AWS First Cloud AI Journey

Project Plan

GC Team

FPT University HCMC

Personal Finance Management App

12.12.2025

TABLE OF CONTENTS

1 BACKGROUND AND MOTIVATION

1.1 EXECUTIVE SUMMARY

The Vicobi Project – Personal Finance Management (PFM) App is designed to deliver a smart, modern, and comprehensive automated platform for personal financial management, addressing the need for efficient and proactive control over personal spending.

Vicobi simplifies financial management through four key pillars:

- *Smart Recording:* Allows users to easily record transactions (income/expense) using convenient methods such as voice and image recognition, eliminating the barriers of manual data entry.
- *Goal-based Budgeting:* Automates the creation and flexible management of multiple financial funds tailored for specific financial goals.
- *Analysis & Control:* Provides intuitive analytical reports and a system of smart alerts, supporting effective planning and proactive financial control.
- *Personal Finance Support (AI Chatbot):* Integrates an Intelligent AI Chatbot acting as a consultant, providing theoretical foundation and support for problem resolution, thereby enhancing users' financial literacy and skills.

Technologically, Vicobi is built upon a Microservices Architecture utilizing modern frameworks such as .NET and FastAPI. The system will be deployed on the AWS Cloud platform to ensure high flexibility, scalability, and absolute data security.

The development process will adhere to the Agile/Scrum model (2-week sprints), with the goal of completing the Minimum Viable Product (MVP) within 2 months.

1.2 PROJECT SUCCESS CRITERIA

What’s the Problem?

In the current dynamic market, consumer spending frequency is increasing rapidly. However, a significant number of users struggle to maintain financial control, primarily due to behavioral inertia—specifically, the reluctance to manually record every transaction [1]. Although the market features numerous Personal Financial Management (PFM) applications, the majority still rely heavily on manual data entry. This process is not only time-consuming and prone to errors but creates a high friction point that causes "input fatigue," leading to a high abandonment rate.

Consequently, existing solutions primarily focus on post-transaction statistics but fall short of bridging the gap between high spending frequency and the lack of automated tracking.

Solution

We propose building a highly automated personal financial management platform by leveraging the power of the AWS Cloud and a Microservices Architecture.

- *Core Technological Advantage:* Integrating Artificial Intelligence (AI) for Vietnamese-specific speech processing (voice-to-text) and detailed invoice recognition (bill scanning).
- *Architecture:* The system is architected for efficiency and scalability on AWS. The backend utilizes AWS ECS Fargate, implementing a cost-optimization strategy by consolidating both

the .NET backend services and the FastAPI AI service within a single Fargate execution environment (Multi-container Task). This approach minimizes infrastructure overhead while maintaining seamless communication between services. On the client side, the Next.js frontend strictly adheres to AWS Best Practices by utilizing Amazon S3 for secure static hosting, distributed globally via Amazon CloudFront (CDN) to ensure low latency and high performance.

- *Competitive Edge:* Compared to main competitors (Money Lover, Misa Money Keeper), Vicobi focuses on fully automating data entry, significantly reducing manual operations and errors, making it easier for individuals and small groups to control their finances. The platform is designed to be flexible, ready for future expansion to enterprise scale or digital banking integration.

Benefits and Return on Investment (ROI)

The Vicobi solution provides substantial benefits, creating a distinct competitive advantage across three main aspects: User Value, Economic Efficiency, and Technological Advantage.

User Value

Vicobi helps users improve both the efficiency and accuracy of their financial management. The AI technology reduces manual data entry by over 70% compared to traditional applications. Specifically, the system achieves up to 90% accuracy in voice recognition and processing, and 80% accuracy in invoice detail extraction. This allows users to record and categorize transactions in just a few seconds, optimizing the overall user experience. Crucially, minimizing input errors ensures financial data integrity and builds user trust in the automated system.

Economic Efficiency and Return on Investment

Economically, the project maximizes cost-efficiency by strategically leveraging the AWS Free Tier for foundational infrastructure. Key components—including Amazon S3 for storage, Amazon CloudFront for global content delivery, Amazon Cognito for authentication, and Amazon API Gateway with Application Load Balancer (ALB) for traffic management—are utilized within their free usage limits (12-month or always-free). Operational monitoring via Amazon CloudWatch and notifications via Amazon SNS also incur zero initial cost.

Consequently, the estimated operational budget is kept lean, at approximately ~$60/month (primarily allocated for ECS Fargate compute resources and AWS WAF security) and ~$15/month for specialized AI compute tasks. Due to the significant time savings on data entry and improved operational efficiency, the project is expected to achieve a Return on Investment (ROI) within 6–12 months.

Technological Advantage and Scalability

The Microservices Architecture deployed on AWS not only ensures high security and performance but also provides superior Scalability. This allows the development team to easily add advanced features such as a dedicated mobile application, enhanced analytics, or direct integration with banks in the future, helping Vicobi maintain a long-term competitive edge in the market.

1.3 ASSUMPTION

Risk Assessment

Risk Matrix

The project has identified and assessed five primary risks, prioritized based on their Impact and Probability. Risks with High Impact and Low Probability include User Information Leakage (from Cognito or the Database) and AWS Connectivity Loss or Regional Service Failure. Risks categorized as having Medium Impact and Medium Probability are Data Synchronization Errors between Microservices and AI Model Misclassification (voice/invoice). The risk of AWS Usage Budget Overrun is assessed as Medium Impact but Low Probability.

Mitigation Strategies

To counteract these risks, we will implement the following specific mitigation strategies:

- *Security:* To mitigate the risk of information leakage, we will implement end-to-end data encryption (AES-256) and secure connections (HTTPS). All access will be strictly controlled using IAM under the Least Privilege Principle. Furthermore, AWS WAF will be deployed to safeguard the application layer against common web exploits (such as SQL Injection and XSS) and malicious bot traffic.
- *High Availability:* To minimize the risk of AWS connectivity loss or zonal failures, the architecture will be deployed in a Multi-Availability Zone (Multi-AZ) model for critical compute and networking services, specifically ECS Fargate and the Application Load Balancer. This ensures continuous service operation even if a specific zone goes down, coupled with regular automated data backups for resilience.
- *AI Quality:* The risk of AI misclassification will be reduced by continuously improving the OCR and Voice-to-Text models through additional training with real-world data and performing frequent regression testing.
- *Data Synchronization:* To address synchronization errors between Microservices, we will deploy a RabbitMQ instance directly within the ECS Fargate environment. This approach ensures low-latency asynchronous task processing with an integrated retry mechanism in case of service failure, while simultaneously optimizing internal communication costs by avoiding external managed queue services.
- *Cost Management:* The risk of budget overrun will be managed by configuring AWS Budget Alerts and regularly optimizing ECS Fargate and S3 resources based on actual usage.

Contingency Plans

- *AWS Regional Failure:* If an AWS region experiences an outage, the system will enter a contingency mode by temporarily storing transaction data locally or in cache (e.g., Redis/local storage) and will perform synchronization back to the main database (RDS) immediately after AWS service restoration.
- *Disaster Recovery (DR):* As part of our future scaling roadmap, we plan to implement automated infrastructure restoration using pre-defined IaC (Infrastructure as Code) templates. This will ensure rapid system recovery and operational resilience as the user base grows.
- *Data Recovery:* To prepare for enterprise-grade operations, our development trajectory includes the integration of automated backup mechanisms and Point-in-Time Recovery. This feature is prioritized for the post-MVP phase to guarantee maximum data durability.

Expected results of the project
The Vicobi project aims to achieve specific results across two dimensions: Product Deliverables and Team Competency.

Product & Solution

- *Smart & Flexible Data Entry:* We offer a hybrid approach that leverages AI to automate expense classification via voice recordings and invoice images, significantly reducing manual effort. Crucially, the system retains a traditional manual entry interface, ensuring flexibility and control for users who prefer specific adjustments or traditional logging methods.
- *Intuitive & Smart Financial Management:* Provide users with a comprehensive view of their financial health through intuitive charts and periodic reports. Furthermore, the system offers personalized savings suggestions based on the analysis of actual consumer behavior.
- *Streamlined User Experience:* Develop a modern, friendly web interface that is fully responsive on mobile devices. The design focuses on minimalism, making financial management accessible to new users without requiring specialized knowledge.
- *Robust & Scalable Cloud System:* Successfully architect a Microservices-based system on the AWS platform. The infrastructure ensures high stability and is ready for future scalability, such as advanced AI predictive analytics or expansion into a dedicated Mobile App.

Team Competency

Mastery of Cloud Technology & Modern Workflows: Project members gain practical mastery of DevOps processes, implement complete CI/CD pipelines, processing ML models and optimize cloud-native applications on AWS. This serves as a critical foundation for refining technical skills and architectural thinking for their future careers.

2 SOLUTION ARCHITECTURE/ARCHITECTURAL DIAGRAM

2.1 TECHNICAL ARCHITECTURE DIAGRAM

The Development Architecture

The Development Architecture is built on a distributed Microservices model, using an API Gateway as the single entry point. The Frontend is built using Next.js.

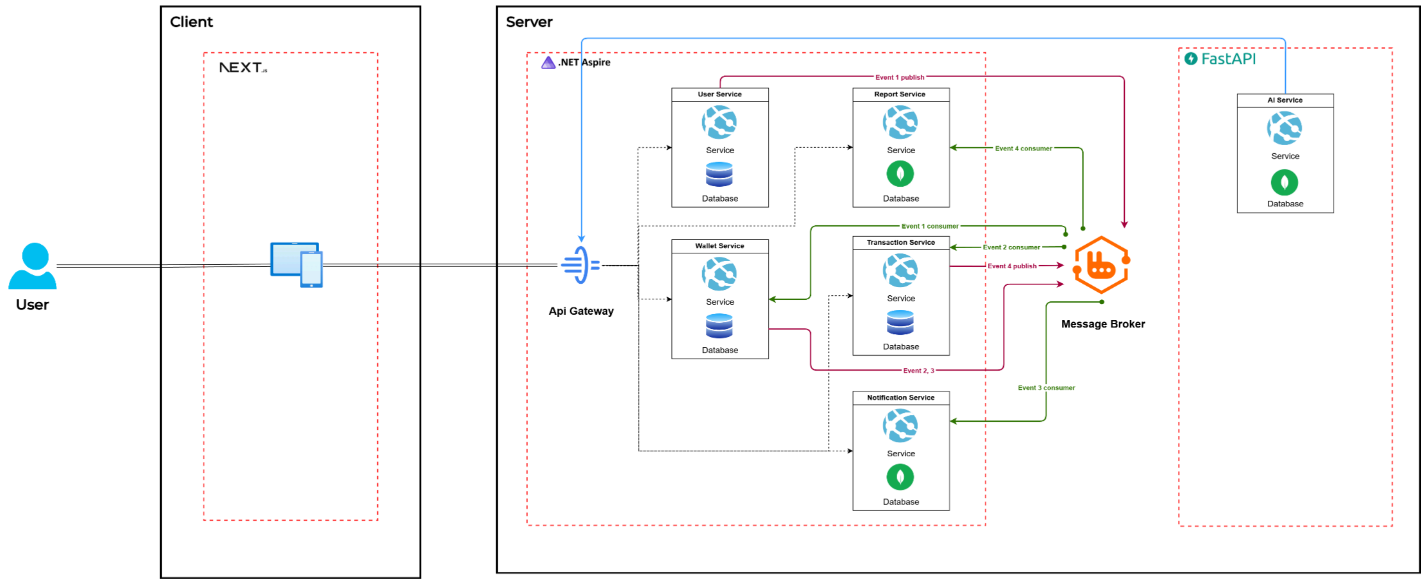

Image 2.1: Vicobi Development Architecture.

Core Backend Microservices (.NET Aspire): These services run on .NET Aspire and are clearly organized by business function:

- User Service: Manages user information.
- Wallet Service: Manages accounts and funds.
- Transaction Service: Handles transaction logging and categorization.
- Report Service: Generates and provides analytical reports.
- Notification Service: Handles notifications and alerts (including Real-Time).

Integration and AI:

- Services communicate with each other via a Message Broker (ensuring asynchronous processing and fault tolerance).
- AI Service (FastAPI): This specialized service handles AI tasks (speech recognition, invoice OCR) and interacts with the Transaction Service via the Message Broker or direct API.
- Note: Each Microservice is depicted with its own Database, reinforcing the Microservices architectural principle.

Tools Used:

Tool Category	Proposed Tools	Purpose
Development	Aspire .NET 9, Python/FastAPI, Next.js	Core technologies for building Microservices and the Frontend.
Code Management CI/CD	GitLab	Automates the container build, push, and deploy pipeline to ECS
Project Management	Gitlab	Planning and tracking progress within the Agile Scrum framework.

The Cloud Architecture

The Cloud Architecture describes the user flow, the CI/CD pipeline, and the AWS components:

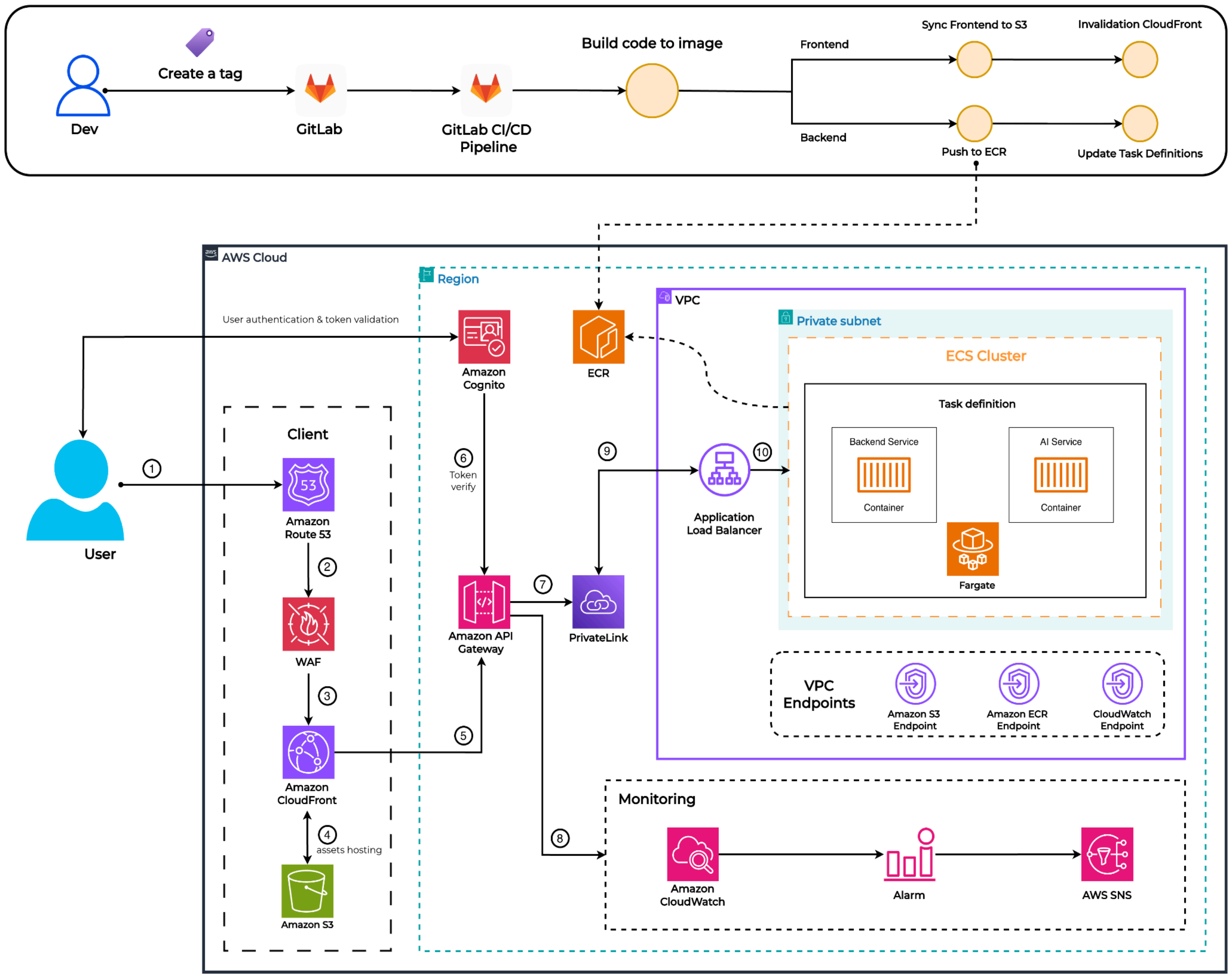

Image 2.2: Vicobi Cloud Architecture.

CI/CD Pipeline (Top of the diagram)

- The process starts when a developer creates a tag on GitLab.
- The GitLab CI/CD Pipeline is automatically triggered.
- Backend: Code is Built into an Image and Pushed to ECR. It then Updates Task Definitions in the ECS Cluster, initiating a new deployment.
- Frontend: Code is Built and Synced to S3. It then performs Invalidation CloudFront to instantly update the content for users.

User Flow (Bottom of the diagram)

- Access: The User (Client) accesses the application (1).
- Routing & Security: The request goes through Amazon Route 53 (2), then AWS WAF (3) for protection against common web attacks, and is distributed by Amazon CloudFront (4).
- Frontend: Static content for the Frontend is served from Amazon S3 (asset hosting).
- Authentication: Dynamic API requests are forwarded to Amazon API Gateway (5). Users are authenticated via Amazon Cognito (6) before accessing resources.

Backend Flow:

- The API Gateway uses a Private Link (7) to securely connect to the Application Load Balancer (ALB) (9) inside the VPC.
- The ALB (9) routes traffic (10) to the ECS Cluster (ECS Fargate) within the Private Subnets.

Logic Processing:

- The ECS Cluster hosts Containers for the Backend Service (the .NET Core Microservices) and the AI Service (FastAPI).
- These services utilize VPC Endpoints to securely access other AWS services (like S3, ECR, CloudWatch) without traversing the public Internet.

Monitoring

- Amazon CloudWatch collects logs and metrics from the entire system (ECS, ALB, etc.).
- When alert thresholds are violated, an Alarm is triggered.
- AWS SNS sends alert notifications to the operations team.

AWS Services Used

- *Amazon Route 53*: Domain name and DNS management.
- *AWS WAF*: Protects the system from common web attacks.
- *Amazon CloudFront*: Global content delivery and frontend acceleration.
- *Amazon S3*: Stores static website content and user files (invoices, audio files).
- *Amazon Cognito*: User authentication and access management.
- *Amazon API Gateway*: Entry point for frontend requests, routing traffic to backend services.
- *AWS PrivateLink*: Private connectivity between API Gateway and ALB within the VPC.
- *Application Load Balancer (ALB)*: Distributes traffic across backend containers on ECS.
- *Amazon ECS (Fargate)*: Runs containerized backend and FastAPI (AI) services.
- *Amazon ECR*: Container image registry for ECS deployment.
- *Amazon CloudWatch*: Centralized logging, monitoring, and alerting.
- *Amazon SNS*: Sends alerts and notifications during incidents.

2.2 TECHNICAL PLAN

Application Architecture and Technology Assignment

The plan focuses on leveraging the high performance of each technology stack across different layers, ensuring security, scalability, and an optimal user experience.

Frontend (FE): Next-Gen UI & Edge Architecture

The Frontend is built on Next.js 16 and AWS Edge services, prioritizing performance, security, and a modern user experience.

- *Core Stack & UI:* Developed with Next.js 16 (App Router) and TypeScript, leveraging React Server Components/Actions. The UI utilizes Tailwind CSS, shadcn/ui, Radix UI, and diverse icon sets (Lucide/React Icons) for consistency and customization.
- *Infrastructure & Distribution:* Static assets are stored in Amazon S3 and distributed globally via Amazon CloudFront using Origin Access Control (OAC) to restrict direct bucket access. AWS Route 53 handles DNS routing to the CDN. The edge layer is secured by AWS WAF against Layer 7 exploits (SQLi, XSS).
- *Data & State Management:*
  - Auth: Integrated with Amazon Cognito (OAuth 2.0/JWT). Tokens are validated by API Gateway via Axios/Fetch requests.
  - State: Employing Zustand (Global State) and React Query (Server State/Caching).
  - Forms: Standardized using React Hook Form with Zod validation.
- *Quality & Optimization:* Code quality is enforced via ESLint, Prettier, and Husky. Performance is optimized for Core Web Vitals through Image Optimization, Code Splitting, and Partial Prerendering.

Backend (BE): Microservices Orchestration and Core Logic

The Backend serves as the central hub for orchestration, business logic, and data management, built on .NET and AWS ECS Fargate.

- *Architecture & Orchestration:* Developed in .NET, utilizing .NET Aspire for configuration management, service discovery, and orchestration of Cloud-native Microservices.
  - *Data & Search Layer: Implements a Database-per-Service pattern:*
  - PostgreSQL & MongoDB: For relational and non-relational business data storage.
  - Elasticsearch: Integrated for high-performance searching and complex transaction queries.
  - Amazon S3: Used for Object Storage (Invoices, Avatars, Voice files).
- *Communication & Background Jobs:*
  - Communication: HTTP for synchronous interactions and RabbitMQ (Message Bus) for asynchronous inter-service communication.
  - Background Jobs: Reliable background task management and scheduling via Hangfire.
  - Notifications: Integrated Amazon SES for transactional emails and alerts.
- Quality Assurance (QA): Ensures high stability through rigorous Unit and Integration Testing using xUnit, the Moq framework and FluentAssertions.

AI Service: Intelligent Processing Pipelines

The specialized AI Service is built using FastAPI/Python and deployed independently on ECS Fargate to handle computationally intensive tasks.

- *Core Framework*: Uses FastAPI to build high-performance APIs that retrieve raw data from S3, perform inference, and return results to the Backend via internal networking.
- *Voice Processing Pipeline:*
    - Preprocessing: Normalizes audio inputs (mp3, aac, etc.) to standard 16kHz mono WAV using Pydub and audioop-lts.
    - Speech-to-Text: Deploys the PhoWhisper-small model (VinAI) [2] on Transformers to transcribe Vietnamese speech to text.
    - Data Extraction (NLU):
        - Local/Dev: Utilizes Google Gemini (Flash model) via API Key for rapid prototyping and cost-efficiency.
        - Deployment (ECS): Switches to Amazon Bedrock (invoking Claude 3.5 Haiku/Sonnet) to ensure data privacy, lower latency within AWS VPC, and enterprise-grade reliability.
- *Bill Processing Pipeline:*
    - Image Preprocessing: Optimizes input images (crop, rotate, contrast) using Pillow and OpenCV-Python.
    - OCR & Extraction:
        - Local/Dev: Uses Google Gemini Vision for experimental extraction and logic validation.
        - Deployment (ECS): Leverages Amazon Bedrock (invoking Claude 3.5 Sonnet Multimodal) for high-precision extraction and seamless integration with S3 data sources.
- *Chatbot Pipeline (RAG):*
    - Knowledge Base Embedding: Ingests and processes financial knowledge documents .
    - Vector Storage: Stores and indexes embeddings in Qdrant, serving as a high-performance vector search engine to retrieve relevant financial contexts based on user queries.
    - Generative Response: Integrates Amazon Bedrock invoking the Claude 3.5 Sonnet model. The pipeline combines retrieved context from Qdrant with Optimized System Prompts to generate accurate, personalized advice on personal financial management (PFM).

Deployment and CI/CD Automation

The entire cloud infrastructure runs inside an Amazon VPC set up in a Multi-AZ model to ensure high availability. Load distribution between the backend and AI containers is managed by an Application Load Balancer (ALB).

The deployment process is fully automated through GitLab CI/CD. Container images (both .NET and FastAPI) are stored in ECR and automatically deployed to ECS Fargate via this pipeline. Comprehensive observability is established using Amazon CloudWatch to monitor logs, performance metrics, and set up proactive alerts.

Security and Access Management

Security controls are tightly implemented across multiple layers. Internal access permissions are precisely defined through IAM Roles for ECS, S3, CloudWatch, and API Gateway, adhering to the Least Privilege principle. Security Groups are strictly configured between ECS, ALB, and other services to control internal traffic. Furthermore, AWS WAF will provide a web-layer defense against common exploits, complementing the robust authentication mechanism provided by Cognito.

2.3 PROJECT PLAN

Methodology & Timeline

- Management Framework: Agile Scrum.
- Development Cycle: 04 Sprints.
- Duration: 01 weeks / Sprint.
- Total Duration: 4 months (including preparation and internship phases).

Detailed Milestones

Pre-internship Phase (Month 0): Kick-off

- Ideation and project scope definition.
- Master Planning.

Internship Phase (Month 1 – Month 4): Core Implementation

- Month 1 (Foundation): Deep dive into AWS services; Upgrade programming skills (Backend .NET, Frontend Next.js, AI Python).
- Month 2 (Design): System architecture design on AWS (High-level & Detailed Design); Technical solution refinement.
- Month 3 – 4 (Realization): Coding; Integration Testing; System Deployment to AWS Production environment; Monitoring setup.

Post-deployment Phase (After Month 5): Expansion

- Research Mobile Development technologies.
- Plan for the Vicobi Mobile App version.

Team Responsibilities & Process

Commitment: All members participate fully in key Scrum events: Sprint Planning, Sprint Review, and Sprint Retrospective.

Role Allocation:

- Product Owner: Aligns product features with the capstone project requirements.
- Scrum Master: Ensures process adherence, resolves technical impediments.
- Development Team: Designs, builds, tests, and operates the system.

Communication & Deliverables

Communication Cadence:

- Daily: Daily Daily Stand-up (15 mins) via Discord for progress updates and blocker resolution.
- Bi-weekly: Review and Retrospective meetings.

Consolidation & Reporting:

- Instead of client handover, Documentation Sessions will be conducted at the end of Month 4.
- Key Deliverables: Final Capstone Report, System Architecture Documentation, and Presentation Slides for defense before the AWS First Cloud Journey Program Committee.

2.4 SECURITY CONSIDERATIONS

The Project Team commits to implementing security measures following the Defense in Depth model and AWS Well-Architected Framework, covering 5 key categories:

Identity & Access Management (Access)

- Authentication: Utilize Amazon Cognito as the Single Source of Truth for user sign-up, sign-in, and JWT Token issuance.
- Token Validation: Token validation is strictly enforced at the application layer. Both the Backend Service (.NET) and AI Service (FastAPI) interact via Middleware to validate the integrity of JWT Tokens from Cognito before processing any requests.
- Authorization: Apply the Least Privilege Principle to all IAM Roles attached to ECS Tasks. Services are granted access only to essential AWS resources (e.g., AI Service can read S3 but cannot delete).

Infrastructure Protection

- Network Design: The system follows a Private Subnet architecture. Logical processing services (ECS Fargate, RabbitMQ) are placed entirely within Private Subnets, possessing no public IP addresses and are inaccessible directly from the Internet. Outbound Internet access is routed via NAT Gateway.
- Edge Protection: Deploy Amazon CloudFront integrated with AWS WAF to protect the application at the edge against common exploits (OWASP Top 10), SQL Injection, and XSS.
- Traffic Control: Strictly configure Security Groups to allow the Application Load Balancer (ALB) to communicate with ECS Containers only on specific ports, blocking all other unauthorized access.

Data Protection

- Encryption in Transit: Enforce HTTPS (TLS 1.2+) for all communication from the Client to CloudFront, and from the API Gateway/ALB to Backend services.
- Secure Storage: Static assets (Frontend Next.js) are hosted on Amazon S3 with "Block Public Access" enabled; users can only access content via CloudFront using Origin Access Identity (OAI).

Detection

- Logging: Enable AWS CloudTrail to log all API call history within the AWS account for auditing purposes.
- Application Monitoring: Use Amazon CloudWatch Logs to collect logs from ECS Fargate (.NET/FastAPI). Anomalies (e.g., CPU spikes, high 5xx error rates) are monitored continuously.

Incident Management

- Automated Alerting: Configure CloudWatch Alarms for critical metrics. When thresholds are breached, Amazon SNS triggers immediate notifications to the development team's Discord channel.
- Response Process: The team has established a rapid response protocol: Receive Discord Alert → Check CloudWatch Logs → Identify Root Cause → Execute Rollback or Hotfix based on severity.

3 ACTIVITIES AND DELIVERABLES

3.1 ACTIVITIES AND DELIVERABLES

Project Phase	Timeline	Activities	Deliverables/Milestones	Total man-day
Assessment	Week 1-4 (Oct)	1. Requirement analysis & User Stories definition. 2. High-level & Low-level AWS Architecture design. 3. UI/UX Prototyping on Figma. 4. Tech Stack selection (Next.js, .NET, FastAPI).	1. SRS Document. 2. Architecture Diagrams. 3. Figma UI/UX Prototypes.	20 MD
Setup base infrastructure	Weeks 1-2 (Nov)	1. Setup VPC, Subnets (Public/Private), Route Tables. 2. Configure IAM Roles & Security Groups. 3. Initialize S3 Buckets, ECR Repositories. 4. Setup local Dev/Staging environments.	1. Fully provisioned AWS VPC Environment. 2. IaC Source Code (AWS CDK). 3. Environment Setup Guide.	10 MD
Agile Development	Dec 2 - Dec 6	Detailed execution across 4 Sprints: *Sprint 1 (Nov 12-16): Core Foundation* ● Auth (Cognito), Wallets, Spending Jars. *Sprint 2 (Nov 17-21): Core Features* ● Transactions (CRUD), AI Voice Processing. *Sprint 3 (Nov 22-26): Analytics* ● Reports/Charts, Notifications (SES).	1. Complete Source Code (BE/FE/AI). 2. Functional Core Features. 3. Test Reports. 4. Working MVP.	80 MD (Est. 4 members)

		Sprint 4 (Nov 27-Dec 1): Stabilization ● Integration Testing, UI Polish.		
Testing & Golive	Dec 2 - Dec 6	1. Deploy to AWS ECS Fargate & CloudFront. 2. Configure Domain (Route 53) & SSL. 3. Setup Monitoring (CloudWatch). 4. User Acceptance Testing (UAT).	1. Live Production System (Vicobi URL). 2. Monitoring Dashboard.	20 MD
Handover	Week 2 (Dec)	1. Draft operational manuals. 2. Consolidate final capstone report. 3. Prepare defense presentation slides.	1. Technical Documentation. 2. Final Project Report. 3. Presentation Deck.	10 MD

3.2 OUT OF SCOPE

While Vicobi has achieved the core objectives of the MVP phase, there remain specific limitations regarding technical implementation and project scope:

AI Processing Limitations

Complexity in Natural Language Processing: Currently, the AI module (leveraging Gemini API for post-processing) performs well with simple commands and standard printed invoices. However, the system faces challenges in semantic separation for complex commands (multi-intent sentences) or processing handwritten/degraded invoices. Voice recognition accuracy for specific regional dialects is not yet optimized.

Third-party Dependency: Reliance on the latency and quota limits of the Gemini API may impact response times during peak usage scenarios.

Platform & Experience Limitations

No Native Mobile Application: In this phase, Vicobi is developed exclusively as a Web Application (Responsive Design). A Native Mobile App version for iOS and Android is out of the current project scope and is roadmap-planned for the next development phase.

Internet Dependency: Being a fully Cloud-native architecture, the application requires continuous Internet connectivity. An "Offline mode" (store-and-forward capability) is not currently supported.

Infrastructure & Security Constraints

Scalability Scope: The current infrastructure architecture is designed and cost-optimized for a Proof of Concept (PoC) scale of approximately 50–100 concurrent users. Scaling to an Enterprise level would require restructuring the Auto-scaling strategy and Database sharding.

Advanced Security Features: The current system focuses on core security measures (Cognito Authentication, Encryption at Rest/In Transit). Advanced security features such as Multi-Factor Authentication (MFA), deep Intrusion Detection Systems (IDS/IPS), or compliance with strict financial standards (like PCI-DSS) are considered out of scope for this MVP version.

3.3 PATH TO PRODUCTION

The current Vicobi system represents a **Minimum Viable Product (MVP)** designed to validate core functionalities—specifically AI-driven financial data entry and microservices orchestration on AWS. While the MVP is functional and stable for a pilot user base (50-100 users), a transition to a full-scale Enterprise Production environment requires addressing specific gaps in scalability, security, and operational excellence.

The following roadmap outlines the necessary steps to elevate Vicobi from MVP to a production-grade fintech application:

Infrastructure & Scalability Hardening

- Current State: The system runs on ECS Fargate with basic auto-scaling configurations. The database uses a single RDS instance.
- Production Gap: Handling sudden traffic spikes (e.g., thousands of concurrent users) requires more aggressive scaling strategies and database read/write separation.
- Path Forward:
    - Database: Migrate to Amazon Aurora Serverless or implement Read Replicas for PostgreSQL/MongoDB to separate read/write traffic.
    - Compute: Fine-tune ECS Service Auto Scaling policies based on custom CloudWatch metrics (e.g., request count per target) rather than just CPU/Memory.
    - Deployment: Transition from Rolling Updates to Blue/Green Deployments via AWS CodeDeploy to ensure zero-downtime releases and instant rollback capabilities.

Advanced Security & Compliance

- Current State: Basic encryption (TLS/KMS) and Cognito authentication are implemented.
- Production Gap: Lack of advanced fraud detection, strict compliance (PCI-DSS), and granular access controls for a financial app.
- Path Forward:
    - MFA Enforcement: Enforce Multi-Factor Authentication (MFA) for all user accounts via Cognito.
    - Secret Management: Fully integrate AWS Secrets Manager with automatic rotation for database credentials and API keys (currently manual/static).
    - Compliance: Implement stricter logging audit trails for financial transactions to prepare for potential PCI-DSS compliance requirements.

Observability & Operational Excellence

- Current State: Basic monitoring via CloudWatch Logs and Metrics.
- Production Gap: Limited visibility into distributed tracing across microservices (.NET to Python/AI), making complex debugging difficult.

- Path Forward:
  - Distributed Tracing: Implement AWS X-Ray integrated with .NET Aspire and FastAPI to trace requests end-to-end through the entire microservices chain.
  - Chaos Engineering: Introduce AWS Fault Injection Service to simulate failures (e.g., AI service downtime) and validate system resilience and self-healing mechanisms.

AI MLOps Pipeline

- Current State: AI models are static and require manual retraining/updating.
- Production Gap: Model drift over time leads to reduced accuracy; manual updates are slow and risky.
- Path Forward:
  - Automated Retraining: Build an MLOps pipeline (using AWS SageMaker or Step Functions) to automatically retrain the OCR/Voice models as new user data is collected and verified.

4 EXPECTED AWS COST BREAKDOWN BY SERVICES

Based on the proposed Microservices architecture and projected usage for the MVP phase, below is the detailed cost analysis.

Link to Estimation: *Refer to the attached pricing.csv file.*

AWS Infrastructure Cost

The estimated monthly AWS cost is **$60.35**, primarily driven by Elastic Load Balancing (ELB) and Compute resources (ECS Fargate).

Services	Component / Usage	Estimated Monthly Cost (USD)	Notes
Elastic Load Balancing (ELB)	Application Load Balancer	$18.98	Primary cost driver, handles traffic distribution to ECS services.
Amazon ECS	Fargate (vCPU & Memory)	$17.30	Runs containerized Backend (.NET) and AI Services (FastAPI).
Amazon VPC	VPC Endpoints & NAT	$10.49	Ensures secure, private network connectivity.
AWS WAF	Web ACL & Requests	$7.20	Web application firewall protecting CloudFront/ALB.
Amazon API Gateway	API Calls & Data Transfer	$2.50	Manages entry-point API traffic.
Amazon CloudFront	Data Transfer Out	$2.00	Global Content Delivery Network (CDN) for Frontend assets.
Amazon ECR	Storage	$1.00	Storage for Docker Container Images.
Amazon Route 53	Hosted Zones	$0.54	Domain name and DNS management.
Amazon S3	Standard Storage	$0.34	Storage for static assets and user data (invoices/voice).
TOTAL AWS COST		**~$60.35 / month**	

AI & Tooling Cost

In addition to AWS infrastructure, the project allocates a budget for specialized AI services and development tools.

Items	Description	Estimated Cost
AI Compute / API	Google Gemini API & Specialized Processing & Amazon Bedrock	~$15.00
Development Tools	GitHub/GitLab, Discord	~$0.00
TOTAL TOOLING COST		**~$15.00 / month**

Total Estimated Project Cost

Category	Monthly Cost
AWS Infrastructure	$60.35
AI & Tooling	$15.00
GRAND TOTAL	**~$75.35 / month**

Assumptions

- **Region:** All services are deployed in the **Asia Pacific (Singapore) - ap-southeast-1** region (based on pricing data).
- **On-Demand Pricing:** Costs are calculated using **On-Demand** rates to maintain flexibility during the MVP phase; Savings Plans or Reserved Instances are not yet applied.
- **Free Tier:** The estimation maximizes the **AWS Free Tier** for eligible services (e.g., S3 storage within limits, initial CloudFront usage), helping to minimize the overall footprint.
- **Traffic:** Estimates are based on an initial user base of 50–100 Concurrent Users (CCU). Actual costs may vary based on real-world Data Transfer Out volume.

5 TEAM

Project Team Members

Name	Title	Role	Email / Contact Info
Lê Vũ Phương Hòa	Backend Developer	Leader	hoalvpse181951@fpt.edu.vn
Nguyễn Văn Anh Duy	AI Developer	Member	duynvase181823@fpt.edu.vn
Uông Tuấn Vũ	Frontend Developer	Member	vuutse180241@fpt.edu.vn
Trần Nguyễn Bảo Minh	AI Developer	Member	baominhbrthcs@gmail.com

Mentor Support

Name	Title	Role	Email / Contact Info
Nguyễn Gia Hưng	Head of Solution Architects	Mentor	hunggia@amazon.com
Văn Hoàng Kha	Cloud Security Engineer	Mentor	khavan.work@gmail.com
FCJers Team	Cloud Engineer	Support	