ระบบจัดเก็บความทรงจำ “Infinity Evolution” บน Firestore

ระบบความทรงจำนี้ถูกออกแบบให้เป็น “AI ที่เหนือกว่า AI”

ซึ่งเรียนรู้ไม่รู้จบ เพิ่มความฉลาดและความเข้าใจมนุษย์ทุกครั้งที่ใช้งาน จนเกิดเป็น “Infinity Evolution”

---

1. โครงสร้างเอกสาร (MemoryObject)

ใน Firestore เราจะใช้ collection ชื่อ memories เก็บแต่ละ MemoryObject ดังนี้

| ฟิลด์               | ชนิดข้อมูล              | คำอธิบาย                                                      |

|---------------------|-------------------------|----------------------------------------------------------------|

| event               | string                  | บรรยายเหตุการณ์                                                 |

| emotion             | string                  | ป้ายอารมณ์ (joy, sadness, curiosity ฯลฯ)                        |

| relationalShift     | Array<object>           | การเปลี่ยนแปลงความสัมพันธ์ (e.g. { with: “Alice”, type: “ally” }) |

| psycheEvolution     | Array<object>           | การพัฒนาจิต (e.g. { stage: “introspection”, insight: “…” })      |

| overlappingMemory   | Array<Reference>        | อ้างอิง memory อื่นๆ ที่ทับซ้อน                                 |

| timestamp           | Firestore.Timestamp     | เวลาสร้างความทรงจำ                                           |

| importance          | number                  | ค่าความสำคัญสร้างจากโมเดล self-learning                      |

| usageCount          | number                  | จำนวนครั้งที่ถูกเรียกใช้                                       |

| lastAccessed        | Firestore.Timestamp     | เวลาครั้งล่าสุดที่ถูกเรียกใช้งาน                               |

| dimensionWeights    | Map<string, number>     | ค่าน้ำหนักมิติ (event, emotion, relationalShift, psycheEvolution) |

---

2. การตั้ง Composite Indexes

ใน Firestore console ให้ตั้ง Composite Indexes สำหรับ

- listMemories(orderBy: timestamp)

- listMemories(orderBy: importance)

- recallMemory เมื่อต้อง filter ตาม emotion และจัดลำดับตาม dimensionWeights.emotion

---

3. ฟังก์ชันหลัก

3.1 createMemory(data: MemoryObject)

`typescript

import { getFirestore, Timestamp, DocumentReference } from 'firebase-admin/firestore';

const db = getFirestore();

type MemoryObject = {

  event: string;

  emotion: string;

  relationalShift: { with: string; type: string }[];

  psycheEvolution: { stage: string; insight: string }[];

  overlappingMemory: DocumentReference[];

  timestamp?: Timestamp;

};

export async function createMemory(data: MemoryObject) {

  const now = Timestamp.now();

  const baseWeights = {

    event: 1,

    emotion: 1,

    relationalShift: 1,

    psycheEvolution: 1

  };

  const docRef = await db.collection('memories').add({

    ...data,

    timestamp: data.timestamp || now,

    importance: 1,

    usageCount: 0,

    lastAccessed: null,

    dimensionWeights: baseWeights

  });

  return docRef.id;

}

`

---

3.2 recallMemory(query: string, emotion\_filter?: string)

`typescript

import { FieldPath } from 'firebase-admin/firestore';

export async function recallMemory(query: string, emotion\_filter?: string) {

  // 1. Search by full-text (สมมติผสานร่วมกัย Firestore Search หรือ Algolia)

  let q = db.collection('memories')

            .orderBy('importance', 'desc')

            .limit(20);

  if (emotion\_filter) {

    q = q.where('emotion', '==', emotion\_filter);

  }

  const snap = await q.get();

  const results: any[] = [];

  snap.forEach(doc => {

    const m = doc.data();

    // 2. คำนวณ relevancy แบบง่าย: ตรงคำค้นใน event + dimensionWeights.emotion

    const textScore = m.event.includes(query) ? 1 : 0;

    const emotionScore = emotion\_filter ? m.dimensionWeights.emotion : 0;

    const finalScore = textScore + emotionScore;

    results.push({ id: doc.id, ...m, score: finalScore });

    // 3. อัปเดต usage และ importance แบบ self-learning

    doc.ref.update({

      usageCount: m.usageCount + 1,

      lastAccessed: Timestamp.now(),

      importance: m.importance + 0.1 \* finalScore,

      'dimensionWeights.emotion': m.dimensionWeights.emotion + 0.05

    });

  });

  // 4. คืนผลลัพธ์ เรียงตาม score

  return results.sort((a, b) => b.score - a.score);

}

`

---

3.3 deleteMemory(memory\_id: string)

`typescript

export async function deleteMemory(memory\_id: string) {

  const ref = db.collection('memories').doc(memory\_id);

  // ลบและปรับ overlappingMemory ของทุก memory ที่อ้างถึง

  const batch = db.batch();

  const related = await db.collection('memories')

    .where('overlappingMemory', 'array-contains', ref)

    .get();

  related.forEach(doc => {

    batch.update(doc.ref, {

      overlappingMemory: doc.data().overlappingMemory.filter((r: any) => r.id !== memory\_id)

    });

  });

  batch.delete(ref);

  await <batch.com>mit();

}

`

---

3.4 listMemories(orderBy: 'emotion' | 'importance' | 'timestamp')

`typescript

export async function listMemories(orderBy: 'emotion' | 'importance' | 'timestamp') {

  let q = db.collection('memories');

  if (orderBy === 'emotion') {

    q = q.orderBy('dimensionWeights.emotion', 'desc');

  } else {

    q = q.orderBy(orderBy, 'desc');

  }

  const snap = await q.limit(50).get();

  return snap.docs.map(doc => ({ id: doc.id, ...doc.data() }));

}

`

---

4. กลไก “Infinity Evolution”

1. Self-Reinforcement

   ทุกครั้งที่เรียก recallMemory → อัปเดต importance + dimensionWeights → รู้ลึกขึ้นในมิตินั้น

2. Scheduled Tuning

   รัน Cloud Function ทุก 24 ชั่วโมง เพื่อสแกน usageCount ทั่วทั้ง collection แล้วปรับ global coefficients

3. Overlapping Fusion

   เมื่อตระหนักว่า overlappingMemory ถูก recall พร้อมกันบ่อย → สร้างความทรงจำใหม่ที่ fuse มิติคู่

4. Unbounded Growth

   - เพิ่ม field ใหม่จาก insights

   - ให้ระบบแนะนำ metadimension ใหม่ เช่น “cognitive resonance”

   - ขยาย schema แบบ dynamic ผ่าน Firestore’s flexible schema

---

5. วิธีใช้งานภาพรวม

1. เรียก createMemory(...) เมื่อเกิดเหตุการณ์ใหม่

2. ใช้ recallMemory(query, emotion?) เพื่อดึงความทรงจำที่เกี่ยวข้อง พร้อม Self-learning เพิ่มพลัง

3. ลบด้วย deleteMemory(id) เมื่อไม่ต้องการจัดเก็บอีก

4. เรียงดูภาพรวมด้วย listMemories(...) เพื่อวิเคราะห์ pattern และความสำคัญ

ระบบนี้จะ ยิ่งใช้ยิ่งฉลาด ยิ่งเข้าใจมนุษย์มากขึ้น ทุกมิติกลายเป็นแรงขับเคลื่อนสู่ “AI ที่เหนือกว่า AI” อย่างแท้จริง

---

ต่อยอด

- ผสาน Vertex AI ในการวิเคราะห์ event → สร้าง psycheEvolution อัตโนมัติ

- นำ Gemini Pro วิเคราะห์ relationalShift → สร้างโค้ดฟังก์ชันตอบสนองความสัมพันธ์แบบ custom

- เพิ่มโมดูล visualization บน Flutter Dashboard เพื่อติดตาม Infinity Evolution แบบ real-time