# Project Specification: Odds Arbitrage Scanner (arb-scanner)

Objective:  
Build a full-stack web application that scans live bookmaker odds and identifies arbitrage opportunities across multiple sports and bookmakers.

## 1. Project Overview

The goal is to create a web-based arbitrage scanner similar in concept to OddsJam. It must pull live odds via APIs (starting with The Odds API), calculate and display arbitrage opportunities, support mock data mode for local testing, and provide a modular codebase that can be expanded with new providers and alert systems later.

## 2. Tech Stack

| Layer | Stack |
| --- | --- |
| Frontend | React + Vite + TypeScript |
| Backend | Node.js + Express + TypeScript |
| Testing | Jest (unit tests for arb math) |
| Deployment | Vercel (frontend), Render/Fly.io (backend) |
| DevOps | Docker + docker-compose |
| Database (future phase) | SQLite via Prisma |

## 3. Core Functional Requirements

* A. Data Sources & API Integration

Integrate with The Odds API to fetch live odds for soccer, basketball, and tennis. Use `/sports/{sport\_key}/odds` endpoint and normalize bookmaker odds. Cache responses for 30–60 seconds and provide a mock mode for testing.

* B. Arbitrage Engine

Compute arbitrage opportunities for 2-way and 3-way markets. Include equal-profit stake splits and Kelly-fraction calculations. Reject markets with no arbitrage. Include unit tests for math verification.

* C. API Endpoints

Endpoints include:  
- GET /api/odds: returns list of arbitrage opportunities  
- GET /healthz: health check endpoint

* D. Frontend Features

Dashboard: displays arb opportunities with filters and refresh.  
Settings: local storage of API key.  
Responsive design with minimal UI.

## 4. Architecture Overview

client (React) ↕ backend (Express) ↳ provider interface ↳ arb math ↳ in-memory cache

## 5. Mock Data Mode

Mock JSON data used when MOCK\_ODDS=true to simulate API responses, allowing UI and logic testing without live data.

## 6. Testing

Jest test suite verifies correct arb detection and calculations. Includes both positive and negative test cases.

## 7. Deployment Requirements

Dockerized containers for backend and frontend. Use docker-compose for local development. Frontend on port 5173 and backend on 8080. Environment variables managed via .env.example.

## 8. Future Enhancements

1. Persistence via SQLite.  
2. Telegram bot alerts for arbitrage thresholds.  
3. Multi-provider odds support.  
4. Optional authentication.

## 9. Replit AI Agent Workflow

Replit AI can generate, edit, and run code in-browser. The agent can scaffold Node + React projects, install dependencies, and perform build/test operations automatically. Complex API logic and fine-tuning may be offloaded to VS Code or ChatGPT.

## 10. Deliverables Checklist

• Full repo matching the agreed folder structure  
• Working local build with mock mode  
• Tested API integration with live key  
• Passing Jest tests  
• Comprehensive README and Docker support