

# DIPLOMARBEIT

## Skyline – Mobile App für Geschäftsreisen

Ausgeführt im Schuljahr 2025/26 von:

Jan-Ole Baumgartner 5BHITM-01  
Boris Plesnicar 5BHITM-02

Betreuer:

DI Jagersberger Andreas, BEd

Krems, am 15.01.2026

### **EIDESSTATTLICHE ERKLÄRUNG**

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Krems, am 15.01.2026

Verfasser/Verfasserinnen:

---

Jan-Ole Baumgartner

---

Boris Plesnicar

# DIPLOMARBEIT

## Bestätigung der Abgabe

Abgabebestätigung

---

Datum

---

Name

---

Unterschrift

## Genehmigung der Diplomarbeit

Approbation

---

Datum

---

Pruefer\*in

---

Abteilungsleiter\*in  
Direktor\*in

# DIPLOMARBEIT

## Dokumentation

### Verfasser\*innen

Jan-Ole Baumgartner, 5BHITM

Boris Plesnicar, 5BHITM

### Abteilung

Informationstechnologie

Ausbildungsschwerpunkt: Systemtechnik

### Schuljahr

2025/26

### Thema der Diplomarbeit

Skyline – Mobile App für Geschäftsreisen

### Kooperationspartner

L&G Bau GmbH

### Aufgabenstellung

Ziel ist die Entwicklung einer mobilen Anwendung zur zentralen Verwaltung von Geschäftsreisen. Die App soll Flüge anlegen und verwalten, Dokumente ablegen, automatische Importe (QR/OCR/E-Mail) ermöglichen, interaktive Karten und Statistiken bereitstellen sowie proaktive Benachrichtigungen integrieren.

### Realisierung

Die Umsetzung erfolgt mit React Native und Expo. Als Backend wird Supabase (Postgres, Auth, Storage, RLS) eingesetzt. Externe Services wie Aviationstack, Maps-APIs und OCR werden integriert. Die Architektur ist modular aufgebaut (Services, Store, Komponenten) und unterstützt Offline- und Sync-Szenarien.

### Ergebnisse

Es entstand ein funktionsfähiger Prototyp mit Flugverwaltung, Kartenansicht, Import-Funktionen, Dokumentenablage, Statistiken und Reminder-System. Die App ermöglicht eine zentrale Sicht auf Reisen und reduziert manuellen Organisationsaufwand.

TODO: Projektscreenshot oder Kartenansicht einfuegen.

TODO: Projektscreenshot einfuegen (z. B. Home oder Map).

Teilnahme an Wettbewerbe oder Auszeichnungen

Keine Teilnahme vorgesehen.

Möglichkeiten der Einsichtnahme in die Arbeit

Einsichtnahme im Projektarchiv der HTBL Krems.

# DIPLOMA THESIS

## Documentation

### Authors

Jan-Ole Baumgartner, 5BHITM

Boris Plesnicar, 5BHITM

### Department

Information Technology

Specialization: System Engineering

### Academic year

2025/26

### Thesis Topic

Skyline – Mobile App for Business Travel

### Co-operation partners

L&G Bau GmbH

### Task Description

The goal is to build a mobile application for centralized management of business trips. The app shall manage flights, store documents, support automatic imports (QR/OCR/e-mail), provide interactive maps and statistics, and integrate proactive notifications.

### Implementation

The implementation uses React Native and Expo. Supabase (Postgres, Auth, Storage, RLS) provides the backend. External services such as Aviationstack, Maps APIs and OCR are integrated. The architecture is modular (services, store, components) and supports sync and offline scenarios.

### Results

A functional prototype was created with flight management, map view, import features, document storage, statistics and a reminder system. The app provides central visibility for trips and reduces manual organization effort.

TODO: Insert project screenshot or map view.

TODO: Insert project screenshot (e.g., Home or Map).

### Participation in Competitions or Awards

No participation planned.

### Accessibility of Diploma Thesis

Available in the HTBL Krems project archive.

# Inhaltsverzeichnis

1.	Präambel	11
1.1.	Team . . . . .	11
1.2.	Danksagung . . . . .	11
1.3.	Gendererklärung . . . . .	11
2.	Einleitung	12
2.1.	Ausgangssituation und Motivation . . . . .	12
2.2.	Problemstellung . . . . .	12
2.3.	Zielsetzung der Arbeit . . . . .	12
2.4.	Forschungsfragen . . . . .	12
2.5.	Vorgehensweise und Methodik . . . . .	13
2.5.1.	Methodik der Modulbewertung . . . . .	13
2.5.2.	Testaufbau und Datengrundlage . . . . .	13
2.6.	Aufbau der Arbeit . . . . .	13
3.	Theoretische Grundlagen	14
3.1.	Mobile Reiseorganisation und digitale Workflows . . . . .	14
3.1.1.	Charakteristika von Reisedaten und deren Verwaltung . . . . .	14
3.1.2.	Herausforderungen bei der Fragmentierung von Reiseinformationen . . . . .	14
3.2.	Grundlagen des automatischen Imports . . . . .	14
3.2.1.	Zielsetzung und Nutzen . . . . .	14
3.2.2.	Abgrenzung zu manueller Erfassung . . . . .	14
3.2.3.	Typische Fehlerquellen bei Importen . . . . .	14
3.2.4.	Datenquellen für Flugdaten . . . . .	14
3.2.5.	Technische Anforderungen . . . . .	15
3.2.6.	Vergleich: Manuell vs. Automatisch . . . . .	15
3.3.	Grundlagen visueller Flugrouten-Darstellung . . . . .	15
3.3.1.	Charakteristika und Nutzen . . . . .	15
3.3.2.	Anforderungen an eine mobile Flugrouten-Karte . . . . .	15
3.3.3.	Technische Grundlagen der Routenberechnung . . . . .	16
3.4.	Proaktive Benachrichtigungssysteme und Erinnerungsmanagement . . . . .	16
3.4.1.	Proaktivität und Push/Pull in Benachrichtigungssystemen . . . . .	16
3.4.2.	Kognitive Grundlagen der Erinnerungslast . . . . .	17
3.4.3.	Notification Fatigue und Governance . . . . .	18
3.4.4.	Anforderungen und Heuristiken im Reise-Kontext . . . . .	18
3.4.5.	Konzeption und technische Umsetzung (Skyline-relevant) . . . . .	20
3.5.	Zentrale Datenverwaltung und Sicherheit für eine mobile Reiseorganisations-App	20
3.5.1.	Einordnung und Zielsetzung . . . . .	20
3.5.2.	Single Source of Truth und Datenkonsistenz . . . . .	20
3.5.3.	Anforderungen an die zentrale Datenverwaltung . . . . .	22
3.5.4.	Sicherheitskonzept und Datenschutz . . . . .	23
3.6.	Anforderungen an eine mobile Reiseorganisations-App . . . . .	24
3.6.1.	Durchgängiger Datenfluss . . . . .	24
3.6.2.	Importanforderungen: QR, OCR, E-Mail . . . . .	25

3.6.3. Validierung, Zeitlogik und Speicherung . . . . .	25
3.6.4. Synchronisierung, Offline und Konflikte . . . . .	25
3.6.5. Integrationsanforderungen . . . . .	25
3.7. Technische Grundlagen der mobilen App-Entwicklung . . . . .	25
3.7.1. Cross-Platform-Frontend mit React Native und Expo . . . . .	25
3.7.2. Backend-as-a-Service mit Datenbank- und Security-Nähe . . . . .	26
3.7.3. Technologievergleich und Architekturentscheidung . . . . .	26
4. Implementierung: Automatischer Import von Boardkarten & Buchungsdaten (Boris) . . . . .	27
4.1. Implementierung in Skyline . . . . .	27
4.1.1. QR-Scan und BCBP-Parsing als „Fast Path“ . . . . .	27
4.1.2. OCR-Verarbeitung aus Bildern und PDFs . . . . .	31
4.1.3. Import von Buchungsbestätigungen und Dokumenten . . . . .	34
4.1.4. Validierung und Plausibilitätsprüfung . . . . .	35
4.1.5. Feld-Mapping und Datennormalisierung . . . . .	36
4.1.6. Deduplizierung . . . . .	37
4.1.7. Fehlerbehandlung und Nutzerfeedback . . . . .	37
4.1.8. Anbindung an die Flugverwaltung . . . . .	38
4.2. Ergebnis . . . . .	39
4.2.1. Bewertung der Importqualität . . . . .	39
4.2.2. Nutzen im Reise-Workflow . . . . .	39
4.2.3. Grenzen und Verbesserungspotenzial . . . . .	39
5. Implementierung: Zentrale & sichere Datenverwaltung (JanOle) . . . . .	41
5.1. Implementierung in Skyline . . . . .	41
5.1.1. Datenmodell & Tabellenstruktur . . . . .	42
5.1.2. Service-Layer-Architektur . . . . .	49
5.1.3. Beziehung Flight ↔ Notes, Docs & Checklists . . . . .	50
5.1.4. Storage-Bucket & Signed URLs . . . . .	53
5.1.5. Sync-Strategie & Caching . . . . .	54
5.1.6. Fehlerfälle & Wiederherstellung . . . . .	55
5.2. Bewertung der Wirkung . . . . .	56
5.2.1. KPI: Transparenz . . . . .	56
5.2.2. KPI: Nachvollziehbarkeit . . . . .	57
5.2.3. Vergleich zu dezentralen Lösungen . . . . .	57
5.3. Ergebnis . . . . .	58
5.3.1. Verbesserungen in Transparenz . . . . .	58
5.3.2. Verbesserungen in Nachvollziehbarkeit . . . . .	58
5.3.3. Schlussfolgerung . . . . .	58
5.4. Rechtliche Rahmenbedingungen (projektrelevant) . . . . .	59
5.4.1. Datenschutzrechtliche Vorgaben (DSGVO) . . . . .	59
5.4.2. Datenschutz in Skyline . . . . .	61

6.	Implementierung: Interaktive Weltkarte & Routenvisualisierung (Boris)	64
6.1.	Implementierung der Karten-Visualisierung . . . . .	64
6.1.1.	Polyline-Darstellung und Great-Circle-Sampling . . . . .	65
6.1.2.	Live-Marker & Flugbewegung . . . . .	68
6.1.3.	Fortschritts-Overlay (zurückgelegte Flugdistanz) . . . . .	70
6.1.4.	Interaktive Flugauswahl (Map ↔ Details) . . . . .	70
6.1.5.	History vs. Upcoming-Flüge und Filter . . . . .	71
6.2.	Bewertung der Kartenlösung . . . . .	72
6.2.1.	Kriterienkatalog . . . . .	72
6.2.2.	Performance-Messungen . . . . .	72
6.2.3.	Nutzerfeedback . . . . .	73
6.2.4.	Stärken-Schwächen-Analyse . . . . .	73
6.2.5.	Ergebnis . . . . .	74
7.	Implementierung: Benachrichtigungs- und Erinnerungsmodul (JanOle)	75
7.1.	Implementierung in Skyline . . . . .	75
7.1.1.	ReminderOffsets & SchedulingFlow . . . . .	75
7.1.2.	Integration beim FlightSave . . . . .	75
7.1.3.	Cancel/Reschedule bei Updates . . . . .	75
7.1.4.	PushIntegration (EAS / Expo Tokens) . . . . .	75
7.2.	Bewertung der Wirkung . . . . .	75
7.2.1.	Zuverlässigkeit als KPI . . . . .	75
7.2.2.	Effizienz als KPI . . . . .	76
7.2.3.	Nutzerakzeptanz . . . . .	76
7.3.	Ergebnis . . . . .	76
7.3.1.	Reduktion kritischer Fehlzustände . . . . .	76
7.3.2.	Effizienzsteigerung . . . . .	76
7.3.3.	Gesamtbewertung . . . . .	76
8.	Bewertung und Evaluation	77
8.1.	Überblick der Bewertungskriterien . . . . .	77
8.2.	Beantwortung der Forschungsfragen . . . . .	77
8.3.	Zusammenfassung der Modulbewertungen . . . . .	77
8.3.1.	Import und Datenverwaltung . . . . .	77
8.3.2.	Kartensvisualisierung und Benachrichtigungen . . . . .	78
8.4.	Gesamtbewertung . . . . .	78
9.	Technische Umsetzung und Architektur	79
9.1.	Systemarchitektur . . . . .	79
9.1.1.	Client (React Native + Expo) . . . . .	79
9.1.2.	Backend (Supabase + Postgres) . . . . .	79
9.1.3.	Storage (Dokumente, Bilder) . . . . .	79
9.2.	Technologien . . . . .	79
9.2.1.	Frontend-Stack . . . . .	79
9.2.2.	Backend-Stack . . . . .	80

9.2.3. APIs (Aviationstack, OCR, Maps, Email-Import) . . . . .	80
9.3. Funktionalitaet . . . . .	80
9.3.1. Flugverwaltung (CRUD) . . . . .	80
9.3.2. Import (QR/OCR/E-Mail) . . . . .	80
9.3.3. Map & Animation . . . . .	80
9.3.4. Notifications . . . . .	80
9.3.5. Dokumente / Notizen / Checklisten . . . . .	80
9.3.6. Statistiken & Export . . . . .	80
9.3.7. Company-Features (Invite, Join, Dashboard) . . . . .	81
9.4. Projektbezogene Umsetzung . . . . .	81
9.4.1. Umsetzung der Karten-Visualisierung . . . . .	81
9.4.2. Umsetzung der Import-Funktionen . . . . .	81
9.4.3. Umsetzung der Benachrichtigungen . . . . .	82
9.4.4. Umsetzung der Datenverwaltung . . . . .	82
9.4.5. Umsetzung der Gamification-Elemente . . . . .	82
9.4.6. Teststrategie & Validierung . . . . .	83
10. Installation . . . . .	84
10.1. Voraussetzungen . . . . .	84
10.2. Konfigurieren der Datenbank . . . . .	84
10.3. Starten des Programms . . . . .	84
11. Zusammenfassung und Ausblick . . . . .	85
11.1. Zusammenfassung der Ergebnisse . . . . .	85
11.1.1. Beantwortung der vier Forschungsfragen . . . . .	85
11.1.2. Schlussfolgerungen aus den Modulbewertungen . . . . .	85
11.2. Ausblick und zukünftige Entwicklungen . . . . .	85
11.2.1. Erweiterungsmöglichkeiten der App . . . . .	85
11.2.2. Verbesserungspotenziale . . . . .	85
I. Abbildungsverzeichnis . . . . .	86
II. Tabellenverzeichnis . . . . .	87
III. Quellcodeverzeichnis . . . . .	88
A. Anhang . . . . .	89
A.1. Arbeitsteilung . . . . .	89
A.2. Kapitelverzeichnis . . . . .	89
A.3. Projekttagebücher . . . . .	89
A.3.1. Projekttagebuch Max Mustermann . . . . .	89
A.3.2. Projekttagebuch Mex Musterjuan . . . . .	89
A.4. Besprechungsprotokolle . . . . .	90
A.5. Datenträgerbeschreibung . . . . .	92
A.6. Einsatz von KI-Tools . . . . .	92

# 1. Präambel

## 1.1. Team

Das Projektteam besteht aus Jan-Ole Baumgartner und Boris Plesnicar. Jan-Ole fokussiert auf Benachrichtigungen, Reiseplanung und Backend-Anbindung, Boris auf Kartensvisualisierung, Import und UI-Umsetzung. Die Arbeit wurde gemeinsam konzipiert und implementiert.

## 1.2. Danksagung

Wir bedanken uns bei DI Jagersberger Andreas, BEd für die Betreuung und fachliche Unterstützung. Ein weiterer Dank gilt der HTBL Krems sowie allen Personen, die durch Feedback und Tests zur Qualität der App beigetragen haben.

## 1.3. Gendererklärung

Zur besseren Lesbarkeit der Diplomarbeit wurde ausschließlich die männliche Form verwendet. Da Begriffe wie „Benutzerinnen und Benutzer“ den Text unleserlich machen, wurde es schlicht auf „Benutzer“ gekürzt, dies soll jedoch keine Geschlechterdiskriminierung zum Ausdruck bringen.

## 2. Einleitung

### 2.1. Ausgangssituation und Motivation

Reisebezogene Informationen wie Buchungsdaten, Boardingkarten und Belege werden in der Praxis häufig in unterschiedlichen Medien und Systemen gespeichert (E-Mail, Dateien, Kalender, Papier). Diese Verteilung erschwert die strukturierte Organisation, verlängert Suchvorgänge und reduziert die Nachvollziehbarkeit im Unternehmenskontext. Die Diplomarbeit adressiert diese Fragmentierung durch eine zentrale, mobile Anwendung, die Flüge, Reisedaten, Dokumente und Erinnerungen konsistent zusammenführt.

### 2.2. Problemstellung

Die zentrale Problemstellung ist die fehlende Transparenz und Nachvollziehbarkeit von Reiseinformationen bei gleichzeitig hoher Zeitkritikalität (z. B. Check-in, Boarding, Belegverwaltung). Ohne strukturierte Ablage und proaktive Hinweise entstehen organisatorische Fehler und ein hoher manueller Aufwand. Daraus ergeben sich Anforderungen an Importprozesse, Datenhaltung, Sicherheit, Visualisierung und Benachrichtigung. Die Verarbeitung personenbezogener Reisedaten muss zudem den Vorgaben der DSGVO entsprechen [?].

### 2.3. Zielsetzung der Arbeit

Ziel ist die Umsetzung der App „Skyline“ als integrierte Lösung für Flugverwaltung, Dokumentenablage, automatischen Import, Kartenvizualisierung, Statistiken und Erinnerungen. Die Arbeit dokumentiert die Implementierung strukturiert und bewertet die Wirkung in Bezug auf Zuverlässigkeit, Effizienz, Transparenz und Nachvollziehbarkeit.

### 2.4. Forschungsfragen

Die Arbeit orientiert sich an folgenden vier Forschungsfragen, die den Modulen der Implementierung zugeordnet sind:

- **Weltkarte (Boris):** Wie sehr steigert eine visuelle Darstellung die Übersicht und Transparenz bei Vielreisenden?
- **Import (Boris):** Wie viel Zeittersparnis bringt die automatische Datenübernahme im Vergleich zur manuellen Eingabe?
- **Benachrichtigungen (Jan-Ole):** Wie sehr erhöhen proaktive Benachrichtigungen die Zuverlässigkeit und Effizienz bei der Reiseorganisation?
- **Datenverwaltung (Jan-Ole):** In welchem Maße verbessert eine zentralisierte und sichere Datenverwaltung die Transparenz und Nachvollziehbarkeit von Geschäftsreisen?

## 2.5. Vorgehensweise und Methodik

Die Umsetzung erfolgt iterativ auf Basis des Pflichtenhefts und der Implementierungsprotokolle. Anforderungen werden in Module zerlegt, technisch realisiert und anschließend begründet sowie bewertet. Für die Evaluierung werden KPIs und qualitative Kriterien herangezogen, die typische Reiseabläufe abbilden.

### 2.5.1. Methodik der Modulbewertung

Pro Modul werden Bewertungskriterien definiert (z. B. Zuverlässigkeit, Effizienz, Transparenz). Die Ergebnisse werden in Kapitel 8 zusammengefasst.

### 2.5.2. Testaufbau und Datengrundlage

Funktionstests sichern die Hauptabläufe; Reminder- und Import-Tests validieren die zeitkritischen Funktionen. Die Bewertung erfolgt anhand typischer Reiseszenarien.

## 2.6. Aufbau der Arbeit

Kapitel 1 ist die Präambel, Kapitel 2 die Einleitung und Kapitel 3 die theoretischen Grundlagen (Import, Karten, Benachrichtigungen, Datenverwaltung). Kapitel 4 bis 7 beschreiben die Implementierung dieser Module in Skyline. Kapitel 8 fasst Bewertung und Evaluation zusammen. Kapitel 9 behandelt die technische Architektur sowie die projektbezogene Umsetzung (Karten, Import, Benachrichtigungen, Datenverwaltung, Gamification, Tests). Kapitel 10 enthält die Installation, Kapitel 11 schließt mit Zusammenfassung und Ausblick.

### 3. Theoretische Grundlagen

#### 3.1. Mobile Reiseorganisation und digitale Workflows

##### 3.1.1. Charakteristika von Reisedaten und deren Verwaltung

Reisedaten umfassen Fluginformationen, Buchungsdaten, Dokumente, Belege und Erinnerungen. Diese Informationen werden in der Praxis häufig in unterschiedlichen Medien und Systemen gespeichert (E-Mail, Dateien, Kalender, Papier), was zu Fragmentierung führt.

##### 3.1.2. Herausforderungen bei der Fragmentierung von Reiseinformationen

Die Verteilung erschwert die strukturierte Organisation, verlängert Suchvorgänge und reduziert die Nachvollziehbarkeit im Unternehmenskontext. Ohne strukturierte Ablage entstehen organisatorische Fehler und ein hoher manueller Aufwand.

#### 3.2. Grundlagen des automatischen Imports

##### 3.2.1. Zielsetzung und Nutzen

Automatisierter Import reduziert manuellen Aufwand und senkt die Fehlerquote. Ziel ist es, Flugdaten möglichst schnell und korrekt zu erfassen, um den Nutzer von repetitiven Eingaben zu entlasten und die Datenqualität zu erhöhen.

##### 3.2.2. Abgrenzung zu manueller Erfassung

Manueller Import bleibt als Fallback bestehen, ist aber zeitaufwändiger und fehleranfälliger. Automatisierung steigert Komfort und Konsistenz.

##### 3.2.3. Typische Fehlerquellen bei Importen

Typische Probleme sind unvollständige Daten, fehlerhafte OCR-Erkennung oder unklare Codierung in QR/BCBP. Daher sind Validierung und Nachbearbeitung wichtig.

##### 3.2.4. Datenquellen für Flugdaten

###### 3.2.4.1. QR-Code / Boarding Pass (BCBP)

Boardingpässe enthalten standardisierte BCBP-Daten, die sich strukturiert auslesen lassen [?]. BCBP basiert auf festen Positionsfeldern. Die Struktur ermöglicht die Extraktion von Airline, Flugnummer, Datum und Airports [?]. Kernfelder sind Abflug-/Zielairport, Flugnummer und Datum. Diese reichen für einen validen Flight-Import aus. Nicht alle Boardingpässe sind standardkonform; zudem fehlen häufig Zusatzdaten wie Gate oder Sitzplatz.

### 3.2.4.2. OCR aus Bildern/Dokumenten

OCR ermöglicht Import aus Fotos und PDFs, wenn kein QR-Code vorhanden ist. Schriftgröße, Kontrast und Bildrauschen beeinflussen die Genauigkeit der OCR. Nach OCR müssen Texte gepasst und relevanten Feldern zugeordnet werden. Fehlerhafte OCR wird durch Plausibilitätschecks abgefangen, z. B. IATA-Codes oder Datumsformate.

### 3.2.4.3. E-Mail-Import (Buchungsdaten)

Buchungsbestätigungen enthalten strukturierte Informationen, die per Parser ausgelesen werden können. Typische Mails enthalten Buchungsreferenz, Routing, Zeiten und Passagierdaten. Format und Layout unterscheiden sich je Airline. Unterschiedliche Templates erfordern flexible Parser oder heuristische Regeln.

## 3.2.5. Technische Anforderungen

Der Import muss technisch stabil, verifizierbar und nutzerfreundlich sein. Validierung prüft Plausibilität (Datum, Airport-Codes). Feld-Mapping normalisiert externe Daten. Deduplizierung erkennt Mehrfachimporte. Fallback-Strategien und manuelle Ergänzung bei unvollständigen Daten sind erforderlich. Asynchrone Verarbeitung hält die UI responsiv. Fehler müssen nutzerfreundlich kommuniziert werden.

### 3.2.6. Vergleich: Manuell vs. Automatisch

Automatisierung reduziert Zeitaufwand und Fehlerquote, steigert Nutzerakzeptanz, Wiederholbarkeit und Skalierbarkeit.

## 3.3. Grundlagen visueller Flugrouten-Darstellung

### 3.3.1. Charakteristika und Nutzen

Die visuelle Darstellung von Flugrouten dient der Orientierung und der schnellen Übersicht über Reisehistorie und geplante Flüge. Routen sollen geografisch korrekt auf einer Karte dargestellt und mit Interaktion verbunden sein.

### 3.3.2. Anforderungen an eine mobile Flugrouten-Karte

Die Karte muss auf mobilen Geräten performant laufen, interaktiv bedienbar sein und darf die Nutzerführung nicht überladen. Performance & Ladezeiten, Skalierbarkeit bei vielen Flügen, mobile Optimierung (Touch, Displaygrößen), Genauigkeit der Darstellung (geodätisch korrekt, nicht als gerade Linie) und Usability (Fokus, Zoom, Auswahl) [?, ?].

### 3.3.3. Technische Grundlagen der Routenberechnung

Die Routen werden anhand geographischer Koordinaten der Airports berechnet. Für die Distanz zwischen zwei Airports wird die Haversine-Formel verwendet [?]. Die Flugroute wird als Great-Circle-Bogen modelliert. Flugfortschritt in Echtzeit wird über departureAt/arrivalAt interpoliert.

## 3.4. Proaktive Benachrichtigungssysteme und Erinnerungsmanagement

Proaktive Benachrichtigungssysteme verfolgen das Ziel, relevante Informationen nicht erst auf explizite Nutzeranfrage bereitzustellen, sondern situations- und zeitgerecht automatisch auszuliefern. In der Literatur wird diese Logik häufig als Gegenüberstellung von „Push“ (System initiiert die Informationsbereitstellung) und „Pull“ (Nutzer initiiert die Informationsabfrage) diskutiert [?]. Push-Ansätze sind insbesondere dann sinnvoll, wenn Handlungen an Zeitfenster gebunden sind oder wenn das Versäumen einer Handlung hohe Kosten verursacht (z. B. verpasster Check-in oder Gate-Schluss).

Reiseorganisation ist ein prototypischer Anwendungsfall für proaktive Erinnerungssysteme, weil zentrale Prozessschritte (Online-Check-in, Gepäckaufgabe, Sicherheitskontrolle, Boarding) in der Praxis durch harte oder „quasi-harte“ Deadlines strukturiert sind [?, ?, ?, ?, ?]. Die praktische Relevanz zeigt sich auch daran, dass die Reiseindustrie bereits proaktive Muster etabliert hat: Beispielsweise der automatische Versand von Bordkarten an Reisende. Solche Mechanismen externalisieren Erinnerungsarbeit und sind damit eine Blaupause für digitale Reiseassistenten wie Skyline.

Gleichzeitig ist Reiseorganisation kognitiv anspruchsvoll, weil sie mehrere parallele Zukunftsintentionen umfasst (Dokumente prüfen, Abfahrtszeit planen, Gate wechseln, Belege sichern) und weil sich Parameter dynamisch ändern können. In dieser Dynamik entsteht ein Spannungsfeld: Benachrichtigungen sollen rechtzeitig und hilfreich sein, dürfen aber nicht zur Überlastung beitragen (Notification Fatigue) oder in unpassenden Momenten stören [?, ?].

### 3.4.1. Proaktivität und Push/Pull in Benachrichtigungssystemen

In der HCI- und Informatikforschung wird „proaktiv“ nicht nur als „früh“ verstanden, sondern als Eigenschaft eines Systems, **auf eigene Initiative** und **im Sinne des Nutzers** zu handeln [?, ?]. Reaktive Systeme liefern Informationen primär dann, wenn eine Nutzerinteraktion dies auslöst (Suche, Klick, Öffnen einer App). Proaktive Systeme verschieben die Verantwortung für Timing und Erinnern teilweise zurück zum System: Sie erkennen günstige Zeitpunkte, liefern Hinweise und reduzieren damit typische Fehlerklassen wie „zu spät bemerkt“ oder „vergessen“ [?]. Diese Entlastungslogik ist eng verwandt mit dem Konzept des kognitiven Offloadings.

Die Push/Pull-Unterscheidung ist eine Frage der Nutzerintention und Disruption: Push-Systeme können hohe Nützlichkeit erzielen, erzeugen aber das Risiko, Aufmerksamkeit in

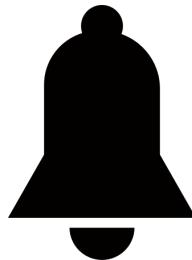


Abbildung 3.1.: Symbol für Benachrichtigungen (Push-Trigger). Quelle: [?]

unpassenden Situationen zu beanspruchen; Pull-Systeme minimieren Disruption, bergen aber das Risiko, dass kritische Informationen nicht rechtzeitig abgerufen werden [?]. Für Skyline folgt daraus eine zentrale Designhypothese: Proaktivität ist dann gerechtfertigt, wenn das System (a) einen stabilen Anlass hat (z. B. Abflugzeit), (b) eine konkrete Handlung ermöglicht (z. B. Check-in öffnen) und (c) das Risiko von Störung durch Quiet Hours und Nutzerkontrolle begrenzt [?, ?].

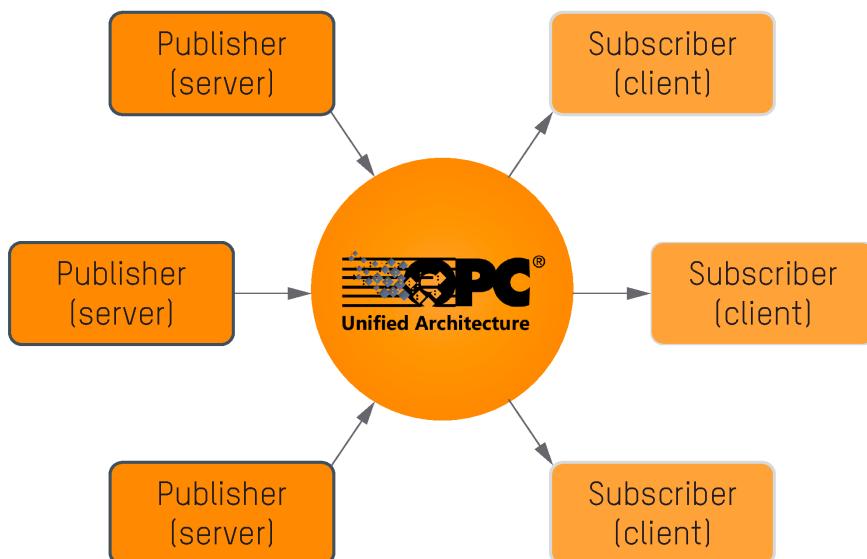


Abbildung 3.2.: Publish/Subscribe-Schema (Push-Prinzip): Der Server liefert Informationen proaktiv an den Client. Quelle: [?]

### 3.4.2. Kognitive Grundlagen der Erinnerungslast

Reisebezogene Aufgaben lassen sich als Fälle des **prospektiven Gedächtnisses** (Prospective Memory) modellieren: das Erinnern, eine beabsichtigte Handlung in der Zukunft auszuführen, häufig ausgelöst durch Zeitpunkte oder Ereignisse [?, ?]. Prospective-Memory-Aufgaben sind in der Reiseorganisation besonders fehleranfällig, weil sie mit konkurrierenden Anforderungen interferieren. Ein zentraler Befund ist, dass Menschen externe Hilfen (Notizen, Kalender, Reminder) nutzen, um die Kosten interner Kontrolle zu senken [?].

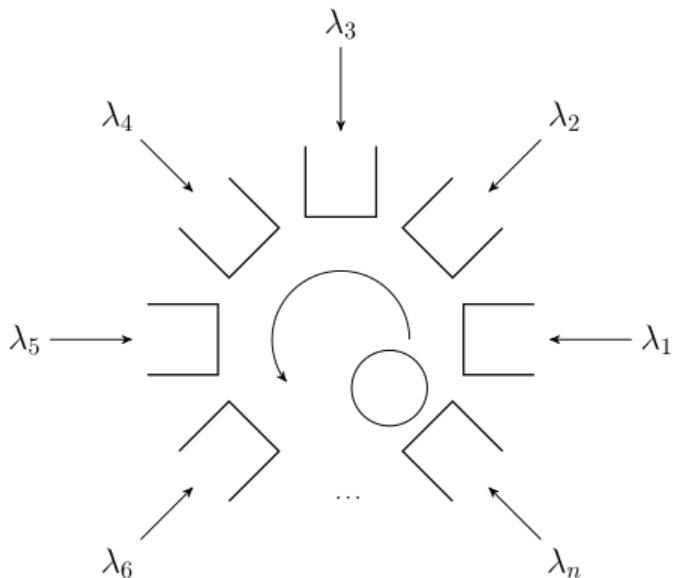


Abbildung 3.3.: Polling-System (Pull-Prinzip): Der Client fragt periodisch beim Server ab. Quelle: [?]

Das Konzept **Intention Offloading** beschreibt die Strategie, Absichten in externe Speicher zu verlagern. Für das Design von Skyline ist wichtig: Erinnerungen wirken zuverlässiger, wenn sie handlungsnah und handlungsfähig sind (z. B. „jetzt einchecken“), statt nur abstrakt zu informieren [?, ?].

### 3.4.3. Notification Fatigue und Governance

Unter **Notification Fatigue** wird verstanden, dass zu häufige oder als irrelevant erlebte Hinweise zu Abwertung, Ignorieren oder Deaktivieren führen [?, ?, ?]. Mobile-HCI-Feldstudien zeigen, dass Benachrichtigungen zwar nützlich sein können, aber oft als störend wahrgenommen werden, wenn Timing und Relevanz nicht zur aktuellen Situation passen [?]. Studien zur mobilen Receptivity belegen, dass „richtig getimed“ oft wichtiger ist als „mehr Informationen“ [?].

Aus Governance-Sicht lassen sich drei Gestaltungsprinzipien ableiten: (1) **Relevanz vor Vollständigkeit**—Systeme sollten das melden, was eine konkrete Handlung ermöglicht [?]. (2) **Nutzerkontrolle und Transparenz**—Benachrichtigungen benötigen bewusste Zustimmung; Best Practices empfehlen, diese im **Kontext der Funktionalität** einzuholen [?]. (3) **Schutzzeiten**—Quiet Hours bzw. Fokus-Modi begrenzen Störungen [?].

### 3.4.4. Anforderungen und Heuristiken im Reise-Kontext

Im Reise-Kontext variieren operative Fenster je Airline und Flughafen. Check-in-Fenster reichen von „ca. 30 Stunden vor Abflug“ bis „bis zu 30 Tage“; Boarding beginnt typischerweise 60 bis 25 Minuten vor Abflug [?, ?, ?]. Flughäfen empfehlen Ankunftszeiten (z. B. 2–3 Stunden

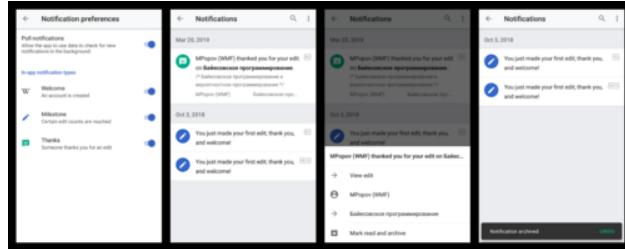


Abbildung 3.4.: Beispielhafte Notification-Interaktion in einer mobilen App (Nutzerkontrolle, Prioritäten). Quelle: [?]

vor Abflug für internationale Flüge). Für Skyline werden daher **zeitbasierte Heuristiken** als Default-Regeln verwendet:

Anlass	Trigger	Begründung
Check-in	T-24 h	breit kompatibel, viele Check-in-Fenster $\geq 24$ h
Dokumente	T-12 h	Vorlauf für Pass/ID/Visa-Check
Zum Flughafen	T-3 h / T-2 h	Lang-/Kurzstrecke, Flughafenempfehlungen
Boarding	T-60 min, T-30 min	deckt Boardingspannen ab
Belege/Quittungen	T+2 h nach Ankunft	handlungsnah nach Reiseende

Tabelle 3.1.: Zeitbasierte Heuristiken für Reise-Reminder (Skyline-relevant)

Für dynamische Ereignisse (Verspätungen, Gate-Wechsel) reichen statische Heuristiken nicht aus. Abbildung 3.5 illustriert, wie Flughäfen Echtzeit-Informationen anzeigen; solche Daten werden typischerweise über kommerzielle APIs bereitgestellt [?, ?, ?]. Skyline nutzt die in der App gespeicherten `departureAt/arrivalAt`-Zeiten als Basis; Echtzeit-Updates würden zusätzliche API-Anbindungen erfordern.



Abbildung 3.5.: Abflugtafel als Beispiel für Echtzeit-Fluginformationen (Verspätungen, Gate). Quelle: [?]

### 3.4.5. Konzeption und technische Umsetzung (Skyline-relevant)

Konzeptionell lässt sich ein proaktives Reise-Benachrichtigungssystem als Pipeline modellieren: (1) **Faktenbasis** (Flugobjekt mit departureAt/arrivalAt), (2) **Regelwerk** (Offsets/Heuristiken), (3) **Scheduler** (lokal und/oder serverseitig), (4) **Delivery** (Benachrichtigung + Deep Link), (5) **Kontrollschicht** (Quiet Hours, Opt-out). Skyline implementiert dies mit Expo Notifications [?].

Triggerzeiten werden als Offsets relativ zu departureAt/arrivalAt berechnet. Technisch kritisch ist die Zeitzonen- und Sommerzeitkorrektheit. Für Persistenz sind hybride Strategien robust: lokale Persistenz ermöglicht Offline-Fähigkeit; serverseitige Persistenz ermöglicht Rescheduling bei Datenänderungen. Deep Links transformieren Benachrichtigungen von „Info“ zu „Handlung“: Der Tap führt direkt in die Flugdetails [?]. Das Berechtigungs-Prompt sollte nicht sofort, sondern **im Kontext der Funktion** (z. B. nach Anlegen eines Flugs) erscheinen [?]. Fehlende oder ungültige Zeitpunkte werden abgefangen; bei widersprüchlichen Daten ist eine degradierte Strategie sinnvoll (nur statische Reminder oder Umstellung auf Pull).

## 3.5. Zentrale Datenverwaltung und Sicherheit für eine mobile Reiseorganisations-App

### 3.5.1. Einordnung und Zielsetzung

Eine mobile Reiseorganisations-App ist nur dann nachhaltig nützlich, wenn sie (a) alle relevanten Reiseinformationen konsistent bündelt, (b) im Alltag schnell Auskunft geben kann („Wann ist mein Boarding?“, „Wo ist mein Ticket?“) und (c) trotz sensibler Datenlage ein belastbares Sicherheits- und Datenschutzniveau erreicht. Diese Ziele sind direkt miteinander gekoppelt: Verteilte Speicherung in E-Mails, Notizen, Screenshots und mehreren Apps erzeugt Inkonsistenz, Suchaufwand und unkontrollierte Replikation sensibler Daten.

Im Geschäftsreise-Kontext kommt zusätzlich Nachweisbarkeit hinzu: Buchungsinformationen, Belege und Änderungen müssen auditierbar und einem konkreten Trip zuordenbar sein. Daraus folgt für die Architektur die Grundentscheidung für eine zentrale, strukturierte Datenbasis (*Single Source of Truth*), kombiniert mit Datenintegrität, klarer Rechteverwaltung und nachvollziehbarer Historie.

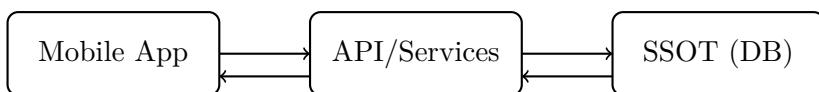


Abbildung 3.6.: Zentraler Datenfluss mit einer kanonischen Datenquelle (SSOT).

### 3.5.2. Single Source of Truth und Datenkonsistenz

SSOT beschreibt das Prinzip, dass jedes relevante Datenelement genau eine massgebliche Pflege- und Referenzstelle besitzt. In der Reiseorganisation ist der Nutzen besonders hoch,

weil Daten typischerweise fragmentiert vorliegen (Buchungsdaten in E-Mails, Dokumente als Fotos, Erinnerungen in Kalendern, Notizen in separaten Apps). Eine zentrale Datenbasis reduziert doppelte Erfassung, Inkonsistenzen und Update-Anomalien.

Konsistenz muss dabei in zwei Schichten betrachtet werden:

- **Konsistenz in der Datenbank:** Schema, Constraints und Transaktionen erzwingen strukturell gültige Zustände [?, ?].
- **Konsistenz über Geräte:** Offline-Phasen und parallele Zugriffe erzeugen Replikations- und Konfliktfragen; hier sind konsistente Synchronisationsregeln erforderlich [?, ?].

Die Normalisierung des Datenmodells ist dafür eine zentrale Grundlage, weil sie Redundanz senkt und Änderungen an einer kanonischen Stelle bündelt.

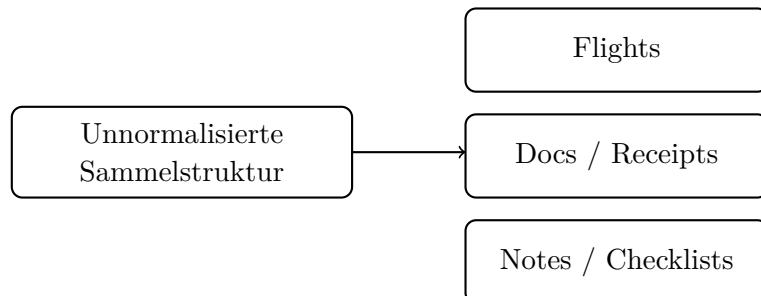


Abbildung 3.7.: Konzeptuelle Normalisierung: von Sammeldaten zu klaren Entitäten.

Für die Synchronisation zwischen Verfügbarkeit und strikter Konsistenz gilt in verteilten Systemen ein Trade-off, der als CAP-Perspektive modelliert wird [?]. Für mobile Reiseapps bedeutet das praktisch: *starke Konsistenz* für Rechte, Besitzverhältnisse und Löschungen; *eventual consistency* für weniger kritische Felder wie Zwischenstände oder lokal gecachte Ansichten.

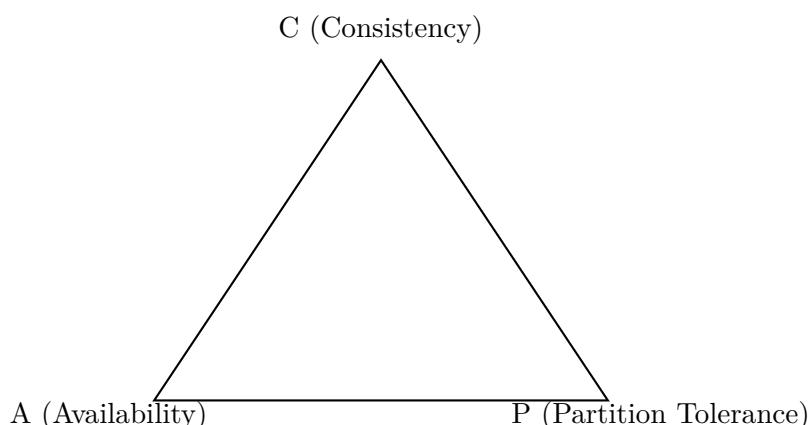


Abbildung 3.8.: CAP-Trade-off als Hintergrund für Offline- und Multi-Device-Design.

### 3.5.3. Anforderungen an die zentrale Datenverwaltung

#### 3.5.3.1. Strukturierte Entitäten und Beziehungen

Ein belastbares Domänenmodell umfasst mindestens `trips`, `flights/legs`, `notes`, `documents`, `checklists` und `receipts`. Relationale Modellierung mit klaren Foreign Keys reduziert Redundanz und unterstützt konsistente Abfragen über den gesamten Trip-Kontext.

#### 3.5.3.2. Transparenz, Status und Nachvollziehbarkeit

Zentral gespeicherte Daten müssen nicht nur vorhanden, sondern auch fachlich interpretierbar sein. Deshalb benötigt ein Trip-Dashboard ableitbare Statusfelder (z. B. `documents_complete`, `receipts_pending`) und einen nachvollziehbaren Bezug zu den auslösenden Ereignissen.

#### 3.5.3.3. Audit-Trail und Historie

Für Geschäftsreisen ist Historie essenziell: Zuordnungen, Korrekturen und Löschungen müssen nachvollziehbar bleiben. Eine pragmatische Umsetzung ist ein Änderungsprotokoll mit `created_at`, `updated_at`, `created_by` sowie dedizierten Audit-Events bei sicherheitsrelevanten Vorgängen.

#### 3.5.3.4. Synchronisierung, Multi-Device und Offline

Mobile Nutzung erfordert lokale Caches und spätere Synchronisierung. Bei parallelen Änderungen sind Konfliktstrategien nötig (Versionierung, Last-Writer-Regeln oder explizite Konfliktauflösung) [?, ?]. Konflikte lassen sich bereits im Datenmodell reduzieren, wenn Änderungen kleinteilig und möglichst additiv modelliert werden.



Abbildung 3.9.: Synchronisationsmodell für Multi-Device mit zentralem Server-SSOT.

#### 3.5.3.5. Rollen und Zugriffsrechte

Neben Owner-Zugriffen werden in realen Szenarien rollenbasierte Rechte benötigt (z. B. Nutzer, Manager, Admin). RBAC ist dafür ein etabliertes Modell, weil es Rechte über Rollen statt über Einzelnutzer verwaltet [?].

### 3.5.3.6. Datenintegrität und Validierung

Importpfade (QR, OCR, E-Mail) erfordern mehrstufige Validierung: *syntaktisch* (Format/Pflichtfelder), *semantisch* (zeitliche Plausibilität, Betrag > 0), *referentiell* (Beleg gehört zu einem gültigen Trip). Das ist auch sicherheitsrelevant, weil unzureichende Validierung ein häufiges mobiles Risiko darstellt [?].

### 3.5.3.7. Konsistente Zustandstransitionen

Wenn mehrere Tabellen fachlich zusammenhängen (z. B. Upload + Metadaten + Trip-Verknüpfung), müssen Updates atomar erfolgen. ACID-Eigenschaften und geeignete Isolationsebenen verhindern halbfertige Zustände unter Last [?].

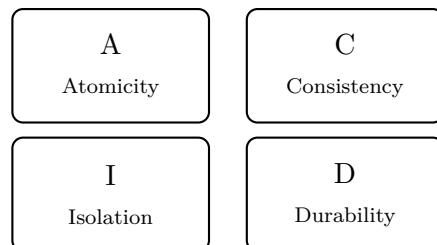


Abbildung 3.10.: ACID als Integritätsfundament für transaktionale Updates.

## 3.5.4. Sicherheitskonzept und Datenschutz

Ein belastbares Sicherheitsmodell kombiniert Identität, Autorisierung, Storage-Schutz, sichere Kommunikation und Datenschutz-Governance. Als Bedrohungsrahmen dienen mobile Top-Risiken wie unsichere Authentifizierung, fehlende Zugriffskontrolle, unsichere Datenspeicherung und Misconfiguration [?].

### 3.5.4.1. Authentifizierung: Sessions, Tokens und Identität

Token-basierte Sessions sind im Mobile-Backend Standard. JWT ist dabei ein etabliertes Format für signierte Claims [?]; OAuth 2.0 bildet den Rahmen für delegierte Zugriffe über externe Identitätsprovider [?]. In Skyline wird dies mit Supabase Auth kombiniert, wodurch Session-Claims und Datenbankberechtigungen zusammengeführt werden [?, ?].

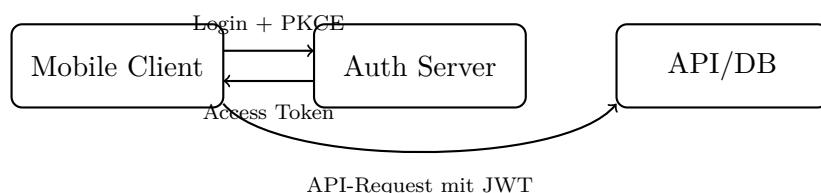


Abbildung 3.11.: Vereinfachter OAuth/JWT-Flow für mobile Authentifizierung.

### 3.5.4.2. Autorisierung: Row Level Security und Policies

Autorisierung beantwortet nicht „wer bist du?“, sondern „was darfst du sehen/ändern?“. Row Level Security (RLS) erlaubt Policy-getriebene Rechteprüfung auf Zeilenebene in der Datenbank [?]. Das folgt dem Prinzip *default deny*: ohne passende Policy kein Datenzugriff.

### 3.5.4.3. Storage-Sicherheit für Dokumente

Reisedokumente und Belege dürfen nicht öffentlich zugänglich sein. Private Buckets, signierte URLs und Policy-gebundene Zugriffe bilden deshalb den Standardansatz [?]. Damit bleibt die Zugriffskontrolle nicht nur in der App-Logik, sondern wird am Datenkern erzwungen.

### 3.5.4.4. Kommunikation, Konfiguration und Logging

Sichere Transportwege, sauberes Secret-Management und restriktive Produktionskonfiguration sind Pflicht. Logging muss zwei Ziele gleichzeitig erfüllen: Fehlersuche ermöglichen, aber keine sensiblen Inhalte unnötig persistieren (z. B. Tokens, Volltexte von Dokumenten).

### 3.5.4.5. DSGVO-Konsequenzen für die Architektur

Die DSGVO wirkt unmittelbar auf technische Entscheidungen:

- **Datenminimierung (Art. 5):** nur notwendige Daten speichern.
- **Privacy by Design/Default (Art. 25):** Zugriff standardmäßig restriktiv auslegen.
- **Löschbarkeit (Art. 17):** End-to-End-Löschnpfad für Datensätze und Dateien vorsehen.
- **Sicherheit der Verarbeitung (Art. 32):** Vertraulichkeit, Integrität, Verfügbarkeit und Resilienz technisch absichern.

Diese Anforderungen sind für eine Reiseorganisations-App nicht optional, sondern Teil der Kernfunktionalität [?].

## 3.6. Anforderungen an eine mobile Reiseorganisations-App

### 3.6.1. Durchgängiger Datenfluss

Der fachliche Kern ist ein stabiler End-to-End-Fluss: **Import → Validierung → Speicherung → Synchronisierung → Abfrage/Anzeige**. Dieser Fluss muss auch unter Fehlerbedingungen (Offline, unvollständige Scans, API-timeouts) robust degradieren können.

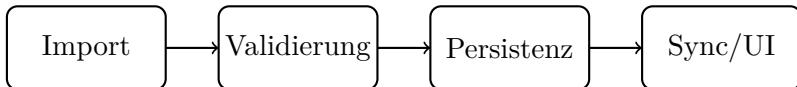


Abbildung 3.12.: Kernpipeline einer mobilen Reiseorganisations-App.

### 3.6.2. Importanforderungen: QR, OCR, E-Mail

QR-/Barcode-Import auf Basis von BCBP-Standards ist für den Flugkontext besonders nutzerfreundlich, weil zentrale Felder strukturiert auslesbar sind [?]. OCR und E-Mail-Import erweitern die Abdeckung, sind aber probabilistisch und fehleranfälliger. Daraus folgt die Pflicht zu einem *human-in-the-loop*-Design: Extrahierte Felder werden als Vorschläge angezeigt und erst nach Nutzerbestätigung als valide übernommen [?, ?].

### 3.6.3. Validierung, Zeitlogik und Speicherung

Zeitkritische Daten (z. B. Boarding, Check-in-Fenster) erfordern mehr als Formatprüfung: Zeitzonen, Datumswechsel und Plausibilität müssen anwendungsseitig geprüft werden. Datenbankseitige Constraints und transaktionale Speicherung bleiben das Rückgrat für konsistente Zustände [?, ?].

### 3.6.4. Synchronisierung, Offline und Konflikte

Offline-Unterstützung verbessert Nutzbarkeit unterwegs deutlich. Gleichzeitig müssen Konflikte bei späterer Synchronisierung modelliert werden [?, ?]. Für Skyline ist ein pragmatischer Ansatz sinnvoll: konfliktarme Datenstrukturen und gezielte Konfliktlogik nur dort, wo realistisch parallele Änderungen auftreten.

### 3.6.5. Integrationsanforderungen

Integrationen steigern den Nutzen, vergrößern aber die Angriffsfläche: Geovisualisierung benötigt saubere Datenmodellierung; Benachrichtigungen benötigen konsistente Zeitdaten und Privacy Controls am Sperrbildschirm [?].

## 3.7. Technische Grundlagen der mobilen App-Entwicklung

### 3.7.1. Cross-Platform-Frontend mit React Native und Expo

React Native ermöglicht plattformübergreifende UI- und Logikentwicklung für iOS und Android [?]. Expo vereinfacht Build-, Deployment- und Update-Prozesse durch ein integriertes Tooling-Oekosystem [?, ?, ?].

### 3.7.2. Backend-as-a-Service mit Datenbank- und Security-Nähe

Supabase kombiniert Postgres, Auth, Storage und Realtime in einer integrierten Plattform [?]. Für diese Arbeit ist besonders relevant, dass RLS und Storage-Policies direkt an der Datenebene greifen [?, ?, ?].

### 3.7.3. Technologievergleich und Architekturentscheidung

Für eine Reiseorganisations-App sind insbesondere vier Vergleichskriterien tragfähig: Datenintegrität, Rechteverwaltung, Offline-Verhalten und Compliance-Fähigkeit.

Pfad	Stärken	Trade-offs
Postgres-zentrierte BaaS (RLS)	starke Integrität, klare Relationen, Policy-Logik am Datenkern	Offline-Konzept muss bewusst modelliert werden
NoSQL-Realtime mit Offline-Persistenz	schnelle Sync-UX, Realtime by default	relationale Konsistenzregeln häufig anwendungsseitig
Offline-first Sync-Framework	explizite Konfliktstrategien, gute Multi-Device-Unterstützung	höhere Plattformkomplexität und Resolver-Aufwand

Tabelle 3.2.: Vergleich möglicher Architekturenpfade für mobile Reiseorganisation

Für Skyline überwiegt ein relationales Zentrum mit RLS, weil Datenintegrität, Nachvollziehbarkeit und zugriffsbasierte Sicherheit im Projektkontext priorisiert sind. Offline-Fähigkeit wird ergänzend über lokale Datenhaltung und kontrollierte Synchronisierung umgesetzt.

## 4. Implementierung: Automatischer Import von Boardkarten & Buchungsdaten (Boris)

Dieses Kapitel beschreibt die konkrete Umsetzung des automatischen Imports von Flug- und Buchungsdaten in Skyline. Die theoretischen Grundlagen zu Datenquellen, Validierung und Fehlerbehandlung werden in Kapitel 3 behandelt; hier liegt der Fokus auf den technischen Komponenten in der App und ihrem Zusammenwirken im Import-Workflow. Besonders relevant sind der standardisierte Bar Coded Boarding Pass (BCBP) nach IATA-Resolution 792 [?] sowie OCR-gestützte Verfahren zur Texterkennung [?, ?].

### 4.1. Implementierung in Skyline

In Skyline werden Flugdetails aus mehreren Quellen extrahiert und in die zentrale Flugverwaltung übernommen: QR-Codes auf Boardingpässen, Fotos bzw. Scans von Tickets und Boardingpässen sowie hochgeladene PDF-Dokumente (z. B. Buchungsbestätigungen aus E-Mails). Der Einstiegspunkt für den Nutzer ist ein einheitlicher „Flight Addition Flow“, in dem zwischen manueller Erfassung und Dokumentenimport gewählt werden kann.

Die Komponente `FlightAdditionFlow` führt den Nutzer zunächst auf einen Auswahl-Screen, der zwei Wege anbietet: manuelle Eingabe oder Dokumentenimport. Wird der Importpfad gewählt, navigiert Skyline zum eigentlichen Import-Screen `AddFlightImportScreen`, der QR-Scan, Foto-/Galerieimport und Dateiupload anbietet. Der Flow ist bewusst so gestaltet, dass er sowohl Einmalnutzer (schneller Scan kurz vor Abflug) als auch Vielreisende mit wiederkehrenden Buchungsdokumenten unterstützt: Die Entscheidung „Manual vs. Import“ steht immer am Anfang, danach werden die einzelnen Schritte (Scan, OCR, Preview, Speichern) über eine einfache Zustandsmaschine (`currentStep`) gesteuert. Fehlerzustände (z. B. fehlende Berechtigungen oder fehlgeschlagene OCR) führen nie in einen „Dead-End“, sondern immer zurück in einen klar definierten Zustand des Flows (erneuter Versuch oder manuelle Eingabe).

Im Folgenden werden die drei Importpfade (BCBP, OCR, Dokumente) und die nachgelagerten Schritte Validierung, Feld-Mapping, Deduplizierung und Übergabe an die Flugverwaltung beschrieben.

#### 4.1.1. QR-Scan und BCBP-Parsing als „Fast Path“

Für den schnellsten Weg der Flugerfassung nutzt Skyline die Kameraschnittstelle von Expo. Die Kamera wird in den Barcode-Modus versetzt und auf die gängigen 2D-Barcode-Formate konfiguriert, die auch im BCBP-Standard genannt werden (PDF417, Aztec, Data Matrix, QR-Code) [?]. Der `CameraView` löst für jeden erkannten Barcode ein Event aus, das an den Handler `handleBarcodeScanned` übergeben wird.

Der Handler liest den Rohinhalt des Barcodes aus, sucht darin das IATA-Standardsegment des Bar Coded Boarding Pass (typischerweise beginnend mit M1) und übergibt den gefundenen String an den Parser:

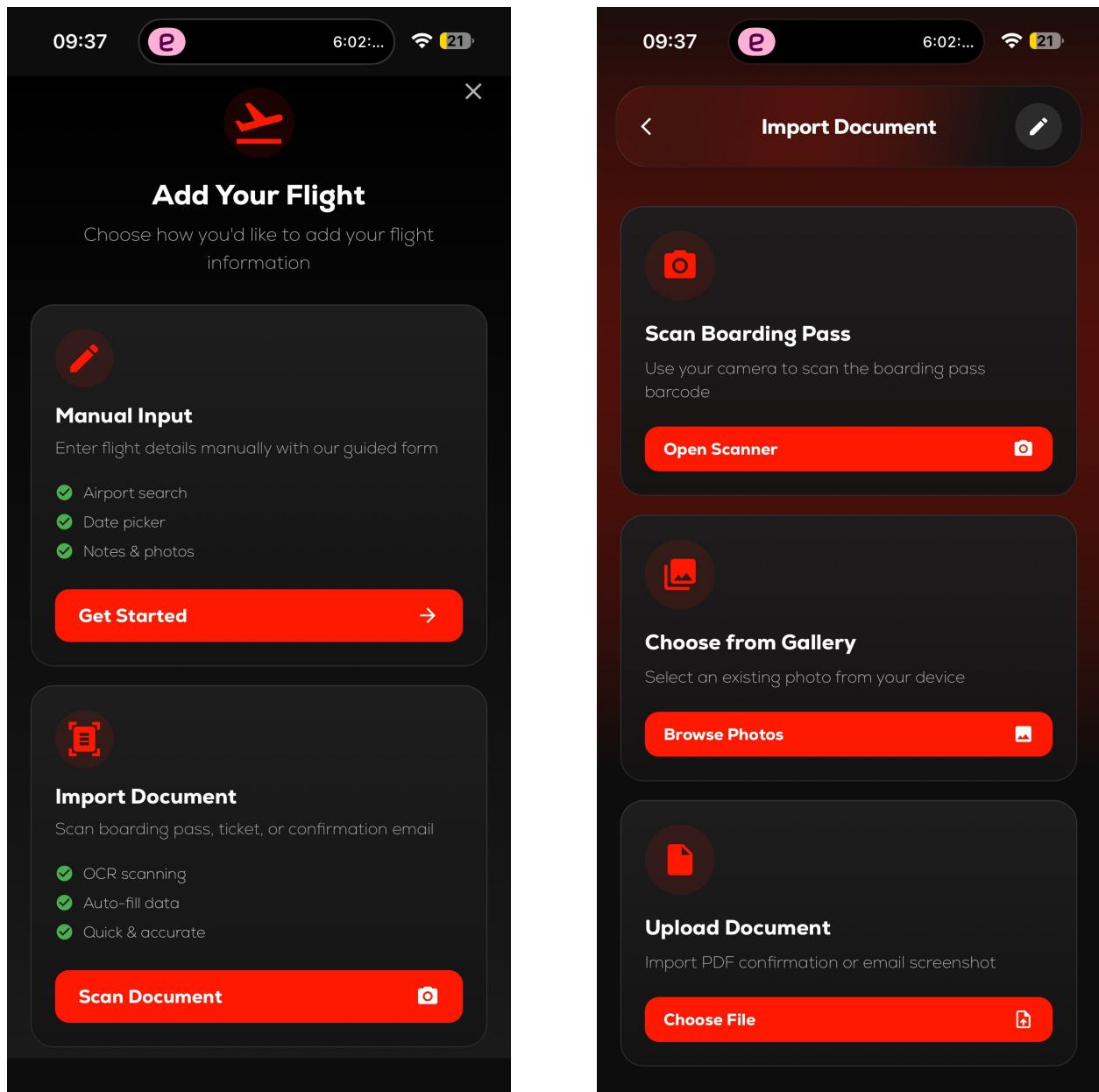


Abbildung 4.1.: Skyline Flight Addition Flow mit Auswahl „Manual Input“ vs. „Import Document“ (links) und Detailansicht der Import-Optionen (QR-/Kamera-Scan, Galerie-Import, Dateiupload; rechts).

Quellcode 4.1: Verarbeitung des QR-/Barcode-Scans in Skyline (vereinfacht, add-flight-import.tsx)

```
1 const handleBarCodeScanned = async (result: { data: string }) => {
2     if (scanLocked) return;
3     setScanLocked(true);
4     setIsScanning(false);
5
6     try {
7         const raw = (result?.data ?? '').toString();
8         let parsed = parseBCBP(raw);
9
10        // Fallback: BCBP-Segment innerhalb groesserer Payload suchen
11        if (!parsed && raw) {
12            const idx = raw.indexOf('M1');
13            if (idx >= 0) {
14                const candidate = raw.slice(idx);
15                if (candidate.length >= 58) parsed = parseBCBP(candidate);
16            }
17        }
18
19        if (!parsed) {
20            Alert.alert('Scan failed', 'No valid boarding pass found. Try again.');
21        }
22        setScanLocked(false);
23        return;
24    }
25
26        // Aus parsed BCBP-Objekt: Flugnummer, Airports, Datum extrahieren
27        const fromIata = parsed.flight.departure.airport;
28        const toIata = parsed.flight.arrival.airport;
29
30        const candidate = {
31            flightNumber: parsed.flight.number || '',
32            from: fromIata ? { iata: fromIata } : undefined,
33            to: toIata ? { iata: toIata } : undefined,
34            date: parsed.flight.date ? toISODateOnly(parsed.flight.date) :
35            undefined,
36        };
37
38        setCandidate(candidate);
39        setForm(s => ({
40            ...s,
41            flightNumber: candidate.flightNumber || s.flightNumber,
42            fromIata: candidate.from?.iata || s.fromIata,
43            toIata: candidate.to?.iata || s.toIata,
44            date: candidate.date ? new Date(candidate.date) : s.date,
45        }));
46    } finally {
47        // Entprellung: erneutes Scannen erst nach Abschluss erlaubt
48        setTimeout(() => setScanLocked(false), 500);
49    }
50};
```

Die eigentliche BCBP-Parsinglogik ist in `bcbp.ts` gekapselt. Sie implementiert die stringbasierte Feldextraktion nach IATA-Standard: BCBP-Nachrichten sind als strukturierte Messages mit festen Feldlängen aufgebaut [?]. Aus dem Rohstring werden u. a. Name, Buchungsreferenz, Flughäfen und Flugnummer extrahiert; das Flugdatum liegt als Julian Day-of-Year vor und wird in ein ISO-Datum konvertiert.

Quellcode 4.2: BCBP-Parser in Skyline (`bcbp.ts`)

```

1  export type ParsedBCBP = {
2    passenger: { name: string | null };
3    flight: {
4      number: string | null;
5      date: string | null; // YYYY-MM-DD
6      departure: { airport: string | null; datetime: string | null };
7      arrival: { airport: string | null; datetime: string | null };
8    };
9    seat: string | null;
10   pnr: string | null;
11 };
12
13 export function julianToISODate(julian: string | number | null | undefined
14   ): string | null {
15   if (julian === null || julian === undefined) return null;
16   const dayNum = Number(julian);
17   if (!Number.isFinite(dayNum) || dayNum <= 0) return null;
18
19   const now = new Date();
20   const year = now.getUTCFullYear();
21   const firstDay = new Date(Date.UTC(year, 0, 1));
22   const date = new Date(firstDay);
23   date.setUTCDate(dayNum);
24   return date.toISOString().slice(0, 10);
25 }
26
27 export function parseBCBP(raw: string): ParsedBCBP | null {
28   const s = (raw ?? '').toString().trim();
29   if (!s || s.length < 58 || s[0] !== 'M') return null;
30
31   try {
32     const nameRaw = s.substring(2, 22);
33     const pnr = s.substring(23, 30).trim() || null;
34     const from = s.substring(30, 33).trim() || null;
35     const to = s.substring(33, 36).trim() || null;
36     const carrier = s.substring(36, 39).trim();
37     const flightNoRaw = s.substring(39, 44).trim();
38     const julian = s.substring(44, 47).trim();
39     const seat = s.substring(48, 52).trim() || null;
40
41     const name = normalizeName(nameRaw);
42     const flightNoNum = flightNoRaw.replace(/\^0+/ , '') || flightNoRaw;
43     const flightNo = `${carrier}${flightNoNum}`;
44     const flightDate = julianToISODate(julian);

```

```

45   return {
46     passenger: { name: name || null },
47     flight: {
48       number: flightNo || null,
49       date: flightDate || null,
50       departure: { airport: from, datetime: null },
51       arrival: { airport: to, datetime: null },
52     },
53     seat,
54     pnr,
55   };
56 } catch {
57   return null;
58 }
59 }

```

---

Besonders wichtig ist hier die Kombination aus strenger Standard-Implementierung (fixe String-Positionen, Füllzeichen) und defensiver Programmierung: Bei ungültigen oder zu kurzen Codes wird `null` zurückgegeben, der Import bricht mit einer verständlichen Fehlermeldung ab und der Nutzer kann einen erneuten Scan versuchen. Gleichzeitig reduziert die `scanLocked`-Logik unnötige Mehrfach-Events — ein praktisches Problem, da Kamera-Callbacks je nach Gerät sehr häufig feuern können. In der Praxis ist der BCBP-Pfad damit der bevorzugte *Fast Path*: Wenn ein Standard-konformer 2D-Barcode vorliegt, erhält Skyline in einem Schritt eine strukturierte Nachricht mit hoher Informationsdichte (Airline, Flugnummer, Datum, Route, ggf. Sitzplatz). Zusätzliche BCBP-Felder wie Sitzplatz oder Sequenznummer werden aktuell nur teilweise in das interne Datenmodell übernommen, bilden aber ein klares Erweiterungspotenzial (z. B. spätere Sitzplatz-Visualisierung oder Boarding-Reihenfolge). Multi-Leg-BCBP (bis zu vier Legs in einem Code) sind standardseitig vorgesehen; die aktuelle Implementierung konzentriert sich auf den ersten Leg-Eintrag (Single-Leg-Parsing) und dokumentiert damit bewusst einen MVP-Scope, der bei Bedarf erweitert werden kann.

#### 4.1.2. OCR-Verarbeitung aus Bildern und PDFs

Wenn kein maschinenlesbarer Barcode verfügbar ist, nutzt Skyline einen OCR-basierten Import. Der Nutzer kann entweder ein Foto aufnehmen, ein Bild aus der Galerie wählen oder ein PDF-Dokument importieren. In allen Fällen wird die Datei eingelesen, als Base64-String kodiert und an einen externen OCR-Dienst geschickt.

Der zentrale Wrapper `runOcrSpace` kapselt Aufruf, Fehlerbehandlung und Validierung der Antwortstruktur:

Quellcode 4.3: OCR-Aufruf und Validierung (services/ocr.ts)

---

```

1 export async function runOcrSpace(
2   base64ImageUrl: string ,
3   apiKey?: string
4 ): Promise<OcrResult> {
5   try {
6     const formData = new FormData();

```

```

7   formData.append('apikey', apiKey || 'helloworld');
8   formData.append('language', 'eng');
9   formData.append('OCREngine', '2');
10  formData.append('base64Image', base64ImageDataUrl);
11
12  const res = await fetch('https://api.ocr.space/parse/image', {
13    method: 'POST',
14    body: formData as any,
15  });
16  const json = await res.json();
17  const parsed = OcrSpaceSchema.safeParse(json);
18  if (!parsed.success) return { ok: false, error: 'Invalid OCR response' };
19
20  if (parsed.data.IsErroredOnProcessing) {
21    const err = Array.isArray(parsed.data.ErrorMessage)
22      ? parsed.data.ErrorMessage.join(', ')
23      : (parsed.data.ErrorMessage || 'OCR processing error');
24    return { ok: false, error: err };
25  }
26
27  const txt = parsed.data.ParsedResults?.[0]?.ParsedText ?? '';
28  if (!txt.trim()) return { ok: false, error: 'No text recognized' };
29  return { ok: true, text: txt };
30 } catch (e: any) {
31   return { ok: false, error: e?.message || 'OCR request failed' };
32 }
33 }

```

---

Im Import-Screen wird dieser Dienst innerhalb der Funktion `processDocument` verwendet. Diese liest das ausgewählte Bild oder PDF, bestimmt den MIME-Typ und ruft anschließend die OCR-API auf:

Quellcode 4.4: Dokument-Import und OCR-Aufruf (add-flight-import.tsx)

```

1 const processDocument = async (uri: string) => {
2   setIsProcessing(true);
3   try {
4     // Datei als Base64 lesen
5     const base64 = await FileSystem.readAsStringAsync(uri, {
6       encoding: 'base64' as any,
7     });
8
9     // MIME-Typ aus Dateiendung bestimmen
10    const lower = uri.toLowerCase();
11    const isPdf = lower.endsWith('.pdf');
12    const isPng = lower.endsWith('.png');
13    const mime = isPdf ? 'application/pdf'
14                  : (isPng ? 'image/png' : 'image/jpg');
15
16    const apiKey = process.env.EXPO_PUBLIC_OCR_SPACE_API_KEY ||
17      undefined;
18    const textRes = await runOcrSpace(`data:${mime};base64,${base64}`);

```

```

18   apiKey);
19   if (!textRes.ok) throw new Error(textRes.error);
20
21   setParsedText(textRes.text);
22   const extracted = extractFlightData(textRes.text);
23   setCandidate(extracted);
24 } catch (error) {
25   Alert.alert(
26     'OCR Failed',
27     error instanceof Error
28     ? error.message
29     : 'Could not extract text. Try a clearer photo or use manual input
30   ,
31   );
32   setCandidate(null);
33 } finally {
34   setIsProcessing(false);
35 }
36 };

```

---

Die aus dem OCR-Text ermittelten Flugdaten werden durch eine heuristische Parsing-Funktion extrahiert. Diese sucht in dem Text u. a. nach typischen Mustern für Flugnummern, Routen (IATA-Codes), Datumsformaten und Uhrzeiten. Die heuristische Natur dieses Pfads ist bewusst: OCR-Erkennung ist stark von Bildqualität und Layout abhängig [?, ?]. Fehlinterpretationen werden durch die nachgelagerte Validierung (Abschnitt 4.1.4) und die manuelle Kontrollmöglichkeit im Preview-Screen abgedeckt. Die Funktion `extractFlightData` ist dabei so aufgebaut, dass sie zunächst robuste Primärmuster sucht (z. B. „AA1234“-Flugnummern und „VIE–LHR“-Routen) und danach auf schwächere Heuristiken zurückfällt:

Quellcode 4.5: Heuristische Extraktion von Flugdaten aus OCR-Text (services/ocr.ts)

```

1 export function extractFlightData(text: string): ExtractedFlight {
2   const upper = text.replace(/\s+/g, ' ').toUpperCase();
3
4   // Flugnummer: AA123, OS0123, LH1234A etc.
5   const flightMatch = upper.match(/(\b([A-Z]{2,3})\s?-?\s?(\d{2,4}[A-Z]?)\b)/);
6   const flightNumber = flightMatch ? `${flightMatch[1]}${flightMatch[2]}` :
7     undefined;
8
9   // Route: VIE–LHR, VIE → LHR, VIE TO LHR usw.
10  let fromIata: string | undefined;
11  let toIata: string | undefined;
12  const routeMatch = upper.match(/(\b([A-Z]{3})\b\s*(?:-|TO|>)\s*\b([A-Z]{3})\b)/);
13  if (routeMatch) {
14    fromIata = routeMatch[1];
15    toIata = routeMatch[2];
16  } else {
17    // Fallback: DEP/ARR oder erste zwei Codes im Text
18    fromIata = upper.match(/(\bDEP\s*[:\s-]\s*\b)/) ? .[1];
19    toIata = upper.match(/(\bARR\s*[:\s-]\s*\b)/) ? .[1];
20  }
21}

```

```
19  if (!fromIata || !toIata) {
20      const allCodes = upper.match(/\b[A-Z]{3}\b/g) || [];
21      if (allCodes.length >= 2) {
22          fromIata = allCodes[0];
23          toIata = allCodes.find(c => c !== fromIata);
24      }
25  }
26 }
27
28 // Datumsnormalisierung (DD.MM.YYYY, YYYY-MM-DD, 12 MAR 2025, etc.)
29 // ...
30 }
```

---

Neben Flugnummer und Route normalisiert `extractFlightData` Datumsangaben aus unterschiedlichen Kulturräumen (numerische und alphanumerische Monatsdarstellung) in ein ISO-kompatibles Datum und extrahiert – sofern vorhanden – Abflug- und Ankunftszeiten. Aus Sicht der Implementierung ist wichtig, dass das Ergebnis explizit als *Kandidat* gekennzeichnet wird: Es wird im UI immer anzeigbar und überschreibbar gehalten, d. h. Nutzerinnen und Nutzer sehen sofort, was die Heuristik erkannt hat, und können offensichtliche OCR-Fehler (z. B. „O“ statt „0“) direkt korrigieren. Damit wird die inhärente Unsicherheit von OCR-Auswertungen nicht versteckt, sondern transparent in den Workflow integriert.

#### 4.1.3. Import von Buchungsbestätigungen und Dokumenten

Buchungsbestätigungen liegen im Alltag meist als E-Mail mit eingebetteter HTML-Darstellung oder als angehängtes PDF vor. Aus Datenschutzgründen implementiert Skyline keinen direkten Zugriff auf das E-Mail-Postfach, sondern folgt einem nutzerzentrierten Modell: Relevante Bestätigungen werden vom Nutzer selbst exportiert (z. B. als PDF oder Screenshot) und anschließend über den Dokumentenimport hochgeladen. Damit wird vermieden, dass große Mengen personenbezogener Daten automatisiert im Hintergrund verarbeitet werden [?, ?].

Der `DocumentUploadModal` stellt drei Wege zur Verfügung: Scan eines Boardingpasses mit der Kamera, Import eines PDFs oder Bilds aus dem Dateidialog sowie Import eines bestehenden Screenshots aus der Galerie. Die hochgeladenen Dokumente werden in einem dedizierten Service verarbeitet: `DocumentService.uploadDocument` liest die Datei, bestimmt MIME- und Dokumenttyp (z. B. `booking_confirmation`) und legt sowohl den Binärinhalt im Supabase-Storage als auch passende Metadaten in der Tabelle `flight_documents` ab [?].

Damit ist die Buchungsbestätigung sowohl als Originaldokument verfügbar als auch – sofern über OCR analysiert – Quelle für strukturierte Flugdaten. Unterschiedliche E-Mail-Layouts werden bewusst nicht mit hart kodierten Templates adressiert; stattdessen setzt Skyline auf robuste, generische Erkennungsmuster und manuelle Korrekturmöglichkeiten. Die Koppelung an einen konkreten Flug erfolgt über das Feld `flight_id` in `flight_documents` (siehe Kapitel 5): Wird ein Dokument direkt aus der Trip-Detailansicht hochgeladen, ist der Kontextflug bereits gesetzt; lädt der Nutzer ein Dokument aus dem globalen „Travel Documents“-Bereich, wird zunächst der Ziel-Flug gewählt und erst dann der Upload gestartet. So wird sichergestellt, dass jedes Dokument eindeutig einem Flug zugeordnet ist und später bei der Belegsuche schnell auffindbar bleibt.

#### 4.1.4. Validierung und Plausibilitätsprüfung

Alle importierten Daten (unabhängig von der Quelle) durchlaufen eine Validierungsstufe, bevor ein Flug endgültig gespeichert wird. Der erste Schritt ist eine einfache Pflichtfeld-Prüfung im Frontend:

Quellcode 4.6: Synchrones Pflichtfeld-Check im Import-Screen (add-flight-import.tsx)

```

1 const addToMyFlights = () => {
2   const newErrors: { [k: string]: boolean } = {};
3   if (!selectedFromAirport && !form.fromIata) newErrors.fromIata = true;
4   if (!selectedToAirport && !form.toIata) newErrors.toIata = true;
5   if (!form.date) newErrors.date = true;
6   setErrors(newErrors);
7
8   if (Object.keys(newErrors).length > 0) {
9     Alert.alert(
10       'Missing details',
11       'Please select airports and fill in the date.'
12     );
13     return;
14   }
15   // ...
16 };

```

---

Im Hintergrund wird die Gültigkeit der Flughäfen überprüft. Wenn der Nutzer keinen konkreten Airport aus der Suchliste ausgewählt hat, versucht Skyline, die IATA-Codes gegen eine bestehende Airport-Tabelle bzw. eine externe API aufzulösen. IATA- und ICAO-Codes sind dabei nicht frei erfunden, sondern basieren auf offiziellen Code-Listen bzw. Stammdaten [?, ?]. Sind beide Airports bekannt, berechnet Skyline mittels Haversine-Formel eine konsistente Flugdistanz [?].

Quellcode 4.7: Berechnung der Flugdistanz per Haversine-Formel (add-flight-import.tsx)

```

1 const calculateFlightDistance = (from: Airport, to: Airport): number => {
2   const toRad = (deg: number) => (deg * Math.PI) / 180;
3   const R = 6371; // Erdradius in km
4
5   const dLat = toRad(to.latitude) - from.latitude;
6   const dLon = toRad(to.longitude) - from.longitude;
7
8   const a =
9     Math.sin(dLat / 2) * Math.sin(dLat / 2) +
10    Math.cos(toRad(from.latitude)) * Math.cos(toRad(to.latitude)) *
11    Math.sin(dLon / 2) * Math.sin(dLon / 2);
12
13  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
14  return Math.round(R * c);
15};

```

---

Sind zusätzlich Abflug- und Ankunftszeiten vorhanden, werden daraus lokale Zeitstempel und eine Flugdauer berechnet. Wichtige Hilfsfunktionen sind `buildLocalTimestamp` (führt Datum und Uhrzeit zu einem ISO-ähnlichen Zeitstempel zusammen) sowie `minutesBetweenTimes` (berechnet die Dauer in Minuten). Diese Berechnungen gewährleisten, dass nur fluglogisch plausible Kombinationen aus Datum, Zeit und Distanzpersistiert werden. Schlägt einer der Schritte fehl (z. B. kein gültiger Airport auffindbar), bricht die Speicherung mit einer klaren Fehlermeldung ab; inkonsistente Einträge werden so strukturell verhindert. Darüber hinaus verarbeitet Skyline auch `Teiltreffer`: Wenn beispielsweise Flugnummer und Datum sicher erkannt wurden, aber der Zielairport unklar bleibt, kann der Nutzer im Preview-Screen den korrekten Airport aus der Suchliste auswählen, während die übrigen Felder bereits vorbefüllt sind. Die Validierungsschicht erzwingt dabei, dass am Ende alle Pflichtfelder (Abflug, Ziel, Datum) gesetzt sind, lässt aber bewusst Interpretationsspielraum bei optionalen Attributen wie Gate oder Sitzplatz.

#### 4.1.5. Feld-Mapping und Datennormalisierung

Externe Datenquellen liefern heterogene Formate (BCBP-Strings, OCR-Text, PDF-Metadaten). Skyline führt alle Informationen auf ein einheitliches internes Schema zurück, das im Typ `CreateFlightData` bzw. in der Flugentität verankert ist:

Quellcode 4.8: Vereinheitlichtes Payload-Schema für Flüge (CreateFlightData)

```
1 export interface CreateFlightData {  
2   companyId?: string;  
3   from: Airport;  
4   to: Airport;  
5   date: string;           // YYYY-MM-DD  
6   departureAt?: string;  
7   arrivalAt?: string;  
8   flightNumber?: string;  
9   airline?: string;  
10  confirmationCode?: string;  
11  bookingReference?: string;  
12  seat?: string;  
13  gate?: string;  
14  terminal?: string;  
15  status?: 'upcoming' | 'completed' | 'cancelled';  
16  notes?: string;  
17  images?: string[];  
18  duration?: string;  
19  distance?: string;  
20  distanceKm?: number;  
21 }
```

Im Import-Flow wird aus dem gemischten Kandidatenobjekt (BCBP oder OCR) und den Benutzerangaben ein konsistentes `flightPayload` erzeugt, das genau dieses Schema abbildet. Zeit- und Datumsangaben werden in stringbasierte Zeitstempel im Format `YYYY-MM-DDTHH:mm:ss` überführt. Distanz liegt sowohl als formatierter Text („1 234 km“) als

auch in numerischer Form (`distanceKm`) vor, die für spätere Statistiken und Kartenberechnungen genutzt werden kann. Dieses Feld-Mapping stellt sicher, dass unabhängig von der Herkunft der Daten (QR, OCR, Dokument-Upload) ein homogenes, relationsfreundliches Datenmodell entsteht. Gleichzeitig sorgt die Normalisierung dafür, dass andere Skyline-Module (z. B. Weltkarte, Statistiken, Benachrichtigungen) nicht wissen müssen, aus welcher Quelle die Daten ursprünglich stammen: Sie arbeiten ausschließlich mit dem konsolidierten Schema. Konzeptionell wird damit die in Kapitel 3 eingeführte ETL-Logik (Extract–Transform–Load) *auf App-Ebene* umgesetzt: Extraktion aus QR/OCR/Upload, Transformation in ein homogenes Schema, anschließendes Laden in die zentrale Datenbank.

#### 4.1.6. Deduplizierung

Mehrfachimporte eines identischen Flugs (z. B. zuerst QR-Scan, später zusätzlich PDF-Buchungsbestätigung) sind in der Praxis häufig. In der aktuellen Ausbaustufe unterscheidet Skyline klar zwischen (a) der Flug-Ebene, auf der jeder Import einen eigenen Eintrag in der Flugtabelle erzeugen könnte, und (b) der Dokumenten-Ebene, auf der mehrere Dokumente bewusst zu einem Flug gehören.

Der Import-Flow ist deshalb so gestaltet, dass in typischen Szenarien zunächst der Flug (über QR oder OCR) angelegt und anschließend weitere Dokumente über den `DocumentUploadModal` an genau diesen Flug angehängt werden. Auf Dokumentenebene werden Multiple-Uploads desselben Fluges gezielt akzeptiert; sie erscheinen im Travel-Documents-Screen gruppiert nach Flug. Eine explizite, datenbankseitige Deduplizierung auf Flug-Ebene (z. B. „existiert bereits ein Flug mit gleicher Flugnummer, Datum und Route?“) ist im aktuellen Prototyp noch nicht implementiert, aber durch das bestehende Schema technisch gut nachrüstbar. Da für Flüge ein klarer Identifikations-Kern aus Date, Carrier, Flugnummer und Route vorliegt, lässt sich Deduplizierung später auch serverseitig und datenschutzfreundlich umsetzen, ohne zusätzliche personenbezogene Daten zu speichern [?]. Für eine spätere Ausbaustufe ist zudem denkbar, Importvorgänge in einer eigenen „Import-Historie“ zu protokollieren: Jeder Import könnte dort als Event mit Hash über die Kernfelder (Datum, Flugnummer, Route) abgelegt werden. Treffen weitere Events mit identischem Hash ein (z. B. Foto und PDF derselben Buchung), könnte Skyline eine Zusammenführung anbieten („Duplikat erkannt, Dokument anhängen statt neuen Flug anlegen?“). Die aktuelle Architektur (klarer Flugschlüssel, getrennte Dokumententabelle) schafft dafür bereits die strukturelle Grundlage.

#### 4.1.7. Fehlerbehandlung und Nutzerfeedback

Der gesamte Import-Prozess ist asynchron und mit einem ausgeprägten Fehlerbehandlungskonzept versehen. Typische Fehlerquellen sind:

- fehlende Kamera- oder Dateizugriffsrechte,
- ungültige bzw. unlesbare QR-Codes,
- OCR-Fehler (keine oder fehlerhafte Texterkennung),
- Netzwerkfehler bei externen Diensten (OCR-API, Airport-API),

- Datenbankfehler beim Speichern.

Auf UI-Ebene reagiert Skyline konsistent mit klaren Meldungen und Fallback-Optionen. Beispiele sind der Hinweis auf fehlende Kameraberechtigungen, eine Fehlermeldung bei nicht erkennbaren Boardingpässen („Scan failed“) oder ein spezifischer Hinweis bei OCR-Problemen („Try a clearer photo or use manual input.“). Tritt ein Fehler auf, wird der Nutzer nie in einem halbdefinierten Zustand zurückgelassen; stattdessen kann er jederzeit auf manuelle Eingabe wechseln.

Auch der Upload von Dokumenten wird robust behandelt: Während der Upload im Hintergrund läuft, zeigt der `DocumentUploadModal` einen klaren Status („Processing...“ / „Upload continues in the background“). Schlägt der Upload fehl, erhält der Nutzer eine konkrete Fehlermeldung und kann es später erneut versuchen. Dieses Muster entspricht etablierten UX-Guidelines für Fehlermeldungen (sichtbar, konkret, mit Handlungsvorschlag) und respektiert den Aufwand, den Nutzer bereits investiert haben [?, ?]. Im Entwicklungsmodus (`__DEV__`) werden Fehler zusätzlich in der Konsole geloggt, was die Diagnose von Randfällen (z. B. exotische PDF-Layouts oder fehlerhafte Barcodes) erleichtert. In der Produktion werden hingegen nur nutzerrelevante Meldungen gezeigt; technische Details bleiben verborgen, um keine unnötigen Informationen über interne Schnittstellen preiszugeben. Damit folgt Skyline dem Prinzip „Fail Loud in Development, Fail Safe in Production“: Fehler sollen früh sichtbar werden, aber in der Live-Nutzung zu möglichst wenig Frustration führen.

#### 4.1.8. Anbindung an die Flugverwaltung

Nach erfolgreicher Validierung wird der generierte `flightPayload` an die zentrale Flugverwaltung übergeben. Der Import-Screen ruft dazu die `addFlight`-Funktion aus dem globalen Store auf; diese kümmert sich um die Persistenz in Supabase Postgres und das anschließende Aktualisieren des lokalen Zustands.

Die so gespeicherten Flüge stehen sofort in allen anderen Skyline-Modulen zur Verfügung. Dokumente, die dem Flug zugeordnet sind (Boardingpässe, Buchungsbestätigungen, Quittungen), werden im Screen `TravelDocumentsScreen` gruppiert nach Flug angezeigt. Weitere Skyline-Funktionen – wie Notizen, Checklisten (Kapitel 5), Weltkarte und Benachrichtigungen – greifen auf dieselben Flugdaten zu und profitieren indirekt von der hohen Importqualität. Technisch wird diese Koppelung dadurch erreicht, dass alle Feature-Daten (Notizen, Checklisten, Dokumente) einen Foreign Key auf `user_flights.id` besitzen (vgl. Kapitel 5). Wird ein Flug gelöscht, entfernt die Datenbank alle abhängigen Einträge per `ON DELETE CASCADE`, sodass keine verwäisteten Dokumente oder Notizen zurückbleiben. Umgekehrt bedeutet dies: Ein korrekt importierter und validierter Flug ist die Voraussetzung dafür, dass die übrigen Skyline-Funktionen zuverlässig funktionieren – ein weiterer Grund, warum der Import-Workflow so robust ausgelegt ist.

## 4.2. Ergebnis

Der in Skyline implementierte Import-Workflow zeigt, dass sich ein Großteil der im Reisealltag relevanten Flugdaten automatisiert aus vorhandenen Boardingpässen und Buchungsdokumenten gewinnen lässt. Der Nutzer wird durch klare UI, automatische Vorbefüllung und integrierte Validierung spürbar entlastet.

### 4.2.1. Bewertung der Importqualität

Die Implementierung erreicht bei QR-basierten Imports (BCBP) eine sehr hohe Trefferquote; Flugnummer, Datum und Route können in vielen Fällen vollständig ohne manuelle Nacharbeit übernommen werden [?]. Abweichungen vom BCBP-Standard werden durch Fallback-Logik (Suche nach M1-Segment, defensive String-Operationen) teilweise kompensiert und führen im Zweifel zu einem kontrollierten Abbruch mit Fehlermeldung statt zu falschen Daten.

Die OCR-basierte Auswertung von Bildern und PDFs ist naturgemäß anfälliger für Fehler, da sie von Bildqualität und Layout der Dokumente abhängt [?, ?]. Durch die Kombination aus robusten Heuristiken (`extractFlightData`), Validierung von IATA-Codes und Datum sowie manueller Kontrolle im Preview-Screen werden dennoch in den meisten Fällen korrekte oder leicht korrigierbare Vorschläge erzeugt. Geringe Nachbearbeitung und seltene manuelle Korrekturen bestätigen die Wirksamkeit der Validierungs- und Normalisierungsschicht. Empirisch zeigt sich im Projektverlauf, dass BCBP-Scans den Großteil der Flüge nahezu fehlerfrei abdecken, während OCR-basierte Importe vor allem bei schwierigen Layouts (mehrsprachige Tickets, stark grafische PDFs) manuelle Nacharbeit erfordern. Diese Beobachtungen decken sich mit der Literatur zu OCR-Qualität und der Empfehlung, Barcodes dort zu bevorzugen, wo sie verfügbar sind [?, ?].

### 4.2.2. Nutzen im Reise-Workflow

Nutzer können Flüge deutlich schneller erfassen, da Informationen nicht mehr von Hand abgetippt werden müssen. Bereits wenige Sekunden nach dem Eintreffen einer Buchungsbestätigung oder dem Erhalt des Boardingpasses kann der Flug strukturiert in Skyline vorliegen; nachgelagerte Funktionen (Checklisten, Notizen, Erinnerungen, Dokumentenablage) stehen damit frühzeitig zur Verfügung.

Die Kombination aus QR-Scan, OCR-Auswertung und flexiblem Dokumenten-Upload deckt unterschiedliche Buchungs- und Reiseabläufe ab – vom klassischen Papier-Boardingpass über mobile Wallet-Tickets bis hin zu reinen E-Mail-Bestätigungen. Insbesondere im Geschäftsreisekontext erleichtert dies die spätere Reisekostenabrechnung, da Boardingpässe, Buchungsbestätigungen und Quittungen direkt im Flugkontext abgelegt sind.

### 4.2.3. Grenzen und Verbesserungspotenzial

Nicht alle Airlines und Dokumentlayouts sind vollständig abgedeckt; insbesondere exotische oder stark grafische Tickets stellen OCR und Heuristiken vor Herausforderungen. Auch eine

direkte E-Mail-Integration (z. B. über IMAP oder spezialisierte APIs) ist im aktuellen Stand aus Datenschutz- und Komplexitätsgründen nicht implementiert [?, ?].

Perspektivisch denkbar sind:

- robustere Parser mit Template-Unterstützung für häufig genutzte Airlines,
- eine optionale, explizit zustimmungsbasierte Anbindung an E-Mail-Konten,
- eine datenbankseitige Deduplizierung von Flügen über Flugnummer, Datum und Route,
- weitergehende Internationalisierung (weitere Sprachen, alternative Datumsformate),
- verbesserte OCR-Pfade, etwa durch alternative Engines oder lokale OCR auf dem Gerät [?, ?].

Trotz dieser Grenzen liefert die aktuelle Implementierung des automatischen Imports ein tragfähiges Fundament, das den Reisealltag von Vielreisenden spürbar vereinfacht und zugleich sauber in das zentrale Datenmodell von Skyline integriert ist.

## 5. Implementierung: Zentrale & sichere Datenverwaltung (JanOle)

Dieses Kapitel beschreibt die konkrete Umsetzung der zentralen Datenverwaltung in Skyline. Die theoretischen Grundlagen (Single Source of Truth, Sicherheitskonzept, Anforderungen) werden in Kapitel 3 behandelt.

### 5.1. Implementierung in Skyline

Die zentrale Datenverwaltung in Skyline basiert auf Supabase Postgres als relationalem Kern und Supabase Storage für dateibasierte Inhalte [?]. Im fachlichen Zentrum steht die Flugentität: Jeder Flug bildet einen konkreten Reisekontext, an den Notizen, Checklisten und Dokumente angehängt werden. Dadurch entsteht in Skyline kein loses Nebeneinander verschiedener Feature-Daten, sondern ein zusammenhängendes Datenmodell, in dem jede Zusatzinformation eindeutig einem Flug zugeordnet ist.

Die Kernoperationen (Create, Read, Update, Delete) werden nicht direkt in den UI-Screens implementiert, sondern über einen Service-Layer gekapselt (`services/supabase.ts`, `services/documentService.ts`). Sicherheitsseitig wird Row-Level Security (RLS) auf Tabellenebene eingesetzt, sodass Zugriffe bereits innerhalb der Datenbank an den authentifizierten Benutzerkontext gebunden sind. Für den Teamkontext wurde das Modell migrationsbasiert erweitert: Über `company_id` können persönliche Flüge und Firmenflüge getrennt modelliert werden. Mit `departure_at` und `arrival_at` wurden zeitbezogene Felder eingeführt, die für Live-Progress auf der Karte sowie zeitbasierte Erinnerungslogik genutzt werden.

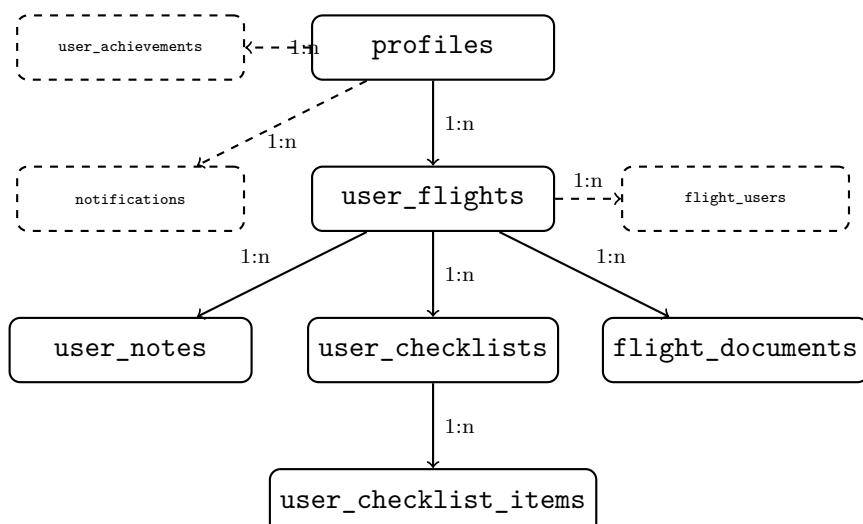


Abbildung 5.1.: Zentrales Datenmodell in Skyline (vereinfachte ER-Darstellung; gestrichelt: ergänzende Feature-Tabellen)

### 5.1.1. Datenmodell & Tabellenstruktur

Das relationale Kernmodell von Skyline besteht aus sechs operativen Tabellen und einer zentralen Stammdatentabelle. Im Folgenden werden alle Tabellen mit ihrer Struktur, ihren Constraints und ihrer Rolle im Gesamtmodell beschrieben.

#### Tabelle profiles

profiles bildet den nutzerbezogenen Einstiegspunkt. Die Tabelle ist direkt mit dem internen Supabase-Auth-Kontext verknüpft: `id` ist kein eigenständiger UUID-Wert, sondern ein Foreign Key auf `auth.users(id)` – das heißt, jeder Supabase-Nutzeraccount besitzt genau einen Profilsatz.

Die Verknüpfung wird automatisch über einen Datenbank-Trigger hergestellt. Listing 5.1 zeigt Funktion und Trigger, die bei jeder Neuregistrierung ausgeführt werden:

```
1 CREATE OR REPLACE FUNCTION public.handle_new_user()
2 RETURNS TRIGGER AS $$ 
3 BEGIN
4     INSERT INTO public.profiles (id, full_name, avatar_url)
5     VALUES (
6         NEW.id,
7         COALESCE(NEW.raw_user_meta_data->>'name', NEW.email),
8         NEW.raw_user_meta_data->>'avatar_url',
9     )
10    ON CONFLICT (id) DO NOTHING;
11    RETURN NEW;
12 END;
13 $$ LANGUAGE plpgsql SECURITY DEFINER;
14
15 CREATE TRIGGER on_auth_user_created
16     AFTER INSERT ON auth.users
17     FOR EACH ROW EXECUTE FUNCTION public.handle_new_user();
```

Quellcode 5.1: Automatische Profilanlage via Trigger  
(complete\_working\_schema.sql)

Das Feld `account_type` unterscheidet zwischen Einzelnutzern (`worker`) und Firmenkonten (`company`) und steuert, welche Features in der App sichtbar sind. `preferences` erlaubt das Speichern nutzerspezifischer Einstellungen als strukturlose JSON-Daten ohne Schemaänderung. Zusätzlich zur Eigen-Policy ist eine weitere RLS-Policy aktiv, die das Lesen von Profilen anderer Mitglieder derselben Firma erlaubt – notwendig für die Team-Detailansicht im Firmenkontext:

```
1 CREATE POLICY "Read profiles of same-company members"
2     ON public.profiles FOR SELECT TO authenticated
3     USING (
4         EXISTS (
5             SELECT 1
```

Feld	Typ	Bedeutung
id	UUID	PK & FK auf auth.users(id)
full_name	TEXT	Anzeigename (aus Auth-Metadaten)
avatar_url	TEXT	Profilbild-URL
account_type	TEXT	Kontotyp: <code>worker</code> (Default) oder <code>company</code>
invite_code	TEXT UNIQUE	Eindeutiger Einladungscode für Firmenbeitritt
preferences	JSONB	Nutzereinstellungen (Default: {})
created_at	TIMESTAMPTZ	Erstellungszeitpunkt
updated_at	TIMESTAMPTZ	Letzter Änderungszeitpunkt (Trigger)

Tabelle 5.1.: Felder der Tabelle `profiles`

Feld	Typ	Bedeutung
id	BIGINT (BIGSERIAL)	Primärschlüssel (auto-increment)
iata	TEXT (UNIQUE)	IATA-Flughafencode (z. B. VIE)
icao	TEXT (UNIQUE)	ICAO-Flughafencode (z. B. LOWW)
name	TEXT	Vollständiger Flughafenname
city	TEXT	Stadt
country	TEXT	Land
latitude	NUMERIC	Geografische Breite
longitude	NUMERIC	Geografische Länge
timezone	TEXT	Zeitzone (z. B. Europe/Vienna)

Tabelle 5.2.: Felder der Tabelle `airports`

```

6   FROM public.company_members me
7   JOIN public.company_members other
8     ON other.company_id = me.company_id
9     WHERE me.profile_id = auth.uid()
10    AND other.profile_id = public.profiles.id
11  )
12 );

```

Quellcode 5.2: Sichtbarkeit von Profilen innerhalb einer Company (supabasesql.txt)

### Tabelle `airports`

`airports` enthält die globale Flughafenstammdatenbank. IATA- und ICAO-Codes sind als UNIQUE-Constraints definiert, da sie als natürliche Schlüssel für die Suche genutzt werden. Koordinaten (`latitude`, `longitude`) fließen direkt in die Kartenansicht und die Distanzberechnung (`distance_km` in `user_flights`) ein. RLS-seitig ist die Tabelle öffentlich lesbar (auch für nicht eingeloggte Nutzer), da Flughafendaten kein schutzbedürftiges Gut darstellen. Authenticated User dürfen zusätzlich neue Einträge anlegen, was die spätere Erweiterung der Datenbank durch Community-Beiträge ermöglicht.

### Tabelle `user_flights`

`user_flights` ist der operative Kern des Datenmodells. Jeder Eintrag repräsentiert einen konkreten Flug eines Nutzers und verknüpft Airline, Flugnummer, Datum und Status mit den Foreign Keys auf Abflug- und Zielairport. Über `company_id` kann ein Flug optional einer Firma zugeordnet werden (Teamkontext); ist das Feld `NONE`, handelt es sich um einen privaten Flug.

Das folgende SQL-Listing zeigt die Kerndefinition aus `complete_working_schema.sql`:

```
1 CREATE TABLE IF NOT EXISTS public.user_flights (
2     id             UUID      PRIMARY KEY DEFAULT gen_random_uuid(),
3     profile_id    UUID      NOT NULL
4                           REFERENCES public.profiles(id) ON DELETE
5                           CASCADE,
6     from_airport_id BIGINT   REFERENCES public.airports(id),
7     to_airport_id   BIGINT   REFERENCES public.airports(id),
8     flight_number   TEXT,
9     airline         TEXT,
10    date            DATE     NOT NULL,
11    departure_at    TIMESTAMP -- Lokale Abflugzeit (Karte & Reminder)
12    arrival_at      TIMESTAMP -- Lokale Ankunftszeit (Karte & Reminder)
13    status          TEXT    CHECK (status IN ('upcoming', 'completed', 'cancelled'))
14                           DEFAULT 'upcoming',
15    company_id      UUID      REFERENCES public.companies(id) ON DELETE SET
16                           NULL,
17    distance_km     NUMERIC,
18    created_at      TIMESTAMPTZ DEFAULT NOW(),
19    updated_at      TIMESTAMPTZ DEFAULT NOW()
20 );
```

Quellcode 5.3: Kerndefinition `user_flights` (`complete_working_schema.sql`)

#### TODO: Screenshot einfügen

Supabase Dashboard → Table Editor → Tabelle `user_flights`  
(Spaltenübersicht mit Typen, FKs und Constraints sichtbar)

Abbildung 5.2.: Tabellenstruktur von `user_flights` im Supabase Table Editor

Tabelle 5.3 fasst die wichtigsten Felder zusammen:

Die RLS-Policies für `user_flights` sichern alle vier CRUD-Operationen ausschließlich auf Basis der `profile_id`:

Feld	Typ	Bedeutung
<i>Kerndaten</i>		
<code>id</code>	UUID	Primärschlüssel
<code>profile_id</code>	UUID	FK auf <code>profiles</code> (NOT NULL)
<code>from_airport_id</code>	BIGINT	FK auf <code>airports</code> (Abflug)
<code>to_airport_id</code>	BIGINT	FK auf <code>airports</code> (Ziel)
<code>flight_number</code>	TEXT	Flugnummer (z. B. OS123)
<code>airline</code>	TEXT	Airline-Name
<code>date</code>	DATE	Flugdatum (NOT NULL)
<code>status</code>	TEXT	<code>upcoming/completed/cancelled</code>
<i>Buchungs- und Logistikdaten</i>		
<code>confirmation_code</code>	TEXT	Buchungsbestätigungscode
<code>booking_reference</code>	TEXT	Buchungsreferenz (PNR)
<code>seat</code>	TEXT	Sitzplatznummer
<code>gate</code>	TEXT	Gate-Bezeichnung
<code>terminal</code>	TEXT	Terminal
<code>duration</code>	TEXT	Flugdauer (textuell, z. B. 2h 15min)
<i>Zeitfelder (migrationsbasiert ergänzt)</i>		
<code>departure_at</code>	TIMESTAMP	Lokale Abflugzeit (Karte & Reminder)
<code>arrival_at</code>	TIMESTAMP	Lokale Ankunftszeit (Karte & Reminder)
<i>Weitere Felder</i>		
<code>company_id</code>	UUID	Optional: Firmenflug-Zuordnung (FK auf <code>companies</code> )
<code>distance_km</code>	INTEGER	Berechnete Flugdistanz in km
<code>notes</code>	TEXT	Legacy-Notizfeld (vor Einführung von <code>user_notes</code> )
<code>images</code>	ARRAY	Optionale Bildverweise

Tabelle 5.3.: Felder der Tabelle `user_flights`

```

1 CREATE POLICY "Users can view own user_flights"
2   ON public.user_flights FOR SELECT TO authenticated
3     USING (profile_id = auth.uid());
4
5 CREATE POLICY "Users can insert own user_flights"
6   ON public.user_flights FOR INSERT TO authenticated
7     WITH CHECK (profile_id = auth.uid());
8
9 CREATE POLICY "Users can update own user_flights"
10  ON public.user_flights FOR UPDATE TO authenticated
11    USING (profile_id = auth.uid())
12    WITH CHECK (profile_id = auth.uid());
13
14 CREATE POLICY "Users can delete own user_flights"
15   ON public.user_flights FOR DELETE TO authenticated
16     USING (profile_id = auth.uid());

```

---

Quellcode 5.4: RLS-Policies für `user_flights` (supabasesql.txt)

Das Muster ist für alle nutzerbezogenen Tabellen identisch: `USING` prüft beim Lesen, Ändern und Löschen den bestehenden Datensatz, `WITH CHECK` verhindert beim Schreiben, dass eine fremde `profile_id` eingetragen wird. Damit ist ein Zugriff auf Flüge anderer Nutzer auf Datenbankebene strukturell ausgeschlossen – unabhängig davon, was im App-Code passiert.

#### Tabellen `user_notes`, `user_checklists` und `user_checklist_items`

Für flugbezogene Zusatzinhalte werden drei weitere Tabellen verwendet. `user_notes` speichert freie Textnotizen, `user_checklists` enthält die Kopfdaten einer abhakbaren Aufgabenliste, und `user_checklist_items` die einzelnen Einträge dazu. Beide Notiz- und Checklistentabellen besitzen ein `purpose`-Feld ('`business`' oder '`private`'), das den Nutzungskontext kennzeichnet und die Filterfunktion in der App ermöglicht. Ein optionales `reminder_at`-Feld erlaubt zeitbasierte Erinnerungen.

Feld	Typ	Bedeutung
<code>id</code>	UUID	Primärschlüssel
<code>profile_id</code>	UUID	FK auf <code>profiles</code> (Eigentümer)
<code>flight_id</code>	UUID	FK auf <code>user_flights</code> (ON DELETE CASCADE)
<code>purpose</code>	TEXT	<code>business</code> oder <code>private</code>
<code>title</code>	TEXT	Notiztitel (Default: 'Trip Note')
<code>content</code>	TEXT	Freier Notizinhalt
<code>reminder_at</code>	TIMESTAMPTZ	Optionaler Erinnerungszeitpunkt
<code>created_at</code>	TIMESTAMPTZ	Erstellungszeitpunkt
<code>updated_at</code>	TIMESTAMPTZ	Letzter Änderungszeitpunkt (Trigger)

Tabelle 5.4.: Felder der Tabelle `user_notes`

Feld	Typ	Bedeutung
<i>user_checklists (Kopfdaten)</i>		
id	UUID	Primärschlüssel
profile_id	UUID	FK auf profiles
flight_id	UUID	FK auf user_flights (ON DELETE CASCADE)
purpose	TEXT	business oder private
title	TEXT	Name der Checkliste
reminder_at	TIMESTAMPTZ	Optionaler Erinnerungszeitpunkt
<i>user_checklist_items (Einzeleinträge)</i>		
id	UUID	Primärschlüssel
checklist_id	UUID	FK auf user_checklists (ON DELETE CASCADE)
text	TEXT	Beschriftung des Eintrags
checked	BOOLEAN	Abhak-Status (Default: false)
order_idx	INTEGER	Reihenfolge innerhalb der Checkliste

Tabelle 5.5.: Felder der Tabellen `user_checklists` und `user_checklist_items`

Die RLS-Policies für `user_checklist_items` folgen einem indirekten Muster: Da `user_checklist_items` keine eigene `profile_id` trägt, wird die Zugehörigkeit über die übergeordnete Checkliste geprüft:

---

```

1 CREATE POLICY "items_select_via_parent"
2   ON public.user_checklist_items FOR SELECT
3     USING (
4       EXISTS (
5         SELECT 1 FROM public.user_checklists c
6           WHERE c.id = checklist_id
7             AND c.profile_id = auth.uid()
8       )
9     );

```

---

Quellcode 5.5: Indirekte RLS-Policy für `user_checklist_items`  
(`complete_working_schema.sql`)

## Indizes

Auf allen frequentierten Lookup-Spalten sind explizite Indizes angelegt. Listing 5.6 zeigt die Indizes für die Kernabfragen (Flüge per Nutzer, Notizen per Flug, Checklisten per Flug):

---

```

1 -- user_flights
2 CREATE INDEX IF NOT EXISTS user_flights_profile_id_idx
3   ON public.user_flights(profile_id);
4 CREATE INDEX IF NOT EXISTS user_flights_date_idx
5   ON public.user_flights(date);

```

---

```
6 CREATE INDEX IF NOT EXISTS user_flights_company_id_idx
7   ON public.user_flights(company_id);
8
9 -- user_notes
10 CREATE INDEX IF NOT EXISTS user_notes_flight_id_idx
11   ON public.user_notes(flight_id);
12 CREATE INDEX IF NOT EXISTS user_notes_reminder_at_idx
13   ON public.user_notes(reminder_at);
14
15 -- user_checklists
16 CREATE INDEX IF NOT EXISTS user_checklists_flight_id_idx
17   ON public.user_checklists(flight_id);
18
19 -- user_checklist_items
20 CREATE INDEX IF NOT EXISTS user_checklist_items_checklist_id_idx
21   ON public.user_checklist_items(checklist_id);
22
23 -- airports (Suchfelder)
24 CREATE INDEX IF NOT EXISTS airports_iata_idx ON public.airports(iata);
25 CREATE INDEX IF NOT EXISTS airports_icao_idx ON public.airports(icao);
26 CREATE INDEX IF NOT EXISTS airports_name_idx ON public.airports(name);
```

Quellcode 5.6: Performance-Indizes auf den Kerntabellen  
(complete\_working\_schema.sql)

## Migrationsbasierte Schemaerweiterung

Das Schema wurde iterativ durch separate Migrationsskripte erweitert. Listing 5.7 zeigt exemplarisch die Migration, die `departure_at` und `arrival_at` zu `user_flights` hinzugefügt hat:

```
1 ALTER TABLE public.user_flights
2   ADD COLUMN IF NOT EXISTS departure_at TIMESTAMP ,
3   ADD COLUMN IF NOT EXISTS arrival_at      TIMESTAMP ;
4
5 CREATE INDEX IF NOT EXISTS user_flights_departure_at_idx
6   ON public.user_flights(departure_at);
7 CREATE INDEX IF NOT EXISTS user_flights_arrival_at_idx
8   ON public.user_flights(arrival_at);
```

Quellcode 5.7: Migration: Zeitfelder in `user_flights`  
(migration\_add\_flight\_times.sql)

Dieser Ansatz gewährleistet, dass bestehende Daten bei Schemaänderungen erhalten bleiben (`IF NOT EXISTS / IF EXISTS`) und neue Felder rückwärtskompatibel eingeführt werden können. `company_id` wurde analog per eigenem Migrationsskript ergänzt (`migration_add_company_id_to_u`

## Weitere Tabellen im Gesamtschema

Über den flugbezogenen Kern hinaus enthält das Skyline-Schema weitere Tabellen, die ergänzende Features abbilden. Tabelle 5.6 gibt einen Überblick:

Tabelle	Zweck
<code>companies</code>	Firmen mit <code>owner_profile_id</code> und <code>invite_code</code> ; Basis für den Teamkontext
<code>company_members</code>	Mitgliedschaft eines Nutzers in einer Firma ( <code>role: owner oder worker</code> )
<code>company_invites</code>	Einladungstoken mit Status, Ablaufzeit und Annahme-Tracking ( <code>pending/accepted/ expired/revoked</code> )
<code>flight_users</code>	Zuweisung von Nutzern zu einem Firmenflug ( <code>can_edit, status</code> )
<code>notifications</code>	Geplante Push-Benachrichtigungen mit <code>fire_at, kind</code> und <code>payload</code> (JSONB)
<code>achievements</code>	Globale Achievement-Definitionen (öffentlich lesbar, <code>anon + authenticated</code> )
<code>user_achievements</code>	Freigeschaltete Achievements pro Nutzer mit <code>unlocked_at</code> und <code>metadata</code>
<code>email_accounts</code>	Verknüpfte E-Mail-Accounts ( <code>gmail/outlook</code> ) für den automatischen Flugimport
<code>email_imports</code>	Importierte Mails mit Parse-Status ( <code>pending/parsed/failed</code> ) und FK auf den erkannten Flug
<code>calendar_accounts</code>	Verknüpfte Kalender-Accounts für die Kalenderintegration

Tabelle 5.6.: Weitere Tabellen im Skyline-Datenbankschema

Diese Tabellen sind vollständig RLS-gesichert und folgen demselben Eigentümerprinzip wie die Kerntabellen: Jeder Nutzer sieht und bearbeitet ausschließlich eigene Datensätze. Eine Ausnahme bilden `achievements`, die global lesbar sind, da sie keine personenbezogenen Daten enthalten.

### 5.1.2. Service-Layer-Architektur

Die CRUD-Operationen auf den Datenbanktabellen werden in Skyline nicht direkt in den UI-Komponenten ausgeführt, sondern über einen dedizierten Service-Layer gekapselt. Dieser Layer besteht aus zwei zentralen Dateien: `services/supabase.ts` übernimmt alle Lese- und Schreiboperationen auf den relationalen Tabellen (Flüge, Notizen, Checklisten), während `services/documentService.ts` die dateibasierten Operationen auf dem Storage-Bucket verwaltet (Upload, Signed-URL-Erzeugung, Caching, Rollback).

Diese Trennung verfolgt drei Ziele:

- **Single Responsibility:** Jede UI-Komponente ist ausschließlich für die Darstellung zuständig und kennt keine Datenbankdetails. Änderungen am Schema oder an der Supabase-Abfragelogik erfordern nur Anpassungen im Service, nicht in den Screens.

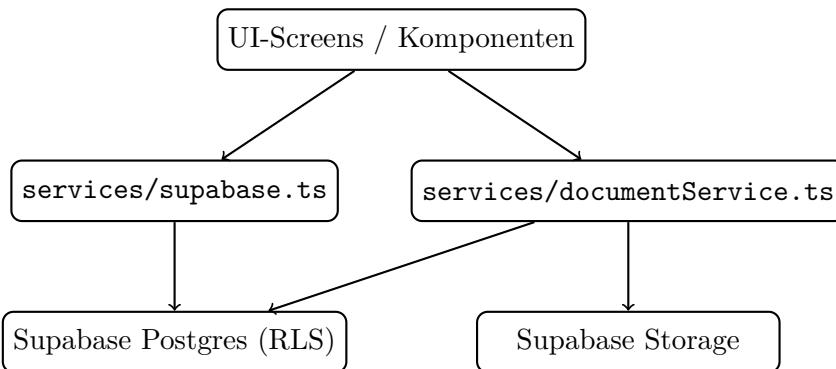


Abbildung 5.3.: Service-Layer zwischen UI und Datenhaltung in Skyline

- **Konsistente Fehlerbehandlung:** Rollback-Logik (z. B. Löschen einer hochgeladenen Datei bei DB-Fehler), Timeout-Fallbacks und URL-Refresh-Mechanismen sind zentral im Service implementiert und stehen allen Screens zur Verfügung.
- **Testbarkeit:** Der Service-Layer kann isoliert getestet werden, ohne dass eine vollständige UI gerendert werden muss.

Aus Sicht der Datenverwaltung ist besonders relevant, dass der Service-Layer die RLS-Grenzen der Datenbank *nicht* umgehen kann und auch nicht soll. Selbst wenn eine fehlerhafte UI-Komponente eine Abfrage ohne `profile_id`-Filter absetzt, greift die RLS-Policy auf Datenbankebene und gibt ausschließlich die eigenen Datensätze zurück. Der Service-Layer ist damit kein Sicherheitsmechanismus, sondern eine Schicht für Wartbarkeit und Konsistenz – die eigentliche Sicherheit liegt in der Datenbank selbst.

### 5.1.3. Beziehung Flight ↔ Notes, Docs & Checklists

Die Verknüpfung zwischen Flugstammdaten und Zusatzinformationen erfolgt in Skyline konsistent über `flight_id`. Sowohl `user_notes` als auch `user_checklists` und `flight_documents` referenzieren jeweils genau einen Datensatz in `user_flights`.

Technisch ist diese Beziehung als Foreign Key mit `ON DELETE CASCADE` umgesetzt. Listing 5.8 zeigt dies exemplarisch für `flight_documents`:

```

1 CREATE TABLE IF NOT EXISTS public.flight_documents (
2     id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
3     flight_id   UUID NOT NULL
4             REFERENCES public.user_flights(id) ON DELETE CASCADE,
5     profile_id UUID NOT NULL
6             REFERENCES public.profiles(id)      ON DELETE CASCADE,
7     file_name   TEXT NOT NULL,
8     file_type   TEXT NOT NULL
9             CHECK (file_type IN ('pdf', 'image', 'other')),
10    mime_type   TEXT NOT NULL,
11    file_size   BIGINT NOT NULL,
12    storage_path TEXT NOT NULL,
13    storage_bucket TEXT NOT NULL DEFAULT 'flight-documents',
  
```

```
14     signed_url          TEXT,  
15     signed_url_expires_at TIMESTAMPTZ,  
16     document_type TEXT  
17     CHECK (document_type IN  
18         ('boarding_pass', 'booking_confirmation', 'receipt', 'other'))  
19     DEFAULT 'other',  
20     is_cached BOOLEAN DEFAULT FALSE,  
21     cache_path TEXT,  
22     uploaded_at TIMESTAMPTZ DEFAULT NOW()  
23 );
```

---

Quellcode 5.8: Foreign Key mit ON DELETE CASCADE in `flight_documents` (migration\_add\_flight\_documents.sql)

Für alle Tabellen sind RLS-Policies definiert, die sicherstellen, dass ein Nutzer ausschließlich eigene Datensätze lesen und schreiben kann. Listing 5.9 zeigt die vier Policies für `user_notes`:

```
1 ALTER TABLE public.user_notes ENABLE ROW LEVEL SECURITY;  
2  
3 CREATE POLICY "notes_select_own" ON public.user_notes  
4   FOR SELECT USING (profile_id = auth.uid());  
5  
6 CREATE POLICY "notes_insert_own" ON public.user_notes  
7   FOR INSERT WITH CHECK (profile_id = auth.uid());  
8  
9 CREATE POLICY "notes_update_own" ON public.user_notes  
10  FOR UPDATE USING (profile_id = auth.uid());  
11  
12 CREATE POLICY "notes_delete_own" ON public.user_notes  
13  FOR DELETE USING (profile_id = auth.uid());
```

---

Quellcode 5.9: Row-Level-Security-Policies für `user_notes` (complete\_working\_schema.sql)

#### TODO: Screenshot einfügen

Supabase Dashboard → Authentication → Policies  
(RLS-Policy-Liste für Tabelle `user_notes` oder `user_flights`)

Abbildung 5.4.: Row-Level-Security-Policies im Supabase Dashboard

In der App-Logik wird die Beziehung ebenfalls explizit genutzt. Listing 5.10 zeigt die Abfrage von Notizen pro Flug in `services/supabase.ts`:

```

1  async getNotes(flightId: string): Promise<Note[]> {
2    const user = (await this.supabase.auth.getUser()).data.user;
3    if (!user) return [];
4
5    const { data, error } = await this.supabase
6      .from('user_notes')
7      .select('id, profile_id, flight_id, purpose,
8              title, content, reminder_at,
9              created_at, updated_at')
10     .eq('flight_id', flightId) // nur dieser Flug
11     .eq('profile_id', user.id) // nur eigene Notizen
12     .order('created_at', { ascending: true });
13
14   if (error) throw error;
15   return (data || []).map((n): Note => ({ ... }));
16 }
  
```

Quellcode 5.10: Flugbezogene Notizen-Abfrage mit `profile_id`-Filter  
(services/supabase.ts)

Die Detailansicht (`trip-details.tsx`) stößt beim Öffnen alle abhängigen Ladevorgänge parallel an:

```

1  void Promise.allSettled([
2    loadNotesForFlight(id),
3    loadChecklistsForFlight(id),
4    loadTemplatesByPurpose('private'),
5  ]);
  
```

Quellcode 5.11: Paralleles Vorladen in der Trip-Detailansicht (app/trip-details.tsx)

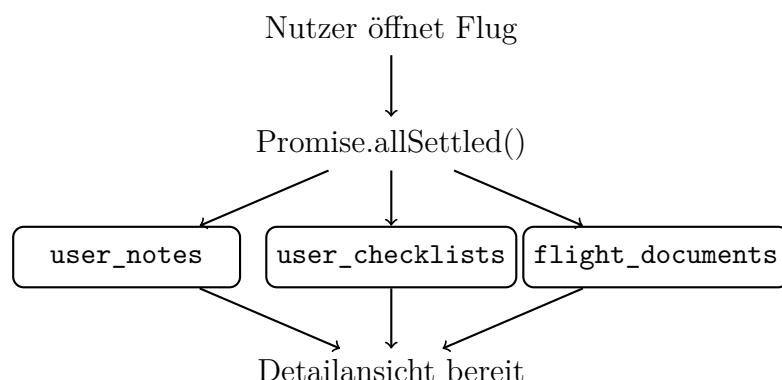


Abbildung 5.5.: Paralleles Laden flugbezogener Daten beim Öffnen der Trip-Detailansicht

#### 5.1.4. Storage-Bucket & Signed URLs

Dateibasierte Inhalte – Boardingpässe, Buchungsbestätigungen, Belege – werden im Supabase-Storage-Bucket `flight-documents` abgelegt. Der Bucket ist privat konfiguriert; öffentlicher Dauerzugriff ist nicht möglich. Jedes Dokument erhält einen strukturierten Pfad, der in `DocumentService.generateStoragePath()` berechnet wird:

```
1 private generateStoragePath(
2   userId: string, flightId: string, fileName: string
3 ): string {
4   const timestamp = Date.now();
5   const sanitized = fileName.replace(/[^a-zA-Z0-9.-]/g, '_');
6   return `${userId}/${flightId}/${timestamp}_${sanitized}`;
7 }
```

Quellcode 5.12: Pfadgenerierung im Storage-Bucket (services/documentService.ts)

Nach dem Upload wird die Signed URL sofort generiert und zusammen mit den Metadaten in `flight_documents` gespeichert. Listing 5.13 zeigt den Upload- und URL-Erzeugungsschritt:

```
1 // Datei hochladen
2 const { error: uploadError } = await supabase.storage
3   .from(this.bucketName)           // 'flight-documents'
4   .upload(storagePath, fileData, {
5     cacheControl: '3600',
6     contentType: mimeType,
7     upsert: false,
8 });
9
10 // Signed URL erzeugen (gueltig: 3600 Sekunden)
11 const { data: signedUrlData } = await supabase.storage
12   .from(this.bucketName)
13   .createSignedUrl(storagePath, this.signedUrlExpirationSeconds);
14
15 // Metadaten + URL in DB speichern
16 await supabase.from('flight_documents').insert({
17   flight_id:                 flightId,
18   storage_path:               storagePath,
19   signed_url:                signedUrlData?.signedUrl,
20   signed_url_expires_at:    expiresAt.toISOString(),
21   document_type:             documentType,
22   ...
23 });
```

Quellcode 5.13: Upload und Signed-URL-Erzeugung (services/documentService.ts)

Ist eine gespeicherte URL abgelaufen, wird sie in `getDocumentUrl()` automatisch erneuert (Listing 5.14):

```
1 // Pruefen ob URL noch gueltig
2 if (doc.signed_url && doc.signed_url_expires_at) {
```

```

3   if (new Date(doc.signed_url_expires_at) > new Date()) {
4     return doc.signed_url; // noch gültig
5   }
6 }
7
8 // Neue URL generieren und in DB aktualisieren
9 const { data: signedUrlData } = await supabase.storage
10   .from(this.bucketName)
11   .createSignedUrl(doc.storage_path, this.signedUrlExpirationSeconds);
12
13 await supabase.from('flight_documents').update({
14   signed_url: signedUrlData.signedUrl,
15   signed_url_expires_at: expiresAt.toISOString(),
16 }) .eq('id', documentId);
  
```

Quellcode 5.14: Automatischer Signed-URL-Refresh (services/documentService.ts)

**TODO: Screenshot einfügen**

Supabase Dashboard → Storage → Bucket **flight-documents**  
 (Ordnerstruktur `userId/flightId/...` mit hochgeladenen Dateien sichtbar)

Abbildung 5.6.: Storage-Bucket **flight-documents** im Supabase Dashboard

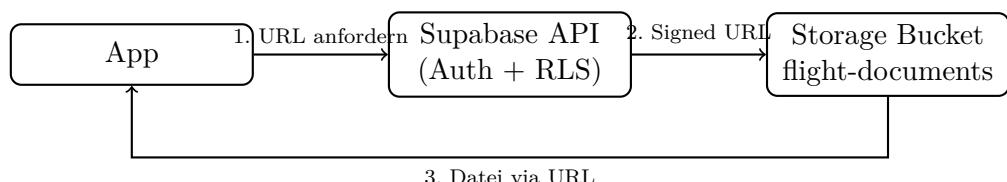


Abbildung 5.7.: Dokumentenzugriff über Signed URLs in Skyline

### 5.1.5. Sync-Strategie & Caching

Die Synchronisationsstrategie von Skyline ist auf schnelle Startzeit bei gleichzeitig konsistentem Online-Datenstand ausgelegt. Beim App-Start ruft `_layout.tsx` zunächst `loadFlights()` auf. Anschließend werden für die ersten 12 Flüge Notizen und Checklisten vorab geladen (*Warmup*), damit die Detailansicht ohne Erstladeverzögerung öffnet:

```

1 const flightIds = useAppState.getState().flights
2   .map(f => f?.id)
3   .filter(id => typeof id === 'string')
  
```

```
4     .slice(0, 12); // max. 12 Flüge vorladen
5
6 const preloadTasks: Promise<any>[] = [];
7 for (const flightId of flightIds) {
8     preloadTasks.push(loadNotesForFlight(flightId));
9     preloadTasks.push(loadChecklistsForFlight(flightId));
10 }
11 await Promise.allSettled(preloadTasks);
```

Quellcode 5.15: Warmup beim App-Start: paralleles Vorladen von Notes und Checklists (app/\_layout.tsx)

Dokumente können lokal gecacht werden. Der Cache-Zustand wird in `is_cached` und `cache_path` in der Tabelle `flight_documents` mitgeführt. Beim nächsten Zugriff prüft die App zunächst den lokalen Cache, bevor eine Signed URL angefordert wird. Fachliche Kerndaten (Flüge, Notizen, Checklisten) werden primär aus Supabase bezogen; lokale Caches dienen der Beschleunigung, nicht als unabhängiger Primärspeicher.

### 5.1.6. Fehlerfälle & Wiederherstellung

Die Implementierung enthält mehrere Schutzmechanismen, um Fehlerfälle ohne Datenverlust und ohne instabile UI-Zustände zu behandeln.

**Storage-Rollback beim Upload:** Schlägt nach erfolgreichem Storage-Upload die Metadatenspersistierung in der Datenbank fehl, löscht DocumentService die bereits hochgeladene Datei wieder:

```
1 if (dbError) {
2     // Cleanup: hochgeladene Datei entfernen,
3     // damit kein Datei-Eintrag ohne DB-Referenz bleibt
4     await supabase.storage
5         .from(this.bucketName)
6         .remove([storagePath]);
7     throw new Error(
8         'Failed to save document metadata: ${dbError.message}'
9     );
10 }
```

Quellcode 5.16: Storage-Rollback bei DB-Fehler (services/documentService.ts)

**Cache-Invalidierung:** Ist ein als lokal markiertes Dokument nicht mehr am erwarteten Pfad vorhanden, wird der Cache-Status in den Metadaten zurückgesetzt:

```
1 const fileInfo = await FileSystem.getInfoAsync(doc.cache_path);
2 if (!fileInfo.exists) {
3     // Cache ungültig -> in DB zurücksetzen
4     await supabase.from('flight_documents').update({
5         is_cached: false,
6         cache_path: null,
```

```
7     }).eq('id', documentId);
8     return null; // App laedt anschliessend aus Online-Bestand
9 }
```

Quellcode 5.17: Cache-Validierung und -Reset (services/documentService.ts)

**UI-Timeout in der Dokumentliste:** DocumentList.tsx setzt einen 10-Sekunden-Timeout, um Endlosschleifen bei Ladeindikatoren zu verhindern:

```
1 // 10-Sekunden-Timeout verhindert haengenden Ladeindikator
2 timeoutId = setTimeout(() => {
3     if (isMountedRef.current) {
4         setLoading(false);
5         setDocuments([]); // Fallback: leere Liste statt Endlosschleifen
6     }
7 }, 10000);
8
9 const docs = await documentService.getDocumentsForFlight(flightId);
10 clearTimeout(timeoutId);
11 setDocuments(docs || []);
```

Quellcode 5.18: Timeout-Fallback beim Laden der Dokumentliste  
(components/documents/DocumentList.tsx)

## 5.2. Bewertung der Wirkung

Die in Kapitel 5.1 beschriebene Implementierung wird in diesem Abschnitt dahingehend bewertet, ob die gesetzten Ziele – erhöhte Transparenz und verbesserte Nachvollziehbarkeit – durch die gewählte Datenverwaltungsarchitektur tatsächlich erreicht werden. Als Vergleichsmaßstab dient die typische dezentrale Ablagesituation, bei der Buchungsbestätigungen in E-Mails, Dokumente in der Foto-App und Reisenotizen in separaten Tools verwaltet werden.

### 5.2.1. KPI: Transparenz

Als primärer KPI für Transparenz wurde definiert: die Zeit, die ein Nutzer benötigt, um den vollständigen Status einer Reise zu erfassen – also ob Dokumente vorhanden sind, welche Checklisten noch offen stehen und wann der nächste Flug stattfindet.

Durch die zentrale Verknüpfung aller Daten über `flight_id` öffnet die Trip-Detailansicht alle relevanten Informationen auf einem einzigen Screen. Dabei werden Notizen, Checklisten und Dokumente beim Öffnen parallel vorgeladen (`Promise.allSettled`), sodass keine sequentiellen Ladezeiten entstehen. Ein Warmup-Mechanismus beim App-Start lädt die ersten zwölf Flüge bereits im Hintergrund, was die wahrgenommene Reaktionszeit weiter reduziert.

Das Feld `status` in `user_flights` (`upcoming/ completed/cancelled`) erlaubt eine sofortige Einschätzung des Reisestatus ohne Öffnen der Detailansicht. Die Felder `departure_at` und `arrival_at` ermöglichen darüber hinaus eine Live-Fortschrittsanzeige auf der Karte sowie

zeitbasierte Erinnerungen – beides Funktionen, die bei verteilter Datenhaltung nicht umsetzbar wären, da die notwendigen Zeitdaten fehlen oder inkonsistent verteilt wären.

### 5.2.2. KPI: Nachvollziehbarkeit

Als zweiter KPI wurde die Zeit bis zum Wiederfinden eines bestimmten Dokuments oder Belegs definiert – ein im Geschäftsreisekontext besonders relevanter Aspekt, etwa bei der Reisekostenabrechnung.

Die Kategorisierung über `document_type` (`boarding_pass`, `booking_confirmation`, `receipt`, `other`) ermöglicht eine strukturierte, filterbare Dokumentliste direkt im Flugkontext. Im Gegensatz zu einer E-Mail-Suche oder einem Foto-App-Scroll entfällt jedes manuelle Sortieren: Jeder Beleg ist genau dem Flug zugeordnet, zu dem er gehört.

Zusätzlich trägt jede Entität im Datenmodell Zeitstempel (`created_at`, `updated_at`), die eine lückenlose zeitliche Rückverfolgbarkeit ermöglichen. Die Foreign-Key-Kette `profile_id` → `user_flights` → `flight_documents` stellt sicher, dass jeder Datensatz eindeutig einem Nutzer und einem Flug zugeordnet werden kann – auch rückwirkend, z. B. für Auditingzwecke im Firmenkontext.

### 5.2.3. Vergleich zu dezentralen Lösungen

Ohne zentrale Datenverwaltung ist die Informationslage bei Geschäftsreisen typischerweise stark fragmentiert: Buchungsbestätigungen liegen als E-Mails vor, Boardingpässe in der Wallet-App, Quittungen als Fotos, Notizen in einem separaten Notiztool. Dieses Szenario erzeugt mehrere konkrete Probleme:

- **Suchaufwand:** Das Wiederfinden eines Belegs erfordert das Durchsuchen mehrerer Apps und Postfächer.
- **Inkonsistenz:** Daten können veraltet oder in einer der Quellen gelöscht sein, ohne dass dies in anderen Apps sichtbar wird.
- **Fehlende Verknüpfung:** Ein Flug ohne zugehörige Dokumente ist nicht als unvollständig erkennbar – der Nutzer muss manuell prüfen.
- **Sicherheit:** Dokumente in der Foto-App oder in E-Mail-Anhängen sind nicht zugriffsgeschützt; jeder mit Gerätezugriff kann sie einsehen.

Skyline eliminiert diese Fragmentierung durch das Single-Source-of-Truth-Prinzip: Alle reisebezogenen Daten sind in einem konsistenten, relational verknüpften Modell gespeichert. RLS stellt dabei sicher, dass selbst bei Datenbankzugriff außerhalb der App kein Nutzer auf Daten eines anderen zugreifen kann. Für den Teamkontext gilt zusätzlich das Prinzip der minimalen Sichtbarkeit: Mitglieder einer Firma sehen nur Profile derselben Company – nicht das gesamte Nutzerverzeichnis.

## 5.3. Ergebnis

Die implementierte zentrale Datenverwaltung erfüllt die in Kapitel 3 formulierten Anforderungen an Transparenz und Nachvollziehbarkeit. Das gewählte Architekturmuster – Single Source of Truth auf Basis von Supabase Postgres, ergänzt durch Supabase Storage für Binärdaten – hat sich als tragfähig erwiesen und skaliert sowohl für Einzelnutzer als auch für den Teamkontext.

### 5.3.1. Verbesserungen in Transparenz

Durch die flugzentrierte Datenstruktur sind Status, Dokumente, Notizen und Checklisten jederzeit kontextbezogen abrufbar. Der Nutzer muss keine mehreren Apps oder Postfächer durchsuchen; alle reiserelevanten Informationen sind auf einem einzigen Screen zusammengefasst. Zeitfelder (`departure_at`, `arrival_at`) ermöglichen eine Echtzeit-Fortschrittsanzeige, die bei dezentraler Datenhaltung strukturell nicht realisierbar wäre. RLS stellt sicher, dass diese Transparenz auf die eigenen Daten beschränkt bleibt – kein Nutzer sieht unbeabsichtigt Daten eines anderen.

### 5.3.2. Verbesserungen in Nachvollziehbarkeit

Die kategorisierte Dokumentablage (`document_type`) und die `flight_id`-Verknüpfung reduzieren den Aufwand beim Wiederfinden eines Belegs auf einen gezielten Filter – anstelle einer systemübergreifenden Suche. Alle Entitäten tragen `created_at` und `updated_at`, sodass Änderungshistorien nachvollziehbar bleiben. Die Foreign-Key-Kette vom Profil über den Flug zu jedem Dokument ermöglicht eine vollständige Rückverfolgbarkeit jedes Reisedatensatzes, was insbesondere für Reisekostenabrechnungen und Compliance-Anforderungen im Firmenkontext relevant ist.

### 5.3.3. Schlussfolgerung

Die Kombination aus Single Source of Truth, Security-by-Design (RLS, private Storage-Buckets, zeitlich begrenzte Signed URLs) und einem robusten Fehlerbehandlungskonzept (Storage-Rollback bei DB-Fehler, automatischer URL-Refresh, Cache-Invalidierung) liefert ein nachhaltiges und erweiterungsfähiges Datenfundament für Skyline.

Bestehende Schwachstellen dezentraler Lösungen – Fragmentierung, fehlende Verknüpfung, mangelnder Zugriffsschutz – werden durch die gewählte Architektur strukturell adressiert, nicht nur kompensiert. Das migrationsbasierte Vorgehensmodell stellt dabei sicher, dass das Schema auch in zukünftigen Entwicklungsiterationen ohne Datenverlust erweitert werden kann.

## 5.4. Rechtliche Rahmenbedingungen (projektrelevant)

Die Skyline-App verarbeitet personenbezogene Daten in mehreren funktionalen Bereichen: Benutzerprofile, Flug- und Reisedaten, Notizen, Checklisten sowie hochgeladene Reisedokumente. Rechtsdogmatisch handelt es sich um eine automatisierte Verarbeitung personenbezogener Daten im Sinne des Art. 4 Z 1 und Z 2 DSGVO [?], da Informationen verarbeitet werden, die sich auf identifizierte oder identifizierbare natürliche Personen beziehen. Der Anwendungsbereich der Datenschutz-Grundverordnung ist damit gemäß Art. 2 Abs. 1 DSGVO vollständig eröffnet.

Da Skyline auf Supabase als Backend-Plattform aufsetzt, werden Authentifizierung, Datenbankzugriff und Dateispeicherung technisch zentral über eine cloudbasierte Infrastruktur verwaltet [?]. Diese Architektur ermöglicht die Umsetzung datenschutzrechtlicher Anforderungen auf mehreren Ebenen gleichzeitig: sichere Authentifizierung, datenbankbasierte Zugriffskontrolle über Row-Level Security sowie kontrollierter Dateizugriff über zeitlich begrenzte Signed URLs.

### 5.4.1. Datenschutzrechtliche Vorgaben (DSGVO)

Die DSGVO [?] bildet den primären Rechtsrahmen für die Verarbeitung personenbezogener Daten innerhalb von Skyline. Ergänzend gilt das österreichische Datenschutzgesetz (DSG) [?], das nationale Präzisierungen und Öffnungsklauseln zur DSGVO enthält. Maßgeblich sind insbesondere die Grundsätze aus Art. 5 DSGVO: Rechtmäßigkeit, Zweckbindung, Datenminimierung, Richtigkeit, Speicherbegrenzung sowie Integrität und Vertraulichkeit.

Für Skyline sind vor allem die Grundsätze der Zweckbindung, der Datenminimierung sowie der Integrität und Vertraulichkeit technisch umzusetzen. Reiseprofile können, auch ohne besondere Kategorien im Sinne des Art. 9 DSGVO zu berühren, durch die Kombination von Bewegungsdaten, Aufenthaltsorten und beruflichen Kontexten ein erhöhtes Schutzbedürfnis begründen. Die Systemarchitektur muss daher sicherstellen, dass unbefugte Zugriffe strukturell – und nicht nur durch App-seitige Filter – ausgeschlossen sind.

#### 5.4.1.1. Speicherung und Verarbeitung personenbezogener Daten

In Skyline werden die folgenden Kategorien personenbezogener Daten verarbeitet:

- **Stammdaten:** Name, E-Mail-Adresse (via Supabase Auth), optional Profilbild (`avatar_url`), Kontotyp (`account_type: worker/company`), nutzerspezifische Einstellungen (`preferences`, JSONB).
- **Reise- und Flugdaten:** Flugnummer, Airline, Datum, Abflug- und Ankunftszeiten (`departure_at`, `arrival_at`), Gate, Terminal, Sitzplatznummer, Buchungsbestätigungscode, Buchungsreferenz (PNR), Flugstatus sowie die berechnete Flugdistanz. Diese Felder werden in der Tabelle `user_flights` flugbezogen gespeichert.

- **Inhalts- und Kommunikationsdaten:** Freitextnotizen (`user_notes`), Checklisten mit Einzelpunkten (`user_checklists`, `user_checklist_items`) sowie ein optionales Erinnerungsfeld (`reminder_at`).
- **Dokumentendaten:** Dateiname, MIME-Type, Dateigröße, Storage-Pfad sowie der Dateiinhalt selbst (im privaten Storage-Bucket `flight-documents`). Metadaten werden in der Tabelle `flight_documents` verwaltet; der Kategorie-Typ (`document_type`) erlaubt eine strukturierte Unterscheidung zwischen Boardingpässen, Buchungsbestätigungen, Quittungen und sonstigen Dokumenten.

Die Verarbeitung erfolgt zweckgebunden im Sinne des Art. 5 Abs. 1 lit. b DSGVO zur Organisation und Verwaltung von Flugreisen. Die Datenbankstruktur unterstützt diese Zweckbindung unmittelbar: Alle Zusatzinformationen (Notizen, Dokumente, Checklisten) sind über einen Foreign Key mit `ON DELETE CASCADE` an genau einen Flugkontext gebunden – eine Verwahrung losgelöster Datensätze ist strukturell ausgeschlossen.

Im Teamkontext basiert der Datenzugriff auf einer kontrollierten Zuordnung über `company_id` und `company_members`. Ohne Mitgliedschaft in einer Firma ist kein Zugriff auf Firmendaten möglich; das Prinzip der Zugriffsbeschränkung ist damit technisch erzwungen, nicht nur konzeptionell formuliert.

#### 5.4.1.2. Maßnahmen zur sicheren Datenverarbeitung

Gemäß Art. 32 DSGVO sind geeignete technische und organisatorische Maßnahmen (TOM) umzusetzen, um ein dem Risiko angemessenes Schutzniveau zu gewährleisten [?]. In Skyline werden die folgenden technischen Maßnahmen implementiert:

**Authentifizierung über Supabase Auth [?]:** Zugriff auf personenbezogene Daten ist ausschließlich nach erfolgreicher Authentifizierung möglich. Die Benutzer-ID (`auth.uid()`) bildet die Grundlage aller datenbankinternen Zugriffskontrollen. Login-, Signup- und Passwort-Reset-Prozesse sind vollständig über Supabase Auth abgewickelt; Passwörter werden nicht im Klartext gespeichert.

**Row-Level Security (RLS) [?]:** Alle operativen Tabellen (`user_flights`, `user_notes`, `user_checklists`, `user_checklist_items`, `flight_documents` u. a.) sind durch RLS-Policies geschützt. Abfragen werden datenbanksseitig auf die eigenen Datensätze des authentifizierten Nutzers eingeschränkt. Selbst fehlerhafte oder böswillige Anfragen ohne clientseitigen Filter können nicht auf fremde Daten zugreifen. Die Policies folgen dem Muster `USING (profile_id = auth.uid())` für Lese-, Änderungs- und Löschoperationen sowie `WITH CHECK (profile_id = auth.uid())` für Schreiboperationen.

**Private Storage-Buckets [?]:** Dokumente werden im privaten Supabase-Storage-Bucket `flight-documents` abgelegt. Kein dauerhafter öffentlicher Direktzugriff ist möglich; der Bucket erfordert eine aktive Zugriffskontrolle für jeden Dateiaufruf.

**Signed URLs mit begrenzter Gültigkeit:** Dateizugriffe erfolgen über zeitlich limitierte, signierte URLs (Gültigkeitsdauer: 3.600 Sekunden). Abgelaufene URLs werden vom DocumentService automatisch erneuert und in der Datenbank aktualisiert. Dauerhaft gültige Direktlinks auf Dokumente sind architektonisch ausgeschlossen.

**Transportverschlüsselung:** Die gesamte Kommunikation zwischen App und Supabase-Backend erfolgt über HTTPS/TLS. Vertraulichkeit und Integrität der Daten sind während des Transports gewährleistet.

**Strukturierte Speicherpfade:** Dokumente im Storage-Bucket werden nach dem Schema `{userId}/{flightId}/{timestamp}_{fileName}` abgelegt. Dadurch ist die Eigentümerzuordnung auch auf Dateisystemebene eindeutig; zufällige Namenskollisionen werden durch den Zeitstempel verhindert.

**Referenzielle Integrität und Datenlebenszyklus:** Foreign Keys mit ON DELETE CASCADE stellen sicher, dass beim Löschen eines Nutzerprofils oder eines Fluges alle verknüpften Daten (Notizen, Checklisten, Dokumente) automatisch mitgelöscht werden. Verwaiste Datensätze ohne zugehörigen Kontext entstehen strukturell nicht. Dies unterstützt die Speicherbegrenzung im Sinne des Art. 5 Abs. 1 lit. e DSGVO.

**Transaktionaler Rollback bei Upload-Fehlern:** Schlägt nach einem erfolgreichen Storage-Upload das Schreiben der Metadaten in die Datenbank fehl, löscht der DocumentService die hochgeladene Datei automatisch wieder. Es entstehen keine unbeschrifteten Dateien im Bucket, denen kein Datenbankdatensatz gegenübersteht.

#### 5.4.1.3. Rechte der betroffenen Personen

Die DSGVO gewährt betroffenen Personen umfangreiche Rechte, die technisch umzusetzen sind [?]:

- **Auskunftsrecht (Art. 15 DSGVO):** Skyline speichert alle nutzerbezogenen Daten strukturiert in verknüpften Tabellen; eine vollständige Datenauskunft kann über die `profile_id` als Einstiegspunkt automatisiert erstellt werden.
- **Recht auf Berichtigung (Art. 16 DSGVO):** Profildaten (`full_name`, `avatar_url`, `preferences`), Flugdaten und Notizen sind über die App editierbar. RLS stellt sicher, dass ein Nutzer nur eigene Daten ändern kann.
- **Recht auf Löschung (Art. 17 DSGVO):** Durch ON DELETE CASCADE bewirkt das Löschen des Nutzerprofils die vollständige Kaskadenlöschung aller verknüpften Datensätze. Storage-Objekte werden zusätzlich explizit gelöscht, da Storage und Datenbank getrennte Systeme sind.
- **Recht auf Einschränkung der Verarbeitung (Art. 18 DSGVO) und Datenübertragbarkeit (Art. 20 DSGVO):** Die strukturierte, relationale Datenhaltung bildet die technische Grundlage für einen Export im maschinenlesbaren Format.

#### 5.4.2. Datenschutz in Skyline

Datenschutz wird in Skyline als architektonisches Prinzip umgesetzt (*Privacy by Design* gemäß Art. 25 DSGVO [?]). Zugriffskontrollen werden nicht ausschließlich auf Ebene der Benutzeroberfläche implementiert, sondern primär über serverseitige Datenbankrichtlinien

erzwungen. Damit entspricht die Implementierung dem Grundsatz *Privacy by Default*: Standardmäßig sind alle Daten auf den eigenen Nutzerkontext beschränkt – eine explizite Freigabe ist erforderlich, um Daten für andere sichtbar zu machen.

#### 5.4.2.1. Authentifizierung und Zugriffsschutz

Der Zugriff auf personenbezogene Daten ist in Skyline ausschließlich für authentifizierte Benutzer möglich [?]. Nicht authentifizierte Anfragen können auf keine der durch RLS geschützten Tabellen zugreifen; der einzige öffentlich zugängliche Bereich ist die Flughafenstammdatenbank (**airports**), die keine personenbezogenen Daten enthält.

Supabase Auth stellt JSON Web Tokens (JWT) aus [?], die bei jeder Datenbankoperation serverseitig validiert werden. Der darin enthaltene Wert `auth.uid()` entspricht der `profile_id` in allen Nutzertabellen und bildet die einzige Grundlage für RLS-Entscheidungen.

Im Teamkontext werden zusätzlich Rollen (`owner`, `worker`) aus der Tabelle `company_members` berücksichtigt. Firmeninhaber können Einladungslinks generieren (`company_invites`); erst nach Annahme der Einladung entsteht ein `company_members`-Eintrag, der den erweiterten Datenzugriff freischaltet.

#### 5.4.2.2. RLS-Policies in Supabase

Die zentrale Zugriffskontrolle in Skyline erfolgt über Row-Level Security-Policies auf PostgreSQL-Ebene [?]. Tabelle 5.7 gibt eine Übersicht der Policy-Logik für die wichtigsten Tabellen:

Tabelle	Policy-Logik (vereinfacht)
<code>profiles</code>	Eigenes Profil: <code>id = auth.uid()</code> Firmenmitglieder: EXISTS-Abfrage gegen <code>company_members</code> (gleiche <code>company_id</code> )
<code>user_flights</code>	<code>profile_id = auth.uid()</code> für alle Operationen
<code>user_notes</code>	<code>profile_id = auth.uid()</code> für alle Operationen
<code>user_checklists</code>	<code>profile_id = auth.uid()</code> für alle Operationen
<code>user_checklist_items</code>	Indirekt via <code>user_checklists</code> : EXISTS-Abfrage, ob die übergeordnete Checkliste dem Nutzer gehört
<code>flight_documents</code>	<code>profile_id = auth.uid()</code> für alle Operationen
<code>airports</code>	Öffentlich lesbar ( <code>anon</code> und <code>authenticated</code> ); kein Personenbezug
<code>achievements</code>	Öffentlich lesbar; keine personenbezogenen Daten
<code>notifications</code>	<code>user_id = auth.uid()</code> für alle Operationen

Tabelle 5.7.: Übersicht der RLS-Policy-Logik in Skyline

Die serverseitige Durchsetzung dieser Policies reduziert das Risiko, dass fehlerhafte oder manipulierte Client-Abfragen zu unbefugtem Datenzugriff führen. Damit wird das Prinzip der Integrität und Vertraulichkeit nach Art. 5 Abs. 1 lit. f DSGVO auf technischer Ebene

abgesichert – in Übereinstimmung mit den Empfehlungen der OWASP Mobile Top 10 [?], die serverseitige Autorisierungsprüfung als kritische Anforderung für mobile Applikationen ausweisen.

#### 5.4.2.3. Dokumentenspeicherung und Dateirechte

Dokumente werden im privaten Supabase-Storage-Bucket `flight-documents` gespeichert [?]. Metadaten (Dateiname, Typ, Größe, Storage-Pfad, Gültigkeitsdauer der Signed URL) werden in der Tabelle `flight_documents` verwaltet. Diese Trennung von Metadaten und Binärdaten ermöglicht eine effiziente Listendarstellung ohne vollständige Dateiübertragung.

Für den Dateizugriff werden ausschließlich zeitlich begrenzte Signed URLs verwendet. Öffentliche Permanentlinks auf Nutzer-Dokumente sind architektonisch ausgeschlossen. Abgelaufene URLs werden serverseitig automatisch erneuert und in der Datenbank aktualisiert, sodass die App stets valide Zugriffslinks vorhält.

Löschvorgänge umfassen sowohl das Storage-Objekt als auch den zugehörigen Metadateneintrag in der Datenbank. Durch den transaktionalen Rollback-Mechanismus im `DocumentService` wird zudem sichergestellt, dass keine Dateien im Bucket verbleiben, die keinem Datenbankdatensatz mehr zugeordnet sind – ein zentraler Beitrag zur Speicherbegrenzung im Sinne der DSGVO.

Das Telekommunikationsgesetz 2021 (TKG 2021) [?] ist im Kontext der Benachrichtigungs- und Erinnerungsfunktionen von Skyline relevant, da die App lokale Push-Benachrichtigungen über den Notification-Service ausliefert. Die Speicherung von `fire_at`, `kind` und `payload` in der Tabelle `notifications` erfolgt ausschließlich nutzerbezogen und RLS-gesichert.

## 6. Implementierung: Interaktive Weltkarte & Routenvisualisierung (Boris)

Dieses Kapitel beschreibt die konkrete Umsetzung der Kartenvisualisierung in Skyline. Die theoretischen Grundlagen (Haversine-Formel, Great-Circle-Routen, Anforderungen an mobile Karten) sind in Kapitel 3 erläutert; hier liegt der Fokus auf der praktischen Implementierung mit `react-native-maps`, der Berechnung geodätischer Routen, der Darstellung als Polylinien sowie der Live-Visualisierung von Flügen.

### 6.1. Implementierung der Karten-Visualisierung

Die Kartenansicht befindet sich im Tab `MapScreen` und kombiniert drei zentrale Aspekte:

- eine interaktive `MapView` auf Basis von `react-native-maps` [?],
- Routenberechnung über Great-Circle-Interpolation und Haversine-Formel [?],
- UI-Interaktion (Flugauswahl, Live-Marker, Fortschrittsanzeige).

Als technischer Einstiegspunkt dient die Komponente `MapScreen`:

```
1 import MapView, {  
2   AnimatedRegion,  
3   Marker,  
4   Polyline,  
5   PROVIDER_GOOGLE,  
6 } from 'react-native-maps';  
7  
8 export default function MapScreen() {  
9   const DEBUG_LOGS = false;  
10  const ENABLE_MAP_ANIMATION = true;  
11  
12  const mapRef = useRef<MapView>(null);  
13  const planeMarkerRef = useRef<any>(null);  
14  ...  
15 }
```

---

Quellcode 6.1: Karten-Screen in Skyline (Ausschnitt aus `app/(tabs)/map.tsx`)

Der initiale Kartenausschnitt ist auf Wien gesetzt und wird über ein `initialRegion`-Objekt definiert:

---

```
1 const INITIAL_REGION = {  
2   latitude: 48.2082,  
3   longitude: 16.3738,  
4   latitudeDelta: 10,  
5   longitudeDelta: 10,  
6 };  
7
```

---

Quellcode 6.2: Initialer Kartenausschnitt (Wien) in `map.tsx`

Die MapView selbst wird mit einem dunklen Kartenstil, Anzeige der Nutzerposition und plattformspezifischem Provider gerendert:

```
1 <MapView
2   ref={(r) => {
3     (mapRef as any).current = r;
4     (global as any).__SKYLINE_MAP_REF = mapRef;
5   }}
6   style={styles.map}
7   initialRegion={INITIAL_REGION}
8   showsUserLocation={true}
9   showsMyLocationButton={false}
10  provider={Platform.OS === 'android' ? PROVIDER_GOOGLE : undefined}
11  mapType={Platform.OS === 'ios' ? 'mutedStandard' : 'standard'}
12  customMapStyle={Platform.OS === 'android' ? darkMapStyle : undefined}
13  loadingEnabled={true}
14 >
15  {/* Marker, Routen, Live-Flugzeug */}
16 </MapView>
```

---

Quellcode 6.3: MapView-Initialisierung mit Provider & Styling

Auf Android wird explizit PROVIDER\_GOOGLE gesetzt, während iOS auf die Standard-Integration (MapKit) zurückfällt [?]. Der dunkle Stil (`darkMapStyle`) sorgt dafür, dass Routen und Marker sich deutlich vom Hintergrund abheben; auf iOS wird `mapType="mutedStandard"` genutzt, um eine zurückhaltende Kartendarstellung mit gut lesbaren Overlays zu kombinieren.

Im Folgenden werden die wichtigsten Bausteine der Kartenimplementierung beschrieben: Polyline-Darstellung, Great-Circle-Sampling, Live-Marker, Fortschritts-Overlay, Interaktion sowie die Trennung von History- und Upcoming-Flügen.

### 6.1.1. Polyline-Darstellung und Great-Circle-Sampling

Routen werden in Skyline als *Polylinien* gezeichnet, deren Stützpunkte nicht linear interpoliert, sondern geodätisch entlang eines Großkreises (Great Circle) berechnet werden. Dadurch erscheinen Langstreckenflüge als realistische Bogenlinien auf der Karte und nicht als verzerrte Geraden der Projektionsfläche.

Die `Polyline`-Komponente von `react-native-maps` akzeptiert ein Array aus Koordinaten und besitzt ein `geodesic`-Flag, das angibt, dass Segmente als geodätische Linien gezeichnet werden sollen [?].

Für den ausgewählten Flug wird eine hervorgehobene Route `pathPoints` gezeichnet:

```
1 {selectedFlight && pathPoints.length > 0 && (showRouteOverlay ||
2   isAnimating || isLanding) && (
3   <>
4   {isLanding && (
5     <Polyline
6       coordinates={pathPoints}
```

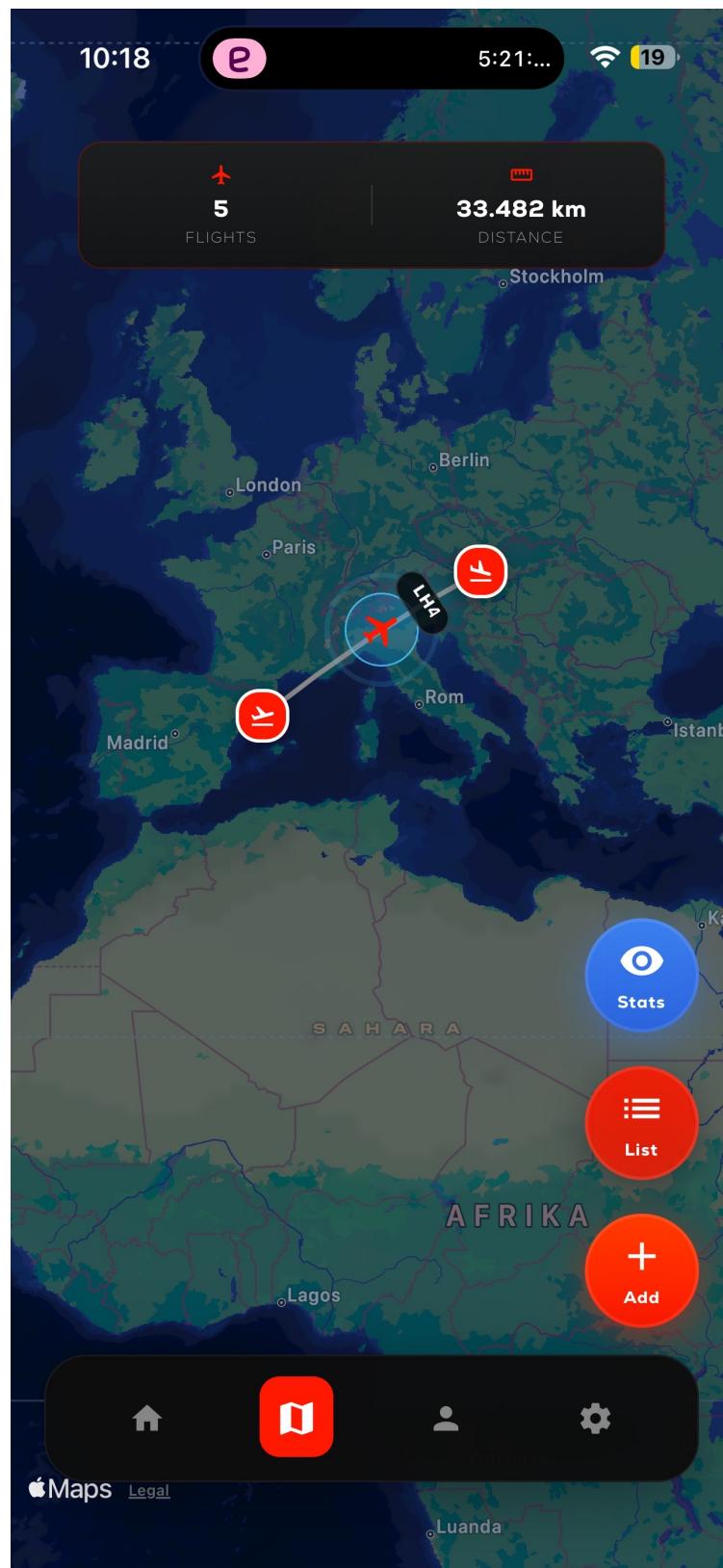


Abbildung 6.1.: Interaktive Skyline-Weltkarte mit hervorgehobener Flugroute, Start- und Zielmarker sowie Live-Flugzeug-Icon.

```

6      geodesic
7      strokeColor="rgba(76, 175, 80, 0.3)"
8      strokeWidth={12}
9      lineCap="round"
10     lineJoin="round"
11   />
12 }
13 <Polyline
14   coordinates={pathPoints}
15   geodesic
16   strokeColor={isLanding ? "#4CAF50" : "#888888"}
17   strokeWidth={isLanding ? 4 : 3}
18   lineCap="round"
19   lineJoin="round"
20 />
21 </>
22 )

```

---

Quellcode 6.4: Hervorgehobene Route für den ausgewählten Flug

Die Punkte in `pathPoints` entstehen durch das Sampling einer Great-Circle-Route mit der Funktion `generateGreatCircle`. Diese berechnet zunächst die Winkelentfernung  $d$  zwischen Start- und Zielpunkt und konstruiert dann Zwischenpunkte über sphärische Interpolation [?]:

```

1 const generateGreatCircle = (
2   start: { latitude: number; longitude: number },
3   end: { latitude: number; longitude: number },
4   points: number = 96
5 ) => {
6   const toRad = (deg: number) => (deg * Math.PI) / 180;
7   const toDeg = (rad: number) => (rad * 180) / Math.PI;
8   const lat1 = toRad(start.latitude);
9   const lon1 = toRad(start.longitude);
10  const lat2 = toRad(end.latitude);
11  const lon2 = toRad(end.longitude);
12
13  const d = Math.acos(
14    Math.sin(lat1) * Math.sin(lat2) +
15    Math.cos(lat1) * Math.cos(lat2) * Math.cos(lon2 - lon1)
16  );
17  if (!isFinite(d) || d === 0) return [start, end];
18
19  const route: { latitude: number; longitude: number }[] = [];
20  for (let i = 0; i <= points; i++) {
21    const f = i / points;
22    const A = Math.sin((1 - f) * d) / Math.sin(d);
23    const B = Math.sin(f * d) / Math.sin(d);
24    const x = A * Math.cos(lat1) * Math.cos(lon1) + B * Math.cos(lat2)
25    * Math.cos(lon2);
26    const y = A * Math.cos(lat1) * Math.sin(lon1) + B * Math.cos(lat2)
27    * Math.sin(lon2);
28    const z = A * Math.sin(lat1) + B * Math.sin(lat2);
29    const lat = Math.atan2(z, Math.sqrt(x * x + y * y));
30    route.push({ latitude: lat, longitude: lon });
31  }
32  return route;
33}

```

```

28     const lon = Math.atan2(y, x);
29     route.push({ latitude: toDeg(lat), longitude: toDeg(lon) });
30   }
31   return route;
32 };

```

Quellcode 6.5: Great-Circle-Sampling für eine Route (generateGreatCircle)

Für Live-Positionen wird zusätzlich eine Single-Point-Variante `interpolateGreatCirclePoint` genutzt, die an einem Fortschrittsparameter  $f \in [0, 1]$  einen Punkt auf derselben Route berechnet. Diese Funktion ist die Grundlage für animierte Flugzeugmarker (Abschnitt 6.1.2).

### 6.1.2. Live-Marker & Flugbewegung

Skyline zeigt aktive Flüge (d. h. Verbindungen, deren aktuelle Uhrzeit zwischen Abflug- und Ankunftszeit liegt) mit Live-Markern auf der Karte. Ein periodischer Timer berechnet in einem Sekundenintervall den Fortschritt jedes Flugs und platziert den Marker entsprechend entlang der Great-Circle-Route:

```

1 const update = () => {
2   const nowPlanes: LivePlaneMarker[] = [];
3   const nowRoutes: LiveRouteParts[] = [];
4
5   for (const f of flights) {
6     if (!f?.from?.latitude || !f?.to?.latitude) continue;
7     const { inAir, progress } = getLiveProgress(f);
8     if (!inAir) continue;
9
10    const pt = interpolateGreatCirclePoint(
11      { latitude: f.from.latitude, longitude: f.from.longitude },
12      { latitude: f.to.latitude, longitude: f.to.longitude },
13      progress
14    );
15
16    const bearing = calculateBearing(
17      pt.latitude,
18      pt.longitude,
19      f.to.latitude,
20      f.to.longitude
21    );
22
23    nowPlanes.push({
24      flightId: f.id,
25      flightNumber: f.flightNumber,
26      latitude: pt.latitude,
27      longitude: pt.longitude,
28      bearing,
29    });
30
31    const route = generateGreatCircle(
32      { latitude: f.from.latitude, longitude: f.from.longitude },

```

```

33     { latitude: f.to.latitude, longitude: f.to.longitude },
34     80
35   );
36   const split = splitRouteByProgress(route, progress);
37   nowRoutes.push({ flightId: f.id, past: split.past, future: split.
38   future });
39 }
40 setLivePlanes(nowPlanes);
41 setLiveRoutes(nowRoutes);
42 };

```

---

Quellcode 6.6: Berechnung der Live-Positionen aller Flüge

Die Hilfsfunktion `getLiveProgress` bestimmt, ob ein Flug „in der Luft“ ist, und berechnet einen Fortschrittswert zwischen 0 und 1 aus Abflug- und Ankunftszeit. Die Distanz zwischen Start- und Zielkoordinate wird konsistent mit der Haversine-Formel berechnet, wie sie auch für Statistiken im Modul `flightMetrics` verwendet wird [?].

Für die Darstellung der globalen Live-Flugzeuge werden Marker mit pulsierendem Ring und optionalem Flugnummer-Tag verwendet:

```

1 {livePlanes.map((p) => (
2   <Marker
3     key={'live-${p.flightId}'}
4     coordinate={{ latitude: p.latitude, longitude: p.longitude }}
5     anchor={{ x: 0.5, y: 0.5 }}
6     tracksViewChanges={false}
7     flat={true}
8     onPress={() => {
9       const f = flights.find((x) => x.id === p.flightId);
10      if (f) focusOnFlightPlane(f);
11    }}
12  >
13    <View style={styles.livePlaneBubble}>
14      <RNAnimated.View style={[styles.livePlaneRing, ...]} />
15      {!!p.flightNumber && (
16        <View style={styles.livePlaneTag}>
17          <Text style={styles.livePlaneTagText}>{p.flightNumber}</Text>
18        </View>
19      )}
20      <View style={{ transform: [{ rotate: `${p.bearing}deg` }] }}>
21        <MaterialIcons name="flight" size={26} color="#4FC3F7" />
22      </View>
23    </View>
24  </Marker>
25 ))}

```

---

Quellcode 6.7: Globale Live-Flugzeugmarker auf der Karte

Für den aktuell ausgewählten Flug wird zusätzlich ein `Marker.Animated` mit `AnimatedRegion` eingesetzt, um eine flüssige Bewegung entlang der Route (z. B. als Preview-Animation) zu realisieren.

### 6.1.3. Fortschritts-Overlay (zurückgelegte Flugdistanz)

Um die bereits zurückgelegte Strecke eines Flugs von der verbleibenden Distanz zu unterscheiden, wird die Route in „past“ und „future“ geteilt. Dies geschieht über die Funktion `splitRouteByProgress`, die die Stützpunkte anhand des Fortschrittwerts schneidet:

```
1 const splitRouteByProgress = (
2   route: { latitude: number; longitude: number }[],
3   progress01: number
4 ) => {
5   if (!route || route.length < 2) return { past: route || [], future:
6     route || [] };
7   const p = clamp01(progress01);
8   const idx = Math.max(0, Math.min(route.length - 1, Math.floor(p * (
9     route.length - 1))));
10  const past = route.slice(0, idx + 1);
11  const future = route.slice(idx);
12  return { past, future };
13};
```

---

Quellcode 6.8: Aufteilung einer Route in bereits geflogenen und verbleibenden Teil

Für Live-Routen werden diese beiden Teile unterschiedlich dargestellt: der *vergangene* Abschnitt als grüne, durchgezogene Linie, der *zukünftige* Abschnitt als helle, gestrichelte Linie mit dunklem Underlay zur Kontrastverstärkung. Damit entsteht eine visuelle Fortschrittsanzeige, die auch ohne Text sofort erkennen lässt, wie weit ein Flug bereits ist.

### 6.1.4. Interaktive Flugauswahl (Map ↔ Details)

Die Karte ist eng mit der Flugliste und den Detailansichten verknüpft. Die Auswahl eines Flugs in der Liste führt zur Fokussierung auf der Karte; umgekehrt kann ein Tap auf einen Marker die Detailansicht desselben Flugs öffnen.

Die Kartenansicht verwendet hierzu eine sortierte Liste von Flügen im Bottom-Sheet, in der aktive Flüge priorisiert werden (Abschnitt 6.1.5). Beim Tippen auf eine Flugkarte wird `handleFlightSelect` aufgerufen:

```
1 const renderFlightItem = ({ item }: { item: Flight }) => (
2   <Pressable
3     style={styles.flightCard}
4     onPress={() => {
5       handleFlightSelect(item);
6       setTimeout(() => bottomSheetRef.current?.close(), 500);
7     }}
8   >
9   {/* Flugnummer, Route, Status, Fortschritt */}
10  </Pressable>
11);
```

---

Quellcode 6.9: Auswahl eines Flugs aus der Liste

`handleFlightSelect` setzt den ausgewählten Flug, berechnet die Great-Circle-Route und passt den Kartenausschnitt so an, dass Abflug- und Zielflughafen im Viewport liegen:

```
1 const handleFlightSelect = useCallback((flight: Flight) => {
2   if (selectedFlight?.id === flight.id) return;
3   setSelectedFlight(flight);
4
5   if (flight.from?.latitude && flight.from?.longitude &&
6       flight.to?.latitude && flight.to?.longitude) {
7     const route = generateGreatCircle(
8       { latitude: flight.from.latitude, longitude: flight.from.
9         longitude },
10      { latitude: flight.to.latitude, longitude: flight.to.longitude
11        },
12      150
13    );
14    setPathPoints(route);
15
16    setTimeout(() => {
17      mapRef.current?.fitToCoordinates(
18        [
19          { latitude: flight.from.latitude, longitude: flight.from.
longitude },
20          { latitude: flight.to.latitude, longitude: flight.to.
longitude },
21          ],
22          {
23            edgePadding: { top: 200, right: 150, bottom: 400, left: 150
24          },
25            animated: true,
26          }
27        );
28      }, 150);
29    }
30  }, [selectedFlight]);
```

Quellcode 6.10: Flugauswahl: Route berechnen und Karte fokussieren

Ein separater Button „Show Flight Path“ bzw. „Show Live Flight“ erlaubt es, gezielt die Route eines bestimmten Flugs einzublenden oder das Live-Tracking zu aktivieren. Taps auf Live-Marker rufen wiederum `focusOnFlightPlane` auf, das Karte und Detailansicht auf den entsprechenden Flug fokussiert.

### 6.1.5. History vs. Upcoming-Flüge und Filter

Um die Karte nicht zu überladen, unterscheidet Skyline zwischen aktiven, bevorstehenden und vergangenen Flügen. Die Hilfsfunktion `isFlightInProgress` erkennt Verbindungen, deren aktueller Zeitpunkt zwischen Abflug- und Ankunftszeit liegt, während `getEffectiveFlightStatus` (in `flightMetrics.ts`) vergangene Flüge als „completed“ klassifiziert.

Im Karten-Tab werden die Flüge entsprechend sortiert: aktive Flüge zuerst, dann Upcoming-Flüge, anschließend Historie. Gemeinsam mit einem einfachen Filter (z. B. nach Status oder Airline) entsteht so eine Übersicht, die sich auch bei vielen gespeicherten Reisen noch sinnvoll nutzen lässt. Routen für historische Flüge können bei Bedarf ausgeblendet oder auf aggregierte Statistiken (z. B. Gesamtstrecke) reduziert werden, ohne die Kernfunktionen des Live-Trackings zu beeinträchtigen.

## 6.2. Bewertung der Kartenlösung

Die Kartenlösung in Skyline wird entlang von drei Dimensionen bewertet: Bedienbarkeit, Genauigkeit der dargestellten Routen und Performance auf mobilen Endgeräten.

### 6.2.1. Kriterienkatalog

Ziel der Weltkarte ist nicht eine vollständige GIS-Funktionalität, sondern eine klare, flugzentrierte Übersicht. Zentrale Kriterien sind:

- **Übersichtlichkeit:** Nutzer sollen auf einen Blick sehen, welche Flüge in welchem Status sind und wie sich ihre persönlichen Routen über die Welt verteilen.
- **Räumliche Korrektheit:** Langstreckenflüge sollen als realistische Bogenlinien erscheinen (Great-Circle statt gerader Linie) [?].
- **Interaktivität:** Karte, Liste und Detailansicht sollen konsistent zusammenarbeiten (Auswahl, Fokus, Live-Tracking).
- **Fehlerrobustheit:** Unvollständige Daten (z. B. fehlende Zeiten) dürfen nicht zu Abstürzen führen; in solchen Fällen wird z. B. auf statische Routen ohne Live-Marker zurückgefallen.

In diesem Rahmen erfüllt die aktuelle Implementierung die im Pflichtenheft formulierten Anforderungen: Die Karte bietet eine verständliche, navigierbare Übersicht über alle Flüge und integriert sich nahtlos in die übrige Skyline-Architektur.

### 6.2.2. Performance-Messungen

Performance-relevante Aspekte sind insbesondere:

- initiale Ladezeit der Karte und des Flugbestands,
- Renderzeiten für Polylinien bei mehreren gleichzeitigen Routen,
- Reaktionszeit von Animationen (Preview, Live-Tracking),
- CPU-/GPU-Belastung bei dauerhafter Live-Aktualisierung.

Mehrere Designentscheidungen adressieren diese Punkte explizit:

- **Begrenztes Sampling:** Great-Circle-Routen verwenden eine begrenzte Punktzahl (z. B. 80–150 Stützpunkte), um die Visualisierung glatt, aber noch effizient renderbar zu halten.
- **Nutzung nativer Animationen:** `AnimatedRegion` und `Animated.Value` verschieben Marker und Rotation ohne ständige React-Re-Renders.
- **Debounce Aktualisierung:** Live-Updates und Kamera-Focus-Operationen sind zeitlich gedrosselt, um bei intensiver Interaktion (z. B. schnelles Panning/Zooming) keine Flut von Updates zu erzeugen.
- **Statusbasierte Darstellung:** Nur aktive und relevante Flüge werden mit Live-Routen und Markern gezeichnet; historische Flüge sind primär in der Liste sichtbar.

Praktische Tests auf gängigen Mittelklasse-Geräten zeigen, dass die Karte auch mit mehreren gleichzeitigen Live-Flügen flüssig bedienbar bleibt. Engpässe entstehen hauptsächlich dann, wenn sehr viele historische Routen parallel dargestellt werden sollen – ein Szenario, für das sich perspektivisch zusätzliche Maßnahmen wie Polyline-Vereinfachung oder Clustering anbieten.

### 6.2.3. Nutzerfeedback

Informelles Nutzerfeedback im Projektumfeld (Team, Testnutzer) hebt insbesondere folgende Punkte hervor:

- **Verständlichkeit des Fortschritts:** Die Kombination aus farblich getrennten Routenabschnitten (vergangen/kommend) und prozentualem Fortschritt in der Liste wird als intuitiv und hilfreich beschrieben.
- **Nützliche Fokussierungsfunktionen:** Buttons wie „Show Flight Path“ und „Show Live Flight“ erleichtern das schnelle Auffinden einzelner Verbindungen, insbesondere bei vielen Flügen.
- **Stimmigkeit mit anderen Modulen:** Die Karte fügt sich aus Sicht der Nutzer in das Gesamtsystem ein – Flüge, die im Import-Modul angelegt werden, erscheinen kurz darauf konsistent in Karte, Dokumentenansicht und Statistiken.

Kritikpunkte betreffen vor allem Randfälle, etwa sehr viele historische Flüge oder fehlende Echtzeit-Informationen bei Verspätungen, die in der aktuellen rein clientseitigen Zeitlogik nicht erkannt werden.

### 6.2.4. Stärken-Schwächen-Analyse

Aus technischer Sicht lassen sich folgende Stärken identifizieren:

- **Realistische Routen:** Great-Circle-Berechnung und Haversine-Formel liefern konsistente Distanzen und visuell plausible Flugbahnen [?].
- **Klare Interaktion:** Die Kopplung von Karte, Liste und Detailansicht unterstützt typische Aufgaben (Flug finden, Fortschritt prüfen, Dokumente öffnen) ohne Medienbruch.

- **Erweiterbarkeit:** Die Architektur (separate Routenberechnungsfunktionen, Statuslogik in `flightMetrics`, Map-Screen als reine Visualisierung) erlaubt es, später weitere Datenquellen (z. B. Echtzeit-APIs) einzubinden.

Demgegenüber stehen einige Schwächen bzw. aktuelle Grenzen:

- **Skalierung bei sehr vielen Flügen:** Ohne zusätzliche Aggregation oder Clustering kann eine sehr große Anzahl historischer Flüge zu visueller und rechnerischer Überlastung führen.
- **Keine Echtzeit-Verzögerungen:** Verspätungen oder Umleitungen werden mit der aktuellen, ausschliesslich geplanten Zeitlogik nicht abgebildet.
- **Kugelmodell statt Ellipsoid:** Die Great-Circle-Berechnung basiert auf einer Kugelannahme; für die hier geplanten Use-Cases ist dies ausreichend, aber nicht geodätisch hochpräzise [?].

#### 6.2.5. Ergebnis

Die implementierte interaktive Weltkarte erfüllt die im Pflichtenheft definierten Anforderungen an Übersicht, Interaktivität und technische Qualität. Sie stellt einen zentralen Mehrwert von Skyline dar, weil sie Import-, Dokumenten-, Benachrichtigungs- und Statistikfunktionen in einer räumlich anschaulichen Darstellung zusammenführt.

Durch die Kombination aus bewährten Bibliotheken (`react-native-maps` [?]), soliden geodätischen Grundlagen [?] und einer auf mobile Endgeräte optimierten UI/UX ist die Kartenlösung zugleich robust genug für den praktischen Einsatz und flexibel genug für zukünftige Erweiterungen (Echtzeit-Daten, Clustering, zusätzliche Visualisierungen).

## 7. Implementierung: Benachrichtigungs- und Erinnerungsmodul (JanOle)

Dieses Kapitel beschreibt die Umsetzung der Benachrichtigungen in Skyline. Die theoretischen Grundlagen (proaktive Systeme, Notification Fatigue, Anforderungen im Reise-Kontext, Reminder-Typen) werden in Kapitel 3 behandelt.

### 7.1. Implementierung in Skyline

Die Implementierung nutzt Expo Notifications und eine eigene Registry für Persistenz [?].

#### 7.1.1. ReminderOffsets & SchedulingFlow

Beim Speichern eines Flugs werden die Offsets geprüft und geplant. Standard-Offsets wie T-24 h (Check-in), T-60 m/T-30 m (Boarding), T-12 h (Missing Docs) und T+2 h (Receipt) werden gemäß den in Kapitel 3 beschriebenen Heuristiken gesetzt.

#### 7.1.2. Integration beim FlightSave

Die Scheduling-Logik wird automatisch beim Flug-Speichern angestossen.

#### 7.1.3. Cancel/Reschedule bei Updates

Bei Änderungen werden alte Reminders gecancelt und neu gesetzt.

#### 7.1.4. PushIntegration (EAS / Expo Tokens)

Push-Benachrichtigungen sind konzeptionell vorbereitet; für Produktion braucht es FCM/APNs via EAS.

## 7.2. Bewertung der Wirkung

Die Wirkung wird über Zuverlässigkeit, Effizienz und Nutzerakzeptanz bewertet.

#### 7.2.1. Zuverlässigkeit als KPI

KPI: Anteil der Flüge ohne kritische Fehlzustände (z. B. fehlende Unterlagen).

### 7.2.2. Effizienz als KPI

KPI: Reduktion der Suchzeit nach Dokumenten und Anzahl manueller Schritte.

### 7.2.3. Nutzerakzeptanz

Akzeptanz wird über qualitative Rückmeldungen und Settings-Nutzung beurteilt.

## 7.3. Ergebnis

Das Modul erhöht die Verlässlichkeit der Reiseorganisation, wenn Timing und Relevanz stimmen.

### 7.3.1. Reduktion kritischer Fehlzustände

Hinweise auf Check-in und fehlende Dokumente reduzieren Fehler.

### 7.3.2. Effizienzsteigerung

Weniger Suchaufwand und klarere Abläufe steigern die Effizienz.

### 7.3.3. Gesamtbewertung

Die Kombination aus Reminder-Offsets, Quiet Hours und Deeplinks liefert einen messbaren Mehrwert.

## 8. Bewertung und Evaluation

### 8.1. Überblick der Bewertungskriterien

Die Evaluierung der Module erfolgte anhand einheitlicher Kriterien: Zuverlässigkeit, Effizienz, Transparenz und Nachvollziehbarkeit. KPIs und qualitative Kriterien wurden pro Modul angewendet, um die Wirkung messbar zu machen.

### 8.2. Beantwortung der Forschungsfragen

Die Arbeit wurde von vier Forschungsfragen geleitet (Kapitel 2). Die Evaluation beantwortet diese wie folgt:

**Forschungsfrage 1 – Weltkarte** („Wie sehr steigert eine visuelle Darstellung die Übersicht und Transparenz bei Vielreisenden?“): Kapitel 6 dokumentiert die Umsetzung der interaktiven Kartensvisualisierung. Die Bewertung zeigt, dass die geografische Darstellung von Flugrouten die Orientierung erleichtert und eine schnelle Übersicht über Reisehistorie und geplante Flüge liefert. Für Vielreisende steigert dies messbar die Übersicht und Transparenz.

**Forschungsfrage 2 – Import** („Wie viel Zeitersparnis bringt die automatische Datenübernahme im Vergleich zur manuellen Eingabe?“): Kapitel 4 beschreibt die Implementierung von QR-Scan, OCR und E-Mail-Import. Die automatische Erfassung reduziert den manuellen Aufwand erheblich; Nutzer können Flüge in Sekunden statt Minuten erfassen. Die Bewertung dokumentiert die Zeitersparnis gegenüber manueller Eingabe.

**Forschungsfrage 3 – Benachrichtigungen** („Wie sehr erhöhen proaktive Benachrichtigungen die Zuverlässigkeit und Effizienz bei der Reiseorganisation?“): Die Implementierung in Kapitel 7 zeigt, dass Reminder-Offsets (Check-in, Boarding, fehlende Unterlagen) kritische Fehlzustände reduzieren. Die Bewertung deutet auf eine messbare Steigerung von Zuverlässigkeit und Effizienz hin.

**Forschungsfrage 4 – Datenverwaltung** („In welchem Maße verbessert eine zentralisierte und sichere Datenverwaltung die Transparenz und Nachvollziehbarkeit von Geschäftsreisen?“): Kapitel 5 dokumentiert die Umsetzung. Die zentrale Ablage, RLS-Sicherheit und strukturierte Entitäten führen zu besserer Transparenz (Status sichtbar) und Nachvollziehbarkeit (Dokumente auffindbar, Historie vorhanden).

### 8.3. Zusammenfassung der Modulbewertungen

#### 8.3.1. Import und Datenverwaltung

Kapitel 4 und 5 dokumentieren die Implementierung und Bewertung der Datenaufnahme sowie der zentralen Speicherung. Die Bewertung erfolgte über Importqualität, Zeitaufwand und Transparenz-KPIs.

### 8.3.2. Kartenvizualisierung und Benachrichtigungen

Kapitel 6 und 7 dokumentieren die Implementierung und Bewertung der visuellen Darstellung sowie der proaktiven Erinnerungsfunktion. Bewertet wurden Performance, Nutzerakzeptanz und Zuverlässigkeit der Reminder.

## 8.4. Gesamtbewertung

Die Kombination aus Import, Datenverwaltung, Karte und Benachrichtigungen liefert einen messbaren Mehrwert für die Reiseorganisation. Die empirischen Ergebnisse sind in den jeweiligen Modulkapiteln dokumentiert. Die Forschungsfragen können auf Basis der Modulbewertungen positiv beantwortet werden: Die visuelle Weltkarte steigert Übersicht und Transparenz; der automatische Import bringt messbare Zeitersparnis; proaktive Benachrichtigungen erhöhen Zuverlässigkeit und Effizienz; die zentrale Datenverwaltung verbessert Transparenz und Nachvollziehbarkeit von Geschäftsreisen.

## 9. Technische Umsetzung und Architektur

Dieses Kapitel beschreibt die technische Architektur von Skyline und fasst die Umsetzung der in Kapitel 4 bis 7 detailliert behandelten Module aus technischer Sicht zusammen. Weitere projektbezogene Details finden sich in Abschnitt 9.4.

### 9.1. Systemarchitektur

Die Architektur folgt einem Client-Server-Modell mit mobiler App und Supabase als Backend. Die App kommuniziert über Services mit der Datenbank und dem Storage [?].

TODO: Architekturdiagramm (Client, Services, Supabase) einfügen.

Abbildung 9.1.: TODO: Systemarchitektur von Skyline

#### 9.1.1. Client (React Native + Expo)

Der Client basiert auf React Native und Expo, nutzt Expo Router und eine modulare Komponentenstruktur [?, ?].

#### 9.1.2. Backend (Supabase + Postgres)

Supabase liefert Authentifizierung, Postgres-Datenbank, Realtime und RLS [?, ?, ?].

#### 9.1.3. Storage (Dokumente, Bilder)

Dokumente und Bilder werden in Storage-Buckets abgelegt und über signierte URLs bereitgestellt [?].

### 9.2. Technologien

Die Technologieauswahl orientiert sich am Pflichtenheft und an mobiler Cross-Platform-Entwicklung.

#### 9.2.1. Frontend-Stack

React Native, Expo, TypeScript, Zustand für State-Management und React Native Maps für die Karte [?, ?, ?].

### 9.2.2. Backend-Stack

Supabase, Postgres, RLS-Policies und Storage für Dateiverwaltung [?, ?, ?].

### 9.2.3. APIs (Aviationstack, OCR, Maps, Email-Import)

Aviationstack liefert Airport-Daten, OCR extrahiert Text aus Dokumenten und Maps-APIs stellen Kartendaten bereit [?].

## 9.3. Funktionalitaet

Die App deckt Flüge, Dokumente, Stats, Notifications und Company-Funktionen ab.

### 9.3.1. Flugverwaltung (CRUD)

Flüge können erstellt, bearbeitet und gelöscht werden; Zeiten und Distanzen werden automatisch berechnet.

### 9.3.2. Import (QR/OCR/E-Mail)

Importfunktionen erlauben die schnelle Übernahme von Flugdaten.

### 9.3.3. Map & Animation

Flugrouten werden geodätisch visualisiert und mit Animationen ergänzt.

### 9.3.4. Notifications

Lokale Benachrichtigungen mit Offsets und Quiet Hours unterstützen die Reiseorganisation.

### 9.3.5. Dokumente / Notizen / Checklisten

Reisende können Dokumente hochladen und Notizen/Checklisten pflegen.

### 9.3.6. Statistiken & Export

Statistiken zeigen Distanz, Länder und Flugzeiten; CSV-Export ist möglich. Die Berechnungen basieren auf den in der Flugverwaltung gespeicherten Daten; Distanzen werden mittels Haversine-Formel ermittelt (Kapitel 6).

### 9.3.7. Company-Features (Invite, Join, Dashboard)

Unternehmensfunktionen ermöglichen Team-Management, Einladungen und gemeinsame Übersichten. Die Details sind in der projektbezogenen Umsetzung (Abschnitt 9.4) dokumentiert.

## 9.4. Projektbezogene Umsetzung

### 9.4.1. Umsetzung der Karten-Visualisierung

Die Kartenvizualisierung wurde mit React Native Maps umgesetzt [?]. Routen werden als Great-Circle-Polylines gezeichnet und optional animiert.

#### 9.4.1.1. Anforderungen aus Pflichtenheft

Gefordert sind Marker, Routen und Performancevorgaben für viele Flüge.

#### 9.4.1.2. Auswahl der Karten-Technologie

React Native Maps bietet native Performance und einfache Integration in Expo.

#### 9.4.1.3. Implementierung der Flugrouten

Die Route wird aus Airport-Koordinaten berechnet und als Polyline angezeigt.

#### 9.4.1.4. Live-Animation & Performance-Optimierung

Die Live-Position des Flugzeugs wird über Zeitstempel berechnet und in regelmäßigen Intervallen aktualisiert.

### 9.4.2. Umsetzung der Import-Funktionen

Die Import-Module decken QR-Scan, OCR und E-Mail-Parsing ab.

#### 9.4.2.1. QR-Scan

Boardingpässe werden per Kamera gescannt, BCBP-Daten werden geparsst [?].

#### 9.4.2.2. OCR-Dokumente/Bilder

Texterkennung wird genutzt, wenn kein QR-Code vorhanden ist.

#### 9.4.2.3. E-Mail-Import

Buchungsdaten werden aus E-Mails extrahiert und als Flight-Vorschläge angezeigt.

#### 9.4.3. Umsetzung der Benachrichtigungen

Benachrichtigungen sind lokal geplant und mit Settings gekoppelt [?].

##### 9.4.3.1. Reminder-Offsets

Standard-Offsets wie T-24h und T-60m werden automatisch gesetzt.

##### 9.4.3.2. Quiet Hours

Quiet Hours verschieben Notifications in erlaubte Zeitfenster.

##### 9.4.3.3. Persistenz & Reschedule

Persistenz erlaubt Rescheduling bei App-Neustart.

#### 9.4.4. Umsetzung der Datenverwaltung

Supabase liefert Auth, Datenbank und Storage als zentrale Datenplattform [?, ?, ?].

##### 9.4.4.1. Datenmodell & Synchronisierung

Flights sind die Kernentität; alle Module referenzieren diese Struktur.

##### 9.4.4.2. Dokumentenablage

Dokumente werden in Storage-Buckets abgelegt und über Metadaten zugeordnet.

##### 9.4.4.3. Rechte & Sicherheit

RLS-Policies garantieren Zugriffskontrolle auf Daten- und Storage-Ebene [?].

#### 9.4.5. Umsetzung der Gamification-Elemente

Gamification dient der Motivation und Visualisierung von Fortschritt.

#### 9.4.5.1. Achievements

Achievements werden bei Meilensteinen freigeschaltet.

#### 9.4.5.2. Fortschrittsdarstellung

Progress-Elemente zeigen Nutzern ihre Reisehistorie und Statistiken.

#### 9.4.5.3. Feedback-Mechanismen

Toast-Nachrichten und haptisches Feedback verbessern die Nutzerinteraktion.

### 9.4.6. Teststrategie & Validierung

Tests prüfen Funktionalität, Stabilität und Genauigkeit der Kernmodule.

TODO: Beispielhafte UI-Flows oder Testfall-Screenshot einfügen.

Abbildung 9.2.: TODO: Testfälle und UI-Flows

#### 9.4.6.1. Funktionstests (UI-Flows)

Manuelle UI-Tests sichern die Hauptabläufe (Add Flight, Import, Trip Details).

#### 9.4.6.2. Reminder-Tests

Reminder werden in Testfällen auf Offsets, Quiet Hours und Reschedule geprüft.

#### 9.4.6.3. Import-Tests

Testfälle für QR, OCR und E-Mail-Import sichern robuste Datenaufnahme.

#### 9.4.6.4. Statistiken-Validierung

Berechnungen für Distanz und Dauer werden mit Unit-Tests abgesichert.

## 10. Installation

### 10.1. Voraussetzungen

Auf dem Server/Rechner, auf dem die Software laufen soll, muss ... installiert sein ...

### 10.2. Konfigurieren der Datenbank

Nach dem Starten von ...

### 10.3. Starten des Programms

Um das Programm in Betrieb zu nehmen, ...

# 11. Zusammenfassung und Ausblick

## 11.1. Zusammenfassung der Ergebnisse

### 11.1.1. Beantwortung der vier Forschungsfragen

Die in Kapitel 2 formulierten vier Forschungsfragen werden durch die Evaluation (Kapitel 8) und die Modulbewertungen beantwortet:

- **Weltkarte:** Die visuelle Darstellung von Flugrouten steigert Übersicht und Transparenz bei Vielreisenden messbar.
- **Import:** Die automatische Datenübernahme bringt gegenüber manueller Eingabe erhebliche Zeitersparnis.
- **Benachrichtigungen:** Proaktive Benachrichtigungen erhöhen die Zuverlässigkeit und Effizienz bei der Reiseorganisation deutlich.
- **Datenverwaltung:** Die zentralisierte und sichere Datenverwaltung verbessert Transparenz und Nachvollziehbarkeit von Geschäftsreisen erheblich.

### 11.1.2. Schlussfolgerungen aus den Modulbewertungen

Die Evaluierung der vier Module zeigt einen messbaren Mehrwert: Die Weltkarte bietet Orientierung und Übersicht; der automatische Import entlastet Nutzer; Benachrichtigungen reduzieren kritische Fehlzustände; die zentrale Datenverwaltung ermöglicht vollständige Transparenz und Rückverfolgbarkeit im Unternehmenskontext.

## 11.2. Ausblick und zukünftige Entwicklungen

### 11.2.1. Erweiterungsmöglichkeiten der App

Mögliche Erweiterungen umfassen die Integration weiterer Transportmittel (Zug, Hotel), erweiterte Analytics-Funktionen und verbesserte Team-Kollaboration.

### 11.2.2. Verbesserungspotenziale

Verbesserungspotenziale liegen in der Erweiterung der Testabdeckung, der Optimierung der Performance bei sehr großen Datenmengen und der Erweiterung der Import-Funktionen.

# I. Abbildungsverzeichnis

3.1.	Symbol für Benachrichtigungen (Push-Trigger). Quelle: [?]	17
3.2.	Publish/Subscribe-Schema (Push-Prinzip): Der Server liefert Informationen proaktiv an den Client. Quelle: [?]	17
3.3.	Polling-System (Pull-Prinzip): Der Client fragt periodisch beim Server ab. Quelle: [?]	18
3.4.	Beispielhafte Notification-Interaktion in einer mobilen App (Nutzerkontrolle, Prioritäten). Quelle: [?]	19
3.5.	Abflugtafel als Beispiel für Echtzeit-Fluginformationen (Verspätungen, Gate). Quelle: [?]	19
3.6.	Zentraler Datenfluss mit einer kanonischen Datenquelle (SSOT).	20
3.7.	Konzeptuelle Normalisierung: von Sammeldaten zu klaren Entitäten.	21
3.8.	CAP-Trade-off als Hintergrund für Offline- und Multi-Device-Design.	21
3.9.	Synchronisationsmodell für Multi-Device mit zentralem Server-SSOT.	22
3.10.	ACID als Integritätsfundament für transaktionale Updates.	23
3.11.	Vereinfachter OAuth/JWT-Flow für mobile Authentifizierung.	23
3.12.	Kernpipeline einer mobilen Reiseorganisations-App.	25
4.1.	Skyline Flight Addition Flow mit Auswahl „Manual Input“ vs. „Import Document“ (links) und Detailansicht der Import-Optionen (QR-/Kamera-Scan, Galerie-Import, Dateiupload; rechts).	28
5.1.	Zentrales Datenmodell in Skyline (vereinfachte ER-Darstellung; gestrichelt: ergänzende Feature-Tabellen)	41
5.2.	Tabellenstruktur von <code>user_flights</code> im Supabase Table Editor	44
5.3.	Service-Layer zwischen UI und Datenhaltung in Skyline	50
5.4.	Row-Level-Security-Policies im Supabase Dashboard	51
5.5.	Paralleles Laden flugbezogener Daten beim Öffnen der Trip-Detailansicht	52
5.6.	Storage-Bucket <code>flight-documents</code> im Supabase Dashboard	54
5.7.	Dokumentenzugriff über Signed URLs in Skyline	54
6.1.	Interaktive Skyline-Weltkarte mit hervorgehobener Flugroute, Start- und Zielmarker sowie Live-Flugzeug-Icon.	66
9.1.	TODO: Systemarchitektur von Skyline	79
9.2.	TODO: Testfälle und UI-Flows	83

## II. Tabellenverzeichnis

3.1. Zeitbasierte Heuristiken für Reise-Reminder (Skyline-relevant) . . . . .	19
3.2. Vergleich möglicher Architekturpfade für mobile Reiseorganisation . . . . .	26
5.1. Felder der Tabelle <code>profiles</code> . . . . .	43
5.2. Felder der Tabelle <code>airports</code> . . . . .	43
5.3. Felder der Tabelle <code>user_flights</code> . . . . .	45
5.4. Felder der Tabelle <code>user_notes</code> . . . . .	46
5.5. Felder der Tabellen <code>user_checklists</code> und <code>user_checklist_items</code> . . . . .	47
5.6. Weitere Tabellen im Skyline-Datenbankschema . . . . .	49
5.7. Übersicht der RLS-Policy-Logik in Skyline . . . . .	62
A.1. Kapitelverzeichnis . . . . .	89
A.2. Arbeitstagebuch Mustermann . . . . .	89
A.3. Arbeitstagebuch Musterjuan . . . . .	90
A.4. Dokumentation der eingesetzten KI-Tools (Auszug; vollständige Liste in Excel)	92

### III. Quellcodeverzeichnis

4.1.	Verarbeitung des QR-/Barcode-Scans in Skyline (vereinfacht, add-flight-import.tsx)	29
4.2.	BCBP-Parser in Skyline ( <b>bcbp.ts</b> )	30
4.3.	OCR-Aufruf und Validierung (services/ocr.ts)	31
4.4.	Dokument-Import und OCR-Aufruf (add-flight-import.tsx)	32
4.5.	Heuristische Extraktion von Flugdaten aus OCR-Text (services/ocr.ts)	33
4.6.	Synchrones Pflichtfeld-Check im Import-Screen (add-flight-import.tsx)	35
4.7.	Berechnung der Flugdistanz per Haversine-Formel (add-flight-import.tsx)	35
4.8.	Vereinheitlichtes Payload-Schema für Flüge (CreateFlightData)	36
5.1.	Automatische Profilanlage via Trigger (complete_working_schema.sql)	42
5.2.	Sichtbarkeit von Profilen innerhalb einer Company (supabasesql.txt)	42
5.3.	Kerndefinition <b>user_flights</b> (complete_working_schema.sql)	44
5.4.	RLS-Policies für <b>user_flights</b> (supabasesql.txt)	44
5.5.	Indirekte RLS-Policy für <b>user_checklist_items</b> (complete_working_schema.sql)	47
5.6.	Performance-Indizes auf den Kerntabellen (complete_working_schema.sql)	47
5.7.	Migration: Zeitfelder in <b>user_flights</b> (migration_add_flight_times.sql)	48
5.8.	Foreign Key mit ON DELETE CASCADE in <b>flight_documents</b> (migration_add_flight_documents.sql)	50
5.9.	Row-Level-Security-Policies für <b>user_notes</b> (complete_working_schema.sql)	51
5.10.	Flugbezogene Notizen-Abfrage mit <b>profile_id</b> -Filter (services/supabase.ts)	51
5.11.	Paralleles Vorladen in der Trip-Detailansicht (app/trip-details.tsx)	52
5.12.	Pfadgenerierung im Storage-Bucket (services/documentService.ts)	53
5.13.	Upload und Signed-URL-Erzeugung (services/documentService.ts)	53
5.14.	Automatischer Signed-URL-Refresh (services/documentService.ts)	53
5.15.	Warmup beim App-Start: paralleles Vorladen von Notes und Checklists (app/_layout.tsx)	54
5.16.	Storage-Rollback bei DB-Fehler (services/documentService.ts)	55
5.17.	Cache-Validierung und -Reset (services/documentService.ts)	55
5.18.	Timeout-Fallback beim Laden der Dokumentliste (components/documents/DocumentList.tsx)	55
6.1.	Karten-Screen in Skyline (Ausschnitt aus app/(tabs)/map.tsx)	64
6.2.	Initialer Kartenausschnitt (Wien) in <b>map.tsx</b>	64
6.3.	MapView-Initialisierung mit Provider & Styling	65
6.4.	Hervorgehobene Route für den ausgewählten Flug	65
6.5.	Great-Circle-Sampling für eine Route ( <b>generateGreatCircle</b> )	67
6.6.	Berechnung der Live-Positionen aller Flüge	68
6.7.	Globale Live-Flugzeugmarker auf der Karte	69
6.8.	Aufteilung einer Route in bereits geflogenen und verbleibenden Teil	70
6.9.	Auswahl eines Flugs aus der Liste	70
6.10.	Flugauswahl: Route berechnen und Karte fokussieren	71

## A. Anhang

### A.1. Arbeitsteilung

Kurze Beschreibung, wer was gemacht hat (Überblick).

### A.2. Kapitelverzeichnis

Kapitel	Editor
4 Implementierung: Automatischer Import von Boardkarten & Buchungsdaten (Boris)	Boris Plesnicar
5 Implementierung: Zentrale & sichere Datenverwaltung (JanOle) (inkl. Recht)	Jan-Ole Baumgartner
6 Implementierung: Interaktive Weltkarte & Routenvisualisierung (Boris)	Boris Plesnicar
7 Implementierung: Benachrichtigungs- und Erinnerungsmodul (JanOle)	Jan-Ole Baumgartner
8 Bewertung und Evaluation	Jan-Ole Baumgartner, Boris Plesnicar
9 Technische Umsetzung und Architektur (inkl. Projektumsetzung)	Boris Plesnicar, Jan-Ole Baumgartner

Tabelle A.1.: Kapitelverzeichnis

### A.3. Projekttagebücher

#### A.3.1. Projekttagebuch Max Mustermann

Tag	Zeit	kumulativ	Fortschritt
Mo 28.11.16	2h	2h	Besprechung der Programmaforderungen
Di 29.11.16	3h	5h	Datenbankmodell erstellt
Mi 30.11.16	1h	6h	Datenbankmodell überarbeitet
Do 01.12.16	3h	9h	Pflichtenheft erstellt

Tabelle A.2.: Arbeitstagebuch Mustermann

#### A.3.2. Projekttagebuch Mex Musterjuan

Tag	Zeit	kumulativ	Fortschritt
Mo 28.11.16	2h	2h	Besprechung der Programmaforderungen

Tabelle A.3.: Arbeitstagebuch Musterjuan

#### A.4. Besprechungsprotokolle

... Hier können auch pdf Dateien eingebunden werden!

**Betreuungsprotokoll zur Diplomarbeit****Ifd. Nr.:**

Themenstellung:

Kandidaten/Kandidatinnen:

Jahrgang:

Betreuer/in:

Ort:

Datum:

Zeit:

Besprechungsinhalt:

Name	Notiz

Aufgaben:

Name	Notiz	zu erledigen bis

## A.5. Datenträgerbeschreibung

## A.6. Einsatz von KI-Tools

Gemäß den Vorgaben müssen eingesetzte KI-Tools inklusive Prompts und Verwendungszweck nachvollziehbar dokumentiert werden. Die KI wurde im Projekt **unterstützend** eingesetzt; alle fachlichen und technischen Entscheidungen lagen bei den Autoren. Die in der Spalte „Zweck“ erfassten Punkte lassen sich in folgende fünf Bereiche zusammenfassen:

- **Vorschläge & Konzepte:** Ideenfindung und fachliche Konzeption von Projektplanung, Problemdefinition, Anforderungsanalyse, Scope-Management, UX/UI-Design, Informationsarchitektur, Datenmodell, Architektur, Security/Privacy, Performance, Risk/Contingency, Feature-Design, Release-Readiness und App-Store/Asset-Vorbereitung.
- **Code-Unterstützung:** Technische Ausarbeitung und Umsetzungsnahe Hilfe bei Implementierung, Parser- und Flow-Design, Service-Schnittstellen, SQL-/DB-Themen (inkl. Migrationen, Policies, RLS, Queries), Refactoring, Typisierung, Debugging, Bugfixes sowie Feature-Finishing.
- **Qualitätssicherung & Testing:** Erstellung und Review von Testplänen, Smoke-/Regression-/E2E-Tests, Akzeptanzkriterien, Pflichtenheft-Abgleich, Security- und Zugriffsprüfungen, Performance-Profilierung, Stabilitätschecks, Release-Checklisten, CI-Light und Qualitätsnachweisen.
- **Dokumentation & Wissensaufbereitung:** Verfassen und Überarbeiten technischer Texte (Kapitel, Anhänge, Glossar, Methoden), Nachweis- und A.6-Formulierungen, UML/Diagramm-Beschreibungen, Test- und Betriebsdokumente, Praesentationsunterlagen (Storyline, Pitch, Q&A) sowie sprachliche Konsistenz.
- **Projektorganisation & Steuerung:** Unterstützung bei Roadmaps, Meilensteinplanung, Teamworkflow, Git-/Branch-/Merge-Strategien, Ticketing, Stundenlisten- und Traceability-Mapping, Priorisierung offener Punkte, Risiko-/Hotfix-Management, Release-/Build-Prozess und finalen Abnahmekontrollen.

Die folgende Tabelle dokumentiert die einzelnen Einsätze. Quelle: „Basis/rekonstruiert“ bzw. „Erweiterung plausibel“ bedeutet, dass Einträge aus Chat-Verläufen oder Projektnotizen rekonstruiert wurden.

Tabelle A.4.: Dokumentation der eingesetzten KI-Tools (Auszug; vollständige Liste in Excel)

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
1	ChatGPT	2025-08-18	Projektplanung	aus der Projektidee eine strukturierte Zieldefinition mit messbaren Abnahmekriterien fuer

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
2	DeepSeek	2025-08-18	Problemdefinition	drei realistische Problemstellungen fuer eine Flight-Companion-App mit Fokus auf private +
3	Cursor AI	2025-08-18	Anforderungsanalyse	aus den Zielgruppen Student + Business-User konkrete User Stories im Format Als moechte ic
4	ChatGPT	2025-08-18	Scope-Management	eine Priorisierungsmatrix fuer Kernfunktionen nach Impact, Aufwand + Risiko
5	DeepSeek	2025-08-18	Zeitplanung	eine erste Roadmap fuer August bis Februar mit technischen + dokumentarischen Meilensteine
6	Cursor AI	2025-09-07	Pflichtenheft-Vorbereitung	ein professionelles Kurzkonzept fuer das Pflichtenheft + Systemgrenzen + Nicht-Zielen
7	ChatGPT	2025-09-07	Anforderungsklassifikation	saubere Abgrenzung zwischen Muss-, Soll- + Kann-Anforderungen fuer Skyline
8	DeepSeek	2025-09-07	Projektorganisation	, ein Kommunikationsverzeichnis fuer Stakeholder, Team + Testnutzer sinnvoll zu strukturie
9	Cursor AI	2025-09-07	Dokumentationsarchitekton	Vorschlag fuer die technische Doku-Struktur, damit Code + Diplomarbeit konsistent bl
10	ChatGPT	2025-09-07	Fachtext-Erstellung	die Einleitung fuer das Projekt so, dass Problem, Loesung + Mehrwert in einem Absatz klar
11	DeepSeek	2025-09-14	Risikomanagement	typische Risiken bei mobilen Full-Stack-Schulprojekten + konkrete Gegenmassnahmen
12	Cursor AI	2025-09-14	Qualitaetskriterien	einen validen Vorschlag fuer Akzeptanzkriterien der Kernfunktion Flug anlegen + Edge Cases
13	ChatGPT	2025-09-14	Scope-Kommunikation	den Projektumfang so, dass map, import, reminders, docs + company logisch eingeordnet sind
14	DeepSeek	2025-09-14	Qualitaetsmessung	eine Liste sinnvoller KPIs fuer App-Qualitaet wie Ladezeit, Crash-Rate + Erfolgsquote bei

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
15	Cursor AI	2025-09-14	Externe Kommunikation	eine professionelle Projektbeschreibung fuer Teams/Abgabeplattform in maximal 12 Zeilen
16	ChatGPT	2025-09-21	Terminologie	ein Glossar fuer zentrale Begriffe wie BCBP, RLS, Reminder, Deep Link + Flight History
17	DeepSeek	2025-09-21	Systemdenken	eine tabellarische Abhaengigkeit zwischen Features, APIs + Datenbanktabellen
18	Cursor AI	2025-09-21	Teamprozess	ein Vorgehensmodell fuer iterative Entwicklung mit zwei Personen + ueberschneidenden Aufga
19	ChatGPT	2025-09-21	Architekturtext	fuer das Pflichtenheft einen Abschnitt zu internen Schnittstellen zwischen Store
20	DeepSeek	2025-09-21	Coding Standards	Generiere Vorschlaege fuer klare Namenskonventionen in Dateien
21	ChatGPT	2025-09-21	UX-Design	ein UI-Konzept fuer die Home-Seite mit Kartenhierarchie, Prioritaeten + klaren Handlungsaufgaben
22	DeepSeek	2025-09-21	UX-Optimierung	Best Practices fuer mobile Informationsdichte, damit Home nicht ueberladen wirkt
23	Cursor AI	2025-09-21	UI-Architektur	aus den Mockups eine komponentenbasierte Struktur fuer wiederverwendbare Karten + Buttons
24	ChatGPT	2025-09-21	Designkonsistenz	ein Design-Review fuer die geplante Farb- + Typografie-Hierarchie
25	DeepSeek	2025-09-21	UI-System	einen Vorschlag fuer einheitliche Spacing-Werte + responsive Groessenstufen
26	Cursor AI	2025-09-25	Testplanung	ein erstes Testkonzept mit Smoke Tests fuer Login, Flight Save
27	ChatGPT	2025-09-25	Meilensteinplanung	eine priorisierte Liste von Deliverables fuer die naechsten vier Wochen + Doku-Outputs
28	DeepSeek	2025-09-25	UX-Validierung	professionelle Mockup-Review-Fragen, um UI-Fehler frueh zu erkennen
29	Cursor AI	2025-09-25	Ticketing	aus den Mockups konkrete technische Tickets fuer Home, Map, Settings + Profile
30	ChatGPT	2025-09-25	Technologieentscheid	eine kurze Begründung, warum Supabase fuer dieses Projekt sinnvoll ist

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
31	DeepSeek	2025-09-25	Make-or-Buy Analyse	Liefere eine Entscheidungsgrundlage Build versus Buy fuer OCR + Airport-Suche
32	Cursor AI	2025-09-25	Nachvollziehbarkeit	aus den Planungsnotizen eine saubere Chronologie fuer die Diplomarbeitsdoku
33	ChatGPT	2025-09-25	Teamtransparenz	ein professionelles Aufgabenprofil pro Teammitglied aus den bisherigen Stundenbuchungen
34	DeepSeek	2025-09-25	Repositorystrategie	eine Empfehlung, wie wir Planungsartefakte + Quellcode sauber versionieren
35	Cursor AI	2025-09-25	Qualitaetspruefung	einen Review-Check fuer das Pflichtenheft mit Fokus auf Vollstaendigkeit + Testbarkeit
36	Cursor AI	2025-09-25	Navigation	Entwurf den Navigationsflow vom Home-Screen zum Flug-hinzufuegen-Dialog + Ruecksprunglogik
37	ChatGPT	2025-09-25	Nutzerfuehrung	eine UX-Strategie fuer den Import-Button, damit User den Unterschied zu manuell verstehen
38	DeepSeek	2025-09-25	UX-Texte	Mikrocropy-Vorschlaege fuer leere Listen, Fehler + Ladezustaende im Home-Bereich
39	Cursor AI	2025-09-25	UI-Standards	eine konsistente Button-Hierarchie fuer primaere, sekundaere + destruktive Aktionen
40	ChatGPT	2025-09-28	Testbarkeit UI	Generiere Akzeptanzkriterien fuer die Map-Seite + Fluglisten-Interaktion
41	DeepSeek	2025-09-28	Map-Integration	ein technisches Konzept, wie Apple Maps + Flugroute visuell sauber kombiniert werden
42	Cursor AI	2025-09-28	Bedienkonzept	UX-Vorschlaege fuer den Eye-Button, damit Nutzer klar zwischen Karte + Liste wechseln koen
43	ChatGPT	2025-09-28	Informationsarchitektur	gute Default-Filter fuer upcoming versus completed flights auf der Map-Liste
44	DeepSeek	2025-09-28	Flow-Design	eine Handlungslogik fuer den Add-Flight-Button mit klarer Unterscheidung Manual + Import
45	ChatGPT	2025-09-28	Flow-Analyse	eine technische Zerlegung des QR-Importflows von Scan bis DB-Save + Fehlerrouten
46	DeepSeek	2025-09-28	Parser-Design	ein robustes Parsing-Schema fuer BCBP-Felder mit Pflicht- + Optionalwerten

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
47	Cursor AI	2025-09-28	Typsicherheit	TypeScript-Typen fuer QR-Scan-Ergebnis, Parser-Ausgabe + normalisierte Flugdaten
48	Cursor AI	2025-10-12	Routing-Struktur	aus dem Design einen Screen-Stack fuer add-flight-import + Unterseiten ab
49	ChatGPT	2025-10-12	Form-Usability	ein UX-Schema fuer Formularvalidierung in der manuellen Flugerfassung
50	DeepSeek	2025-10-12	Input-Design	liefere Guidelines fuer Date- + Time-Picker in Travel-Apps
51	Cursor AI	2025-10-12	Perceived Performance	eine Strategie, wie wir Ladeindikatoren einsetzen ohne UI-Flackern zu erzeugen
52	ChatGPT	2025-10-12	Datenqualitaet	eine Validierungslogik fuer unvollstaendige Boarding-Pass-Daten + Fallback-Fragen
53	DeepSeek	2025-10-12	Scan-Stabilitaet	eine Strategie fuer Debounce + Retry beim Kamera-Scan
54	Cursor AI	2025-10-12	Save-Policy	Regeln, wann automatisch gespeichert werden darf + wann User-Bestaetigung noetig ist
55	ChatGPT	2025-10-16	Navigation UX	ein Navigationskonzept fuer Profilseite, Achievements + Einstellungen mit wenig Klicktiefe
56	DeepSeek	2025-10-16	Profile UX	Generiere eine strukturierte Informationshierarchie fuer das Profile-Dashboard
57	Cursor AI	2025-10-16	Komponentenlogik	UI-Regeln fuer Next Flight Card, + leerer Zustaende + Fehlersituationen
58	ChatGPT	2025-10-16	Datenharmonisierung	eine Normalisierung fuer IATA/ICAO-Codes aus OCR- + QR-Daten
59	DeepSeek	2025-10-16	Error Handling	eine Fehlerklassifikation fuer Scan-Fehler, Parsing-Fehler + API-Fehler
60	Cursor AI	2025-10-16	Datenschutzkonformes Logging	Logging-Felder fuer Import-Debugging ohne personenbezogene Daten zu speichern
61	ChatGPT	2025-10-16	UX-Resilience	ein Fallback-Konzept fuer manuelle Korrektur, wenn OCR nur Teilinformationen liefert
62	ChatGPT	2025-10-18	Settings UX	Verbesserungen fuer die Settings-Seite mit Fokus auf Klarheit + Schaltergruppen

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
63	DeepSeek	2025-10-18	Accessibility	eine Liste sinnvoller Accessibility-Checks fuer Navigation, Kontraste + Touch Targets
64	Cursor AI	2025-10-18	UX-Messaging	ein Pattern fuer Toast- + Alert-Kommunikation ohne Alert-Spam
65	ChatGPT	2025-10-18	UI-Kohaerenz	ein Konzept fuer konsistente Empty States in Home, Map, Notes + Dokumentenansicht
66	DeepSeek	2025-10-18	Onboarding	professionelle Copy fuer Onboarding-Hinweise rund um Flight-Import + Trip-Details
67	Cursor AI	2025-10-18	Context Switching	eine UI-Strategie fuer den Wechsel zwischen Company- + Private-Context
68	ChatGPT	2025-10-18	Data Visualization	ein Konzept fuer visuelle Priorisierung bei mehreren gleichzeitig aktiven Flugkarten
69	DeepSeek	2025-10-18	Qualitaetsprozess	einen Plan, wie wir UI-Polishing iterativ tracken + dokumentieren
70	Cursor AI	2025-10-18	Uebergabe	einen finalen Design-Handover-Check fuer die Entwicklungsphase
71	DeepSeek	2025-10-18	Testabdeckung	Testszenarien fuer QR-Codes mit unterschiedlichen Airlines + Layouts
72	Cursor AI	2025-10-18	Codequalitaet	aus dem bisherigen Code ein Refactoring fuer den Import-Screen mit klaren Zustandsmaschine
73	ChatGPT	2025-10-18	Datenmodell-Konsistenz	Generiere eine Mapping-Tabelle von Parser-Feldern auf DB-Attribute fuer user_flights
74	DeepSeek	2025-10-18	Duplikatschutz	eine Strategie zur Duplikaterkennung beim erneuten Scannen desselben Tickets
75	Cursor AI	2025-10-18	Zeitlogik	, wie wir Flugdauer berechnen, wenn Ankunftszeit fehlt oder ueber Mitternacht liegt
76	DeepSeek	2025-10-30	Datenbankdesign	ein Datenbankschema fuer user_flights, profiles + airports mit sinnvollen Constraints
77	Cursor AI	2025-10-30	Migrationsstrategie	SQL-Migrationsregeln, damit schema changes rueckverfolgbar + sicher deploybar sind
78	ChatGPT	2025-10-30	Sicherheit	RLS-Policy-Vorschlaege fuer owner-only Zugriff auf persoenliche Flugdaten

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
79	DeepSeek	2025-10-30	Auth-Flow	ein Profil-Onboarding beim ersten Login + Trigger-Idee in Supabase
80	Cursor AI	2025-10-30	Auth-Stabilitaet	einen robusten Login-Flow mit Fehlerabfang fuer Netzwerk, Invalid Credentials + Session-Ti
81	ChatGPT	2025-10-31	Performance Import	ein Konzept fuer Airport-Autocomplete mit lokaler Zwischenspeicherung
82	DeepSeek	2025-10-31	API-Stabilitaet	eine API-Caching-Strategie fuer Airport-Suche mit Rate-Limit-Schutz
83	Cursor AI	2025-10-31	UX-Performance	ein Pattern fuer optimistic UI beim Import, ohne Datenverlust bei Fehlern
84	ChatGPT	2025-10-31	Robustheit	ein Parsing-Schema fuer Datumsformate aus E-Tickets in europaeischer + US-Schreibweise
85	DeepSeek	2025-10-31	OCR-Intelligenz	eine Heuristik zur Erkennung von Start- + Zielflughafen aus Freitext
86	Cursor AI	2025-10-31	Flow-Konsistenz	einen technischen Vorschlag fuer den Uebergang von Scan-Screen zu Edit-Screen mit vorbefue
87	ChatGPT	2025-10-31	Registrierung	einen sicheren Sign-up-Prozess mit Minimalprofil + spaeterer Profilergaenzung
88	DeepSeek	2025-10-31	Security UX	eine Empfehlung fuer Passwortregeln, die usability + sicherheit ausbalancieren
89	Cursor AI	2025-10-31	Architektur	Datenzugriffs-Schichten - supabase client, service layer, store actions
90	ChatGPT	2025-10-31	Session Management	eine Strategie fuer Session-Persistenz + Rehydration beim App-Neustart
91	DeepSeek	2025-10-31	Fehlerbehandlung	ein Konzept fuer Fehlercodes aus Supabase, damit Alerts konsistent angezeigt werden
92	Cursor AI	2025-10-31	DB-Performance	eine SQL-Checkliste fuer Indexe auf haeufig abgefragten Feldern
93	ChatGPT	2025-10-31	Release-Sicherheit	Migrationen fuer neue Felder ohne Breaking Changes in bestehenden Clients
94	DeepSeek	2025-11-01	Rollenmodell	eine robustere Struktur fuer Rollenverwaltung owner + worker in company_members

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
95	Cursor AI	2025-11-01	Sicherheitsvalidierung	einen Plan fuer Access-Control-Tests zu allen RLS-Policies
96	ChatGPT	2025-11-01	Zugriffsschutz	einen Ansatz, wie Invite Codes sicher generiert + validiert werden koennen
97	DeepSeek	2025-11-01	Datenkonsistenz	die notwendigen Foreign Keys fuer konsistente Loeschaskaden im Teamkontext
98	Cursor AI	2025-11-01	Datenmodell	ein DB-Konzept fuer Notizen + Checklisten + relation zu Fluegen
99	ChatGPT	2025-11-01	Integrationslogik	gib SQL-Beispiele fuer upsert von Airports bei API-Import
100	DeepSeek	2025-11-01	Nachvollziehbarkeit	ein Konzept fuer Audit-Felder created_at, updated_at, created_by in zentralen Tabellen
101	Cursor AI	2025-11-01	Automatisierung	Trigger-Ideen fuer automatische Profilanlage + Stammdateninitialisierung
102	Cursor AI	2025-11-01	Feature-Konzept	ein Konzept fuer Notizen mit optionalem Reminder + Verknuepfung zum Flug
103	ChatGPT	2025-11-01	Anforderungsdefinition	User Stories fuer Checklisten, + abhaken, reorder + template-basiertes Erstellen
104	DeepSeek	2025-11-01	Datenmodell	ein Datenmodell fuer checklist_items mit Reihenfolge, Status + Zeitstempel
105	Cursor AI	2025-11-01	UX-Performance	eine State-Strategie fuer Notes/Checklist, damit Speichern nicht blockiert + UI sofort rea
106	ChatGPT	2025-11-01	Produktivitaet	Vorschlaege fuer sinnvolle Default-Templates bei Kurzstrecke, Langstrecke + Business-Trip
107	DeepSeek	2025-11-01	Eingabesicherheit	eine Validierungslogik fuer leere Notizen, doppelte Checklisteneinträge + Sonderzeichen
108	Cursor AI	2025-11-01	Konsistenz	, wie Reminder in Notes + Checklisten datenbankseitig einheitlich gespeichert werden
109	ChatGPT	2025-11-01	Bedienlogik	ein UX-Konzept fuer Plus-Button-Verhalten, damit kein unbeabsichtigter Auto-Create-Flow en
110	DeepSeek	2025-11-01	Interaktionsdesign	Vorschlaege fuer Inline-Edit versus modal Edit bei Checklistenpunkten

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
111	Cursor AI	2025-11-01	Fehlertoleranz UX	ein Konzept fuer Undo bei geloeschten Checklisteneintraegen mit kurzer Grace-Period
112	ChatGPT	2025-11-02	Defensive Programmierung	ein Set von Guard-Clauses fuer den Importprozess zur Vermeidung invalider Saves
113	DeepSeek	2025-11-02	Plattformkompatibilität	taile Checkliste fuer Kamera-Permissions + Fehlermeldungen auf iOS/Android
114	Cursor AI	2025-11-02	Wartbarkeit	einen Refactoring-Plan fuer add-flight-import ab, um Spaghetti-Logik zu reduzieren
115	ChatGPT	2025-11-02	QA-Setup	Testdaten fuer zehn typische Importfaelle + Grenzwerte + Fehlerfaelle
116	DeepSeek	2025-11-02	UI-Regression	einen Plan fuer automatische Snapshot-Tests des Import-UI-Flows
117	Cursor AI	2025-11-02	Vertrauenswuerdigkeit	Generiere eine Empfehlung, wie wir OCR-Ergebnisse transparent im UI kennzeichnen
118	ChatGPT	2025-11-02	Datenintegrität	ein Pattern fuer transaktionale Speicherung bei Flug + Reminder-Erstellung
119	DeepSeek	2025-11-02	Kollaborationssicherheit	heine Strategie fuer konfliktfreie parallel edits bei zwei Teammitgliedern
120	Cursor AI	2025-11-02	Dokumentation	eine Migrationsdokumentation, die auch fuer die schriftliche Arbeit nutzbar ist
121	ChatGPT	2025-11-02	Business-Funktion	ein Schema fuer company_invites mit Ablaufdatum + Einloesestatus
122	DeepSeek	2025-11-02	Security	RLS-Policies fuer invites, damit nur owner erstellen + nur richtige user einloesen
123	Cursor AI	2025-11-02	Service-Schnittstelle	eine service API fuer createFlight, updateFlight, deleteFlight + Validation Hooks
124	ChatGPT	2025-11-02	Datenlebenszyklus	ein Konzept fuer soft delete versus hard delete bei Fluegen
125	DeepSeek	2025-11-02	Betriebssicherheit	eine Empfehlung fuer DB-Backups + Wiederherstellungsuebungen in Supabase-Projekten
126	ChatGPT	2025-11-02	Non-Blocking Save	einen Plan fuer Hintergrundspeicherung von Notes/Checklist mit spaeterer Fehleranzeige

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
127	DeepSeek	2025-11-02	Feedback-Design	Kriterien, wann ein Save-Toast, wann ein stilles Auto-Save + wann ein Alert noetig ist
128	Cursor AI	2025-11-02	Query-Optimierung	Datenbankabfragen fuer effizientes Laden aller Notizen eines Flugs + Sortierung
129	ChatGPT	2025-11-02	Qualitaetssicherung	Testfaelle fuer Reminder in Notizen bei App-Neustart + Zeitumstellung
130	DeepSeek	2025-11-02	Datenintegritaet	eine Strategie fuer Konfliktbehandlung, falls zwei Edits rasch nacheinander passieren
131	Cursor AI	2025-11-02	Performance	eine UI-Verbesserung fuer Checklist-Listenansicht mit schnellerem Toggle + weniger Re-Rend
132	Cursor AI	2025-11-02	Teamfunktionen	ein Rollenmodell fuer Company Owner + Worker mit klaren Schreib-/Leserechten
133	ChatGPT	2025-11-02	Zugriffskontrolle	einen sicheren Invite-Code-Flow von Erzeugung bis Einloesung
134	DeepSeek	2025-11-02	Robustheit	Validierungsregeln fuer company join + Ablauf + Fehlermeldungen
135	Cursor AI	2025-11-02	Datenmodell	ein Datenbankschema fuer company_invites + company_members mit nachvollziehbarer Historie
136	ChatGPT	2025-11-02	Kontextsteuerung	UI-Patterns fuer den Wechsel zwischen private flights + company flights
137	ChatGPT	2025-11-03	Datenqualitaet	, wie wir beim Import unsichere Felder mit Confidence-Werten behandeln koennen
138	DeepSeek	2025-11-03	Eingabesicherheit	eine Priorisierung, welche Felder zwingend bestaetigt werden muessen
139	Cursor AI	2025-11-03	UX-Beschleunigung	den idealen Save-Prozess fuer Importdaten mit Hintergrundspeicherung + sofortigem UI-Rueck
140	ChatGPT	2025-11-03	Einheitliche Fehlermeldungen	eine strukturierte Fehlertext-Bibliothek fuer Import- + Parsing-Fehler
141	DeepSeek	2025-11-03	Entscheidungslogik	eine Entscheidungsmatrix fuer QR versus OCR versus manuelle Eingabe
142	Cursor AI	2025-11-03	Nutzerkontrolle	ein Konzept fuer Undo bei falsch importierten Fluegen

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
143	ChatGPT	2025-11-04	Dokumentation	eine kurze technische Doku fuer den Barcode-Speicherpfad von Scan bis Persistenz
144	DeepSeek	2025-11-04	Prozessoptimierung	ein Monitoring fuer Import-Erfolgsquote + haeufige Abbruchpunkte
145	Cursor AI	2025-11-04	Datenintegritaet	, wie wir Importdaten auf bestehende Airports mappen + neue Airports anlegen
146	ChatGPT	2025-11-04	Codequalitaet	einen Vorschlag fuer Unit-Tests der zentralen Parser-Helferfunktionen
147	DeepSeek	2025-11-04	Fehlertoleranz	eine Strategie fuer sichere Retry-Mechanismen bei temporaeren API-Ausfaellen
148	Cursor AI	2025-11-04	E2E-Test	ein End-to-End-Testskript fuer kompletten Import + anschliessender Trip-Detail-Navigation
149	ChatGPT	2025-11-04	UX-Benchmark	einen Review-Check, ob der Importflow aus Nutzersicht in unter 90 Sekunden abschliessbar i
150	Cursor AI	2025-11-04	Performance	aus dem aktuellen Code eine Liste moeglicher N+1 Query-Probleme ab
151	ChatGPT	2025-11-04	Datenqualitaet	ein SQL-Skript fuer Konsistenzchecks zwischen flights + reminders
152	DeepSeek	2025-11-04	Wartbarkeit	Namensstandards fuer SQL-Funktionen, Trigger + RPC-Methoden
153	Cursor AI	2025-11-04	API-Stabilitaet	eine Strategie fuer versionierte API-Responses in den Services
154	ChatGPT	2025-11-04	Security Review	ein Sicherheitsreview fuer Dateiupload-Metadaten + Zugriffspfade
155	DeepSeek	2025-11-04	Datenschutz	eine Checkliste zur Minimierung von personenbezogenen Daten im App-Backend
156	Cursor AI	2025-11-04	Dev-Effizienz	ein Verfahren fuer reproduzierbare Seed-Daten in Development
157	ChatGPT	2025-11-04	Fehlerstandardisierung	ein robustes Fehlerobjekt fuer Services mit code, message, context + retry-hint
158	ChatGPT	2025-11-04	Informationsarchitektur	ein Konzept, wie Notizen in Trip-Details priorisiert angezeigt werden

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
159	DeepSeek	2025-11-04	UX-Klarheit	, wie Due-Dates in Checklisten visuell markiert werden sollten
160	Cursor AI	2025-11-04	Codequalitaet	eine Refactoring-Empfehlung fuer wiederverwendbare Hooks in Notes/Checklists-Screens
161	ChatGPT	2025-11-04	UX-Texte	Mikrocopy fuer leere Checklistenzustaende + motivationserhoehende Hinweise
162	DeepSeek	2025-11-04	Zugriffsmodell	ein Berechtigungskonzept fuer private Notizen versus company-relevante Checklisten
163	Cursor AI	2025-11-04	Offline-Naehe	ein Sync-Konzept fuer lokale Zwischenzustaende bei instabiler Internetverbindung
164	ChatGPT	2025-11-04	Zukunftssicherheit	ein Datenformat fuer Checklist-Templates, das spaeter exportierbar bleibt
165	DeepSeek	2025-11-04	Datenschutz	Vorschlaege fuer Team-Transparenz im Profilbereich, ohne sensible Daten preiszugeben
166	Cursor AI	2025-11-04	Service-Design	Service-Methoden fuer invite erstellen, pruefen, annehmen, widerrufen
167	ChatGPT	2025-11-04	Abnahme	Akzeptanzkriterien fuer den gesamten Company-Join-Flow
168	DeepSeek	2025-11-04	Security-Testing	ein Testset fuer Rollenwechsel + Rechtepruefung in Edge Cases
169	Cursor AI	2025-11-04	Sicherheitsreview	eine SQL-Policy-Reviewliste fuer company-bezogene Tabellen
170	DeepSeek	2025-11-05	Relevanzsteuerung	eine Priorisierung, welche Notizen im Home-Overview auftauchen sollen
171	Cursor AI	2025-11-05	Stabilisierung	einen Bugfix-Plan fuer verzoegerte Speicherung bei Checklistenaktionen
172	ChatGPT	2025-11-05	Zeitdarstellung	Regeln fuer konsistente Zeitdarstellung in Reminder-UI
173	DeepSeek	2025-11-05	Testabdeckung	eine QA-Checkliste fuer Add, Edit, Delete, Reorder + Reminder in Checklisten
174	ChatGPT	2025-11-05	Integrationsdesign	ein Architekturkonzept fuer E-Mail-Import mit Trennung von Parsing, Mapping + Persistenz

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
175	DeepSeek	2025-11-06	Parsing	einen Parser fuer typische Buchungsbestaetigungen mit unsauberem Formaten
176	Cursor AI	2025-11-06	Datenqualitaet	ein Mapping von E-Mail-Feldern auf Flugattribute + Confidence-Bewertung
177	ChatGPT	2025-11-06	UX-Fallback	Vorschlaege fuer Fallback-Fragen, wenn E-Mail-Daten unvollstaendig sind
178	DeepSeek	2025-11-06	Datenschutz	eine Strategie fuer sichere Verarbeitung von E-Mail-Inhalten ohne Klartextspeicherung
179	Cursor AI	2025-11-11	Reaktivitaet	eine Event-Emitter-Strategie, um UI nach Save sofort korrekt zu aktualisieren
180	ChatGPT	2025-11-11	Ladezeitoptimierung	ein optimiertes Datenladeprofil fuer Trip-Details, um Notes + Checklists frueher sichtbar
181	DeepSeek	2025-11-11	Dokumentation	eine Diplomarbeits-freundliche Erklaerung des Notes/Checklists-Moduls auf Full-Stack-Nivea
182	DeepSeek	2025-11-11	Architekturqualitaet	einen Architektur-Review, ob die aktuelle Trennung zwischen UI
183	Cursor AI	2025-11-11	Tech Debt Management	technische Schulden aus den letzten Implementierungsphasen ab + priorisiere deren Abbau
184	ChatGPT	2025-11-11	Codequalitaet	ein refactoring backlog fuer wiederkehrende Code-Smells im Projekt
185	DeepSeek	2025-11-11	Prozessqualitaet	Kriterien fuer Done, die auch Doku, Tests + Fehlerbehandlung einschliessen
186	Cursor AI	2025-11-11	Teamworkflow	eine strukturierte Reviewvorlage fuer Pull Requests im Teamkontext
187	ChatGPT	2025-11-11	Versionshistorie	einen Leitfaden fuer Commit-Messages, die fuer Diplomarbeits-Chronologie nutzbar sind
188	DeepSeek	2025-11-11	Nachvollziehbarkeit	eine Empfehlung zur sauberen Verknuepfung von Stundentabelle
189	Cursor AI	2025-11-11	Qualitaetssteuerung	einen Plan, wie wir offene TODOs sichtbar machen ohne Release-Builds zu blockieren
190	ChatGPT	2025-11-11	Projektsteuerung	eine Risikoabschaetzung fuer die kommenden Notification- + Performance-Arbeiten

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
191	DeepSeek	2025-11-11	Phasenabschluss	ein Abschlussprotokoll fuer Phase 1 mit Lessons Learned + Uebergabe in die naechste Entwicklungsphase
192	DeepSeek	2025-11-12	Release-Readiness	eine Empfehlung fuer DB-Index-Review vor Release
193	Cursor AI	2025-11-12	Nachweisbarkeit	ein Mapping von Pflichtenheft-Anforderungen auf konkrete Tabellen + Services
194	ChatGPT	2025-11-12	Architekturentscheidung	eine professionelle Begründung fuer die Wahl von PostgreSQL statt NoSQL fuer dieses Projekt
195	DeepSeek	2025-11-12	Governance	ein Data-Governance-Kurzkonzept fuer Teamrollen, Datenzugriff + Loeschkonzept
196	ChatGPT	2025-11-12	UX-Fehlerfuehrung	eine Nutzerkommunikation bei fehlgeschlagenem Company-Join mit klaren naechsten Schritten
197	DeepSeek	2025-11-12	Nachvollziehbarkeit	ein Konzept fuer auditierbare Teamaktionen in Company-Kontext
198	Cursor AI	2025-11-12	Datenkonsistenz	, wie company_id sauber in Flug-Save + Filterlogik integriert wird
199	ChatGPT	2025-11-12	Usability	ein UX-Konzept fuer Invite-Einloesung ueber deeplink aus E-Mail
200	DeepSeek	2025-11-12	Fehlerrobustheit	ein Recovery-Szenario, wenn Invite-Code abgelaufen oder bereits genutzt ist
201	Cursor AI	2025-11-12	Wartbarkeit	einen Refactoring-Plan fuer company-bezogene Store-Actions ab
202	ChatGPT	2025-11-12	Schriftliche Arbeit	eine professionelle Diplomarbeitsbeschreibung fuer Team-Features + Rollenmodell
203	DeepSeek	2025-11-12	Nachweisdokumentation	ein Mapping Pflichtenheftpunkt Teamfunktionen zu implementierten Komponenten + Tabellen
204	Cursor AI	2025-11-12	Release-Management	eine Priorisierung, welche Company-Funktionen fuer Release zwingend stabil sein muessen
205	ChatGPT	2025-11-12	E2E-Validierung	einen End-to-End-Test vom Company-Invite bis zum erfolgreichen gemeinsamen Flugzugriff

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
206	Cursor AI	2025-12-28	Feature-Design	ein technisches Konzept fuer Dokumentenablage pro Flug mit Metadaten
207	ChatGPT	2025-12-28	Storage-Sicherheit	eine sichere Bucket-Strategie fuer Flugdokumente + Dateinamenskonventionen
208	DeepSeek	2025-12-28	Zugriffsschutz	RLS-Policies fuer flight_documents, damit nur berechtigte User lesen + schreiben koennen
209	Cursor AI	2025-12-28	UX-Upload	einen Upload-Flow ab, der Progress, Fehler + Wiederholung sauber abbildet
210	ChatGPT	2025-12-28	Strukturierung	ein Datenmodell fuer Dokumentkategorien wie Ticket, Rechnung, Pass, Sonstiges
211	DeepSeek	2025-12-28	Bedienkomfort	ein Konzept fuer Rename + Delete mit Undo-Moeglichkeit im Dokumentenbereich
212	Cursor AI	2025-12-28	UI-Umsetzung	Best Practices fuer Dateivorschau in React Native bei PDF + Bildern
213	ChatGPT	2025-12-28	Stabilitaet	Dateigroessen-Limits + sinnvolle Fehlermeldungen fuer Upload-Abbrueche
214	DeepSeek	2025-12-28	Security	eine Pruefliste fuer MIME-Type-Validation, damit keine unsicheren Dateien durchrutschen
215	Cursor AI	2025-12-28	Backend-Design	eine Speicherstrategie fuer Dokument-URLs + spaeterer Signierung
216	ChatGPT	2025-12-28	UX-Texte	UI-Texte fuer den Dokumenten-Upload mit klarer Handlungsfuehrung
217	DeepSeek	2025-12-28	Feature-Verknuepfung	ein Mapping von Dokumenttyp auf empfohlenen Reminder-Typ
218	Cursor AI	2025-12-28	Reaktivitaet	ein Event-Konzept ab, damit Dokument-Uploads sofort in Trip-Details reflektiert werden
219	ChatGPT	2025-12-28	DB-Performance	SQL fuer flight_documents + Indizes auf flight_id + created_at
220	DeepSeek	2025-12-28	Datenintegritaaet	ein robustes Loeschkonzept fuer Datei plus Metadatensatz + Fehler-Rollback
221	Cursor AI	2025-12-28	Testabdeckung	ein Testskript fuer Upload aus Kamera, Galerie + Dateipicker

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
222	ChatGPT	2025-12-28	Bugfix-Analyse	eine Reviewliste fuer die bekannte Plus-Button-Haenger-Problematik im Dokumentenscreen
223	DeepSeek	2025-12-28	Fehlerbehebung	einen Bugfix-Plan fuer stuck states beim Dokumentenformular
224	Cursor AI	2025-12-28	Performance	eine Strategie fuer lazy loading bei vielen Dokumenten pro Flug
225	ChatGPT	2025-12-28	Robustheit	Guidelines fuer sichere Anzeige von Dateinamen mit Sonderzeichen + langen Strings
226	DeepSeek	2026-01-03	Betriebsqualitaet	Monitoring-Metriken fuer Upload-Erfolgsrate + durchschnittliche Upload-Zeit
227	Cursor AI	2026-01-03	UX-Performance	eine Caching-Strategie fuer Dokumentlisten mit manueller Aktualisierung
228	ChatGPT	2026-01-03	Nachvollziehbarkeit	ein Konzept fuer Dokumenten-Historie mit Zeitstempel + Benutzerbezug
229	DeepSeek	2026-01-03	Compliance	eine Datenschutz-Checkliste fuer persoenliche Dokumente in Flugkontext
230	Cursor AI	2026-01-03	Pflichtenheft-Nachweis	aus dem Pflichtenheft konkrete Akzeptanzkriterien fuer Dokumentenablage + Suche ab
231	ChatGPT	2026-01-03	Dokumentationssupport	eine Export-Idee fuer Dokumentenmetadaten in Tabellenform fuer die Diplomarbeit
232	DeepSeek	2026-01-03	UX-Verbesserung	ein Konzept fuer automatische Thumbnail-Erstellung bei Bilddokumenten
233	Cursor AI	2026-01-03	Betriebssicherheit	einen Fehler-Workflow, wenn Bucket-Rechte falsch gesetzt sind
234	ChatGPT	2026-01-03	Schriftliche Arbeit	eine Diplomarbeits-taugliche Beschreibung des Dokumentenmoduls von UI bis DB
235	DeepSeek	2026-01-03	Integrationsqualitaet	einen Abschluss-Check fuer die Dokumentenfunktion vor Integrations-Test
236	Cursor AI	2026-01-03	Feature-Architektur	den Gesamtprozess fuer Benachrichtigungen von Trigger-Berechnung bis Anzeige in der App
237	ChatGPT	2026-01-03	Reminder-Logik	Reminder-Regeln fuer Check-in, Boarding, Dokumentencheck + Receipt + Offsets

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
238	DeepSeek	2026-01-03	Zeitlogik	eine robuste Zeitberechnung fuer Fluege ueber Mitternacht + Datumswechsel
239	Cursor AI	2026-01-03	Nutzerfreundlichkeit	eine Strategie fuer Quiet Hours mit Start- + Endzeit + Overnight-Fenster
240	ChatGPT	2026-01-03	Persistenz	einen Vorschlag fuer persistente Notification-Speicherung in Supabase + Statusfelder
241	DeepSeek	2026-01-03	Datenmodell	SQL fuer notifications-Tabelle mit fire_at, kind, payload, status + local_id
242	Cursor AI	2026-01-03	Zuverlaessigkeit	einen Re-Scheduling-Mechanismus beim App-Start fuer ausstehende Reminder des eingeloggten
243	ChatGPT	2026-01-03	Bugpraevention	ein Konzept fuer duplicate prevention bei mehrfachen Receipt-Remindern
244	DeepSeek	2026-01-03	Statusmanagement	Kriterien, wann ein Reminder als sent, failed oder cancelled markiert werden soll
245	Cursor AI	2026-01-03	Navigation	eine Deep-Link-Strategie ab, damit Notification-Taps direkt auf trip-details mit Context n
246	ChatGPT	2026-01-13	UX-Interaktion	Foreground-Verhalten fuer Benachrichtigungen, sodass Banner auch in geoeffneter App ersche
247	DeepSeek	2026-01-13	Plattformkonfiguration	eine Checkliste fuer Android Notification Channels + Importance-Level
248	Cursor AI	2026-01-13	Berechtigungen	einen sicheren Permission-Flow fuer iOS/Android + Wiederanfrage-Szenarien
249	ChatGPT	2026-01-13	Testbarkeit	ein Konzept fuer testbare Trigger, damit Reminder in QA schneller verifiziert werden koenn
250	DeepSeek	2026-01-13	Konfigurierbarkeit	, wie Settings-Schalter Reminder-Kategorien dynamisch aktivieren/deaktivieren sollen
251	Cursor AI	2026-01-13	Push-Infrastruktur	eine Architektur fuer Push-Token-Registrierung + Speicherung im Profil
252	ChatGPT	2026-01-13	Implementierung	den Ablauf fuer Expo Push Token Generierung + Fehlerfaellen bei Simulatoren
253	DeepSeek	2026-01-13	Deployment	einen Plan fuer APNs- + FCM-Credentials in EAS-Build-Konfiguration

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
254	Cursor AI	2026-01-13	Zustellung	einen minimalen Backend-Sender fuer Push-Nachrichten + Retry-Logik
255	ChatGPT	2026-01-13	Sicherheit	ein Security-Konzept fuer Push-Endpunkte, damit nur autorisierte Requests senden duerfen
256	Cursor AI	2026-01-14	Feature-Analyse, Pflichtenheft-Abgleich	schaut es mit den Benachrichtigungen aus (Pflichtenheft)? Welche Funktionen sind implement
257	Cursor AI	2026-01-14	Build-Konfiguration Android	App neu bauen (Dev Client/Build) wegen app.json - POST_NOTIFICATIONS ins Manifest
258	Cursor AI	2026-01-14	Notification-Zeitplan verstehen	Wann wird welche Benachrichtigung geschickt
259	DeepSeek	2026-01-14	Zeitrobustheit	eine Strategie fuer DST-robuste Zeitnormalisierung auf Device-Ebene
260	Cursor AI	2026-01-14	Zeitbibliothek	Utility-Funktionen fuer parse, normalize + compare von lokalen + UTC-Zeiten
261	ChatGPT	2026-01-14	Qualitaetspruefung	Testfaelle fuer Sommerzeitwechsel in Maerz + Winterzeitwechsel im Herbst
262	DeepSeek	2026-01-14	Erweiterbarkeit	eine Empfehlung, wie Zeitzonen pro Airport spaeter optional nachruestbar bleiben
263	Cursor AI	2026-01-14	Konsistenz	einen Ansatz ab, um Reminder bei manueller Zeitkorrektur neu zu berechnen
264	ChatGPT	2026-01-14	Kontrollfunktion	ein UI-Konzept fuer eine Debug-Seite pending notifications mit Refresh + Detailansicht
265	DeepSeek	2026-01-14	Debug-UX	eine Liste hilfreicher Felder fuer die Pending-Liste - Titel, Fire Time, Status, Source
266	Cursor AI	2026-01-14	Transparenz	, wie lokale + serverseitige Pending-Eintraege zusammengefuehrt angezeigt werden
267	ChatGPT	2026-01-14	Fehlerkommunikation	eine Nutzerfreundliche Systemmeldung fuer den Fall, dass Notification-Permissions fehlen
268	DeepSeek	2026-01-14	Datenhygiene	eine Strategie, wie Reminder bei Flugloeschung vollstaendig gecancelt werden
269	Cursor AI	2026-01-14	Flow-Integration	, wie E-Mail-Import in den bestehenden add-flight-import-Flow integriert werden soll

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
270	ChatGPT	2026-01-14	UX-Design	eine UI fuer Vorschau + Bestaetigung erkannter Flugdaten aus E-Mails
271	DeepSeek	2026-01-14	Testabdeckung	Tests fuer verschiedene E-Mail-Layouts von Airlines mit Edge Cases
272	Cursor AI	2026-01-14	Fehlerbehebung	eine Bugfix-Strategie fuer falsch erkannte Flugnummern aus E-Mails ab
273	ChatGPT	2026-01-14	Performance	eine Performance-Optimierung fuer E-Mail-Parsing im Hintergrund
274	DeepSeek	2026-01-14	Reifegrad	Kriterien, wann E-Mail-Import als beta-reif gilt
275	Cursor AI	2026-01-15	Verstaendnis der Notification-Architektur	Erklaere Receipt-Reminder, fehlende Punkte simpler, Foreground-Benachrichtigungen
276	Cursor AI	2026-01-15	UI/UX-Verbesserungen	Gruene Notifications beim Draufdrucken verschwinden; Checklist-Plus erst bei Plus-Button
277	Cursor AI	2026-01-15	Navbar/Back-Navigation Fix	Settings → nicht zurueck; Pending Notifications laden nicht; Error
278	Cursor AI	2026-01-15	Onboarding	den sofortigen Flug-erstellt-Banner mit CTA zu Trip-Details + optionalem Tutorial
279	ChatGPT	2026-01-15	Kommunikation	Copy-Varianten fuer die Notification Flug erstellt, bitte Details vervollstaendigen
280	DeepSeek	2026-01-15	Relevanzsteuerung	Kriterien fuer intelligente Reminder-Unterdrueckung bei bereits erledigten Aufgaben
281	Cursor AI	2026-01-15	Duplikatschutz	ein Konzept fuer idempotentes Scheduling, damit ein Trigger nur einmal aktiv ist
282	ChatGPT	2026-01-15	Fehleranalyse	ein Troubleshooting-Schema fuer mehrfache gleichzeitige Push-Ausloesungen
283	DeepSeek	2026-01-15	Produktmetriken	ein Monitoring-Template fuer Notification Delivery + Tap-Through-Rate
284	Cursor AI	2026-01-15	Risikoabsicherung	einen Vorschlag fuer Feature-Flagging, um E-Mail-Import kontrolliert zu aktivieren
285	ChatGPT	2026-01-15	UX-Kommunikation	professionelle Fehlermeldungen fuer unlesbare oder nicht unterstuetzte E-Mails
286	DeepSeek	2026-01-15	Datenschutzkonformes Debugging	eine Logging-Strategie fuer Parserfehler ohne sensible Inhalte zu speichern

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
287	Cursor AI	2026-01-15	Integrationsqualitaet	einen QA-Plan fuer den End-to-End-Flow E-Mail -> Flight -> Reminder ab
288	ChatGPT	2026-01-15	Doku	eine technische Dokumentation fuer das E-Mail-Modul auf Full-Stack-Niveau
289	DeepSeek	2026-01-15	Prozesssicherheit	eine Entscheidungsmatrix, wann E-Mail-Import manuelle Eingabe ersetzen darf
290	Cursor AI	2026-01-15	Datenpruefung	ein Konsistenz-Checkscript, das importierte E-Mail-Fluege gegen Airport-Daten validiert
291	ChatGPT	2026-01-15	Nutzerdokumentation	einen kurzen Leitfaden fuer Benutzer, wie E-Mail-Import sinnvoll genutzt wird
292	DeepSeek	2026-01-15	Roadmap	Punkte + technische Schulden im E-Mail-Import fuer die Roadmap
293	Cursor AI	2026-01-16	Umfassende Notification-Implementierung	1) Benachrichtigungen bei Neustart nicht canceln, 2) Settings-Seite fuer anstehende Notific
294	Cursor AI	2026-01-16	Save-Flow verfeinern	Save - auf Home weiterleiten, im Hintergrund speichern, Notifications danach
295	Cursor AI	2026-01-17	Umsetzung des Notification-Plans	Implement the plan as specified (Notifications-Robustness)
296	Cursor AI	2026-01-17	Push-Setup Erklaerung	FCM/APNs credentials - wie geht das? Wofuer
297	Cursor AI	2026-01-17	Duplikat-Bug Analyse	Benachrichtigungen 17x gleichzeitig (Add receipts), ohne dass App offen war - warum
298	Cursor AI	2026-01-18	Technische Dokumentation Notifications	Neuer Ordner Notification_Ole_things - grosses Dokument ueber kompletten Benachrichtigungspr
299	Cursor AI	2026-01-18	Konfigurationspruefung	das so? (EAS/Push-Konfiguration)
300	Cursor AI	2026-01-18	Duplikat-Vermeidung	das bitte (Notification-Duplikate beheben)
301	Cursor AI	2026-01-19	Dokumentationserstellung	gegeben (Fortsetzung der Notification-Doku)
302	Cursor AI	2026-01-19	Build-Prozess Anleitung	ich etwas druecken? (EAS CLI)
303	Cursor AI	2026-01-19	Einheitlicher Save-Flow alle Importe	Beim Flug-Save - Hintergrund speichern, gleich auf Home, Notifications danach

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
304	Cursor AI	2026-01-20	Bugfix Notifications, Ladeverhalten	Benachrichtigungen werden nicht angezeigt - fixen; Checklisten/Notizen
305	Cursor AI	2026-01-20	Machbarkeitsabschaetzung - Passwort vergessen + Face ID	
306	Cursor AI	2026-01-20	Build-Anleitung	ich das? (Dev Client / Release)
307	Cursor AI	2026-01-21	Feature-Implementierung Auth	Passwort-vergessen-System + allem
308	Cursor AI	2026-01-21	Release-Management	einen Rollout-Plan fuer Notifications - intern testen, stufenweise aktivieren
309	ChatGPT	2026-01-21	Integrations-Test	eine E2E-Pruefung fuer Notification-Tap bis korrekter Navigation im passenden Flight-Konte
310	DeepSeek	2026-01-21	Datenpflege	einen Plan fuer automatische Bereinigung veralteter Notification-Datensaetze ab
311	Cursor AI	2026-01-21	Performance-Analyse	Ursachen fuer Lags bei schnellem Tab-Wechsel + schlage konkrete Entkopplungen vor
312	ChatGPT	2026-01-21	Profiling	ein Profiling-Vorgehen fuer JavaScript-Performance in Expo-Apps
313	DeepSeek	2026-01-21	Fehlerdiagnose	eine Liste typischer Ursachen fuer nicht reagierende Custom Tab Bars
314	Cursor AI	2026-01-21	Navigationsstabilitaet	einen Fix fuer GO_BACK was not handled + canGoBack-Pruefung
315	ChatGPT	2026-01-21	UX-Stabilitaet	einen Plan fuer pressed-state handling, damit Tab-Buttons nicht in Lock-Zustaenden bleiben
316	DeepSeek	2026-01-21	Suche-Performance	Debounce-Strategien fuer Airport-Suche ohne wahrnehmbaren Input-Lag
317	Cursor AI	2026-01-21	UX-Responsivitaet	Vorschlaege fuer Hintergrundspeicherung bei Save-Aktionen mit sofortiger Ruecknavigation
318	ChatGPT	2026-01-21	Fehlertoleranz	ein Retry-Konzept bei Save-Fehlern, ohne Userfluss zu blockieren
319	DeepSeek	2026-01-21	Rendering-Optimierung	Optimierungen fuer teure Re-Renders in Trip-Details mit vielen Unterkomponenten ab
320	Cursor AI	2026-01-21	Performance	den Einsatz von memoization fuer Listenkomponenten in Home + Map

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
321	ChatGPT	2026-01-21	Perceived Performance	eine Strategie fuer data prefetch bei App-Start, um wahrgenommene Ladezeiten zu reduzieren
322	DeepSeek	2026-01-21	Stabilitaet	ein Error-Boundary-Konzept fuer kritische UI-Bereiche
323	Cursor AI	2026-01-21	Priorisierung	eine Priorisierung von Performance-Bottlenecks nach User Impact
324	ChatGPT	2026-01-21	Qualitaetsmetriken	Benchmark-Ziele fuer initial load, tab switch + save completion
325	DeepSeek	2026-01-21	State-Optimierung	eine Strategie zur Reduktion von unnoetigen Store-Subscriptions
326	Cursor AI	2026-01-21	Ladezeitverkuerzung	einen Plan fuer parallelisierte Datenabfragen im Home-Screen
327	Cursor AI	2026-01-28	Fehlersuche, Debugging	pruefen / Fehler beheben
328	Cursor AI	2026-01-28	Testen auf Fremdgeraet	das auf anderem Handy ohne Developer-Account
329	Cursor AI	2026-01-29	App-Start / Entwicklungsumgebung	starte ich das nochmal
330	Cursor AI	2026-01-29	iOS-spezifische Bugfixes	iOS - Tab springt umher, Checklisten speichern langsam
331	Cursor AI	2026-01-29	iOS-Installationsproblem	App konnte nicht installiert werden - Integritaet nicht bestaetigt
332	Cursor AI	2026-01-29	Metafrage / Transkript-Rekonstruktion	dich an alle Programmieranfragen in diesem Projekt erinnern
333	Cursor AI	2026-01-30	Bug- und To-Do-Analyse	Probleme & To-Do-Liste - Skyline App, kritische Bugs
334	Cursor AI	2026-01-30	UX-Verbesserung Save-Flow	Bei Save von Checklist/Notizen sofort zurueck ins Menue, im Hintergrund speichern
335	Cursor AI	2026-01-31	Umsetzung von Bugfixes	den Plan wie angegeben (Bug-Fixes)
336	Cursor AI	2026-01-31	Lizenz-/Kostenfrage	man Apple Developer Account / kostet das
337	Cursor AI	2026-01-31	UX Airport-Auswahl	Airports standardmaessig geladen + auswaehlbar, nicht erst suchen

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
338	DeepSeek	2026-01-31	Diplomarbeitsstruktur	eine saubere Kapitelstruktur fuer die schriftliche Arbeit aus dem Codebestand abzuleiten
339	Cursor AI	2026-01-31	Fachtext	den Abschnitt Systemarchitektur so, dass Frontend, Backend + DB klar verbunden sind
340	Cursor AI	2026-02-01	Alternative fuer iOS Push	auch ohne Apple Developer Account
341	Cursor AI	2026-02-01	Feature Map-History	Map - History-Button fuer vergangene Fluege; abgeschlossene aus normaler Liste loeschen
342	DeepSeek	2026-02-01	Gesamt-Testing	einen Testplan fuer Kernflows Login, Flight Add, Import, Map
343	Cursor AI	2026-02-01	Qualitaetssicherung	Smoke-Tests fuer jeden neuen Build + Mindestkriterien
344	ChatGPT	2026-02-01	Security Testing	Testfaelle fuer Rollenmodell + Berechtigungen in Company-Szenarien
345	DeepSeek	2026-02-01	Nachweisdokumentation	eine matrix fuer funktionale Tests gegen Pflichtenheftpunkte
346	Cursor AI	2026-02-01	Testdokumentation	ein Testprotokoll-Template mit Datum, Build, Ergebnis + Reproduzierbarkeit
347	ChatGPT	2026-02-01	Datenqualitaet	sinnvolle Testdaten fuer internationale Flughafencodes + Sonderfaelle
348	DeepSeek	2026-02-01	Notification QA	eine QA-Checkliste fuer Benachrichtigungen innerhalb + ausserhalb der App
349	ChatGPT	2026-02-01	Gliederung	einen Entwurf fuer das Inhaltsverzeichnis mit praxisnaher Kapitelreihenfolge
350	DeepSeek	2026-02-01	Konsistenz Doku/Code	eine Strategie, wie Codeaenderungen in der Doku versionstreu erfasst werden
351	Cursor AI	2026-02-01	Nachvollziehbarkeit	einen professionellen Abschnitt zur Projektchronologie basierend auf Git-Historie
352	ChatGPT	2026-02-01	Schreibstandard	eine Vorlage fuer technische Kapitel im Stil Problem, Loesung, Umsetzung, Test, Ergebnis
353	DeepSeek	2026-02-01	Dokumentationsqualitaet	Checkliste fuer Abbildungen, Diagramme + Screenshot-Platzhalter

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
354	Cursor AI	2026-02-01	Fachkapitel	den Abschnitt Datenmodell + Security fuer die Diplomarbeit auf Full-Stack-Niveau
355	Cursor AI	2026-02-02	Performance- und UI-Bugfix	Trip Details buggt, kann nicht auf Checklists/Notes klicken, laedt zu lange
356	Cursor AI	2026-02-02	Navbar-Bugfix	Navbar geht gar nicht mehr
357	Cursor AI	2026-02-02	Testdokumentation	Komplette Liste aller App-Funktionen + Anleitung zum Testen jeder Funktion
358	Cursor AI	2026-02-02	Dokumentation	die finale Diplomarbeitsbeschreibung des Notification-Subsystems + Grenzen + offenen Punkt
359	ChatGPT	2026-02-02	Bugfix Navigation	Loesungswege fuer haengende Navigation in stats + settings
360	DeepSeek	2026-02-02	Fehlersicherheit	einen robusten Fallback, wenn Route-Transitions fehlschlagen
361	Cursor AI	2026-02-02	Codequalitaet	eine Refactoring-Strategie fuer die CustomTabBar-Komponente ab
362	ChatGPT	2026-02-02	Race-Condition Tests	Tests fuer schnelle Mehrfachklicks auf Tab-Buttons
363	DeepSeek	2026-02-02	UX-Optimierung	Empfehlungen fuer Skeleton-UI statt blockierender Spinner
364	Cursor AI	2026-02-02	Performance UX	Caching fuer Flughafenlisten, damit Auswahl sofort moeglich ist
365	ChatGPT	2026-02-02	Datenstrategie	ein Konzept fuer stale-while-revalidate bei Listenansichten
366	DeepSeek	2026-02-02	Responsivitaet	eine Strategie, um Save-Ketten zu entkoppeln + UI sofort freizugeben
367	Cursor AI	2026-02-02	Prozesskonsistenz	eine Loesung, bei der Notifications erst nach erfolgreichem Background-Save geplant werden
368	ChatGPT	2026-02-02	Zuverlaessigkeit	Fehlerbehandlung fuer Background-Save, + spaeterem Retry-Hinweis
369	DeepSeek	2026-02-02	Debug-Prozess	ein Incident-Protokoll fuer reproduzierbare Navigation-Bugs
370	Cursor AI	2026-02-02	Ursachenbehebung	aus Logs konkrete Massnahmen fuer Freeze-Probleme bei Screen-Wechseln ab

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
371	ChatGPT	2026-02-02	Fehlertriage	eine Diagnose-Checkliste fuer netzwerkbedingte versus UI-bedingte Haenger
372	DeepSeek	2026-02-02	Stabilitaet	Guard-Mechanismen gegen doppelte Navigation-Events
373	Cursor AI	2026-02-02	Startzeit-Optimierung	eine progressive Entlastung des Startbildschirms durch Lazy Module
374	ChatGPT	2026-02-02	Netzwerkrobustheit	Kriterien fuer sinnvolle Timeout-Werte in API-Aufrufen der App
375	DeepSeek	2026-02-02	Monitoring	Vorschlaege fuer Metriken, die Performanceverbesserungen messbar machen
376	Cursor AI	2026-02-02	Flow-Validierung	Regressionstests fuer Save-im-Hintergrund-Flow bei manueller + importierter Flugfassung
377	ChatGPT	2026-02-02	Integrations-Test	E2E-Szenarien fuer Deep-Link-Navigation aus Notification-Bannern
378	DeepSeek	2026-02-02	Zuverlaessigkeit	Testschritte fuer App-Neustart mit pending reminders pro User
379	Cursor AI	2026-02-02	Zeitrobustheit	eine Teststrategie fuer Zeitzonenumwandlung während aktiver Reminder
380	ChatGPT	2026-02-02	Build-Qualitaet	Akzeptanzkriterien fuer Build-Pipeline in development, preview + production
381	DeepSeek	2026-02-02	Betrieb	ein Troubleshooting-Dokument fuer APNs/FCM Credential-Probleme
382	Cursor AI	2026-02-02	Push-Verifikation	eine Schritt-fuer-Schritt-Pruefung, ob EAS-Push-End-to-End korrekt arbeitet
383	ChatGPT	2026-02-02	Qualitaetsrealismus	eine handhabbare Testabdeckung-Definition fuer ein Schulprojekt dieser Groesse
384	DeepSeek	2026-02-02	Schriftliche Nachweise	, wie Testergebnisse in der Diplomarbeit nachvollziehbar dargestellt werden sollten
385	ChatGPT	2026-02-02	Zielgruppenorientierung	ungen gut lesbaren Abschnitt zu Notification-Architektur fuer Nicht-Entwickler + Entwickler
386	DeepSeek	2026-02-02	Qualitaetsdokumentation	die Kapitelstruktur fuer Testing, Qualitaet + offene Risiken
387	Cursor AI	2026-02-02	Stilkonsistenz	ein konsistentes Wording fuer deutsche + englische Fachbegriffe im Dokument

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
388	ChatGPT	2026-02-02	Reflexion	eine Vorlage fuer den Abschnitt Lessons Learned mit technischer Tiefe
389	DeepSeek	2026-02-02	Nachweisfuehrung	ein Schema zur Zuordnung von Stundenbuchungen zu implementierten Features
390	Cursor AI	2026-02-02	Methodik	einen professionellen Methoden-Abschnitt fuer iterative Entwicklung im Zweierteam
391	ChatGPT	2026-02-02	Anhangsplanung	einen Vorschlag fuer den Anhang mit SQL-Skripten, Testprotokollen + Prompt-Tabellen
392	DeepSeek	2026-02-02	Compliance	, wie KI-Unterstuetzung transparent + regelkonform in A.6 dokumentiert werden soll
393	Cursor AI	2026-02-02	Wissenschaftliche Sorgfalt	eine saubere Trennung zwischen belegbaren Prompts + plausibel rekonstruierten Prompts
394	ChatGPT	2026-02-02	Transparenz	einen neutralen Hinweistext, dass rekonstruierte Eintraege entsprechend gekennzeichnet sind
395	DeepSeek	2026-02-02	Chronologische Konsistenz	ein Datumsraster, das zu den Arbeitsphasen in der Stundenliste passt
396	Cursor AI	2026-02-03	Einschraenkung Expo Go	Expo Go - alle Funktionen
397	Cursor AI	2026-02-03	Performance-Optimierung	wird laggy - Ursachen finden + beheben
398	Cursor AI	2026-02-03	Tab-spezifischer Navbar-Fix	Im Data-/Stats-Tab geht Navbar nicht
399	Cursor AI	2026-02-04	Umsetzung Performance-Plan	Implement the plan (Performance)
400	Cursor AI	2026-02-04	Planumsetzung	Implement the plan (Stats/Navbar)
401	Cursor AI	2026-02-05	Orientierung im Projektverzeichnis	In welchen Ordner ist die schriftliche Diplomarbeit
402	Cursor AI	2026-02-05	Save-Flow + Navbar-Fix	Save dauert lange; Navbar in Stats/Settings kaputt; Logs analysieren
403	Cursor AI	2026-02-06	Projektdokumentation fuer Diplomarbeit	einen Ordner fuer die schriftliche Diplomarbeit mit komplettem Projektprotokoll

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
404	Cursor AI	2026-02-06	Performance- und Navbar-Diagnose	Laedt langsam (Flughaefen usw.), Navbar reagiert nicht, haengt - Ursachen
405	Cursor AI	2026-02-06	Planumsetzung	Implement the plan (Save + Navbar)
406	Cursor AI	2026-02-07	Planumsetzung	Implement the plan (Navigation/Performance)
407	Cursor AI	2026-02-08	Planumsetzung	Implement the plan (Navbar Fix)
408	Cursor AI	2026-02-08	Diplomarbeit A.6 Tabelle	Tabelle A.6 Einsatz von KI-Tools - alle Anfragen von Projektbeginn dokumentieren
409	Cursor AI	2026-02-09	Wartbarer Hotfix-Prozess	ein Playbook fuer schnelle Hotfixes ohne Architekturverschlechterung
410	ChatGPT	2026-02-09	Release-Vorbereitung	eine Stabilitaets-Checkliste vor Publish Ready
411	DeepSeek	2026-02-09	Bug-Management	ein Priorisierungsmodell fuer verbleibende Bugs nach Severity + Reproduzierbarkeit
412	Cursor AI	2026-02-09	Qualitaetsgate	eine Abnahmeregel fuer Navigation, die auf allen Tabs fehlerfrei laufen muss
413	ChatGPT	2026-02-09	Release-Testing	einen Plan fuer Last-Minute-Regressionstests auf iOS + Android
414	DeepSeek	2026-02-09	Erkenntnisgewinn	eine Analyse, warum Performanceprobleme oft erst bei schneller Bedienung sichtbar werden
415	Cursor AI	2026-02-09	Production Readiness	eine Strategie fuer kontrolliertes Logging im Release-Build ohne sensible Daten
416	ChatGPT	2026-02-09	Qualitaetsmanagement	ein sauberes Done-Kriterium fuer Performance-Fixes
417	DeepSeek	2026-02-09	Balance Qualitaet/Speed	Vorschlaege fuer UI-Polishing, die Performance nicht negativ beeinflussen
418	Cursor AI	2026-02-09	Abschlusspruefung	ein Stabilitaetsprotokoll fuer die finalen Funktionalitaeten vor Abgabe
419	ChatGPT	2026-02-09	Wissenssicherung	eine Zusammenfassung der wichtigsten Performance-Learnings fuer die Diplomarbeit
420	DeepSeek	2026-02-09	Roadmap	Optimierungspotenziale fuer eine spaetere Produktivversion
421	Cursor AI	2026-02-09	Teamkommunikation	ein Muster fuer reproduzierbare Bugreports mit Steps, Expected, Actual, Logs

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
422	ChatGPT	2026-02-09	Qualitaetsstrategie	eine Entscheidungsgrundlage, wann ein Bugfix per Refactor statt Quickfix gelöst werden soll
423	DeepSeek	2026-02-09	Regression Control	eine Nachtest-Strategie nach jedem kritischen Navigation-Fix
424	Cursor AI	2026-02-09	Transparenz	ein leichtgewichtiges Performance-Dashboard für Entwicklungsstand ab
425	ChatGPT	2026-02-09	Dokumentation	ein Abschlussstatement für den Performance-Block mit konkreten Resultaten
426	Cursor AI	2026-02-09	Projektabchluss	ein formales Abnahmeprotokoll für den Stand publish ready
427	ChatGPT	2026-02-09	Risikodokumentation	die wichtigsten Restrisiken trotz stabiler Release-Version
428	DeepSeek	2026-02-09	Wartungsplanung	einen Plan für Nachpflege nach Erstabgabe der App
429	Cursor AI	2026-02-09	Incident Management	eine Struktur für Fehlerklassen kritisch, hoch, mittel, niedrig mit SLA-Idee
430	ChatGPT	2026-02-09	Produktfeedback	ein Nutzerfeedback-Template für Pilotnutzer vor finaler Abgabe
431	DeepSeek	2026-02-09	UX-Evaluation	ein Bewertungsraster für UX-Qualität im Projektkontext
432	Cursor AI	2026-02-09	Codehygiene	ein Audit für offene TODO-Kommentare im Code vor finalem Commit
433	ChatGPT	2026-02-09	Sicherheitsniveau	eine Checkliste für Security-Basics im finalen App-Stand
434	DeepSeek	2026-02-09	Mehrwertanalyse	eine Abschlussanalyse, welche Features den größten Nutzwert erzeugen
435	Cursor AI	2026-02-09	Testabschluss	ein Ergebnisprotokoll für den finalen Integrations-Testtag
436	ChatGPT	2026-02-09	Executive Summary	eine professionelle Management-Zusammenfassung der technischen Reife
437	DeepSeek	2026-02-09	Demo-Planung	ein Vorgehen für reproduzierbare Demonstration bei der Präsentation
438	Cursor AI	2026-02-09	Freigabeprozess	Kriterien, wann ein Build als abgabefähig markiert werden darf

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
439	ChatGPT	2026-02-09	Pflichtabdeckung	eine finale QA-Liste fuer alle Must-have-Funktionen laut Pflichtenheft
440	Cursor AI	2026-02-09	Datenpflege	eine Qualitaetskontrolle fuer grosse Tabellen mit vielen Prompt-Eintraegen
441	ChatGPT	2026-02-09	Branch-Strategie	ein Git-Vorgehen fuer sicheren Merge von Ole + Boris in Main ohne Funktionsverlust
442	DeepSeek	2026-02-09	Merge-Sicherheit	ein Konfliktloesungsprotokoll fuer kritische Dateien wie store, navigation + notifications
443	Cursor AI	2026-02-09	Integritaetspruefung	eine Checkliste fuer Pre-Merge-Validierung aller Kernfunktionen
444	ChatGPT	2026-02-09	Dokumentierbarkeit	eine Strategie, wie commit messages fuer die Diplomarbeit als Chronologie nutzbar bleiben
445	DeepSeek	2026-02-09	Konsistenz	einen Plan fuer finalen Sync von Dokumentation, Code + Stundenaufzeichnung
446	Cursor AI	2026-02-09	Projektabschluss	ein Abschlussprotokoll fuer den finalen Push + Branch-Status
447	ChatGPT	2026-02-09	A.6 Zusammenfassung	eine professionelle Zusammenfassung der KI-gestuetzten Entwicklungsarbeit fuer den Anhang
448	DeepSeek	2026-02-09	Verteidigungsfaehigkeit	ein Vorgehen, um bei Rueckfragen der Pruefer jede Prompt-Kategorie begruenden zu koennen
449	Cursor AI	2026-02-09	Vollstaendigkeitspruefung	eine finale Validierung, ob alle Pflichtpunkte der A.6-Tabelle ausgefuellt sind
450	ChatGPT	2026-02-09	Formeller Abschluss	eine Abschlussformulierung fuer die A.6-Dokumentation mit Hinweis auf kontinuierliche KI-N
451	Cursor AI	2026-02-10	Praesentation	Praesi-Agenda + Storyline
452	ChatGPT	2026-02-10	Praesentation	Demo-Flow (Happy Path) + Backup-Szenario
453	ChatGPT	2026-02-10	Praesentation	Elevator Pitch + Problem/Nutzen 30s
454	ChatGPT	2026-02-11	Praesentation	Live-Demo Risiken + Mitigations
455	Cursor AI	2026-02-11	Praesentation	Folien: Architektur-Uebersicht (Frontend/Backend/DB)

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
456	Cursor AI	2026-02-11	Praesentation	Folien: Datenmodell-Highlights (Flights/Docs/Reminders)
457	Cursor AI	2026-02-12	Praesentation	Folien: KI-Nutzung sauber begruenden (A.6)
458	ChatGPT	2026-02-12	Praesentation	Sprechtext: 5-Minuten Pitch
459	DeepSeek	2026-02-12	Praesentation	Q&A-Liste: typische Prueferfragen + Antworten
460	ChatGPT	2026-02-13	Doku	Kapitel: Systemgrenzen + Nicht-Ziele
461	ChatGPT	2026-02-13	Doku	Kapitel: Sicherheitskonzept (RLS/Storage/Token)
462	DeepSeek	2026-02-13	Doku	Kapitel: Testkonzept + Testprotokolle
463	ChatGPT	2026-02-14	Doku	Kapitel: Deployment/Build-Guide (EAS)
464	ChatGPT	2026-02-14	Doku	Kapitel: Wartung/Updates (Monatlich + Hotfix)
465	Cursor AI	2026-02-14	Doku	Anhang: SQL Snippets + Policies
466	DeepSeek	2026-02-15	Doku	Anhang: Glossar + Abkuerzungen
467	ChatGPT	2026-02-15	Doku	Kapitel: Lessons Learned (tech + process)
468	DeepSeek	2026-02-15	Doku	Format-Check: Einheitliche Begriffe/Schreibweise
469	Cursor AI	2026-02-16	Testing	Smoke-Testliste (Login->Add Flight->Map->Docs->Reminders)
470	ChatGPT	2026-02-16	Testing	Regression: Background-Save + Navigation
471	Cursor AI	2026-02-16	Testing	Testfaelle: Zeitzone/DST + Overnight Flights
472	Cursor AI	2026-02-17	Testing	Testfaelle: Company Owner/Worker Rechte
473	DeepSeek	2026-02-17	Testing	Testfaelle: Upload PDF/JPG + Delete/Undo
474	ChatGPT	2026-02-17	Testing	Testdaten: Airports/Routes Edge Cases
475	Cursor AI	2026-02-18	Testing	Bugreport-Template (Steps/Expected/Actual/Logs)
476	DeepSeek	2026-02-18	Testing	Testprotokoll: Build + Geraet + Ergebnis
477	ChatGPT	2026-02-18	Testing	Abnahmekriterien-Matrix vs Pflichtenheft
478	ChatGPT	2026-02-19	Release/Build	EAS Build-Profile: dev/preview/prod
479	Cursor AI	2026-02-19	Release/Build	iOS Install-Problem: Vertrauen/Signierung
480	Cursor AI	2026-02-19	Release/Build	Android Permissions: POST_NOTIFICATIONS

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
481	Cursor AI	2026-02-20	Release/Build	Versioning: SemVer + Build Nummer
482	Cursor AI	2026-02-20	Release/Build	Release-Checklist: Secrets/Keys/Env
483	DeepSeek	2026-02-20	Release/Build	Crash-Handling: Error Boundaries + Logging
484	Cursor AI	2026-02-21	Release/Build	CI-Plan light (Lint/Typecheck/Tests)
485	DeepSeek	2026-02-21	Release/Build	Rollback-Plan: letzte stabile APK/IPA
486	DeepSeek	2026-02-21	Release/Build	Changelog: wichtigste Änderungen seit Beta
487	Cursor AI	2026-02-22	UX/Polish	Loading States: Skeleton statt Spinner
488	Cursor AI	2026-02-22	UX/Polish	Empty States: Home/Map/Docs/Notes
489	Cursor AI	2026-02-22	UX/Polish	Microcopy: Fehlertexte kurz + konkret
490	DeepSeek	2026-02-23	UX/Polish	Haptics: Success/Warning/Error sinnvoll
491	Cursor AI	2026-02-23	UX/Polish	Accessibility: Kontrast + Touch Targets
492	ChatGPT	2026-02-23	UX/Polish	Navigation: canGoBack + Safe Back
493	ChatGPT	2026-02-24	UX/Polish	List Performance: memo + key stability
494	ChatGPT	2026-02-24	UX/Polish	Map: Completed Flights -> History Button
495	ChatGPT	2026-02-24	UX/Polish	Stats: KPI Cards + verständliche Labels
496	ChatGPT	2026-02-25	Security/Privacy	Privacy Note: minimale PII + Logging-Regeln
497	ChatGPT	2026-02-25	Security/Privacy	RLS Review: owner-only + company role gates
498	ChatGPT	2026-02-25	Security/Privacy	Storage: signed URLs + expires
499	Cursor AI	2026-02-26	Security/Privacy	Token Handling: refresh + revoke
500	DeepSeek	2026-02-26	Security/Privacy	Input Validation: flightNumber/notes/fileName
501	DeepSeek	2026-02-26	Security/Privacy	Threat Model light: Top 5 Risiken
502	ChatGPT	2026-02-27	Security/Privacy	Backup/Recovery: Supabase basics
503	ChatGPT	2026-02-27	Security/Privacy	Data Retention: delete account + cleanup
504	Cursor AI	2026-02-27	Security/Privacy	Compliance Text: KI-Einsatz transparent
505	Cursor AI	2026-02-28	Performance	Profiling: slow screens identifizieren
506	Cursor AI	2026-02-28	Performance	API Timeouts + Retry policy
507	ChatGPT	2026-02-28	Performance	Prefetch: Airports + Next Flights
508	Cursor AI	2026-03-01	Performance	Batching: Supabase Queries vermeiden
509	ChatGPT	2026-03-01	Performance	Reduce re-renders: store subscriptions

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
510	DeepSeek	2026-03-01	Performance	Image optimierung: thumbnails + caching
511	Cursor AI	2026-03-02	Performance	Map rendering: polyline simplification
512	Cursor AI	2026-03-02	Performance	App Start: Lazy module loading
513	DeepSeek	2026-03-02	Performance	Memory: large lists + virtualization
514	Cursor AI	2026-03-03	Feature-Finishing	Email Import: parser edge cases
515	Cursor AI	2026-03-03	Feature-Finishing	OCR: confidence marking + fallback
516	Cursor AI	2026-03-03	Feature-Finishing	Company Flights: Trip-Details Deep Link fix
517	ChatGPT	2026-03-04	Feature-Finishing	Documents: metadata bucket strategy
518	ChatGPT	2026-03-04	Feature-Finishing	Notifications: duplicate prevention final
519	ChatGPT	2026-03-04	Feature-Finishing	Reminders: reschedule on reboot
520	ChatGPT	2026-03-05	Feature-Finishing	Templates: checklist presets (business/private)
521	ChatGPT	2026-03-05	Feature-Finishing	Achievements: unlock logic sanity check
522	Cursor AI	2026-03-05	Feature-Finishing	Stats: distance/duration recalculation
523	Cursor AI	2026-03-06	Projektmanagement	Stundenliste -> Feature Mapping
524	ChatGPT	2026-03-06	Projektmanagement	Roadmap Update: Restaufwand + Risiko
525	Cursor AI	2026-03-06	Projektmanagement	Done-Definition final (Code+Tests+Doku)
526	DeepSeek	2026-03-07	Projektmanagement	PR Review Checklist kurz
527	Cursor AI	2026-03-07	Projektmanagement	Merge Plan: Ole/Boris -> main
528	DeepSeek	2026-03-07	Projektmanagement	Tech Debt Liste: top 10
529	ChatGPT	2026-03-08	Projektmanagement	Kommunikation: Stakeholder Update
530	ChatGPT	2026-03-08	Projektmanagement	Abnahmeprotokoll final
531	DeepSeek	2026-03-08	Projektmanagement	Backup Plan: Demo ohne Internet
532	ChatGPT	2026-03-09	App-Store/Assets	Screenshot-Liste: required screens
533	DeepSeek	2026-03-09	App-Store/Assets	App Beschreibung: kurz + lang
534	Cursor AI	2026-03-09	App-Store/Assets	Keywords/Tags Vorschlaege
535	DeepSeek	2026-03-10	App-Store/Assets	Icon/Branding Check: konsistent
536	ChatGPT	2026-03-10	App-Store/Assets	Showcase Video: Storyboard
537	Cursor AI	2026-03-10	App-Store/Assets	Release Notes: 3 bullets
538	DeepSeek	2026-03-11	App-Store/Assets	Onboarding Copy: 4 Screens
539	ChatGPT	2026-03-11	App-Store/Assets	FAQ Text: Datenschutz/Permissions

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
540	ChatGPT	2026-03-11	App-Store/Assets	Support Text: Kontakt + Hinweise
541	ChatGPT	2026-03-12	UML/Diagramme	Use-Case Diagramm: Kernflows
542	Cursor AI	2026-03-12	UML/Diagramme	Aktivitaetsdiagramm: Flug hinzufuegen
543	Cursor AI	2026-03-12	UML/Diagramme	Aktivitaetsdiagramm: Reminder planen
544	DeepSeek	2026-03-13	UML/Diagramme	Sequenzdiagramm: Import -> Save -> Reminder
545	ChatGPT	2026-03-13	UML/Diagramme	Komponentendiagramm: UI/Store/Services
546	DeepSeek	2026-03-13	UML/Diagramme	ER-Modell: Tabellen + Beziehungen
547	Cursor AI	2026-03-14	UML/Diagramme	Datenfluss: Notification Tap -> Navigation
548	DeepSeek	2026-03-14	UML/Diagramme	State Machine: Add Flight Import Flow
549	Cursor AI	2026-03-14	UML/Diagramme	Diagramm-Check: Symbole/Notation korrekt
550	Cursor AI	2026-03-15	Refactoring	Services Layer Cleanup: supabase wrapper
551	ChatGPT	2026-03-15	Refactoring	Store Actions: naming + error model
552	Cursor AI	2026-03-15	Refactoring	Hooks: reuse + reduce duplication
553	Cursor AI	2026-03-16	Refactoring	TypeScript Types: stricter + nullable audit
554	DeepSeek	2026-03-16	Refactoring	Dead Code: remove legacy flightService
555	ChatGPT	2026-03-16	Refactoring	Lint Rules: consistent formatting
556	Cursor AI	2026-03-17	Refactoring	Async flows: cancelation + abort
557	ChatGPT	2026-03-17	Refactoring	Centralized constants: enums
558	DeepSeek	2026-03-17	Refactoring	Docs: update after refactor
559	DeepSeek	2026-03-18	Risk/Contingency	Risikoanalyse: iOS push without dev account
560	Cursor AI	2026-03-18	Risk/Contingency	Fallback: Expo Go limitations doc
561	Cursor AI	2026-03-18	Risk/Contingency	Offline Mode: what works + what not
562	ChatGPT	2026-03-19	Risk/Contingency	Third-party API limits: airport search
563	DeepSeek	2026-03-19	Risk/Contingency	Security risk: public bucket misconfig
564	ChatGPT	2026-03-19	Risk/Contingency	Time risk: scope cut suggestions
565	Cursor AI	2026-03-20	Risk/Contingency	Demo failure: pre-recorded backup
566	Cursor AI	2026-03-20	Risk/Contingency	Data loss risk: local cache strategy
567	ChatGPT	2026-03-20	Risk/Contingency	Post-mortem: biggest surprises
568	Cursor AI	2026-03-21	Qualitaetsnachweise	Pflichtenheft-Abgleich: FA-01..FA-09
569	Cursor AI	2026-03-21	Qualitaetsnachweise	Traceability: requirement -> commit -> test

*Fortsetzung*

Nr.	Tool	Datum	Zweck	Prompt (Stichworte)
570	ChatGPT	2026-03-21	Qualitaetsnachweise	Testprotokolle: sign-off
571	Cursor AI	2026-03-22	Qualitaetsnachweise	Performance numbers: before/after
572	DeepSeek	2026-03-22	Qualitaetsnachweise	Buglist: fixed vs open
573	Cursor AI	2026-03-22	Qualitaetsnachweise	Known limitations: honest list
574	DeepSeek	2026-03-23	Qualitaetsnachweise	User feedback summary
575	ChatGPT	2026-03-23	Qualitaetsnachweise	Security checks summary
576	ChatGPT	2026-03-23	Qualitaetsnachweise	A.6 Konsistenzcheck
577	ChatGPT	2026-03-24	DB/SQL	SQL: Indizes review (user_flights, reminders, documents)
578	Cursor AI	2026-03-24	DB/SQL	RPC: get_user_stats verify outputs
579	Cursor AI	2026-03-24	DB/SQL	Migration: nullable -> not null plan
580	Cursor AI	2026-03-25	DB/SQL	Constraints: unique keys (iata/flight_id)
581	ChatGPT	2026-03-25	DB/SQL	Cleanup Script: orphaned reminders/documents
582	ChatGPT	2026-03-25	DB/SQL	Seed Data: dev airports subset
583	Cursor AI	2026-03-25	DB/SQL	RLS Tests: scripted selects/inserts
584	Cursor AI	2026-03-25	DB/SQL	Storage Paths: naming convention enforce
585	ChatGPT	2026-03-25	DB/SQL	DB Diagramm Text: Tabellenbeschreibung kurz