Card Content Extractor

1. Project Title

ID Card Content Extractor (Offline AI OCR + Entity Mapping Microservice)

2. Objective

Build a microservice that extracts **structured data** from a student-uploaded **college ID card image**. The service must:

- Run **offline OCR** on the image
- Extract key fields (Name, College, Roll Number, Branch, Validity)
- Use **local AI models** to parse and map the fields accurately
- Be built with **FastAPI**, configurable via `config.json`, and deployable via Docker
- Accept **base64 image input** and return **JSON output**

3. Functional Requirements

A. Input

- **Endpoint**: `POST /extract`
- **Payload**:

```
json

{
  "user_id": "stu_4821",
  "image_base64": "<base64_encoded_id_card_image>"
}
```

Field	Type	Required	Description
`user_id`	string	yes	Unique student identifier
`image_base64`	string	yes	ID card image (base64-encoded)

B. Output

- **Response**:

```
json

{
  "user_id": "stu_4821",
  "extracted_fields": {
    "name": "Anjali Sharma",
    "college": "RGMCET Nandyal",
    "roll_number": "21RGME1032",
    "branch": "Mechanical Engineering",
    "valid_upto": "2025"
  },
  "confidence_score": 0.91,
  "missing_fields": ["branch"],
  "status": "partial_success"
}
```

Field	Description
`extracted_fields`	Key-value output from ID card
`confidence_score`	Average confidence (0–1)
`missing_fields`	Fields that couldn't be parsed
`status`	`success`, `partial_success`, or `failure`

4. AI Model Requirements

A. Model Type

- Use **Tesseract OCR (locally)** for raw text extraction
- Use **NER model** (trained using `spaCy` or `sklearn`) to extract structured fields
- Use regex & entity matching fallback for improved accuracy
- Final output must include confidence per field and a total score

B. Dataset

- Build or use a sample of **50–100 synthetic Indian student ID card images**
- Each must have labeled ground-truth fields in JSON
- Train/test the field extractor to achieve >85% overall accuracy

C. Serving

- Use a trained entity classifier OR template+regex model for each field
- Model must be loaded at app startup
- Inference must be done offline — no API to external OCR/LLMs

5. Software Architecture Requirements

- Modular design: OCR layer → Text cleaner → Entity extractor
- Field detection must be driven by:
  - Regex
  - Classifier (ML-based)
  - Configurable templates (if needed)
- `config.json` must allow:
  - Field definitions
  - OCR thresholds
  - Regex patterns
- All inputs must be validated using **Pydantic**

6. API Endpoints

Method	Endpoint	Description

POST	`/extract`	Upload image and extract fields
GET	`/health`	Returns `{ "status": "ok" }`
GET	`/version`	Returns `{ "model_version": "1.0.0", "config_version": "1.0.0" }`

7. FastAPI Requirement

- Service **must be built using FastAPI**
- All routes must use **Pydantic models** for input/output
- Serve with **Uvicorn** on port 8000
- Auto-generated Swagger docs available at `/docs`

8. Testing & Validation

- Must include:
  - Valid image with full field extraction
  - Image with 2–3 missing fields
  - Garbage image or unrelated image (should fail)
  - Low quality image with partial success
- Confidence score must guide the `status` response
- Include at least 5 sample ID card images in `tests/data/`

9. Deliverables

- Source code organized under `app/`, `model/`, `tests/`
- Trained model (`.pkl`) and vectorizer (if applicable)

- Sample images + expected output JSON
- `config.json` with thresholds and patterns
- `README.md` explaining how to run and test
- Input/output schema definitions
- Health and version endpoints

10. Timeline (6 Weeks)

Week	Deliverables
1	Sample dataset and regex pattern design
2	OCR setup + baseline extraction logic
3	Train NER model and test on held-out samples
4	Integrate with FastAPI and config.json
5	Add field-wise confidence, logging, and tests
6	Final documentation and packaging

11. Constraints

- Must be **100% offline**
- No GPT/OpenAI/cloud OCR APIs
- Use only open-source tools like Tesseract, spaCy, scikit-learn
- Must run inside Docker
- Input must be `base64` image in JSON
- Output must be structured JSON

12. Deployment Expectations

- The service **must be containerized using Docker**
- Image should include:

    - All Python dependencies
    - Pretrained models
    - Tesseract and supporting binaries
- Intern must provide working build/run commands:

```
docker build -t idcard-extractor .
docker run -p 8000:8000 idcard-extractor
```