Basketball Analysis Assessment App - Deployment Analysis

📋 Executive Summary

This document provides a comprehensive analysis of the Basketball Analysis Assessment App repository structure, dependencies, configurations, and deployment requirements. The application consists of two main components:

1. **Frontend**: Next.js 14 application with TypeScript, React 18, and Prisma ORM
2. **Backend**: FastAPI Python application with MediaPipe pose detection

---

🏗️ Repository Structure

Root Directory

```
basketball_app/
├── basketball-analysis/      # Next.js Frontend
├── python-backend/           # FastAPI Backend
├── generate_skeleton_overlay.py
├── run_replicate_skeleton.py
└── .git/
```

Frontend Structure (`basketball-analysis/`)

```
basketball-analysis/
├── prisma/
│   └── schema.prisma                # PostgreSQL database schema
├── public/
│   └── images/
│       ├── player-shooting.jpg
│       └── test-player.jpg
├── src/
│   ├── app/                         # Next.js App Router
│   │   ├── elite-shooters/          # Elite shooters comparison page
│   │   ├── fonts/                   # Custom fonts
│   │   ├── results/                 # Analysis results pages
│   │   │   └── demo/                # Demo results page
│   │   ├── favicon.ico
│   │   ├── globals.css              # Global styles
│   │   ├── layout.tsx               # Root layout
│   │   ├── page.tsx                 # Home page
│   │   └── providers.tsx            # React Query provider
│   ├── components/
│   │   ├── analysis/                # Analysis UI components
│   │   │   ├── AnalysisDashboard.tsx
│   │   │   ├── AnalysisOverlay.tsx
│   │   │   ├── AnalysisProgress.tsx
│   │   │   ├── EnhancedSkeletonOverlay.tsx
│   │   │   ├── ExportButton.tsx
│   │   │   ├── FormScoreCard.tsx
│   │   │   ├── OverlayControls.tsx
│   │   │   └── SkeletonOverlay.tsx
│   │   ├── layout/
│   │   │   ├── Footer.tsx
│   │   │   └── Header.tsx
│   │   ├── ui/                      # Shadcn UI components
│   │   │   ├── badge.tsx
│   │   │   ├── button.tsx
│   │   │   ├── card.tsx
│   │   │   ├── input.tsx
│   │   │   ├── label.tsx
│   │   │   ├── progress.tsx
│   │   │   ├── select.tsx
│   │   │   └── tabs.tsx
│   │   └── upload/
│   │       ├── MediaUpload.tsx
│   │       └── PlayerProfileForm.tsx
│   ├── data/
│   │   └── eliteShooters.ts         # Elite shooters data
│   ├── lib/
│   │   ├── biomechanicalAnalysis.ts
│   │   ├── formAnalysis.ts
│   │   ├── mediapipePoseDetection.ts
│   │   ├── poseDetection.ts
│   │   ├── stagedFormAnalysis.ts
│   │   ├── tensorflowPoseDetection.ts
│   │   └── utils.ts                 # Utility functions
│   ├── services/
│   │   ├── eliteShooters.ts
│   │   ├── poseDetection.ts
│   │   ├── pythonBackendApi.ts      # Python backend API client
│   │   └── reportGeneration.ts
│   ├── stores/
│   │   └── analysisStore.ts         # Zustand state management
│   └── types/
│       └── index.ts                 # TypeScript type definitions
```

```
├── .env                              # Environment variables (local)
├── .env.example                      # Environment variables template
├── .eslintrc.json                    # ESLint configuration
├── next.config.mjs                   # Next.js configuration
├── package.json                      # Node dependencies
├── package-lock.json
├── postcss.config.mjs                # PostCSS configuration
├── prisma.config.ts                  # Prisma configuration
├── README.md
├── tailwind.config.ts                # Tailwind CSS configuration
├── tsconfig.json                     # TypeScript configuration
└── yarn.lock
```

Backend Structure (`python-backend/`)

```
python-backend/
├── app/
│   ├── __init__.py
│   ├── anatomical_skeleton.py        # Anatomical skeleton overlay generator
│   ├── main.py                       # FastAPI application entry point
│   ├── models.py                     # Pydantic models
│   ├── pose_detection.py             # MediaPipe pose detection service
│   └── skeleton_drawing.py           # Skeleton overlay drawing utilities
├── .env                              # Environment variables (created)
├── .env.example                      # Environment variables template (created)
├── Dockerfile                        # Docker configuration
├── process_image.py                  # CLI image processing script
├── requirements.txt                  # Python dependencies (updated)
└── run_dev.sh                        # Development server script
```

📦 Dependencies Analysis

Frontend Dependencies (package.json)

Core Framework

- **next**: ^14.2.33 - Next.js framework
- **react**: ^18 - React library
- **react-dom**: ^18 - React DOM renderer

Database & ORM

- **@prisma/client**: ^7.1.0 - Prisma ORM client
- **prisma**: ^7.1.0 (dev) - Prisma CLI

UI Components & Styling

- **@radix-ui/*** - Multiple Radix UI primitives for accessible components
- **tailwindcss**: ^3.4.1 - Utility-first CSS framework
- **class-variance-authority**: ^0.7.1 - CVA for component variants
- **clsx**: ^2.1.1 - Class name utility
- **tailwind-merge**: ^3.4.0 - Tailwind class merging
- **lucide-react**: ^0.555.0 - Icon library
- **framer-motion**: ^12.23.25 - Animation library

Data Visualization

- **recharts**: ^3.5.1 - Charting library
- **plotly.js**: ^3.3.0 - Plotly charting
- **react-plotly.js**: ^2.6.0 - React wrapper for Plotly

Machine Learning / Computer Vision

- **@mediapipe/pose**: ^0.5.1675469404 - MediaPipe pose detection
- **@tensorflow-models/pose-detection**: ^2.1.3 - TensorFlow pose models
- **@tensorflow/tfjs**: ^4.22.0 - TensorFlow.js
- **@tensorflow/tfjs-backend-webgpu**: ^4.22.0 - WebGPU backend

State Management & Data Fetching

- **zustand**: ^5.0.9 - State management
- **@tanstack/react-query**: ^5.90.11 - Server state management

Form Management

- **react-hook-form**: ^7.67.0 - Form handling
- **@hookform/resolvers**: ^5.2.2 - Form validation resolvers
- **zod**: ^4.1.13 - Schema validation

File Handling

- **react-dropzone**: ^14.3.8 - File upload component
- **html-to-image**: ^1.11.13 - HTML to image conversion

Cloud Storage (AWS)

- **@aws-sdk/client-s3**: ^3.943.0 - AWS S3 client
- **@aws-sdk/s3-request-presigner**: ^3.943.0 - S3 presigned URLs

Authentication

- **next-auth**: ^4.24.13 - Authentication for Next.js

Utilities

- **uuid**: ^13.0.0 - UUID generation
- **@types/uuid**: ^11.0.0 - UUID TypeScript types

Development Dependencies

- **typescript**: ^5 - TypeScript compiler
- **@types/node**: ^20 - Node.js type definitions
- **@types/react**: ^18 - React type definitions
- **@types/react-dom**: ^18 - React DOM type definitions
- **@types/plotly.js**: ^3.0.8 - Plotly type definitions
- **eslint**: ^8 - JavaScript linter
- **eslint-config-next**: 14.2.33 - Next.js ESLint config
- **postcss**: ^8 - CSS transformations

Backend Dependencies (requirements.txt)

FastAPI Framework

- **fastapi**: 0.109.2 - Modern web framework
- **uvicorn[standard]**: 0.27.1 - ASGI server
- **python-multipart**: 0.0.9 - Multipart form data parsing

- **starlette**: 0.36.3 - ASGI framework (CORS support)

Computer Vision & ML

- **mediapipe**: 0.10.9 - Google's MediaPipe library
- **opencv-python**: 4.9.0.80 - OpenCV for image processing
- **Pillow**: 10.2.0 - Python Imaging Library
- **numpy**: 1.26.4 - Numerical computing

AI/ML APIs

- **replicate**: 0.25.1 - Replicate API client

HTTP & Networking

- **requests**: 2.31.0 - HTTP library

Data Validation & Configuration

- **pydantic**: 2.6.1 - Data validation
- **python-dotenv**: 1.0.1 - Environment variable management

🔐 Environment Variables & Configuration Files

Frontend Environment Variables

Current `.env` (Local Development)

```
DATABASE_URL="prisma+postgres://localhost:51213/?api_key=..."
```

`.env.example` Template

```
# Python Backend API URL
# For local development, use http://localhost:8000
# For production, set to your deployed API URL
NEXT_PUBLIC_PYTHON_API_URL=http://localhost:8000
```

Required Environment Variables for Deployment

- **DATABASE_URL**: PostgreSQL connection string (Prisma format)
- **NEXT_PUBLIC_PYTHON_API_URL**: Python backend API endpoint URL
- **Optional AWS S3 variables** (if using S3 for media storage):
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- AWS_REGION
- AWS_S3_BUCKET

Backend Environment Variables

Created `.env` (With Replicate Token)

```
# Replicate API Token for AI-powered skeleton detection
REPLICATE_API_TOKEN=r8_XVbSqNpDmahHdfRWDjmivN2ZNPk3MUH2w1N4x

# Server Configuration
HOST=0.0.0.0
PORT=8000

# CORS Configuration
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001,https://*.vercel.app

# MediaPipe Pose Detection
MEDIAPIPE_MODEL_COMPLEXITY=2
```

Created `.env.example` Template

```
# Replicate API Token for AI-powered skeleton detection
# Get your token from: https://replicate.com/account/api-tokens
REPLICATE_API_TOKEN=your_replicate_api_token_here

# Server Configuration
HOST=0.0.0.0
PORT=8000

# CORS Configuration
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001

# MediaPipe Pose Detection (0=Lite, 1=Full, 2=Heavy)
MEDIAPIPE_MODEL_COMPLEXITY=2
```

Required Environment Variables for Deployment

- **REPLICATE_API_TOKEN**: Replicate API token (REQUIRED)
- **HOST**: Server host (default: 0.0.0.0)
- **PORT**: Server port (default: 8000)
- **ALLOWED_ORIGINS**: Comma-separated CORS allowed origins
- **MEDIAPIPE_MODEL_COMPLEXITY**: Pose detection model complexity (0-2)

Frontend Configuration Files

`next.config.mjs`

```
/** @type {import('next').NextConfig} */
const nextConfig = {};

export default nextConfig;
```

- Currently minimal configuration
- May need to add API proxy, image domains, or other settings for production

`prisma/schema.prisma`

- Comprehensive database schema with 10 models

- Models: User, Account, Session, PlayerProfile, EliteShooter, Analysis, BiomechanicalMeasurement, Flaw, Report
- PostgreSQL database
- Extensive enums for categorization (Position, SkillLevel, BodyType, MediaType, AnalysisStatus, etc.)

`tailwind.config.ts`

- Tailwind CSS configuration with custom theme
- Animation configurations for UI components

`tsconfig.json`

- TypeScript configuration for strict type checking
- Path aliases configured for imports

Backend Configuration Files

`Dockerfile`

- Based on Python 3.11-slim
- Installs OpenCV system dependencies
- Exposes port 8000
- Runs with uvicorn

`run_dev.sh`

- Development server startup script
- Creates virtual environment
- Installs dependencies
- Runs with auto-reload

🔌 API Endpoints Exposed by Backend

Base URL

- **Local Development**: `http://localhost:8000`
- **Production**: TBD (to be deployed externally)

Endpoints

1. Health Check

GET `/health`

Response Model: `HealthResponse`

```json
{
  "status": "healthy",
  "version": "1.0.0",
  "mediapipe_available": true
}
```

Description: Health check endpoint to verify backend status and MediaPipe availability.

---

2. Analyze Image

POST `/analyze`

Request:
- **Type**: `multipart/form-data`
- **Field**: `file` (image file)

Response Model: `AnalysisResponse`

```
{
  "success": true,
  "keypoints": [
    {
      "name": "left_shoulder",
      "x": 0.45,
      "y": 0.32,
      "z": -0.15,
      "visibility": 0.95
    }
  ],
  "confidence": 0.89,
  "is_shooting_pose": true,
  "message": "Detected 33 keypoints"
}
```

Description: Analyzes an uploaded image for pose detection using MediaPipe. Returns detected keypoints with normalized coordinates (0-1), confidence scores, and shooting pose classification.

MediaPipe Keypoints (33 total):
- Face: nose, eyes (inner, center, outer), ears, mouth
- Upper Body: shoulders, elbows, wrists, hands (pinky, index, thumb)
- Lower Body: hips, knees, ankles, heels, feet (foot_index)

---

3. Export Annotated Image

POST `/export`

Request:
- **Type**: `multipart/form-data`
- **Field**: `file` (image file)
- **Query Parameters**:
- `skeleton_color` : Hex color (default: #FFFFFF)
- `joint_color` : Hex color (default: #FFFFFF)
- `label_color` : Hex color (default: #FFFFFF)
- `show_callouts` : Boolean (default: true)
- `output_format` : “png” | “jpeg” (default: “png”)
- `quality` : 1-100 (default: 95)

Response Model: `ExportResponse`

```json
{
  "success": true,
  "image_base64": "iVBORw0KGgoAAAANSUhEUg...",
  "content_type": "image/png",
  "message": "Generated annotated image with 33 keypoints"
}
```

Description: Exports an image with skeleton overlay annotation. Runs pose detection, draws skeleton lines and joint circles, adds callout labels (WRISTS, ELBOWS, SHOULDERS, CORE/ABS, HIPS, KNEES, ANKLES), and returns base64-encoded image.

---

4. AI Skeleton Overlay (Replicate)

POST `/ai-skeleton`

Request:
- **Type**: `multipart/form-data`
- **Field**: `file` (image file)

Response:

```json
{
  "success": true,
  "image_base64": "iVBORw0KGgoAAAANSUhEUg...",
  "content_type": "image/png",
  "message": "Detailed anatomical skeleton overlay generated successfully"
}
```

Description: Generates detailed anatomical skeleton overlay with medical illustration quality. Creates X-ray style skeleton with individual bones (excluding head, hands, feet), joints, and labeled callouts. Uses MediaPipe for pose detection and custom anatomical rendering.

Anatomical Components:
- **Spine**: Thoracic and lumbar vertebrae (17 vertebrae)
- **Ribcage**: 12 pairs of ribs + sternum (manubrium, body, xiphoid)
- **Shoulder Girdle**: Scapulae (shoulder blades) + clavicles (collar bones)
- **Pelvis**: Iliac crests, acetabulum (hip sockets), pubic symphysis, sacrum
- **Upper Limbs**: Humerus, radius, ulna (with visible separation)
- **Lower Limbs**: Femur, patella (kneecap), tibia, fibula (with visible separation)

🔗 Frontend-Backend Connection

API Client Service (`src/services/pythonBackendApi.ts`)

Configuration

```typescript
const PYTHON_API_URL = process.env.NEXT_PUBLIC_PYTHON_API_URL || 'http://localhost:
8000';
```

Key Functions

1. `checkBackendHealth()`

```
async function checkBackendHealth(): Promise<HealthResponse>
```

- Checks if Python backend is available
- Used for health monitoring

2. `analyzeImage(file: File)`

```
async function analyzeImage(file: File): Promise<AnalysisResponse>
```

- Sends image file to `/analyze` endpoint
- Returns pose keypoints and confidence

3. `exportAnnotatedImage(file, config, format, quality)`

```
async function exportAnnotatedImage(
  file: File,
  config: Partial<SkeletonConfig>,
  format: 'png' | 'jpeg',
  quality: number
): Promise<ExportResponse>
```

- Sends image with skeleton customization options
- Returns base64-encoded annotated image

4. Utility Functions

- `base64ToBlob()` : Converts base64 to Blob for download
- `downloadExportedImage()` : Downloads exported image to user's device

Data Flow

1. **User Upload** → MediaUpload component captures file
2. **Frontend Processing** → Optional client-side pose detection (TensorFlow.js or MediaPipe)
3. **Backend Analysis** → Sends to Python backend via `analyzeImage()`
4. **Pose Detection** → Python backend processes with MediaPipe
5. **Return Results** → Keypoints sent back to frontend
6. **Visualization** → SkeletonOverlay component renders pose
7. **Export** → User can export annotated image via `/export` endpoint

CORS Configuration

Backend allows these origins:
- `http://localhost:3000`
- `http://localhost:3001`
- `http://127.0.0.1:3000`
- `http://127.0.0.1:3001`
- `https://*.vercel.app` (wildcard for Vercel deployments)

Note: For Abacus AI deployment, the Abacus AI app URL must be added to `ALLOWED_ORIGINS` in the backend `.env` file.

🗄️ Database Schema Analysis (Prisma)

Database Provider

- **PostgreSQL** (configured via `DATABASE_URL`)

Core Models

1. User

- Authentication and profile management
- Relations: PlayerProfile, Analysis, Account, Session

2. PlayerProfile

- Physical attributes (height, weight, wingspan, age)
- Position, skill level, body type
- One-to-one with User

3. EliteShooter

- Reference database of professional shooters
- Shooting mechanics profile (release height/angle, joint angles)
- Career statistics (FG%, 3P%, FT%)
- Media references (images, videos)

4. Analysis

- Main analysis record
- Links to user, player info, media, measurements, report, flaws, matched elite shooter
- Status tracking (PENDING, PROCESSING, COMPLETED, FAILED)
- Report tier (BASIC, ULTRA, PREMIUM)

5. BiomechanicalMeasurement

- Joint angles: shoulder, elbow, hip, knee, ankle
- Heights: elbow, release, hip
- Release metrics: release angle, entry angle
- Raw pose data (JSON)

6. Flaw

- Identified shooting form flaws
- Category, severity, description
- Correction guidance and recommended drills

7. Report

- Generated analysis report
- Tiered content (executive summary, detailed analysis, corrections, training plan)
- PDF generation capability

8. Account & Session (NextAuth)

- NextAuth authentication models

Enums

- Position: POINT_GUARD, SHOOTING_GUARD, SMALL_FORWARD, POWER_FORWARD, CENTER
- SkillLevel: BEGINNER, INTERMEDIATE, ADVANCED, PROFESSIONAL

- BodyType: ECTOMORPH, MESOMORPH, ENDOMORPH, GUARD_BUILD, FORWARD_BUILD, CENTER_BUILD
- MediaType: IMAGE, VIDEO
- AnalysisStatus: PENDING, PROCESSING, COMPLETED, FAILED
- ReportTier: BASIC, ULTRA, PREMIUM
- FormCategory: OPTIMAL, GOOD, NEEDS_IMPROVEMENT, CRITICAL
- FlawCategory: Multiple categories for different form aspects
- FlawSeverity: MINOR, MODERATE, MAJOR, CRITICAL

🚀 Deployment Preparation

Backend Deployment Readiness ✅

Files Created/Updated

1. ✅ `.env.example` - Environment variable template
2. ✅ `.env` - Environment variables with Replicate token
3. ✅ `requirements.txt` - Complete with all dependencies (replicate + requests added)

Backend Features

- ✅ Health check endpoint
- ✅ CORS configuration for cross-origin requests
- ✅ Docker support (Dockerfile included)
- ✅ MediaPipe pose detection (CPU-based, works anywhere)
- ✅ Replicate API integration for AI skeleton overlay
- ✅ Error handling and validation
- ✅ Base64 image encoding for easy transport

Backend Deployment Options

1. **Docker Deployment** (Recommended)
   - Use provided `Dockerfile`
   - Set environment variables
   - Expose port 8000

2. **Traditional Server Deployment**
   - Use `run_dev.sh` or directly run with uvicorn
   - Install system dependencies for OpenCV
   - Configure reverse proxy (nginx/Apache)

3. **Cloud Platforms**
   - AWS EC2, ECS, Lambda (with container support)
   - Google Cloud Run, App Engine
   - Azure Container Instances, App Service
   - DigitalOcean App Platform
   - Railway, Render, Fly.io

Backend Environment Variables Checklist

- ✅ REPLICATE_API_TOKEN (configured)
- ✅ HOST (default: 0.0.0.0)

- ✅ PORT (default: 8000)
- ⚠️ ALLOWED_ORIGINS (must add Abacus AI frontend URL)
- ✅ MEDIAPIPE_MODEL_COMPLEXITY (default: 2)

Frontend Deployment Considerations

Required Updates for Abacus AI Deployment

1. **Database Configuration**
   - Replace local Prisma Postgres with Abacus AI PostgreSQL
   - Update `DATABASE_URL` environment variable
   - Run migrations: `npx prisma migrate deploy`

2. **Backend API URL**
   - Set `NEXT_PUBLIC_PYTHON_API_URL` to external Python backend URL
   - Update after backend deployment

3. **CORS Configuration**
   - Add Abacus AI frontend URL to backend `ALLOWED_ORIGINS`

4. **Authentication** (if using NextAuth)
   - Configure NextAuth secret: `NEXTAUTH_SECRET`
   - Set up OAuth providers (if needed)
   - Configure session strategy

5. **File Storage** (AWS S3)
   - Configure AWS credentials if using S3 for media uploads
   - Or use alternative storage solution

Frontend Dependencies Status

- ✅ All dependencies listed in package.json
- ✅ Next.js 14 with App Router
- ✅ Prisma ORM configured
- ✅ TypeScript configured
- ✅ Tailwind CSS configured

📝 Backend Functionality Documentation

Core Capabilities

1. Pose Detection Engine

- **Technology**: Google MediaPipe BlazePose
- **Model Complexity**: Configurable (0=Lite, 1=Full, 2=Heavy)
- **Keypoints**: 33 landmarks covering full body
- **Accuracy**: High confidence detection for upper body (shooting form)
- **Speed**: Real-time capable on CPU

2. Skeleton Overlay Rendering

- **Basic Skeleton**: Lines and joints with customizable colors
- **Callout Labels**: Anatomical labels (WRISTS, ELBOWS, etc.)
- **Customization**: Line thickness, joint radius, label visibility

- **Output**: Base64-encoded PNG/JPEG

3. Anatomical Skeleton (Advanced)

- **Detailed Rendering**: Individual bones with proper anatomy
- **Components**: Spine, ribcage, pelvis, limbs (no head/hands/feet)
- **Style**: X-ray/medical illustration quality
- **Technique**: Hollow outline drawing with anti-aliasing

4. Shooting Pose Classification

- **Logic**: Checks if either arm is raised (wrist above shoulder)
- **Threshold**: Minimum 30% visibility for keypoints
- **Output**: Boolean flag `is_shooting_pose`

Technical Implementation

MediaPipe Integration

```
# Pose Detection Configuration
mp_pose.Pose(
    static_image_mode=True,
    model_complexity=2,  # Most accurate
    enable_segmentation=False,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5
)
```

Skeleton Connections

- Torso: 4 connections (shoulder-to-shoulder, shoulder-to-hip, hip-to-hip)
- Arms: 4 connections per arm (shoulder-elbow-wrist)
- Legs: 4 connections per leg (hip-knee-ankle)
- Total: 12 main connections for shooting form analysis

Image Processing Pipeline

1. Receive uploaded image (multipart/form-data)
2. Decode with OpenCV (BGR format)
3. Convert to RGB for MediaPipe
4. Run pose detection
5. Extract 33 keypoints with confidence scores
6. Classify shooting pose
7. (Optional) Render skeleton overlay
8. Encode result as base64
9. Return JSON response

🔍 Key Observations & Recommendations

Strengths

1. ✅ Well-structured codebase with clear separation of concerns
2. ✅ Comprehensive database schema with Prisma ORM
3. ✅ Modern tech stack (Next.js 14, React 18, FastAPI)

4. ✅ Multiple pose detection options (MediaPipe, TensorFlow.js)
5. ✅ Detailed biomechanical analysis capabilities
6. ✅ Elite shooter comparison feature
7. ✅ Docker support for easy backend deployment

Areas Requiring Attention

Backend

1. ⚠️ **CORS Origins**: Must add Abacus AI frontend URL to `ALLOWED_ORIGINS`
2. ⚠️ **Environment Variables**: Currently hardcoded Replicate token in `main.py` (line 20) - should use env var
3. 💡 **Recommendation**: Update `main.py` to use `os.getenv("REPLICATE_API_TOKEN")`

Frontend

1. ⚠️ **Database URL**: Must update to Abacus AI PostgreSQL
2. ⚠️ **Backend API URL**: Must set after backend deployment
3. ⚠️ **S3 Configuration**: May need AWS credentials for media storage
4. 💡 **Recommendation**: Consider using Abacus AI's storage solutions instead of S3

Security

1. ⚠️ **Replicate Token Exposure**: Currently in code, should be environment-only
2. 💡 **Recommendation**: Never commit `.env` file with real credentials

Deployment Sequence Recommendation

1. **Deploy Python Backend First**
   - Deploy to external hosting (Railway, Render, AWS, etc.)
   - Set all environment variables
   - Note deployed URL

2. **Update Backend CORS**
   - Add Abacus AI app URL to `ALLOWED_ORIGINS`
   - Redeploy backend

3. **Configure Frontend Environment**
   - Set `DATABASE_URL` to Abacus AI PostgreSQL
   - Set `NEXT_PUBLIC_PYTHON_API_URL` to backend URL
   - Configure any other required variables

4. **Run Database Migrations**
   - `npx prisma generate`
   - `npx prisma migrate deploy`

5. **Deploy Frontend to Abacus AI**
   - Deploy Next.js app to Abacus AI
   - Verify database connection
   - Test backend connectivity

6. **Test End-to-End**
   - Upload test image
   - Verify pose detection
   - Check skeleton overlay
   - Validate database writes

🔗 External Dependencies & APIs

Required External Services

1. Replicate API

- **Purpose**: AI-powered skeleton detection (used in `/ai-skeleton` endpoint)
- **Token**: `r8_XVbSqNpDmahHdfRWDjmivN2ZNPk3MUH2w1N4x` (provided)
- **Documentation**: https://replicate.com/docs
- **Usage**: Model inference for enhanced skeleton overlay

2. PostgreSQL Database

- **Current**: Prisma Postgres (local development)
- **Production**: Abacus AI PostgreSQL
- **Schema**: Comprehensive schema with 10 models
- **Migrations**: Required on deployment

3. (Optional) AWS S3

- **Purpose**: Media file storage
- **SDK**: @aws-sdk/client-s3 included in dependencies
- **Alternative**: Use Abacus AI storage or other cloud storage

Python Libraries with Native Dependencies

OpenCV (opencv-python)

- **System Packages Required**:
- libgl1-mesa-glx
- libglib2.0-0
- libsm6
- libxext6
- libxrender-dev
- **Note**: Dockerfile includes these dependencies

MediaPipe (mediapipe)

- **Platform**: Cross-platform (Linux, macOS, Windows)
- **Architecture**: x86_64, ARM64
- **Note**: CPU-based, no GPU required

📊 Application Features Overview

Core Features

1. Media Upload

- Drag-and-drop interface
- Support for images and videos
- Player profile form integration

2. Pose Detection

- Client-side detection (TensorFlow.js, MediaPipe)

- Server-side detection (Python backend)
- 33 keypoint detection with confidence scores

3. Biomechanical Analysis

- Joint angle calculations
- Release height and angle measurement
- Staged form analysis (setup, dip, release, follow-through)
- Form scoring and categorization

4. Elite Shooter Comparison

- Database of professional shooters
- Similarity matching based on physical attributes and mechanics
- Visual comparison of shooting forms

5. Flaw Detection & Correction

- Automated identification of form flaws
- Severity classification (MINOR to CRITICAL)
- Correction guidance and drill recommendations

6. Report Generation

- Tiered reports (BASIC, ULTRA, PREMIUM)
- Executive summary, detailed analysis, training plans
- PDF export capability

7. Skeleton Overlay Visualization

- Customizable skeleton overlay
- Anatomical labels and callouts
- Export with transparent background option

8. Interactive Dashboard

- Real-time analysis progress
- Form score cards
- Overlay controls for visualization

🎯 Next Steps for Deployment

Immediate Actions

1. **Backend Deployment**
   - [ ] Choose hosting platform (Railway, Render, AWS, etc.)
   - [ ] Deploy Docker container or Python app
   - [ ] Configure environment variables
   - [ ] Test all endpoints
   - [ ] Note deployed URL

2. **Backend CORS Update**
   - [ ] Add Abacus AI frontend URL to `ALLOWED_ORIGINS`
   - [ ] Redeploy backend

3. **Code Cleanup (Recommended)**
   - [ ] Remove hardcoded Replicate token from `main.py`
   - [ ] Update to use `os.getenv("REPLICATE_API_TOKEN")`

4. **Frontend Configuration**
   - [ ] Create Abacus AI PostgreSQL database
   - [ ] Update `DATABASE_URL` environment variable
   - [ ] Set `NEXT_PUBLIC_PYTHON_API_URL` to backend URL
   - [ ] Run Prisma migrations

5. **Frontend Deployment**
   - [ ] Deploy to Abacus AI
   - [ ] Verify database connection
   - [ ] Test backend connectivity
   - [ ] Perform end-to-end testing

6. **Testing & Validation**
   - [ ] Upload test images
   - [ ] Verify pose detection
   - [ ] Check database writes
   - [ ] Test report generation
   - [ ] Validate elite shooter comparison

📞 Support & Resources

Documentation Links

- **Next.js**: https://nextjs.org/docs
- **Prisma**: https://www.prisma.io/docs
- **FastAPI**: https://fastapi.tiangolo.com
- **MediaPipe**: https://google.github.io/mediapipe
- **Replicate**: https://replicate.com/docs

Repository

- **GitHub**: https://github.com/baller70/BasketballAnalysisAssessmentApp
- **Owner**: baller70
- **Access**: Private repository (admin access granted)

📄 Conclusion

The Basketball Analysis Assessment App is a sophisticated application combining modern web technologies with advanced computer vision for basketball shooting form analysis. The codebase is well-structured, and with the preparations documented in this analysis, the application is ready for external deployment.

Key Takeaways:
- ✅ Python backend is deployment-ready with all dependencies documented
- ✅ Environment configuration files created (.env.example, .env)

- ✅ Comprehensive API documentation provided
- ✅ Database schema thoroughly documented
- ⚠️ Minor code cleanup recommended (remove hardcoded token)
- 🎯 Follow the deployment sequence for smooth rollout

The application maintains a clean separation between frontend and backend, making independent deployment straightforward. The Python backend can be hosted on any platform supporting Docker or Python 3.11+, and the Next.js frontend can be deployed on Abacus AI with PostgreSQL integration.

Document Version: 1.0
Last Updated: December 4, 2025
Prepared By: DeepAgent - Abacus.AI