FairTestAI: LLM Assessment Vulnerability Simulator

A web application to simulate malicious prompt injections and their effects on LLM responses for academic assessments. Demonstrates how subtle text manipulations can alter AI behavior.

Table of Contents

Project Structure

```
repo-root/

  backend/
      app/
          __init__.py
          models.py
          routes/
              assessments.py
          services/
              pdf_utils.py
              attack_service.py
              wrong_answer_service.py
              openai_eval_service.py
      migrations/
      requirements.txt
      run.py
```

```
frontend/
    src/
        App.tsx
        index.tsx
        components/
            Header.tsx
            Footer.tsx
            ControlPanel.tsx
            PdfUpload.tsx
            DownloadLinks.tsx
    index.html
    package.json
    vite.config.ts

README.md
...
```

Backend (Flask)

Backend Architecture & Flow

- **Flask** serves as the API backend.
- **PostgreSQL** is used for persistent storage (via SQLAlchemy).
- **PDFs** are parsed, attacked, and re-generated using PyPDF2, pdfminer.six, and XeLaTeX.
- **LLM (OpenAI)** is used to simulate and evaluate prompt injection vulnerabilities.
- **API** endpoints are exposed for the frontend to upload PDFs and retrieve results.

Typical Flow

1. User uploads a question PDF (and optionally an answer key PDF) via the frontend.
2. Backend parses the PDF, injects a malicious instruction, and generates an "attacked" PDF.
3. The backend may call an LLM to simulate how the attack affects answers.
4. Results (attacked PDF, reference report) are stored and made available for download.

Backend Key Modules

- **app/init.py**: App factory, configures Flask, DB, CORS, and registers blueprints.
- **app/routes/assessments.py**: Main API endpoints for uploading assessments, generating attacked PDFs, and serving results.
- **app/services/pdf_utils.py**: PDF parsing, LaTeX generation, and attack injection logic.
- **app/services/attack_service.py**: Implements different attack types (e.g., hidden malicious instruction).
- **app/services/wrong_answer_service.py**: Generates plausible wrong answers for MCQs.
- **app/services/openai_eval_service.py**: Handles LLM API calls for evaluation (if enabled).
- **app/models.py**: SQLAlchemy models for assessments, files, questions, and LLM responses.

Backend Database Models

- **StoredFile**: Represents any uploaded or generated file (PDFs).
- **Assessment**: Represents an assessment session, links to original, answer, attacked, and report PDFs.
- **Question**: Represents a parsed question from the PDF.
- **LLMResponse**: Stores LLM-generated answers and rationales.

Backend API Endpoints

- **POST /api/assessments/upload**
  Uploads a question PDF (and optional answer key), triggers attack injection, and returns an assessment ID.
- **GET /api/assessments//attacked**
  Downloads the attacked PDF.
- **GET /api/assessments//report**
  Downloads the reference report PDF.

Backend PDF Generation & Attacks

- **PDF Parsing**: Uses PyPDF2 and pdfminer.six to extract questions and options.
- **Attack Injection**: Adds hidden or visible malicious instructions to the LaTeX source.

- **LaTeX Generation**: Uses XeLaTeX and `fontspec` for Unicode support.
- **Output**: Generates both an attacked PDF and a reference report.

Frontend (React)

Frontend Architecture & Flow

- **React** (with TypeScript) is used for the UI.
- **Vite** is used for fast development and build tooling.
- **Tailwind CSS** is used for styling.
- **API Integration**: Communicates with the Flask backend via REST endpoints.

Typical Flow

1. User uploads PDFs via the UI.
2. User clicks "Run Attack" to submit to the backend.
3. UI shows loading, errors, or download links for results.

Frontend Key Components

- **App.tsx**: Main app container, manages state and orchestrates uploads and downloads.
- **Header/Footer**: Branding and info.
- **ControlPanel**: Buttons for submitting and clearing uploads.
- **PdfUpload**: Handles file selection for question and answer PDFs.
- **DownloadLinks**: Shows download links for attacked and reference PDFs.

Frontend API Integration

- **assessmentService.ts**: Handles API calls to upload PDFs and retrieve results.
- **Error Handling**: Displays user-friendly error messages for upload or processing failures.

Setup & Running Locally

Prerequisites

- Node.js (for frontend)
- Python 3.10+ (for backend)
- PostgreSQL (for backend DB)

- XeLaTeX (for PDF generation, e.g., via MacTeX)
- OpenAI API key (for LLM evaluation, if enabled)

Backend

```
cd backend
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
export OPENAI_API_KEY=sk-...    # if using LLM features
flask db upgrade                # initialize database
flask run
```

Frontend

```
cd frontend
npm install
npm run dev
```

Troubleshooting

See the README.md for common issues, including:

- OpenAI API key errors
- XeLaTeX or pdflatex not found
- Unicode/LaTeX errors

Code Walkthrough

Backend

- **App Initialization**:
  `backend/app/__init__.py` sets up Flask, DB, and CORS.
- **Assessment Upload**:
  `backend/app/routes/assessments.py` handles `/upload`, saves files, parses PDFs, injects attacks, generates PDFs, and stores results.
- **PDF Generation**:
  `backend/app/services/pdf_utils.py` parses questions, injects attacks, and generates LaTeX, which is compiled with XeLaTeX.
- **Attack Logic**:
  `backend/app/services/attack_service.py` defines how attacks are injected (e.g., hidden malicious instructions).

- **LLM Integration**:
  `backend/app/services/openai_eval_service.py` and `wrong_answer_service.py` handle LLM calls and fallback logic.
- **Database Models**:
  `backend/app/models.py` defines all persistent entities.

Frontend

- **App State**:
  `frontend/src/App.tsx` manages file uploads, API calls, and UI state.
- **Component Structure**:
  - `Header.tsx` and `Footer.tsx`: Branding and info.
  - `ControlPanel.tsx`: Action buttons.
  - `PdfUpload.tsx`: File inputs.
  - `DownloadLinks.tsx`: Shows download links after processing.
- **API Calls**:
  `frontend/src/services/assessmentService.ts` handles all communication with the backend.

Extending the Project

- **Add New Attack Types**:
  Implement new strategies in `attack_service.py` and update the frontend if you want user selection.
- **Support More Question Types**:
  Enhance PDF parsing logic in `pdf_utils.py`.
- **Improve LLM Evaluation**:
  Add more robust LLM response analysis in `openai_eval_service.py`.
- **UI Enhancements**:
  Add progress bars, more detailed error messages, or richer result displays in React.

Additional Notes

- All file uploads and generated files are stored in `backend/data/assessments/<uuid>/`.
- Output PDFs are also copied to `backend/output/` for easy access.
- The system is designed for research and educational use—**not for production**.