Fossil Identification App - Full Stack Roadmap

1. Architecture Overview

The Fossil Identification App integrates Flutter (frontend), FastAPI (backend), Supabase (auth, storage, database), and an AI model (Claude/GPT-4 Vision) for fossil recognition using a structured rule-based system inspired by a 66-page document.

---

2. Should You Use RAG?

No. RAG is unnecessary because the fossil identification document is structured and rule-based, not a knowledge base.

Instead, pass the document as a **system prompt** to the AI model.

---

3. Backend Choice: FastAPI ■

- Async, lightweight, and easy to scale.

- Handles file uploads and integrates well with Supabase.

- Generates automatic Swagger documentation.

Alternative: Express.js with TypeScript if you prefer Node, but FastAPI is better for AI workloads.

---

4. Core Architecture Layout

project/

■■■ app/

■ ■■■ main.py

■ ■■■ routers/

■ ■ ■■■ fossils.py

■ ■■■ services/

■ ■ ■■■ ai_service.py

■ ■ ■■■ storage.py

■ ■■■ models/

■ ■■■ schemas.py

■■■ prompts/

■ ■■■ fossil_prompt.txt

■■■ .env

---

5. Backend Essentials

Security:

- Use API keys from environment variables.

- Enable JWT/Supabase Auth.

- Rate-limit requests (e.g., 10/min per user).

- Restrict CORS to your Flutter app domain.

- Validate uploads (JPG/PNG only, <10MB).

Critical Tasks:

| Priority | Task | Description |

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

| ■ | Image Preprocessing | Resize/compress before sending to AI |

| ■ | Error Handling | User-friendly messages for AI/DB failures |

| ■ | Caching | Prevent duplicate analyses |

| ■ | Logging | Track API requests and AI cost |

| ■ | Monitoring | Sentry or similar |

---

6. Supabase Integration

- Auth: Secure login with email/password or OAuth.

- Storage: Fossil images.

- Database: User data, history, AI results.

- Optional: Realtime updates via Supabase Realtime.

---

7. AI Integration (Claude/OpenAI)

No RAG needed.

- Load fossil prompt as `fossil_prompt.txt`

- Send images and metadata together.

- Expect structured JSON output from AI.

```python
class FossilAI:
async def analyze_fossil(self, image_bytes, metadata):
response = await client.messages.create(
model="claude-sonnet-4",
max_tokens=4096,
messages=[{
"role": "user",
"content": [
{"type": "image", "source": {"type": "base64", "data": base64.b64encode(image_bytes).decode()}},
{"type": "text", "text": f"{self.prompt}\n\nMetadata: {metadata}"}
]
}]
```

)

return response.content[0].text

```

---

## 8. Rule Engine Overview

Implements the **4-step fossil classification funnel** from the document:

1. Coarse Category → (tooth, bone, shell, plant, etc.)

2. Environment → (marine, terrestrial, freshwater)

3. Family Candidates → based on diagnostic rules

4. Species/Genus → only when diagnostic cues are clear

This gives deterministic, testable results.

---

## 9. Production Deployment Checklist

■ HTTPS enabled (Railway, Render, or Cloud Run)

■ Environment variables set (.env)

■ Health endpoint (/health)

■ Logging and rate limiting

■ Sentry for errors

---

## 10. Cost Optimization

- Cache results by image hash.

- Resize images to max 1024×1024.

- Limit 5 analyses/min per user.
```

- Store repeated results in Supabase.

---

11. Phase-wise Implementation Plan

Phase 1: Core Setup (Weeks 1–2)

- Set up Supabase (auth, DB, storage)
- Implement FastAPI skeleton & image upload
- Integrate Claude/OpenAI API

Phase 2: Rule Engine (Weeks 3–4)

- Implement CoarseCategory & Environment logic
- Add whitelist for families
- Validate JSON schema outputs

Phase 3: Production & Optimization (Weeks 5–6)

- Add caching, error handling
- Optimize model prompts
- Deploy to Railway/Render
- Add monitoring & logs

---

12. Success Metrics

- Accuracy: >90% on test fossils
- Response time: <2s
- Cost: <$0.02 per image
- Reliability: 99.9% uptime