**Bababos Mini Project Documentation**

***Version 1.0***

**Ahyo Haryanto (**[**ahyo.haryanto@gmail.com**](mailto:ahyo.haryanto@gmail.com)**)**

**February 2024**

# Background

This Price Engine project for Bababos aims to create a robust and scalable system for determining optimal pricing strategies to maximize profitability while maintaining competitiveness in the market. This documentation provides a comprehensive overview of design choices, architecture am implementation details of the project.

# Design System

The design of the Price engine was guided by the following key principles:

## Modularity

This system is designed to be modular, allowing for easy maintenance, scalability, and extensibility. Each component of the system is decoupled, making it easier to test and modify independently.

## Scalability

The system is built with scalability, allowing it to handle increase load and accommodate future growth. Horizontal scaling techniques, such as deploying multiple instance behind a load balancer, are employed to ensure high availability and performance.

## Security

Security is prioritized throughout the system design and implementation. Measures such as input validation, authentication, authorization, and encryption are implemented to protect against common security threats.

## Performance

Performance optimizations are applied at various level of the system, including database queries, caching asynchronous processing, and load balancing. There optimizations ensure the that the system performs efficiently even under high traffic conditions.

# Architecture

The Price Engine system for Bababos follows a microservices architecture, consisting of several independent services that communicate with each other via APIs. The architecture is divided into the following components:

## Web Application

The web application built using Flask serves as the interface for the Price Engine system. It handles incoming requests, processes data, and returns pricing recommendations to clients.

## Pricing Engine Service

The Pricing Engine service contains the core logic for determining optimal pricing strategies. It incorporates algorithms for cost-based pricing, competitive adjustment, dynamic pricing, value-based pricing, and promotional pricing.

## Database

The system utilizes a relational database PostgreSQL to store historical sales data, COGS data, market data, and pricing recommendations. The database schema is designed to support efficient querying and data retrieval.

## Authentication Service

An authentication service is responsible for user authentication and authorization. It verifies user credentials, issues access tokens, and enforces access control policies.

## External Integrations

The system integrates with external services and data sources, such as supplier price lists, competitor pricing APIs, and market research databases, to collect relevant data for pricing analysis.

# Implementation Details:

The Price Engine system is implemented primarily using Python and Flask for the web application and microservices. SQLAlchemy is used as the ORM for interacting with the database.

Key implementation details include:

## API Endpoints

The web application exposes RESTful API endpoints for receiving input data such as customer id and sku id and then returning pricing recommendations.

Here the endpoint’s lists:

Host = <http://127.0.0.1:8000>,

|  |  |  |  |
| --- | --- | --- | --- |
| Path | Method | Params | Function |
| / | Get | - | Index application, can be hit for checking if application still running |
| /dashboard | Get | Authorization header | Dashboard for login user |
| /customers | Get | - | Get all customers data |
| /pricelist | Get | - | Get all market prices |
| /suppliers | Get | - | Get all suppliers data |
| /quotatios | Get | - | Gell all quotations data |
| /purchaseorders | Get | - | Get all customer’s PO |
| /price/inquiry | Post | Customer\_id  Sku\_id | Get best price for certain product and customer |

## Pricing Algorithms

The Pricing Engine service implements various pricing algorithms to calculate optimal prices based on different factors, such as cost, demand, competition, and market trends. In the future this algorithm can be replace by machine learning approach using regression functions.

The workflow price engine can be described as follow:

1. Checking customer and sku from the system, if one of them is invalid then rejected.
2. Checking customer transaction history, if exist then get the minimum price from transaction then compare with market price, if history price is higher than market price, the best price will be the market price, otherwise the best price will be same with historical price
3. If customer does not have historical transaction, get the best price with the formula: **price = cost price \* ( 1 + desired profit margin + demand factor),** where cost price is the price from supplier, if missing it will be taken from 90% of market price. Desired profit margin will be the value profit margin, we can take 0.1 for 10% profit margin. Demand factor can be 0.1 – 0.2 for high demand, 0.05 – 0.1 for moderate and 0.01 – 0.05 for low demand. Compared to market price, if higher then market price will be best price.

## Security Measures

Input validation, authentication, and authorization mechanisms are implemented to ensure the security of the system. Passwords are hashed securely using bcrypt, and JWT tokens are used for user authentication.

## Performance Optimization

Database queries are optimized using indexes, caching techniques, and query optimization. Asynchronous processing is used for handling time-consuming tasks, and load balancing ensures even distribution of traffic across multiple instances.

# Conclusion

The Price Engine project for Bababos provides a scalable, secure, and efficient system for determining optimal pricing strategies. By following best practices in design, architecture, and implementation, the project aims to enable other developers to understand the code and replicate the system effectively.

We can improve our pricing engine system using machine learning for future release.