Assignment Agent: Architectural Review and Project Analysis

Welcome to the comprehensive architectural review of **Assignment Agent**. As a Senior Engineering Manager and System Architect, I have analyzed the end-to-end implementation of this AI-powered automated assignment evaluation system. Below is a detailed breakdown of the system's workflow, file-level functionality, technical rationale, and operational strategy.

1. COMPLETE PROJECT WORKFLOW

The Assignment Agent is designed to automate the grading process by combining traditional file parsing, OCR, and advanced LLM reasoning. The workflow is split into three primary streams:

1.1 File Upload Assignment (PDF / DOC / Image)

1. **Ingestion:** The user uploads a file via the frontend (React). The file is sent to the

/files/upload

endpoint in FastAPI.

2. **Storage:** The file is saved in the

uploads/

directory with a unique UUID, and metadata (original name, etc.) is stored in a companion

.json file.

3. **Extraction:** The

FileProcessor service identifies the file type and uses specialized libraries (

pdfplumber

for PDFs,

python-docx

for Word,

pywin32

for legacy

.doc

).

4. **OCR Fallback:** If a PDF is scanned or an image is uploaded, the system triggers OCR using **NVIDIA OCR API** (primary) or local **Tesseract** (fallback).
5. **LLM Reasoning:** The extracted text is sent to **OpenRouter** (GPT-4o-mini or similar). The prompt includes a strict rubric and instructions to return data in a specific JSON schema.
6. **Scoring & Persistence:** The backend parses the LLM's JSON response, calculates the final score based on

is_correct

and

partial_credit

flags, and saves the results into the **PostgreSQL** database.

1.2 PPT Upload Assignment

1. **Dual Analysis:** Unlike standard files, PPTs undergo both **Content Evaluation** and **Visual Design Evaluation**.
2. **Content extraction:**

PPTProcessor

extracts text from every slide.

3. **Design Analysis:** The system captures slide metadata (backgrounds, font sizes, image counts) to evaluate the "Visual Design" (Glassmorphism, Dark Modes, layouts).
4. **Multi-Model Evaluation:**

PPTDesignEvaluator

uses a vision-capable prompt or metadata analysis to judge aesthetics, while

PPTEvaluator

handles the subject matter.

5. **Consolidation:** The results are merged into a single comprehensive report for the student.

1.3 GitHub Repository Evaluation

1. **Cloning/Fetching:** Instead of local files, the user provides a GitHub URL.

GitHubService uses the GitHub API to recursively fetch file contents from the repository.

2. **Repo Mapping:** The system maps the directory structure and ignores non-code directories (like

node_modules

).

3. **Architectural Analysis:**

GitEvaluator analyzes the entire codebase to detect the tech stack, features, and overall project purpose.

4. **Grading against Rules:** The system compares the actual code implementation against the user's provided "Task Description" to check if the student followed specific architectural requirements.

2. FILE-WISE EXPLANATION (FRONTEND + BACKEND)

Backend Core

- main.py: The entry point. It initializes the FastAPI app, configures CORS, includes routers, and triggers the background cleanup task. Without it, the server cannot start.
- database.py: Manages the SQLAlchemy engine and PostgreSQL connection. It defines the

get_db dependency used across all APIs.

- models.py: Defines the database schema (Users, Assignments, EvaluationResults). It is the source of truth for how data is structured on disk.
- auth.py: Handles JWT token generation, password hashing (bcrypt), and user authentication. Removing this would leave the system insecure.

Backend Services (

services/

)

- file_processor.py: The heavy lifter for file parsing. It contains the logic for reading 10+ file formats and the OCR bridge.
- openrouter_service.py: The interface for AI. It handles API calls to OpenRouter, retries, and error management.
- generate_service_complete.py: Contains the core "grading logic." It builds the massive prompts sent to the AI and processes the resulting JSON.

- cleanup_service.py: A background worker that deletes files and database entries older than 15 days to save storage.
- ppt_evaluator.py **&**

git_evaluator.py: Specialized logic for PowerPoint design and GitHub repository structure analysis.

Backend Routers (

routers/

)

- files.py: Endpoints for uploading files and triggering the evaluation.
- github.py: Endpoints specifically for GitHub repo analysis.
- reevaluate.py: Allows users to re-run the AI evaluation without re-uploading files.

Frontend (

src/

)

- App.jsx: The main React component that handles routing and protected routes.
- Dashboard.jsx: The primary UI where users see their previous evaluations.
- Services.jsx: A massive component handling the complex file upload UI, progress states, and result display.
- axios.js: Pre-configured Axios instance with base URLs and interceptors for auth tokens.

3. SENIOR MANAGER / INTERVIEW QUESTIONS

Q: Why did you choose FastAPI?

- **Answer:** FastAPI was chosen for its asynchronous support (essential for long-running LLM and OCR tasks), automatic OpenAPI (Swagger) documentation, and high performance. Its Pydantic integration ensures strict data validation, which is critical when handling varied student submissions.

Q: How does your scoring mechanism work?

- **Answer:** We use a **Deterministic Hybrid Scoring** model. The LLM is responsible for the qualitative evaluation (marking

is_correct: true/false

and providing feedback). However, the final percentage calculation is done by the Python backend. This prevents "LLM Math Errors" and ensures consistency.

Q: How do you handle incorrect OCR output?

- **Answer:** We implement a **Multi-Stage Fallback**. First, we attempt high-fidelity extraction using

pdfplumber

. If it returns too little text, we trigger NVIDIA's OCR. If that fails, we fallback to local Tesseract. The prompt also instructs the LLM to flag if the extracted text looks like gibberish.

Q: How do you scale this system?

- **Answer:** To scale, we would move the long-running evaluation tasks (OCR/LLM) to a task queue like **Celery with Redis**. The

uploads/

directory would be replaced with **S3-compatible storage**. Since the backend is stateless, it can be deployed in multiple containers behind a Load Balancer.

4. REQUIREMENTS.TXT – DETAILED EXPLANATION

- fastapi

&

uvicorn

: The web framework and the high-speed server to run it.

- sqlalchemy

&

psycopg2-binary

: For database ORM and connecting to PostgreSQL.

- python-jose

&

passlib

: For handling JWT security and secure password storage.

- pdfplumber

&

PyPDF2

: Essential for extracting text from PDFs. Removing them would break assignment reading.

- python-docx

: Parses Word files. Without it,

.docx

files cannot be graded.

- pywin32

: A specialized library to handle legacy

.doc

files on Windows systems via COM automation.

- pdf2image

&

pytesseract

: The foundation of the local OCR system.

- python-pptx

: For structural analysis of PowerPoint presentations.

- requests

: Used to communicate with external APIs like OpenRouter and GitHub.

5. MANAGER DEEP-DIVE QUESTIONS

5.1 How did you connect FastAPI to the frontend? Through a RESTful API. The React frontend uses **Axios** to send multipart/form-data (files) or JSON (GitHub URLs) to FastAPI. We configured **CORS Middleware** in

main.py to allow the frontend (running on port 3000) to safely talk to the backend (port 8000).

5.2 How does the LLM work internally? The LLM acts as an "Intelligent Reasoner." It receives the student's text, the teacher's instructions, and a set of grading rules. It performs

a semantic match rather than a keyword search, allowing it to understand that "Powerhouse of the cell" and "Produces ATP" are conceptually similar.

5.3 How does OCR extract text? The system converts document pages (PDF/DOCX) or images into high-resolution PNGs using

pdf2image

. These images are then processed by Tesseract or NVIDIA OCR, which performs optical character recognition to turn pixels into strings of text.

5.4 How did you connect OpenRouter? We built an

OpenRouterService class that handles the API authentication, model selection (e.g., GPT-4o-mini), and **Exponential Backoff** retries. If the API is busy (429 error), the system waits and tries again automatically.

5.5 How do you pass prompts to the LLM? We use **Templated Prompts**. The system prepends the "Global Instructions" (e.g., "Be a strict grader") and the "Task Description" to the student's extracted text. We use a hidden system message to force the LLM to output valid JSON.

5.6 How do you handle LLM failures? We use

try-except

blocks around the JSON parsing. If the LLM returns invalid JSON or an error, the system catches it and returns a user-friendly error message instead of crashing. We also use a

timeout

of 300 seconds for long documents.

5.7 How do you store scores in the database? We use a relational structure. An

Assignment record is created, which links to multiple

EvaluationResult records (one per student file). Each result has many

EvaluationDetail records (one per question) for granular feedback.

5.8 How do you delete old data automatically? In

main.py, we initialize a **Scheduled Background Task** using

asyncio

. Once every 24 hours, it calls

CleanupService.run_cleanup()

, which queries the DB for files older than 15 days and deletes them from both the disk and the database.

6. ADDITIONAL ARCHITECTURAL QUESTIONS

Q: How does the system ensure "Deterministic" grading?

- **Answer:** We set the LLM

temperature

to 0.0. This ensures the model always picks the most likely next token, minimizing "creativity" and ensuring that the same input consistently produces the same grade.

Q: How do you handle very large files that exceed the LLM's token limit?

- **Answer:** We implemented a **Chunking and Truncation Strategy** in

generate_service_complete.py. Files are truncated at a configurable character limit (e.g., 20,000 chars) to stay within context windows while ensuring the most relevant content (usually at the start) is processed.

Q: Is the system multi-tenant?

- **Answer:** Yes. By using the

user_id

foreign key in the

Assignment model, the system ensures that users only see the assignments they uploaded, providing data isolation.

7. FINAL SUMMARY

Assignment Agent is a sophisticated education-tech solution that solves the "Grading Bottleneck." By replacing manual reading with AI-driven semantic analysis, it allows educators to provide instant, detailed feedback to students.

- **Scalability:** The choice of FastAPI and PostgreSQL allows for high concurrency.
- **Versatility:** Supporting PDF, DOCX, PPT, and GitHub makes it a universal tool for various subjects (from Biology to Computer Science).
- **Professionalism:** The use of background tasks for cleanup, structured logging, and JWT authentication demonstrates a production-ready mindset.

- **AI Mastery:** The project shows a deep understanding of prompt engineering, LLM integration, and the critical need for deterministic scoring in academic settings.

Professionalism: The use of background tasks for cleanup, structured logging, and JWT authentication demonstrates a production-ready mindset.

AI Mastery: The project shows a deep understanding of prompt engineering, LLM integration, and the critical need for deterministic scoring in academic settings.

For your first internship review, it is very common for senior engineers to move from "what it does" to **"how it's built"** and **"how it stores data."** They want to see if you understand the **Data Lifecycle**—how a file becomes a database row.

Here is a deep dive into the areas they are likely to probe during your selection process.

1. TECHNICAL DEEP DIVE: DATABASE & SCHEMAS

In a professional review, the database is considered the **"Brain"** of the system. You should be able to explain the relationships between your tables.

The Database Architecture

We use **PostgreSQL** (a Relational Database) managed via **SQLAlchemy ORM**. This allows us to handle complex relationships between users, files, and AI results smoothly.

The Schema Flow (One-to-Many Relationships)

Explain your 5 main tables in this order:

1. User **Table:** Stores authentication data (Email/Hashed Password). Every assignment belongs to a User.
2. Assignment **Table:** This is the "Parent" container. It holds the high-level details: *Title*, *Global Description*, and *Status* (Draft/Completed).
3. AssignmentFile **Table:** Stores the actual files linked to an assignment.
    - *Key Fields:*

file_id

(UUID),

file_path

,

extracted_text

.

- *Architecture Tip:* Mention that we store the file on the **disk** (filesystem) but store the **path** and **extracted text** in the DB for fast retrieval.

4. EvaluationResult **Table:** This is where the AI's "Report Card" lives.
    - It stores the

score_percent

,

reasoning

, and

summary

.

    - *Professional Note:* We store raw JSON response data here as well (

raw_response_data

) for debugging purposes.

5. EvaluationDetail **Table:** This is the most granular table. It stores every single question and answer pair.
    - *Relationship:* One

EvaluationResult has many

EvaluationDetails

.

    - *Key Fields:*

is_correct

(Boolean),

partial_credit

(Float),

feedback

(Text).

Why this Schema is good: It is **Normalized**. If a student re-uploads a file, we don't have to delete the whole assignment; we only update the specific

AssignmentFile and its

EvaluationResult.

2. WILL THEY ASK ABOUT THE "CODE PART"?

Yes. But they won't just ask you to read code. They will ask **"What happens if..."** questions.

Common "Code Logic" Questions:

- **"How do you handle the Async part?"** - They want to hear about

asyncio.create_task(scheduled_cleanup())

in

main.py. Explain that cleanup runs in the background so the API stays fast.

- **"How do you handle 'Dirty Data' (bad files)?"** - Explain your

try-except

blocks in

file_processor.py. You don't let the whole app crash if one file is corrupt.

- **"How do you prevent SQL Injection?"** - Answer: "By using **SQLAlchemy ORM**, which automatically parameterizes queries."

3. INTERNSHIP SELECTION: THE "HIDDEN" EVALUATION

Beyond the code, a Senior Manager is looking for these 3 things to decide if they should hire you:

A. Ownership

Don't say "The code does this." Say **"I implemented this logic because..."** Show that you made choices, not just copied a tutorial.

- *Example:* "I chose to separate the

EvaluationDetail from the

EvaluationResult so that we could easily generate a question-by-question report for students."

B. Error Handling Mindset

A junior dev writes code that works when everything is perfect. A **Senior Intern** writes code that handles failure.

- *Tip:* Talk about how you handle **OpenRouter timeouts** or **OCR failures**. Mention that you added a "Fallback" to local Tesseract if the Cloud OCR fails.

C. Security Awareness

Even if it's a small project, mention:

- "We use **Bcrypt** for passwords because storing plain text is a security risk."
- "We use **JWT Tokens** to ensure that User A cannot see User B's assignments."

4. KEY PHRASES TO USE (To Sound Like a Pro)

Use these "Buzzwords" during your review to impress the interviewers:

- **"Stateless Backend":** "The FastAPI app is stateless, meaning we can scale it easily horizontally."
- **"Separation of Concerns":** "I kept the file extraction logic in a separate service (

FileProcessor) so that the API routers stay clean."

- **"Graceful Degradation":** "If the vision model is unavailable for PPTs, the system gracefully degrades to text-only evaluation."
- **"Data Integrity":** "We use Foreign Key constraints in PostgreSQL to ensure we don't have 'orphan' evaluation results if an assignment is deleted."

Final Advice for your First Review:

If you don't know an answer, **don't guess.** Instead, say:

"That's a great architectural question. In the current version, I handled it this way, but for a production-scale system, I would look into [Redis/S3/Websockets] to improve that."

This shows you have the "Growth Mindset" they are looking for in an intern! Good luck!

FRONTEND

For the **Frontend**, reviewers specifically look for how you handle complexity, user experience (UX), and communication with the backend. Since this is your first internship review, they want to see that you understand **State Management** and **Asynchronous UI**.

Here is what you need to know about your frontend:

1. THE ARCHITECTURE: REACT + TAILWIND + AXIOS

Your project uses a modern "Single Page Application" (SPA) architecture:

- **Framework: React.js** (Functional Components + Hooks).
- **Styling: Tailwind CSS** (for responsive, premium UI design).
- **Communication: Axios** (for API calls to the FastAPI backend).
- **Routing: React Router** (handles

/login

,

/dashboard

, and

/services

).

2. CORE FRONTEND LOGIC (Crucial for Interviews)

A. Protected Routes (

App.jsx**)**

You've implemented **Route Guards**. If a user is not logged in (

isAuthenticated === false

), the app automatically redirects them to

/login

. This shows you understand application security.

B. API Interceptors (

axios.js**)**

This is a "Senior-level" move. You have a **Request Interceptor** that automatically grabs the JWT token from

localStorage

and attaches it to every single outgoing API request in the

Authorization

header.

- *Why this matters:* It's efficient. You don't have to manually add the token in every component; the Axios instance does it for you.

C. Complex State Management (

Services.jsx**)**

This is the most important file in your frontend. You are managing multiple "Modes":

- mode === 'files'

(Basic File Upload)

- mode === 'ppt'

(Presentation Evaluation)

- mode === 'github'

(Codebase Analysis)

The "Generate" Flow:

1. You use a

FormData

object to bundle physical files for the

/files/upload

endpoint.

2. You handle **Progressive States**:
    - setIsGenerating(true)

shows a loading spinner.

- Once the API responds, you update

setResult

,

setSummary

, and

setScores

.

- setIsGenerating(false)

hides the spinner and shows the data.

3. WHAT QUESTIONS WILL THEY ASK?

Q: "How do you handle multiple file uploads?"

- **Answer:** "I use an array state (

files). When a user selects files, I convert the

FileList

to an array and append it to the state. During submission, I loop through these and append them to a

FormData

object to send them to the backend."

Q: "What happens if the backend server is down?"

- **Answer:** "I've implemented

try-catch

blocks around my API calls. If the backend is unreachable, the error is caught, and I update the

error

state to show a friendly 'Cannot connect to server' message to the user."

Q: "How do you handle such large JSON results from the LLM?"

- **Answer:** "I created utility functions in

utils/helpers.js

(like

parseJsonResult

and

renderJsonResult

). They take the raw string from the AI, validate that it's proper JSON, and then map it into the UI components."

Q: "Why did you use Tailwind CSS?"

- **Answer:** "Speed and consistency. Tailwind allows me to build a responsive, professional UI without writing thousands of lines of custom CSS. It also makes things like 'Hover effects' and 'Transitions' very easy to implement."

4. KEY UI/UX FEATURES YOU BUILT

If they ask "What did you do for the user?", point these out:

1. **Drag and Drop:** You implemented a

dragActive

state that changes the border color when a file is hovering over the upload box.

2. **Excel/PDF Export:** You didn't just show results on screen; you added the ability to **download reports**. This makes the tool "Business Ready."
3. **Real-time Status Check:** In

useEffect

, the app immediately checks if the backend and the AI model (OpenRouter) are online and shows a status indicator.

4. **Re-evaluation Button:** Users can change their grading description and hit "Re-evaluate" on a specific student without re-uploading the file. This shows you thought about the "User Workflow."

FINAL TIPS for selecting you as an Intern:

1. **"Component-Based Thinking":** Mention how you broke the UI into small pieces like

Navbar

,

GitHubRepo

, and

PPTUpload

so the code is easy to maintain.

2. **"Responsiveness":** Open your project in a browser and resize it to show it works on mobile. This proves you understand **Mobile-First Design**.
3. **"Validation":** Point out that your

Generate button stays disabled until the user fills in both the Title and Description. This prevents "bad requests" to the server.

In summary: Your frontend isn't just a "pretty face"—it's a robust application that manages complicated data states, handles errors gracefully, and communicates securely with the backend. You are ready!

To give you the best chance of selection, here are the **Top 10 "Senior Architect" Questions** that combine frontend, backend, and database knowledge. These are the "hard" questions that internal reviewers use to see if an intern truly understands the whole system.

---

1. THE FULL-STACK FLOW

Q: "If a user clicks 'Generate', walk me through exactly what happens across the 3 layers (Browser, Server, DB)?"

- **Answer:**
    - **Frontend:** React bundles the files into a

FormData

object and sends a POST request with a JWT token.

    - **Backend:** FastAPI validates the token, saves the physical file to the

uploads/

folder, and creates an

AssignmentFile entry in the **Database**.

    - **Processing:** The logic in

generate_service_complete.py extracts the text, sends it to the AI, and calculates the score.

    - **Persistence:** The final JSON result is saved in the

EvaluationResult and

EvaluationDetail tables.

- **Frontend Update:** React receives the success response and maps the DB result into the UI.

2. DATABASE RELATIONS

Q: "Why did you use a One-to-Many relationship between

EvaluationResult **and**

EvaluationDetail**?"**

- **Answer:** "Because one student evaluation (Result) consists of multiple individual questions (Details). This structure allows us to store specific feedback for Question #1 separately from Question #2, making the data searchable and easier to display in a table on the frontend."

3. PERFORMANCE & SCALING

Q: "AI requests take a long time (30+ seconds). How do you prevent the frontend from timing out or freezing?"

- **Answer:** "On the **frontend**, I use an

isGenerating

state to show a loading spinner and disable the button to prevent double-clicks. On the **backend**, I've set an

OPENROUTER_TIMEOUT

of 300 seconds to allow for large documents, and I use

async

endpoints in FastAPI so the server can handle other users while waiting for the AI."

4. DATA INTEGRITY

Q: "What happens if a user deletes an Assignment? Do the files stay on the disk forever?"

- **Answer:** "I've handled this in two ways. First, I use

cascade='all, delete-orphan'

in the **SQLAlchemy models**. If an Assignment is deleted, the database automatically deletes all linked Results. Second, I have a **Cleanup Service** in the backend that runs every 24 hours to delete files from the disk that are no longer referenced in the database."

5. SECURITY & VALIDATION

Q: "How do you ensure a user can't upload a malicious script instead of a PDF?"

- **Answer:** "We have three layers of defense.

    1. **Frontend:** I use the

accept

attribute on the file input.

    2. **Backend:** The

FileProcessor only attempts to read files with specific extensions.

    3. **Extraction:** Since we only extract **text** from the files to send to the AI, even if a script is uploaded, it is never 'executed'—it's just treated as a string of words."

6. STATE SYNCHRONIZATION

Q: "After a Re-evaluation, how do you update the UI without refreshing the whole page?"

- **Answer:** "In

Services.jsx, I use index-based updating. When the re-evaluation API returns the new score for one student, I find their index in the

scores

array and update ONLY that specific object using

setScores(updatedScores)

. This makes the UI feel fast and reactive."

7. ERROR MANAGEMENT

Q: "What if the LLM returns a broken JSON string? How does the system handle it?"

- **Answer:** "In the backend, I use **Regex** as a fallback to extract JSON from the AI's response if it includes extra text. If it's completely unparseable, the

generate_service_complete.py service catches the error and returns a structured error message, which the frontend displays in a red error alert."

8. USER EXPERIENCE (UX)

Q: "Why did you choose to store the 'Last Description' and 'Last Title' in the frontend state?"

- **Answer:** "To support the **'Re-evaluate' workflow**. It allows the teacher to tweak the grading instructions and click 'Re-evaluate' without having to re-type everything or re-upload 50 student files. It saves the user time and reduces server bandwidth."

9. API DESIGN

Q: "Why use separate routers for

/auth

,

/files

, and

/github

instead of putting everything in

main.py**?"**

- **Answer:** "This is for **Code Maintainability**. By using

APIRouter

, we follow the 'Single Responsibility Principle.' It makes it easier for a team of developers to work on different features (like Auth vs. Evaluation) without causing merge conflicts in one giant file."

10. AUTHENTICATION FLOW

Q: "Explain how the 'Remember Me' functionality works (JWT in LocalStorage)?"

- **Answer:** "When the user logs in, the backend generates a **JWT (JSON Web Token)**. We store this in

localStorage

. In

App.jsx, we have a

useEffect

that checks for this token on page load. If found, we set

isAuthenticated(true)

. This prevents the user from having to log in every time they refresh the page."

💡 **Pro-Tip for your Interview:**

If they ask **"Which part was the hardest to build?"**, talk about the **Integration of the three parts**.

"The hardest part was ensuring that the data extracted from the Physical File (Backend) was correctly indexed so that the AI's feedback (AI Layer) matched the right student name in the Database and displayed correctly in the React UI (Frontend)."

Saying this proves you have "Big Picture" thinking!

Good

Bad

Terminal (0 Background Processes Running)