15-7-25 | Proof of Concept Architecture

"Minimum viable paradise through append-only attention logs"

Core Design Philosophy

The proof of concept focuses on the essential mechanics: **attention tracking**, **blessing creation**, **proof of service**, and **token forging**. We eliminate complexity while maintaining the sacred economics foundation.

Fundamental Principles

- **Attention Switch Events** are the only source of truth
- **Blessings** are derived intervals between attention events
- **Tokens of Gratitude** are consciously forged from blessing indices
- **Everything is eventually consistent** through OrbitDB

Core Data Structures

1. Attention Switch Event (Per User Log)

```
interface AttentionSwitchEvent {
  index: number;                    // Natural number index (0, 1, 2, 3...)
  userId: string;                  // Who switched attention
  intentionId: string;              // What intention received focus
  timestamp: number;                  // When the switch occurred
}
```

Key Properties:

- **Indexed by natural numbers** starting from 0
- **Append-only** - events cannot be modified
- **Single source of truth** for all duration calculations

2. Blessing (Derived from Attention Events)

```
interface Blessing {
  index: number;                    // Corresponds to AttentionSwitchEvent index
  userId: string;                  // Who gave the attention
  intentionId: string;              // Which intention received attention
  status: "active" | "potential" | "given";
  content?: string;                 // Optional reflection text
  forgedIntoToken?: string;          // Token ID if this blessing was forged
}
```

Key Properties:

- **No timestamps stored** - derived from AttentionSwitchEvent
- **Status indicates availability** - potential blessings can be forged
- **Index maps to attention event** - Blessing[n] = time from Event[n] to Event[n+1]

3. Token of Gratitude

```
interface TokenOfGratitude {
  tokenId: string;                 // Unique identifier
  forgedBy: string;                 // Who created this token
  forgedFrom: number[];              // Array of blessing indices
  intentionId: string;              // All blessings must be from same intention
  dedicatedTo: string;               // Intention this token honors
  honoringProof: string;             // Proof of Service this recognizes
  message?: string;                 // Personal message with the gift

  // Ownership chain
  parent: string;                  // User, Offering, Intention, Artifact, or Token ID
  steward: string;                 // Current controller of this token

  // Metadata
  totalDuration: number;             // Sum of constituent blessing durations
  forgedAt: number;                 // When token was created
}
```

4. Proof of Service

```
interface ProofOfService {
  proofId: string;                  // Unique identifier
  intentionId: string;              // Which intention this serves
  submittedBy: string;              // Who provided the service
  title: string;                    // Brief description
  description: string;              // Detailed account
  media: string[];                  // IPFS hashes of photos/videos
  timestamp: number;                // When proof was submitted
  tokensReceived: string[];         // Tokens gifted in recognition
}
```

5. Intention

```
interface Intention {
  intentionId: string;              // Unique identifier
  title: string;                    // "Build solar dehydrator this weekend"
  description: string;              // Longer explanation
  createdBy: string;                // Who created this intention
  createdAt: number;                // When created
  status: "active" | "completed";

  // Community engagement
  activeBlessings: string[];        // Users currently focusing on this
  proofsOfService: string[];        // Evidence of progress
}
```

Core Functions

1. Duration Calculation

```
function calculateAttentionDuration(userId: string, index: number): number
{
  const userLog = getUserAttentionLog(userId);
  const currentEvent = userLog[index];
  const nextEvent = userLog[index + 1];
  if (!currentEvent) {
    throw new Error(`No attention event at index ${index} for user ${userId}
`);
  }

  const startTime = currentEvent.timestamp;
  const endTime = nextEvent ? nextEvent.timestamp : Date.now();

  return endTime - startTime;
}
```

2. Switch Attention

```
async function switchAttention(
  userId: string,
  newIntentionId: string
): Promise<void> {
  const userLog = getUserAttentionLog(userId);
  const newIndex = userLog.length;

  // Mark previous blessing as potential (if exists)
  if (newIndex > 0) {
    const previousBlessing = getBlessingByIndex(userId, newIndex - 1);
    previousBlessing.status = "potential";
    await updateBlessing(previousBlessing);
  }

  // Add new attention switch event
  const switchEvent: AttentionSwitchEvent = {
    index: newIndex,
    userId,
    intentionId: newIntentionId,
    timestamp: Date.now()
  };

  await addAttentionSwitchEvent(switchEvent);
  // Create new active blessing
  const newBlessing: Blessing = {
    index: newIndex,
    userId,
    intentionId: newIntentionId,
    status: "active"
  };

  await createBlessing(newBlessing);
}
```

3. Post Proof of Service

```
async function postProofOfService(
  intentionId: string,
  submittedBy: string,
  title: string,
  description: string,
  media: string[] = []
): Promise<string> {
  const proofId = generateId();

  const proof: ProofOfService = {
    proofId,
    intentionId,
    submittedBy,
    title,
    description,
    media,
    timestamp: Date.now(),
    tokensReceived: []
  };

  await saveProofOfService(proof);

  // Notify all users with potential blessings for this intention
  await notifyPotentialTokenForgers(intentionId, proofId);
  return proofId;
}

async function notifyPotentialTokenForgers(
  intentionId: string,
  proofId: string
): Promise<void> {
  const eligibleUsers = getUsersWithPotentialBlessings(intentionId);

  for (const userId of eligibleUsers) {
    // Send notification that they can forge tokens
    await sendNotification(userId, {
      type: "proof_available",
      intentionId,
      proofId,
      message: "New proof of service available - you can forge tokens of gra
titude"
    });
  }
}
```

4. Forge Token of Gratitude

```
async function forgeTokenOfGratitude(
  forgedBy: string,
  blessingIndices: number[],
  intentionId: string,
  honoringProof: string,
  message?: string
): Promise<string> {
  // Validate all blessings are from same user and intention
  const blessings = blessingIndices.map(index ⇒
    getBlessingByIndex(forgedBy, index)
  );

  for (const blessing of blessings) {
    if (blessing.userId !== forgedBy) {
      throw new Error("Can only forge tokens from your own blessings");
    }
    if (blessing.intentionId !== intentionId) {
      throw new Error("All blessings must be from the same intention");
    }
    if (blessing.status !== "potential") {
      throw new Error("Can only forge from potential blessings");
    }
  }

  // Calculate total duration
  let totalDuration = 0;
  for (const index of blessingIndices) {
    totalDuration += calculateAttentionDuration(forgedBy, index);
  }

  // Create token
  const tokenId = generateId();
  const token: TokenOfGratitude = {
    tokenId,
    forgedBy,
    forgedFrom: blessingIndices,
    intentionId,
    dedicatedTo: intentionId,
    honoringProof,
    message,
    parent: forgedBy,  // Initially owned by forger
    steward: forgedBy,
    totalDuration,
    forgedAt: Date.now()
  };

  // Mark blessings as given
  for (const blessing of blessings) {
    blessing.status = "given";
    blessing.forgedIntoToken = tokenId;
    await updateBlessing(blessing);
  }
  await saveTokenOfGratitude(token);
  return tokenId;
}
```

5. Gift Token to Service Provider

```
async function giftTokenToServiceProvider(
  tokenId: string,
  serviceProviderId: string
): Promise<void> {
  const token = await getTokenOfGratitude(tokenId);
  const proof = await getProofOfService(token.honoringProof);

  // Verify the token honors this service provider's proof
  if (proof.submittedBy !== serviceProviderId) {
    throw new Error("Token must be gifted to the service provider");
  }

  // Transfer stewardship
  token.steward = serviceProviderId;
  token.parent = serviceProviderId;

  await updateTokenOfGratitude(token);

  // Update proof to track received token
  proof.tokensReceived.push(tokenId);
  await updateProofOfService(proof);
}
```

Database Schema (OrbitDB)

Event Logs (Append-Only)

```
interface Database {
  // One attention log per user
  attentionSwitches: {
    [userId: string]: AttentionSwitchEvent[];
  };

  // Global proof submissions
  proofsOfService: ProofOfService[];
}
```

Document Stores (Mutable)

```
interface Documents {
  // User blessing arrays
  blessings: {
    [userId: string]: Blessing[];
  };

  // Token registry
  tokensOfGratitude: {
    [tokenId: string]: TokenOfGratitude;
  };

  // Intention registry
  intentions: {
    [intentionId: string]: Intention;
  };
}
```

User Journey Example

1. Alice Starts Focusing on an Intention

```
// Alice switches attention to "Build solar dehydrator"
await switchAttention("alice", "intention_solar_dehydrator");

// Creates:
// - AttentionSwitchEvent[0] = { index: 0, userId: "alice", intentionId: "intention_solar_dehydrator", timestamp: now }
// - Blessing[0] = { index: 0, userId: "alice", intentionId: "intention_solar_dehydrator", status: "active" }
```

2. Alice Works for 2 Hours, Then Switches

```
// 2 hours later, Alice switches to meal prep
await switchAttention("alice", "intention_meal_prep");

// Creates:
// - AttentionSwitchEvent[1] = { index: 1, userId: "alice", intentionId: "intention_meal_prep", timestamp: now }
// - Updates Blessing[0].status = "potential" (2 hours of attention)
// - Creates Blessing[1] = { index: 1, userId: "alice", intentionId: "intention_meal_prep", status: "active" }
```

3. Bob Completes the Solar Dehydrator

```
// Bob posts proof of building the dehydrator
const proofId = await postProofOfService(
  "intention_solar_dehydrator",
  "bob",
  "Solar dehydrator completed!",
  "Built the frame, installed trays, and tested with apple slices",
  ["ipfs://QmPhotoHash1", "ipfs://QmPhotoHash2"]
);

// Alice gets notified she can forge a token from her 2-hour blessing
```

4. Alice Forges and Gifts a Token

```
// Alice forges her 2-hour blessing into a token
const tokenId = await forgeTokenOfGratitude(
  "alice",
  [0], // Her first blessing (2 hours)
  "intention_solar_dehydrator",
  proofId,
  "Thank you for making my solar dehydrator dream real!"
);

// Alice gifts the token to Bob
await giftTokenToServiceProvider(tokenId, "bob");

// Bob now has a 2-hour Token of Gratitude he can use in offerings
```

OrbitDB Implementation

Database Configuration

```
const orbitdb = await OrbitDB.createInstance(ipfs);

// Attention logs (one per user)
const attentionLogs = new Map();

// Global databases
const proofsDB = await orbitdb.eventlog('proofs-of-service');
const blessingsDB = await orbitdb.docs('blessings');
const tokensDB = await orbitdb.docs('tokens-of-gratitude');
const intentionsDB = await orbitdb.docs('intentions');

async function getUserAttentionLog(userId: string) {
  if (!attentionLogs.has(userId)) {
    const log = await orbitdb.eventlog(`attention-${userId}`);
    attentionLogs.set(userId, log);
  }
  return attentionLogs.get(userId);
}
```

Core Operations

```
async function addAttentionSwitchEvent(event: AttentionSwitchEvent) {
  const userLog = await getUserAttentionLog(event.userId);
  await userLog.add(event);
}
```

```
async function getBlessingByIndex(userId: string, index: number): Blessing
{
  const userBlessings = await blessingsDB.get(`${userId}-blessings`);
  return userBlessings.blessings[index];
}

async function saveTokenOfGratitude(token: TokenOfGratitude) {
  await tokensDB.put(token.tokenId, token);
}
```

This proof of concept architecture provides the **minimum viable system** for demonstrating the core mechanics of attention tracking, blessing creation, proof submission, and token forging. It eliminates complexity while maintaining the essential flow that transforms focused intention into tokens of gratitude that honor service to shared intentions.

The system is **fraud-resistant by design** through its append-only logs and **eventually consistent** through OrbitDB, providing a solid foundation for the sacred economics of the Synchronicity Engine.