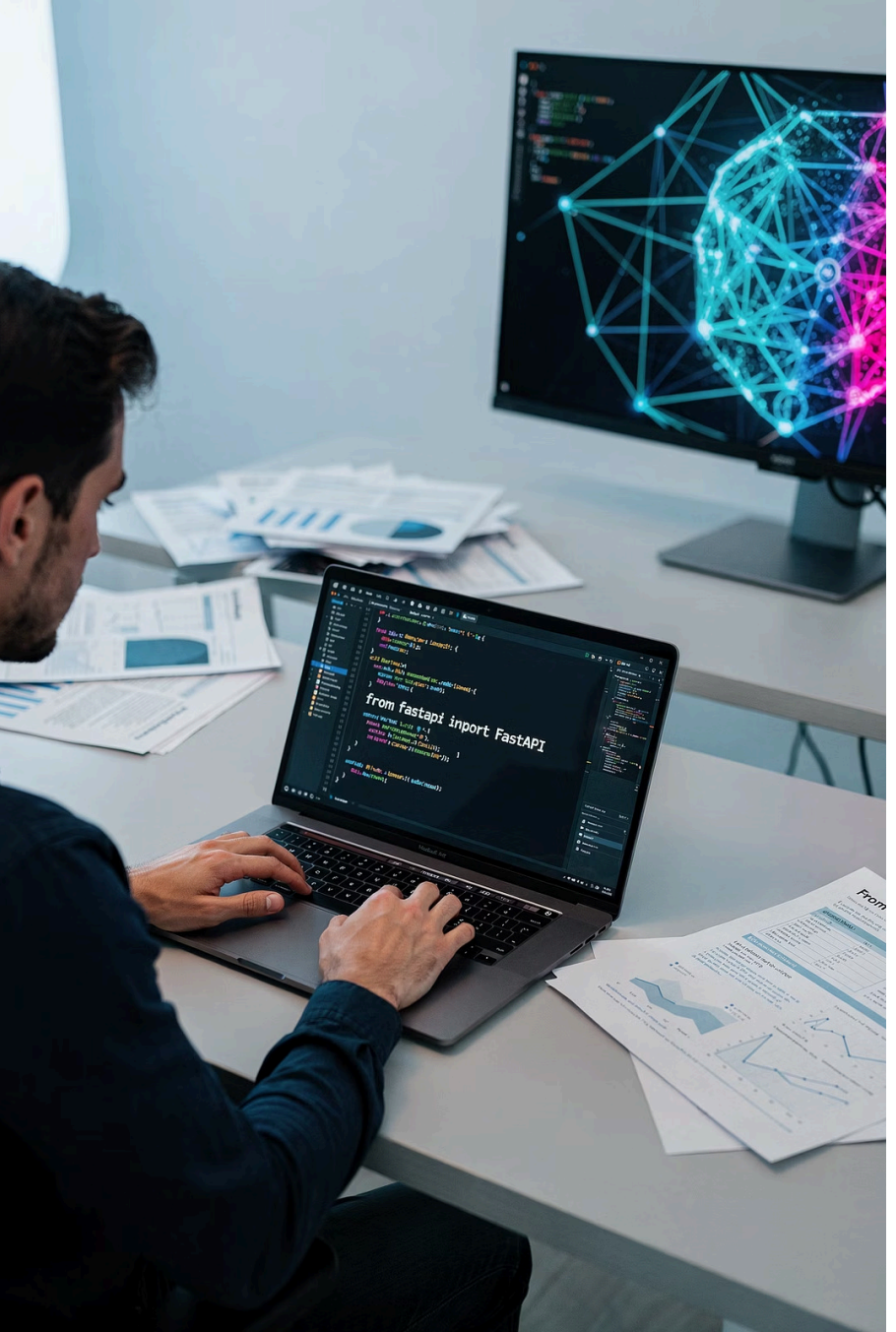


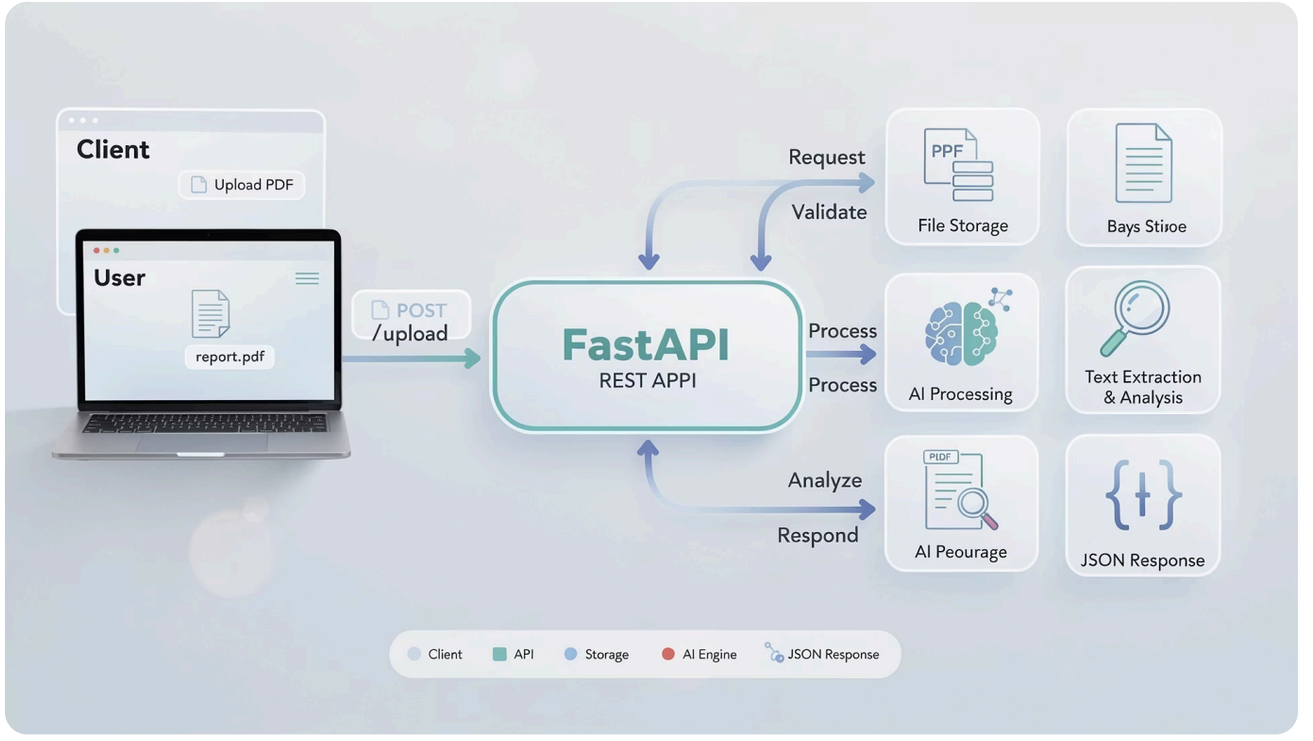

## Guided Project: CV Summarizer PDF

# FastAPI + SQLite + Free AI Integration

Master the art of building production-ready REST APIs by creating an intelligent CV processing system that combines document handling, database persistence, and AI-powered analysis.

PROJECT GOALS

# Final Objective: Your API Workflow




Build a comprehensive REST API that handles the complete CV analysis pipeline from upload to intelligent summarization.

The entire system will be documented automatically via FastAPI's interactive **/docs** interface, making testing and integration seamless.

01

## Receive & Upload

Accept PDF CV files via HTTP POST with validation and size limits

02

## Extract Text

Parse PDF content into machine-readable text for processing

03

## AI Summarization

Generate structured JSON summaries using free AI models

04

## Persist & Expose

Store all data in SQLite and expose documented REST endpoints

TECHNICAL REQUIREMENTS

# Project Assumptions & Constraints

## Minimum Requirements

We'll focus on text-based PDFs where content can be directly extracted. This covers the majority of modern CVs created from Word processors or online builders.

**Bonus Challenge:** Scanned PDFs using OCR technology can be added later as an advanced feature, but aren't required for core functionality.

## Technology Stack

- **Framework:** FastAPI for high-performance async API
- **Database:** SQLite with ORM for simple, embedded persistence
- **AI Provider:** OpenRouter with :free models (no API costs)
- **Output Format:** Structured JSON, not free-form text

### Why FastAPI?

Automatic API documentation, type validation, and excellent async support

### Why SQLite?

Zero configuration, file-based database perfect for learning and prototypes

### Why Free AI?

Learn AI integration without billing concerns or credit card requirements

STEP 0

# Setup & Project Conventions

## Pedagogical Goal

Understanding your development environment, project structure, and Python import system is critical before writing any API code. We'll establish professional conventions that scale.

## Initial Setup Tasks

1. Create an isolated Python virtual environment
2. Install FastAPI plus a production ASGI server (Uvicorn)
3. Define a clean, maintainable project structure
4. Verify your setup with a running server

**Pro Tip:** Always use virtual environments to avoid dependency conflicts between projects. Use `python -m venv venv` to create one.

## Recommended Structure

Build this structure progressively as you implement each step:

```
cv-summarizer/
├── app/
│   ├── __init__.py
│   ├── main.py          # FastAPI app entry
│   ├── routers/         # Endpoint definitions
│   │   ├── __init__.py
│   │   └── cvs.py
│   ├── schemas/         # Pydantic models
│   │   └── cv.py
│   ├── models/          # Database models
│   │   └── cv.py
│   ├── services/        # Business logic
│   │   ├── pdf.py
│   │   └── ai.py
│   └── db/              # Database setup
│       └── session.py
├── uploads/
│   └── cvs/             # Stored PDF files
├── requirements.txt
└── README.md
```

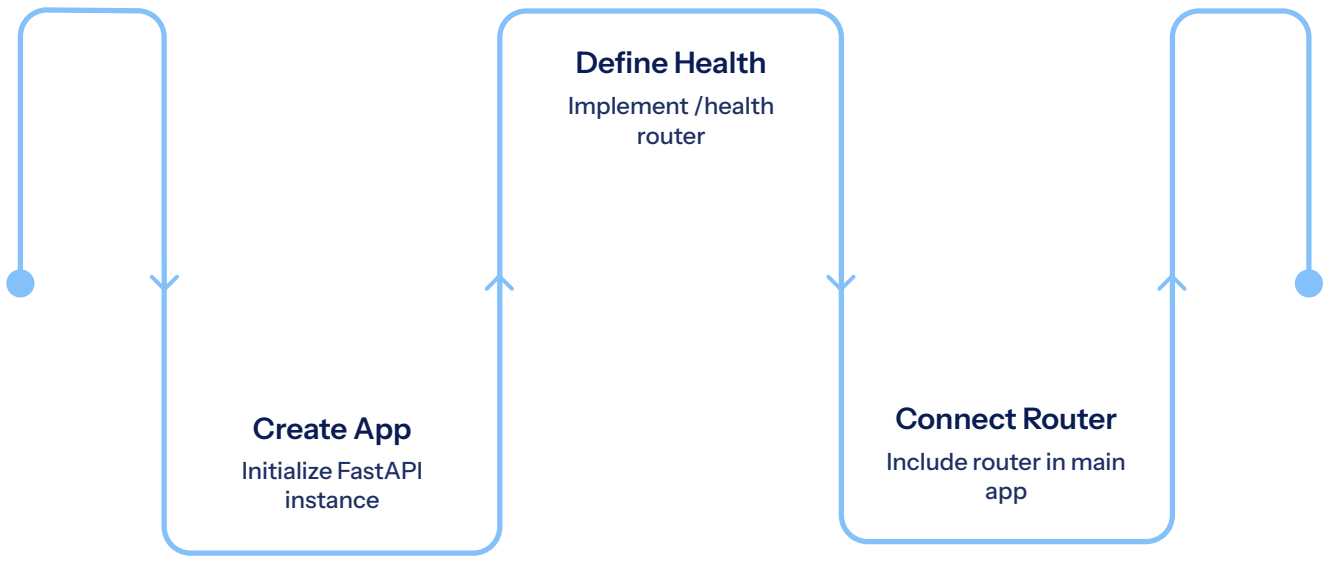
✓

### Checkpoint Success

API starts successfully and Swagger UI is accessible at `/docs`

STEP 1

# Minimal API + First Route




Start with a minimal "Hello World" API to understand FastAPI's routing system and automatic documentation generation.

## Implementation Pseudocode

**Main Application (app/main.py):**

```
create FastAPI app
include router health
configure CORS if needed
```

**Health Router (app/routers/health.py):**

```
router = APIRouter()

@router.get("/health")
def health_check():
    return {"status": "ok",
            "service": "cv-summarizer"}
```

The `/health` endpoint is an industry-standard practice for monitoring service availability in production environments.


### Checkpoints

GET /health returns OK status and /docs displays the endpoint with full documentation

STEP 2

# Upload CV PDF Files

Learn to handle file uploads securely via HTTP multipart/form-data, with proper validation, storage, and error handling.

1

### Endpoint Definition

`POST /cvs/upload` receives file and returns unique identifier

2

### Content Type Validation

Accept only `application/pdf` — reject other types with 415 status

3

### Size Limits

Enforce maximum file size (5 MB recommended) to prevent abuse

4

### Generate ID & Store

Create unique UUID, save to `uploads/cvs/{cv_id}.pdf`

## Pseudocode Implementation

```
POST /cvs/upload(file):
    if file.content_type != "application/pdf":
        return 415 Unsupported Media Type

    if file.size > MAX_FILE_SIZE:
        return 413 Payload Too Large

    cv_id = generate_uuid()
    file_path = f"uploads/cvs/{cv_id}.pdf"
    save_file_bytes(file_path, file.content)

    return {
        "id": cv_id,
        "filename": file.filename,
        "size_bytes": file.size
    }
```

### Success Indicator

PDF file physically exists in `uploads/cvs/` directory

### Response Validation

API returns valid UUID identifier and original filename

STEP 3

# Extract Text from PDF

## Objective

Transform binary PDF files into machine-readable text without OCR. This works for PDFs created digitally (Word, Google Docs, LaTeX) where text is selectable.

## New Endpoint

`POST /cvs/{id}/extract` processes the PDF and returns a text preview.

## Validation Rules

- File doesn't exist → 404 Not Found
- Extracted text too short (< 300 chars) → 422 Unprocessable with message "Likely scanned PDF"
- Success → Return character count + 300-char preview

## Extraction Logic

```
def extract_text(pdf_path):
    open pdf file
    text = ""

    for page in pdf.pages:
        page_text = extract_page_text(page)
        text += page_text + "\n"

    return text.strip()
```

## Endpoint Implementation

```
POST /cvs/{id}/extract:
    path = f"uploads/cvs/{id}.pdf"

if not file_exists(path):
    return 404

text = extract_text(path)

if len(text) < MIN_CHARS:
    return 422, "PDF likely scanned"

return {
    "id": id,
    "extracted_chars": len(text),
    "preview": text[:300]
}
```

✓

### Text PDF Checkpoint

Successfully extracts and returns readable content

### Scanned PDF Checkpoint

Returns explicit 422 error with helpful message

STEP 4

# Database Layer: SQLite + ORM

Persist all CV data — uploaded files, extracted text, and AI summaries — using SQLite with an Object-Relational Mapping layer for clean, Pythonic database operations.

## Database Schema: cvs Table

| Field | Type | Description |
|---|---|---|
| id | Integer/UUID | Primary key (auto-increment or UUID) |
| original_filename | String | User's uploaded filename |
| pdf_path | String | Server file path |
| extracted_text | Text (nullable) | Parsed PDF content |
| summary_json | JSON (nullable) | AI-generated summary |
| created_at | DateTime | Upload timestamp |

## Required Tasks

1. Configure database connection and session management
2. Create ORM model class for CV table
3. Define Pydantic schemas for API input/output validation
4. Update `POST /upload` to create database records
5. Update `POST /extract` to persist extracted text

Using an ORM (like SQLAlchemy) prevents SQL injection and makes database operations type-safe and testable.

## Updated Pseudocode

1

### Upload with DB

```
validate pdf
save to disk
db_cv = create_row(
    filename, path, now)
return cv.id
```

2

### Extract with DB

```
db_cv = find_by_id()
text = extract(cv.path)
if too_short: 422
cv.extracted_text = text
commit()
return preview
```

## Additional CRUD Endpoints

### GET /cvs

List all CVs with id, filename, created_at

### GET /cvs/{id}

Full CV details with text/summary availability flags

### DELETE /cvs/{id}

Remove database record AND delete physical file

STEP 5

# AI Integration: Free Structured Summarization

Connect to OpenRouter's free AI models to generate intelligent, structured CV summaries in JSON format — no API costs, no credit card required.

## New Endpoints

**POST /cvs/{id}/summarize**
Generates and stores AI summary

**GET /cvs/{id}/summary**
Retrieves stored summary JSON

## Critical Requirements

- Use OpenRouter with `:free` model tier
- Prompt must force valid JSON output
- Non-JSON responses → 502 Bad Gateway
- Log all AI errors for debugging

## Required JSON Fields

- **profile_title**: Professional headline (string)
- **experience_years**: Years of experience (number)
- **key_skills**: 8-12 core competencies (array)
- **technologies**: Technical stack (array)
- **experiences**: Work history (array of objects)
- **education**: Academic background (array)
- **languages**: Spoken languages (array)
- **strengths**: 3-5 key strengths (array)
- **questions**: 2-4 interview questions (array)

## Prompt Engineering Strategy

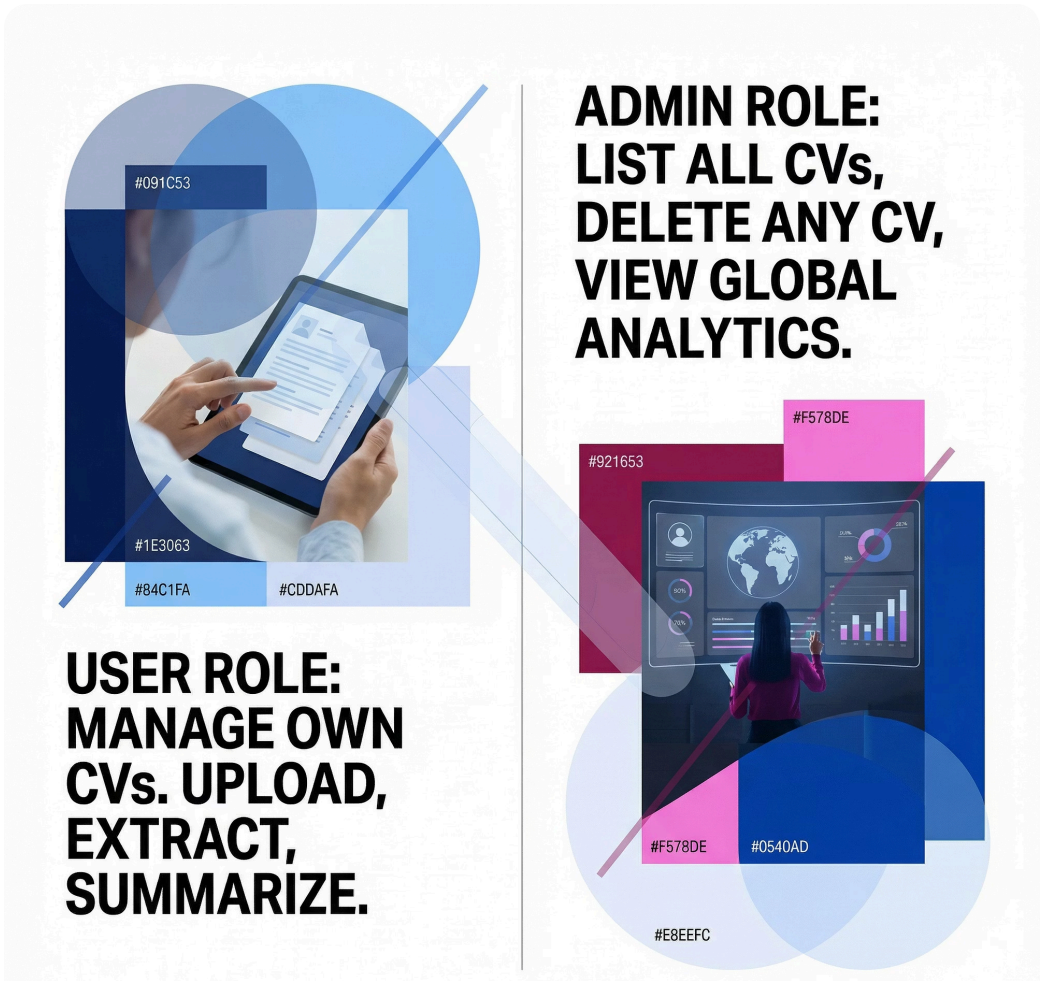
```
SYSTEM MESSAGE:
"You are a CV analysis expert. Respond ONLY with valid JSON.
No markdown formatting, no code blocks, no explanations."

USER MESSAGE:
"Analyze this CV and return EXACTLY this JSON structure:
{
  'profile_title': '...',
  'experience_years': 0,
  'key_skills': [...],
  ...
}

CV TEXT:
{extracted_text}"
```

## Summarize Endpoint Pseudocode

```
POST /cvs/{id}/summarize:
 db_cv = find_by_id(id) or return 404

if db_cv.extracted_text is None:
return 409 "Must extract text first"

response = call_llm(
messages=[system_msg, user_msg],
model="openrouter/free-model"
)

if not is_valid_json(response):
log_error(response)
return 502 "AI returned invalid JSON"

db_cv.summary_json = response
commit()

return parse_json(response)
```

BONUS CHALLENGE

# Step 6: JWT Authentication & Role-Based Access




## Security Enhancement Goal

Add production-grade authentication using JWT tokens with role-based permissions. Separate user-level CV management from admin-level system oversight.

## Implementation Requirements

01

### User Management

Create users table with username, hashed password, and role field

02

### Authentication Flow

Implement `/register` and `/login` endpoints that issue JWT tokens

03

### Middleware Protection

Add `get_current_user` dependency and `require_role()` decorator

## Access Control Rules

### Upload / Extract / Summarize

Requires authenticated user token

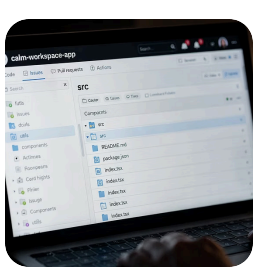
### Global CV List

Admin role only

### Delete CV

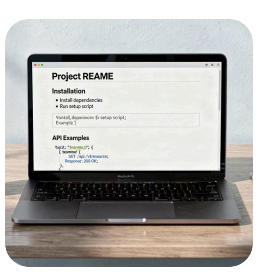
Admin or resource owner only

# Project Deliverables



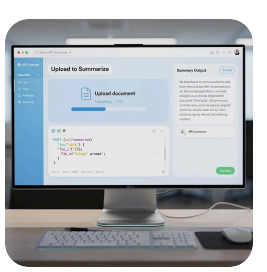
### Complete Source Code

Well-organized repository following project structure conventions with all six steps implemented




### Comprehensive README

Installation guide, AI configuration steps, environment variables, and endpoint usage examples




### 5-Minute Demo

Video or live demonstration: upload PDF → extract text → generate summary → retrieve JSON