

**Architecture Document for**

**Dynamic pricing for Financial Products**

Table of Contents

[1 Introduction 1](#_Toc114544600)

[1.1 Purpose 1](#_Toc114544601)

[1.2 Scope 1](#_Toc114544602)

[1.3 Definitions, Acronyms and Abbreviations 1](#_Toc114544603)

[1.4 References](#_Toc114544604) 2

[2 Architectural Goals and Constraints 3](#_Toc114544605)

[2.1 Reusability 3](#_Toc114544606)

[2.2 Scalability 3](#_Toc114544607)

[2.3 Customizability 3](#_Toc114544608)

[2.4 Extendibility 3](#_Toc114544609)

[2.5 Use of Existing Business Logic 4](#_Toc114544610)

[2.6 Time to Market 4](#_Toc114544611)

[2.7 Portability 4](#_Toc114544612)

[2.8 Availability 4](#_Toc114544613)

[2.9 Performance 5](#_Toc114544614)

[2.10 Any Other Critical Goals Applicable 5](#_Toc114544615)

[3 Productization Assessment 6](#_Toc114544616)

[3.1 Re-Usable Components 6](#_Toc114544617)

[3.2 Analyze Architectural Frameworks in Repository 6](#_Toc114544618)

[3.3 Identify and Analyze Open Source and COTS Products 6](#_Toc114544619)

[4 System Architecture 7](#_Toc114544620)

[4.1 Overview 8](#_Toc114544621)

[4.2 Logical/Functional View 9](#_Toc114544622)

[4.3 Use Case View 11](#_Toc114544623)

[4.4 Deployment View 12](#_Toc114544626)

[5 General Architecture for Core Technical Services 13](#_Toc114544627)

[5.1 Persistence 13](#_Toc114544628)

[5.2 Inter-process Communication 13](#_Toc114544629)

[5.3 Authentication and Authorization 14](#_Toc114544630)

[5.4 Error Handling 14](#_Toc114544631)

[5.5 Logging 14](#_Toc114544632)

[5.6 Transaction Management 14](#_Toc114544633)

[5.7 Other Applicable Technical Services 14](#_Toc114544634)

[6 Risks/Limitation 15](#_Toc114544635)

[7 Alternative Solutions Considered 16](#_Toc114544636)

[Appendix 17](#_Toc114544637)

[1. Expected Software Response 17](#_Toc114544638)

[2. Performance Bounds 17](#_Toc114544639)

[3. Identification of Critical Components 17](#_Toc114544640)

[4. Review Comments on Architectural POC 17](#_Toc114544641)

[5. Justification of Changes to Existing Architecture (Optional) 18](#_Toc114544642)

**Document Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
|  |  |  |  |
|  |  |  |  |

**Note:** These documents are strictly for specific Virtusa/client use only. They shall not be shared with an external party other than the client concerned. These documents should always be kept securely and employees shall use reasonable care protecting these documents from unauthorized use or disclosure to a third party. This category also covers client intellectual property where Virtusa has a non-disclosure agreement with the client.

**Document Approval**

Virtusa Corporation and <Client>have reviewed this document and hereby agree that the contents herein are accurate. Any changes to this document must be communicated in writing and signed-off by both parties.

|  |  |
| --- | --- |
| **Signature** | **Signature** |
| **Date:** | **Date:** |
| **Name:** | **Name:** |
| **Client:** | **Virtusa Corporation** |

# Introduction

## Purpose

The purpose of this document is to outline the development and implementation of a dynamic interest rate model for Fixed Deposits (FD) and Home Loans. This model aims to provide personalized interest rates based on various customer attributes and market conditions. By leveraging a genetic algorithm (Gen Al), the model will suggest competitive rates to enhance customer retention while ensuring the bank's profitability. The solution intends to reward trustworthy customers with premium discounts and minimize risk exposure by adjusting rates accordingly.

## Scope

This project will cover the following:

* **Fixed Deposits (FD):** Dynamic interest rate suggestions based on customer investments, tenor, and risk profile.
* **Home Loans:** Interest rate recommendations considering credit history, risk rating, and customer association.
* The model will integrate factors such as:
  + Customer's credit history
  + Risk rating
  + Age factor
  + Customer's tenure with the bank
  + Past transactions
  + Current market trends
* The solution will assist the bank in maintaining a balance between offering competitive rates and managing financial risks.

## Definitions, Acronyms and Abbreviations

* **Employment Status:** Describes the typical employment situation for each age group.
* **Annual Income:** The typical yearly income range for each age group.
* **Savings & Assets:** The range of savings and assets typically held by each age group.
* **Credit Score:** A numerical representation of creditworthiness.
* **DTI Ratio (Debt-to-Income Ratio):** The percentage of income used to pay debts.
* **Loan Repayment History:** The duration of loan repayment history.
* **Existing Loans:** The range of existing loans held by each age group.
* **Loan Amount:** The typical loan amount sought by each age group.
* **Loan Tenure:** The duration over which the loan is expected to be repaid.
* **LTV Ratio (Loan-to-Value Ratio):** The ratio of the loan amount to the value of the asset purchased.
* **Down Payment:** The percentage of the asset's value paid upfront.
* **Interest Rate**: The typical interest rate on loans for each age group.
* **Repo Rate Influence**: The impact of the central bank's repo rate on loan interest rates.
* **Lender’s Risk:** The perceived risk to lenders when lending to each age group.
* **Promotional Interest Rates:** Availability of special interest rates.
* **Home Loan:** A home loan is a sum of money borrowed from a financial institution or lender to purchase, construct, renovate, or refinance a residential property.

## References

* **SBI Home Loans and Fixed Deposits (FDs):** Information related to State Bank of India's home loan products and fixed deposit schemes.
* **HDFC Home Loans and Fixed Deposits (FDs):** Details about HDFC's home loan offerings and fixed deposit options.
* **AXIS Home Loans and Fixed Deposits (FDs):** Information on Axis Bank's home loan products and fixed deposit plans.
* **Kaggle:** A platform for data science and machine learning, often used for datasets, competitions, and collaborative projects.
* **Sample Data Created by Data Analytical tools for Prototyping**

**Example**: Datasets generated by data analysts for testing, prototyping, or demonstrating analytical models.

# Architectural Goals and Constraints

This section describes the key objectives and limitations that influence the architecture of the dynamic interest rate model. It considers factors such as reusability, scalability, performance, and constraints like time-to-market and system availability.

## Reusability

* The system ensures that core components, such as the interest rate calculation engine and data handling modules, are reusable in future enhancements.
* The front-end UI components adapt to the next phase of back-end development, minimizing redevelopment efforts.
* Business logic remains modular, enabling seamless integration with other banking applications and future extensions.

## Scalability

* The architecture supports a growing number of users and increasing data volumes without significant performance degradation.
* The system accommodates future upgrades, such as transitioning from a flat-file storage system to a relational database for enhanced data management.
* Cloud-based deployment dynamically handles high computational loads.

## Customizability

* The UI allows modifications in terms of themes, layouts, and dashboards to align with business needs and branding requirements.
* Interest rate calculation parameters remain adjustable to reflect changes in banking regulations and economic conditions.
* Admin users control modifications to algorithm rules, customer segments, and rate adjustment factors without requiring code changes.

## Extendibility

* The system supports additional financial products beyond FDs and home loans, such as personal loans and auto loans.
* New AI/ML models integrate seamlessly to enhance risk assessment and rate prediction accuracy.
* The architecture enables plug-and-play functionality, allowing developers to introduce new interest rate adjustment criteria.

## Use of Existing Business Logic

* Existing banking rules and financial models remain encapsulated into reusable components (e.g., DLLs, APIs, or microservices).
* Core banking functionalities, such as credit history analysis, customer profiling, and risk assessment, integrate without requiring redevelopment.
* The system ensures backward compatibility with legacy banking software.

## Time to Market

* The development approach prioritizes rapid prototyping and agile methodologies to meet project deadlines.
* Technology choices favour frameworks that speed up development, such as Python (for AI models) and React.js (for UI).
* A minimum viable product (MVP) deploys first, followed by iterative enhancements based on user feedback.

## Portability

* The application remains compatible across multiple operating systems (Windows, Linux, macOS) and database environments (MySQL, PostgreSQL, MongoDB).
* Cloud compatibility ensures smooth deployment on AWS, Azure, or Google Cloud.
* The architecture supports easy migration from on-premise to cloud environments.

## Availability

* The system guarantees high availability with minimal downtime, ensuring uninterrupted banking services.
* Load balancing and redundancy strategies handle peak usage times.
* Automated backup and recovery mechanisms prevent data loss.

## Performance

* The system delivers real-time interest rate calculations without noticeable delays.
* Response times for UI interactions remain optimized to ensure a seamless customer experience.
* The model processes large datasets efficiently without excessive computational overhead.

## Any Other Critical Goals Applicable

* **Security**: Strong authentication and encryption protocols protect sensitive customer data.
* **Regulatory Compliance**: The model adheres to banking regulations and financial laws governing interest rate adjustments.
* **Auditability**: All interest rate recommendations remain logged and traceable for audit purposes.
* **User Experience**: The system provides an intuitive and user-friendly interface for both bank employees and customers.

# Productization Assessment

## Re-Usable Components

The reuse analysis identified the following components from the Home Loan Prediction Tool that can be applied across similar applications:

* **Customer Management Module**: Reusable for managing static customer data in any domain (e.g., banking, insurance).
* **Prediction Module**: Can be adapted with different models for use cases like credit scoring or insurance pricing.
* **Form UI Components**: HTML forms styled with Tailwind CSS can be reused with modified fields.
* **MongoDB Integration**: Database connection and CRUD logic can be generalized for any data-driven app.
* **ID Generation Logic**: The unique ID generation mechanism can be reused for identifying entities across applications.

## Analyze Architectural Frameworks in Repository

The project follows a lightweight **Flask-based MVC architecture**, which is modular and easy to extend. The following frameworks and patterns from the repository can be reused:

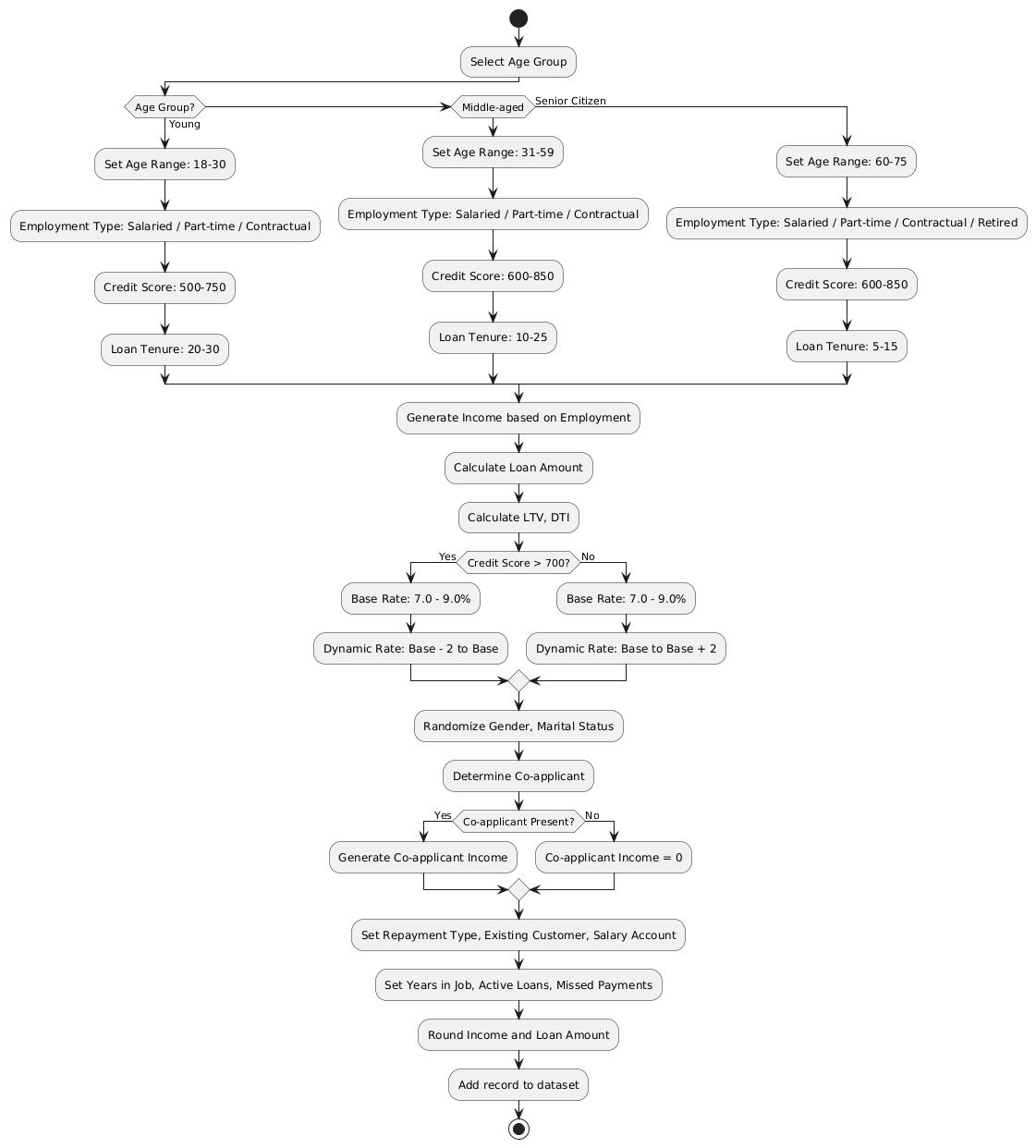
* **Flask Framework**: A micro web framework used for routing, request handling, and backend logic. It can be reused for other RESTful services or ML-based web apps.
* **MongoDB Integration**: The application includes MongoDB connectivity using pymongo, a reusable setup for storing and retrieving structured data.
* **Template Rendering (Jinja2)**: Flask’s templating engine is used for rendering HTML forms, enabling dynamic content delivery and reusability across interfaces.
* **Modular Structure**: The separation of prediction logic and customer data handling supports plug-and-play use in other financial or data-driven systems.

## Identify and Analyze Open Source and COTS Products

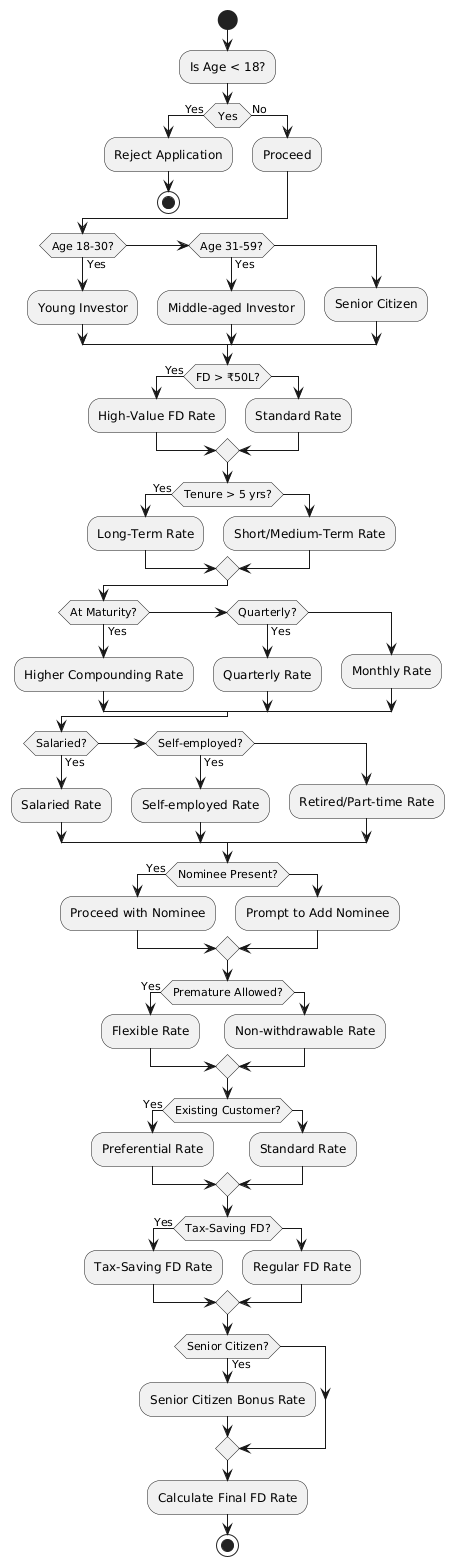
The project utilizes the following open-source and third-party (COTS – Commercial Off-The-Shelf) components:

* **Flask (Open Source)**
  + **License**: BSD License
  + **Usage**: Web framework for handling routing, requests, and server logic.
  + **Approval**: No restrictions for commercial use; attribution required.
* **pymongo (Open Source)**
  + **License**: Apache License 2.0
  + **Usage**: MongoDB client used for database operations.
  + **Approval**: Permissive use; attribution recommended.
* **Tailwind CSS (Open Source)**
  + **License**: MIT License
  + **Usage**: Front-end styling for HTML templates.
  + **Approval**: No restrictions; attribution optional.
* **Jinja2 (Open Source)**
  + **License**: BSD License
  + **Usage**: HTML template rendering in Flask.
  + **Approval**: Permissive license; attribution required.

# System Architecture



Architecture for Home-Loan



Architecture of FD

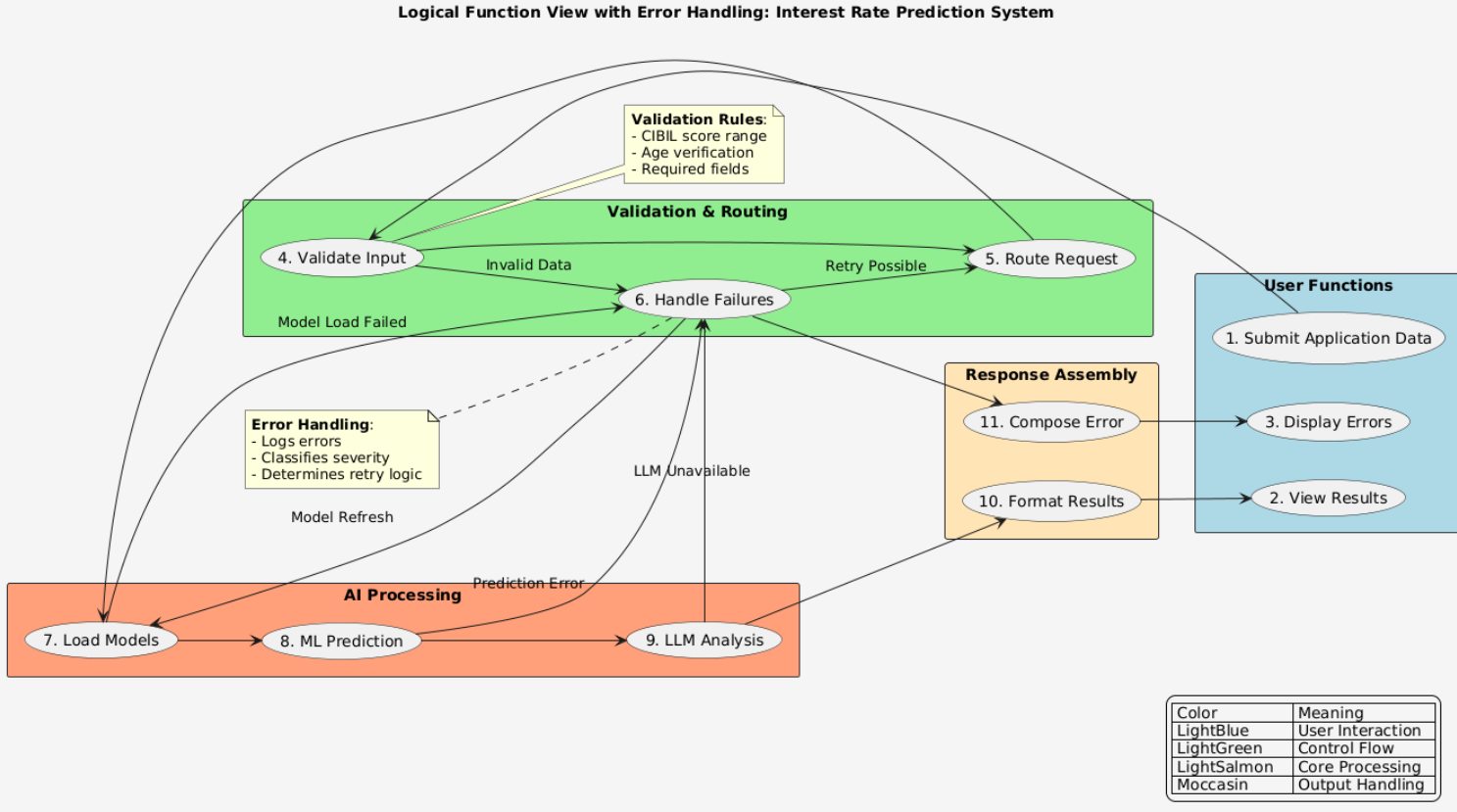
## Overview

The application is built using a **three-tier architecture**, promoting separation of concerns and reusability:

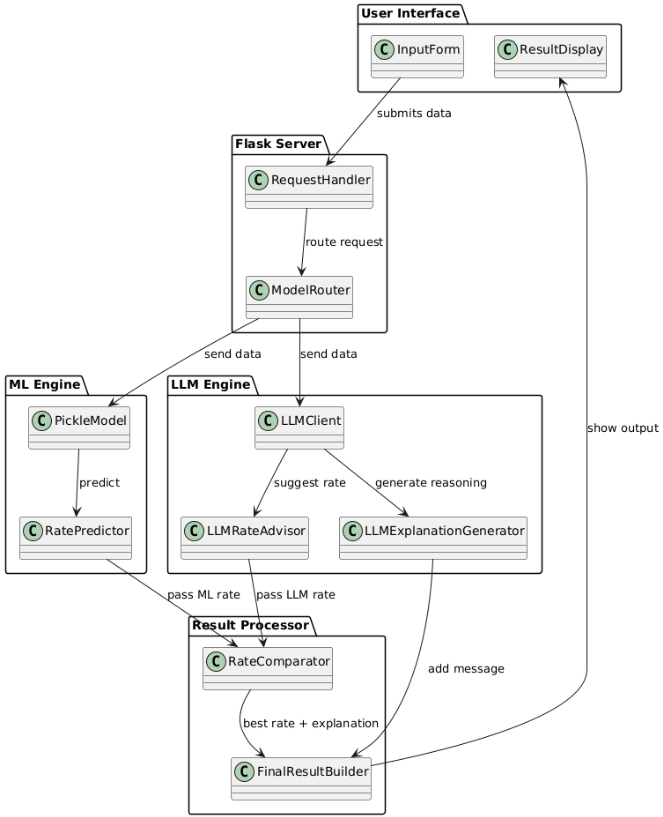
1. **User Interface Services**
   * Built with HTML and styled using Tailwind CSS.
   * Captures user inputs for customer information and prediction criteria.
   * Uses Flask’s templating engine (Jinja2) for dynamic rendering.
2. **Business Services**
   * Implements core logic including input validation, data transformation, and prediction using ML models.
   * Encapsulated in Flask routes and Python functions (app.py, predictor.py).
3. **Data Services**
   * Handles communication with the MongoDB database using pymongo.
   * Responsible for CRUD operations on static customer data and storing results.

These components interact through well-defined RESTful endpoints, ensuring scalability and maintainability.

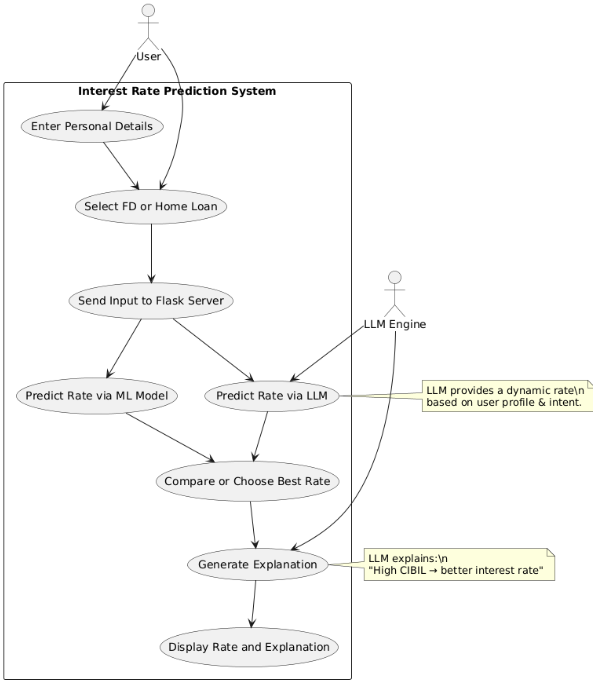
## Logical/Functional View



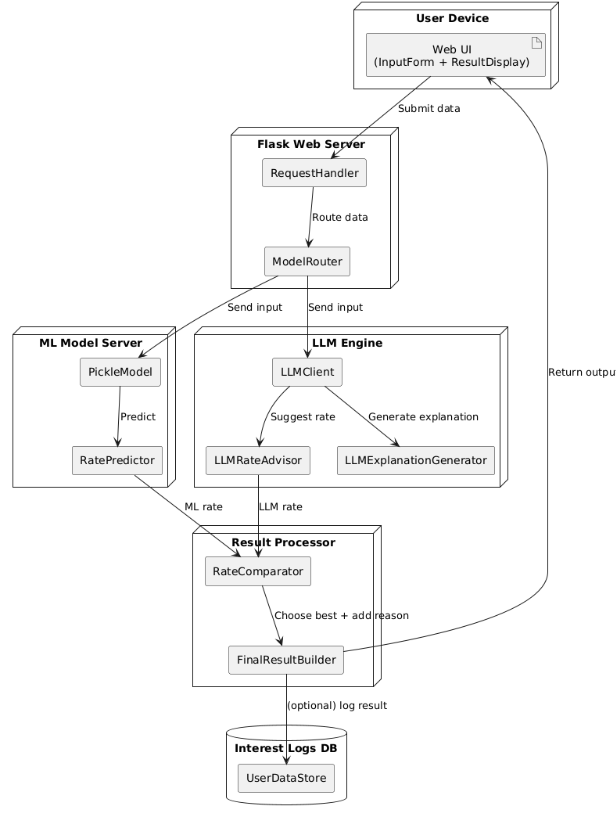
Option 2: Package Diagram



## Use Case View



## Deployment View



# General Architecture for Core Technical Services

* Defines the core components that provide essential system-level services.
* Ensures technical consistency, reliability, and maintainability across the application.
* Adopts a modular design strategy to allow flexibility and scalability.
* Services are selected and designed based on performance, security, and operational efficiency.
* Focus is placed on separating concerns (e.g., data, communication, logging) for better maintainability.
* Evaluates trade-offs like ease of development vs. performance, and complexity vs. robustness.
* Lays the groundwork for detailed sub-sections on persistence, communication, security, etc.

## Persistence

* Uses a relational database with structured data storage.
* ORM tools simplify object-database mapping.
* Separation of data access and business logic.
* Ensures scalability, concurrency control, and performance.
* Caching and connection pooling are used for optimization.

## Inter-process Communication

* Employs both synchronous (REST APIs) and asynchronous (message queues) communication.
* Supports decoupled architecture and microservices.
* Asynchronous queues enhance fault tolerance and scalability.
* Enables efficient service-to-service communication.

## Authentication and Authorization

 Implements OAuth 2.0 and JWT for token-based security.

 Centralized identity management with role-based access control.

 Ensures stateless, scalable, and secure login mechanisms.

 Supports integration with enterprise identity providers.

## Error Handling

* Centralized error capture and handling across layers.
* Standardized error codes and messages.
* Logging of errors aids in monitoring and debugging.
* Retry and fallback strategies help in graceful failure recovery.

## Logging

* Structured logging at INFO, DEBUG, and ERROR levels.
* Centralized log aggregation for real-time monitoring.
* Supports trace and correlation IDs for distributed systems.
* Aids in debugging, auditing, and root cause analysis.

## Transaction Management

* **APIs**: Flask RESTful endpoints for frontend-backend communication.
* **LLM Integration**: Async calls to NVIDIA API for explanations.
* **Data Format**: JSON payloads for consistency.
* **WebSocket Support:** Real-time updates for long-running LLM processes
* **API Versioning:** Implemented through URL path (v1/api/predict)

## Other Applicable Technical Services

* Includes installation scripts and configuration management.
* Uses caching to improve performance.
* Supports internationalization for multi-language use.
* Employs validation frameworks for data quality.
* Implements fault tolerance and client/server initialization.

# 6 Risks/Limitation

* Real-Time Interest Rate Updates: Real-time updates for interest rates can be complex to synchronize across systems, risking delays or inconsistencies.
* High System Load During Month-End: High transaction volumes during month-end processing or interest crediting may overload the system, requiring efficient load balancing and scalability.
* Security Risks: Sensitive financial data requires strong encryption and tokenization to prevent unauthorized access and breaches.
* Incorrect Interest Calculations: Errors in dynamic interest formulas or rate calculations can lead to financial inaccuracies, affecting customers and compliance.
* Dependency on External APIs: Reliance on third-party services (e.g., credit score APIs) can introduce risks like downtime, versioning issues, or licensing conflicts, impacting system reliability.
* Integration complexity can increase due to multiple interacting services and modules.
* Performance overhead may arise from centralized error handling, logging, or transaction control.
* Potential single points of failure in services like authentication or database if not designed for high availability.
* Scalability issues may emerge if caching or load balancing is not optimally configured.
* Security risks exist if token management or data validation are not strictly enforced.
* Dependency on third-party services may introduce versioning or licensing challenges.
* Network delays or failures can impact inter-process communication in distributed systems.

# 

# 7 Alternative Solutions Considered

* Django was evaluated but rejected in favour of Flask for its lightweight architecture and faster development cycle for our MVP needs
* OpenAI GPT-4 was tested but NVIDIA Llama 3 was chosen due to comparable performance at significantly lower operational costs.
* AWS Lambda serverless was considered but traditional Flask servers were chosen for simpler debugging and state management.
* A monolithic architecture was reviewed but rejected in favour of modular services for better independent scaling of components.
* Traditional rule-based rate calculation was explored but machine learning models provided more accurate personalized predictions.

# Appendix

## Expected Software Response

* **Fixed Deposits (FD):**
  + Accurately calculate and credit interest upon maturity.
  + Apply dynamic rate adjustments based on customer profile and market conditions.
* **Home Loans:**
  + Adjust interest rates in real-time based on credit score, loan tenure, and risk factors.
  + Provide clear breakdowns of rate calculations.
* **System Performance:**
  + User login, account dashboard, and transaction history must load within **2 seconds** under normal load.
  + Prediction results (loan/FD rates) should display within **5 seconds** (including LLM explanations).

## Performance Bounds

* Latency:
  + API responses ≤ 500ms for CRUD operations.
  + LLM explanations ≤ 3 seconds (90th percentile).
* Data Volume: Handle 10,000+ customer records with efficient querying.
* Uptime: 99.9% availability (excluding scheduled maintenance).

## Identification of Critical Components

* **NVIDIA LLM API:** Critical for generating user-facing explanations.
* **MongoDB Connection Pool**: Failure disrupts all customer data operations
* **Flask Prediction Endpoints:** Core to loan/FD rate calculations.
* **Authentication Service:** Must be fault-tolerant to prevent unauthorized access

## Review Comments on Architectural POC

* **Scalability Concerns**: The system showed performance issues under high load; reviewers recommended implementing auto-scaling, caching, and database optimization.
* **Real-Time Sync Issues**: Interest rate updates were not consistently synchronized across services; adopting event-driven architecture was suggested.
* **Security Enhancements Needed**: Basic encryption was present, but improvements like token rotation, secure key storage, and stricter validation were advised.
* **Lack of Fault Tolerance for APIs**: No fallback mechanisms for third-party API failures; reviewers recommended adding retries, circuit breakers, and timeout handling.
* **Incomplete Validation of Interest Calculations**: Edge cases and formula changes were not thoroughly tested; version control and validation layers were suggested.
* **Single Point of Failure**: Authentication and database components lacked redundancy; high availability and clustering solutions were recommended.

## Justification of Changes to Existing Architecture

* **From Flat Files to MongoDB:**
  + Enabled scalable customer data management (Slide 7).
* **Added Modular Services:**
  + Replaced monolith to isolate ML/LLM components for easier scaling.
* **Fallback to OpenAI API:**
  + Mitigated risks of NVIDIA API outages.