**Digital Harvest Simulator - Phase 2 Project Plan**

**Notes:**

Database schema is digital\_harvest, pw is Hecther.  
  
/digital-harvest/

|

|-- /frontend/

|

|-- index.html <-- The React application lives here.

|

|-- /src/

| |-- api.py <-- The Flask backend server.

| |-- digital\_harvest.py <-- The business app itself

| |-- setup\_business.py <-- The setup / clear database app

|

|-- /data/

| |-- Dashboard.pbix <-- The ordering dashboard

|

|-- /screenshots/

| |-- dashboard images

|

|-- Project\_Plan.docx <-- Planning document (this one)

|-- README.md

|-- requirements.txt

|-- .gitignore

This playthrough

* I started to play by buying some of each switch just to start. They all sold fairly quickly. I advanced the time 7 days and all but 1 switch sold out in the first 3 days.
* I bought as many as I could of the cheap but solid selling Gateron Yellow Switches. I advanced 7 days.
* I bought almost as many as I could of the Zealios V2 Tactile Switches, and few “Aurouras”. I advanced the time 7 days.
* I bought as many as I could of the “aruroras” still having a decent amount of the other 2 styles left. I made the price a hard $6.50 on this switch as it’s a house specialty, but the other 2 switches started to receive discounts. I advanced the time 7 days.
* I got to a point I could keep all 3 in stock advancing 7 days at a time and spending everything.

This plan outlines the development path for transforming the Digital Harvest simulator from a command-line tool into a full-stack portfolio project with a web-based frontend.

**Phase 1: Foundational Rearchitecture (You Are Here)**

The core goal of this phase is to build a robust, scalable backend and database that can support complex business logic and analytics.

* [x] **Identify Limitations:** Recognize that the initial simple schema makes advanced analytics (like inventory over time) difficult.
* [x] **Design Accounting-Focused Schema:** Re-design the database to use a proper ledger system.
  + Create tables for categories, products, and a central inventory\_ledger.
  + The inventory\_ledger is the key: it will track every stock movement and store the resulting quantity\_on\_hand, eliminating the need for complex DAX calculations.
* [x] **Refactor Python Backend:**
  + Update setup\_business.py to create the new tables and relationships.
  + Rewrite business\_simulator.py to use the new ledger system for all inventory operations (purchases, sales).
  + Add logic to simulate and record general business expenses (rent, utilities, etc.).
* [x] **Validate with Power BI:** Connect Power BI to the new schema and confirm that building dashboards (sales, inventory over time) is now simple and intuitive.

**Phase 2: Web Frontend Development**

The goal is to create an intuitive user interface that makes "running" the business straightforward and engaging. Based on your experience and modern web standards, **React** is an excellent choice.

* **Technology Stack:**
  + **Frontend:** React (using Create React App or Vite for setup).
  + **Styling:** Tailwind CSS for rapid, modern UI development.
  + **Backend API:** Python (using a framework like Flask or FastAPI) to serve data from the MySQL database.
* **Key Features:**
  + **Dashboard View:** A main dashboard showing key metrics (cash, revenue, profit) and visualizations (sales charts). This will replace the Power BI dashboard for the live application.
  + **Inventory Management Page:** A view to see current stock levels, average costs, and initiate new purchase orders.
  + **Pricing Control Page:** An interface to update the sale price for each product.
  + **"End Day" Button:** A button that replaces the [A]dvance Time command, running the simulation for one or more days and updating the UI.

**Phase 3: Advanced Features & Scalability**

Once the core application is functional, expand its capabilities to simulate a more complex business environment.

* **Employee Management:**
  + Add an employees table to the database.
  + Simulate payroll as a recurring expense.
  + Potentially link employees to sales to track performance.
* **Advanced Reporting:**
  + Build dedicated dashboard tabs for different business perspectives (Financials, Sales, Inventory).
  + Generate classic accounting reports like an Income Statement or Balance Sheet.
* **Market Dynamics:**
  + Introduce more complex AI logic, such as seasonal demand fluctuations or competitor price changes that affect your sales.
* **User Authentication:**
  + Allow users to create accounts and run their own separate business simulations.

Old Plan:  
**AI Business Simulator: Project Plan (Phase 1)**

This document outlines a phased approach to building the AI-powered business simulator. The goal is to start with a simple, functional core and iteratively add complexity.

**Phase 1: The Core Engine (Minimum Viable Product)**

**Objective:** Build the fundamental mechanics of the business without any complex AI.

1. **Data Models:**
   * Product: id, name, purchase\_cost, selling\_price, stock\_quantity.
   * Ledger: A simple log of all transactions (e.g., date, transaction\_type ('Purchase' or 'Sale'), product\_id, quantity, amount).
   * BusinessState: A central object to hold the current date, cash\_on\_hand, etc.
2. **Core Logic:**
   * **Time Progression:** A main loop that advances the simulation day by day.
   * **Inventory Management:** Functions to buy\_stock (decreases cash, increases stock\_quantity) and sell\_stock (increases cash, decreases stock\_quantity).
   * **Financials:** Basic tracking of cash flow, assets (value of inventory), and profit/loss.
3. **Interface:**
   * Start with a simple Command-Line Interface (CLI). The user can input commands like buy 10 product\_A or advance 30\_days.

**Output:** A program that can simulate the basic financial and inventory flow of a business. Data is exported to CSV files (products.csv, ledger.csv).

**Phase 2: The AI Sales Oracle**

**Objective:** Replace manual or simple random sales with an AI-driven model that simulates a realistic market.

1. **Simple Sales Model:**
   * Create a function generate\_daily\_sales().
   * Initial logic: Sales probability is based on selling\_price (lower is better) and stock\_quantity (must be > 0). Add a random "hot item" factor.
2. **Introducing Historical Data & Trends:**
   * Create a baseline sales history (e.g., a CSV with date and sales\_volume\_modifier). This can simulate seasonality (e.g., higher sales in December).
   * The AI model now considers the base trend for a given day.
3. **Competitive Analysis (Advanced):**
   * The AI will simulate competitor prices. You can start with a simple rule: competitor\_price = product.purchase\_cost \* 1.5.
   * The sales generation logic will now heavily penalize your sales if your\_selling\_price is significantly higher than the competitor\_price. This simulates price sensitivity in the market.

**Output:** The simulation generates dynamic sales that react to your pricing and inventory decisions.

**Phase 3: Analytics & Visualization (Power BI)**

**Objective:** Create a feedback loop by visualizing the simulated data to inform your decisions.

1. **Data Pipeline:**
   * Ensure your Python script outputs clean CSV files at the end of each simulation run.
2. **Power BI Dashboards:**
   * **Sales Dashboard:** Sales over time, sales by product, average selling price.
   * **Inventory Dashboard:** Current stock levels, inventory value, stock turnover rate.
   * **Financial Dashboard:** Cash flow statement (daily/weekly/monthly), Profit & Loss by product, overall business profitability.
3. **Decision Making:**
   * Run the simulation for a "year".
   * Analyze the Power BI dashboards.
   * Formulate a new strategy (e.g., "I need to keep less stock of Product B and lower the price of Product A in Q4").
   * Reset the simulation and run it again with your new strategy to see if results improve.

**Output:** A complete cycle of simulation -> data output -> analysis -> strategic decision.

**Phase 4: Bridging to Reality**

**Objective:** Enhance the simulation with real-world elements and prepare for a live business.

1. **Real-World Data:**
   * Write scripts to scrape or use APIs to pull real-world pricing for similar items to feed your "Competitive Analysis" model.
   * Incorporate real shipping costs and lead times for inventory purchases.
2. **"Real Money" Mode:**
   * Create a feature where you can switch from "fake money" to tracking a real budget. When you buy\_stock, you log a real purchase you made.
   * The AI continues to generate sales, but now you are managing real inventory against a simulated market, validating if your purchasing decisions would have been profitable.

**Output:** A fine-tuned business model and operational plan, ready for a live launch.