Architecture Documentation

Financial Management: Full Stack Web Application

V0.1

[1 Introduction 2](#_Toc964093502)

[2 Design Goals 2](#_Toc815120823)

[3 System Behavior 2](#_Toc1695075801)

[4 Logical View 3](#_Toc1940748734)

[5 Process View 4](#_Toc181726215)

[6 Development View 5](#_Toc1108346109)

[7 Physical View 5](#_Toc341791965)

[8 Non-Functional Requirements 6](#_Toc399918630)

[8 Use Case View 6](#_Toc1543076290)

[9 Technology Stack 7](#_Toc791429667)

[10 Contributors/ Team Roles 7](#_Toc1577269322)

**Revision History**

**Version:** 0.1

**Modifier:** Madeline Brothers

**Date:** 10/01/2025

**Description of Change:** Creation of document

**----------------------------------------------------------------------------------------------------------------**

**Version:**

**Modifier:**

**Date:**

**Description of Change:**

## **1 Introduction**

This document describes the architecture and design for the Banking Web Application that enables customers to manage their finances securely and conveniently. The system provides features such as account dashboards, budgeting, transaction history, fund transfers, statements, and an integrated AI chatbot.

The document addresses the concerns of multiple stakeholders:

* **Users/Customers**: expect a secure, reliable, and user-friendly interface.
* **Developers**: need a modular, maintainable, and testable system.
* **Project Managers**: want clear separation of responsibilities and manageable dependencies.
* **Compliance/Security Teams**: require strong encryption, authentication, and adherence to regulations.

## **2 Design Goals**

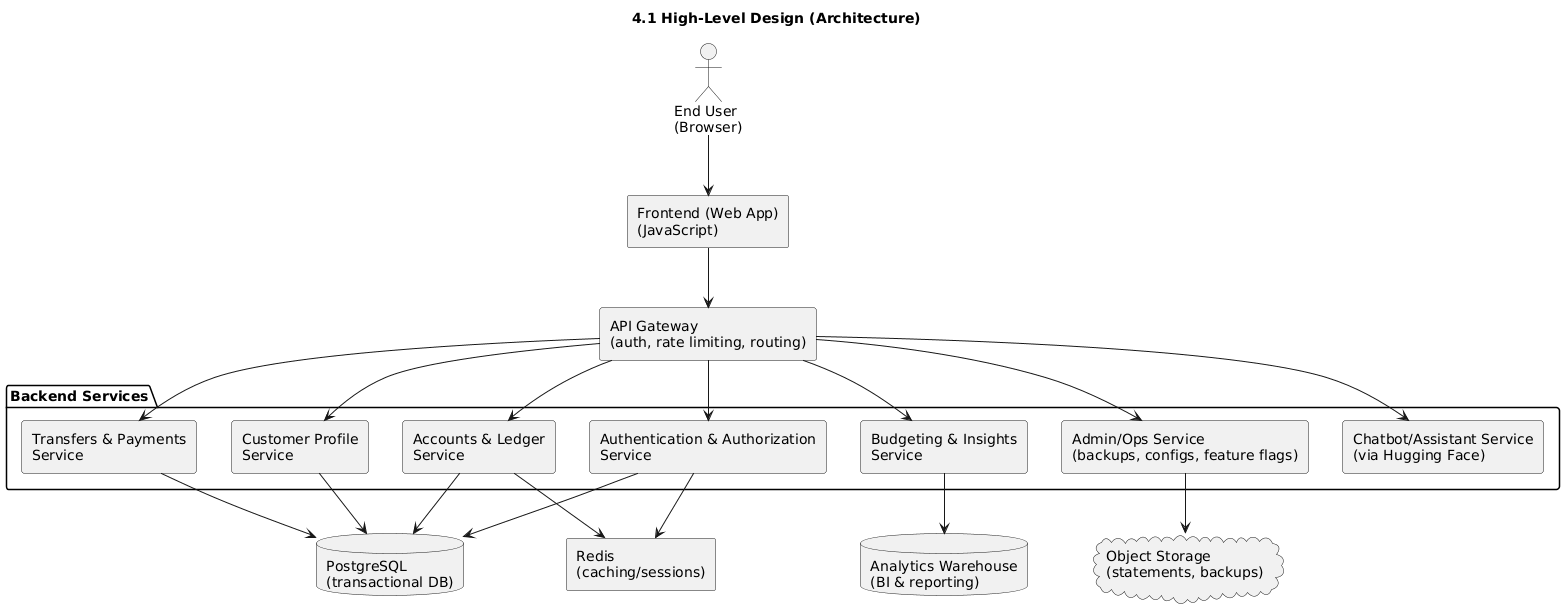
* **Accessibility**: Cross-platform (desktop and mobile browsers), multilingual support, WCAG compliance.
* **Security**: MFA, TLS encryption, secure password storage, regular audits.
* **Scalability**: Cloud-based, elastic scaling, load balancing.
* **Availability & Reliability**: ≥99.9% uptime, daily backups, disaster recovery plan.
* **Performance**: Response time ≤2s under normal load.

## **3 System Behavior**

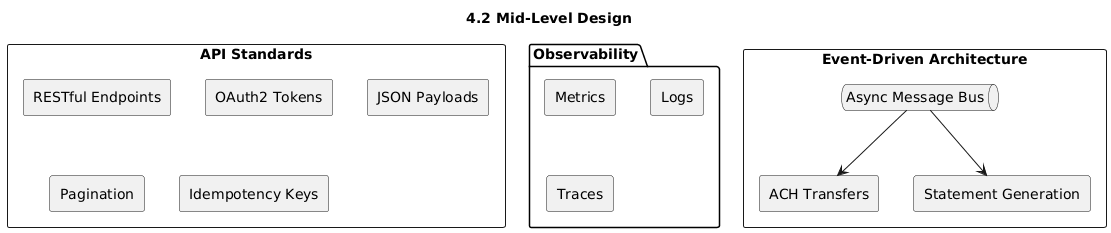
System use cases include:

* User registration, login with MFA.
* Profile management (edit personal data, preferences).
* Dashboard navigation with access to budgets, statements, transfers.
* Transaction history with filtering and search.
* Backup/restore operations for admins.
* CI/CD operations (deployment, automated scans).

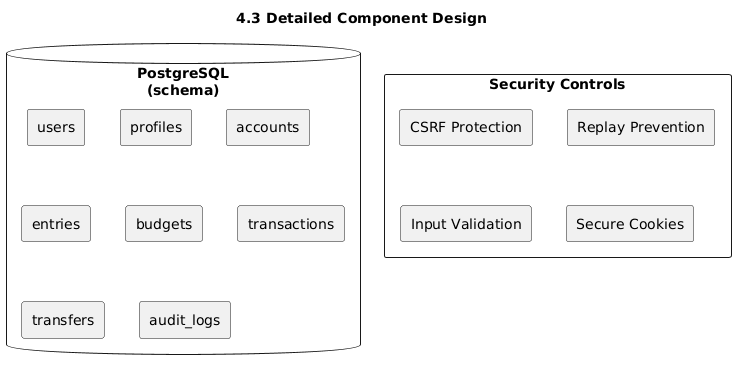
## **4 Logical View**

**4.1 High-Level Design (Architecture)**

* **Frontend (Web App)**: JavaScript
* **API Gateway**: Manages authentication, rate limiting, routing.
* **Backend Services**:
  + Authentication & Authorization Service
  + Customer Profile Service
  + Accounts & Ledger Service
  + Transfers & Payments Service
  + Budgeting & Insights Service
  + Admin/Ops Service (backups, configs, feature flags)
  + Chatbot/Assistant Service via Hugging Face
* **Datastores**:
  + PostgreSQL (transactional DB)
  + Redis (caching/sessions)
  + Object Storage (statements, backups)
  + Analytics Warehouse (BI & reporting)

**4.2 Mid-Level Design**

* **Event-driven architecture**: asynchronous message bus for ACH transfers, statement generation.
* **API Standards**: RESTful endpoints with OAuth2 tokens, JSON, pagination, and idempotency keys.
* **Observability**: metrics, logs, traces.

**4.3 Detailed Component Design**

* **Database schema** includes users, profiles, accounts, entries, budgets, transactions, transfers, audit\_logs.
* **Security controls** include CSRF protection, replay prevention, input validation, and secure cookies.

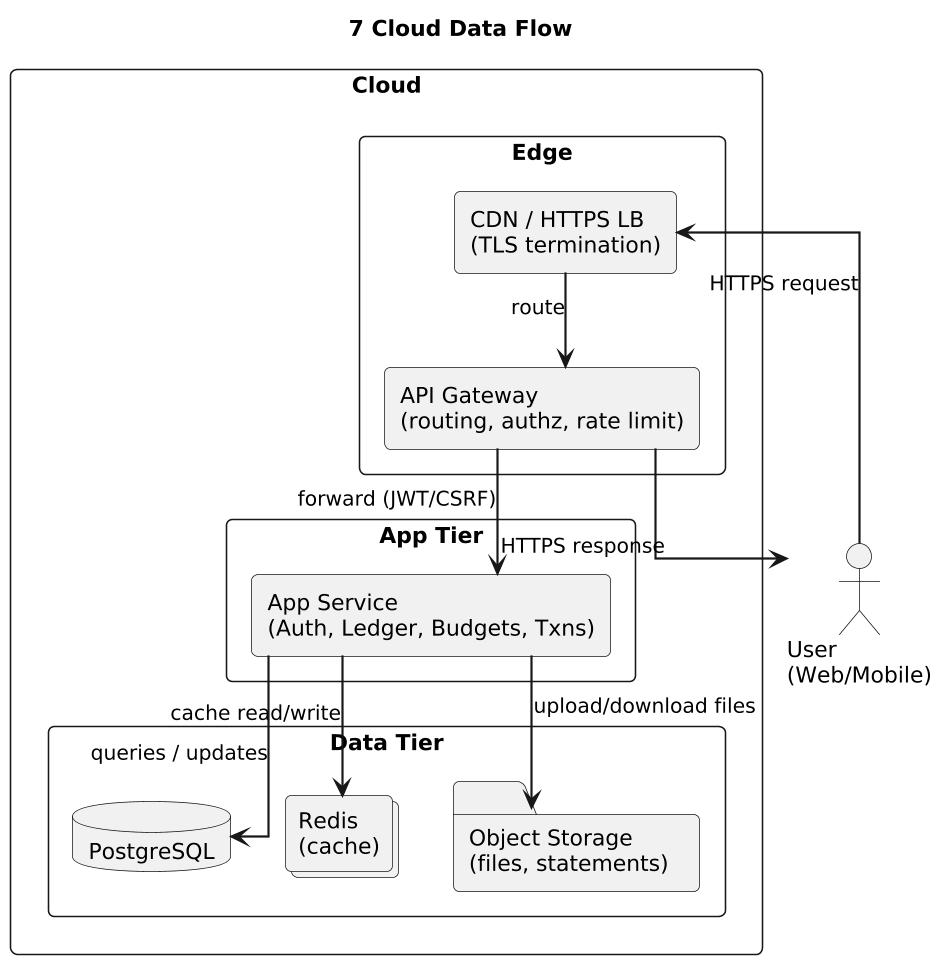
## **5 Process View**

* Separate processes for frontend, backend services, database, and monitoring.
* Services communicate via REST APIs and async message queues.
* CI/CD pipeline processes: build → test → security scans → deploy → rollback if needed.

## **6 Development View**

* **Teams**: Frontend, Backend Services, Security, DevOps/SRE.
* **Codebase**:
* **Version Control**: Git with protected branches, peer reviews, CI/CD integration.

## **7 Physical View**



* **Platform:** Docker-based containerization for consistent runtime environments across development, staging, and production.
* **Deployment:** Containers are orchestrated with Docker Compose for local testing and Kubernetes or another managed container service for scalable production environments.
* **Microservices:** Each domain service (Auth, Accounts & Ledger, Transactions, Budgets, Analytics) runs in an isolated container and exposes RESTful APIs through a central API Gateway.
* **Networking:** Service-to-service communication occurs over internal Docker networks; external ingress passes through an HTTPS load balancer for TLS termination and routing.
* **Persistence Layer:** PostgreSQL and Redis run as containers or managed services. Object Storage handles receipts, statements, and generated exports.
* **Observability:** Custom Observability Service records logs, metrics, and traces directly to database tables (e.g., event\_logs, audit\_logs, metrics\_rollups, traces\_spans) using background roll-ups and TTL pruning.
* **Security:**
  + Minimal base images scanned for vulnerabilities prior to deployment.
  + Secrets managed via environment variables or a secure vault.
  + Network segmentation and restricted ingress/egress policies to protect sensitive data.
* **CI/CD Pipeline:** Source repository integrated with GitHub Actions for automated build, testing, image publishing, and deployment to the container registry.

## 8 Non-Functional Requirements

* 1. Useability: The system must be intuitive and easy to navigate for users with varying levels of technical proficiency. Interfaces should follow consistent design and provide help prompts with choice of AI assistance. A new user should be able to perform basic tasks, such as checking balances, creating a budget, or locking a card within the first five minutes of use.
  2. Efficiency: The application should respond to user requests within two seconds under normal operating conditions. Resource utilization (CPU, memory, network) must remain within acceptable limits to support concurrent users.
  3. Security: All data must be encrypted in transit and at rest. The system must enforce strong password policies, session timeouts, and multi-factor authentication. Regular testing and code scanning will be conducted.
  4. Availability: the application should have a 99.9% uptime, excluding scheduled maintenance. Cloud-based redundancy and failover strategies must ensure continuity of service.
  5. Reliability: Backups must be performed daily and retained according to policy. The system should recover from failure without data loss beyond the most recent backup. Error handling must gracefully notify users without exposing sensitive information.

## **8 Use Case View**

**Use Case 1: Sign Up/Login**  
Actors: Customer  
Flow: User enters details → system verifies → MFA → session created.

**Use Case 2: Profile Settings**  
Actors: Customer  
Flow: User edits profile → system validates → updates database → confirms changes.

**Use Case 3: Transfer Funds**  
Actors: Customer  
Flow: User enters transfer details → system checks limits & risk → transaction posted to ledger → confirmation sent.

**Use Case 4: Backup/Restore**  
Actors: Admin/DevOps  
Flow: Admin schedules backup → system encrypts & stores → restore validated on staging → promoted to production.

**Use Case 5: Deployment with CI/CD**  
Actors: DevOps, CI/CD System (GitHub)  
Flow: Pipeline builds image → runs scans → deploys to staging → canary release → production promotion.

## **9 Technology Stack**

* Python, C, Java for backend services
* PostgreSQL for data management
* JavaScript, HTML, and CSS for the frontend interface
* BERT Financial Model
* Docker.io for Containerization
* Redis for cache
* NGINX for Gateway
* CSRF/JWT Tokens

## **10 Contributors/ Team Roles**

* Project Manager | Cloud Security Engineer | [Jeffery Kimbrow](https://github.com/kimbrow-slice)
* Backend Developer | AI Developer | [Madeline Brothers](https://github.com/madelinebro)
* Frontend Developer | API Developer | [Yasmin Hassan](https://github.com/madelinebro/HashSlingingSlashers/blob/main/***)
* Documentation Lead | Prototype Developer | [Samantha Saunsaucie](https://github.com/madelinebro/HashSlingingSlashers/blob/main/***)