```
print("♠ CLEAN FIXED BACKEND RUNNING ♠")

import os
import re
import requests
from datetime import datetime, timedelta, date
from typing import Optional, Dict, Any, List
from fastapi.staticfiles import StaticFiles
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import FileResponse
from pydantic import BaseModel
from openai import OpenAI

# =========================
# ENV KEYS
# =========================
FOOTBALL_DATA_API_KEY = os.getenv("FOOTBALL_DATA_API_KEY")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
ANTHROPIC_API_KEY = os.getenv("ANTHROPIC_API_KEY")  # Your Claude API key
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY", "AIzaSyC62FAT55vqRGVDAxgV9f-3rUY2eXn
API_FOOTBALL_KEY = os.getenv("API_FOOTBALL_KEY", "39d7d3d558fa860cae92c38fca89246

if not FOOTBALL_DATA_API_KEY:
    raise RuntimeError("FOOTBALL_DATA_API_KEY is NOT set")
if not OPENAI_API_KEY:
    raise RuntimeError("OPENAI_API_KEY is NOT set")

openai_client = OpenAI(api_key=OPENAI_API_KEY)

# Claude client (for web search AI chat)
try:
    import anthropic
    if ANTHROPIC_API_KEY:
        anthropic_client = anthropic.Anthropic(api_key=ANTHROPIC_API_KEY)
    else:
        anthropic_client = None
        print("Warning: ANTHROPIC_API_KEY not set - Claude AI chat disabled")
except ImportError:
    anthropic_client = None
    print("Warning: anthropic package not installed - Claude AI chat disabled")

# Google Gemini client (FREE web search!)
try:
    import google.generativeai as genai
    if GOOGLE_API_KEY:
        genai.configure(api_key=GOOGLE_API_KEY)
        gemini_available = True
    else:
        gemini_available = False
        print("Warning: GOOGLE_API_KEY not set - Gemini AI chat disabled")
except ImportError:
    gemini_available = False
    print("Warning: google-generativeai package not installed - Gemini AI chat di

# Football-Data.org (keeping as backup)
FOOTBALL_API_BASE = "https://api.football-data.org/v4"
HEADERS = {"X-Auth-Token": FOOTBALL_DATA_API_KEY}

# API-Football (new primary source)
API_FOOTBALL_BASE = "https://v3.football.api-sports.io"
API_FOOTBALL_HEADERS = {
    "x-rapidapi-host": "v3.football.api-sports.io",
    "x-rapidapi-key": API_FOOTBALL_KEY
}

# League ID mapping (Football-Data codes to API-Football IDs)
LEAGUE_MAPPING = {
    "PL": 39,      # Premier League
    "PD": 140,     # La Liga
    "SA": 135,     # Serie A
    "BL1": 78,     # Bundesliga
    "FL1": 61,     # Ligue 1
    "CL": 2,       # Champions League
    "ELC": 40,     # Championship
    "PPL": 94,     # Primeira Liga
    "DED": 88,     # Eredivisie
    "BSA": 71,     # Serie A Brasil
    "EC": 848,     # Euro Championship
    "WC": 1,       # World Cup
}


# ==========================
# APP
# ==========================
app = FastAPI(title="Global Football Intelligence")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# =========================
# FRONTEND SERVING
# =========================
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
FRONTEND_DIR = os.path.join(BASE_DIR, "..", "frontend")

@app.get("/")
def serve_index():
    return FileResponse(os.path.join(FRONTEND_DIR, "index.html"))

app.mount("/static", StaticFiles(directory=FRONTEND_DIR), name="static")


# =========================
# MODELS
# =========================
class AnalyzeRequest(BaseModel):
    query: str
    competition_code: Optional[str] = None


# =========================
# SIMPLE CACHES
# =========================
TEAM_CACHE: Dict[str, Dict[str, Any]] = {}
CACHE_TS: Dict[str, float] = {}
CACHE_TTL_SECONDS = 60 * 30  # 30 mins


def _normalize_name(s: str) -> str:
    s = s.lower().strip()
    s = re.sub(r"\s+", " ", s)
    return s


def football_get(url: str) -> Dict[str, Any]:
    r = requests.get(url, headers=HEADERS, timeout=25)
    if r.status_code != 200:
        raise HTTPException(status_code=500, detail=f"Football API error {r.statu
    return r.json()
```

```
def _ensure_team_cache(competition_code: str):
    now = datetime.utcnow().timestamp()
    if competition_code in TEAM_CACHE and (now - CACHE_TS.get(competition_code, 0
        return

    url = f"{FOOTBALL_API_BASE}/competitions/{competition_code}/teams"
    data = football_get(url)

    by_name = {}
    by_id = {}
    for t in data.get("teams", []):
        tid = t.get("id")
        name = t.get("name", "")
        short = t.get("shortName", "")
        tla = t.get("tla", "")
        crest = t.get("crest")

        obj = {
            "id": tid,
            "name": name,
            "shortName": short,
            "tla": tla,
            "crest": crest,
        }
        if tid:
            by_id[str(tid)] = obj

        # store a few keys for matching
        for key in {name, short, tla}:
            k = _normalize_name(key)
            if k:
                by_name[k] = obj

    TEAM_CACHE[competition_code] = {"by_name": by_name, "by_id": by_id}
    CACHE_TS[competition_code] = now


def find_team_in_competition(team_name: str, competition_code: str) -> Optional[D
    _ensure_team_cache(competition_code)
    cache = TEAM_CACHE.get(competition_code, {})
    by_name = cache.get("by_name", {})

    n = _normalize_name(team_name)

    # direct hit
    if n in by_name:
        return by_name[n]
    # soft contains match
    for k, v in by_name.items():
        if n and (n in k or k in n):
            return v

    return None



def get_standings_total(competition_code: str) -> List[Dict[str, Any]]:
    url = f"{FOOTBALL_API_BASE}/competitions/{competition_code}/standings"
    data = football_get(url)

    standings = data.get("standings", [])
    total = next((s for s in standings if s.get("type") == "TOTAL"), None)
    if not total:
        return []

    out = []
    for row in total.get("table", []):
        team = row.get("team", {})
        out.append({
            "position": row.get("position"),
            "team": team.get("name"),
            "team_id": team.get("id"),
            "crest": team.get("crest"),
            "played": row.get("playedGames"),
            "won": row.get("won"),
            "draw": row.get("draw"),
            "lost": row.get("lost"),
            "gf": row.get("goalsFor"),
            "ga": row.get("goalsAgainst"),
            "gd": row.get("goalDifference"),
            "points": row.get("points"),
        })
    return out



def get_team_table_row(team_id: int, competition_code: str) -> Optional[Dict[str,
    table = get_standings_total(competition_code)
    for r in table:
        if r.get("team_id") == team_id:
            return r
    return None



def get_last_matches_form(team_id: int, limit: int = 5) -> Dict[str, Any]:
```

```
# last ~90 days, finished only
date_to = datetime.utcnow().date()
date_from = date_to - timedelta(days=120)

url = (
    f"{FOOTBALL_API_BASE}/teams/{team_id}/matches"
    f"?status=FINISHED"
    f"&dateFrom={date_from.isoformat()}"
    f"&dateTo={date_to.isoformat()}"
    f"&limit={limit}"
)
data = football_get(url)
matches = data.get("matches", [])[:limit]

form = []
goals_for = 0
goals_against = 0

for m in matches:
    score = m.get("score", {}).get("fullTime", {})
    home = m.get("homeTeam", {})
    away = m.get("awayTeam", {})
    home_id = home.get("id")
    away_id = away.get("id")

    hg = score.get("home", 0) or 0
    ag = score.get("away", 0) or 0

    is_home = (home_id == team_id)
    gf = hg if is_home else ag
    ga = ag if is_home else hg
    goals_for += gf
    goals_against += ga

    if gf > ga:
        res = "W"
    elif gf == ga:
        res = "D"
    else:
        res = "L"

    form.append({
        "result": res,
        "utcDate": m.get("utcDate"),
        "opponent": away.get("name") if is_home else home.get("name"),
        "score": f"{hg}-{ag}",
        "homeAway": "H" if is_home else "A"
        })

    return {
        "form": form,
        "gf_last5": goals_for,
        "ga_last5": goals_against
    }

def get_head_to_head(home_id: int, away_id: int, limit: int = 10):
    url = (
        f"{FOOTBALL_API_BASE}/teams/{home_id}/matches"
        f"?status=FINISHED&limit={limit}"
    )
    data = football_get(url)

    out = []
    for m in data.get("matches", []):
        if m["homeTeam"]["id"] == away_id or m["awayTeam"]["id"] == away_id:
            score = m["score"]["fullTime"]
            out.append({
                "date": m["utcDate"],
                "home": m["homeTeam"]["name"],
                "away": m["awayTeam"]["name"],
                "score": f'{score["home"]}-{score["away"]}'
            })

    return out

# ========================
# HEALTH
# ========================
@app.get("/health")
def health():
    return {
        "status": "ok",
        "openai_key_loaded": bool(OPENAI_API_KEY),
        "football_key_loaded": bool(FOOTBALL_DATA_API_KEY),
    }

# ==============================
# LEAGUES (STATIC LIST)
# ==============================

LEAGUES = [
    # England
    {"code": "PL",  "name": "Premier League", "country": "England", "flag": "GB",
    {"code": "ELC", "name": "Championship",   "country": "England", "flag": "GB",
    # Spain
    {"code": "PD",  "name": "La Liga", "country": "Spain", "flag": "ES", "crest":

    # Germany
    {"code": "BL1", "name": "Bundesliga", "country": "Germany", "flag": "DE", "cr

    # Italy
    {"code": "SA",  "name": "Serie A", "country": "Italy", "flag": "IT", "crest":

    # France
    {"code": "FL1", "name": "Ligue 1", "country": "France", "flag": "FR", "crest"

    # Portugal
    {"code": "PPL", "name": "Primeira Liga", "country": "Portugal", "flag": "PT",

    # Netherlands (Holland)
    {"code": "DED", "name": "Eredivisie", "country": "Netherlands", "flag": "NL",
]


@app.get("/leagues")
def get_leagues():
    """
    Returns a static list of supported leagues
    """
    return LEAGUES


# =========================
# STANDINGS (LIVE TABLE)
# =========================
@app.get("/standings/{competition_code}")
def standings(competition_code: str):
    # returns array of table rows
    return get_standings_total(competition_code)



# =========================
# FIXTURES (NEXT 7 DAYS, GROUPED)
# =========================
@app.get("/fixtures/{competition_code}")
def fixtures(competition_code: str):
    today = datetime.utcnow().date()
    end = today + timedelta(days=7)

    url = (
        f"{FOOTBALL_API_BASE}/competitions/{competition_code}/matches"
        f"?dateFrom={today.isoformat()}&dateTo={end.isoformat()}"
    )
    data = football_get(url)
    matches = data.get("matches", [])

    grouped: Dict[str, List[Dict[str, Any]]] = {}

    for m in matches:
        match_date = (m.get("utcDate") or "")[:10]
        if not match_date:
            continue

        home = m.get("homeTeam", {})
        away = m.get("awayTeam", {})

        grouped.setdefault(match_date, []).append({
            "date": m.get("utcDate"),
            "home": home.get("name"),
            "away": away.get("name"),
            "home_crest": home.get("crest"),
            "away_crest": away.get("crest"),
        })

    return [{"date": d, "matches": grouped[d]} for d in sorted(grouped.keys())]


# ========================
# HEAD TO HEAD (LAST 10)
# ========================
class H2HRequest(BaseModel):
    query: str
    competition_code: Optional[str] = None

@app.post("/h2h")
def h2h(req: H2HRequest):
    """
    Get head-to-head record between two teams
    Accepts format: "Team A vs Team B"
    """
    competition_code = (req.competition_code or "PL").strip()

    # Parse "Team A vs Team B"
    if "vs" not in req.query.lower():
        raise HTTPException(status_code=400, detail="Use format: Team A vs Team B

    parts = re.split(r"\s+vs\s+", req.query, flags=re.IGNORECASE)
```

```
if len(parts) != 2:
    raise HTTPException(status_code=400, detail="Use format: Team A vs Team B

home_name = parts[0].strip()
away_name = parts[1].strip()

home_team = find_team_in_competition(home_name, competition_code)
away_team = find_team_in_competition(away_name, competition_code)

if not home_team or not away_team:
    raise HTTPException(status_code=404, detail="Teams not found in this comp

matches = get_head_to_head(home_team["id"], away_team["id"])

# Calculate stats
home_wins = 0
away_wins = 0
draws = 0

for m in matches:
    score_parts = m["score"].split("-")
    if len(score_parts) == 2:
        home_score = int(score_parts[0])
        away_score = int(score_parts[1])

        # Check who was home in that match
        was_home_at_home = m["home"] == home_team["name"]

        if was_home_at_home:
            if home_score > away_score:
                home_wins += 1
            elif home_score < away_score:
                away_wins += 1
            else:
                draws += 1
        else:
            if home_score > away_score:
                away_wins += 1
            elif home_score < away_score:
                home_wins += 1
            else:
                draws += 1

return {
    "home": home_team["name"],
    "away": away_team["name"],
    "home_crest": home_team.get("crest"),
        "away_crest": away_team.get("crest"),
        "matches": matches,
        "stats": {
            "home_wins": home_wins,
            "away_wins": away_wins,
            "draws": draws,
            "total": len(matches)
        }
    }

# ========================
# LIVE MATCHES (IN-PLAY)
# ========================
@app.get("/live/{competition_code}")
def live_matches(competition_code: str):
    url = (
        f"{FOOTBALL_API_BASE}/competitions/{competition_code}/matches"
        f"?status=LIVE"
    )

    data = football_get(url)
    matches = data.get("matches", [])

    out = []

    for m in matches:
        home = m.get("homeTeam", {})
        away = m.get("awayTeam", {})
        score = m.get("score", {}).get("fullTime", {})

        out.append({
            "minute": m.get("minute") or 0,
            "home": home.get("name"),
            "away": away.get("name"),
            "home_crest": home.get("crest"),
            "away_crest": away.get("crest"),
            "home_goals": score.get("home", 0),
            "away_goals": score.get("away", 0),
            "status": m.get("status"),
            "match_id": m.get("id")
        })

    return out


# ========================
# MATCH DETAILS (Goals, Stats, Lineups)
```

```python
# ========================
@app.get("/match/{match_id}")
def match_details(match_id: int):
    """
    Get detailed match information including goals, cards, and stats
    """
    url = f"{FOOTBALL_API_BASE}/matches/{match_id}"
    data = football_get(url)

    match = data
    home_team = match.get("homeTeam", {})
    away_team = match.get("awayTeam", {})
    score = match.get("score", {})

    # Extract goals
    goals = []
    for goal in match.get("goals", []):
        scorer = goal.get("scorer", {})
        assist = goal.get("assist", {})

        goals.append({
            "minute": goal.get("minute"),
            "team": goal.get("team", {}).get("name"),
            "scorer": scorer.get("name"),
            "assist": assist.get("name") if assist else None,
            "type": goal.get("type")  # PENALTY, OWN_GOAL, etc.
        })

    # Extract bookings (cards)
    bookings = []
    for booking in match.get("bookings", []):
        player = booking.get("player", {})

        bookings.append({
            "minute": booking.get("minute"),
            "team": booking.get("team", {}).get("name"),
            "player": player.get("name"),
            "card": booking.get("card")  # YELLOW_CARD, RED_CARD
        })

    # Extract referee
    referees = match.get("referees", [])
    referee_name = referees[0].get("name") if referees else "Unknown"

    return {
        "match_id": match.get("id"),
        "status": match.get("status"),
        "utc_date": match.get("utcDate"),
        "venue": match.get("venue"),
        "home": {
            "name": home_team.get("name"),
            "crest": home_team.get("crest"),
            "coach": home_team.get("coach", {}).get("name")
        },
        "away": {
            "name": away_team.get("name"),
            "crest": away_team.get("crest"),
            "coach": away_team.get("coach", {}).get("name")
        },
        "score": {
            "full_time": score.get("fullTime", {}),
            "half_time": score.get("halfTime", {}),
            "winner": score.get("winner")  # HOME_TEAM, AWAY_TEAM, DRAW
        },
        "goals": sorted(goals, key=lambda x: x["minute"] or 0),
        "bookings": sorted(bookings, key=lambda x: x["minute"] or 0),
        "referee": referee_name,
        "attendance": match.get("attendance")
    }


# =========================
# RECENT RESULTS (FINISHED MATCHES)
# =========================
@app.get("/results/{competition_code}")
def recent_results(competition_code: str):
    """
    Get finished matches from the last 7 days
    """
    today = datetime.utcnow().date()
    start = today - timedelta(days=7)

    url = (
        f"{FOOTBALL_API_BASE}/competitions/{competition_code}/matches"
        f"?dateFrom={start.isoformat()}&dateTo={today.isoformat()}&status=FINISHE
    )

    data = football_get(url)
    matches = data.get("matches", [])

    grouped: Dict[str, List[Dict[str, Any]]] = {}

    for m in matches:
        match_date = (m.get("utcDate") or "")[:10]
        if not match_date:
            continue

        home = m.get("homeTeam", {})
        away = m.get("awayTeam", {})
        score = m.get("score", {}).get("fullTime", {})

        grouped.setdefault(match_date, []).append({
            "date": m.get("utcDate"),
            "match_id": m.get("id"),
            "home": home.get("name"),
            "away": away.get("name"),
            "home_crest": home.get("crest"),
            "away_crest": away.get("crest"),
            "home_score": score.get("home", 0),
            "away_score": score.get("away", 0),
        })

    # Sort by date descending (most recent first)
    return [{"date": d, "matches": grouped[d]} for d in sorted(grouped.keys(), re

# =========================
# AI ANALYSIS (REAL DATA INJECTED)
# =========================
@app.post("/analyze")
def analyze(req: AnalyzeRequest):
    # must know competition to pull correct table + teams
    competition_code = (req.competition_code or "PL").strip()

    # parse "Team A vs Team B"
    if "vs" not in req.query.lower():
        raise HTTPException(status_code=400, detail="Use format: Team A vs Team B

    parts = re.split(r"\s+vs\s+", req.query, flags=re.IGNORECASE)
    if len(parts) != 2:
        raise HTTPException(status_code=400, detail="Use format: Team A vs Team B

    home_name = parts[0].strip()
    away_name = parts[1].strip()

    home_team = find_team_in_competition(home_name, competition_code)
    away_team = find_team_in_competition(away_name, competition_code)

    if not home_team or not away_team:
        raise HTTPException(
            status_code=404,
            detail=f"Could not find teams in competition {competition_code}. Try
        )

    home_row = get_team_table_row(home_team["id"], competition_code) or {}
    away_row = get_team_table_row(away_team["id"], competition_code) or {}

    home_form = get_last_matches_form(home_team["id"], limit=5)
    away_form = get_last_matches_form(away_team["id"], limit=5)

    # NOTE: injuries / xG not available from Football-Data
    prompt = f"""
You are a professional football analyst.
Use ONLY the real data provided. Do NOT invent injuries, odds, or unknown stats.

MATCH:
{home_team["name"]} vs {away_team["name"]}
COMPETITION: {competition_code}

LEAGUE TABLE (HOME):
Position: {home_row.get("position","N/A")}
Played: {home_row.get("played","N/A")}  Points: {home_row.get("points","N/A")}
W-D-L: {home_row.get("won","N/A")}-{home_row.get("draw","N/A")}-{home_row.get("lo
GF/GA/GD: {home_row.get("gf","N/A")}/{home_row.get("ga","N/A")}/{home_row.get("gd

LEAGUE TABLE (AWAY):
Position: {away_row.get("position","N/A")}
Played: {away_row.get("played","N/A")}  Points: {away_row.get("points","N/A")}
W-D-L: {away_row.get("won","N/A")}-{away_row.get("draw","N/A")}-{away_row.get("lo
GF/GA/GD: {away_row.get("gf","N/A")}/{away_row.get("ga","N/A")}/{away_row.get("gd

RECENT FORM - LAST 5 (HOME):
Results: {" ".join([x["result"] for x in home_form["form"]])}
Goals last 5: {home_form["gf_last5"]} for, {home_form["ga_last5"]} against

RECENT FORM - LAST 5 (AWAY):
Results: {" ".join([x["result"] for x in away_form["form"]])}
Goals last 5: {away_form["gf_last5"]} for, {away_form["ga_last5"]} against

STRICT FORMAT (headings must match exactly):

Overview:
Recent Form:
Key Players & Absences:
Tactical Matchup:
Stats That Matter:
Risk Factors:

Rules:
```

```
- If injuries/absences are unknown, say "No confirmed absences provided by the da
- Base claims on the table + last 5 form.
- Write naturally like a football website preview.
"""

    try:
        response = openai_client.responses.create(
            model="gpt-4.1-mini",
            input=prompt,
            max_output_tokens=700
        )

        return {
            "analysis": response.output_text,
            "meta": {
                "competition_code": competition_code,
                "home_team": home_team,
                "away_team": away_team,
                "home_table": home_row,
                "away_table": away_row,
                "home_form": home_form,
                "away_form": away_form,
            }
        }

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

# =========================
# ACCA BUILDER
# =========================

def get_fixtures_for_competition(competition_code: str):
    """
    Get upcoming fixtures for a competition
    """
    today = datetime.utcnow().date()
    end = today + timedelta(days=7)

    url = (
        f"{FOOTBALL_API_BASE}/competitions/{competition_code}/matches"
        f"?dateFrom={today.isoformat()}&dateTo={end.isoformat()}"
    )

    data = football_get(url)
    matches = data.get("matches", [])
    fixtures = []
    for m in matches:
        home = m.get("homeTeam", {}).get("name")
        away = m.get("awayTeam", {}).get("name")
        home_id = m.get("homeTeam", {}).get("id")
        away_id = m.get("awayTeam", {}).get("id")

        if home and away:
            fixtures.append({
                "home": home,
                "away": away,
                "home_id": home_id,
                "away_id": away_id,
                "competition": competition_code
            })

    return fixtures


# competitions to scan for accas
ACTIVE_COMPETITIONS = [
    "PL",   # Premier League
    "PD",   # La Liga
    "SA",   # Serie A
    "BL1",  # Bundesliga
    "FL1",  # Ligue 1
]


@app.post("/acca/generate")
def generate_acca(payload: dict):
    """
    AI-powered accumulator builder with odds and comprehensive analysis
    """
    acca_type = payload.get("type", "win")

    selections = []

    # 1. Collect fixtures from all leagues
    fixtures = []
    for code in ACTIVE_COMPETITIONS:
        try:
            fixtures.extend(get_fixtures_for_competition(code))
        except:
            continue

    # Limit to prevent too many API calls
```

```
fixtures = fixtures[:30]

# 2. Load league tables for context
tables = {}
for code in ACTIVE_COMPETITIONS:
    try:
        table_data = get_standings_total(code)
        tables[code] = {row["team"]: row for row in table_data}
    except:
        continue

# 3. Get form data for top fixtures
analyzed_fixtures = []

for f in fixtures[:25]:  # Analyze top 25 fixtures
    home = f.get("home")
    away = f.get("away")
    home_id = f.get("home_id")
    away_id = f.get("away_id")
    league = f.get("competition")

    if not all([home, away, home_id, away_id]):
        continue

    try:
        # Get table positions
        home_row = tables.get(league, {}).get(home, {})
        away_row = tables.get(league, {}).get(away, {})

        # Get recent form
        home_form = get_last_matches_form(home_id, limit=5)
        away_form = get_last_matches_form(away_id, limit=5)

        analyzed_fixtures.append({
            "match": f"{home} vs {away}",
            "home": home,
            "away": away,
            "league": league,
            "home_pos": home_row.get("position", "N/A"),
            "away_pos": away_row.get("position", "N/A"),
            "home_points": home_row.get("points", 0),
            "away_points": away_row.get("points", 0),
            "home_gf": home_form['gf_last5'],
            "home_ga": home_form['ga_last5'],
            "away_gf": away_form['gf_last5'],
            "away_ga": away_form['ga_last5'],
            "home_form_str": " ".join([m["result"] for m in home_form["form"]
                "away_form_str": " ".join([m["result"] for m in away_form["form"]
            })
        except:
            continue

    if not analyzed_fixtures:
        return {
            "type": acca_type,
            "selections": [],
            "ai_summary": "Not enough data available for analysis."
        }

    # 4. Build AI prompt based on acca type
    if acca_type == "win":
        analysis_prompt = f"""You are a professional football betting analyst. An

Look for:
- Significant position gaps (5+ places)
- Strong recent form (3+ wins in last 5)
- Goal-scoring ability vs defensive weakness
- Home advantage for top teams
- Momentum and consistency

FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home #{f['home_pos']} ({f['home_

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: [Team Name] to Win
LEAGUE: [League Code]
ODDS: [1.50-3.00 realistic odds estimate]
REASON: [2-3 sentences with specific stats: position gap, recent form, goals for/
CONFIDENCE: [High/Medium]
---

Provide 4-5 selections with the STRONGEST win probability. Focus on value (odds 1

    elif acca_type == "goals":
        analysis_prompt = f"""You are a professional football betting analyst. An

Look for:
- Combined high scoring (6+ goals in last 5 matches combined)
- Both teams scoring regularly
- Defensive weaknesses (high GA)
- Open, attacking matchups
- Historical high-scoring encounters
```

```
FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home GF/GA: {f['home_gf']}/{f['h

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: Over 2.5 Goals
LEAGUE: [League Code]
ODDS: [1.60-2.20 realistic odds estimate]
REASON: [2-3 sentences explaining combined attacking threat, defensive weaknesses
CONFIDENCE: [High/Medium]
---

Provide 4-5 selections with high goal-scoring potential."""

    elif acca_type == "underdog":
        analysis_prompt = f"""You are a professional football betting analyst. An

Look for:
- Strong away teams (good away form, high position)
- Overrated home teams (recent poor form)
- Position gap less than 5 places
- Away team on winning streak
- Home team defensive issues

FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home #{f['home_pos']} (Form: {f[

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: [Away team] Win OR Draw (specify which)
LEAGUE: [League Code]
ODDS: [2.50-5.00 realistic odds estimate]
REASON: [2-3 sentences explaining why underdog has value: away form, home weaknes
CONFIDENCE: [Medium/Low - underdogs are riskier]
---

Provide 3-4 VALUE underdog selections. These should have genuine upset potential,

    # 5. Call OpenAI for analysis
    try:
        response = openai_client.responses.create(
            model="gpt-4.1-mini",
            input=analysis_prompt,
            max_output_tokens=1500
        )

        ai_output = response.output_text
```

```
# 6. Parse AI response
blocks = ai_output.split("---")

for block in blocks:
    block = block.strip()
    if not block or "MATCH:" not in block:
        continue

    try:
        # Extract fields
        match_line = [l for l in block.split("\n") if l.startswith("MATCH
        pick_line = [l for l in block.split("\n") if l.startswith("PICK:"
        league_line = [l for l in block.split("\n") if l.startswith("LEAG
        odds_line = [l for l in block.split("\n") if l.startswith("ODDS:"
        reason_line = [l for l in block.split("\n") if l.startswith("REAS
        confidence_line = [l for l in block.split("\n") if l.startswith("

        match = match_line.replace("MATCH:", "").strip()
        pick = pick_line.replace("PICK:", "").strip()
        league = league_line.replace("LEAGUE:", "").strip()
        odds_str = odds_line.replace("ODDS:", "").strip()
        reason = reason_line.replace("REASON:", "").strip()
        confidence = confidence_line[0].replace("CONFIDENCE:", "").strip(

        # Parse odds
        try:
            odds = float(odds_str)
        except:
            odds = 2.00  # Default odds

        selections.append({
            "match": match,
            "pick": pick,
            "league": league,
            "odds": odds,
            "reason": reason,
            "confidence": confidence
        })
    except Exception as e:
        print(f"Parse error: {e}")
        continue

# Generate summary
total_odds = 1.0
for s in selections[:5]:
    total_odds *= s.get("odds", 2.0)
```

```
        summary_prompt = f"""You created a {acca_type} accumulator with {len(sele

        summary_response = openai_client.responses.create(
            model="gpt-4.1-mini",
            input=summary_prompt,
            max_output_tokens=150
        )

        return {
            "type": acca_type,
            "selections": selections[:5],
            "total_odds": round(total_odds, 2),
            "ai_summary": summary_response.output_text
        }

    except Exception as e:
        print(f"AI Error: {e}")
        # Fallback logic
        fallback_selections = []

        for f in analyzed_fixtures[:5]:
            if acca_type == "win":
                pos_diff = f.get('away_pos', 99) - f.get('home_pos', 99) if isins
                if pos_diff >= 5:
                    fallback_selections.append({
                        "match": f['match'],
                        "pick": f"{f['home']} to Win",
                        "league": f['league'],
                        "odds": 1.80,
                        "reason": f"Home team {pos_diff} positions higher in tabl
                        "confidence": "Medium"
                    })

        total_odds = 1.0
        for s in fallback_selections:
            total_odds *= s.get("odds", 2.0)

        return {
            "type": acca_type,
            "selections": fallback_selections[:5],
            "total_odds": round(total_odds, 2),
            "ai_summary": "Selections based on league position and recent form an
        }


# ========================
# FIRST HALF ACCA BUILDER (AI-POWERED)
# ========================
@app.post("/fh-acca/generate")
def generate_fh_acca(payload: dict):
    """
    AI-powered first half acca builder
    Analyzes recent match data to find teams with strong first half performance
    """
    acca_type = payload.get("type", "fh-goal")

    selections = []

    # 1. Collect fixtures from all active leagues
    fixtures = []
    for code in ACTIVE_COMPETITIONS:
        try:
            fixtures.extend(get_fixtures_for_competition(code))
        except:
            continue

    # Limit to prevent too many API calls
    fixtures = fixtures[:30]

    # 2. For each fixture, get recent form data for both teams
    analyzed_fixtures = []

    for f in fixtures[:20]:  # Analyze top 20 fixtures
        home = f.get("home")
        away = f.get("away")
        home_id = f.get("home_id")
        away_id = f.get("away_id")
        league = f.get("competition")

        if not all([home, away, home_id, away_id]):
            continue

        try:
            # Get recent form for both teams
            home_form = get_last_matches_form(home_id, limit=5)
            away_form = get_last_matches_form(away_id, limit=5)

            analyzed_fixtures.append({
                "match": f"{home} vs {away}",
                "home": home,
                "away": away,
                "league": league,
                "home_form": home_form,
                "away_form": away_form
            })
        except:
            continue

    if not analyzed_fixtures:
        return {
            "type": acca_type,
            "selections": [],
            "ai_summary": "Not enough data available for analysis."
        }

    # 3. Build AI prompt based on type
    if acca_type == "fh-goal":
        analysis_prompt = f"""You are a professional football betting analyst spe

Analyze these upcoming matches and find 4-5 matches where AT LEAST ONE TEAM is li

Look for:
- Teams that consistently score early (high goals in recent matches)
- Teams with strong attacking form (multiple goals in last 5 games)
- Matches between attacking teams (both teams scoring regularly)

FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home last 5 scored {f['home_form

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: [First Half Goal - Home/Away/Both teams]
LEAGUE: [League Code]
ODDS: [1.50-2.20 realistic odds estimate]
REASON: [1-2 sentence explanation focusing on first half/early goal trends]
CONFIDENCE: [High/Medium]
---

Provide 4-5 selections only. Be specific about first half performance."""

    elif acca_type == "fh-1.5":
        analysis_prompt = f"""You are a professional football betting analyst spe

Analyze these upcoming matches and find 3-4 matches likely to have OVER 1.5 GOALS

Look for:
- Both teams with high scoring rates (combined 4+ goals in recent games)
- High-tempo attacking matchups
- Defensive weaknesses (high goals conceded)
- Historical high-scoring first halves
```

```
FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home GF {f['home_form']]['gf_last

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: Over 1.5 First Half Goals
LEAGUE: [League Code]
ODDS: [2.00-3.50 realistic odds estimate]
REASON: [1-2 sentence explanation focusing on combined attacking threat in first
CONFIDENCE: [High/Medium]
---

Provide 3-4 selections only. Focus on matches with explosive first half potential

    elif acca_type == "early-goal":
        analysis_prompt = f"""You are a professional football betting analyst spe

Analyze these upcoming matches and find 4-5 matches where a goal is likely BEFORE

Look for:
- Teams that score multiple goals (suggesting fast starts)
- Strong attacking home teams
- Matches with high goal expectancy
- Teams that concede early (poor defensive organization)

FIXTURES TO ANALYZE:
{chr(10).join([f"- {f['match']} ({f['league']}): Home scored {f['home_form']]['gf_

FORMAT YOUR RESPONSE EXACTLY LIKE THIS (one selection per line):
MATCH: [Team A vs Team B]
PICK: Goal Before 30 Minutes
LEAGUE: [League Code]
ODDS: [1.40-2.00 realistic odds estimate]
REASON: [1-2 sentence explanation focusing on fast starts/early pressure]
CONFIDENCE: [High/Medium]
---

Provide 4-5 selections only. Focus on matches likely to have early action."""

    # 4. Call OpenAI for analysis
    try:
        response = openai_client.responses.create(
            model="gpt-4.1-mini",
            input=analysis_prompt,
            max_output_tokens=1200
        )
```

```
ai_output = response.output_text

# 5. Parse AI response
blocks = ai_output.split("---")

for block in blocks:
    block = block.strip()
    if not block or "MATCH:" not in block:
        continue

    try:
        # Extract fields
        match_line = [l for l in block.split("\n") if l.startswith("MATCH
        pick_line = [l for l in block.split("\n") if l.startswith("PICK:"
        league_line = [l for l in block.split("\n") if l.startswith("LEAG
        odds_line = [l for l in block.split("\n") if l.startswith("ODDS:"
        reason_line = [l for l in block.split("\n") if l.startswith("REAS
        confidence_line = [l for l in block.split("\n") if l.startswith("

        match = match_line.replace("MATCH:", "").strip()
        pick = pick_line.replace("PICK:", "").strip()
        league = league_line.replace("LEAGUE:", "").strip()
        reason = reason_line.replace("REASON:", "").strip()
        confidence = confidence_line[0].replace("CONFIDENCE:", "").strip(

        # Parse odds
        odds = 2.00  # Default
        if odds_line:
            try:
                odds_str = odds_line[0].replace("ODDS:", "").strip()
                odds = float(odds_str)
            except:
                odds = 2.00

        selections.append({
            "match": match,
            "pick": pick,
            "league": league,
            "odds": odds,
            "reason": reason,
            "confidence": confidence
        })
    except:
        continue

# Generate summary
```

```
        summary_prompt = f"""Based on the {acca_type} acca you just created with

        summary_response = openai_client.responses.create(
            model="gpt-4.1-mini",
            input=summary_prompt,
            max_output_tokens=150
        )

        return {
            "type": acca_type,
            "selections": selections[:5],  # Max 5 selections
            "ai_summary": summary_response.output_text
        }

    except Exception as e:
        print(f"AI Error: {e}")
        # Fallback to simple logic if AI fails
        fallback_selections = []

        for f in analyzed_fixtures[:5]:
            total_goals = f['home_form']['gf_last5'] + f['away_form']['gf_last5']

            if total_goals >= 8:  # High scoring teams
                fallback_selections.append({
                    "match": f['match'],
                    "pick": "First Half Goal" if acca_type == "fh-goal" else "Ove
                    "league": f['league'],
                    "reason": f"Both teams in good scoring form ({total_goals} go
                    "confidence": "Medium"
                })

        return {
            "type": acca_type,
            "selections": fallback_selections[:5],
            "ai_summary": "Analysis based on recent scoring form across competiti
        }


# ========================
# ACCA TRACKING & P&L
# ========================

# In-memory storage (replace with database in production)
SAVED_ACCAS: Dict[str, List[Dict[str, Any]]] = {}

class SaveAccaRequest(BaseModel):
    acca_type: str
    selections: List[Dict[str, Any]]
    total_odds: float
    stake: Optional[float] = 10.0
    ai_summary: Optional[str] = ""

class UpdateAccaRequest(BaseModel):
    acca_id: str
    result: str  # "pending", "won", "lost", "void"

@app.post("/acca/save")
def save_acca(req: SaveAccaRequest):
    """
    Save an acca for tracking
    """
    import uuid
    from datetime import datetime

    acca_id = str(uuid.uuid4())[:8]

    # Calculate potential return
    potential_return = req.stake * req.total_odds

    acca = {
        "id": acca_id,
        "type": req.acca_type,
        "selections": req.selections,
        "total_odds": req.total_odds,
        "stake": req.stake,
        "potential_return": round(potential_return, 2),
        "ai_summary": req.ai_summary,
        "created_at": datetime.utcnow().isoformat(),
        "result": "pending",
        "actual_return": 0.0
    }

    # Store by session (in production, use user ID)
    session_id = "default_user"
    if session_id not in SAVED_ACCAS:
        SAVED_ACCAS[session_id] = []

    SAVED_ACCAS[session_id].append(acca)

    return {
        "success": True,
        "acca_id": acca_id,
        "message": "Acca saved successfully"
    }
```

```
@app.get("/acca/saved")
def get_saved_accas():
    """
    Get all saved accas with P&L summary
    """
    session_id = "default_user"
    accas = SAVED_ACCAS.get(session_id, [])

    # Calculate P&L
    total_staked = sum(a["stake"] for a in accas)
    total_returned = sum(a["actual_return"] for a in accas)
    profit_loss = total_returned - total_staked

    pending_count = len([a for a in accas if a["result"] == "pending"])
    won_count = len([a for a in accas if a["result"] == "won"])
    lost_count = len([a for a in accas if a["result"] == "lost"])

    return {
        "accas": sorted(accas, key=lambda x: x["created_at"], reverse=True),
        "stats": {
            "total_accas": len(accas),
            "pending": pending_count,
            "won": won_count,
            "lost": lost_count,
            "total_staked": round(total_staked, 2),
            "total_returned": round(total_returned, 2),
            "profit_loss": round(profit_loss, 2),
            "roi": round((profit_loss / total_staked * 100) if total_staked > 0 e
        }
    }

@app.post("/acca/update")
def update_acca(req: UpdateAccaRequest):
    """
    Update acca result (won/lost)
    """
    session_id = "default_user"
    accas = SAVED_ACCAS.get(session_id, [])

    for acca in accas:
        if acca["id"] == req.acca_id:
            acca["result"] = req.result

            if req.result == "won":
                acca["actual_return"] = acca["potential_return"]
            elif req.result == "lost":
                acca["actual_return"] = 0.0
            elif req.result == "void":
                acca["actual_return"] = acca["stake"]  # Return stake

            return {"success": True, "message": f"Acca marked as {req.result}"}

    return {"success": False, "message": "Acca not found"}


class UpdateLegRequest(BaseModel):
    acca_id: str
    leg_index: int
    result: str  # "won", "lost", "pending"

@app.post("/acca/update-leg")
def update_acca_leg(req: UpdateLegRequest):
    """
    Update individual leg result within an acca
    """
    session_id = "default_user"
    accas = SAVED_ACCAS.get(session_id, [])

    for acca in accas:
        if acca["id"] == req.acca_id:
            # Initialize leg_results if not exists
            if "leg_results" not in acca:
                acca["leg_results"] = ["pending"] * len(acca["selections"])

            # Update the specific leg
            if 0 <= req.leg_index < len(acca["leg_results"]):
                acca["leg_results"][req.leg_index] = req.result

                # Check if all legs are decided
                if all(r in ["won", "lost"] for r in acca["leg_results"]):
                    # If any leg lost, whole acca lost
                    if "lost" in acca["leg_results"]:
                        acca["result"] = "lost"
                        acca["actual_return"] = 0.0
                    else:
                        # All won
                        acca["result"] = "won"
                        acca["actual_return"] = acca["potential_return"]

                return {
                    "success": True,
                    "message": f"Leg {req.leg_index + 1} marked as {req.result}",
                    "acca_status": acca["result"]
                }

    return {"success": False, "message": "Acca not found"}

@app.delete("/acca/delete/{acca_id}")
def delete_acca(acca_id: str):
    """
    Delete a saved acca
    """
    session_id = "default_user"
    accas = SAVED_ACCAS.get(session_id, [])

    SAVED_ACCAS[session_id] = [a for a in accas if a["id"] != acca_id]

    return {"success": True, "message": "Acca deleted"}

@app.delete("/acca/clear-all")
def clear_all_accas():
    """
    Clear all saved accas
    """
    session_id = "default_user"
    SAVED_ACCAS[session_id] = []

    return {"success": True, "message": "All accas cleared"}


# ==========================
# AI CHAT / WHISPERER
# ==========================

class AIChatRequest(BaseModel):
    message: str
    context: Optional[str] = ""
    history: Optional[List[Dict[str, str]]] = []

@app.post("/ai-chat")
def ai_chat(req: AIChatRequest):
    """
    Conversational AI assistant for betting questions
    """
    try:
        # Build strong system message with real-time context
        system_msg = f"""You are a football betting AI assistant.

🔴 CRITICAL: TODAY IS {datetime.now().strftime('%B %d, %Y')} - This is the 2025-2
```

```
Your training data is OUTDATED (ends 2024). You MUST use ONLY the real-time data

{req.context if req.context else 'No real-time data available - ask user to selec

RULES:
1. ONLY use data from the context above for current season info
2. Quote specific positions, points, and results from the data
3. Say "Based on current standings..." when using the data
4. If data missing, say "Select a league to get current data"
5. Keep responses under 150 words
6. Be conversational and helpful"""

        # Build messages array for Chat Completions API
        messages = [{"role": "system", "content": system_msg}]

        # Add conversation history
        for msg in req.history[-5:]:  # Last 5 messages
            messages.append(msg)

        # Add current message if not in history
        if not req.history or req.history[-1].get('content') != req.message:
            messages.append({"role": "user", "content": req.message})

        # Call OpenAI Chat Completions (NOT Responses)
        response = openai_client.chat.completions.create(
            model="gpt-4o-mini",  # Latest fast model with up-to-date training
            messages=messages,
            max_tokens=300,
            temperature=0.7
        )

        ai_response = response.choices[0].message.content.strip()

        return {
            "response": ai_response,
            "success": True
        }

    except Exception as e:
        print(f"AI Chat Error: {e}")
        return {
            "response": "I'm having trouble right now. Could you rephrase your qu
            "success": False,
            "error": str(e)
        }
```

```
@app.post("/ai-chat-claude")
def ai_chat_claude(req: AIChatRequest):
    """
    Claude AI chat with WEB SEARCH capability!
    NO context needed - Claude searches web automatically!
    """
    if not anthropic_client:
        return {
            "response": "Claude AI not configured. Set ANTHROPIC_API_KEY environm
            "success": False
        }

    try:
        system_prompt = f"""You are an expert football betting assistant.

TODAY: {datetime.now().strftime('%B %d, %Y')} (2025-26 season)

When asked about current standings, scores, injuries, or news:
- Use web_search to find latest information
- Cite reliable sources (Premier League, BBC Sport, Sky Sports, ESPN)
- Provide current 2025-26 season data
- Keep responses under 150 words, conversational and helpful"""

        response = anthropic_client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=1024,
            system=system_prompt,
            tools=[{"type": "web_search_20250305", "name": "web_search"}],
            messages=[{"role": "user", "content": req.message}]
        )

        ai_response = ""
        used_search = False

        for block in response.content:
            if block.type == "text":
                ai_response += block.text
            elif block.type == "tool_use":
                used_search = True

        return {
            "response": ai_response,
            "success": True,
            "used_search": used_search
        }

    except Exception as e:
        print(f"Claude Error: {e}")
        return {
            "response": f"Error: {str(e)}",
            "success": False
        }


@app.post("/ai-chat-gemini")
def ai_chat_gemini(req: AIChatRequest):
    """
    Google Gemini AI chat with Google Search integration (FREE!)
    NO context needed - Gemini searches Google automatically!
    """
    if not gemini_available:
        return {
            "response": "Gemini AI is not configured. Please set GOOGLE_API_KEY."
            "success": False
        }

    try:
        # Configure Gemini model with search
        model = genai.GenerativeModel(
            'gemini-pro',
            tools='google_search_retrieval'  # Enable Google Search!
        )

        # Build prompt
        prompt = f"""You are an expert football betting assistant.

TODAY'S DATE: {datetime.now().strftime('%B %d, %Y')} (2025-26 football season)

User question: {req.message}

When answering:
- Search Google for current information if needed
- Focus on 2025-26 season data
- Cite sources when using search results
- Keep response under 150 words
- Be conversational and helpful"""

        # Generate response with Google Search
        response = model.generate_content(prompt)

        return {
            "response": response.text,
            "success": True,
            "model": "gemini-pro",
            "used_search": True
        }

    except Exception as e:
        print(f"Gemini Error: {e}")
        return {
            "response": f"I had trouble with that. Error: {str(e)}",
            "success": False
        }


# ==========================
# SCREENSHOT ANALYSIS (GPT-4 Vision)
# ==========================

class ScreenshotRequest(BaseModel):
    image: str  # base64 encoded image
    prompt: str

@app.post("/analyze-screenshot")
def analyze_screenshot(req: ScreenshotRequest):
    """
    Analyze acca screenshot using GPT-4 Vision
    """
    try:
        # Build vision prompt
        vision_prompt = f"""{req.prompt}

Provide:
1. List all bets you can see (teams, bet types, odds)
2. Analysis of each pick (form, value, risk)
3. Overall acca assessment (good/bad, why)
4. Suggestions for improvement

Be conversational and helpful. Keep under 200 words."""

        # Call OpenAI Vision API (Chat Completions)
        response = openai_client.chat.completions.create(
            model="gpt-4o-mini",  # Vision-capable model
            messages=[
                {
                    "role": "user",
                    "content": [
                        {
                            "type": "text",
                            "text": vision_prompt
                        },
                        {
                            "type": "image_url",
                            "image_url": {
                                "url": f"data:image/jpeg;base64,{req.image}"
                            }
                        }
                    ]
                }
            ],
            max_tokens=400
        )

        analysis = response.choices[0].message.content.strip()

        return {
            "analysis": analysis,
            "success": True
        }

    except Exception as e:
        print(f"Screenshot Analysis Error: {e}")
        return {
            "analysis": "I couldn't analyze that screenshot. Please make sure it
            "success": False,
            "error": str(e)
        }


# ========================
# API-FOOTBALL ENDPOINTS (Premium Features)
# ========================

@app.get("/api-football/injuries/{league_code}")
def get_injuries(league_code: str):
    """
    Get current injuries for a league from API-Football
    """
    try:
        league_id = LEAGUE_MAPPING.get(league_code)
        if not league_id:
            return {"error": "League not found", "injuries": {}}

        url = f"{API_FOOTBALL_BASE}/injuries"
        params = {
            "league": league_id,
            "season": 2025
        }
        response = requests.get(url, headers=API_FOOTBALL_HEADERS, params=params,
        data = response.json()

        # Group injuries by team
        injuries_by_team = {}

        if data.get("response"):
            for injury in data["response"]:
                team = injury["team"]["name"]

                if team not in injuries_by_team:
                    injuries_by_team[team] = []

                injuries_by_team[team].append({
                    "player": injury["player"]["name"],
                    "type": injury["player"]["type"],
                    "reason": injury["player"]["reason"]
                })

        return {"injuries": injuries_by_team, "success": True}

    except Exception as e:
        print(f"API-Football Injuries Error: {e}")
        return {"error": str(e), "injuries": {}}


@app.get("/api-football/odds/{fixture_id}")
def get_fixture_odds(fixture_id: int):
    """
    Get betting odds for a specific fixture from API-Football
    """
    try:
        url = f"{API_FOOTBALL_BASE}/odds"
        params = {
            "fixture": fixture_id
        }

        response = requests.get(url, headers=API_FOOTBALL_HEADERS, params=params,
        data = response.json()

        odds_comparison = {}

        if data.get("response") and len(data["response"]) > 0:
            bookmakers = data["response"][0].get("bookmakers", [])

            # Get odds from top bookmakers
            for bookmaker in bookmakers[:10]:  # Top 10 bookmakers
                name = bookmaker["name"]

                for bet in bookmaker.get("bets", []):
                    if bet["name"] == "Match Winner":
                        odds_comparison[name] = {}
                        for outcome in bet.get("values", []):
                            odds_comparison[name][outcome["value"].lower()] = out

        return {"odds": odds_comparison, "success": True}

    except Exception as e:
        print(f"API-Football Odds Error: {e}")
        return {"error": str(e), "odds": {}}


@app.get("/api-football/lineups/{fixture_id}")
def get_fixture_lineups(fixture_id: int):
    """
    Get confirmed lineups for a fixture from API-Football
    """
    try:
        url = f"{API_FOOTBALL_BASE}/fixtures/lineups"
        params = {
            "fixture": fixture_id
        }

        response = requests.get(url, headers=API_FOOTBALL_HEADERS, params=params,
        data = response.json()

        lineups = {}

        if data.get("response"):
            for team_lineup in data["response"]:
                team_name = team_lineup["team"]["name"]
                lineups[team_name] = {
                    "formation": team_lineup.get("formation", "Unknown"),
                    "startXI": [p["player"]["name"] for p in team_lineup.get("sta
                    "substitutes": [p["player"]["name"] for p in team_lineup.get(
                    "coach": team_lineup.get("coach", {}).get("name", "Unknown")
                }

        return {"lineups": lineups, "success": True}

    except Exception as e:
        print(f"API-Football Lineups Error: {e}")
        return {"error": str(e), "lineups": {}}
```

```
@app.get("/api-football/match-odds/{home_team}/{away_team}")
def get_match_odds(home_team: str, away_team: str):
    """
    Get pre-match odds for a specific match (by team names)
    Returns best odds from multiple bookmakers
    """
    try:
        # Clean team names (remove FC, AFC, etc.)
        def clean_team_name(name: str) -> str:
            # Remove common suffixes
            cleaned = name.replace(" FC", "").replace(" AFC", "").replace(" Unite
            return cleaned.strip()

        home_clean = clean_team_name(home_team)
        away_clean = clean_team_name(away_team)

        # First, find the fixture ID by searching fixtures
        url = f"{API_FOOTBALL_BASE}/fixtures"
        params = {
            "season": 2025,
            "team": home_clean,  # Search by cleaned home team
            "next": 10
        }

        response = requests.get(url, headers=API_FOOTBALL_HEADERS, params=params,
        data = response.json()

        fixture_id = None

        # Find the matching fixture with flexible matching
        if data.get("response"):
            for fixture in data["response"]:
                fixture_home = clean_team_name(fixture["teams"]["home"]["name"])
                fixture_away = clean_team_name(fixture["teams"]["away"]["name"])

                if (fixture_home.lower() in home_clean.lower() or home_clean.lowe
                   (fixture_away.lower() in away_clean.lower() or away_clean.lowe
                    fixture_id = fixture["fixture"]["id"]
                    break

        # If match not found in API-Football, return mock data for demo
        if not fixture_id:
            print(f"Match not found in API-Football: {home_team} vs {away_team}")
            # Return mock data so widget still shows
            return {
```

```
            "odds": {
                "Bet365": {"Home": 1.70, "Draw": 3.40, "Away": 4.50},
                "William Hill": {"Home": 1.65, "Draw": 3.50, "Away": 4.60},
                "Paddy Power": {"Home": 1.68, "Draw": 3.45, "Away": 4.55}
            },
            "best_odds": {
                "home": {"bookmaker": "Bet365", "odds": 1.70},
                "draw": {"bookmaker": "William Hill", "odds": 3.50},
                "away": {"bookmaker": "William Hill", "odds": 4.60}
            },
            "fixture_id": None,
            "success": True,
            "note": "Demo odds - API-Football match not found"
        }

    # Now get odds for this fixture
    odds_url = f"{API_FOOTBALL_BASE}/odds"
    odds_params = {
        "fixture": fixture_id
    }

    odds_response = requests.get(odds_url, headers=API_FOOTBALL_HEADERS, para
    odds_data = odds_response.json()

    all_odds = {}
    best_odds = {
        "home": {"bookmaker": "", "odds": 0},
        "draw": {"bookmaker": "", "odds": 0},
        "away": {"bookmaker": "", "odds": 0}
    }

    if odds_data.get("response") and len(odds_data["response"]) > 0:
        bookmakers = odds_data["response"][0].get("bookmakers", [])

        # Collect all odds and find best
        for bookmaker in bookmakers[:15]:  # Top 15 bookmakers
            name = bookmaker["name"]

            for bet in bookmaker.get("bets", []):
                if bet["name"] == "Match Winner":
                    odds_dict = {}
                    for outcome in bet.get("values", []):
                        odds_value = float(outcome["odd"])
                        odds_dict[outcome["value"]] = odds_value

                        # Track best odds
                        if outcome["value"] == "Home" and odds_value > best_o
```

```
                            best_odds["home"] = {"bookmaker": name, "odds": o
                        elif outcome["value"] == "Draw" and odds_value > best
                            best_odds["draw"] = {"bookmaker": name, "odds": o
                        elif outcome["value"] == "Away" and odds_value > best
                            best_odds["away"] = {"bookmaker": name, "odds": o

                    all_odds[name] = odds_dict

        # If no odds found, return mock data
        if best_odds["home"]["odds"] == 0:
            return {
                "odds": {
                    "Bet365": {"Home": 1.70, "Draw": 3.40, "Away": 4.50},
                    "William Hill": {"Home": 1.65, "Draw": 3.50, "Away": 4.60},
                    "Paddy Power": {"Home": 1.68, "Draw": 3.45, "Away": 4.55}
                },
                "best_odds": {
                    "home": {"bookmaker": "Bet365", "odds": 1.70},
                    "draw": {"bookmaker": "William Hill", "odds": 3.50},
                    "away": {"bookmaker": "William Hill", "odds": 4.60}
                },
                "fixture_id": fixture_id,
                "success": True,
                "note": "Demo odds - no bookmaker odds available yet"
            }

        return {
            "odds": all_odds,
            "best_odds": best_odds,
            "fixture_id": fixture_id,
            "success": True
        }

    except Exception as e:
        print(f"API-Football Match Odds Error: {e}")
        # Return mock data on error so widget still works
        return {
            "odds": {
                "Bet365": {"Home": 1.70, "Draw": 3.40, "Away": 4.50}
            },
            "best_odds": {
                "home": {"bookmaker": "Bet365", "odds": 1.70},
                "draw": {"bookmaker": "Bet365", "odds": 3.40},
                "away": {"bookmaker": "Bet365", "odds": 4.50}
            },
            "success": True,
            "error": str(e),
    "note": "Demo odds - API error"
}
```