Architectural Design Plan

Teacher Timetable Extraction & Visualization Platform

1. Problem Statement & Goals

Teachers currently maintain weekly class timetables in many different formats: scanned images, PDFs, Word documents, screenshots, and even photos of handwritten tables. Each file has its own layout, fonts, colours, and structure.

Our goal is to let a teacher:
1. Upload their existing timetable document (any common format).
2. Automatically extract all Timeblock Events (e.g. “Registration”, “Maths”, “Play”, “Snack Time”).
3. Detect accurate start/end times or durations for each event.
4. Preserve original labels and any free-text notes.
5. View and edit the structured timetable in our own frontend UI.

The solution must be robust to layout and formatting differences and must gracefully handle imperfect OCR or ambiguous inputs.

2. Requirements

2.1 Functional Requirements

- File Upload: Support PDF, Word (.docx), PNG/JPG, scanned/handwritten timetables.
- Timetable Extraction: Detect grid, extract day, time, title, notes.
- Data Output: Normalize into structured JSON.
- Frontend Visualization: Weekly grid view with edit features.
- Feedback Loop: Save corrections for future improvements.

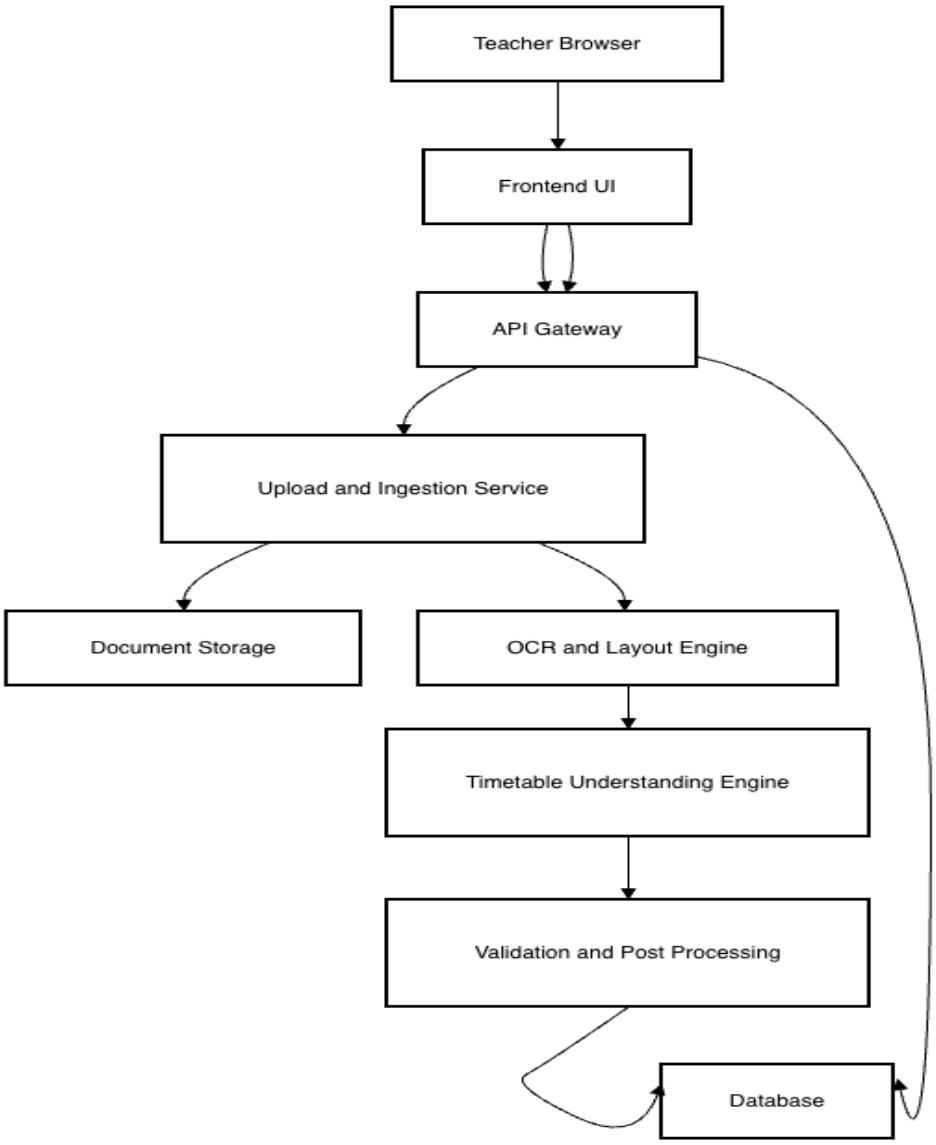
2.2 Non-Functional Requirements

- Robustness
- Accuracy
- Scalability
- Security & Privacy
- Extensibility

3. High-Level Architecture Overview

1. Upload & Ingestion Service
2. Document Pre-processing & Normalization
3. OCR & Layout Analysis
4. Timetable Understanding Engine
5. Validation & Post Processing
6. Persistence Layer
7. API Layer
8. Frontend UI

Figure 1: High-Level System Architecture



4. Detailed Component Design

4.1 Upload & Ingestion Service
Handles authentication, upload validation, file storage, database record creation, and job queue processing.

4.2 Document Pre-processing
Converts files to images/text, normalizes orientation, and enhances OCR quality.

4.3 OCR & Layout Analysis
Extracts text with bounding boxes and identifies table structures using cloud or self-hosted OCR.

4.4 Timetable Understanding Engine
Uses rules + LLM reasoning to detect grid, extract events, normalize time, and infer missing values.

4.5 Validation & Post-Processing
Normalizes output, detects conflicts, flags uncertain blocks.

4.6 Data Model
Core entities: Teacher, Document, Timetable, TimeblockEvent.

4.7 API Layer
Provides endpoints for upload, retrieval, and update of timetables.

4.8 Frontend UI
Upload screen, review/edit grid, and teacher dashboard.

5. Suggested Database schema

1. teachers:

Field	Type	Notes
id	smallint unsigned	Primary key, keep auto increment
name	varchar(150)	Not null
email	varchar(150)	Not null, unique

status	Enum - A,I,D	A= Active, I - Inactive, D - Deleted
created_by_id	smallInt- unsigned	Not null, used to store the id of the user who created this teacher.
updated_by_id	smallInt -unsigned	Allow null - it will be null when user is created, after that every time users details are updated then update this
inactive_date	date	Allow Null - update this when we change users status to inactive
comment	text	Allow Null - add reason for status inactive or any other comment related to the teacher. Visible to admin only
created_at	datetime	Not null - timestamp when user is created. Can set default to current timestamp
updated_at	datetime	Allow null - insert datetime when this record is being updated

Purpose -

The teachers table stores the master profile and lifecycle information of every teacher using the system. It is the root entity for the platform and acts as the primary owner of all timetables and timeblocks.

This table is used for:

- Linking each timetable to its respective teacher.
- Managing teacher status in the system (Active, Inactive, Deleted).
- Supporting audit tracking through created_by_id and updated_by_id.
- Maintaining administrative information such as inactive reasons and comments, which are visible only to admins.
- Enabling future features such as authentication, role-based access, and performance analytics.

Indexing Strategy -

- Id - Primary key
- Email - Unique key

2.timetables:

Field	Types	Notes
id	Int unsigned	PK, auto increment
teacher_id	Smallint unsigned	FK teachers.id
source_file_path	text	
week_start_date	date	null
metadata	JSON	Raw extracted info
created_by_id	Smallint unsigned	Not null teaches.id
updated_by_id	Smallint unsigned	Not null teachers.id
created_at	datetime	Not null - default current timestamp
updated_at	datetime	Allow null

Purpose -

The timetables table represents each uploaded timetable document for a teacher and the corresponding parsed result.

It is used to:

- Store a link between a teacher and a specific timetable upload.
- Track the original source file (source_file_path) so the system can reprocess or audit later.
- Optionally anchor a timetable to a specific week via week_start_date.
- Store raw extracted metadata from OCR/LLM in the metadata JSON field (e.g., parsing notes, original text, model version).
- Maintain audit information (created_by_id, updated_by_id, timestamps) for admin visibility and traceability.
- Act as the parent record for all related timeblocks rows.

This table is the bridge between uploaded documents and structured timetable data shown in the frontend.

Indexing Strategy -

- Id - primary key
- teacher_id - foreign key

- Teacher_id + week_start_date - composite key
- week_start_date

3.timeblocks:

Field	Types	Notes
id	Int - unsigned	PK, auto increment
timetable_id	Int - unsigned	FK, timetables.id
day_of_week	varchar(10)	Mon-Sun
start_time	time	Not null
end_time	time	Not null
label	text	Subject / activity
notes	text	null
room	text	null
extra_metadata	JSON	null
created_at	datetime	Not null - timestamp when user is created. Can set default to current timestamp
updated_at	datetime	Allow null - insert datetime when this record is being updated

Purpose -

The timeblocks table stores the individual building blocks of a timetable.
Each row represents a single scheduled unit such as a lesson, break, registration, or activity.

It is used to:

- Represent the full weekly schedule for a timetable at a fine-grained level (row = one block).
- Link each timeblock to its parent timetable via timetable_id.
- Capture:
    - When the block happens (day_of_week, start_time, end_time)
    - What it is (label – e.g., “Maths”, “Snack Time”)

    - Additional context (notes, room)

- Store AI-related or system metadata in extra_metadata (e.g., confidence scores, ambiguous flags, extraction version).
- Power the frontend timetable view, where blocks are rendered by day and timeline.

This table is what the frontend reads to draw the grid of classes and activities for each teacher.

Indexing Strategy -

- Id - primary key
- Timetable_id - foreign key

Figure 2 - Timetable extraction flow

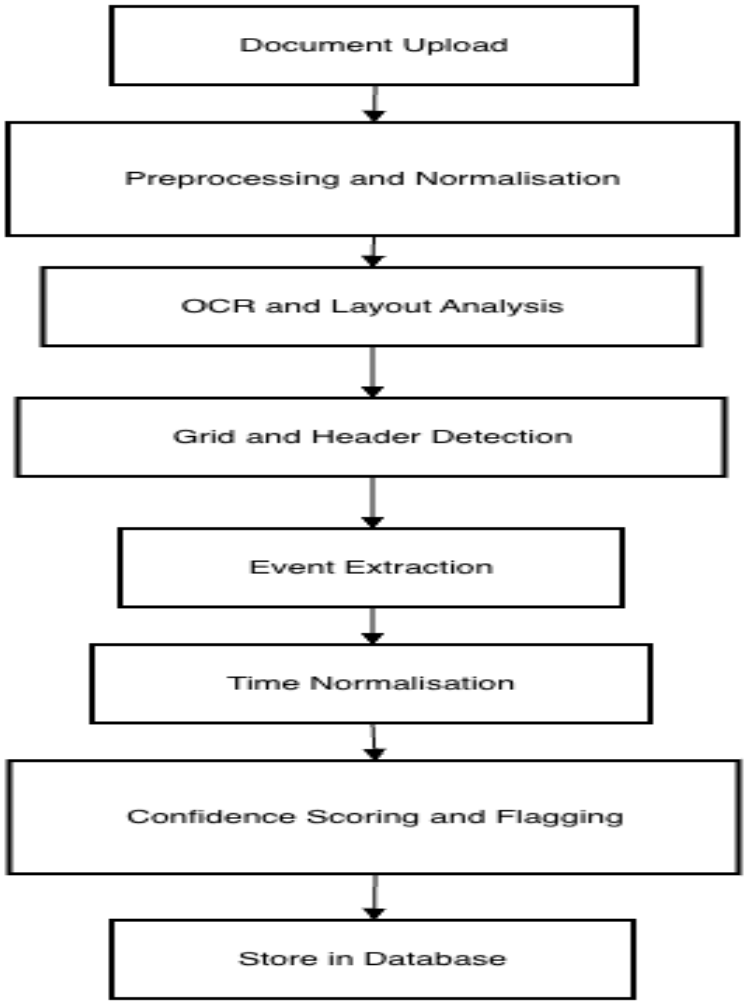
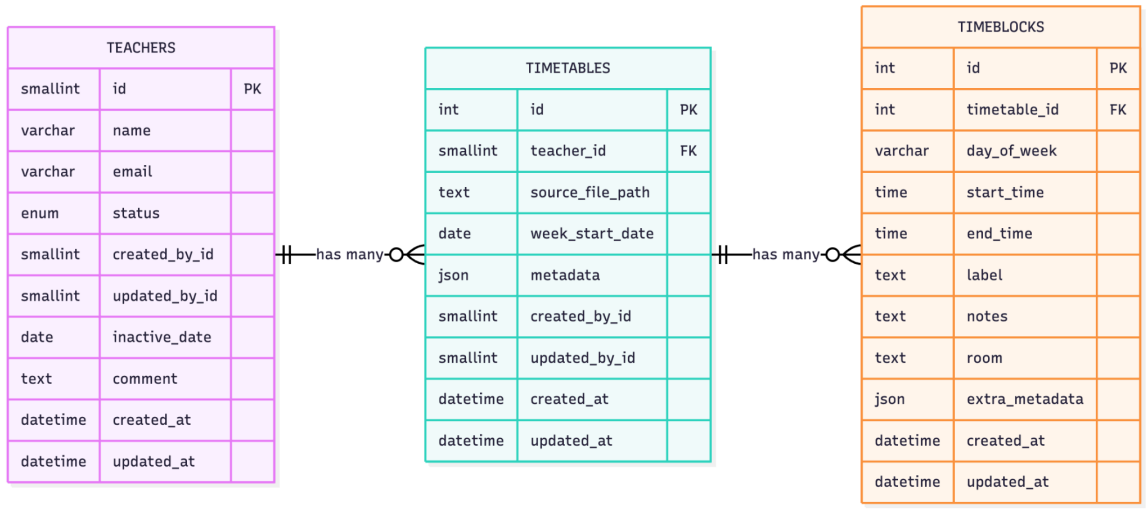


Figure 3 - ER model Diagram



6. Real-World Robustness

Hybrid OCR + Rules + LLM approach with human-in-the-loop editing.

7. Technology Stack

- Backend: Node.js
- Database: PostgreSQL/MySQL
- Storage: GCS
- OCR: GCP Document AI
- Frontend: React
- LLM - Vertex AI - gemini

8. Risks & Mitigations

- Poor OCR → Manual edit tools
- Custom layouts → Semi-structured fallback
- High latency → Async processing
- High cost → Caching & batching

9. Future Enhancements

- Asynchronous processing & Job Queue
- Authentication and role based access
- Admin review & manual validation dashboard
- Multi week & rational timetable support

- Multi language and localization support
- Model feedback and continuous learning pipeline
- Mobile application support