Hiện đại hóa hệ thống backend cho game di động bằng các dịch vụ đám mây của AWS

*Kiến trúc serverless, an toàn và tiết kiệm chi phí dành cho game thủ thành phát triển trên Unity 6*

**Executive Summary**

Sự phát triển mạnh mẽ của ngành công nghiệp game di động trong thập kỷ qua đã tạo ra nhu cầu bức thiết về những giải pháp backend có khả năng mở rộng, an toàn và tiết kiệm chi phí. Dự án này đề xuất một kiến trúc backend hiện đại, áp dụng các dịch vụ đám mây AWS, dành cho một tựa game thủ thành được phát triển trên nền tảng Unity 6 – trong bối cảnh không còn hỗ trợ SDK chính thức từ AWS.

Game mục tiêu sở hữu hệ thống nâng cấp tháp linh hoạt, dữ liệu người chơi được đồng bộ theo thời gian thực và cần đảm bảo khả năng mở rộng trong tương lai. Giải pháp backend được thiết kế theo mô hình serverless-first, sử dụng Amazon Cognito để xác thực người dùng, Amazon API Gateway và AWS Lambda cho xử lý logic nghiệp vụ, Amazon DynamoDB để lưu trạng thái người chơi dưới dạng JSON, và Amazon S3 cho việc cung cấp tài nguyên tĩnh (hình ảnh, UI…).

Toàn bộ luồng giao tiếp giữa Unity và backend được thực hiện qua HTTP sử dụng JWT hoặc ký bằng AWS Signature Version 4, đảm bảo an toàn mà không cần sử dụng SDK.

Từ góc độ kinh doanh, kiến trúc này giúp giảm chi phí vận hành (dưới 5 USD/tháng cho giai đoạn MVP), rút ngắn thời gian phát triển, và loại bỏ nhuc cầu quản lý hạ tầng. Hệ thống cũng dễ dàng mở rộng thành nền tảng backend dùng chung cho các game di động khác trong tương lai.

Proposal này sẽ trình bày chi tiết bài toán, kiến trúc giải pháp, lộ trình triển khai, chi phí dự kiến, đánh giá rủi ro và lợi ích kỳ vọng. Mục tiêu là chứng minh năng lực thiết kế kỹ thuật lẫn tư duy chiến lược khi áp dụng dịch vụ AWS vào thực tiễn phát triển game di động.

**Mục lục**

[1.Problem Statement 5](#_Toc14452)

[1.1.Current Situation Analysis 5](#_Toc1314)

[1.2.Key Challenges 5](#_Toc22998)

[1.3.Stakeholder Impact 5](#_Toc32496)

[1.4.Business Consequences 6](#_Toc27570)

[2. Solution Architecture 7](#_Toc45)

[2.1. Architecture Overview 7](#_Toc9636)

[2.2. AWS Services Used (Sử dụng API trực tiếp, không dùng SDK) 8](#_Toc29433)

[2.2.1 Amazon Cognito – Authentication & Authorization 8](#_Toc24184)

[2.2.2 Amazon API Gateway – Gateway cho Unity Client 8](#_Toc15218)

[2.2.3. AWS Lambda – Business Logic của Game 9](#_Toc14416)

[2.2.4. Amazon DynamoDB – NoSQL Game State Database 9](#_Toc28612)

[2.2.5. Amazon S3 – Lưu trữ asset tĩnh 11](#_Toc17138)

[2.2.6. Amazon CloudWatch – Ghi log và giám sát 11](#_Toc4542)

[2.3. Component Design 11](#_Toc28038)

[2.3.1. Authentication & Identity Component 12](#_Toc24088)

[2.3.2. PlayerData Component 12](#_Toc11089)

[2.3.3. Upgrade System Component 12](#_Toc9929)

[2.3.4. Team Builder Component 13](#_Toc2859)

[2.3.5. Card Collection Component 13](#_Toc6424)

[2.3.6. Logging & Monitoring Component 13](#_Toc16323)

[2.4. Security Architecture 14](#_Toc5469)

[2.4.1. Xác thực người dùng – Amazon Cognito 14](#_Toc18160)

[2.4.2. Bảo vệ API – API Gateway với JWT Authorizer 14](#_Toc11648)

[2.4.3. Bảo vệ dữ liệu – DynamoDB với IAM Policy theo Identity ID 15](#_Toc4258)

[2.4.4. Quản lý quyền truy cập – IAM Role tách biệt 15](#_Toc18618)

[2.4.5. Monitoring & Logging – CloudWatch và Alarm 16](#_Toc18984)

[2.5. Scalability Design 16](#_Toc7724)

[2.5.1. Mục tiêu 16](#_Toc22763)

[2.5.2. Kiến trúc Serverless giúp scale tự động 16](#_Toc1338)

[2.5.3. Cách xử lý các tình huống scale cụ thể 17](#_Toc10199)

[2.5.4. Kiểm thử hiệu năng và khả năng mở rộng 17](#_Toc20954)

[2.5.5. Mở rộng thông minh với chi phí tối ưu 18](#_Toc17651)

[2.5.6. Dự phòng cho mở rộng tương lai 18](#_Toc25649)

[3. Technical Implementation 19](#_Toc30987)

[3.1. Implementation Phases 19](#_Toc3632)

[3.2. Technical Requirements 20](#_Toc4437)

[3.3. Development Approach 20](#_Toc18199)

[3.4. Testing Strategy 21](#_Toc4975)

[3.5. Deployment Plan 21](#_Toc9217)

[5. Budget Estimation 22](#_Toc5679)

[5.1. Infrastructure Costs 22](#_Toc1513)

[5.2. Development Costs 22](#_Toc3613)

[5.3. Operational Costs 23](#_Toc12492)

[5.4. ROI Analysis 23](#_Toc4750)

[6. Risk Assessment 25](#_Toc23756)

[6.1. Risk Matrix 25](#_Toc2228)

[6.2. Mitigation Strategies 26](#_Toc6533)

[6.3. Contingency Plans 26](#_Toc19494)

[6.4. Tổng kết 27](#_Toc23796)

[7. Expected Outcomes 28](#_Toc26794)

[7.1. Expected Outcomes 28](#_Toc8856)

[7.2. Business Benefits 28](#_Toc22343)

[7.3. Technical Improvements 28](#_Toc25796)

[7.4. Long-term Value 29](#_Toc228)

[8. Appendices 30](#_Toc18732)

[8.1. Technical Specifications 30](#_Toc6924)

[8.2. Cost Calculations 30](#_Toc25398)

[8.3. Architecture Diagrams 32](#_Toc22404)

[8.4. References 35](#_Toc16804)

# 1.Problem Statement

## 1.1.Current Situation Analysis

Phần lớn các trò chơi di động hiện nay, đặc biệt là những sản phẩm phát triển độc lập (indie), thường **dựa vào lưu trữ cục bộ (local)** trên thiết bị hoặc backend tự dựng thiếu tiêu chuẩn. Điều này dẫn đến nhiều **hạn chế về độ tin cậy, bảo mật, và khả năng mở rộng,** nhất là khi số lượng người chơi tăng lên hoặc cần triển khai các chức năng như: đồng bộ hóa dữ liệu người dùng, bảng xếp hạng, hệ thống feedback, phân phối nội dung động,…

Ngoài ra, Unity – công cụ phổ biến để phát triển game – tuy mạnh mẽ ở mặt client, nhưng lại không đi kèm backend mạnh nếu không tích hợp dịch vụ ngoài. Việc hiện đại hóa backend bằng **dịch vụ đám mây của AWS** là xu hướng phù hợp để giải quyết triệt để những bất cập này.

## 1.2.Key Challenges

**Mất dữ liệu người chơi:** Không có cơ chế lưu trữ đám mây khiến người dùng dễ bị mất tiến trình chơi sau khi xóa hoặc đổi thiết bị.

**Khó mở rộng và bảo trì hệ thống:** Backend tự dựng thường khó bảo trì, không có khả năng mở rộng theo số lượng người chơi.

**Thiếu tính năng nâng cao:** Các tính năng hiện đại như hệ thống phản hồi, lưu trữ trạng thái, phân tích hành vi người dùng hoặc gửi thông báo thường không thể tích hợp nếu thiếu backend chuyên nghiệp.

**Khó đảm bảo bảo mật và hiệu suất:** Tự xử lý các vấn đề về bảo mật, xác thực, hoặc hiệu năng dẫn đến lỗ hổng hoặc trải nghiệm không ổn định.

## 1.3.Stakeholder Impact

**Người chơi:** Bị ảnh hưởng trực tiếp nếu dữ liệu bị mất, trải nghiệm thiếu liền mạch và thiếu an tâm khi sử dụng.

**Nhóm phát triển:** Mất nhiều thời gian phát triển các chức năng backend thay vì tập trung vào gameplay hoặc UI/UX; dễ gặp lỗi, khó mở rộng.

**Nhà đầu tư hoặc giảng viên**: Dự án không thể hiện được năng lực kỹ thuật toàn diện nếu thiếu phần backend hiện đại; giảm điểm số đánh giá hoặc cơ hội phát triển thương mại.

## 1.4.Business Consequences

**Nếu không áp dụng giải pháp backend hiện đại:**

* Dự án sẽ thiếu tính hoàn thiện và khó triển khai trong môi trường thực tế.
* Khả năng giữ chân người dùng thấp vì không có trải nghiệm liền mạch và an toàn.
* Hệ thống khó mở rộng trong tương lai, đặc biệt nếu cần đưa lên các kho ứng dụng lớn.
* Không thể tái sử dụng backend cho các sản phẩm game khác trong cùng hệ sinh thái.
* Giảm sức cạnh tranh khi so với các game khác đã tích hợp các công nghệ cloud tiên tiến.

# Solution Architecture

## Architecture Overview

Giải pháp backend cho game di động được xây dựng hoàn toàn dựa trên kiến trúc serverless của AWS, nhằm tối ưu chi phí, dễ dàng mở rộng và giảm thiểu công sức bảo trì. Backend sẽ đảm nhiệm các chức năng chính như:

* Đăng nhập / đăng ký người chơi.
* Lưu trữ và đồng bộ hóa dữ liệu tài khoản, bộ thẻ, chỉ số người chơi.
* Hỗ trợ tính năng đăng nhập đa thiết bị.
* Quản lý phiên bản dữ liệu để đảm bảo gameplay nhất quán.

Kiến trúc tổng thể bao gồm các thành phần sau:

Luồng xử lý tổng quát:

1. Unity Client gửi request thông qua UnityWebRequest.
2. API Gateway tiếp nhận và chuyển request đến Lambda tương ứng.
3. AWS Lambda xử lý logic: xác thực, đọc/ghi dữ liệu người chơi, trả kết quả về client.
4. DynamoDB lưu trữ dữ liệu người dùng: card collection, progress, upgrade history.
5. Amazon S3 chứa dữ liệu tĩnh như hình ảnh thẻ, bản đồ, asset JSON.
6. Amazon Cognito cung cấp đăng ký, đăng nhập, xác thực và sinh token JWT.
7. AWS CloudWatch dùng để log lỗi, theo dõi hiệu năng và cảnh báo khi có vấn đề.

Các nguyên tắc kiến trúc chính:

* Stateless & Decoupled: Các Lambda function không giữ trạng thái, đảm bảo tính độc lập giữa các request.
* Fine-grained Access Control: Sử dụng IAM policies để giới hạn quyền từng service.
* Auto-scaling & Resilience: Lambda, API Gateway và DynamoDB có khả năng scale theo số lượng request.
* Pay-per-use: Toàn bộ dịch vụ chỉ tính phí theo mức độ sử dụng, phù hợp với các dự án đang trong giai đoạn phát triển.

Kịch bản sử dụng điển hình:

Ví dụ: Người chơi đăng nhập vào game lần đầu

* Sau khi xác thực thành công qua Amazon Cognito, Unity client nhận được Access Token (JWT) và Identity ID của người dùng.
* Unity gửi một request POST/GET đến endpoint /playerdata, đính kèm JWT token và identity ID.
* API Gateway kiểm tra tính hợp lệ của token thông qua JWT Authorizer.
* Lambda function HandlePlayerData tiếp nhận và thực hiện logic:
* Nếu đã tồn tại dữ liệu người chơi trong DynamoDB (dựa trên Identity ID):
  + Dữ liệu từ backend sẽ ghi đè lên file JSON hiện tại đang lưu trong client (phòng trường hợp người chơi xoá game và cài lại).
* Nếu chưa có dữ liệu:
  + Dữ liệu hiện tại từ client (một JSON chứa danh sách thẻ cơ bản, chỉ số ban đầu...) sẽ được đẩy lên DynamoDB và tạo bản ghi mới cho người dùng đó.
* Kết quả trả về bao gồm: status, message, data (nếu có), dùng để tiếp tục xử lý hiển thị bên phía Unity.

## AWS Services Used (Sử dụng API trực tiếp, không dùng SDK)

### 2.2.1 Amazon Cognito – Authentication & Authorization

* Sử dụng User Pool để xác thực người dùng: đăng ký, đăng nhập và nhận JWT.
* Unity gửi trực tiếp request HTTP đến endpoints như:
* https://cognito-idp.<region>.amazonaws.com/ để xác thực.
* Sau khi đăng nhập, Unity tiếp tục gửi request tới Cognito Identity Pool để đổi lấy:
* AccessKey, SecretKey và SessionToken tạm thời (bằng lệnh GetId, GetCredentialsForIdentity).
* Các thông tin này sẽ được sử dụng để ký các request tiếp theo (DynamoDB, S3...) bằng Signature V4.

### 2.2.2 Amazon API Gateway – Gateway cho Unity Client

* API Gateway định tuyến request từ client đến các Lambda function backend.
* Tích hợp với JWT Authorizer để xác thực token từ Cognito.
* Unity gửi HTTP requests đến các route như /playerdata, /upgrade, đính kèm JWT và body.
* Cấu hình CORS đầy đủ để Unity có thể request từ nền tảng mobile và WebGL.

### 2.2.3. AWS Lambda – Business Logic của Game

* Lambda thực hiện toàn bộ xử lý như:
  + Tạo hoặc ghi đè dữ liệu người chơi.
  + Tăng cấp thẻ, tính toán chỉ số theo config.
  + Giao tiếp với DynamoDB để truy xuất dữ liệu.
* Unity không gọi trực tiếp Lambda mà đi qua API Gateway để đảm bảo bảo mật và quản lý route.

### 2.2.4. Amazon DynamoDB – NoSQL Game State Database

Cách tổ chức dữ liệu:

* Mỗi người chơi được lưu trong DynamoDB theo định dạng:

{

"TableName": "Aetheric\_PlayerData",

"Item": {

"userId": {"S": "test@gmail.com"},

"gameData": {

"S": "{\"levelProgresses\":[],\"ownedTowerCards\":[...],\"Team\":[...],\"username\":\"\",\"email\":\"\",\"phone\":\"\"}"

},

"lastUpdated": {"S": "2025-01-05T12:30:00Z"}

}

}

* Trong đó:
  + userId: Là định danh duy nhất của người chơi (có thể là email hoặc Identity ID).
  + gameData: Là chuỗi JSON chứa toàn bộ trạng thái gameplay.
  + lastUpdated: Dùng để ghi nhận thời điểm đồng bộ gần nhất, hỗ trợ giải quyết xung đột dữ liệu.

Dữ liệu game ví dụ:

{

"levelProgresses": [],

"ownedTowerCards": [

{

"towerName": "Fire",

"level": 1,

"ownedCards": 0,

"cardsToUnlock": 5,

"cardsToUpgrade": 5

},

...

],

"Team": ["Fire", "Ice", "Poison", "Rock", "Electric"],

"username": "",

"email": "",

"phone": ""

}

Giao tiếp từ Unity:

* Unity sẽ:
  + Gửi PUT/POST request đến DynamoDB qua endpoint chính thức.
  + Ký request bằng AWS Signature Version 4, sử dụng AccessKey, SecretKey, SessionToken từ Cognito Identity Pool.
* Tất cả dữ liệu gameplay được serialize thành JSON string, sau đó đẩy lên DynamoDB trong field "S".

Lưu ý triển khai:

* Vì gameData là một chuỗi JSON, việc update từng trường con (như chỉ nâng cấp tháp "Fire") sẽ cần:
  + Tải toàn bộ bản ghi, chỉnh sửa client-side, rồi ghi đè lại.
  + Hoặc viết Lambda để xử lý riêng phần JSON đó (nếu cần granular update sau này).

### 2.2.5. Amazon S3 – Lưu trữ asset tĩnh

* Trong giai đoạn hiện tại, S3 chỉ được sử dụng để lưu trữ các tài nguyên tĩnh của game như:
  + Hình ảnh minh hoạ cho các thẻ.
  + Asset UI (icon, background…).
  + Tài nguyên âm thanh (nếu có).
* Không sử dụng S3 để lưu trữ JSON cấu hình gameplay hay dữ liệu player, vì các file cấu hình game sẽ được lưu nội bộ trên thiết bị client (Unity local) nhằm tối ưu tốc độ tải và đơn giản hóa luồng đồng bộ.
* Unity sẽ tải các asset này từ S3 thông qua các URL công khai hoặc có thể triển khai pre-signed URL nếu sau này cần quyền truy cập có kiểm soát.
* Cấu hình CORS được thực hiện trên bucket S3 để hỗ trợ WebGL hoặc tải nội dung qua HTTPS.

### 2.2.6. Amazon CloudWatch – Ghi log và giám sát

* Không tương tác trực tiếp từ client.
* Lambda ghi log vào CloudWatch theo từng lần gọi.
* CloudWatch được thiết lập cảnh báo nếu Lambda lỗi liên tục hoặc quá thời gian xử lý.

## Component Design

Kiến trúc backend được chia thành các thành phần logic độc lập (components), mỗi thành phần đảm nhiệm một chức năng cụ thể trong luồng hoạt động của game. Mỗi component sẽ được triển khai thành một hoặc nhiều Lambda Function và được gọi thông qua các route cụ thể trên API Gateway. Toàn bộ hệ thống vận hành trên mô hình stateless, serverless và event-driven.

### Authentication & Identity Component

**Mô tả:** Xác thực người chơi khi đăng nhập hoặc đăng ký game.

**Vai trò:**

* Đăng ký tài khoản người chơi (User Pool).
* Đăng nhập và nhận JWT.
* Lấy temporary AWS credentials qua Identity Pool.

**Tương tác:** Unity gửi request đến Cognito endpoints và nhận JWT + temporary keys.

**Giao tiếp:** Không qua Lambda, trực tiếp gọi API của Cognito.

### PlayerData Component

**Mô tả:** Quản lý dữ liệu game của từng người chơi.

**Chức năng chính:**

* Kiểm tra xem người chơi đã có dữ liệu chưa.
* Nếu có: lấy dữ liệu từ DynamoDB và gửi về client.
* Nếu chưa: tạo mới bản ghi với dữ liệu mặc định gửi từ client.
* Ghi đè dữ liệu mới sau mỗi lần gameplay kết thúc.

**Endpoint chính:**

* GET /playerdata
* POST /playerdata

**Dữ liệu lưu trữ:** Full JSON gameplay, được gói trong field "gameData".

### Upgrade System Component

**Mô tả:** Quản lý logic nâng cấp thẻ (Tower Card).

**Chức năng chính:**

* Kiểm tra số thẻ hiện có của một nguyên tố.
* Cho phép nâng cấp khi đủ điều kiện cardsToUpgrade.
* Ghi lại cấp độ mới của tháp trong ownedTowerCards.

**Tương tác với Unity:** Gửi thông tin tháp cần nâng cấp và cập nhật lại dữ liệu.

### Team Builder Component

**Mô tả:** Lưu và cập nhật đội hình 5 tháp được chọn khi vào trận.

**Chức năng chính:**

* Cập nhật mảng "Team" trong JSON gameData.
* Đảm bảo chỉ các tháp đã mở mới được chọn.

**Endpoint chính:**

* POST /saveteam

**Dữ liệu:** ["Fire", "Ice", "Poison", "Rock", "Electric"]

### Card Collection Component

**Mô tả:** Theo dõi toàn bộ bộ sưu tập tháp của người chơi.

**Chức năng chính:**

* Đọc thông tin từ ownedTowerCards trong JSON gameData.
* Trả danh sách cho Unity để hiển thị giao diện nâng cấp.

**Endpoint chính:**

* GET /cardcollection

### Logging & Monitoring Component

**Mô tả:** Theo dõi và ghi log toàn bộ hoạt động backend.

**Chức năng chính:**

* Ghi log chi tiết mỗi lần gọi API.
* Gửi thông báo cảnh báo nếu gặp lỗi bất thường (qua CloudWatch Alarms).
* Không có endpoint riêng, log được ghi trong Lambda.

## Security Architecture

Kiến trúc bảo mật được thiết kế dựa trên 3 nguyên tắc cốt lõi của hệ thống cloud hiện đại:

* **Zero Trust by Default:** Mọi request đều cần xác thực và được cấp quyền rõ ràng.
* **Least Privilege Access:** Tài khoản tạm thời (temporary credentials) chỉ được phép thao tác trên dữ liệu của chính user đó.
* **Stateless Authorization:** Không lưu session trên server, toàn bộ xác thực thực hiện qua token.

### Xác thực người dùng – Amazon Cognito

Unity thực hiện đăng ký/đăng nhập qua Cognito User Pool, nhận về:

* id\_token (JWT): dùng để xác thực với API Gateway.
* access\_token: dùng để gọi một số API của Cognito nếu cần.

Tiếp theo, Unity gọi Cognito Identity Pool để lấy:

* AccessKey, SecretKey, SessionToken tạm thời.
* Các key này sẽ dùng để ký request khi gửi đến DynamoDB hoặc S3 (nếu có).

Lợi ích:

* Không cần lưu bất kỳ thông tin đăng nhập nào trên client.
* Token có thời hạn ngắn, giảm nguy cơ rò rỉ quyền truy cập lâu dài.

### Bảo vệ API – API Gateway với JWT Authorizer

Mỗi route trong API Gateway (như /playerdata, /upgradecard) đều:

* Bắt buộc đính kèm Authorization: Bearer <JWT> từ Cognito.
* JWT được xác thực bằng JWT Authorizer tại Gateway, kiểm tra:
* Issuer (Cognito URL).
* Audience (App client ID).
* Expiration time.

Chỉ khi token hợp lệ, request mới được chuyển tới Lambda tương ứng.

Lợi ích:

* Không cho phép client chưa xác thực truy cập vào API.
* Không cần viết thêm code xác thực trong Lambda.

### Bảo vệ dữ liệu – DynamoDB với IAM Policy theo Identity ID

IAM Role được gán cho mỗi người dùng (qua Identity Pool) có policy như sau:

{

"Effect": "Allow",

"Action": ["dynamodb:GetItem", "dynamodb:PutItem", "dynamodb:UpdateItem"],

"Resource": "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Aetheric\_PlayerData",

"Condition": {

"ForAllValues:StringEquals": {

"dynamodb:LeadingKeys": ["${cognito-identity.amazonaws.com:sub}"]

}

}

}

Điều này đảm bảo mỗi user chỉ được truy cập dòng dữ liệu có khóa chính là chính họ (theo Identity ID).

**Lợi ích:**

* Ngăn chặn việc một người chơi truy cập hoặc sửa dữ liệu của người khác.
* Không cần viết thêm logic kiểm tra ID trong Lambda.

### Quản lý quyền truy cập – IAM Role tách biệt

IAM Role cho Lambda chỉ có quyền:

* Ghi log lên CloudWatch.
* Truy cập DynamoDB (nếu cần).

IAM Role cho người dùng (Identity Pool) chỉ có quyền thao tác trên chính dữ liệu của họ.

Không dùng quyền \*, luôn giới hạn từng action cụ thể.

### Monitoring & Logging – CloudWatch và Alarm

Mỗi Lambda log toàn bộ request đến và phản hồi trả về.

CloudWatch Log Group theo dõi error rate và latency.

Nếu có lỗi 5xx hoặc timeout tăng đột biến, sẽ kích hoạt CloudWatch Alarm.

## Scalability Design

### Mục tiêu

Thiết kế hệ thống backend có khả năng:

* Xử lý mượt mà khi lượng người chơi tăng từ hàng trăm lên hàng chục ngàn.
* Đáp ứng nhanh với độ trễ thấp dưới 300ms cho các thao tác gameplay như lưu đội hình, nâng cấp tháp.
* Tự động mở rộng và thu hẹp tài nguyên để giảm chi phí vận hành.

### Kiến trúc Serverless giúp scale tự động

Hệ thống tận dụng triệt để mô hình serverless, loại bỏ nhu cầu cấu hình hạ tầng thủ công.

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Cơ chế mở rộng** | **Ghi chú** |
| **API Gateway** | Tự động scale theo lưu lượng | Không giới hạn số lượng concurrent connections |
| **Lambda** | Tự động mở rộng theo số lượng request song song | AWS quản lý container backend |
| **DynamoDB** | Sử dụng On-Demand Capacity Mode | Tự động scale RCU/WCU, không cần cấu hình thủ công |
| **S3** | Phục vụ asset tĩnh qua CDN (CloudFront – nếu dùng sau) | Có thể cache lâu dài, giảm chi phí và độ trễ |

### Cách xử lý các tình huống scale cụ thể

Đợt tăng người chơi do quảng bá game

* Khi số user tăng đột biến (ví dụ do viral marketing hoặc hợp tác phát hành), backend không cần can thiệp thủ công.
* API Gateway và Lambda scale ngay lập tức. DynamoDB on-demand scale gần như real-time.
* Hệ thống vẫn hoạt động ổn định mà không cần DevOps trực ca.

Lượt chơi đồng loạt vào giờ cao điểm

* Lambda có thể xử lý hàng nghìn request/giây, không tạo queue vì bản chất stateless.
* Cần cấu hình limit hợp lý ở API Gateway để tránh overload do spam hoặc bot.
* Có thể tích hợp thêm rate limit + IP throttling nếu cần.

Scale đọc asset – khi Unity tải hình ảnh, icon từ S3

* Nếu triển khai CloudFront, tài nguyên sẽ được phân phối qua CDN, giảm tải cho S3 gốc và tăng tốc độ tải.
* Trường hợp không có CDN, S3 vẫn phục vụ tốt hàng triệu request/ngày mà không cần cấu hình thêm.

### Kiểm thử hiệu năng và khả năng mở rộng

Sử dụng công cụ như Artillery, k6, hoặc Postman Runner để kiểm thử tải (load testing) các endpoint chính: /playerdata, /upgradecard, /saveteam.

Quan sát metrics của Lambda (duration, concurrent executions), DynamoDB (read/write capacity), và CloudWatch Logs.

Thiết lập CloudWatch Alarms để cảnh báo khi:

* Số lượng request thất bại > 5% trong 5 phút.
* Thời gian xử lý Lambda vượt ngưỡng.

### Mở rộng thông minh với chi phí tối ưu

Việc chọn DynamoDB on-demand giúp tiết kiệm chi phí ở giai đoạn đầu (ít người dùng), nhưng vẫn đảm bảo scale tốt.

Lambda chỉ tính phí theo thời gian thực thi, không mất phí khi không có người dùng.

Chưa cần đến ECS/EKS hay EC2 – giảm chi phí quản lý hạ tầng.

### Dự phòng cho mở rộng tương lai

Nếu cần thêm các chức năng như PvP real-time, chat, hay matchmaking:

* Có thể tích hợp Amazon API Gateway WebSocket, Amazon ElastiCache (Redis), hoặc Amazon GameLift tuỳ tính chất gameplay.

Hệ thống hiện tại đã được thiết kế modular, có thể mở rộng từng component riêng biệt.

# 3. Technical Implementation

## 3.1. Implementation Phases

**Giai đoạn 1: Phân tích và Thiết kế hệ thống**

* Xác định rõ các tính năng trong game cần sử dụng backend: đăng ký/đăng nhập người dùng, lưu tiến trình, hệ thống nâng cấp tháp, gửi phản hồi người dùng, và đồng bộ tài nguyên.
* Phân tích từng luồng dữ liệu, luồng người dùng (user flow) trong game.

**Giai đoạn 2: Tích hợp AWS Cognito**

* Thiết lập dịch vụ AWS Cognito để xử lý xác thực (authentication) người dùng: SignUp, Login, Token Refresh.
* Tạo User Pool với các trường dữ liệu cần lưu (email, username, phone number).

**Giai đoạn 3: Lưu trữ dữ liệu người chơi bằng DynamoDB**

* Tạo bảng dữ liệu (table) với UserID là Partition Key, lưu các thông tin như: tiến trình nâng cấp, số lượng mảnh tháp, cấp độ, tài nguyên hiện tại,...

**Giai đoạn 4: Tích hợp hệ thống phản hồi người dùng qua SES**

* Triển khai Lambda function nhận nội dung feedback từ Unity, xử lý và gửi email qua Amazon SES đến đội ngũ phát triển.

**Giai đoạn 5: Phân phối tài nguyên động qua Amazon S3**

* Upload các tài nguyên game như hình ảnh, dữ liệu JSON mô tả màn chơi, thông tin tháp,... lên Amazon S3.

**Giai đoạn 6: Kiểm thử tổng thể và triển khai**

* Thực hiện kiểm thử từng thành phần đã tích hợp trong Unity.
* Kiểm thử các chức năng backend: đăng nhập, lưu dữ liệu, phản hồi, tải tài nguyên.
* Triển khai thử nghiệm nội bộ và tối ưu hiệu suất trước khi hoàn tất bản final.

**Giai đoạn 7 (Dự kiến mở rộng): Tích hợp Translate, Rekognition, Billing Alerts**

* Nâng cao khả năng tự động hóa, cá nhân hóa và giám sát chi phí sử dụng dịch vụ cloud.

## 3.2. Technical Requirements

**Unity 2021 trở lên:** Engine phát triển game client.

**AWS Cognito:** Xác thực người dùng (SignUp, Login, Token).

**Amazon DynamoDB:** Lưu trữ phi quan hệ cho dữ liệu người chơi.

**AWS Lambda + SES:** Gửi phản hồi từ người chơi tới email đội phát triển.

**Amazon S3:** Lưu trữ và phân phối tài nguyên game.

**UnityWebRequest:** Giao tiếp REST API giữa Unity và AWS (không dùng SDK).

**API Gateway:** Làm cầu nối giữa Unity và các hàm Lambda.

**Amazon Translate:** Tự động dịch các nội dung game sang nhiều ngôn ngữ, hỗ trợ đa quốc gia.

**Amazon Rekognition:** Phân tích ảnh người dùng gửi feedback hoặc ảnh đại diện (avatar) để đảm bảo nội dung phù hợp.

**AWS Billing Alerts:** Giám sát và cảnh báo khi chi phí dịch vụ AWS vượt ngưỡng cho phép (ví dụ 10 USD/tháng).

## 3.3. Development Approach

**Modular design:** Mỗi hệ thống backend được phát triển thành module riêng (Auth, Save Data, Feedback…).

**RESTful API communication:** Giao tiếp client-server thông qua chuẩn HTTP bằng UnityWebRequest.

**Không lưu dữ liệu local lâu dài:** Giảm rủi ro mất dữ liệu, đảm bảo đồng bộ cloud-first.

**Tách biệt môi trường phát triển & sản phẩm**: Sử dụng hai hệ thống AWS (dev/prod) nếu cần.

Với các dịch vụ mở rộng (Translate, Rekognition), sẽ triển khai dần theo lộ trình phát triển tiếp theo nếu cần mở rộng ra thị trường quốc tế hoặc triển khai các tính năng AI nâng cao.

## 3.4. Testing Strategy

Unit test từng chức năng backend (gửi dữ liệu, đọc dữ liệu, đăng nhập).

Integration test từ Unity với các dịch vụ AWS.

Test phản hồi người dùng qua hệ thống SES xem có nhận email hay chưa.

Dữ liệu test được ghi log lại để kiểm tra dữ liệu trên DynamoDB/S3.

Kiểm thử UI: kiểm tra các nút đăng nhập, gửi feedback, nhận quà/mảnh có hoạt động đúng quy trình hay không.

Load test nhẹ: gửi nhiều yêu cầu cùng lúc lên API Gateway để kiểm tra độ ổn định.

## 3.5. Deployment Plan

**Triển khai backend AWS theo từng bước:**

1. Tạo User Pool (Cognito)
2. Tạo API Gateway và Lambda cho mỗi chức năng
3. Tạo bảng DynamoDB với partition key theo userID
4. Cấu hình SES + Lambda nhận phản hồi
5. Upload tài nguyên game lên S3

Build và deploy game từ Unity lên thiết bị di động (Android/iOS).

Kiểm tra toàn bộ quy trình từ đăng nhập → lưu dữ liệu → gửi phản hồi → tải tài nguyên từ cloud.

Backup định kỳ dữ liệu DynamoDB nếu mở rộng quy mô.

Tích hợp thêm Billing Alerts giúp đội phát triển kiểm soát chi phí chặt chẽ, đặc biệt trong môi trường học thuật và demo.

# 5. Budget Estimation

## 5.1. Infrastructure Costs

|  |  |  |
| --- | --- | --- |
| **Dịch vụ AWS** | **Miêu tả** | **Ước tính chi phí hàng tháng\*** |
| **AWS Cognito** | Xác thực người dùng (Sign-up, Sign-in) | Miễn phí cho 50.000 MAU đầu tiên |
| **Amazon DynamoDB** | Lưu dữ liệu người chơi: tài khoản, tiến trình, nâng cấp, mảnh shard | ~1–2 USD/tháng (dưới 1GB, low traffic) |
| **Amazon S3** | Lưu asset game như hình ảnh, JSON,... | ~1–2 USD/tháng (tối đa vài trăm MB) |
| **Amazon SES** | Gửi feedback từ người chơi → email nhà phát triển | Miễn phí 62.000 email/tháng (qua EC2) |
| **API Gateway + Lambda** | Làm cầu nối giữa Unity và backend (REST API) | ~2–3 USD/tháng (low usage) |
| **Amazon Translate** | Tự động dịch nội dung game sang các ngôn ngữ khác | ~$15/1 triệu ký tự dịch |
| **Amazon Rekognition** | Phân tích ảnh người chơi gửi lên (tùy chọn mở rộng) | ~$1 mỗi 1.000 ảnh phân tích |
| **AWS Billing Alerts** | Miễn phí giám sát chi phí AWS | $0 |

## 5.2. Development Costs

|  |  |  |
| --- | --- | --- |
| **Hạng mục** | **Chi tiết** | **Ước tính nhân lực/giờ làm việc** |
| Phân tích & thiết kế hệ thống | Lên kiến trúc backend và tính năng | 10–12 giờ |
| Tích hợp Cognito + Login | Đăng ký/đăng nhập từ Unity | 8–10 giờ |
| Lưu dữ liệu với DynamoDB | Lưu trạng thái chơi, nâng cấp | 6–8 giờ |
| Gửi feedback bằng SES | Xây Lambda + Gateway → email | 4–6 giờ |
| Tích hợp Translate, Rekognition (mở rộng) | Gọi API dịch, phân tích ảnh | 6–10 giờ |
| Kiểm thử và tối ưu hóa | Test API, sửa lỗi, tối ưu UI | 8–10 giờ |

**Tổng thời gian phát triển** (ước tính nhóm 1–2 người): **~45–55 giờ làm việc**

## 5.3. Operational Costs

|  |  |  |
| --- | --- | --- |
| **Chi phí** | **Mô tả** | **Ước tính** |
| Chi phí giám sát vận hành | Theo dõi log, chỉnh sửa nhỏ, giám sát billing | ~2 giờ/tháng |
| Chi phí bảo trì | Cập nhật API, tối ưu code nếu có | ~2–3 giờ/tháng |
| Chi phí cloud tăng thêm | Nếu lượng người dùng tăng mạnh đột biến | Có thể lên đến 20–30 USD/tháng |

## 5.4. ROI Analysis

|  |  |
| --- | --- |
| **Yếu tố** | **Lợi ích mang lại** |
| **Tiết kiệm chi phí phát triển backend** | Nhờ sử dụng dịch vụ sẵn của AWS thay vì xây từ đầu |
| **Tăng độ tin cậy và bảo mật** | Giảm rủi ro mất dữ liệu, tăng trải nghiệm người dùng |
| **Khả năng mở rộng cao** | Backend dễ dàng phục vụ 1000+ người chơi nếu cần |
| **Tăng giá trị học thuật & thương mại** | Đề tài có tính ứng dụng cao, phù hợp cả với startup game |
| **Tái sử dụng được hệ thống backend** | Có thể dùng lại cho các game Unity khác |

# Risk Assessment

Hệ thống backend và game client sẽ được triển khai trong môi trường cloud phân tán, có các điểm rủi ro cả về kỹ thuật lẫn vận hành. Phân tích rủi ro dưới đây sẽ giúp chủ động lên kế hoạch xử lý, đảm bảo dự án tiến triển suôn sẻ, hiệu quả và bảo mật.

## Risk Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Rủi ro (Risk)** | **Mức độ ảnh hưởng (Impact)** | **Xác suất xảy ra (Likelihood)** | **Mức độ ưu tiên (Priority)** |
| **API Gateway không phản hồi kịp thời** | Cao | Trung bình | Cao |
| **Lambda timeout do xử lý quá tải** | Cao | Trung bình | Cao |
| **Signature V4 khó tích hợp đúng trong Unity** | Trung bình | Cao | Cao |
| **Token Cognito hết hạn gây lỗi xác thực** | Trung bình | Trung bình | Trung bình |
| **Rò rỉ khóa tạm thời (temporary credentials)** | Rất cao | Thấp | Cao |
| **DynamoDB ghi sai dữ liệu do lỗi client** | Trung bình | Trung bình | Trung bình |
| **Người dùng bị mất dữ liệu do overwrite** | Cao | Trung bình | Cao |
| **Asset trên S3 không đồng bộ với client** | Thấp | Thấp | Thấp |

## Mitigation Strategies

|  |  |
| --- | --- |
| **Rủi ro** | **Giải pháp giảm thiểu** |
| **API Gateway không phản hồi** | Cấu hình throttling hợp lý, thiết lập CloudWatch Alarm theo dõi lỗi 5xx |
| **Lambda timeout** | Tối ưu thời gian xử lý, đảm bảo logic đơn giản, chia nhỏ task nếu cần |
| **Signature V4 khó dùng** | Xây dựng class tái sử dụng để ký request, log kỹ mọi bước gửi request để debug |
| **Token Cognito hết hạn** | Thiết kế cơ chế refresh hoặc relogin tự động, cache thời gian hết hạn token |
| **Rò rỉ khóa tạm thời** | Chỉ cấp IAM Role với quyền tối thiểu, dùng CloudWatch để theo dõi usage bất thường |
| **Lỗi ghi dữ liệu DynamoDB** | Kiểm tra logic PUT vs UPDATE, log lại mọi dữ liệu gửi đi để truy vết nhanh |
| **Overwrite dữ liệu người dùng** | Trước khi lưu phải GetItem → so sánh lastUpdated để tránh overwrite ngược |
| **Asset trên S3 lỗi** | Sử dụng CloudFront và cache header hợp lý, đặt checksum trong client để kiểm tra |

## Contingency Plans

|  |  |
| --- | --- |
| **Rủi ro** | **Phương án dự phòng** |
| **Lambda/API lỗi diện rộng** | Tạm thời chuyển sang thông báo lỗi cho client, bật chế độ chỉ chơi offline |
| **Signature V4 không hoạt động** | Chuyển sang fetch dữ liệu từ S3/local tạm thời, log toàn bộ response để gửi lên support |
| **Token Cognito hết hạn** | Tự động chuyển người dùng về màn hình login với thông báo dễ hiểu |
| **Ghi dữ liệu sai** | Sử dụng cơ chế backup định kỳ trên client (local JSON) để khôi phục thủ công nếu cần |
| **Dữ liệu bị ghi đè** | Ghi lại lịch sử cập nhật (lastUpdated) để debug và rollback thủ công nếu có lỗi logic |

## Tổng kết

Thông qua việc đánh giá và chuẩn bị đầy đủ các rủi ro, dự án đảm bảo được tính ổn định, bảo mật và phục hồi nhanh khi gặp sự cố. Hệ thống được thiết kế với triết lý “Fail Safe + Fail Fast” – nghĩa là dễ phát hiện lỗi, dễ cô lập, và nhanh chóng khôi phục.

# Expected Outcomes

## 7.1. Expected Outcomes

|  |  |
| --- | --- |
| **Mục tiêu** | **Tiêu chí đánh giá cụ thể** |
| **Tích hợp thành công backend AWS** | Game Unity kết nối ổn định với Cognito, DynamoDB, SES, S3 qua REST API |
| **Lưu và truy xuất dữ liệu người chơi** | Dữ liệu tài khoản, tiến trình và nâng cấp được lưu/truy hiệu quả |
| **Hệ thống feedback hoạt động trơn tru** | Gửi phản hồi từ trong game và nhận được email qua AWS SES |
| **Tối ưu chi phí cloud** | Chi phí hạ tầng duy trì ở mức dưới 10 USD/tháng |
| **Tài liệu hóa & demo rõ ràng** | Có tài liệu kỹ thuật, bản demo đầy đủ, test case cho từng chức năng |

## 7.2. Business Benefits

**Nâng cao trải nghiệm người chơi:** Game không còn tình trạng mất dữ liệu, có xác thực tài khoản, đồng bộ tiến trình và phản hồi trực tiếp.

**Tăng khả năng phát hành game ra thị trường quốc tế:** Nhờ tích hợp Amazon Translate, game dễ dàng hỗ trợ đa ngôn ngữ.

**Tăng uy tín sản phẩm và đội ngũ phát triển:** Thể hiện năng lực áp dụng công nghệ cloud thực tế, có thể dùng trong demo gọi vốn hoặc ứng tuyển.

**Chuẩn hóa vận hành & kiểm soát chi phí:** Với Billing Alerts, nhóm phát triển có thể giám sát chi phí và quy mô mở rộng một cách minh bạch.

## 7.3. Technical Improvements

**Áp dụng kiến trúc microservice sử dụng API Gateway + Lambda**, tách biệt rõ ràng client và backend.

**Tối ưu hiệu năng và bảo mật dữ liệu người dùng:** Nhờ AWS Cognito và DynamoDB.

**Mở rộng hệ thống dễ dàng:** Có thể bổ sung thêm chức năng như leaderboard, matchmaking, hoặc phân tích người dùng qua dịch vụ AWS khác.

**Tự động hóa dịch vụ qua cloud:** Không cần host máy chủ riêng, giảm chi phí vận hành và bảo trì hệ thống.

## 7.4. Long-term Value

**Hệ thống backend có thể tái sử dụng cho các game Unity khác:** Chỉ cần chỉnh sửa giao diện phía client.

**Tiền đề cho sản phẩm thương mại hóa:** Có khả năng phát hành lên App Store/Google Play với nền tảng kỹ thuật hoàn chỉnh.

**Cập nhật dễ dàng trong tương lai:** Backend serverless của AWS cho phép chỉnh sửa, bổ sung API mà không ảnh hưởng đến toàn hệ thống.

**Kỹ năng áp dụng công nghệ thực tế**: Dự án giúp nhóm phát triển tích lũy kỹ năng thực chiến với các công cụ cloud hiện đại, đặc biệt là AWS.

# Appendices

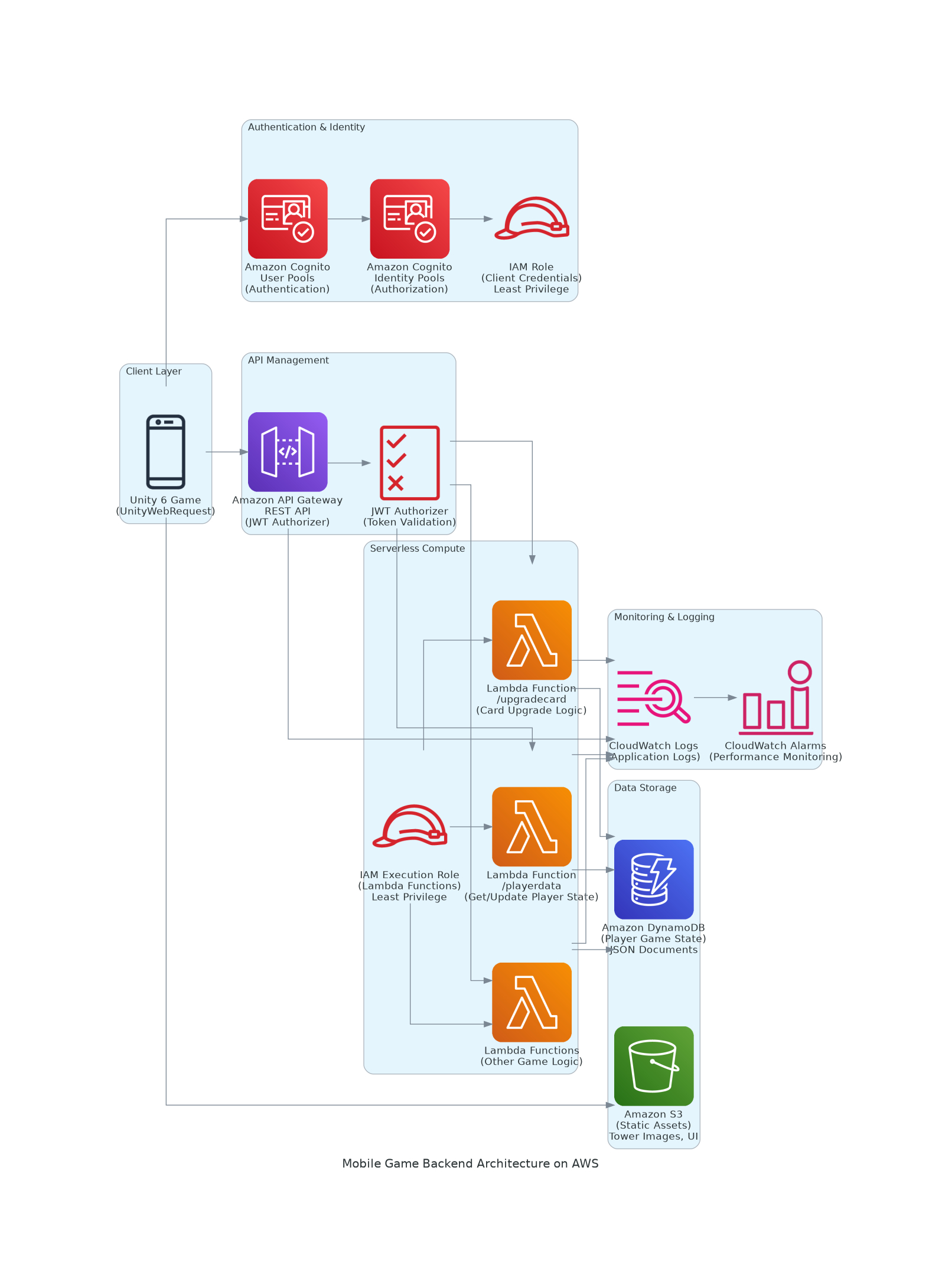
## Technical Specifications

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Công nghệ** | **Mô tả** |
| **Game Engine** | Unity 6 | Giao tiếp với AWS qua UnityWebRequest, không dùng SDK |
| **API Layer** | Amazon API Gateway (REST) | Là cầu nối giữa Unity và Lambda |
| **Backend Logic** | AWS Lambda | Chạy các logic như lưu dữ liệu, nâng cấp thẻ |
| **Database** | Amazon DynamoDB | Lưu gameData dạng JSON, theo userId |
| **Authentication** | Amazon Cognito | Cấp JWT và AWS temporary credentials |
| **Static Content** | Amazon S3 | Lưu ảnh thẻ, asset, bản đồ (không lưu JSON gameplay) |
| **Logging** | Amazon CloudWatch | Ghi log Lambda, giám sát lỗi và cảnh báo |

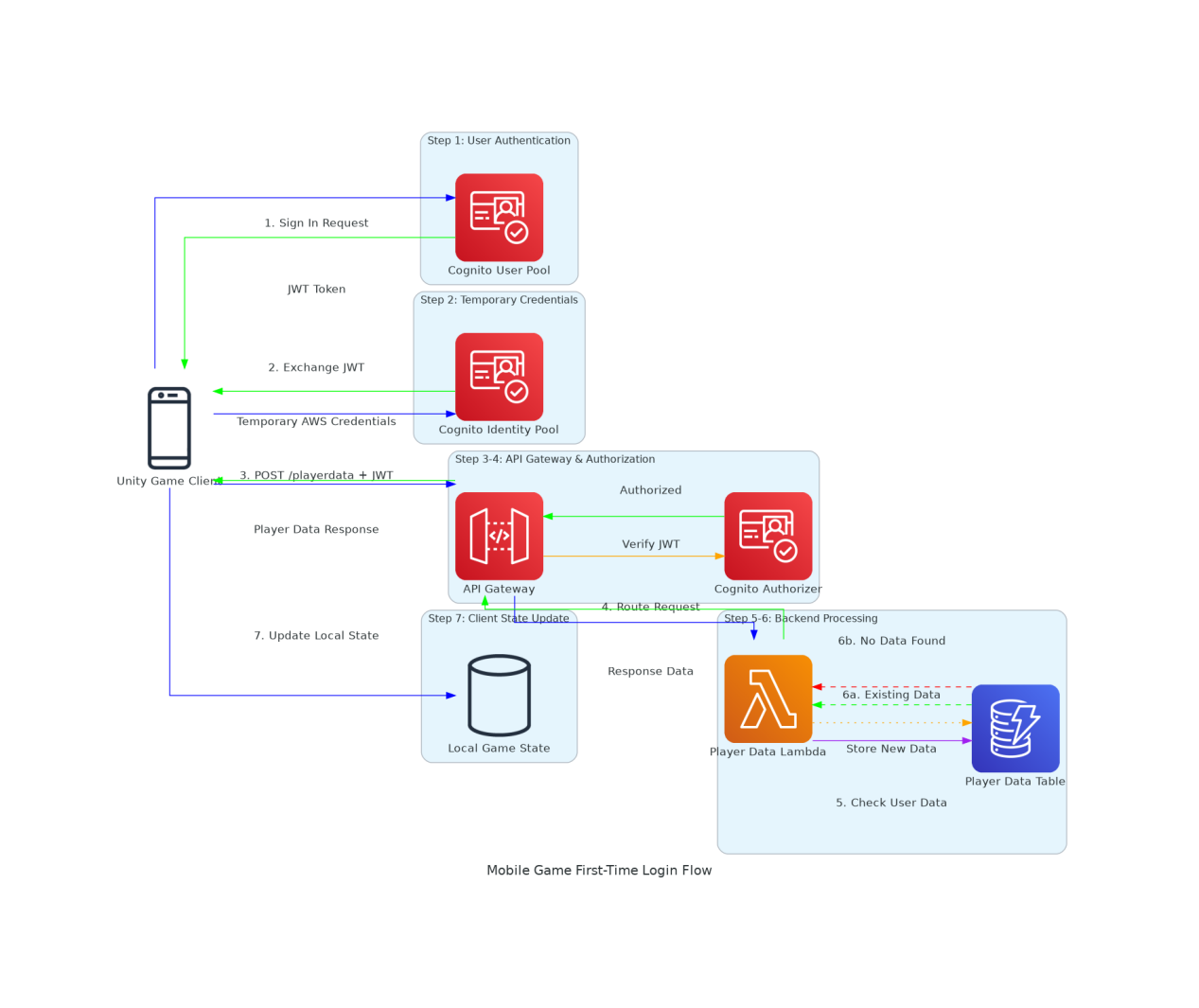
## Cost Calculations

|  |  |  |
| --- | --- | --- |
| **Dịch vụ AWS** | **Miêu tả** | **Ước tính chi phí hàng tháng** |
| **Amazon Cognito** | Xác thực người dùng | Miễn phí (dưới 50K MAU) |
| **Amazon DynamoDB** | Lưu dữ liệu người chơi | ~$1.00 (On-Demand, traffic nhẹ) |
| **Amazon API Gateway** | Giao tiếp REST | ~$1.00 (300K request) |
| **AWS Lambda** | Logic backend | ~$0.20 (300K request, 100ms) |
| **Amazon S3** | Lưu asset | ~$0.50 (500MB + transfer nhỏ) |
| **Amazon SES** | Gửi email feedback | Miễn phí (qua EC2) |
| **Amazon CloudWatch** | Logging và cảnh báo | ~$0.10 (dưới 100MB log) |
| **Translate / Rekognition (mở rộng)** | Dịch nội dung / phân tích ảnh | Chỉ dùng nếu mở rộng |
| **Dự phòng** | Phát sinh tải tăng nhẹ | ~$1.00 |
| **Tổng cộng** | — | ~3.80–4.00 USD/tháng |

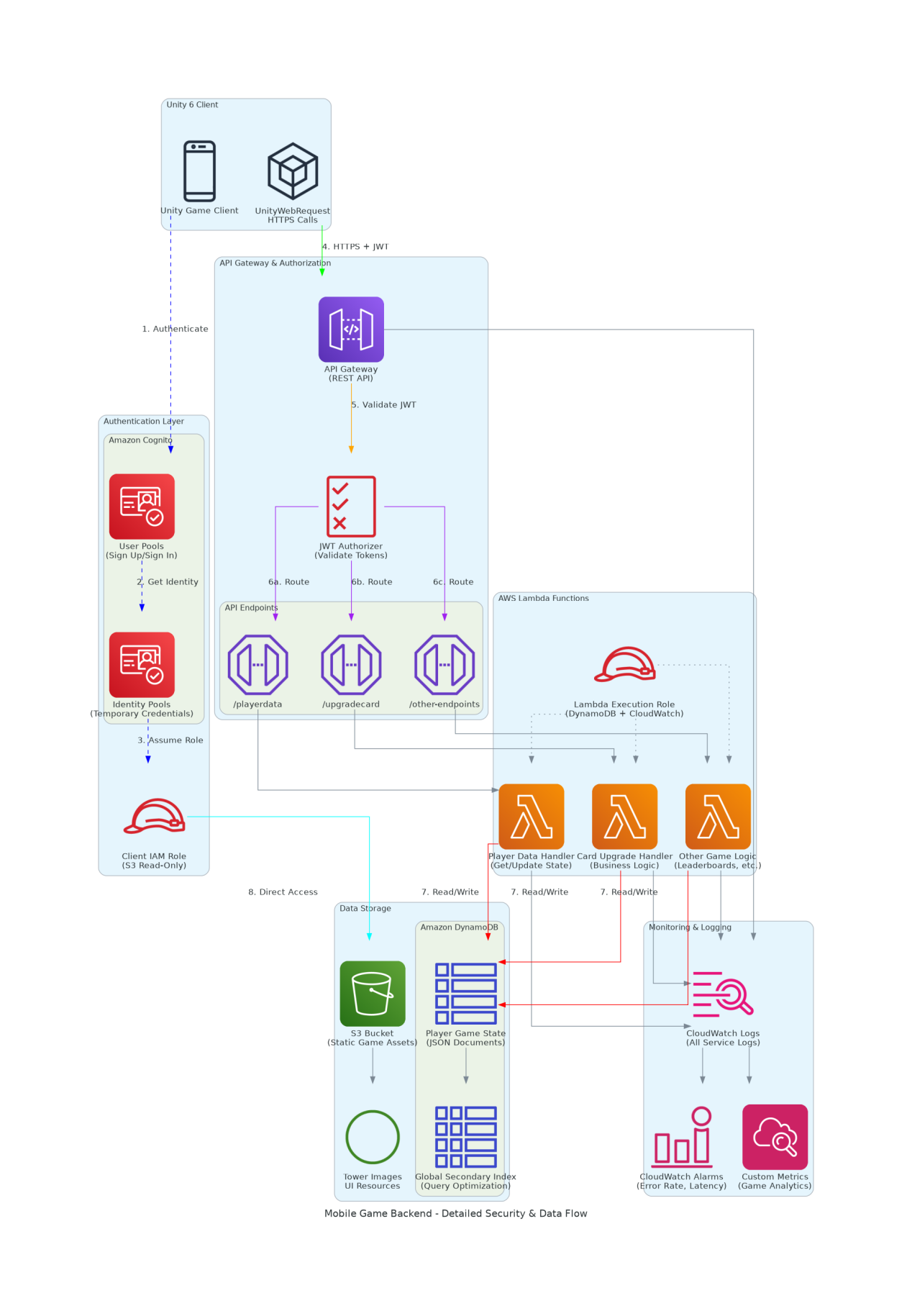
## Architecture Diagrams



Hình 1. Mobile Game Backend Architecture on AWS



Hình 2. Mobile Game First-Time Login Flow



Hình 3. Mobile Game Backend - Detailed Security & Data Flow

**DynamoDB JSON Format**

{

"TableName": "Aetheric\_PlayerData",

"Item": {

"userId": { "S": "test@gmail.com" },

"gameData": {

"S": "{\"levelProgresses\":[],\"ownedTowerCards\":[...],\"Team\":[...],\"username\":\"\",\"email\":\"\",\"phone\":\"\"}"

},

"lastUpdated": { "S": "2025-01-05T12:30:00Z" }

}

}

## References

1. AWS Cognito Documentation – <https://docs.aws.amazon.com/cognito/>
2. AWS Lambda Overview – <https://docs.aws.amazon.com/lambda/>
3. Unity WebRequest Docs – <https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html>
4. AWS Signature Version 4 – <https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>
5. UnityWebRequest Documentation - <https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html>