicht definiert.
7.3	Fehlerbehebungen	29
8	Fazit und Ausblick	29
8.1	Zusammenfassung	29
8.2	Kritische Reflexion.....	29
8.3	Verbesserungspotenzial	30
9	Anhang.....	30
9.1	Quellen.....	30
9.2	Literaturverzeichnis.....	30

9.3	Abbildungsverzeichnis.....	30
-----	----------------------------	----

2 Vorwort

Im Rahmen unserer Weiterbildung zum staatlich geprüften Techniker im Fachbereich Informatik an der Rudolf-Diesel-Fachschule erstreckt sich über das dritte und vierte Semester eine Team-Projektarbeit, die wir eigenständig konzipieren und umsetzen. Diese beinhaltet unter anderem die Erstellung einer technischen Dokumentation sowie die Vorstellung der Zwischen- und Endergebnisse. Ziel dieser Arbeit ist es, die im Unterricht erworbenen Fachkenntnisse praktisch anzuwenden und zu vertiefen, Einblicke in Projektmanagement zu gewinnen und unsere sozialen Kompetenzen zu fördern.

Zur Überprüfung der Machbarkeit des Projekts finden zwei ausführliche Meilensteinsitzungen statt. Hierbei werden insbesondere der Zeitplan und die Umsetzbarkeit der gesteckten Anforderungen bewertet. Zudem reichen wir Stundennachweise und Protokolle an den Meilensteinsitzungen ein. Diese Sitzungen bieten auch die Gelegenheit, gemeinsam mit der betreuenden Lehrkraft – die als Projektbegleiter agiert – den aktuellen Fortschritt, erreichte Ziele und anstehende Schritte zu besprechen. Den Abschluss bildet die Präsentation der Ergebnisse auf der Technikerbörse im April, wo Unternehmen und Interessierte die Möglichkeit haben, unsere Arbeit zu begutachten und uns umfassend dazu zu befragen.

Im Anschluss wird die Projektarbeit „Fahrschul-Manager“ vorgestellt, die sich mit der Entwicklung einer Verwaltungs-App für Fahrschulen beschäftigt. Der Text beschreibt das Projekt und seine Zielsetzungen, erläutert die Planung und Umsetzung und schließt mit einem Fazit sowie einer kritischen Selbstreflexion ab.

3 Einleitung

3.1 Projektbeschreibung

In Fahrschulen ist die Verwaltung von Terminen und Fahrzeugen oft mit großem Aufwand verbunden. Kurzfristig abgesagte Fahrstunden bleiben häufig ungenutzt, da es schwierig ist, schnell Ersatz zu finden. Zudem führt die Organisation gemeinschaftlich genutzter Fahrzeuge zu zusätzlicher administrativer Belastung.

Die geplante App soll diesen Problemen entgegenwirken und die Termin- sowie Fahrzeugverwaltung effizienter gestalten. Fahrlehrer und Fahrschüler erhalten eine zentrale Plattform, über die sie ihre Termine verwalten können. Wird eine Fahrstunde abgesagt, können Fahrlehrer den freigewordenen Timeslot direkt veröffentlichen, sodass Fahrschüler sich für diesen eintragen können. Dadurch lassen sich Leerzeiten reduzieren und die Auslastung der Fahrstunden optimieren.

Zusätzlich wird die Buchung von Fahrzeugen erleichtert, um die gemeinsame Nutzung besser zu koordinieren. Fahrlehrer können sich in der App registrieren, neue Fahrschulen anlegen oder sich

bestehenden Fahrschulen anschließen. Bei der Anmeldung neuer Fahrschüler werden diese automatisch ihrem jeweiligen Fahrlehrer zugeordnet, was den Verwaltungsaufwand weiter minimiert.

3.2 Projektziel

Ziel dieses Projekts ist die Entwicklung einer App, die Fahrschulen bei der effizienten Verwaltung von Terminen und Fahrzeugen unterstützt. Durch eine zentrale Plattform sollen spontane Terminabsagen schnell und unkompliziert neu vergeben werden, um Leerzeiten zu vermeiden. Gleichzeitig wird die Fahrzeugbuchung optimiert, um den administrativen Aufwand zu reduzieren.

Fahrlehrer und Fahrschüler erhalten über die App eine benutzerfreundliche Oberfläche zur Terminverwaltung. Fahrlehrer können neue Fahrschüler direkt zuweisen und freigewordene Fahrstunden mit wenigen Klicks veröffentlichen. Fahrschüler können sich in diese flexibel eintragen. Zudem ermöglicht die App eine übersichtliche Verwaltung gemeinschaftlich genutzter Fahrzeuge.

Durch diese digitalen Lösungen wird die Kommunikation zwischen Fahrschülern und Fahrlehrern verbessert, der organisatorische Aufwand erheblich reduziert und die Produktivität in Fahrschulen nachhaltig gesteigert.

Um dieses Ziel zu erreichen haben wir uns folgende Aufteilung überlegt in der wir Teilziele und Arbeitspakete aufgeführt haben:

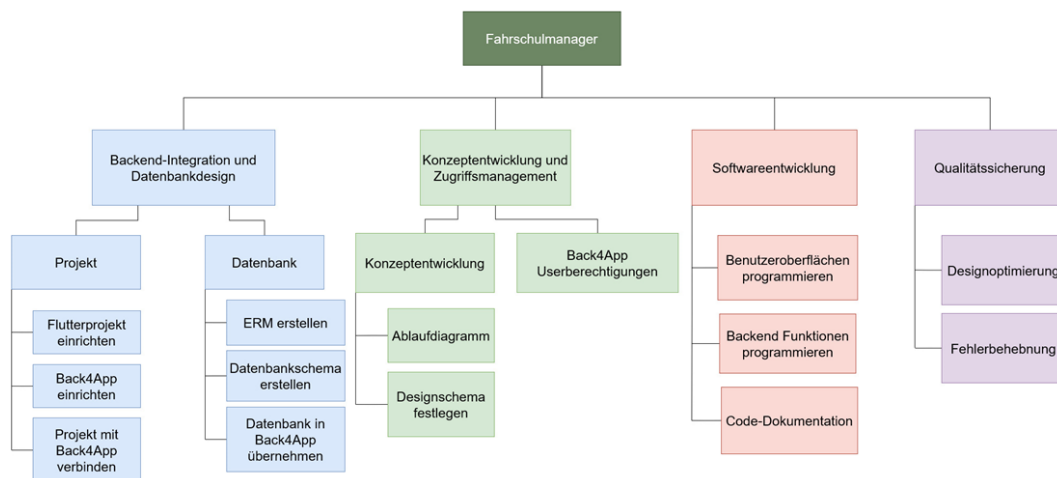


Abbildung 1: Teilziele

3.3 Motivation

Die Motivation für dieses Projekt entstand aus unserem engeren Umfeld, in dem jemand als Fahrlehrer tätig ist. Immer wieder wurde berichtet, dass das Verwalten der Fahrzeuge, und der neuen Terminvergabe bei spontan abgesagten Terminen für Probleme oder Stress gesorgt hatte. Nicht selten entstanden dadurch Leerläufe zwischen Fahrstunden, welche für Fahrlehrer in vielerlei Hinsicht ein Problem sind. Nicht nur das versucht werden muss die Zeit sinnvoll zu überbrücken, sondern vielmehr das Problem, dass, wenn in diesem Zeitraum keine Fahrstunde abgehalten wird, diese auch nicht als Arbeitszeit für die Wochenstunden gezählt werden und somit auch kein Geld verdient wird. Durch die Digitalisierung dieser Prozesse werden Fahrschulen zukunftssicherer und konkurrenzfähiger in einer

zunehmend digitalisierten Branche. Anhand dieser Problematik ergab sich die Idee eine App zu entwickeln welche genau an diesen Stellen eine Erleichterung im Arbeitsalltag für Fahrschulen und Fahrlehrer schaffen soll.

4 Anforderungen und Planung

4.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben konkrete Funktionen und Interaktionen der Fahrschul-Manager App. Sie definieren, welche Aufgaben die App erfüllen muss, um eine effiziente Verwaltung von Fahrstunden, Fahrzeugen und Benutzern zu ermöglichen.

Für unsere App haben wir uns folgende Funktionale Anforderungen überlegt:

Benutzerverwaltung

- Fahrlehrer können eine neue Fahrschule erstellen
- Fahrlehrer können neue Mitarbeiter anlegen
- Fahrlehrer können neue Fahrschüler anlegen und diesen zugeordnet werden
- Rollenbasierte Zugriffskontrolle: Fahrlehrer und Fahrschüler haben unterschiedliche Berechtigungen

Terminverwaltung

- Fahrlehrer können Fahrstunden planen, verwalten und bearbeiten
- Fahrstunden können von Fahrschülern gebucht werden
- Fahrlehrer können freigewordene Zeiten veröffentlichen
- Aktive Fahrschüler können freigegebene Termine vom Fahrlehrer buchen
- Integrierter Kalender für eine Darstellung aller Termine

Fahrzeugverwaltung

- Fahrlehrer können neue Fahrzeuge für ihre Fahrschule anlegen
- Verwaltung der Fahrzeugverfügbarkeit zur Vermeidung von Doppelbuchungen

Datenverwaltung und Sicherheit

- Speicherung und Verwaltung aller Nutzerdaten in Back4app
- Sichere Authentifizierung (E-Mail/Passwort)
- Session Token welcher die aktuelle Sitzung der eingeloggten Person sicherstellt

4.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen legen die Qualitätsmerkmale der Fahrschul-Manager App fest. Sie umfassen Aspekte wie Leistung, Sicherheit, Benutzerfreundlichkeit und Skalierbarkeit, die sicherstellen, dass die App zuverlässig und effizient genutzt werden kann. Wir stellen uns selbst folgende Nicht-Funktionale Anforderungen an die App:

Leistung und Skalierbarkeit

- Die App soll auch bei hoher Anzahl an Nutzern flüssig und performant laufen
- Schnelle Ladezeiten und eine effiziente Datenabfrage über Back4App
- Skalierbare Backend-Struktur

Usability und Benutzerfreundlichkeit

- Intuitive und einfache Bedienung für Fahrlehrer und Fahrschüler
- Übersichtliches UI-Design mit klar strukturierten Menüs und Funktionen

Sicherheit und Datenschutz

- Sichere Authentifizierung
- Verschlüsselte Datenübertragung zwischen App und Backend

Verfügbarkeit und Zuverlässigkeit

- Hohe Verfügbarkeit der App mit einer minimalen Ausfallzeit des Backend
- Daten bleiben auch bei einer kurzzeitigen Unterbrechung der Internetverbindung erhalten

Wartung und Erweiterbarkeit

- Modulare Code-Struktur für eine einfach Wartung und zukünftige Erweiterungen
- Dokumentation der Code Struktur

4.3 Technische Anforderungen

Zur Umsetzung dieser Idee haben wir uns entschieden, das Framework Flutter mit der Programmiersprache Dart zu verwenden. Ein wesentlicher Grund dafür ist, dass Herr Paleyron bereits Erfahrung mit Flutter hat und dieses Wissen gezielt in unser Abschlussprojekt einbringen konnte. Zudem ist Flutter speziell für die App-Entwicklung konzipiert und ermöglicht eine schnelle Bereitstellung auf mobilen Endgeräten.

Als Entwicklungsumgebung nutzten wir VSCode in Kombination mit dem Emulator von Android Studio, um die App unter realistischen Bedingungen zu testen. Für das Backend und die Datenbank fiel die Wahl auf Back4App (Backend-as-a-Service), da wir dort sowohl die Datenbankverwaltung als auch die Benutzerverwaltung für Fahrlehrer und Fahrschüler abbilden können.

Um eine reibungslose und performante App zu gewährleisten, setzen wir zudem auf effiziente State-Management-Techniken, insbesondere Bloc, um eine saubere Trennung von UI- und Anwendungslogik sicherzustellen. Dies verbessert die Wartbarkeit der App und sorgt für eine optimierte Performance, insbesondere bei komplexen Datenstrukturen und Zustandsänderungen.

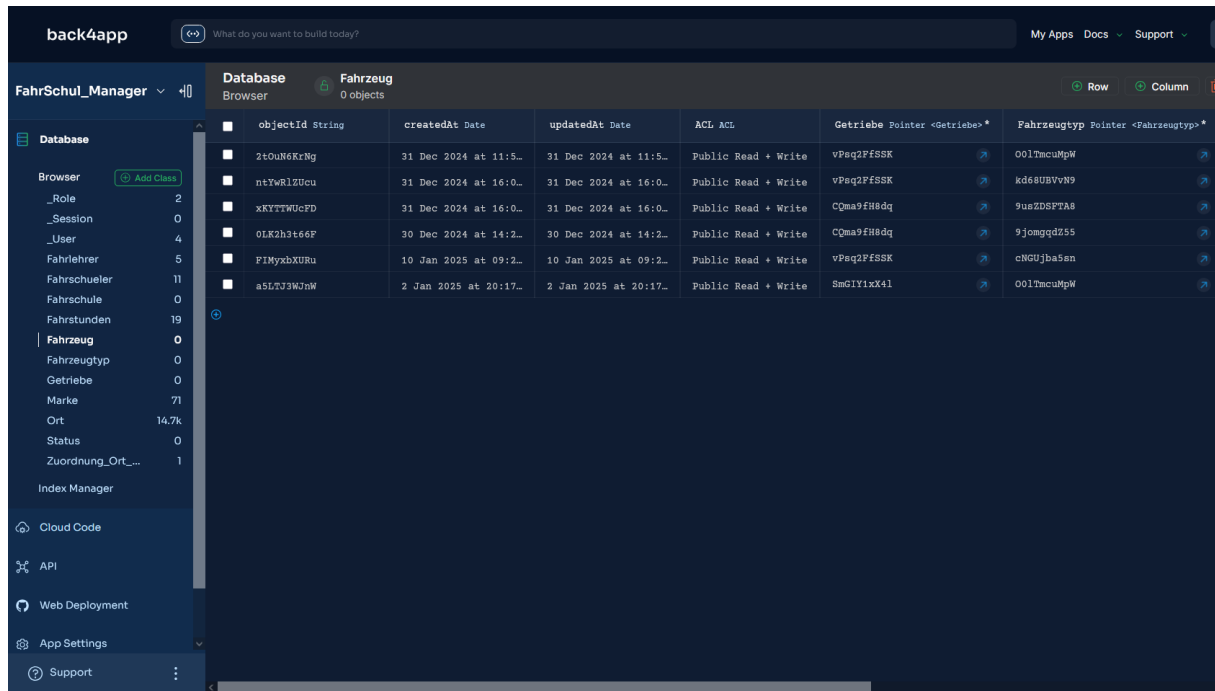


Abbildung 2:Back4app

4.4 Entity-Relationship-Modell

Während unserer Weiterbildung haben wir gelernt, welche zentrale Rolle ein ERM für die weiterführende Programmierung spielt. Im nächsten Abschnitt eine kurze Erklärung warum.

Ein Entity-Relationship-Modell (ERM) ist ein essenzielles Werkzeug für die Datenmodellierung und spielt eine zentrale Rolle in der Entwicklung unserer Fahrschul-Manager App. Es hilft dabei, die Struktur der Datenbank zu definieren und die Beziehungen zwischen den verschiedenen Entitäten (z. B. Fahrlehrer, Fahrschüler, Fahrstunden, Fahrzeuge) klar darzustellen.

Zu Beginn haben wir uns Gedanken gemacht welche Entitäten wir benötigen und welche Informationen wir in der App abbilden wollen. Zugehörig dazu noch die passenden Beziehungen und Kardinalitäten.

Aus unseren Überlegungen entstanden folgende Entitäten (Später in der Datenbank als Tabellen umgesetzt):

- Fahrschule
- Ort
- Fahrzeug
- Fahrschüler
- Fahrlehrer
- Fahrstunden
- Fahrzeugtyp
- Getriebe
- Marke

Unser erster ERM-Entwurf:

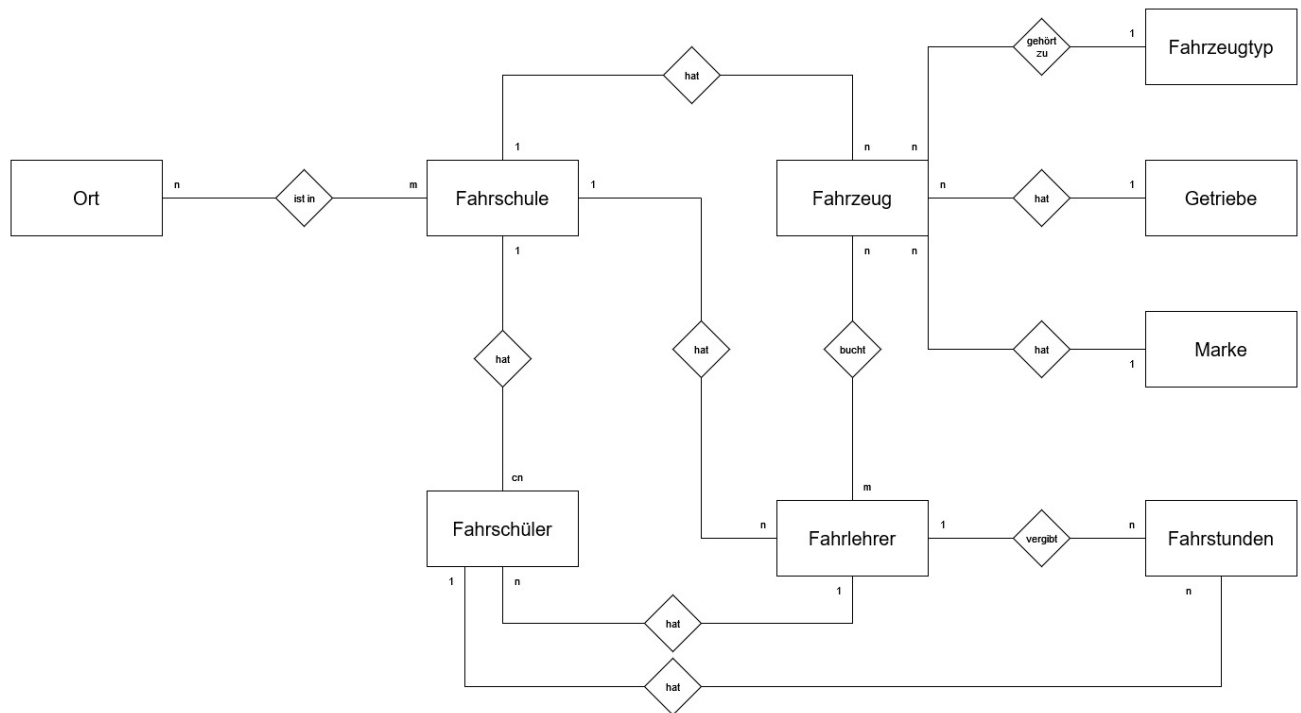


Abbildung 3: ERM-Entwurf 1

Im weiteren Projektverlauf und Umsetzung ist uns allerdings aufgefallen das manche Dinge nicht passend sind oder im ERM sogar fehlen und noch abgebildet werden sollten.

Als erstes ist aufgefallen das Kardinalitäten angepasst werden müssen. In der Bestehenden Logik wäre es nicht möglich gewesen einen Fahrschüler im System zu haben welcher keine Fahrstunden zugewiesen hat (aufgrund der 1:n Beziehung). Neu angelegte Fahrschüler z.B. haben aber noch keine eingetragenen Fahrstunden. Aus diesem Grund musste die 1:n Beziehung in eine 1:cn Beziehung umgewandelt werden damit ein Fahrschüler auch null Fahrstunden haben kann.

Ein vergleichbares Beispiel wäre die Beziehung zwischen der Fahrschule und den Fahrschülern. Hier wird auch eine 1:cn Beziehung angewendet da eine Fahrschule auch existieren darf ohne zugewiesene Fahrschüler.

Als nächstes schien es sinnvoller zu sein das die Fahrzeuge nicht direkt mit dem Fahrlehrer verbunden sind, sondern die Fahrzeuge über die entsprechenden Fahrstunden direkt zugewiesen werden in denen sie auch gebucht wurden.

Abschließend wurden noch zwei weitere Entitäten hinzugefügt. Der Status, welcher einem Fahrschüler zugewiesen ist, ob er Aktiv, Passiv oder Inaktiv ist (Erläuterung im folgenden Absatz). Und der User, welcher nötig ist um darzustellen ob ein User in die Kategorie Fahrlehrer oder Fahrschüler fällt.

Aktiv: Fahrschüler ist einem Fahrlehrer zugewiesen und er nimmt Fahrstunden

Passiv: Fahrschüler ist einem Fahrlehrer zugewiesen, nimmt aber noch keine Fahrstunden

Nicht zugewiesen: Hat keine Fahrstunden, sowie keinen Fahrlehrer

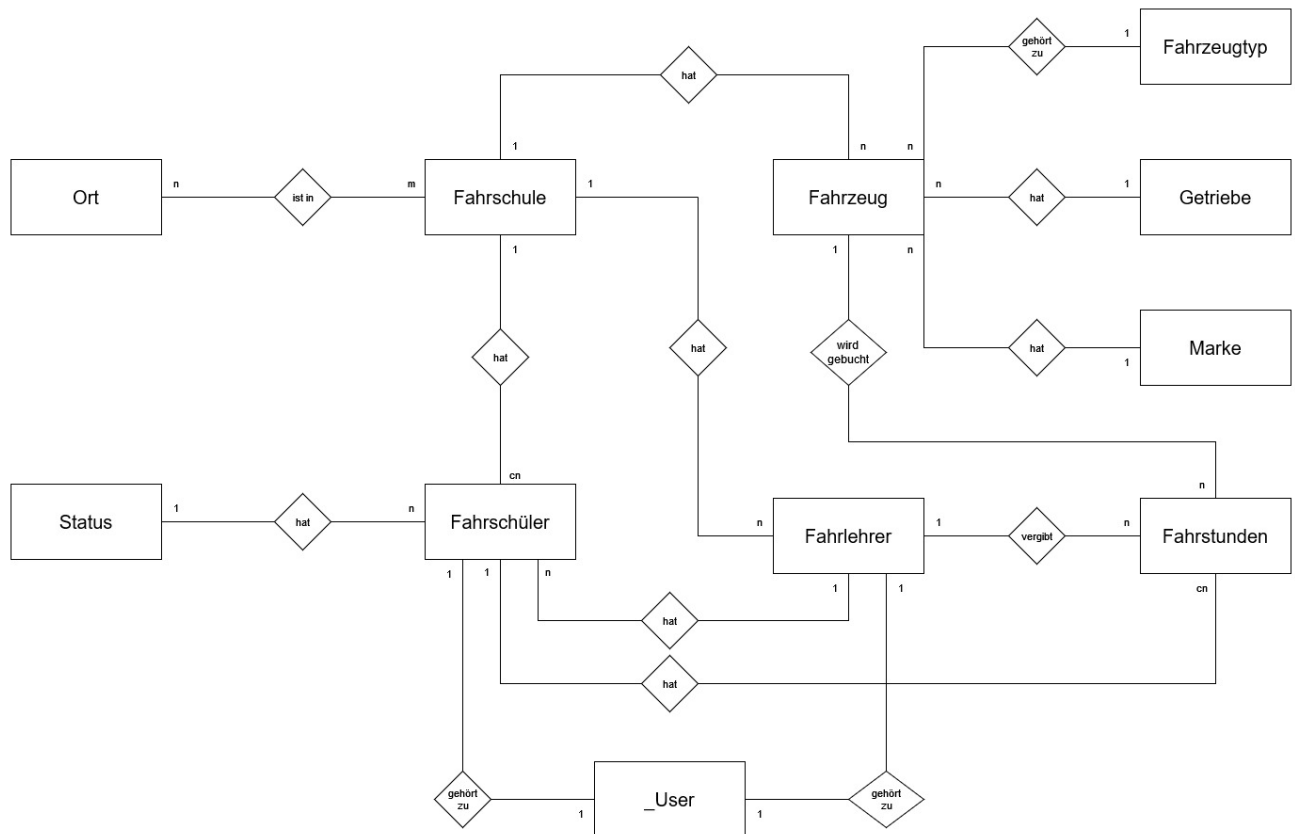


Abbildung 4:ERM-Final

4.5 Logisches Modell

Nachdem das Entity-Relationship-Modell (ERM) die grundlegenden Entitäten und deren Beziehungen definiert hat, muss es in eine technisch umsetzbare Form überführt werden. Dafür wird aus dem konzeptionellen ERM ein logisches Modell erstellt, das die Strukturen der Datenbank detaillierter beschreibt. Dieses Modell berücksichtigt bereits konkrete Datenbankaspekte wie Primär- und Fremdschlüssel, Datentypen und Normalisierung, um eine effiziente Speicherung und Verarbeitung der Daten sicherzustellen.

Auch hier ist in gleichem Maße wie das ERM-Modell das Logische Modell über die Zeit gewachsen und wurde angepasst aufgrund neu gewonnener Kenntnisse im Verlaufe des Projektes.

Aufgrund dessen, dass es sich bei Back4app um eine Objektorientierte Datenbank handelt wurde als Primärschlüssel die objectID der jeweiligen Tabelle verwendet. Diese wird automatisch bei einem neuen Datensatz für den neuen Datensatz generiert und ist einmalig für diesen.

Hier ein Auszug aus einem der ersten Entwürfe dargestellt mit neuen Zuordnungstabellen um die n:m Beziehungen aufzulösen, unseren Attributen zu den Entitäten, sowie die Primär und Fremdschlüssel um die Relation zwischen den Tabellen darzustellen.

Fahrschul-Manager Dokumentation

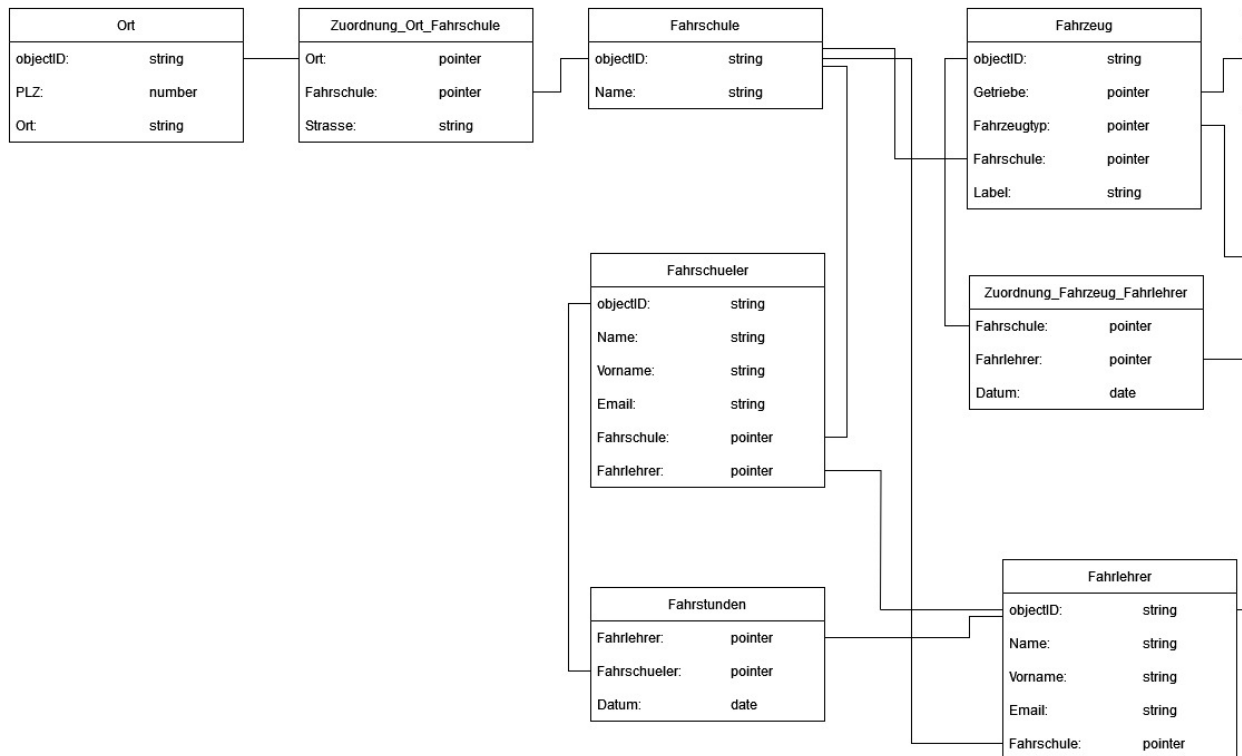


Abbildung 5: Logisches Modell-Auszug-Früher Entwurf

Nach und nach wurde das Logische Modell erweitert und ausgebessert. Es wurden die Fehlenden Kardinalitäten ergänzt, die von Back4app standartmäßig angelegten Attribute wie z.B. createdAt, updatedAt, usw. hinzugefügt sowie immer wieder neue Attribute welche sich erst im Laufe der aktiven Entwicklung der App herauskristallisierten.

Zum Verständnis, die Primärschlüssel der jeweiligen Tabellen (objectID) zeigen immer auf einen Datensatz vom Typ Pointer, welcher in der Datenbank so eingestellt ist das er zu seiner jeweiligen Tabelle verweist. Das liegt an unserer Datenbank welche Objektorientiert ist. Auf diese Weise wird die Primär und Fremdschlüssel Beziehung darstellt.

Spezielle Attribute im folgenden Abschnitt genauer erklärt:

In der Tabelle _User wurde das Attribut firstSession als Boolean hinzugefügt (siehe Abb. 6). Bei der Registrierung eines neuen Benutzers wird ein Passwort generiert, zeitgleich wird das Attribut firstSession auf true gesetzt. Dadurch wird der Benutzer beim ersten Login in der Anwendung aufgefordert, sein Passwort zu ändern. Nach erfolgreicher Änderung des Passworts wird der Wert auf false aktualisiert.

In der Tabelle Fahrstunden wurden die Attribute UpdatedGesamtStd und Freigeben als Booleans hinzugefügt (siehe Abb. 6). Das Attribut UpdatedGesamtStd dient zur automatischen Berechnung der gesamten gefahrenen Fahrstunden eines Fahrschülers.

Hierfür wurde in Back4App ein Scheduled Job (automatisch ausgeführte Aufgabe) eingerichtet, der in regelmäßigen Abständen nach Datensätzen in der Tabelle Fahrstunden sucht, bei denen UpdatedGesamtStd auf false gesetzt ist. Anschließend wird über die gefundenen Einträge iteriert, die gefahrenen Fahrstunden jeweils berechnet und den Gesamtfahrstunden des Fahrschülers hinzugefügt.

Mit dem Attribut Freigeben kann ein Fahrlehrer eine Fahrstunde für seine aktiven Fahrschüler freigeben. Die Fahrschüler können diese Fahrstunde anschließend auf ihrer Homepage einsehen und

buchen. Sobald die Fahrstunde erfolgreich von einem Fahrschüler gebucht wurde, wird der Wert von Freigeben auf false gesetzt, wodurch die Fahrstunde nicht mehr öffentlich sichtbar ist.

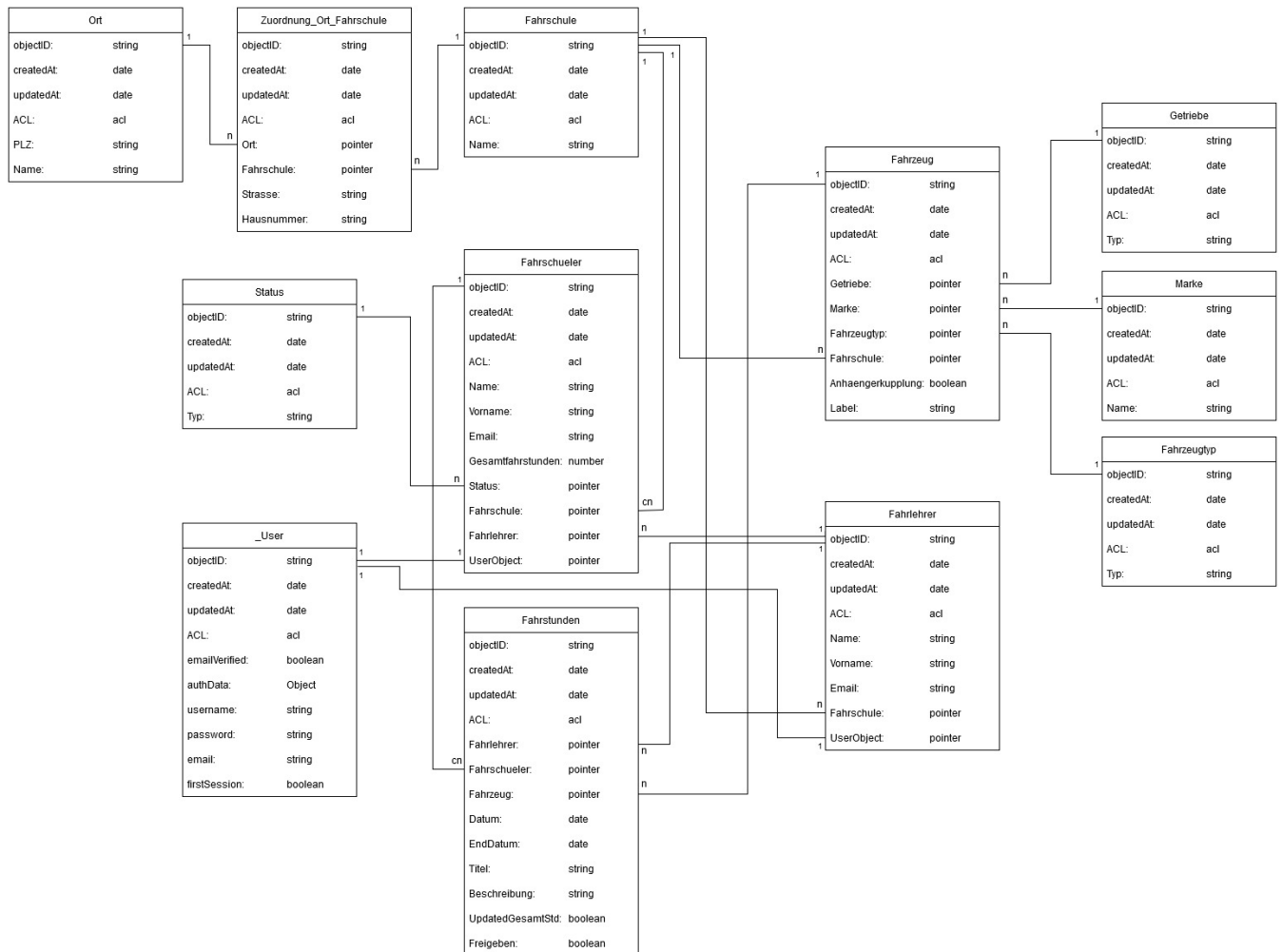


Abbildung 6: Logisches Modell-Final

4.6 Designschema

Das Designschema unserer Fahrschul-Manager App bietet eine erste visuelle Orientierung für die Benutzeroberfläche und die Struktur der App. Es zeigt die grundlegende Anordnung der einzelnen Seiten sowie die wichtigsten UI-Elemente, die für eine intuitive und benutzerfreundliche Navigation sorgen. Des Weiteren ist es bei der Programmierung sehr hilfreich, da anhand dieser Schemas das UI gestaltet werden kann.

Bei der Gestaltung wurde besonderen Wert auf eine klare und übersichtliche Benutzerführung gelegt, sodass sowohl Fahrerlehrer als auch Fahrschüler schnell und effizient auf die benötigten Funktionen zugreifen können. Die einzelnen Seiten werden in einer konsistenten Designsprache umgesetzt, die sich an modernen UI/UX-Prinzipien orientiert und eine einfache Bedienung auf mobilen Endgeräten gewährleistet.

Zur weiteren besseren Orientierung wurde die Seite in drei Bereiche aufgeteilt. Top, Mid und Bottom. Hier zwei Seitenbeispiele von den Insgesamt 16 Designschemas für die jeweiligen Seiten:

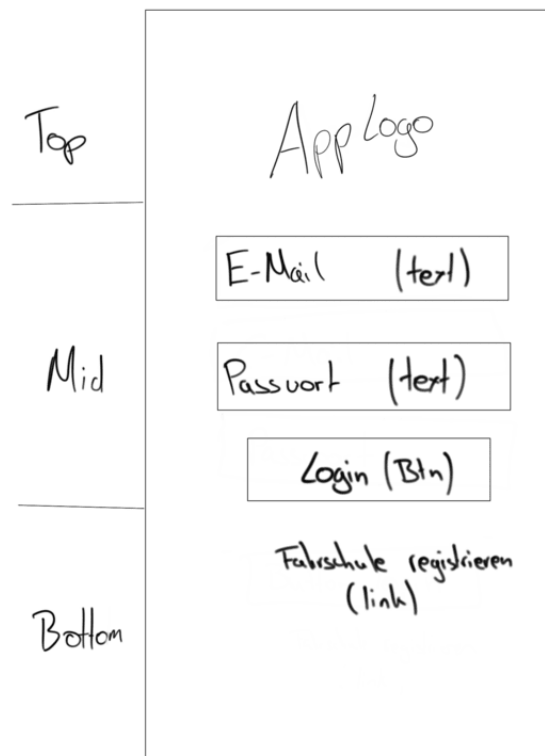


Abbildung 7: Login Seite

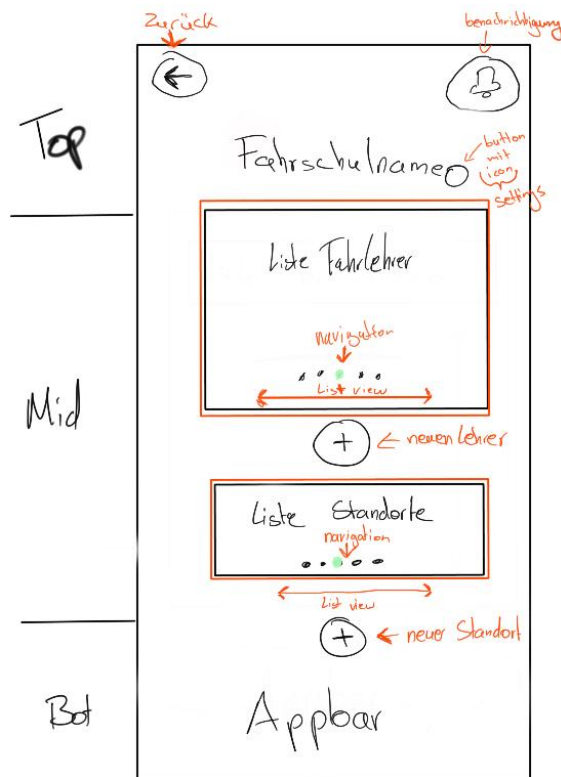


Abbildung 8: Fahrschule Seite-Fahrlehrer

4.7 Ablaufdiagramm

Das Ablaufdiagramm spielt eine wichtige Rolle in der Entwicklungsphase der App, da es die Navigation und Struktur visuell darstellt. Es zeigt auf einfache Weise, wie sich Benutzer durch die App bewegen und welche Verbindungen zwischen den einzelnen Seiten bestehen.

Durch ein Ablaufdiagramm können logische Fehler oder unnötig komplizierte Navigationswege frühzeitig erkannt und optimiert werden. Dies verbessert die Benutzerfreundlichkeit, da klare und intuitive Wege geschaffen werden. Zudem erleichtert es die Zusammenarbeit im Team, da alle Beteiligten eine gemeinsame visuelle Referenz für die Funktionsweise der App haben.

Darüber hinaus dient das Ablaufdiagramm als Dokumentation, die zukünftigen Entwicklern hilft, sich schneller in die Struktur der Anwendung einzuarbeiten. Sowie für uns, um zu einem späteren Zeitpunkt den Ablauf der App erneut nachzuvollziehen.

Für unsere App haben wir jeweils ein Ablaufdiagramm welches für einen Fahrschüler gilt erstellt, sowie eines für den Fahrlehrer.

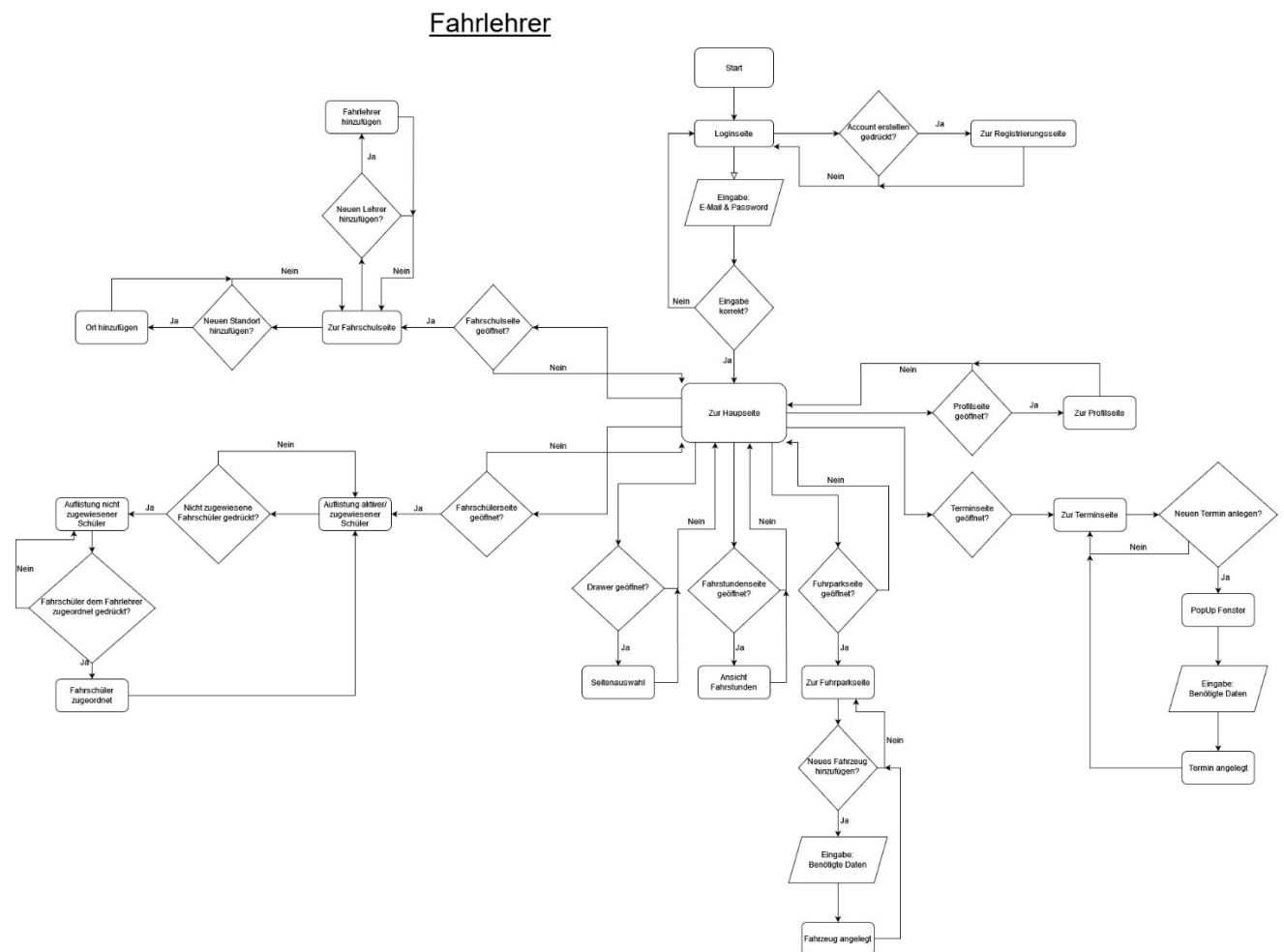


Abbildung 9: Ablaufdiagramm Fahrlehrer

4.8 Zeitplan

Unseren vorgegebenen Zeitplan konnten wir gut einhalten und hatten an dieser Stelle keine Probleme. In den Meilensteinsitzungen haben wir jeweils festgestellt an welcher Stelle im Projekt wir uns befinden, und wie viele Stunden jeder Einzelne bereits investiert hat.

Hier ein Stundennachweis zur ersten Meilensteinsitzung nach ca. 3 Monaten Projektarbeit:

Chris

Tätigkeit	Zeit
Projekt aufsetzen	1 Std
Backend erstellen/anbinden	4 Std
Einlesen Back4App	1 Std
ERM-Diagramm	5 Std
Logisches Modell	8 Std
Designschema	8 Std
Backend Funktionen programmieren	30 Std
Benutzeroberfläche programmieren	8 Std
Dokumentation	1,5 Std
Gesamt:	66,5 Std

Luis

Tätigkeit	Zeit
Projekt anlegen	1,5 Std
Einlesen Back4App	2 Std
Datenbank erstellen	1 Std
ERM-Diagramm	6 Std
Logisches Modell	9 Std
Datenbank befüllen	2 Std
Ablaufdiagramm	4 Std
Benutzeroberfläche programmieren	20 Std
Berechtigungen einlesen	1 Std
Gesamt:	46,5 Std

Abbildung 10: Stundennachweis

Wir lagen also gut in der Zeit und konnten bis dahin einige unserer vorher festgelegten Teilziele erreichen.

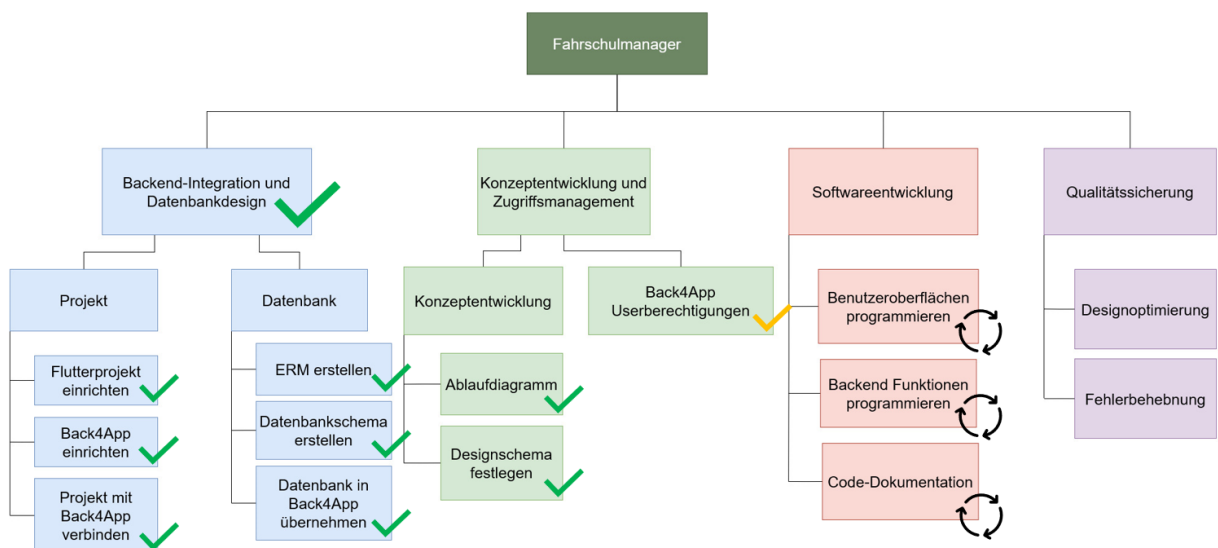


Abbildung 11: Erreichte Zwischenziele

Der weitere Projektverlauf verlief zeitlich ebenfalls wie geplant, und wir konnten unsere Projektdurchführung erfolgreich vor Beginn der Technikerbörse beenden.

5 Architektur und Design

5.1 Systemarchitektur

Die Fahrschul-Manager App basiert auf einer Client-Server-Architektur, bei der das Flutter-Frontend mit einem Backend-as-a-Service (Back4App) kommuniziert. Die Architektur setzt sich aus folgenden Hauptkomponenten zusammen:

Frontend (Client-Seite)

- Entwickelt mit Flutter und der Programmiersprache Dart
- Verantwortlich für die Benutzeroberfläche und die Interaktion mit dem Nutzer
- Implementiert das State-Management (Bloc), um die App Struktur effizient und reaktions-schnell zu halten

Backend (Datenverarbeitung und Logik)

- Wird über Back4App (Parse-Server) bereitgestellt
- Verwaltet die Datenbank, in der Benutzer, Fahrstunden und Fahrzeuge gespeichert werden
- Beinhaltet Cloud-Funktionen zur Verarbeitung spezifischer Aufgaben
- Beinhaltet Scheduled Jobs, z.B. für unser UpdatedGesamtStd Attribut

Datenbank

- Gehostet in Back4App
- Speichert alle relevanten Daten, z.B. Benutzerkonten, Fahrstunden, Fahrzeuge und Buchungen

5.2 Datenbankdesign

Das Datenbankdesign von Back4App basiert auf einer flexiblen und skalierbaren Architektur, die auf der Open-Source-Plattform Parse aufbaut. Back4App verwendet ein objektorientiertes Datenmodell, das relationale Konzepte mit NoSQL-ähnlicher Flexibilität kombiniert. Daten werden in sogenannten Klassen organisiert, die Tabellen in einer traditionellen relationalen Datenbank entsprechen. Jede Klasse enthält Objekte (äquivalent zu Datensätzen oder Zeilen), die aus Feldern (Spalten) bestehen. Diese Felder können verschiedene Datentypen wie Strings, Zahlen, Booleans, Arrays, GeoPoints, Dateien oder Verweise auf andere Objekte speichern.

Jedes Objekt in einer Klasse hat eine eindeutige objectId, die automatisch generiert wird und als Primärschlüssel dient.

Beziehungen werden über Pointer geregelt, ein Feld das auf ein bestimmtes Objekt in einer anderen Klasse verweist.

Back4App stellt vordefinierte Klassen wie `_User` (für Benutzerverwaltung), `_Role` (für Rollen und Berechtigungen) und `_Session` (um Sitzungsinformationen für authentifizierte Benutzer zu speichern) bereit.

FahrSchul_Manager		Database Browser		
		_Role (2 objects)		
		objectId String	createdAt Date	updatedAt Date
		dJFlcBclhZ	31 Oct 2024 at 18:4...	10 Jan 2025 at 09:1...
		pk8wcViEpt	30 Oct 2024 at 20:5...	30 Dec 2024 at 10:4...

Abbildung 12: Back4App Klassen

5.3 Benutzeroberfläche

Die Programmierung der Benutzeroberflächen begann mit der Login und Registrierungsseite. Wir haben uns für ein schlichtes und elegantes Design mit der Farbgebung Grün entschieden. Uns war es wichtig, den Benutzer nicht mit zu vielen Informationen oder unnötigen Grafiken zu stören. Sondern eine intuitive Führung durch die App und Ihre Funktionen zu ermöglichen.

Die Login Seite ist eine Stateful-Widget-Klasse die eine Benutzeranmeldeseite bereitstellt. Sie ermöglicht die Eingabe von E-Mail und Passwort, validiert diese asynchron und leitet den Benutzer je nach Status weiter (z. B. erstmaliger Login oder reguläre Anmeldung).

In Flutter wird mit zwei Arten von Widgets gearbeitet. Stateless oder Statefull. Stateless Widgets sind statisch und einfach, während Stateful Widgets dynamisch sind und Änderungen im Laufzeitverhalten ermöglichen. Es sind grundlegende Baueinheiten für die Benutzeroberfläche (UI). Es ist ein unveränderliches (immutable) Objekt, das einen Teil der UI beschreibt, wie z. B. einen Button, Text, ein Bild oder ein Layout.

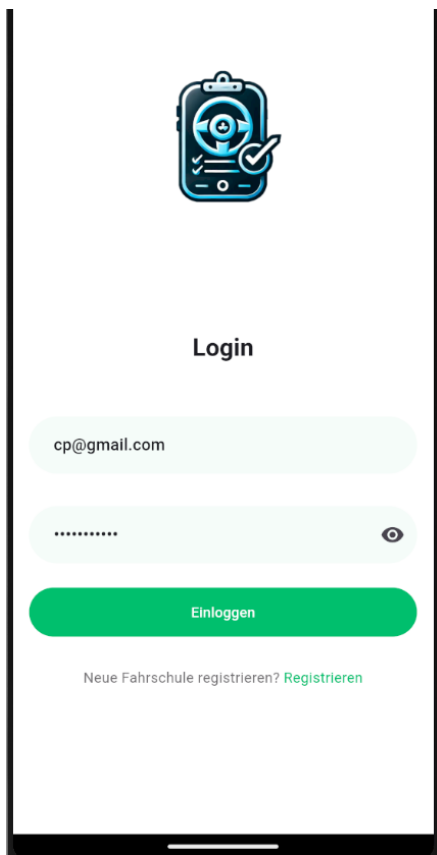


Abbildung 14: Login Seite

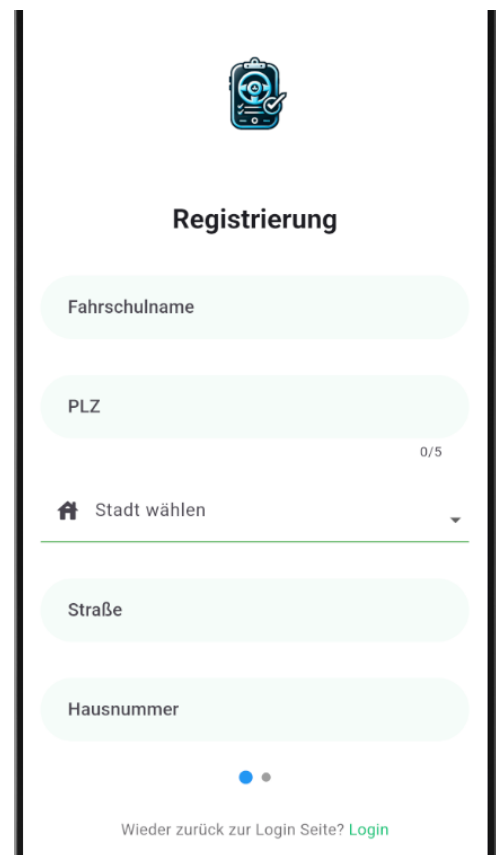
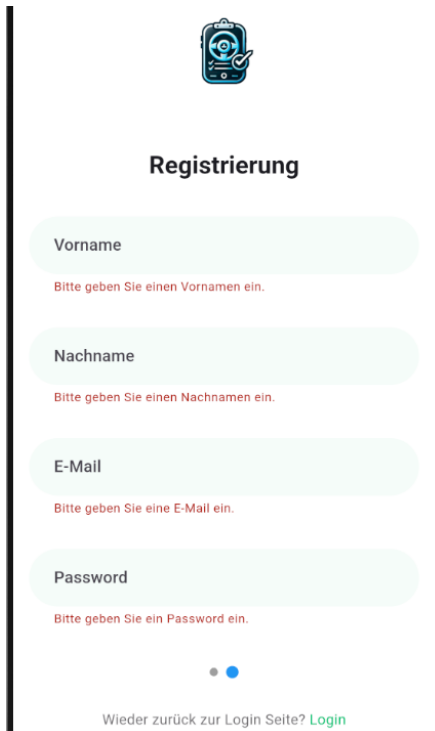


Abbildung 13: Registrierung Seite 1

Bei der Eingabe der Postleitzahl wird nach jeder Ziffer im Hintergrund die Datenbank geprüft, sodass im Dropdown-Feld „Stadt wählen“ nur passende Städte angezeigt werden – für eine schnellere und komfortablere Bedienung. Auf der zweiten Seite der Registrierung werden noch weitere Daten eingegeben. Wenn alles ausgefüllt wurde und das festgelegte Regex eingehalten wurde (falls nicht wird der Benutzer auf die entsprechenden Felder hingewiesen) kann eine neue Fahrschule Registriert werden.

Regex findet in unserer App überall da Verwendung wo wir es für sinnvoll erachtet haben.



Registrierung

Vorname
Bitte geben Sie einen Vornamen ein.

Nachname
Bitte geben Sie einen Nachnamen ein.

E-Mail
Bitte geben Sie eine E-Mail ein.

Password
Bitte geben Sie ein Password ein.

Wieder zurück zur Login Seite? [Login](#)

Abbildung 15: Regex Beispiel

Ein essenzielles Feature zur Bedienung der App und zum einfachen Navigieren über die Benutzeroberfläche ist die Navigationsleiste. Mit ihr kann von jedem Punkt aus in der App schnell und komfortabel die gewünschte Seite erreicht werden. Zu Beginn war die Idee die Navigation über einen Drawer im oberen linken Teil der App zu implementieren. Die Idee wurde allerdings verworfen und mit einer Leiste im unteren Bildschirmrand ersetzt. Da dadurch die Navigation schneller und übersichtlicher möglich ist.

Folgende Seiten lassen sich darüber direkt aufrufen (aus Sicht eines Fahrlehrers):

- Fahrschülerliste
- Kalender zur Terminverwaltung
- Homepage
- Fahrschulseite
- Profil

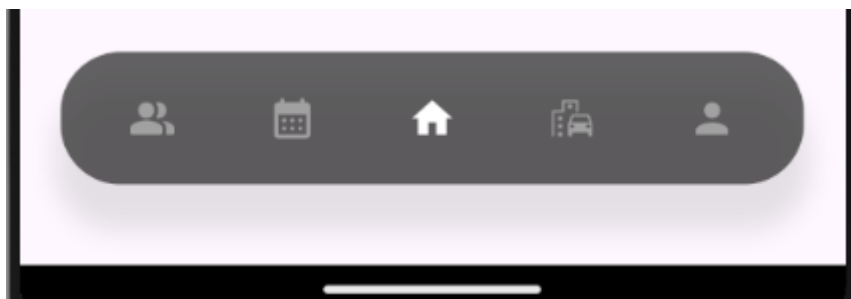


Abbildung 16: Navigationsleiste

Im Folgenden werden die einzelnen Seiten beschrieben. Im Rahmen dieser Dokumentation werden nur die Seiten aus Sicht eines Fahrlehrers beschrieben.

Fahrschülerliste

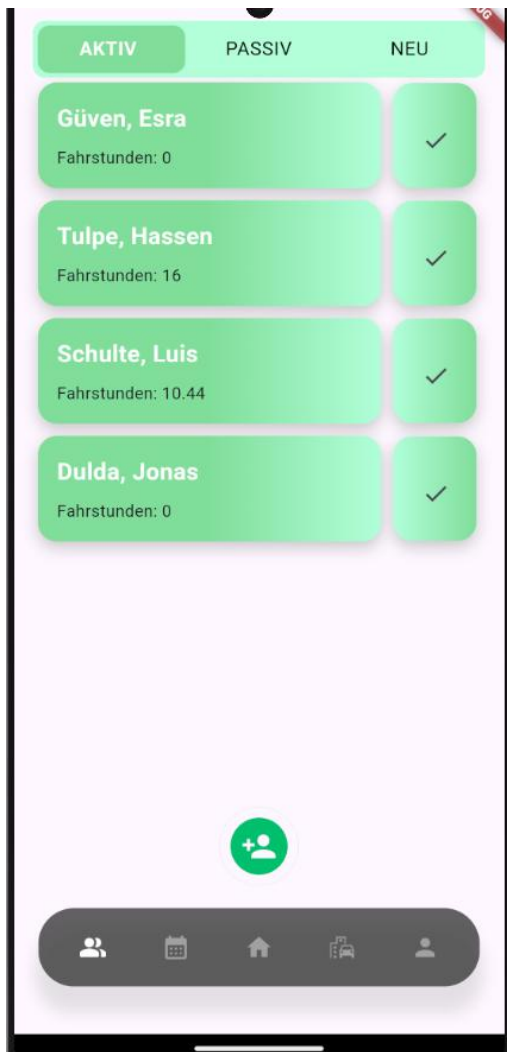


Abbildung 17: Fahrschülerliste Seite

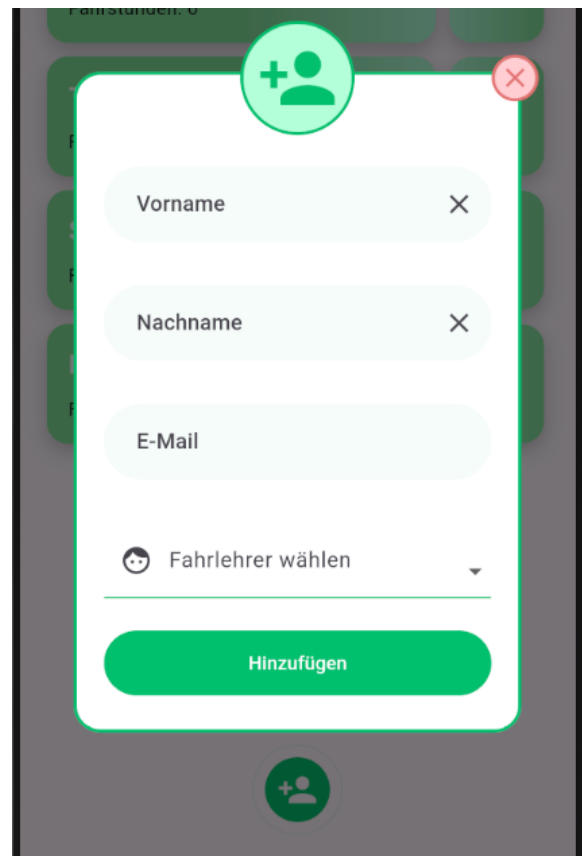


Abbildung 18: Fahrschüler hinzufügen

Auf dieser Seite werden die Aktiven, Passiven und nicht zugewiesenen Fahrschüler angezeigt.

Aktive Fahrschüler nehmen Fahrstunden, Passive Fahrschüler können Aktiv gesetzt werden. Neue Fahrschüler welche nicht direkt einem Fahrlehrer beim anlegen zugeordnet wurden werden in „Neu“ gelistet.

Kalender

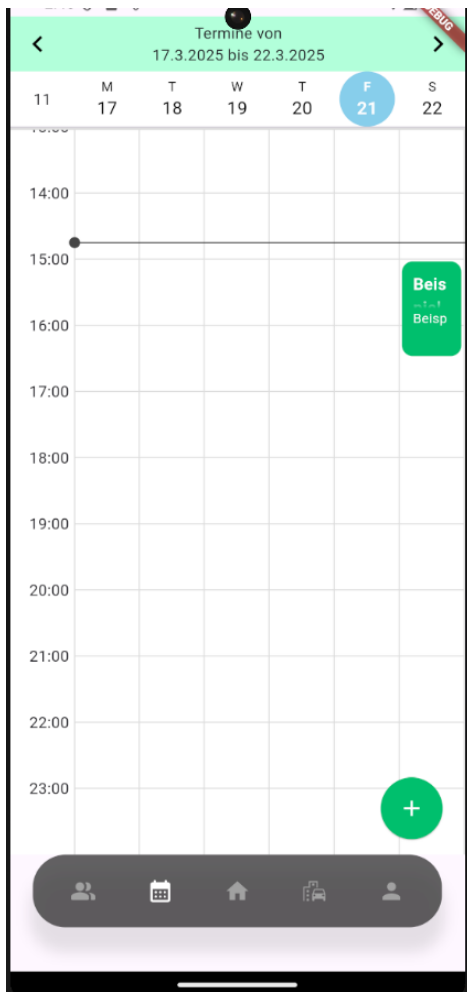


Abbildung 19: Kalender Seite

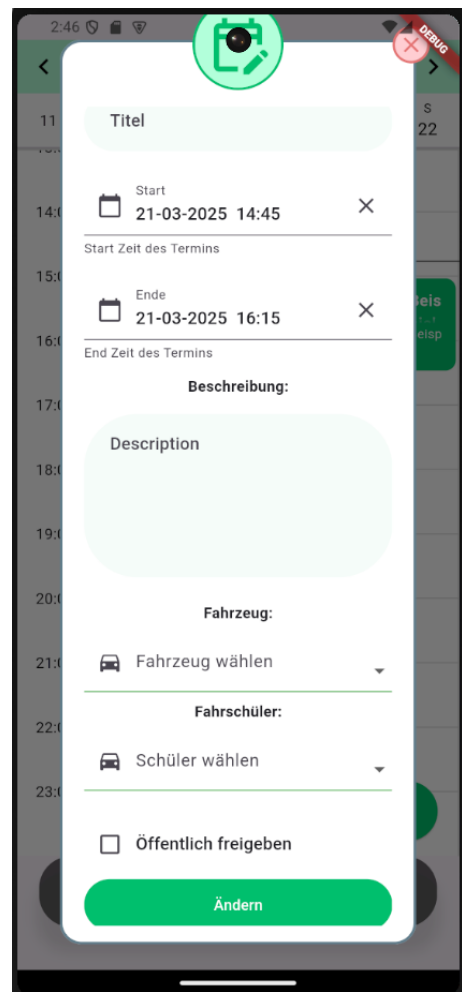


Abbildung 20: Termin hinzufügen

Hier werden die angelegten Termine dargestellt und können durch tippen auf den Termin auch nachträglich bearbeitet oder gelöscht werden. Beim hinzufügen eines Termins müssen alle benötigten Felder ausgefüllt werden. Falls kein Fahrschüler eingetragen wird, besteht die Möglichkeit diesen Öffentlich freizugeben, sodass die Fahrschüler diesen bei sich in der App angezeigt bekommen um sich den Termin zu reservieren.

Homepage

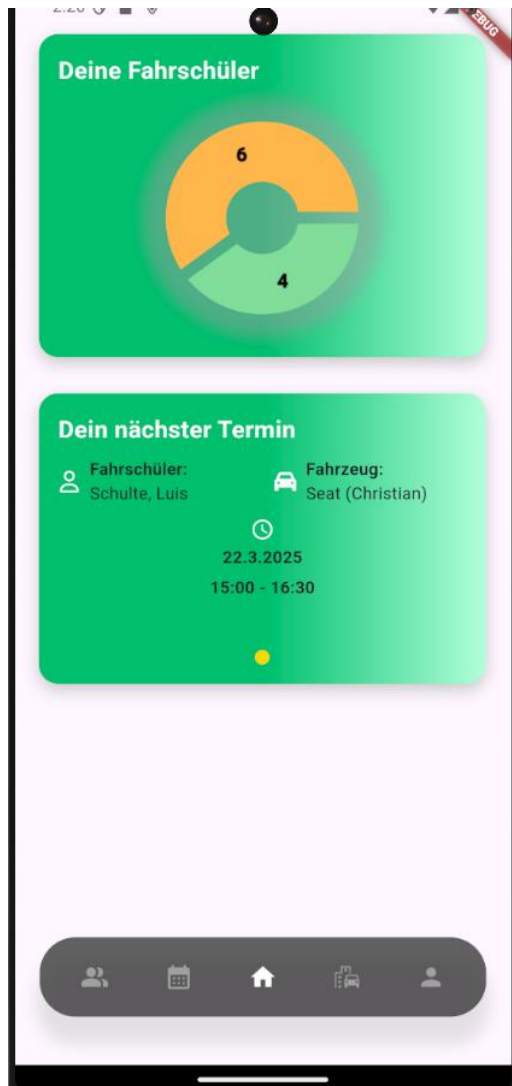


Abbildung 21: Homepage Seite

In der Homepage des Fahrlehrers werden anhand eines Diagramms die Anzahl der ihm zugewiesenen Aktiven und Passiven Fahrschüler angezeigt.

Darunter befinden sich die nächsten Fahrstunden für den Fahrlehrer mit dem Namen des Fahrschülers, dem Fahrzeug mit dem die Fahrstunde abgehalten wird, und das Datum mit der Uhrzeit für den Termin. Bei mehreren Terminen können diese mit wischen nach links oder rechts angezeigt werden.

Fahrschulseite

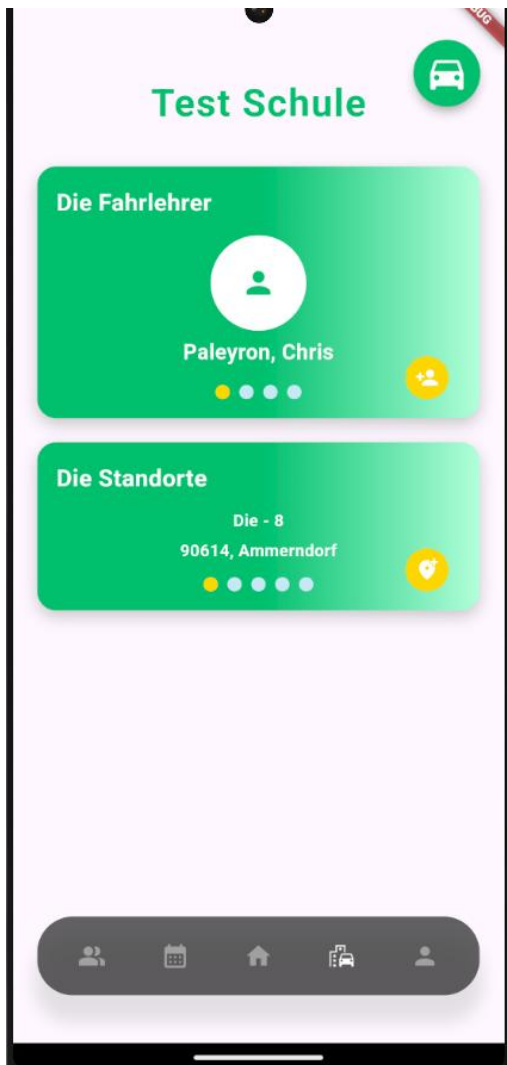


Abbildung 22: Fahrschulseite

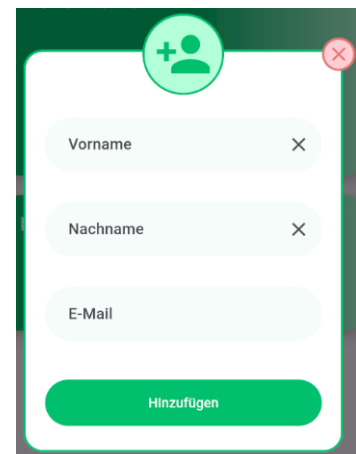


Abbildung 23: Fahrlehrer hinzufügen

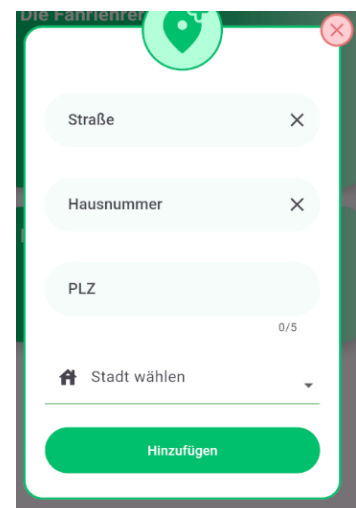


Abbildung 24: Standort hinzufügen

Hier kann die Fahrschule vom Besitzer verwaltet werden. Es bestehen die Möglichkeiten neue Fahrlehrer, sowie neue Standorte hinzuzufügen.

Des Weiteren kommt man von der Fahrschulseite über den Button oben rechts auf die Fuhrpark Seite der Fahrschule. Die Idee dieser und dessen Funktionen werden im nächsten Abschnitt beschrieben.

Fuhrpark

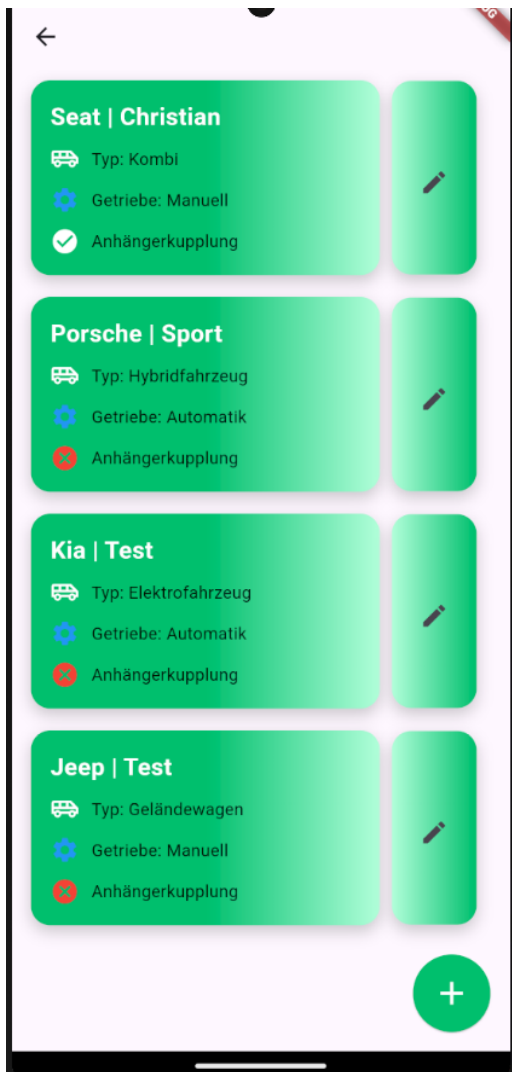


Abbildung 25: Fuhrpark Seite

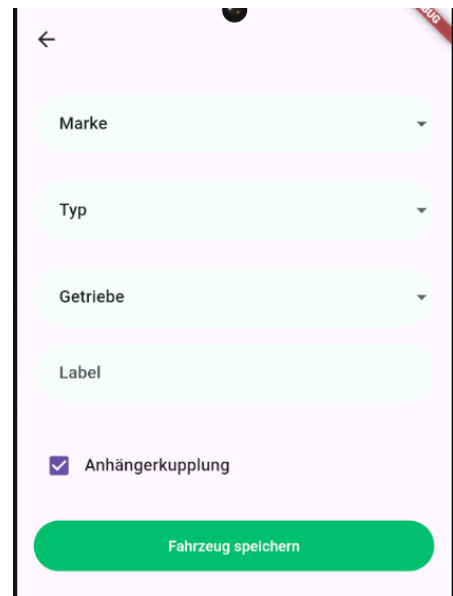


Abbildung 26: Fahrzeug hinzufügen

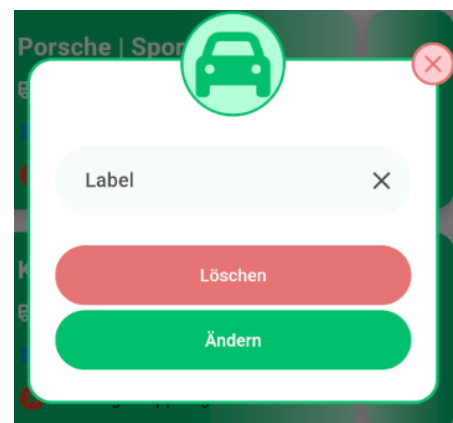


Abbildung 27: Label bearbeiten

Die Idee der Fuhrpark Seite ist es, eine Übersicht über alle Fahrzeuge der Fahrschule zu geben. Die Informationen beinhalten die Marke, den Typ (z.B. Kombi), die Getriebeart und die Info ob das Fahrzeug über eine Anhängerkupplung verfügt. Diese Überprüfung wird mithilfe eines Boolean gelöst beim Hinzufügen eines neuen Fahrzeugs.

Neben der Marke befindet sich noch das Label welches einem Auto individuell vergeben wurde, und auch noch im Nachhinein bei Bedarf wieder geändert werden kann.

Ziel ist es, die Anpassung von Fahrzeugbeschreibungen an die Unternehmensstruktur zu vereinfachen. So können beispielsweise in einer Fahrschule Autos individuell Personen zugeordnet und bei Personalwechsel diese Zuweisungen schnell und unkompliziert geändert werden.

Profil

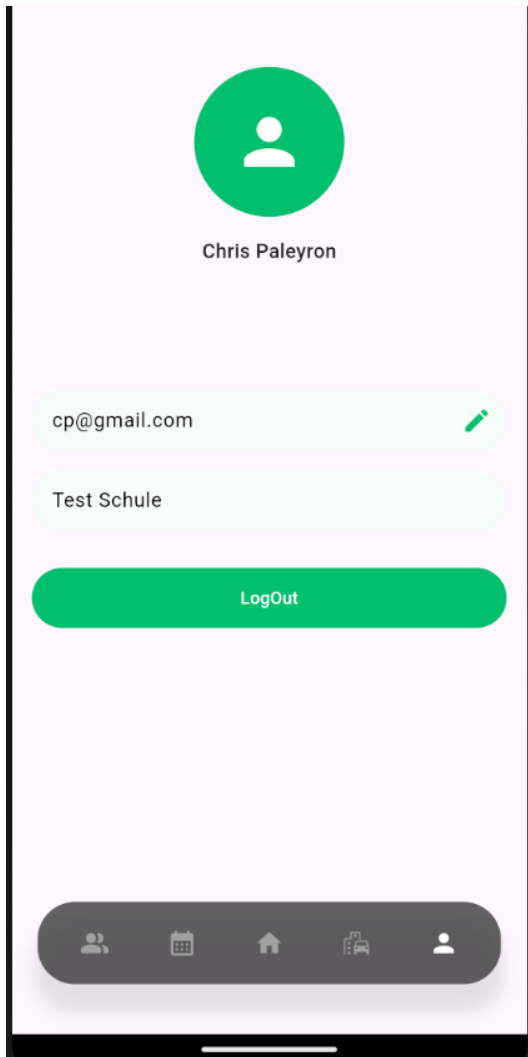


Abbildung 28: Profil Seite

Auf der Profil Seite stehen die Individuellen Daten zum Nutzer. Name, E-Mail und die zugeordnete Fahrschule.

Über den LogOut Button wird der Nutzer ausgeloggt und wieder auf die Login Seite geführt.

6 Implementierung

6.1 Verwendete Technologien

Die Entwicklung der Fahrschul-Management-Anwendung basiert auf einem durchdachten Technologie-Stack, der moderne Flutter-Features mit leistungsstarken Bibliotheken kombiniert, um eine effiziente, skalierbare und benutzerfreundliche App zu schaffen – angefangen mit der asynchronen Programmierung mittels „async/await“ als Schlüsselkomponente für reaktive Prozesse.

Asynchrone Programmierung

Dabei pausiert await die Ausführung einer Funktion, bis ein Future abgeschlossen ist, etwa bei Netzwerkanfragen, und gibt erst dann das Ergebnis zurück, was eine saubere und lesbare Handhabung

asynchroner Abläufe ermöglicht. Asynchrone Operationen sind essenziell für Netzwerkanfragen, Datenbankzugriffe oder andere zeitintensive Prozesse, um die UI reaktiv und flüssig zu halten, während im Hintergrund gearbeitet wird.

Bloc (Event und State)

Block ist eine Architektur zur Trennung von Geschäftslogik und Benutzeroberfläche. Ein Bloc fungiert als Vermittler zwischen Datenquellen (z. B. Datenbank oder API) und der UI, indem er Eingaben (Events) entgegennimmt und entsprechende Ausgaben (States) erzeugt. Diese Struktur fördert Wiederverwendbarkeit und Testbarkeit.

Event und State

Events und States sind Kernbestandteile des BLoc-Patterns.

Events repräsentieren Benutzeraktionen oder Systemereignisse (z. B. „Button gedrückt“ oder „Daten geladen“), die an den BLoc gesendet werden. Der BLoc verarbeitet diese Events und gibt States zurück, die den aktuellen Zustand der UI beschreiben (z. B. „Laden“, „Erfolg“, „Fehler“). Diese Trennung ermöglicht eine präzise Steuerung der App-Logik und ein reaktives UI-Design.

MultiBlocProvider

MultiBlocProvider ist ein spezielles Widget aus Flutter, das in der main.dart verwendet wird, um mehrere Bloc's zentral bereitzustellen. Es ermöglicht, dass verschiedene Teile der App (z. B. Seiten oder Widgets) auf unterschiedliche Bloc's zugreifen können, ohne diese jedes Mal einzeln zu instanziiieren. Dies ist besonders nützlich in größeren Anwendungen wie dieser, wo mehrere unabhängige Zustände verwaltet werden müssen.

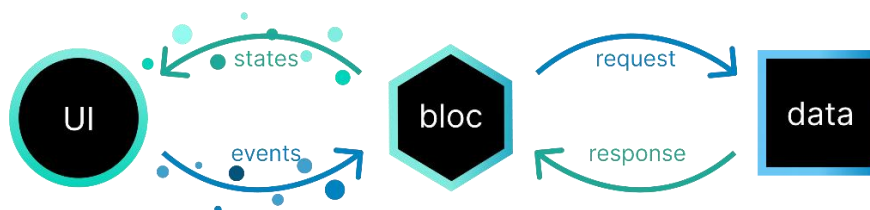


Abbildung 29: Bloc Funktion

Cubit

Ein Cubit ist eine vereinfachte Variante eines Bloc's, die ohne explizite Events auskommt. Statt Events zu definieren, ändert ein Cubit den Zustand direkt über Funktionsaufrufe. Cubits sind ideal für einfachere Zustandsverwaltung, bei der keine komplexe Ereignisverarbeitung erforderlich ist, und bieten eine schlankere Alternative zu vollwertigen Bloc's.



Abbildung 30: Cubit Funktion

6.2 Code-Struktur

Die Code-Struktur des Flutter-Projekts „Fahrschul-Manager“ ist modular und hierarchisch organisiert, um die Wartbarkeit, Skalierbarkeit und Lesbarkeit zu gewährleisten. Sie spiegelt die Integration der genannten Technologien wider – wie Bloc, MultiBlocProvider, Cubit, Event/State, Async und Parse – und orientiert sich an bewährten Flutter-Praktiken. Im Folgenden wird die Struktur anhand von Beispielen beschrieben.

Die typische Flutter-Projektstruktur beginnt mit dem Root-Verzeichnis, das Unterordner wie lib, assets und pubspec.yaml enthält. Die eigentliche Logik liegt im lib-Ordner, der wie folgt aufgeteilt ist:

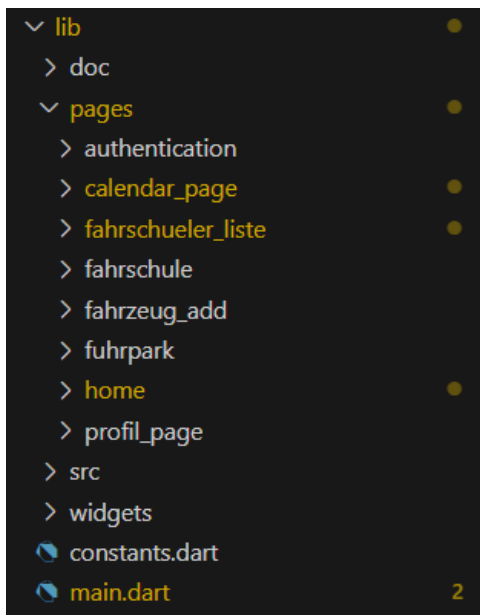


Abbildung 31: VSCode Projektstruktur

main.dart

- Einstiegspunkt der App
- Hier wird MultiBlocProvider, um mehrere Bloc's oder Cubits (z. B. NavBarBloc, FahrzeugAddBloc) zentral bereitzustellen. Dies ermöglicht den Zugriff auf globale Zustände über die gesamte App hinweg

```
@override
Widget build(BuildContext context) {
  return MultiBlocProvider(
    providers: [
      BlocProvider(create: (context) => NavBarBloc()),
      BlocProvider(create: (context) => AsyncRegistrationValidationFormBloc()),
      BlocProvider(create: (context) => AsyncLoginValidationFormBloc()),
      BlocProvider(create: (context) => FahrschuelerListBloc()),
      BlocProvider(create: (context) => PasswordChangeBloc()),
      BlocProvider(create: (context) => ProfilPageBloc()),
    ],
  );
}
```

Abbildung 32: MultiBlocProvider

constants.dart

- Dient als zentrale Sammlung von Konstanten, die projektweit verwendet werden. Sie enthält statische Werte wie Farben und Standardtexte, die in der App konsistent eingesetzt werden sollen

```
// Main Colors
const Color mainColor = Color(0xFF00BF6D);
const Color textFieldColor = Color(0xFFFF5FCF9);

const Color mainColorComplementaryFirst = Color(0xFF87CEEB);
const Color mainColorComplementaryFirstShade100 = Color.fromARGB(255, 199, 233, 245);
const Color mainColorComplementarySecond = Color(0xFFFFD700);
const Color mainColorComplementarySecondShade100 = Color.fromARGB(255, 255, 240, 179);

// States
const String stateActive = "Aktiv";
const String statePassive = "Passiv";
const String stateUnassigned = "Nicht zugewiesen";
const String stateDone = "Abgeschlossen";
```

Abbildung 33: Constants

pages

- Enthält die UI-Seiten der Anwendung als Widgets (z. B. fuhrpark_page.dart)
- Unterordner wie fahrschule/bloc trennen die Bloc-Logik von der UI, wobei Events und States in eigenen Dateien definiert sind. UI wird in diesem Beispiel in der fahrschule_page.dart abgebildet

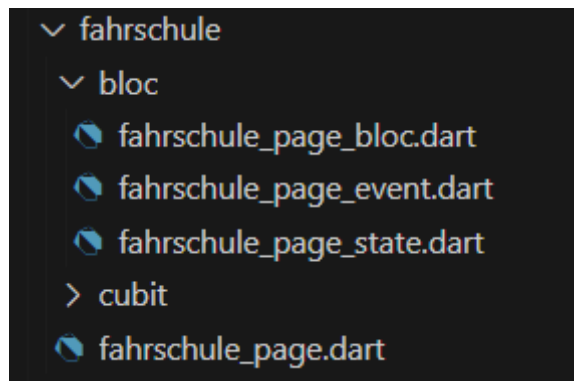


Abbildung 34: VSCode pages Beispiel

src

- Beherbergt wiederverwendbare Logik und Datenklassen
- Beispielsweise die von Hr. Paleyron selbst geschriebene „Benutzer“ Klasse in user.dart die immer wieder im Code Anwendung findet

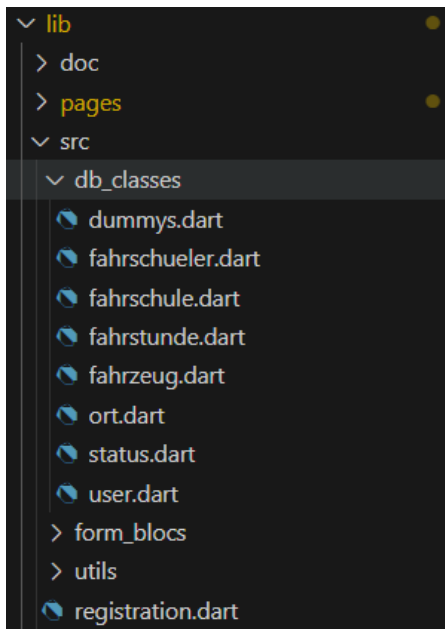


Abbildung 35: src Verzeichnis

6.3 Besondere Herausforderungen

Luis Schulte

Besondere Herausforderungen war es anfangs sich in das Flutter Framework und die darin eingesetzten Techniken wie z.B. Asynchrone Programmierung oder die Bloc Technik mit ihren States und Events einzulesen und einzuarbeiten. Ebenfalls neu war es für mich möglichst viel, wenn es denn Sinn macht, im Code aufzuteilen und auszulagern um so einen möglichst hohen wiederverwendbaren und einfach zu pflegenden Code zu haben.

Des Weiteren das Verständnis wie Back4App arbeitet, und wie die Daten miteinander verbunden sind und wie diese im Code abgefangen werden können. Auch unklar am Anfang war wie mit den Standardtabellen umzugehen ist und was die Funktionen dieser im Hintergrund sind.

Christophe Paleyron

Während des Projekts stand ich vor der Herausforderung, meinem Kollegen Flutter sowie grundlegende und fortgeschrittene Programmierkonzepte näherzubringen. Der schulische Unterricht vermittelte zwar Basiswissen, reichte jedoch nicht aus, um tiefgehendes Wissen über Best Practices, Design Patterns und Clean Code zu vermitteln.

Eine der größten Schwierigkeiten war, dass mein Kollege wenig Erfahrung mit strukturiertem und wartbarem Code hatte. Daher musste ich nicht nur die technischen Grundlagen von Flutter erklären, sondern auch ein Bewusstsein für sauberen Code und bewährte Entwurfsmuster schaffen. Dabei galt es, eine Balance zwischen Theorie und Praxis zu finden, um das Gelernte direkt anwenden zu können.

Besonders anspruchsvoll war es, komplexe Konzepte wie das Bloc-Pattern, Dependency Injection oder effektives State Management verständlich zu vermitteln. Ich musste individuell auf seinen Kenntnisstand eingehen und verschiedene Methoden nutzen – darunter Code-Reviews, Pair Programming und kleine Workshops.

Trotz der anfänglichen Herausforderungen konnte ich durch kontinuierliche Unterstützung und praxisnahe Beispiele seine Lernkurve deutlich steigern. Am Ende des Projekts hatte er nicht nur ein besseres Verständnis für Flutter, sondern auch für saubere und nachhaltige Softwareentwicklung.

Eine weitere große Herausforderung war die benutzerfreundliche Gestaltung der Terminverwaltung. Das Ziel war, eine intuitive und effiziente Oberfläche zu schaffen, die es dem Nutzer ermöglicht, Termine einfach zu verwalten, ohne auf technische Details achten zu müssen.

Ein zentrales Problem war die korrekte und konfliktfreie Buchung von Terminen. Es musste sichergestellt werden, dass doppelte oder sich überschneidende Termine verhindert werden, ohne dabei die Flexibilität der Nutzer einzuschränken. Dazu habe ich Mechanismen zur Validierung und Prüfung der Verfügbarkeit implementiert, um mögliche Kollisionen frühzeitig zu erkennen und dem Nutzer direktes Feedback zu geben.

Zusätzlich war eine Echtzeit-Aktualisierung der Termine erforderlich, damit Änderungen sofort für alle Nutzer sichtbar sind. Dies stellte eine technische Herausforderung dar, da die Synchronisation über mehrere Geräte hinweg zuverlässig und performant funktionieren musste. Ich habe hierfür die Tabellen in der Datenbank angepasst und eine grundlegende Änderung der Architektur vorgenommen, um schnelle und konsistente Updates sicherzustellen.

Durch diese Maßnahmen konnte ich eine Lösung entwickeln, die nicht nur leicht zu bedienen ist, sondern auch zuverlässig arbeitet und Konflikte in der Terminplanung verhindert.

Noch eine Herausforderung war das korrekte Verhalten der Software beim Ausloggen eines Nutzers. Es musste sichergestellt werden, dass der Nutzer automatisch abgemeldet wird, sobald sein Session-Token in Back4App gelöscht wird, und er anschließend direkt zur Login-Seite weitergeleitet wird.

Ein Problem bestand darin, dass die App zunächst nicht in Echtzeit auf das Löschen des Tokens reagierte, sodass Nutzer theoretisch weiterhin auf geschützte Bereiche zugreifen konnten, bis die App manuell geschlossen oder neu gestartet wurde. Das Verhalten der App musste korrigiert werden, um sicherzustellen, dass sie zuverlässig auf eine abgelaufene oder gelöschte Session reagiert.

Hierfür habe ich die Software-Architektur angepasst, um eine zentrale Authentifizierungslogik zu integrieren. Diese überwacht in definierten Intervallen oder bei bestimmten Aktionen den Gültigkeitsstatus des Tokens. Sobald festgestellt wird, dass der Token ungültig oder gelöscht wurde, erfolgt automatisch ein Logout-Prozess, der den Nutzer sicher zurück zur Login-Seite bringt.

Durch diese Anpassungen konnte sichergestellt werden, dass die App stets korrekt auf Sitzungsänderungen reagiert und Nutzer nicht unbemerkt in einem abgelaufenen Zustand bleiben. Dies verbessert sowohl die Sicherheit als auch die Benutzerfreundlichkeit der Anwendung.

7 Test und Qualitätssicherung

7.1 Teststrategie

Getestet wurde der Code im Laufe des Projektes für jede Seite einzeln. Es wurde überprüft ob die vorher festgelegten Funktionen welche die Seite erfüllen soll ohne Probleme funktioniert. Falls es da zu Problemen gekommen ist wurde dieses sich Individuell näher angesehen und behoben.

Dadurch konnte eine Saubere Endphase des Projektes gewährleistet werden, in der nur noch kleinere Probleme behoben werden mussten.

Abschließend haben wir uns parallel als Fahrlehrer und als Fahrschüler eingeloggt, um zu testen ob die Individuellen Funktionen wie das eintragen vom Fahrschüler in einen vom Fahrlehrer freigegeben Termin möglich ist.

7.2 Fehlerbehebungen

Fehler wurden ähnlich wie das Testen des Codes gehandhabt. Im Laufe der Durchführung wurden diese entweder individuell behoben, oder gemeinsam besprochen wie diese am besten behoben werden können.

8 Fazit und Ausblick

8.1 Zusammenfassung

Die Projektarbeit „Fahrschul-Manager“ wurde im Rahmen der Weiterbildung zum staatlich geprüften Techniker (Informatik) an der Rudolf-Diesel-Fachschule als Teamprojekt umgesetzt. Ziel war die Entwicklung einer Flutter-App zur effizienten Verwaltung von Terminen und Fahrzeugen in Fahrschulen sowie die Anwendung schulischer Kenntnisse und Förderung von Projektmanagement- und Teamfähigkeiten.

Die App löst Probleme wie ungenutzte Absagen und aufwändige Fahrzeugkoordination, indem sie Fahrlehrern und Fahrschülern eine Plattform bietet. Freie Timeslots können veröffentlicht und gebucht werden, Fahrzeuge einfacher verwaltet, und neue Fahrschüler automatisch zugeordnet werden. Technologisch nutzt sie Flutter, Bloc für State-Management, Async für asynchrone Prozesse und Parse (Back4App) als Backend.

8.2 Kritische Reflexion

Die Entwicklung des „Fahrschul-Managers“ war ein lehrreiches Projekt, das unsere Kompetenzen auf mehreren Ebenen gefordert hat. Zu den Stärken zählen die gelungene Teamdynamik, die durch offene Kommunikation und gegenseitige Unterstützung geprägt war, sowie ein effektives Zeitmanagement, das uns half, Meilensteine ohne großen Stress zu erreichen. Technisch überzeugte die Nutzung von Flutter und Bloc, die eine intuitive App ermöglichten, während Back4App als Backend eine schnelle Datenintegration ermöglichte.

Herausforderungen ergaben sich jedoch beim Verständnis der Struktur von Back4App, insbesondere wie sie als objektorientierte Datenbank funktioniert – die Logik hinter Klassen, Objekten und

Beziehungen war komplexer als erwartet und erforderte zusätzlichen Lernaufwand. Auch die Planungsphase nahm mehr Zeit in Anspruch als gedacht, etwa durch die Erstellung des ERM (Entity-Relationship-Modells). Eine genauere Einschätzung des Aufwands hätte hier geholfen.

Das Projekt lehrte uns, technische Konzepte gründlicher zu analysieren und Planungszeiten realistischer einzuschätzen. Für zukünftige Arbeiten würden wir mehr Zeit für das Verständnis des Backends und der Modellierung einplanen.

8.3 Verbesserungspotenzial

Zukünftige Funktionen könnten sein, das neben Autos auch Motorräder und Anhänger in der App gepflegt und gebucht werden können.

Des Weiteren könnten mehr Funktionen auf der Profilseite zur Verfügung gestellt werden. Beispielsweise die Möglichkeit ein Profilbild festzulegen, oder die E-Mail sowie das Passwort zu ändern.

9 Anhang

9.1 Quellen

<https://bloclibrary.dev/bloc-concepts/>

<https://www.back4app.com/>

<https://flutter.dev/>

9.2 Literaturverzeichnis

9.3 Abbildungsverzeichnis

Abbildung 1: Teilziele.....	4
Abbildung 2:Back4app.....	7
Abbildung 3: ERM-Entwurf 1	8
Abbildung 4:ERM-Final	9
Abbildung 5: Logisches Modell-Auszug-Früher Entwurf	10
Abbildung 6: Logisches Modell-Final	11
Abbildung 7: Login Seite	12
Abbildung 8: Fahrschule Seite-Fahrlehrer	12
Abbildung 9: Ablaufdiagramm Fahrlehrer	13
Abbildung 10: Stundennachweis	14
Abbildung 11: Erreichte Zwischenziele.....	14
Abbildung 12: Back4App Klassen	15
Abbildung 14: Login Seite	16
Abbildung 13: Registrierung Seite 1	16
Abbildung 15: Regex Beispiel	17
Abbildung 16: Navigationsleiste	17
Abbildung 17: Fahrschülerliste Seite	18
Abbildung 18: Fahrschüler hinzufügen.....	18
Abbildung 19: Kalender Seite	19
Abbildung 20: Termin hinzufügen	19

Abbildung 21: Homepage Seite	20
Abbildung 22: Fahrschulseite	21
Abbildung 23: Fahrlehrer hinzufügen.....	21
Abbildung 24: Standort hinzufügen.....	21
Abbildung 25: Fuhrpark Seite	22
Abbildung 26: Fahrzeug hinzufügen	22
Abbildung 27: Label bearbeiten	22
Abbildung 28: Profil Seite	23
Abbildung 29: Bloc Funktion	24
Abbildung 30: Cubit Funktion.....	24
Abbildung 31: VSCode Projektstruktur.....	25
Abbildung 32: MulitBlocProvider	25
Abbildung 33: Constants	26
Abbildung 34: VSCode pages Beispiel	26
Abbildung 35: src Verzeichnis.....	27