Pflichtenheft

Projekt: "digitales Fluglogbuch”

Version: 3**.0**

Status: <ENTWURF>

Historie der Dokumentversionen

Version	Datum	Autor	Änderungsgrund / Bemerkungen
1.0	24.04.2024	Participant	Ersterstellung
2.0	13.05.2024	Participant	- Person wurde am Logbuch gehängt
3.0	07.04.2024	Participant	- UML Diagram angepasst - MVC Logik hinzugefügt - Frontend Skizzen hinzugefügt Controller/DAO Klassen hinzugefügt

Einleitung

Allgemeines

Zweck und Ziel dieses Dokuments

Dieses Pflichtenheft beschreibt die spezifischen Anforderungen und Funktionalitäten für die Entwicklung eines digitalen Fluglogbuch-Managers. Es dient als Leitfaden für das Projektteam und den Kunden, um einen klaren Überblick über die erwarteten Anforderungen, Pläne und Deadlines zu erhalten, die im Rahmen der Produktplanung und -erstellung festgelegt wurden. Darüber hinaus werden in diesem Dokument die Ziele des Produkts sowie die Anforderungen an den Anwendungsbereich und die Systemübersicht detailliert beschrieben.

Abkürzungen

- SEP: Single-Engine Piston
- UL: Ultraleichtflugzeug
- LAPL(A): Leichtluftfahrzeug-Pilotenlizenz (Aeroplane)
- PPL(A): Privatpilotenlizenz (Aeroplane)
- PIC: Pilot in Command
- IFR: Instrumentenflug (*instrument flight rules*)
- FI: Fluglehrer (Flight Instructor) Dual:
- Andere Person ist PIC

Verteiler und Freigabe

Verteiler für dieses Lastenheft

Rolle / Rollen	Name	Matrikel Nummer	Freigegeben von
Projektleiter Entwickler	Participant	1234	x
Entwickler	Participant	1234	
Entwickler	Participant	1234	
Entwickler	Participant	1234	
Entwickler	Participant	1234	

Konzept und Rahmenbedingungen

Ziele und Nutzen des Anwenders

Die Ziele und der Nutzen für den Anwender liegen in der Vereinfachung des Flugbuchführungsprozesses sowie der Verbesserung der Effizienz und Genauigkeit bei der Aufzeichnung von Flugdaten. Der digitale FluglogbuchManager soll es dem Anwender ermöglichen, Flugdaten präzise zu erfassen, Zeit zu sparen und eine bessere Kontrolle über seine Flugerfahrung zu erhalten.

Benutzer / Zielgruppe

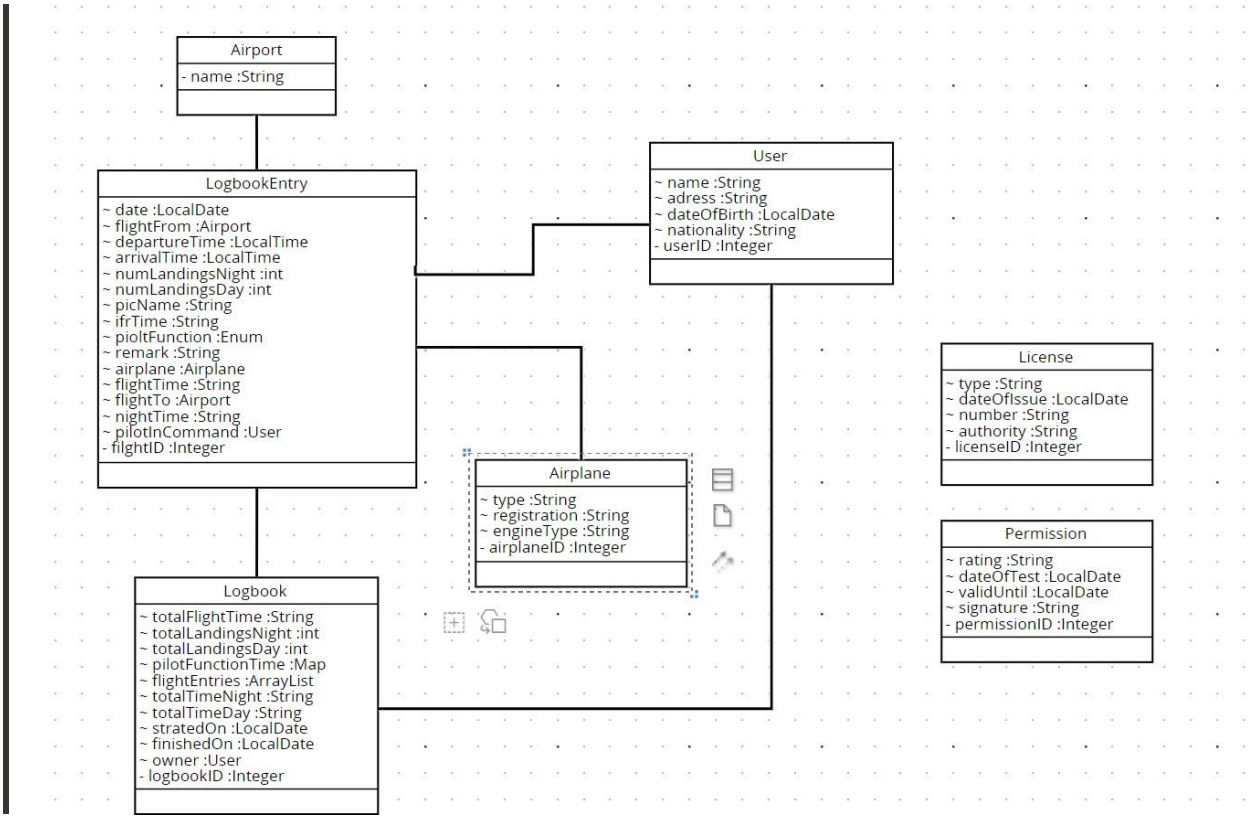
Die Zielgruppe für den digitalen Fluglogbuch-Manager umfasst Privatpiloten, Berufspiloten, Flugschulen und Flugunternehmen, die eine zuverlässige Lösung zur Verwaltung ihrer Flugdaten benötigen.

Systemübersicht

Allgemeines:

- Das digitale Fluglogbuch-Management-Programm wird als StandaloneAnwendung in Java entwickelt.
- Das Programm wird auf verschiedenen Betriebssystemen ausführbar sein, darunter Windows, macOS und Linux.
- Während des Fluges wird keine Verbindung zu einer serverseitigen Datenbank hergestellt. Stattdessen werden sämtliche Flugdaten lokal auf dem Endgerät des Benutzers gespeichert.
- Hierfür wird eine Schnittstelle zu einer lokalen SQLite-Datenbank implementiert. Diese ermöglicht es dem Benutzer, seine Daten bequem im Systemspeicher seines Geräts zu speichern und zu verwalten, ohne auf eine externe Datenbankverbindung angewiesen zu sein.
- Durch die Nutzung einer lokalen Datenbanklösung wird die Abhängigkeit von einer kontinuierlichen Internetverbindung eliminiert, was besonders wichtig ist, da während des Fluges möglicherweise keine zuverlässige Internetverbindung verfügbar ist.
- Die lokale Speicherung gewährleistet zudem eine hohe Datensicherheit und ermöglicht es dem Benutzer, auch offline auf seine Flugdaten zuzugreifen und diese zu bearbeiten.
- Bei der Implementierung wird das V-Modell verwendet.

UML-Diagramm für die Hauptklassen



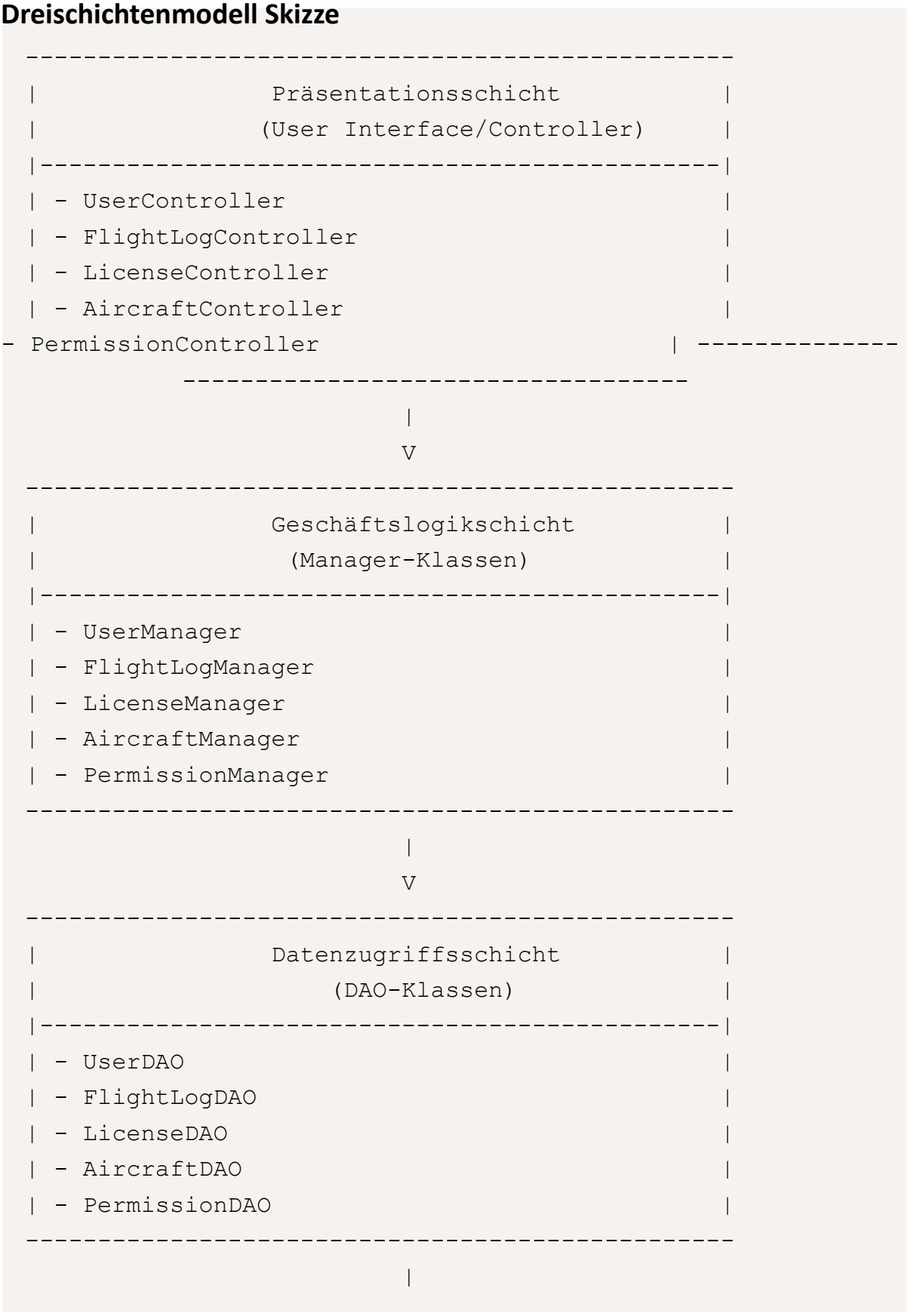
- Die Architektur des Programms ist auf der Grundlage der Drei-Schichten-Architektur aufgebaut.

Um eine klare Vorstellung von der Architektur unseres Projekts zu bekommen, ist es hilfreich, eine Skizze des Dreischichtenmodells zu erstellen. Das Modell besteht aus folgenden Schichten:

1. **Präsentationsschicht (Frontend)**: Interagiert mit dem Benutzer und sendet Anforderungen an die Geschäftslogik.
2. **Geschäftslogikschicht (Manager-Klassen)**: Verarbeitet die Anwendungslogik und führt Geschäftsregeln aus.
3. **Datenzugriffsschicht (DAO-Klassen)**: Interagiert mit der Datenbank, um CRUD-Operationen auszuführen.

Hier ist eine vereinfachte Darstellung dieser Struktur für unser Projekt:

Dreischichtenmodell Skizze



```
                          V
-----------------------------------------------
|                  Datenbankschicht             |
|               (SQLite-Datenbank)              |
|-----------------------------------------------|
| - Tabellen: Users, FlightLogs, Licenses,      |
|   Aircrafts, Permissions                      |
-----------------------------------------------
```

Detaillierte Beschreibung der Schichten und Klassen

- Das Programm ist nach dem Model View Controller Pattern aufgestellt
  a. View-Komponente
  b. Controller
  c. Model-Komponente (in unserem Fall sind Manager Klassen)

Implementierung der Benutzeroberfläche (GUI)

View -Implementierung

1. Die View-Komponente beinhaltet die Klassen:
   a. Jede View ist mit einem Controller verbunden.
   b. Jeder View ist eine .FXML Datei mit JAVAFX erstellt
2. Das Wechseln zwischen den Views wird durch eine Sidebar ermöglicht
3. Die UI ermöglicht dem User vollständigen Zugriff auf seine gespeicherten Daten durch die Tabellen in jedem View.
   (Logbucheinträge, Userdaten, Lizenzen, etc.)

Skizzen:

- Elemente in [ ] sind Buttons

Beispiel-Layout für MainView

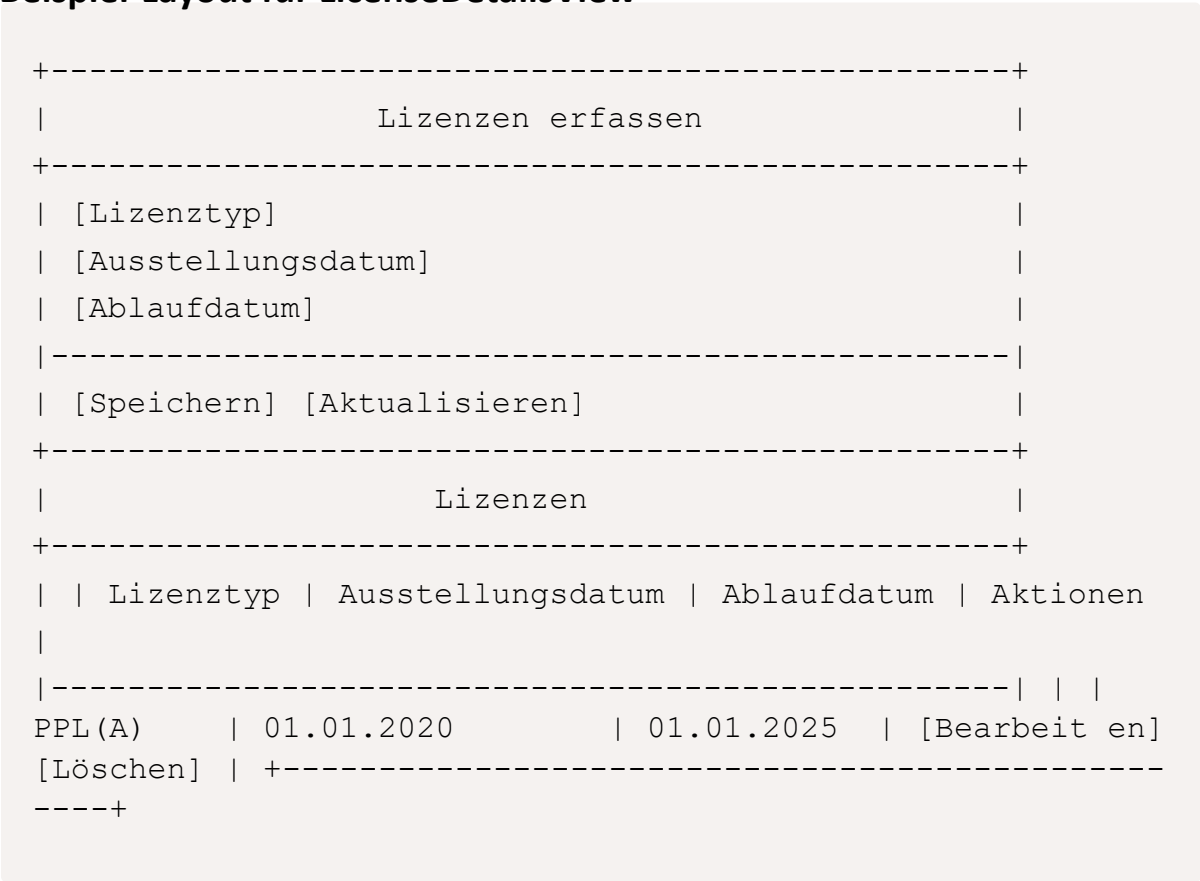
```
+-------------------------------------------------+
|                    Hauptmenü                    |
```

```
+-------------------------------------------------+
| Analyse                                         |
|                                                 |
|                                                 |
|                          [Neuer Flug hinzugügen]
|
|                                                 |
|                                                 |
| [Benutzerdaten verwalten]                       |
| [Lizenzen verwalten]                            |
| [Berechtigungen verwalten]                      |
| [Fluglogbucheinträge verwalten]                 |
| [Flugzeugdaten verwalten]                       |
+-------------------------------------------------+
```

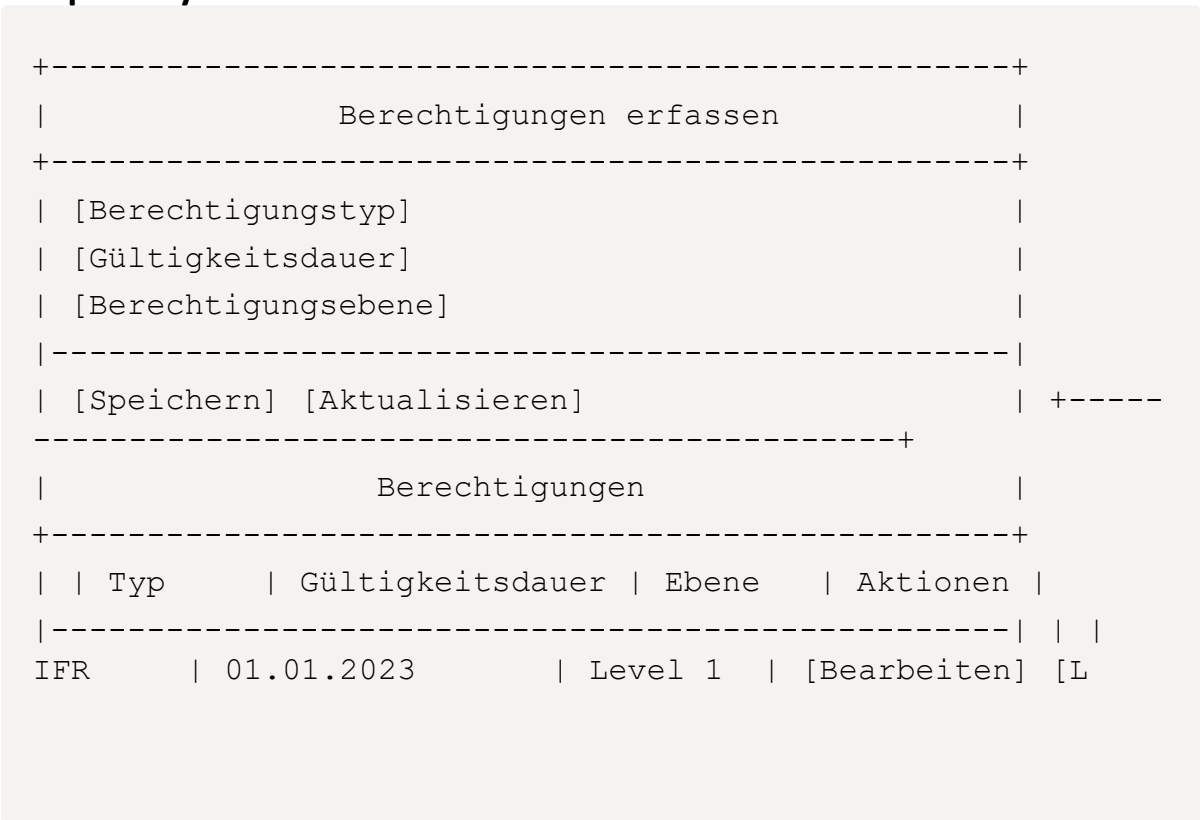
Beispiel-Layout für UserDetailsView

```
+-------------------------------------------------+
|               Benutzerdaten erfassen            |
+-------------------------------------------------+
| [Name]                                          |
| [Anschrift]                                     |
| [Geburtsdatum]                                  |
| [Geburtsort]                                    |
| [Nationalität]                                  |
|-------------------------------------------------|
| [Speichern] [Aktualisieren]                     |
+-------------------------------------------------+ |
Benutzerdaten                   |  # Hie r ist eine Tabelle
+-------------------------------------------------+
| | Name     | Anschrift | Geburtsdatum | Aktionen | |-----
-----------------------------------------| | | Max
Mustermann | Musterstraße 1 | 01.01.1990 | [Bearbei ten]
[Löschen] | +-----------------------------------------
----+
```

Beispiel-Layout für LicenseDetailsView



Beispiel-Layout für PermissionDetailsView



```
öschen] | +-----------------------------------------------
--+
```

Beispiel-Layout für FlightLogEntryView

```
+-------------------------------------------------+
|                Fluglogbuch-Einträge             |
+-------------------------------------------------+
| [Datum] [Flugzeugtyp] [Startzeit] [Landezeit]   |
| [Flugroute] [Bemerkungen]                       |
|-------------------------------------------------|
| [Neuen Eintrag hinzufügen]                      |
+-------------------------------------------------+
|                  Fluglogbucheinträge            |
+-------------------------------------------------+
| | Datum     | Flugzeugtyp  | Startzeit | Aktionen |
|-------------------------------------------------| | |
01.01.2024 | Cessna 172  | 08:00     | [Bearbeiten] [Lö
schen] | +-----------------------------------------------
-+
```

Beispiel-Layout für AircraftDetailsView

```
+-------------------------------------------------+
|               Flugzeugdaten erfassen            |
+-------------------------------------------------+
| [Flugzeugtyp]                                   |
| [Kennzeichen]                                   |
| [Klassifikation]                                | |-----
-----------------------------------------|
| [Speichern] [Aktualisieren]                     |
+-------------------------------------------------+
|                 Flugzeugdaten                   |
+-------------------------------------------------+
| | Typ        | Kennzeichen | Klassifikation | Aktionen |
```

```
|-------------------------------------------------| | |
Cessna 172 | D-ABCD      | SEP             | [Bearbeite n]
[Löschen] | +-----------------------------------------
-----+
```

Präsentationsschicht (Frontend)

- **UserController**: Bietet Methoden zur Benutzerinteraktion, wie das Hinzufügen und Anzeigen von Benutzern.
- **FlightLogController**: Ermöglicht das Hinzufügen, Bearbeiten und Anzeigen von Flugbucheinträgen.
- **LicenseController**: Verarbeitet Lizenzdaten und bietet Methoden zum Hinzufügen und Verwalten von Lizenzen.
- **AircraftController**: Verarbeitet Flugzeugdaten und bietet Methoden zum Hinzufügen und Verwalten von Flugzeugen.
- **PermissionController**: Verarbeitet Berechtigungsdaten und bietet Methoden zum Hinzufügen und Verwalten von Berechtigungen.

Geschäftslogikschicht (Manager-Klassen)

- **UserManager**: Implementiert die Geschäftslogik für Benutzeroperationen (z.B. Hinzufügen, Abrufen von Benutzern).
- **FlightLogManager**: Implementiert die Geschäftslogik für Flugbucheinträge (z.B. Hinzufügen, Abrufen von Einträgen).
- **LicenseManager**: Implementiert die Geschäftslogik für Lizenzen (z.B. Hinzufügen, Abrufen von Lizenzen).
- **AircraftManager**: Implementiert die Geschäftslogik für Flugzeuge (z.B.

  Hinzufügen, Abrufen von Flugzeugen).
- **PermissionManager**: Implementiert die Geschäftslogik für Berechtigungen (z.B. Hinzufügen, Abrufen von Berechtigungen).

Datenzugriffsschicht (DAO-Klassen)

- **UserDAO**: Führt CRUD-Operationen auf der `Users` Tabelle der Datenbank durch.

- **FlightLogDAO**: Führt CRUD-Operationen auf der `FlightLogs` Tabelle der Datenbank durch.
- **LicenseDAO**: Führt CRUD-Operationen auf der `Licenses` Tabelle der Datenbank durch.
- **AircraftDAO**: Führt CRUD-Operationen auf der `Aircrafts` Tabelle der Datenbank durch.
- **PermissionDAO**: Führt CRUD-Operationen auf der `Permissions` Tabelle der Datenbank durch.

Beispiel für eine Klasseninteraktion Hinzufügen eines neuen Benutzers

1. **UserController**
   - Der Benutzer klickt auf "Benutzer hinzufügen".
   - `UserController.addUser("John Doe")` wird aufgerufen.
2. **UserManager**
   - `UserController` ruft `UserManager.addUser("John Doe")` auf.
   - `UserManager` erstellt ein neues `User` Objekt und ruft `UserDAO.insertUser(user)` auf.
3. **UserDAO**
   - `UserDAO.insertUser(user)` führt die SQL-Operation aus, um den neuen Benutzer in die `Users` Tabelle der Datenbank einzufügen.
4. **Datenbankschicht**
   - Die `Users` Tabelle wird aktualisiert und der neue Benutzer wird gespeichert.

Diese Trennung der Schichten stellt sicher, dass die Präsentationslogik, Geschäftslogik und der Datenzugriff klar getrennt sind, was zu einer besser strukturierten und wartbaren Anwendung führt.

Implementierung der Datenbankverwaltung

1. Die Datenbankverwaltungskomponente umfasst verschiedene Klassen:

   a. Eine Klasse "DatabaseManager" zur Erstellung einer lokalen Datenbankinstanz b- Möglicherweise weitere Klassen, die zum Zeitpunkt der Erstellung dieses Pflichtenhefts noch nicht bekannt sind

2. Die in der Datenbank gespeicherten Daten und die Daten, die wieder abgerufen werden sollen:

   a. Benutzer Daten Auch begleitende Personen)

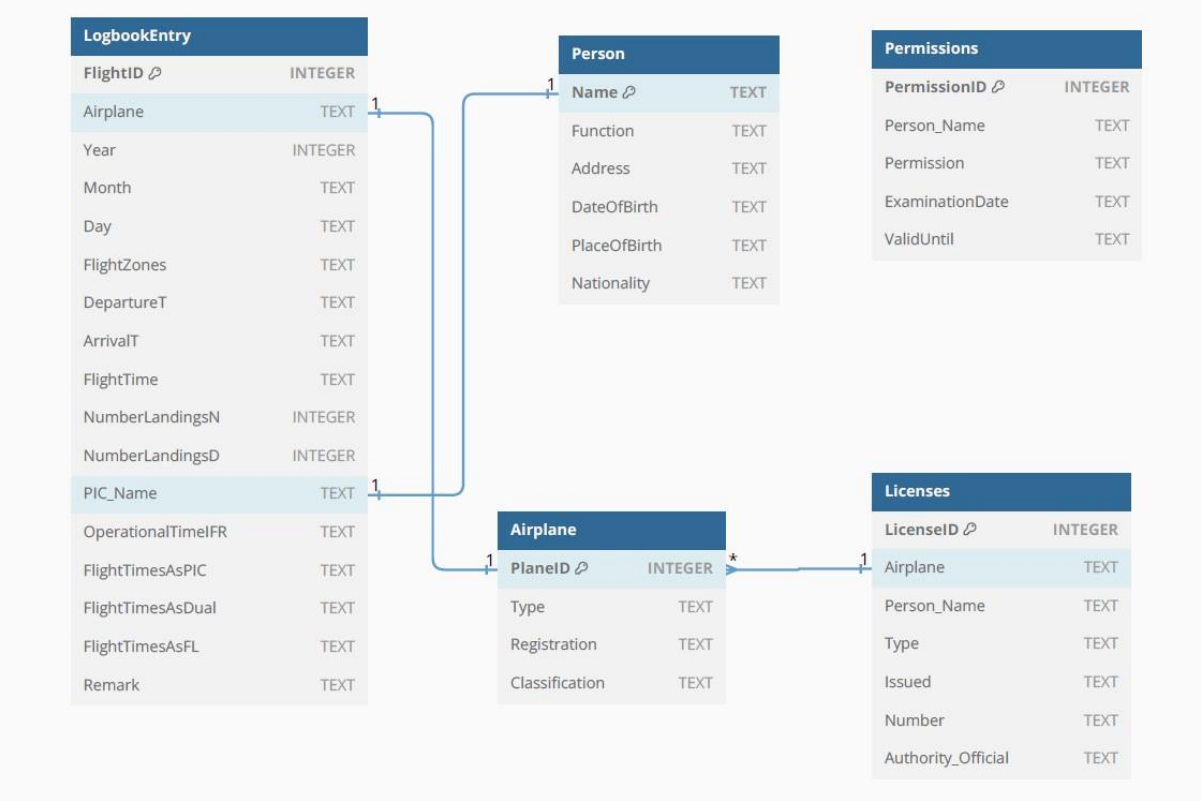
   b. Alle Daten aus dem Fluglogbuch

   c. Lizenzinformationen, die vom Benutzer eingetragen wurden

   d. Berechtigungen, die vom Benutzer eingetragen wurden e

   Flughafen Daten

Relationalen Datenmodell Diagramm zur Grundlegenden Softwarearchitektur



Sicherheitsarchitektur

- Da die Applikation nur lokal auf dem Gerät des Users läuft, sind externe Risiken minimal.
- Gefahren bestehen nur bei Online API□Zugriffen über externe Quellen wie z. B. Maps oder Flugzeugtypen aus dem Internet. Deswegen werden alle genutzten externen Quellen von unserem Team intensiv überwacht und geprüft. Intern auf dem Gerät des Users soll verhindert werden, dass die Software ein Sicherheitsrisiko für das Gerät

darstellt. Genau dieses Risiko wird durch ausreichende Tests auf verschiedenen Endgeräten minimiert.

Beschreibung der Anforderungen

Funktionale Anforderung

[Verwaltung von Pilotenstammdaten]

☐<PH1.1>*Daten erfassen*: Der User muss persönliche Daten, wie Name, Anschrift, Geburtsdatum, Geburtsort und Nationalität innerhalb der Anwendung hinterlegen können.

☐ <PH1.2>*Lizenzen erfassen*: Ermöglicht dem User neue Lizenzen zu erfassen.

☐ <PH1.3>*Berechtigungen erfassen*: Ermöglicht es dem User seine Berechtigungen zu erfassen und abzurufen.

☐<PH2.1>*Daten verwalten:* Der User kann seine persönlichen Daten verwalten und aktualisieren.

☐<PH2.2>*Lizenzen verwalten*: Der User kann seine Lizenzen abrufen und verwalten.

☐<PH2.3>*Berechtigungen verwalten:* Der User kann seine Berechtigungen abrufen und verwalten.

☐<PH3>*Daten speichern:* Das Speicherung aller angelegten Userdaten wird durch eine SQLite Datenbank gewährt.

[Verwaltung von Fluglogbucheinträgen]

☐<PH4>*Logbucheinträge erstellen:* Ermöglicht dem User über die GUI Flugdaten im digitalen Logbuch zu erstellen.

☐<PH5>*Logbucheinträge verwalten:* Ermöglicht dem User Einträge im digitalen Logbuch abzurufen und aktualisieren.

☐<PH6>*Logbucheinträge speichern:* Die Speicherung aller Logbucheinträge wird durch die SQLite Datenbank gewährt.

[Verwaltung von Flugzeugdaten]

☐<PH7>*Flugzeuge hinzufügen*: Der User kann Flugzeuge mit Typ, Kennzeichen und Klassifikation hinzufügen.

☐<PH8>*Flugzeuge verwalten*: Der User kann hinzugefügte Flugzeuge verwalten.

☐<PH9>*Flugzeuge speichern*: Die Speicherung aller Flugzeuge wird durch die SQLite Datenbank gewährt.

[Flugbuchführung]

☐<PH10>*Bemerkungen hinzufügen* Der User kann bei Logbucheinträgen eine Bemerkung schreiben.

[Benutzeroberfläche]

☐<PH11>Ein benutzerfreundliches GUI: Ein benutzerfreundliches GUI ermöglicht dem Benutzer die Nutzung des Programms sowohl im Flugzeug als auch am Boden. Im Flugzeug kann der Benutzer durch große Buttons schnell und mit wenigen Klicks einen Flugbucheintrag erstellen. Das Programm merkt sich dabei automatisch das Datum und die Zeiten. Außerdem werden zwei Buttons für die Landungen erstellt, die beim Drücken die Anzahl der Landungen inkrementieren.

[Andere]

☐<PH12>*Auswahl für gespeicherte Felder* Der User bekommt die Möglichkeit beim Erstellen eines Logbucheintrags Felder mit bereits gespeicherten Inhalten auszuwählen, wie gespeicherte Flugzeuge, Piloten und Flughäfen.

Anhang / Ressourcen

Glossar

PH - Pflichtenheft

SQL - Sequel Query Language

GUI - Graphical User Interface

ANF - Anforderung

UML - Unified Modeling Language