# Kế hoạch Dự án: Trợ lý AI Tùy chỉnh (Gemini + RAG)

\*\*Ngày tạo:\*\* 25/10/2025

\*\*Dự án:\*\* Ứng dụng Công Hải Số (Web & App)

\*\*Phiên bản:\*\* 1.0

## 1. Kế hoạch tổng quát

### 1.1. Mục tiêu

Tích hợp một trợ lý AI (sử dụng Google Gemini) được tùy chỉnh kiến thức nghiệp vụ (sử dụng kỹ thuật RAG) vào cả nền tảng web (`conghaiso-web`) và ứng dụng di động (`conghaiso\_app`). Đồng thời, xây dựng một trang quản trị (Knowledge CMS) cho phép Admin cập nhật, chỉnh sửa, và xóa bỏ kiến thức của AI theo thời gian thực.

### 1.2. Công nghệ chủ chốt

\* \*\*AI Models:\*\* Google Gemini (cho Chat) và Google `text-embedding-004` (cho Embeddings).

\* \*\*Kiến thức:\*\* Kỹ thuật RAG (Retrieval-Augmented Generation).

\* \*\*Cơ sở dữ liệu:\*\* PostgreSQL với extension `pgvector` (dùng làm Vector Database).

\* \*\*Backend:\*\* Node.js / Express (`conghaiso-backend`).

\* \*\*Frontend Web:\*\* React / Vite (`conghaiso-web`).

\* \*\*Frontend App:\*\* React Native / Expo (`conghaiso\_app`).

### 1.3. Các Giai đoạn (Phases)

1. \*\*Phase 1: Nền tảng (Foundation):\*\* Cấu hình hạ tầng (DB, `pgvector`) và xây dựng các API cốt lõi (RAG Chat, Knowledge CMS CRUD).

2. \*\*Phase 2: Tính năng Quản trị (Admin):\*\* Xây dựng giao diện CMS trên web cho Admin quản lý kiến thức.

3. \*\*Phase 3: Tính năng Người dùng (User):\*\* Tích hợp giao diện Chatbot vào web và app cho người dùng cuối.

4. \*\*Phase 4: Kiểm thử & Tối ưu (QA & Ops):\*\* Nạp dữ liệu thực tế (100 trang tài liệu), kiểm thử độ chính xác, và tối ưu tốc độ truy vấn.

---

## 2. Phân chia công việc chi tiết (Task Breakdown)

### 2.1. Đội Database (PostgreSQL)

\*Phụ trách: Chuẩn bị cơ sở dữ liệu để lưu trữ và truy vấn vector ngữ nghĩa.\*

\* \*\*Task DB-01: Cài đặt & Kích hoạt `pgvector`\*\*

\* \*\*Mô tả:\*\* Đảm bảo extension `pgvector` được cài đặt và kích hoạt trên máy chủ PostgreSQL (Development, Staging, và Production).

\* \*\*Hành động:\*\* Chạy lệnh `CREATE EXTENSION IF NOT EXISTS vector;` trên CSDL `conghaiso`.

\* \*\*Task DB-02: Tạo Bảng `ai\_knowledge`\*\*

\* \*\*Mô tả:\*\* Tạo bảng mới để lưu trữ các đoạn văn bản (chunks) và vector (embeddings) tương ứng, hỗ trợ phân loại.

\* \*\*Hành động:\*\* Chạy script SQL sau (sử dụng `pgadmin4`):

```sql

CREATE TABLE ai\_knowledge (

id SERIAL PRIMARY KEY,

-- Hạng mục (ví dụ: 'Chấm công', 'Nghiệp vụ Task')

category VARCHAR(100) NOT NULL,

-- Tên tài liệu gốc (để nhóm)

source\_document VARCHAR(255),

-- Nội dung văn bản (đã được cắt nhỏ)

content TEXT NOT NULL,

-- Vector (model 'text-embedding-004' có 768 chiều)

embedding VECTOR(768),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

```

\* \*\*Task DB-03: Tạo Index Tối ưu (HNSW)\*\*

\* \*\*Mô tả:\*\* Tạo index HNSW để tăng tốc độ tìm kiếm vector.

\* \*\*Hành động:\*\* Chạy lệnh `CREATE INDEX ON ai\_knowledge USING HNSW (embedding vector\_cosine\_ops);`

\* \*\*Phối hợp:\*\* \*Quan trọng: Chỉ chạy sau khi đội Backend hoàn thành Task \*\*BE-05\*\* (Nạp dữ liệu ban đầu).\*

---

### 2.2. Đội Backend (Node.js - `conghaiso-backend`)

\*Phụ trách: Xây dựng toàn bộ logic nghiệp vụ, API cho RAG, và API cho Knowledge CMS.\*

\* \*\*Task BE-01: Cài đặt Thư viện & Cấu hình\*\*

\* \*\*Mô tả:\*\* Thêm các thư viện cần thiết và cấu hình biến môi trường.

\* \*\*Hành động:\*\*

1. Chạy `npm install @google/generative-ai pgvector`.

2. Thêm `GEMINI\_API\_KEY=YOUR\_API\_KEY\_HERE` vào file `.env`.

3. Cập nhật file `src/config/database.js` để đăng ký `pgvector` với `Pool` kết nối (sử dụng `pgvector.registerType(pool)`).

\* \*\*Task BE-02: Xây dựng API Quản lý Tri thức (Knowledge CMS)\*\*

\* \*\*Mô tả:\*\* Tạo các endpoint CRUD cho Admin quản lý kiến thức. Tất cả phải được bảo vệ bằng `authMiddleware` và `admin`.

\* \*\*Hành động:\*\* Tạo `knowledgeRoutes.js` và `knowledgeController.js` với các endpoints:

\* `POST /api/knowledge`: Nhận `content`, `category`, `source\_document`. Bên trong: \*\*tự động gọi Google API để tạo embedding\*\* và `INSERT` (content + vector) vào bảng `ai\_knowledge`.

\* `PUT /api/knowledge/:id`: Nhận `content`, `category`, `source\_document`. Bên trong: \*\*tự động tạo lại embedding\*\* và `UPDATE` (content + vector) trong DB.

\* `DELETE /api/knowledge/:id`: Xóa một đoạn tri thức khỏi DB.

\* `GET /api/knowledge`: Lấy danh sách (phân trang) tất cả các đoạn tri thức cho trang Admin.

\* `GET /api/knowledge/:id`: Lấy chi tiết một đoạn tri thức (cho form "Sửa").

\* \*\*Task BE-03: Xây dựng API Chat (RAG Engine)\*\*

\* \*\*Mô tả:\*\* Tạo endpoint chính cho web và app giao tiếp với AI.

\* \*\*Hành động:\*\* Tạo `geminiRoutes.js` với endpoint `POST /api/chat`. Logic bên trong:

1. Nhận `prompt` từ `req.body`.

2. Gọi Google Embedding API để tạo vector cho `prompt`.

3. Thực hiện tìm kiếm `pgvector`: `SELECT content FROM ai\_knowledge ORDER BY embedding <=> $1 LIMIT 3` (lấy 3 đoạn liên quan nhất).

4. Xây dựng "prompt tăng cường" (augmented prompt) với context (kết quả từ DB) và `prompt` của người dùng.

5. Thiết lập `history` (System Prompt) để định vai trò AI (ví dụ: "Bạn là trợ lý ảo của Công Hải Số...").

6. Gọi Gemini Chat API (`gemini-1.5-flash`) với prompt tăng cường.

7. Trả `reply` (text trả về) cho frontend.

\* \*\*Task BE-04: Đăng ký Routes\*\*

\* \*\*Mô tả:\*\* Thêm các routes mới vào `server.js`.

\* \*\*Hành động:\*\* Thêm `app.use('/api/knowledge', knowledgeRoutes)` và `app.use('/api', geminiRoutes)`.

\* \*\*Task BE-05: Nạp Dữ liệu Ban đầu (Initial Ingestion)\*\*

\* \*\*Mô tả:\*\* Đọc 100 trang tài liệu kỹ thuật, cắt nhỏ, và nạp vào DB lần đầu.

\* \*\*Hành động:\*\* Tạo một script `seedKnowledge.js` (chạy một lần) để thực hiện:

1. Đọc file (ví dụ: `tài liệu kỹ thuật.txt`).

2. Cắt file thành các đoạn (chunks) hợp lý.

3. Vòng lặp qua từng chunk, gán `category` (ví dụ: 'Tổng quan'), gọi API embedding và `INSERT` vào bảng `ai\_knowledge`.

\* \*\*Phối hợp:\*\* Báo cho đội Database thực hiện Task \*\*DB-03\*\* (tạo index) ngay sau khi script này chạy xong.

---

### 2.3. Đội Frontend Web (React - `conghaiso-web`)

\*Phụ trách: Xây dựng giao diện Admin (Knowledge CMS) và giao diện Chat cho người dùng.\*

\* \*\*Task FEW-01: (Admin) Xây dựng Trang "Quản lý Tri thức AI"\*\*

\* \*\*Mô tả:\*\* Tạo một trang mới trong khu vực Admin (tương tự `UserManagementPage.jsx`) để hiển thị và quản lý kiến thức AI.

\* \*\*Hành động:\*\*

1. Thêm route `/admin/knowledge` vào `App.jsx` (bảo vệ bằng `ProtectedRoute` cho admin).

2. Tạo component `KnowledgeManagementPage.jsx`.

3. Sử dụng `apiClient.get('/knowledge')` để lấy dữ liệu và hiển thị ra bảng (sử dụng Tailwind CSS).

4. Thêm nút "Xóa" gọi `apiClient.delete('/knowledge/:id)`.

5. Thêm nút "Thêm mới" và "Sửa" (sẽ mở Task FEW-02).

\* \*\*Task FEW-02: (Admin) Xây dựng Modal Form Tri thức\*\*

\* \*\*Mô tả:\*\* Tạo Modal để Admin Thêm/Sửa các đoạn tri thức (tương tự `TaskFormModal.jsx` hoặc `OrgFormModal.jsx`).

\* \*\*Hành động:\*\*

1. Tạo component `KnowledgeFormModal.jsx`.

2. Thêm các trường: `category` (input), `source\_document` (input), `content` (textarea).

3. Khi "Lưu" (Tạo mới): Gọi `apiClient.post('/knowledge', { ...data })`.

4. Khi "Lưu" (Cập nhật): Gọi `apiClient.put('/knowledge/:id', { ...data })`.

\* \*\*Task FEW-03: (User) Tích hợp Giao diện Chatbot\*\*

\* \*\*Mô tả:\*\* Thêm một widget chat (ví dụ: bong bóng ở góc dưới phải) vào `DashboardLayout.jsx` để người dùng có thể trò chuyện với AI.

\* \*\*Hành động:\*\*

1. Tạo component `ChatbotWidget.jsx` (nút bấm nổi).

2. Tạo component `ChatWindow.jsx` (hiển thị khi nhấn vào widget).

3. Thiết kế giao diện chat: vùng hiển thị tin nhắn (`message-list`), ô input, nút gửi.

\* \*\*Task FEW-04: (User) Kết nối API Chat\*\*

\* \*\*Mô tả:\*\* Gọi API backend khi người dùng gửi tin nhắn.

\* \*\*Hành động:\*\*

1. Khi người dùng gửi `prompt`:

2. Hiển thị tin nhắn người dùng và trạng thái "AI đang gõ...".

3. Gọi `apiClient.post('/chat', { prompt })` (đã có trong `src/api/client.js`).

4. Hiển thị `response.data.reply` của AI.

---

### 2.4. Đội Frontend App (React Native/Expo - `conghaiso\_app`)

\*Phụ trách: Tích hợp giao diện Chat cho người dùng trên ứng dụng di động.\*

\* \*\*Task FEA-01: (User) Tạo Màn hình Chat\*\*

\* \*\*Mô tả:\*\* Thêm một tab "Trợ lý AI" vào thanh điều hướng chính.

\* \*\*Hành động:\*\*

1. Thêm một file mới `app/(tabs)/ai-chat.js`.

2. Cập nhật `app/(tabs)/\_layout.js` để thêm tab mới (sử dụng `MaterialCommunityIcons` name="brain" hoặc tương tự).

\* \*\*Task FEA-02: (User) Xây dựng Giao diện Chat\*\*

\* \*\*Mô tả:\*\* Sử dụng các component của React Native để xây dựng giao diện cửa sổ chat.

\* \*\*Hành động:\*\*

1. Sử dụng `FlatList` để hiển thị danh sách tin nhắn.

2. Sử dụng `TextInput` và `TouchableOpacity` (hoặc `Button`) cho ô nhập liệu.

3. \*Tùy chọn:\* Cân nhắc dùng thư viện `react-native-gifted-chat` để tăng tốc độ phát triển.

\* \*\*Task FEA-03: (User) Kết nối API Chat\*\*

\* \*\*Mô tả:\*\* Gọi API backend (đã được xây dựng ở Task BE-03) từ app.

\* \*\*Hành động:\*\*

1. Sử dụng `apiClient` từ `api/client.js` của app.

2. Khi người dùng gửi `prompt`:

3. Gọi `apiClient.post('/chat', { prompt })`.

4. Cập nhật state (`useState`) để thêm tin nhắn của người dùng và tin nhắn của AI vào `FlatList`.

5. Xử lý trạng thái loading và báo lỗi.