Scan 2 Invest Application High-Level Design

Note:
This high-level design provides a potential approach for developing and deploying an application that allows users to capture images of products, and invest in the company that owns that product. Some aspects of this design may not be implemented due to time constraints.

1. User Interface (Front-End) - Angular

- Image Capture Module: Allows users to take or upload a picture of a product.
- Information Display Module: Displays company and investment information retrieved from the back-end.
- Investment Module: Provides options and forms to facilitate investment actions.

2. API Server (Back-End) - Flask

- Image Processing and Recognition Service: Handles image pre-processing and recognition using OpenCV and Google Cloud Vision API.
- Data Retrieval Service: Fetches company and investment data using financial data APIs.
- Investment Service: Manages investment actions through some investment platform APIs
- Authentication Service: Manages user authentication and authorization.

3. Azure App Service (for Hosting the Front-End and Back-End)

Front-End Deployment:

- Build: Compile the Angular project to generate assets.
- Deploy: Use Azure DevOps, GitHub Actions, or Azure CLI to deploy the built assets to an Azure App Service instance.

Back-End (Flask) Deployment:

- Containerize: Optionally, containerize the Flask application using Docker for better portability.
- Deploy: Deploy the Flask application/container to Azure App Service.
- Database Connection: Ensure that the Flask application can communicate securely with the database.

4. External APIs

- Google Cloud Vision API: Used for image recognition, can be used for trying to figure out the company from the logo.
- Alpha Vantage API: Used to retrieve company ticker and symbol, and also stock information.
- ModePrediction: Prediction model that we trained to try to do image detection, in is capable of recognizing only 27 brands.

5. Database

- User Data: Stores user profiles, authentication data, and investment history.
- Cache: Temporarily stores frequently accessed data, like company information, to reduce API calls.

Basic Data Flow:

A. Image Capture and Recognition

User captures an image: Utilizing the Image Capture Module in the Angular front-end.
Image sent to the back-end: The image is sent to the Flask back-end for processing.
Image Processing: The image is pre-processed using OpenCV.
Image Recognition: Google Cloud Vision API identifies the product/company.

B. Retrieve and Display Information

Fetch Company Information: The recognized product/company data triggers a call to the Alpha Vantage API to retrieve relevant ticker and stock information.
Send Data to Front-End: Company and investment data are sent to the Angular front-end.
Display Information: Data is displayed to the user through the Information Display Module.

C. User Investment

User Initiates Investment: The user decides to invest using the Investment Action Module.
Authentication Check: The Authentication Module checks the user's authentication status.
Investment Action: The Investment Module communicates with the Investment Platform API to facilitate the investment.
Update User Data: The database is updated with the new investment action.

D. User Management

User Registration/Login: Users can register or log in through the Authentication Module.
User Profile Management: Users can view and manage their profiles and investment history.

Additional Considerations:

Since this is a POC and there are time constraints we are aware of the following potential issues that would need to be taken into account.

- Security: Ensure secure data transmission and storage, especially for user data and investment actions.

- Scalability: Design the back-end to handle potential scaling, considering factors like increasing users and data volume.
- User Experience: Ensure a smooth and intuitive user experience, focusing on easy navigation and clear data presentation.
- Error Handling: Implement robust error handling to manage issues like failed API calls, unrecognized images, or investment actions.
- Compliance: Ensure that the application adheres to financial and data protection regulations.

Trained Model:

In addition to using google vision api we also took up ourselves to train our own model to try and recognize brands. We trained the model on the dataset flickr_logos_27_dataset.

A. Preprocess data before training

The pre_process_data.py script is to prepare image data for training a machine learning model aimed at logo recognition and classification. It reads and cleans annotations related to various logos, creates a structured directory system for each unique brand, and organizes image data into designated training and validation sets based on an 80-20 split ratio. By systematically segregating and storing images into respective brand and training/validation directories, the script facilitates efficient data retrieval and utilization during model training and validation enhancing its ability to accurately classify different logos during predictive tasks.

B. Generate Training Model

Script generate_prediction_model.py generates a model to recognize and classify logos using TensorFlow and Keras. The script employs the MobileNetV2 architecture, a lightweight learning model, as a base, and adapts it for the specific task of logo classification. Data augmentation techniques, such as rotation, width shift, height shift, and horizontal flip, are applied to the training data using ImageDataGenerator to enhance the model's robustness and generalization by providing varied instances of logo images during training. The model is structured to utilize the pre-trained MobileNetV2 for feature extraction, followed by additional layers for classification into 27 categories (unique logos). Post-training, the model is saved in the Keras format.