**Day 1: Project Setup and Research**

* **Project Setup**:
  + Create a **GitHub/GitLab repository** for version control.
  + Set up a **Python virtual environment** and install dependencies (ccxt, Flask/Django, SQLAlchemy, Pandas, etc.).
  + Set up the **database schema** (MySQL, PostgreSQL, etc.) for storing arbitrage opportunities, trades, and user settings.
  + Research **ccxt** documentation and identify the key exchanges you'll work with (Binance, Bybit, etc.).
* **Exchange API Exploration**:
  + Review the API documentation for exchanges (Binance, Bybit, etc.) to understand endpoints for market data and trading.

**Day 2: Database Schema & Backend Design**

* **Database Setup**:
  + Design tables: arbitrage\_opportunities, trade\_logs, user\_settings, and any other needed tables.
  + Set up **database connections** in the backend using SQLAlchemy (or any ORM you prefer).
  + Create **basic data models** and migrations for the database.
* **Backend Architecture**:
  + Decide on the structure of the backend (Flask or Django).
  + Set up a basic backend structure with essential routes (API for arbitrage data, user settings, etc.).

**Day 3-4: Implement ccxt Integration**

* **Connect to Exchanges**:
  + Set up ccxt for connecting to the selected exchanges (e.g., Binance, Bybit, Kraken, etc.).
  + Implement functions to fetch live price data (ask/bid prices) for multiple coin pairs (BTC/USDT, ETH/USDT, etc.).
* **Arbitrage Detection Logic**:
  + Develop a function to calculate price differences between exchanges (arbitrage opportunities).
  + Ensure the logic identifies profitable opportunities based on price discrepancies and margin calculations.

**Day 5-6: Store Arbitrage Opportunities in Database**

* **Store Detected Opportunities**:
  + After detecting an arbitrage opportunity, store it in the database (arbitrage\_opportunities table).
  + Include relevant data such as exchange names, coin pairs, price differences, margin, and timestamp.
* **Create Opportunity Tracker**:
  + Implement an **automatic refresh system** to fetch new data every second (or user-defined refresh interval).

**Day 7-8: Frontend Setup - UI Framework**

* **Frontend Structure**:
  + Set up the basic **frontend layout** using HTML, CSS, and **Bootstrap**.
  + Create a **live tracking dashboard** to display arbitrage opportunities (real-time updates).
  + Set up placeholders for key sections like "Active Opportunities", "Trade Logs", "Profit Breakdown", etc.
* **Display Arbitrage Opportunities**:
  + Display live opportunities (with exchange names, coin pairs, price differences, etc.).
  + Ensure the UI dynamically updates as new opportunities are detected.

**Day 9-10: Implement Trade Execution Logic**

* **Auto-Trade Execution Logic**:
  + Write functions to **buy** on the lower-priced exchange and **sell** on the higher-priced exchange.
  + Use **ccxt order creation functions** (create\_market\_buy\_order, create\_market\_sell\_order) to initiate the trades.
* **Fees Calculation**:
  + Implement logic to calculate all relevant fees (trading fees, withdrawal fees) for each exchange.
  + Display a fee breakdown and margin before executing any trades.

**Day 11-12: Real-Time Trade Execution and Performance Logging**

* **Trade Execution**:
  + Implement real-time **trade execution** once an arbitrage opportunity is detected.
  + Calculate the **net profit** after considering fees and expenses.
* **Log Trades**:
  + Save details of executed trades in the **trade\_logs** table.
  + Include details such as the buy/sell price, fees, profit, timestamps, etc.

**Day 13-14: Backend Integration of AI for Optimization**

* **Data Collection for AI**:
  + Collect historical data on price movements and arbitrage opportunities.
  + Clean and preprocess the data using **Pandas**.
* **Train AI Model**:
  + Use a simple machine learning model (e.g., **Random Forest** or **Logistic Regression**) to predict profitable arbitrage opportunities.
  + Train the model using features like price discrepancies, volume, exchange fees, etc.

**Day 15-16: AI Integration for Arbitrage Prediction**

* **Integrate AI Model**:
  + Integrate the trained AI model into the backend to predict optimal arbitrage opportunities.
  + Let the AI model suggest **which coin pairs and exchanges to track** and **which opportunities are most likely to be profitable**.
* **Optimize Opportunity Detection**:
  + Adjust the detection logic to account for AI-based suggestions and improve prediction accuracy.

**Day 17-18: User Interface Enhancements**

* **Opportunity Tracking Interface**:
  + Improve the **live tracking interface** to show detailed information on each opportunity.
  + Add **real-time updates** for active opportunities.
  + Display a **profit breakdown** including fees and estimated net profit after executing the trade.
* **Trade History**:
  + Add a section to the dashboard where users can see the **history of executed trades** with profit/loss details.
  + Show a **summary** of total profit, successful trades, failed trades, etc.

**Day 19-20: User Settings and Customization**

* **User Settings**:
  + Implement settings where users can:
    - Select exchanges to monitor.
    - Choose coin pairs to track.
    - Set a minimum **profit threshold** for trade execution.
    - Set the **refresh interval** for checking arbitrage opportunities.
* **Backend Integration for Settings**:
  + Ensure that user preferences are saved to the **user\_settings** table in the database.
  + Implement logic to use these settings in the backend to control the arbitrage detection.

**Day 21-22: Security Implementation**

* **Secure API Keys**:
  + Store API keys securely using **environment variables** or encrypted storage.
  + Implement authentication (OAuth or API keys) for connecting to exchanges.
* **Encryption**:
  + Ensure that all sensitive information (user data, API keys) is encrypted both in transit and at rest.

**Day 23-24: Testing Trade Execution and User Interface**

* **Test Trade Execution**:
  + Perform **manual trades** and test the auto-execution logic to ensure it works correctly under real market conditions.
  + Verify that the **fee calculation** and profit margins are accurate after a trade is executed.
* **UI Testing**:
  + Test the **frontend interface** for responsiveness and usability.
  + Ensure that live tracking works seamlessly and updates in real-time.

**Day 25-26: Finalize AI and Optimizations**

* **AI Enhancements**:
  + Fine-tune the AI model to improve its accuracy and profitability predictions.
  + Use **historical data** to continuously improve the model's performance over time.
* **System Optimization**:
  + Optimize the backend to handle **high-frequency trading** and real-time data processing without lag.
  + Optimize database queries for quick retrieval of arbitrage opportunities.

**Day 27-28: Deployment Preparation**

* **Prepare for Deployment**:
  + Set up a production environment (AWS, Heroku, or any other cloud provider).
  + Ensure that the backend, database, and frontend are all ready for deployment.
* **Test Deployment**:
  + Test the entire application in a staging environment to ensure everything works as expected.

**Day 29-30: Final Testing, Debugging, and Launch**

* **Final System Testing**:
  + Test all components together: frontend, backend, AI integration, and real-time trade execution.
  + Check for **bugs**, **performance issues**, or any missing features.
* **Launch**:
  + Deploy the application to production.
  + Monitor the live application for any issues and optimize based on user feedback.

**Key Considerations:**

* **Time Management**: The plan is aggressive and will require focused effort. Be mindful of task priorities.
* **Testing**: Thoroughly test each component, especially trade execution and the AI model, to ensure that no trades are executed incorrectly.
* **AI Improvements**: Start with a simple model and iteratively improve it. The AI optimization doesn't need to be perfect initially, but it should get better over time.
* **Security**: Ensure your app is secure, especially when handling API keys and trade execution.