**Big-Picture Take**

You’re tackling a **dream project** for an FRC data nerd: one tool that ingests scouting sheets, live TBA stats and GPT insight—then guides the whole draft process end-to-end. 100 % doable, but there are a few dragons you should plan for up front.

**🚦 High-Leverage Wins**

| **Idea** | **Why it Pays Off** |
| --- | --- |
| **Phase the build** (Setup → Cleanup → Pick-List → Live) | You always have a usable tool, even if later modules slip. |
| **Mock external APIs early** | A tiny JSON file that looks like TBA output lets you code offline, test Git-commits, and avoid rate-limits. |
| **Schema JSON** (we just added) | Removes “new-game-year scramble” from your future to-do list. |
| **Unit-test the analytics** | Simple pytest over synergy.py prevents silent math bugs that GPT might introduce. |
| **Add verbose logging** to every API wrapper | When something mis-behaves live, the log tells you whether it’s Sheets auth, TBA 502, or GPT timeout. |

**🐉 Common Dragons When “GPT Writes the Code”**

| **Dragon** | **Symptom** | **Mitigation** |
| --- | --- | --- |
| **1. Version drift** | GPT pastes FastAPI 0.96 code but you install 0.110. | Pin versions in requirements.txt; copy exact lines GPT gives (fastapi==0.110.0). |
| **2. Undeclared imports** | Code block compiles for GPT but missing pydantic import locally. | After every new file, run python -m pip check and let GPT fix errors before moving on. |
| **3. Circular prompts** | You ask for “one more tweak” and GPT rewrites half the file, breaking earlier parts. | Ask GPT to update *only* a specific function or supply a diff patch. |
| **4. Silent API failures** | Google or TBA changes response format and your JSON key path breaks during an event. | Wrap every external call in try/except that logs the entire response on error. |
| **5. Latency during finals** | GPT call hangs for 6 s and the UI freezes. | Use threading/async + a spinner; display cached last-good pick list if GPT is slow. |
| **6. Windows path pain** | GPT examples use POSIX paths (/home/user). | Remind GPT you’re on Windows; use os.path.join() everywhere. |
| **7. Secret leakage** | You accidentally paste service-account JSON into chat. | Keep secrets in .env only; never paste raw keys in the GPT window. |

**🔧 Real-World Advice Before You Start**

1. **Create a tiny “hello world” FastAPI + React repo yourself** (even copy/paste).  
   *Why*: Gives you mental scaffolding so GPT’s file paths make sense.
2. **Host code on GitHub** (private repo).  
   *Why*: Snapshots every change GPT makes; you can roll back.
3. **Use pre-commit hooks** for Black/flake8.  
   *Why*: GPT sometimes returns mixed spacing—auto-format keeps sanity.
4. **Budget GPT tokens**:
   * Schema detect: < $0.01 (once per event)
   * Pick-list refresh: ~$0.05 per alliance selection run
   * Live advice: maybe $0.10 total per event  
     *Keep a usage log so nothing surprises you.*
5. **Expect UI polish to take time**. GPT will spit out functional React, but aligning Tailwind classes and responsive layouts is still iterative—budget a few hours to click around and adjust.
6. **Have a fallback plan** for no-internet arenas:
   * Cache TBA schedule JSON locally.
   * Cache GPT’s last run pick list to CSV as you described.
   * Wrap every GPT call with “if offline → load cached suggestion.”

**🚀 Green Lights**

* You’re using **FastAPI + React** ⇒ industry-standard, tons of docs.
* Google Sheets + service account ⇒ once it works, it never asks again.
* GPT-generated logic ⇒ perfect for “explain anomalies” where deterministic code is a pain.

With phased milestones, strict version pins, and defensive logging, you’ll sail through. And if anything feels shaky mid-build, you can always ask GPT: “give me unit tests for detect\_anomalies()” before trusting it live.

Ready to unleash Raptor-Stack AI? 🚀