Architecture Document for Reverse Budgeting App

Document Version: 1.0  
Date: 13 March 2025  
Author: Aarav Shah, Swayam Gupta, Sourabh Dhar  
Project: Reverse Budgeting App

## Introduction

This document presents the architecture for the Reverse Budgeting App, designed to automatically manage budgets and suggest investments by interfacing with HDFC Bank APIs. The architecture aims to provide a scalable, maintainable, and secure application using a Microservices approach.

## Architectural Style

The project adopts a Microservices architecture, facilitating the development of independently deployable, modular services. This architecture supports scalability, resilience, and continuous deployment.

## System Overview

The Reverse Budgeting App consists of multiple microservices, each responsible for a specific aspect of the application, such as user authentication, transaction processing, budget calculation, and investment recommendations.

## Microservices Description

* Authentication Service: Manages user authentication and security.
* Transaction Service: Interfaces with HDFC Bank API to retrieve transaction data.
* Budgeting Service: Analyzes transaction data to provide budget recommendations.
* Investment Service: Suggests investment opportunities based on user profiles and market data.

## Components:

Azure App Service:

* Hosts the web and API applications.
* Manages the backend built with Node.js/Python for handling business logic such as transaction synchronization, budget calculation, and investment advice.

Azure SQL Database:

* Stores relational data including user profiles, transaction details, and budget records.
* Ensures data integrity and supports complex queries needed for financial analysis.

Azure Cosmos DB:

* Manages unstructured data such as logs and possibly user interactions which do not fit into the relational model.
* Provides fast read and write capabilities which are essential for real-time data processing.

Azure Active Directory (Azure AD):

* Manages user authentication and authorization.
* Integrates seamlessly with Azure App Service for secure login processes.

Azure API Management:

* Manages and secures communication between the app’s microservices and external APIs.
* Provides a gateway for all API requests, enhancing security and monitoring.

Azure Functions:

* Handles event-driven tasks such as notifications or executing background jobs for investment suggestions.
* Supports scaling out without managing infrastructure, ideal for fluctuating workloads.

## Diagrams

### Architecture Diagram

This diagram provides a high-level overview of the entire system architecture, showing how different components and services are interconnected within the Azure cloud environment. It illustrates the deployment of web and mobile applications on Azure App Service, the use of Azure API Management for routing and securing API calls, integration with Azure Active Directory for authentication, and the roles of Azure SQL Database and Cosmos DB in data management. This diagram is crucial for understanding the flow of data and control across the application.

A diagram of a software company

AI-generated content may be incorrect.

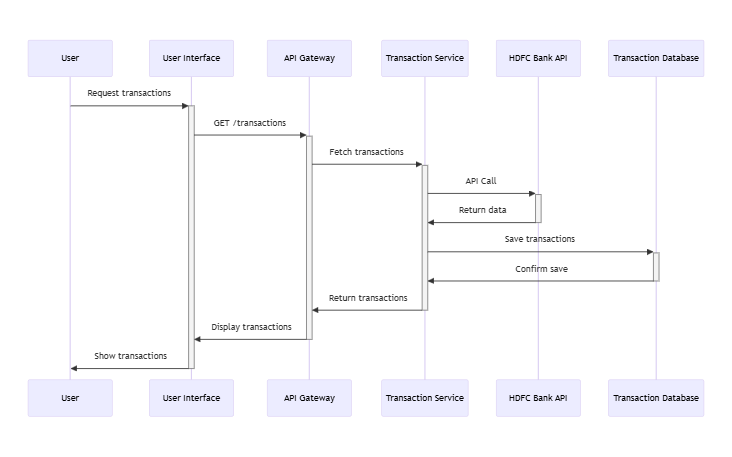
### Component Diagram

The component diagram details the internal structure of the application, breaking down the system into various components that make up the backend and frontend services. It shows how components such as the Authentication Service, Transaction Service, Budget Service, and Investment Service are modularized and interact with each other through interfaces. This diagram is key for developers and architects to understand the responsibilities and boundaries of each component within the system.

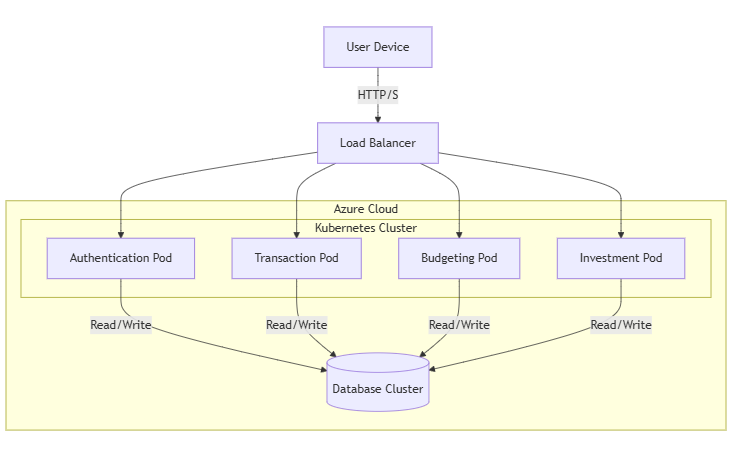
A diagram of a service

AI-generated content may be incorrect.

### Sequence Diagram

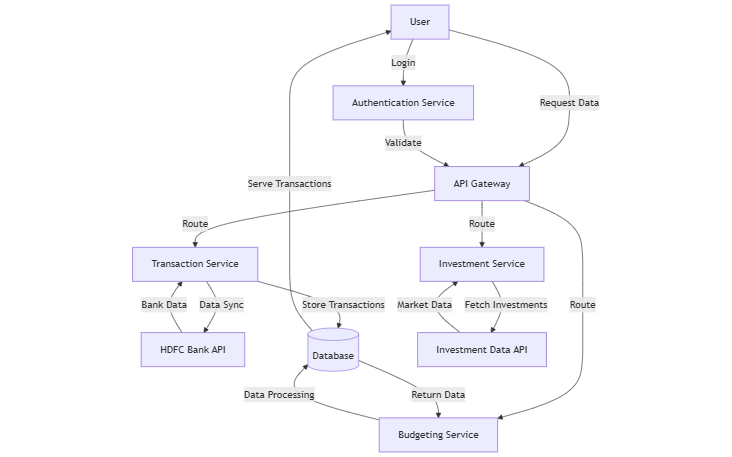
This diagram illustrates the interactions between components over time for specific use cases or processes. For instance, it can show the sequence of steps that occur when a user logs into the app, from the initial request through the web or mobile app, through the API gateway, and finally through the authentication process handled by Azure Active Directory. The sequence diagram is essential for visualizing the order of operations and the interactions between system components for particular scenarios.Deployment Diagram

The deployment diagram visualizes the physical deployment of artifacts (software components, code, and data) on hardware represented by Azure services. It shows how the software components are deployed on the Azure infrastructure, including web servers, application servers, and databases. This diagram helps stakeholders understand how the app’s deployment configuration is managed, detailing the cloud resources used, such as virtual machines, containers, or serverless functions.

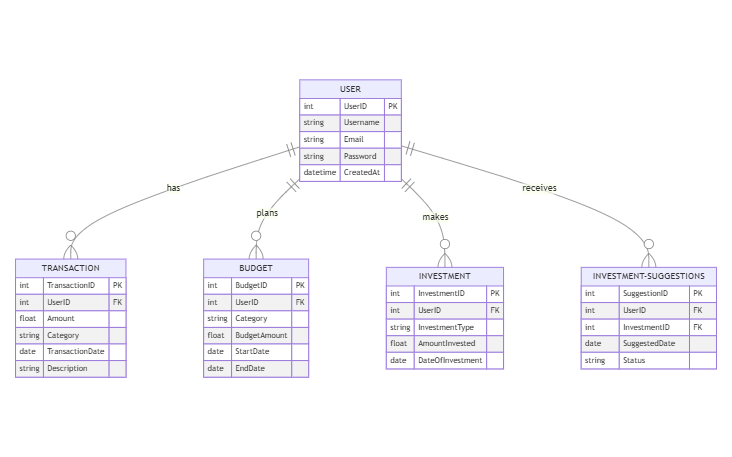


### Data Flow Diagram (DFD)

A DFD provides a visual representation of the data flow within the application. It shows how data inputs into the system are transformed and routed through different processes, and where data is stored. For the Reverse Budgeting App, it might depict how transaction data flows from the bank API, through the transaction service, and into the database, then how it is utilized by the budgeting service to generate financial insights.



### ER Diagram/Schemas



## Data Exchange Contract

The Data Exchange Contract outlines the specifics of how data is communicated between different parts of the system and external APIs. This section defines the APIs used in the Reverse Budgeting App, including endpoints, supported HTTP methods, and the structure of requests and responses.

* **Endpoints and Methods**:
  + **User Authentication**: /api/auth
    - **POST**: Authenticate user credentials and return a token.
  + **Transaction Sync**: /api/transactions
    - **GET**: Retrieve user transactions.
    - **POST**: Sync new transaction data from HDFC Bank.
  + **Budget Management**: /api/budgets
    - **GET**: Fetch user budgets.
    - **POST**: Create a new budget.
    - **PUT**: Update an existing budget.
  + **Investment Suggestions**: /api/investments/suggestions
    - **GET**: Obtain investment suggestions for the user.
  + **Request Formats**: JSON format for sending and receiving data.
  + **Response Structures**: JSON objects that include status codes, messages, and data payloads where applicable.
  + **Frequency of Data Exchanges**:
    - Transactions are synchronized daily.
    - Budget updates and investment suggestions are generated on-demand or when relevant transaction data is updated.
  + **Mode of Exchanges**: Communication with external APIs (e.g., HDFC Bank API) is managed via secure HTTPS requests, facilitated by Azure API Management which ensures secure and efficient API calls.

## Security Measures

This section outlines the security strategies employed to protect data integrity and privacy across the Reverse Budgeting App. It covers the mechanisms in place to secure API access and data transmission.

* **Authentication and Authorization**:
  + Utilizes Azure Active Directory for managing user identities and permissions.
  + Implements OAuth 2.0 for secure API access, ensuring that only authenticated users can access their data.
* **Data Encryption**:
  + All data in transit is encrypted using TLS protocols.
  + Sensitive data at rest, such as passwords and financial information, is encrypted using AES-256 encryption standards.
* **API Security**:
  + Azure API Management is used to secure, monitor, and manage traffic between users and the app’s microservices, as well as between the app and external APIs.
  + Includes rate limiting, IP whitelisting, and logging of all API requests to detect and respond to potential security threats promptly.

## Technology Stack

The technology stack for the Reverse Budgeting App incorporates modern and widely-used technologies that support robust development practices, scalability, and easy maintenance.

* **Frontend**:
  + **React**: Used for building the dynamic web frontend.
  + **React Native**: Employs the same React principles to develop cross-platform mobile applications for iOS and Android.
* **Backend**:
  + **Node.js and Express**: Serve as the runtime environment and the framework for building fast, scalable network applications.
  + **Python with Django**: Utilized for complex backend processing that requires strong framework support for database operations and middleware integration.
* **Database**:
  + **PostgreSQL**: Relational database for storing structured data like user profiles and transactions.
  + **MongoDB**: NoSQL database used for more flexible data requirements such as logs and session data.
* **DevOps**:
  + **Docker**: For containerizing applications, ensuring consistency across multiple development and production environments.
  + **GitHub Actions**: Manages CI/CD pipelines for automating testing, builds, and deployments.
* **Cloud Infrastructure**:
  + **Azure**: Hosts applications and databases, leveraging Azure’s comprehensive cloud services to enhance scalability and reliability.

## Conclusion

This document has detailed the architectural decisions, components, and security strategies of the Reverse Budgeting App. The chosen microservices architecture facilitates scalability, flexibility, and independence of deployment, which are critical for handling complex and variable financial data efficiently. While the architecture promises enhanced scalability and modular maintenance, potential challenges include managing the complexity of interactions between services and ensuring consistent data integrity across distributed systems. The technology stack was selected to optimize both development efficiency and operational performance, ensuring that the app can scale to meet user demand and provide robust, secure financial management tools.