Kulturwissenschaft/en

Visualizing private plane travel

Michael Kaup

michael.kaup@student.htw-berlin.de
Matr.Nr.: s0589545

David Kirchner

david.kirchner@student.htw-berlin.de
Matr.Nr.: s0589227

Datum: 01 February 2026

Inhaltsverzeichnis

1 Einleitung

In Zeiten des Klimawandels ist der Flugverkehr ein wichtiges Thema, das immer wieder medial aufbereitet wird und eine breite Masse an Menschen betrifft. In den Medien wird unserer Auffassung nach vor allem viel darüber gesprochen, inwiefern Privatpersonen ihr Verhalten im Kontext Flugverkehr verändern oder gar einschränken sollten. Als Reaktion darauf hört man häufig, dass der Fokus mehr auf Privatflügen von Menschen mit hohem Vermögen liegen sollte. Nachzuvollziehen, welcher der beiden Bereiche, kommerzielle oder private Flüge, einen größeren Einfluss auf die Umwelt haben, ist allerdings gar nicht so einfach. Statistiken wie die „Anzahl der einsteigenden Flugpassagiere bei Inlandsflügen in Deutschland in den Jahren 2017 bis 2024“ [1] und „Anzahl der Privatflüge inklusive Geschäftsflüge in Deutschland von 2020 bis 2022“ [2] lassen zwar Vermutungen zu, ergeben aber aufgrund unterschiedlicher Zeiträume und Metriken kein klares Bild.

Diesem Problem wollten wir uns annähern. Uns war bekannt, dass es möglich ist über Dienste wie flightradar24 [3] live Flugdaten zu verfolgen. Unsere Annahme war, dass man eine API eines solchen Dienstes nutzen könne, um mittels Live-Daten und historischen Daten einen Flugtracker für prominente und reiche Personen zu erstellen. Dieser sollte Flugdaten dreidimensional darstellen, indem Flugzeuge über eine Miniaturversion der Erde fliegen. Dieser Tracker sollte sowohl am Computer als auch auf einem Smartphone mittels Vuforia Image Tracking im 2D- sowie AR-Raum dargestellt werden können. Darauf aufbauend hätten in Zukunft dann weitere private Flüge sowie kommerzielle Flüge und Frachtverkehr ebenfalls dargestellt werden können. Der Grund dafür, dass wir uns zuerst auf die Gruppe der reichsten Menschen der Welt konzentrieren wollten war vor allem die Neugierde dafür, ob so etwas möglich ist, Flugdaten unter diese Gruppe fallender Menschen zu tracken und zu visualisieren, als auch politische Voreingenommenheit.

Vor allem aufgrund der Schwierigkeit Flugdaten generell allumfänglich abzufragen sowie die Flugzeuge spezifischer Personen wie z.B. Elon Musk, Jeff Bezos oder Taylor Swift zu identifizieren, mussten wir unsere Umsetzung anpassen. Diese und weitere Problemstellungen werden im Kapitel Abschnitt 7 ausführlicher erläutert. Wir haben uns dazu entschieden den anfangs als in Aussicht stehenden Punkt der „weiteren privaten Flüge“ in den Fokus zu stellen und mussten uns aufgrund von Zeitknappheit auf eine 2D-Anwendung beschränken.

Am Ende des Projektes steht nun eine Visualisierung von Flugdaten der Flugzeugtypen, die am ehesten mit Privatflügen assoziiert werden. Gesammelt und gespeichert wurden diese Flugdaten mittels eines selbstgebauten Django Backends, das über eine REST-API Flugdaten abgefragt, nach Flugzeugtypen filtert und in einer eigens erstellten Datenbank speichert. Diese Dokumentation erläutert Aufbau und Funktion der Visualisierung sowie des Backends.

1.1 Aufgabenverteilung

- Michael:
  - Asset Recherche
  - Recherche nach Flugdaten APIs
  - Gestaltung der Unity Szene
  - Implementierung des UI und der Anzeige der Flugzeugdetails
  - Implementierung der Flugzeugauswahl
  - Implementierung der Player-Controls
  - Dokumentation:
    - Abschnitt 1
    - Abschnitt 3
    - Abschnitt 4
    - Abschnitt 5
    - Abschnitt 7

        - Abschnitt 8
- David
    - Asset Recherche
    - Recherche nach Flugdaten APIs
    - Implementierung des Django Backends
    - Implementierung der API Kommunikation
    - Implementierung der Datenstruktur/Datenbank
    - Implementierung der Datenvisualisierung in Unity
    - Dokumentation:
        - Abschnitt 1
        - Abschnitt 2
        - Abschnitt 6
        - Abschnitt 7
        - Abschnitt 8

2 Systemanforderungen

2.1 Unity Anwendung

Zum Ausführen der Anwendung wird ein grafikfähiger Windows-PC benötigt. Die grafischen Anforderungen der Anwendung sind gering und erfordern keine besonders starke Hardware. Falls die Anwendung direkt im Unity Editor ausgeführt werden soll, wird die Unity Version 6000.2.8f1 benötigt. Andernfalls wird zum Ausführen der Anwendung keine externe Software benötigt.

Es wird eine Internetverbindung vorausgesetzt, da die Daten die von der Anwendung vom Backend über das Internet bereitgestellt werden.

2.2 Backend

Für das Django-Backend wird ein Server mit statischer IP-Adresse benötigt, welcher in der Lage ist Python virtual-environments in der Version `3.12` auszuführen. Alternativ kann das Backend auch direkt (wie im deployment) per Docker-Container ausgeführt werden. Folgende python-dependencies zum Ausführen des Backends benötigt:

```py
"celery>=5.5.3",
"django>=5.2.8",
"django-celery-beat>=2.8.1",
"django-celery-results>=2.6.0",
"django-debug-toolbar>=6.1.0",
"django-filter>=25.2",
"djangorestframework>=3.16.1",
"markdown>=3.10",
"python-dotenv>=1.2.1",
"requests>=2.32.5",
"psycopg[binary]>=3.2",
"gunicorn>=23.0.0",
"whitenoise>=6.6.0",
"drf-yasg>=1.21.11",
```

3 Verwendete Assets und Bibliotheken

3.1 Assets

Für das Projekt nutzen wir folgende Assets:

- **Earth**: eine 3D Repräsentation der Erde, um die herum die Flugzeuge fliegen sollen, Quelle siehe 4.2 Unity Packages, EarthRendering Free
- **Low Poly Gulfstream** [4]: eine 3D Repräsentation eines Gulfstream Flugzeugs, welches den privaten Charakter der von uns getrackten Flugzeuge vermitteln soll. Das Asset soll für jedes getrackte Flugzeug auf dem Globus gespawnt werden und die Flugroute verfolgen und veranschaulichen.
- **Low Poly Cloud Pack** [5]: wird verwendet um zufällige Wolken Modelle auf den Flugbahnen der Flugzeuge zu spawnen, um deren Flugroute nachverfolgen zu können.
- **FlightComputerImage** [6]: Abbildung eines Flugcomputerbildschirms, wird genutzt um Flugdaten eines Flugzeuges nach Auswahl des Flugzeuges darzustellen.
- **Milkyway Skybox** [7]: Bild eines Milchstraßen Panoramas, das, auf eine Kugel projiziert, als Skybox der Szene genutzt wird.

3.2 Unity Packages

Folgende Packages haben wir über den Unity-Package-Manager in unser Projekt integriert:

- **TextMeshPro**: Hinzufügen von UI-Schrift Elementen.
- **EarthRendering Free** : stellt das 3D Asset des Erdballs.
- **glTFast** (optional): Import von glTF-Modellen.

3.3 Sonstiges

Folgende weitere Tools und Quellen nutzen wir für das Projekt:

- **Backend-Bibliotheken**: Django, DRF, drf-yasg (Swagger/Redoc), Celery + django-celery-beat/-results, requests (API-Fetch), Whitenoise (Static Files), gunicorn.
- **airplanes.live API** [8]: Unsere Datenquelle, von der wir die Flugdaten der Flugzeuge als Live-ADS-B Feed beziehen.

4 Interaktion

Die Interaktion erfolgt vorerst auf einem Computer mittels Maus und Tastatur. Im Hauptmenü wird per Maus interagiert, wodurch sich mit der linken Maustaste die Buttons des Menüs anwählen lassen. Was die Tracker-Szene angeht gibt es mehrere Möglichkeiten mit dieser zu interagieren:

- **Kamera bewegen**: Mit den WASD-Tasten kann die Kamera auf einer 2D-Ebene (vor, zurück, links, rechts) durch die Szene bewegt werden. Die Leertaste und STRG-Taste fügen dazu Bewegung nach oben und unten hinzu wodurch sich Nutzende im kompletten 3D-Raum bewegen können.
- **Kamera Drehung & Neigung**: Mittels Mausbewegung können Nutzende die Kamera, zusätzlich zur Bewegung, Neigen und Drehen. Die Drehung der Kamera kann in 360° um den Mittelpunkt der Kamera erfolgen. Dafür muss die Maus, von der Oberfläche aus gesehen auf der sie aufliegt, horizontal bewegt werden. Mittels vertikaler Mausbewegung lässt sich die Kamera neigen. Die Neigung ist nur in einem Winkel von max. 90° um die X-Achse der Kamera möglich um zu verhindern, dass sich die Kamera überschlägt.
- **Moduswechsel zwischen Bewegung und UI**: Da es gleichzeitig möglich sein sollte mittels Maus die Kamera zu bedienen sowie mit dem UI zu interagieren, existieren zwei Modi für die Interaktion durch die Maus. Der Standardmodus ist der UI-Modus, bei diesem ist der Mauscursor sichtbar und es kann mit dem UI interagiert werden. Der Bewegungsmodus lässt sich durch Drücken und Halten der rechten Maustaste

aktivieren. Im Bewegungsmodus ist es dann möglich die Kamera wie oben beschrieben zu steuern. Der Mauscursor wird dabei ausgeblendet.

- **Flugzeugauswahl**: Im Bewegungsmodus und UI-Modus wird in der Mitte des Bildschirms ein Fadenkreuz angezeigt. Dieses besteht aus einem orangenen Punkt. Im Bewegungsmodus können mit dem Fadenkreuz auf dem Globus spawnende Flugzeuge angewählt werden. Das ausgewählte Flugzeug wird dann durch eine farbliche Umrandung hervorgehoben. Zusätzlich erscheint ein UI-Element in Form eines Flugcomputerbildschirms in der linken oberen Ecke der Szene. In diesem lassen sich aktuelle Flugdaten wie z.B. Flugzeugkennung, Höhe und Geschwindigkeit des ausgewählten Flugzeugs einsehen. Lässt man die rechte Maustaste los und wechselt so aus dem Bewegungsmodus in den UI-Modus, kann das Computer-UI-Element über einen im Element befindlichen Button geschlossen werden. Alternativ kann im Bewegungsmodus ein anderes Flugzeug ausgewählt werden. Die farbliche Umrandung wird nun für dieses Flugzeug angewendet und dessen Daten werden im Computer-UI-Element angezeigt.

5 Design der Benutzeroberflächen

Zu Beginn des Projektes war für uns klar, dass wir einen LowPoly-Look verfolgen wollten. Leider ließ sich für die Erdkugel kein passendes Asset im Low-Poly-Look finden, bei dem die Skalierung des Assets so möglich gewesen wäre, wie wir es für die Anzeige der Flugzeugrouten benötigten. Wir haben vor allem versucht, die Skalierung der Flugzeuge und Routen zu priorisieren, sodass sie einerseits so weit auseinander dargestellt werden, dass einzelne Routen klar erkennbar sind, und andererseits nah genug beieinander sind, sodass Nutzende sich nicht mit weit zurückzulegenden Strecken mittels Kamerabewegung um ein zu großes Erdasset konfrontiert sehen. Dabei traten markante Eigenschaften der LowPoly-Assets zu sehr hervor, sodass diese hinderlich in der Anzeige der Flugzeuge waren und diese verdeckt.

Deshalb haben wir uns für ein detaillierteres Erdasset entschieden. Da die Erdkugel das markanteste und auffälligste Asset in unserem Projekt ist, stellt dieses natürlich sofort einen Kontrast zu dem anfänglich von uns verfolgten Low-Poly-Stil dar. Nur das Flugzeug-Asset sowie die Wolken sind weiterhin im Low-Poly-Stil gehalten. Das Design ist insofern nicht schlüssig. Durch die Nutzung von High- und Low-Poly-Assets erscheint hier im Stil keine erkennbare Gemeinsamkeit.

Einzig das UI für das Hauptmenü und die Flugdaten hebt sich zwar durch seine zweidimensionale Natur und höheren Detailgrad vom LowPoly-Stil ab, ist allerdings vom Look und der Farbgebung ganzheitlich an das Aussehen eines Flugcomputers angelehnt und in diesem Kontext kohärent, siehe [6]. Die Bedienung des UI wird ausschließlich durch Button-Interaktion ermöglicht, die z.B. für das Starten der Hauptszene, Beenden der App oder schließen von UI Elementen genutzt werden. Weitere Bedienmöglichkeiten waren anfänglich geplant, wie z.B. Slider für das Ändern der Rotationsgeschwindigkeit der Erde auf dem Smartphone. Da aber die Integration von Vuforia verworfen wurde, wie in Abschnitt 1 bereits vorangestellt, entfiel auch die Integration weiterer Bedienelemente.

Grund für das minimalistische und eher unschlüssige Design ist, dass wir vor allem die technische Umsetzbarkeit hervorheben wollten. Da es uns außerdem an Erfahrung im Design fehlt, haben wir uns vorerst für diese provisorische Designstrategie entschieden.

6 Technische Dokumentation

6.1 Backend

Zum Sammeln und Bereitstellen der Positionsdaten der Privatejets haben wir ein Backend mit dem Python Framework `Django` implementiert. Um die Positionsdaten zu sammeln wir die

öffentlich und kostenlos zugängliche API von Airplanes.live abgefragt. Wie in Abb. 1 zu sehen wurde zum Speichern der abgefragten Daten wurde ein simples Datenbankmodell entwickelt.

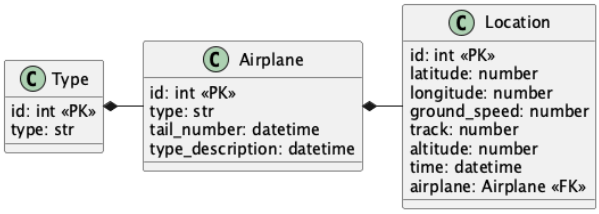

Abb. 1: Relationales Daten-Modell

Das Python package `Celery` wird verwendet um Tasks asynchron auszuführen. Wir verwenden den Celery Task Scheduler `Beat` um in regelmäßigen Zeitintervallen die Abfragen durchzuführen. Wie in Listing 1 zu sehen, fragen wir die API alle 5 Minuten ab.

```py
app.conf.beat_schedule = {
    "fetch-airplane-positions": {
        "task": "api.tasks.fetch_data",
        "schedule": 300.0,
    },
}
```

Listing 1: Celery Beat Schedule

Zum Administrieren der Daten wurde das Admin-Interface von Django konfiguriert. Hier können alle gesammelten `Airplanes` und `Locations` eingesehen und bearbeitet werden.

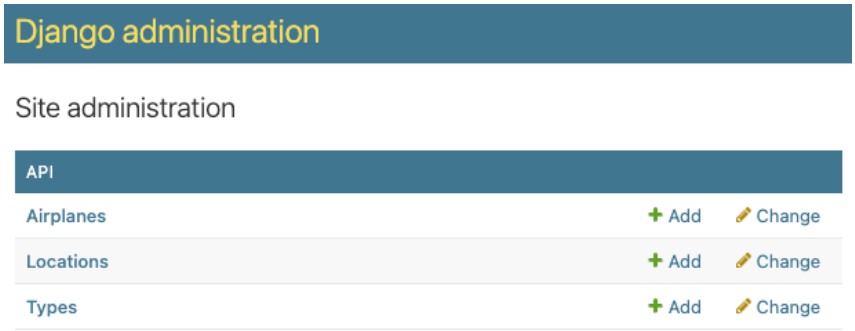

Abb. 2: Django Admin Interface

Jedes `Airplane` hat eine one to many relationship mit `Location`. Wie in Abb. 3 zu sehen, kann jedes `Airplane` und seine zugehörigen `Location`´s eingesehen und bearbeitet werden.

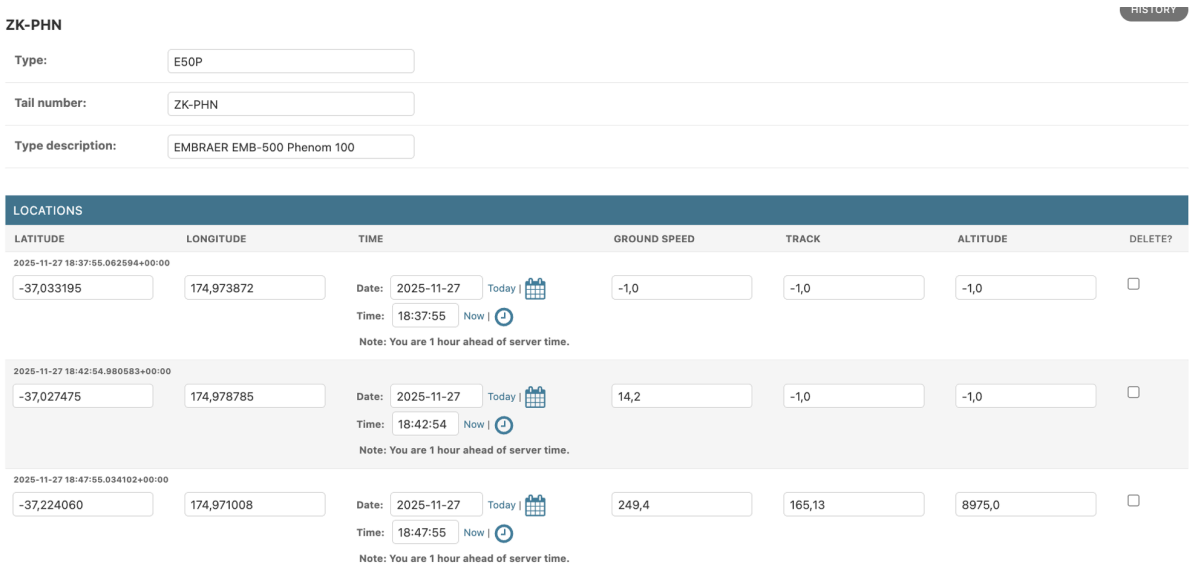

Abb. 3: Django Admin Interface

Das vollständige API-Schema kann unter https://flights.davidkirchner.de/swagger eingesehen werden.

6.2 Unity Anwendung

Die Struktur der Unity Anwendung besteht aus 3 Hauptkomponenten. Den `AirplaneMapper`, dem `PlayerController` und dem `ClickPlaneManager`.

6.2.1 AirplaneMapper Airplane und CoordinateFetcher

Im `AirplaneMapper` liegen alle Verantwortlichkeiten zum Anzeigen der Flugzeuge auf dem Erdball. Er kontrolliert lifetimes für die `Airplane` und `Indicator` Objekte. Zusätzlich nutzt dieser den `CoordinateFetcher` zum Abrufen der Daten von Backend und zum instanziieren der `Airplane` Objekte.

Jedes `Airplane` ist für das abspielen seiner Positionsdaten und das spawnen und zerstören von den zugehörigen Positionsmarker verantwortlich.

6.2.2 PlayerController

Der `PlayerController` beinhaltet alle Funktionalitäten die zum bewegen der Kamera benötigt werden. Es werden mehrere Event-Listener verwendet um die Bewegung der Maus und Tastendrücke abzufangen und zu behandeln. Der `PlayerController` hat durch den Inspector einstellbare Attribute welche die Mausempfindlichkeit, die Winkelgrenzen und die Bewegungsgeschwindigkeit steuern.

6.2.3 ClickPlaneManager

Der `ClickPlaneManager` ist für die Spielerinteraktionen mit den Flugzeugen verantwortlich. Durch einen Linksklick wird ein Strahl aus der Mitte der Kamera ausgesendet und überprüft ob das getroffene Objekt den Tag `airplane` besitzt. Falls dies der Fall ist, über das `Transform` des Objekts das zugehörige `Airplane` abgerufen. Nun wird eine Outline um das Mesh des Airplane Objekts gelegt und die zugehörigen Flugdaten (tailnumber, type, height, speed, heading, latitude, longitude) eingelesen. Dann wird das erstellte Flugdaten Userinterface eingeblendet und die Daten eingefügt.

6.3 Unity-Datenfluss

1. `AirplaneMapper` startet im AR-Scene-Setup und lädt per `CoordinateFetcher` wahlweise populäre Flugzeuge oder eine Liste aus dem Inspector.
2. `CoordinateFetcher` konsumiert die Django-API und liefert JSON-Responses für Tail-Number oder Popular-Endpunkt.
3. `ListCoordinateMapper` wandelt jede Location in Weltkoordinaten auf dem Globus um (Lat/Lon → 3D-Richtung, Altitude -> radialer Offset, sortiert nach Timestamp).
4. Für jedes Flugzeug wird ein Prefab instanziiert (`Airplane`-Komponente). `Airplane.Initialize` startet eine Coroutine, die die Route mit Zeitkompression abspielt, Marker in Intervallen setzt und Rotation an die Globusnormalen anpasst.
5. Marker-Management: begrenzte Anzahl/Lebensdauer via Culling und FIFO-Löschung, damit die Szene performant bleibt.

7 Problemlösung

Im Rahmen der Implementierung ergaben sich verschiedene technische Herausforderungen bezüglich der API-Nutzung, der Szenengestaltung, der Performance und der Datenqualität, die durch gezielte Maßnahmen adressiert wurden.

7.1 API-Ratenbegrenzung und Datenbereinigung

Ein kritischer Aspekt war der Umgang mit der airplanes.live API, die nicht zu aggressiv abgefragt werden darf, um Sperren zu vermeiden. Als Lösung wurde eine künstliche Pause (`sleep`) von einer Sekunde pro abgefragtem Flugzeugtyp sowie ein moderates gesamt-abfrage-Intervall von fünf Minuten eingeführt. Zusätzlich implementierten wir einen Validierungscheck für die Länge der „Tail Number“, um fehlerhafte oder unvollständige Datensätze frühzeitig herauszufiltern. Ein weiteres Datenproblem betraf Höhenangaben, die lediglich als „ground“ zurückgegeben wurden; diese werden nun systemintern in den numerischen Wert -1 umgewandelt, um sie eindeutig als nicht-fliegende Objekte zu kennzeichnen.

7.2 Marker-Performance

Die Darstellung einer großen Anzahl von Markern führte initial zu spürbaren Einbrüchen der Bildwiederholrate (Framerate). Um die Performance zu stabilisieren, wurde eine konfigurierbare Obergrenze für die Marker-Anzahl sowie eine definierte Lebensdauer eingeführt. Überschüssige Marker werden nun nach dem FIFO-Prinzip (First-In, First-Out) entfernt, sodass stets nur die aktuellsten relevanten Daten visualisiert werden, ohne die Render-Leistung zu überlasten.

7.3 Zeitseriendaten und Visualisierung

Bei der Verarbeitung der zeitbasierten Daten zeigte sich, dass eine Darstellung entsprechend der tatsächlichen Beobachtungszeiträume zu einer extremen zeitlichen Streckung führte. Um die Analyse zu vereinfachen und die visuelle Dichte zu erhöhen, haben wir uns dazu entschlossen, die zeitliche Komponente zu normalisieren und alle in einem Zyklus gesammelten Daten so anzuzeigen, als wären sie zum selben Zeitpunkt aufgetreten.

7.4 Abdeckung der ADS-B Daten

Die Verfügbarkeit der ADS-B Daten ist geografisch stark inkonsistent. Während Nordamerika eine sehr hohe Abdeckung aufweist, ist die Datendichte in Europa bereits geringer und nimmt im globalen Süden sowie in Asien noch weiter ab. Dies lässt sich darauf zurückführen, dass die verwendete ADS-B Datenbank primär durch die Community gespeist wird („Crowdsourcing“). In Nordamerika ist die Dichte an Hobbyisten, die private ADS-B Empfänger betreiben und Daten einspeisen, schlicht am höchsten, was die geografische Diskrepanz erklären könnte.

7.5 Verfügbarkeit der ADS-B Daten

ADS-B (Automatic Dependent Surveillance – Broadcast) ist ein Verfahren nach dem vor allem Flugzeuge ungefähr jede Sekunde verschiedene Informationen über sich selbst, wie z.B. Flugnummer, Geschwindigkeit, Flughöhe sowie dazugehörigen Zeitstempel, ungerichtet mittels elektromagnetischem Rundfunk aussenden. Dieser ungerichtete Rundfunk kann von allen aufgefangen und gehört werden, die einen entsprechenden Empfänger besitzen. Neben offiziellen Stellen wie z.B. der amerikanischen FAA (Federal Aviation Administration) existieren mehrere Crowdsourcing Communities wie z.B. ADS-B Exchange [9], die die ADS broadcasts empfangen und speichern. Diese communities stellen meist eine API zur Verfügung über die die von ihnen gespeicherten Daten abgefragt werden können, als Live Daten oder aber auch historisch Daten. Manche verlangen jedoch eine Gebühr für die Nutzung entweder der gesamten API, oder nur der historischen Daten, siehe [9]. Andere wiederum gewähren erst nach einer Bewerbung in der der Nutzungsgrund erläutert wird und nach eingehender Prüfung dieser einen Zugang zur API, siehe [10]. Da wir schnell mit der Entwicklung beginnen wollten, ohne auf eine Genehmigung zu warten oder Kosten zu tragen, haben wir uns letztendlich für den Anbieter Airplanes.LIVE [8] entschieden, welcher eine kostenlose API zur Verfügung stellt.

7.6 Tracking von Flugzeugen prominenter Personen

Es existieren bereits einige Projekte, die das Flugverhalten von prominenten Personen tracken, wie z.B. celebrityflight.com [11] oder celebrityprivatejettracker.com [12]. Sie nutzen unter anderem die oben genannten APIs der crowdsourcing Communities [13] um die öffentlich zugänglichen Registrationsnummern der Privatflugzeuge prominenter Personen zu tracken. In Kombination damit, dass z.B. die FAA Details über Flugzeugregistrationen wie z.B. Namen und Adresse des Besitzers öffentlich macht, lassen sich so die Flugdaten und die Personen assoziieren. Leider war es uns jedoch nicht möglich, das zu verifizieren. Da wir aufgrund der Kosten und Antragsbarrieren keinen Zugriff auf historische Daten hatten, mussten wir selbst Daten über eine live API sammeln. In diesen Daten konnten wir die Registrationsnummern verschiedener prominenter Personen leider nicht ausmachen. Eine Vermutung unsererseits ist, dass dies mit einer Maßnahme der FAA zusammen hängt, die es Besitzern privater Flugzeuge erlaubt, ihre Registrations Details aus der öffentlichen Datenbank zu verbergen [14]. Zwar sind die crowdsourcten Projekte wie in Abschnitt 7.5 bereits erwähnt unabhängig von der FAA, allerdings lässt sich nicht ausschließen, dass diese trotzdem einen Einfluss auf das Sendeverhalten bestimmter Flugzeuge nehmen kann. Das Problem, dass wir diese spezifischen Flugzeuge nicht tracken konnten brachte uns dazu, dass wir, wie in Abschnitt 1 erwähnt nunmehr nicht mehr die Flugzeuge bestimmter Personen tracken, sondern bestimmte Flugzeugtypen, die am ehesten mit Privatflügen assoziiert werden. Beispielhaft hier z.B. die Gulfstream Reihe [15].

8 Fazit

Im Verlaufe des Projektes mussten wir, unsere Zielsetzung immer wieder verwerfen und neu denken. Das letztendliche Ziel, Live-Flugrouten privater Flugzeuge auf einem AR-Globus sichtbar zu machen, wurde jedoch erfolgreich umgesetzt.

Die Pipeline aus Celery-Worker, Django-API und Unity-Client funktioniert end-to-end: Daten werden regelmäßig abgerufen, gespeichert und in der App als animierte Trajektorien mit Markern dargestellt.

Mögliche nächste Schritte:

- On demand Streaming statt periodischer Pulls (z. B. WebSockets von Backend zu Unity).
- Besseres Flugzeugmodell/Material und UI-Overlays für Geschwindigkeit/Höhe je Marker.
- Offline-Cache und Retry-Strategien bei instabilem Netz.

- Mehr Interaktion: Auswahl konkreter Tail-Numbers in der App, Filter nach Typ, Bilder von Flugzeugtypen.
- Vuforia für 3D Visualisierung auf dem Smartphone.

Literatur

[1] Statistisches Bundesamt, „Anzahl der einsteigenden Flugpassagiere bei Inlandsflügen in Deutschland in den Jahren 2017 bis 2024 [Graph]“. [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/1467900/umfrage/passagieraufkommen-bei-inlandsfluegen-in-deutschland-2023/

[2] CE Delft, „Anzahl der Privatflüge inklusive Geschäftsflüge* in Deutschland von 2020 bis 2022 [Graph]“. [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/1389719/umfrage/anzahl-privatfluege-und-geschaeftsfluege-in-deutschland/

[3] www.flightradar24.com, „flightradar24“. [Online]. Verfügbar unter: https://www.flightradar24.com/

[4] Mauro3D, „Low-Poly-Gulfstream-Aircraft“. Zugegriffen: 14. Januar 2026. [Online]. Verfügbar unter: https://sketchfab.com/3d-models/low-poly-gulfstream-aircraft-af86c5f4a3054cd8a30d1ad213f99e57

[5] PolyOne-Studio, „Stylized Clouds Pack - Vol 07 free low-poly 3d model“. Zugegriffen: 14. Januar 2026. [Online]. Verfügbar unter: https://www.cgtrader.com/items/5996888/download-page

[6] vonKleist, „Tutorial (Deutsch) - XPlane 11 Der Flugcomputer erklärt (FMC/CMD/CDU/FMS) [TEIL 2] - Thumbnail Image“. Zugegriffen: 14. Januar 2026. [Online]. Verfügbar unter: https://i.ytimg.com/vi/za2r_Zj4hns/maxresdefault.jpg

[7] MozillaHubs, „Sky Pano - Milkyway“. Zugegriffen: 14. Januar 2026. [Online]. Verfügbar unter: https://sketchfab.com/3d-models/sky-pano-milkyway-0016725c047a4ea18cd0b5e5ef2fe441

[8] Airplanes.live, „Airplanes.live REST API Documentation“. Zugegriffen: 14. Januar 2026. [Online]. Verfügbar unter: https://airplanes.live/api-guide/

[9] https://globe.adsbexchange.com/, „ADS-B Exchange“. [Online]. Verfügbar unter: https://globe.adsbexchange.com/

[10] opensky-network.org, „OpenSky Network“. [Online]. Verfügbar unter: https://opensky-network.org/

[11] celebrityflight.com, „Celebrity Flight“. [Online]. Verfügbar unter: https://celebrityflight.com/

[12] celebrityprivatejettracker.com, „Celebrity Privatejet Tracker“. [Online]. Verfügbar unter: https://celebrityprivatejettracker.com/

[13] https://www.reddit.com/user/Sandlaa/, „I created a page to track flights and emissions from celebrities' private jets“. [Online]. Verfügbar unter: https://www.reddit.com/r/webdev/comments/16wc00o/i_created_a_page_to_track_flights_and_emissions/

[14] J. Joseph, „Good News for Billionaires: FAA to Let Them Block Tracking of Their Private Jets“. [Online]. Verfügbar unter: https://www.pcmag.com/news/good-news-for-billionaires-faa-to-let-them-block-tracking-of-their-private?test_uuid=04IpBmWGZleS0I0J3epvMrC&test_variant=B

[15] G. A. Corporation, „Gulfstream“. [Online]. Verfügbar unter: https://www.gulfstream.com/en/