NFL Intelligence Hub - Development Log

# Project Overview

The NFL Intelligence Hub is a full-stack application designed to collect, process, analyze, and visualize NFL player statistics. It integrates data engineering, backend API development, and frontend design. The tech stack includes React, Tailwind CSS, FastApi, PostgreSQL, Pandas, PySpark, and optional Airflow for ETL scheduling.

# Planning Phase

- Defined project scope and objectives  
- Selected tech stack suitable for junior to intermediate college-level developers  
- Designed database schema: players, teams, rushing\_stats, etc.  
- Established Git workflow and project folder structure

# Data Engineering

- Web scraping scripts created using `requests` and `BeautifulSoup` for stats like rushing, kicking, etc.  
- Extracted and transformed data into structured pandas DataFrames  
- Cleaned and validated data (e.g., removing header rows, missing values)  
- Inserted cleaned data into PostgreSQL using SQLAlchemy  
- Resolved data mismatches, like inconsistent team abbreviations

# Database Design and Issues

- Created tables with proper primary and foreign key constraints  
- Resolved foreign key issues due to mismatched column names or missing references  
- Used `ON CONFLICT` SQL clauses to handle duplicates in `players` and `teams`  
- Inserted base team and player data before adding statistics to maintain referential integrity  
- Example bug: NULL `player\_id` in `rushing\_stats` traced back to FK mismatch

# API Development (Planned)

- Decided to use FastAPI and Uvicorn for serving data  
- Planned to start API with GET requests only  
- Will add Pydantic models if response validation becomes necessary  
- Deferred creation of SQL indexes until after basic endpoint testing

# Bug Fixes and Technical Issues

- Fixed team name abbreviation mismatches using SQL update logic  
- Identified and corrected column naming conflicts causing FK reference issues  
- Addressed insert errors due to incorrect or incomplete data values (e.g., 2TM issue)  
- Ensured proper data copying and Pandas view/assignment warnings were avoided

# Next Steps

- Build out API endpoints and validate with frontend requests  
- Construct frontend UI with team/player filters and stat charts  
- Optional: Automate ETL using Airflow or scheduled cron jobs  
- Finalize README with architecture diagram and setup instructions

## Update - 2025-07-02

- Started API development with FastAPI, exposing endpoints for player stats (passing, rushing, receiving, kicking) by season and team.

- Utilized FastAPI's Query and Path functions to safely validate input parameters (e.g., season year range, player name).

- Handled SQL injection risks by using parameterized queries and enforced input constraints with FastAPI.

- Switched from path to query parameters for more flexible API usage (e.g., /players?season\_year=2023).

- Debugged and resolved issue where only one SQL query was executing under `engine.connect()`. Switched to `engine.begin()` to ensure both queries committed properly.

- Fixed foreign key update bugs by splitting update operations and committing them separately.

- Introduced functionality to remove PII (player and team names) from all stat tables via raw SQL drop column commands.

- Documented inconsistent team abbreviations and resolved via a helper function that standardizes team keys before inserting into DB.

- Addressed uvicorn issues by correcting CLI syntax and ensuring FastAPI app module paths were correct (used `backend.main:app` syntax).

- Integrated project with GitHub and resolved common merge conflicts during collaborative development.

- Created separate insertion scripts per stat category (rushing, receiving, etc.) and applied ON CONFLICT clause for safe inserts.