Technical Architecture & Core System Requirements

1. Core Integration (Second Life ↔ AWS)

Δ Second Life (via LSL scripts) will trigger actions such as:

- User creation and updates
- Creature creation, state updates, aging, hunger, and delivery cycles
- Item drops into a mailbox object in SL
- Payments using Linden Dollars (L$), validated server-side

Δ AWS will act as the authoritative backend for:

- Game logic validation
- State persistence
- Security and anti-exploit checks

2. Data Model (DynamoDB)

Δ Each entity will have a UUID and lifecycle tracking:

- Users (mapped to SL avatars)
- Creatures (age, hunger, delivery timer, death state)
- Items (drops, collectibles, food, tokens)
- Transactions (L$ payments, rewards, Great Beyond points)
- Mailbox events (who dropped what, when)

Δ DynamoDB is well-suited here for:

- High write frequency from SL

- Time-based state changes (TTL, scheduled Lambdas)

- Scalable creature and item storage

3. Creature Lifecycle Logic

Δ Creatures live for 100 days

Δ Stats tracked:

- Munchiez (hunger)

- Age

- Delivery / fervor

Δ Deliveries:

- Normal random delivery every 12 days

- Items dropped via mailbox object in SL

Δ End of life:

- Creature becomes inactive (“blacked out”)

- User options:
  - Send to Great Beyond (points)
  - Convert to Headstone or Casket (collectible / tradable / sellable)
  - Optional later conversion to points

I think all this logic should be validated server-side to prevent manipulation.

4. Payment Flow (Linden Dollars)

Δ L$ payments initiated in SL

Δ Transaction data sent to AWS
Δ Server-side validation before:

- Issuing items

- Unlocking creatures

- Updating stats

This feature against spoofed payments or replay attacks.

5. Authentication & Security

Δ Since SL has limitations, I would suggest:

- Per-object or per-user API keys

- Signed requests with timestamps

- Rate limiting at API Gateway

- Server-side sanity checks for all stat changes

This keeps the system secure without making LSL overly complex.

6. Web Portal — MVP

Δ Simple frontend (React / Next.js or similar)

Δ API-driven

Δ No heavy animations

Δ No advanced trading UI