Scope of Work: AI Investment Bot

Requirements

- **Infrastructure Setup:** Provision and configure cloud resources (database, environments), establish CI/CD pipeline, and implement security measures.
- **Backend Refactoring:** Modularize the Flask application, implement a service layer and design patterns, normalize database models, and add caching.
- **API Development:** Design and build a comprehensive REST API for the frontend.
- **Frontend Development:** Initialize and build the React application with reusable components, and rewrite key UI elements.
- **Authentication:** Implement password reset, email verification, and social login features.
- **LLM Integration (Chat):** Develop backend systems for dynamic prompt construction and chat session management, and build the React chat component.
- **Deep Learning:** Add model training and deployment support.

Scope and Timelines

This section outlines the scope of work required to refactor the AI Investment Agent application into a more robust, maintainable, and scalable system. The tasks cover project cleanup, backend refactoring, database normalization, front-end migration, and DevOps.

Task Name	Task Description	Man-day Estimate
Phase 1: Project & DevOps Setup		
Project Cleanup & Standardization	Consolidate to a single virtual environment. Remove developer-specific and temporary files (e.g., `analysis-DESKTOP-*.html`). Update `.gitignore` to prevent future clutter.	2

Setup Git Repository	Initialize a new Git repository on a platform like GitHub or GitLab. Ensure the main branch is protected and establish a clear branching strategy (e.g., GitFlow).	0.5
Setup Cloud Database	Provision a managed PostgreSQL instance on a cloud provider (e.g., AWS RDS, Google Cloud SQL). Configure security groups, user access, and automated backups.	1
Setup Cloud Infrastructure & CI/CD	Set up staging and production environments on a cloud provider (e.g., AWS, GCP). Create a CI/CD pipeline (e.g., using GitHub Actions) to automate testing and deployments.	4
Phase 2: Backend Refactoring		
Refactor \`app.py\` to Blueprints	Decompose the monolithic \`src/app.py\` file. Group related routes into separate Flask Blueprint files (e.g., \`auth.py\`, \`portfolio.py\`, \`analysis.py\`) to improve modularity and maintainability.	3
Introduce a Service Layer	Abstract business logic out of the route handlers into a dedicated service layer. This will decouple the core logic from the web framework, making it more reusable and easier to test independently.	2
Refactor Data Fetching Logic	Replace the "spaghetti code" in \`EnhancedStockAnalyzer\` with a Strategy Pattern. Each data source (Alpha Vantage, yfinance) will become a separate, interchangeable "strategy".	3

Normalize Database Models	Refactor models that heavily use JSON fields (`Portfolio`, `StockAnalysis`, `InvestmentDecision`). Create new relational tables to store this data properly, enforcing data integrity and improving query performance.	6
Refactor `StockPreference` Model	Split the ambiguous `StockPreference` model into smaller, single-responsibility models (`UserInteractionLog`, `UserFeedback`) to clarify the data architecture.	1
Implement Caching Layer	Replace the manual, inefficient caching with a dedicated library like `Flask-Caching`. Apply caching decorators to data-intensive functions to improve performance.	1
Phase 3: Frontend Migration (to React)		
Build Backend API for Frontend	Develop a comprehensive REST API on the Flask backend. These endpoints will provide all the necessary data for the new React front-end, effectively decoupling the backend from the frontend.	5
Setup React Environment	Initialize a new React project using a modern build tool like Vite. Configure the development environment, including setting up proxying to the Flask API.	1
Create Reusable Component Library	Build a library of reusable React components (e.g., buttons, cards, tables, charts) based on the existing Bootstrap 5 design. This ensures a consistent UI and speeds up page development.	3

Migrate Key Pages to React	Rewrite the most interactive pages (e.g., Stock Analysis, Portfolio Details, User Profiling) as React components. This is an iterative process, starting with the most critical views.	10
Total Estimated Effort		**42.5**

New Feature Development (Post-Refactoring)

This table outlines new features to be built once the core application has been refactored and stabilized. These estimates assume the work is being done on the new, improved architecture.

Task Name	**Task Description**	**Man-day Estimate**
Feature: Complete User Onboarding		
Implement Password Reset	Create a secure "Forgot Password" flow, where users can request a password reset link via their registered email. Includes email service integration and token-based security.	2
Add Email Verification	On signup, send a verification link to the user's email. The user must click this link to activate their account, ensuring the email address is valid.	1.5
Implement Social Logins	Allow users to sign up and log in using third-party providers like Google. This involves implementing OAuth 2.0 flows and linking social profiles to user accounts.	3

Task Name	**Task Description**	**Man-day Estimate**
Feature: Context-Aware AI Chat		
Design Context-Aware Prompting	Build a backend system to dynamically construct detailed prompts for the LLM. The prompts will include the user's portfolio, risk profile, recent analyses, and conversation history for context-rich answers.	4
Build Chat Backend Service	Create a dedicated backend service to manage chat sessions, store message history efficiently, and handle the stateful interaction with the LLM API.	3
Develop React Chat Component	Build a feature-rich chat component in React. This will support real-time streaming responses, rendering formatted data (like tables or charts) from the LLM, and suggesting follow-up questions.	5
Feature: Activate Deep Learning Models		
Data Ingestion & Cleaning Pipeline	Build a data pipeline to gather historical data for a universe of stocks. This involves fetching data from APIs (e.g., yfinance), cleaning it (handling missing values, splits, dividends), and storing it in a data lake (e.g., S3).	5
Feature Engineering Pipeline	Develop a process to transform the raw historical data into the features required by the models in `deep_learning.py`. This includes calculating technical indicators, volatility, etc., and creating the final training datasets.	3

Task Name	**Task Description**	**Man-day Estimate**
Model Training & Experimentation	Create a structured script for training the models. This includes experiment tracking (e.g., with MLflow), hyperparameter tuning, and versioning both the code and the resulting model artifacts.	4
Model Evaluation Framework	Build a framework to rigorously evaluate trained models against a held-out test set. This includes defining key performance metrics (e.g., MSE, MAE, R-squared) and creating visualizations to analyze model performance.	3
Provision MLOps Infrastructure	Set up the necessary cloud infrastructure for the ML lifecycle. This includes a data lake (S3), a compute instance (EC2) for running pipelines, and IAM roles for secure access.	2
Deploy Model as API Service	Deploy the best performing model as a scalable API endpoint for real-time predictions. This could be a custom FastAPI service or a managed platform like AWS SageMaker.	3
Integrate DL Models into App	Modify the application backend to call the new model serving API. Integrate the model's predictions and confidence scores into the stock analysis pages on the React front-end.	4
Total Estimated Effort		**42.5**