Project Documentation

Overview

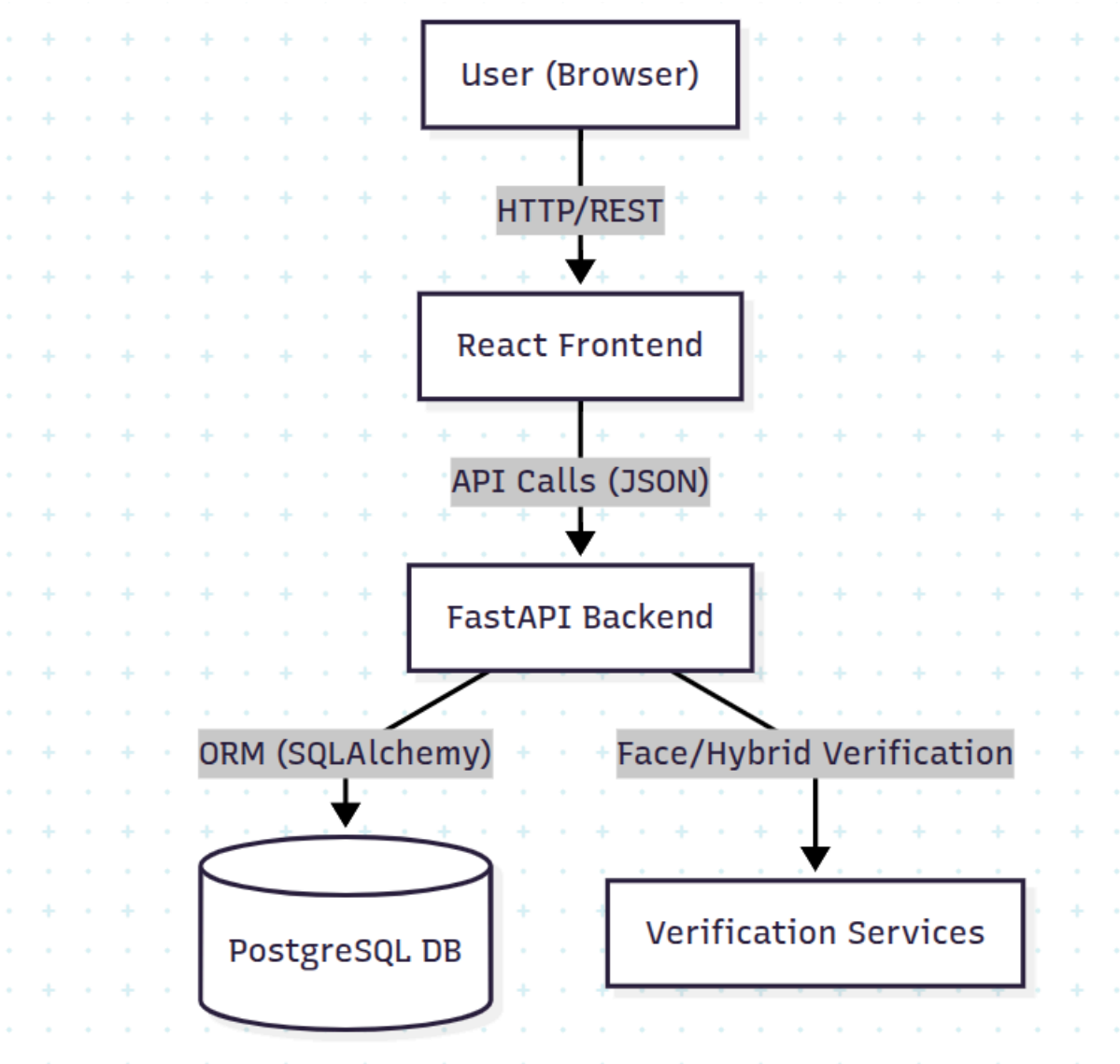
This project is an online proctoring and exam platform with real-time face and behavior monitoring, built with a React frontend and a Python FastAPI backend. It uses computer vision to detect violations (like tab switches, fullscreen exits, multiple faces, device detection, etc.) and stores all events in a PostgreSQL database for review.

Table of Contents

Sure! Here's the cleaned-up version without brackets and hashtags:

12. Troubleshooting

Architecture



Backend - FastAPI

Location: `backend/`
Main entry: `main.py`

Key features:

- REST API for reporting and retrieving violations
- Face and hybrid verification using YOLO, MediaPipe, and custom logic
- Stores violations in PostgreSQL (`report.violations`)
- Handles image uploads and management

Key files:

- `main.py`: FastAPI app, endpoints, DB session management
- `models.py`: SQLAlchemy models (Violation, etc.)
- `face_verification.py`, `hybrid_verification.py`: Computer vision logic
- `detection.py`, `recognition.py`: Supporting detection logic
- `manage_images.py`: Utility for managing face images

Frontend - React

Location: `src/`
Main entry: `src/App.tsx`

Key features:

- User authentication and exam flow
- Real-time webcam feed and face capture
- Exam rules, timer, and progress
- Displays detected violations in a results table
- Communicates with backend via REST API

Key files:

- `src/pages/`: Main pages (ExamDetails, FaceRecognition, Quiz, Results, etc.)
- `src/components/`: UI components (WebcamFeed, WarningDialog, ExamTimer, etc.)
- `src/utils/quizSecurity.ts`: Handles tab switch and fullscreen detection

Database - PostgreSQL

Table: `report.violations`

Columns:

- `id` (PK)
- `student_id`
- `exam_id`
- `violation_type`
- `confidence`
- `timestamp`
- `details`

Connection: Managed via SQLAlchemy in `backend/models.py`
Remote host: e.g., `blackbuck-stage.postgres.database.azure.com`

Setup & Installation

Backend

1. Install dependencies:
   Run this in the `backend` directory:
   `cd backend`

```
pip install -r requirements.txt
```

2. Set environment variables:
    - `DATABASE_URL` (PostgreSQL connection string)
    - Any other required keys (see `.env` or `models.py`)
3. Run backend:
   uvicorn main:app --host 0.0.0.0 --port 5000 --reload

Frontend

1. Install dependencies:
   ```
   npm install
   ```
2. Run frontend:
   ```
   npm run dev
   ```

API Endpoints

Violations

- `POST /report_violation`
  Report a new violation (fields: student_id, exam_id, violation_type, details, confidence)
- `GET /get_violations?student_id=...&exam_id=...`
  Retrieve all violations for a student and exam

Hybrid/Face Verification

- `POST /hybrid_analyze`
  Analyze a webcam frame for violations
- `POST /reset_tracking`
  Reset tracking state

Violation Detection Logic

- Tab switch and fullscreen exit are detected in the frontend (`quizSecurity.ts`) and reported to the backend.
- Multiple faces, looking away, head turning, and device detection are handled in the backend using YOLO, MediaPipe, and custom logic.
- Duplicate prevention is implemented by ignoring violations of the same type that occur within 2 seconds (this interval is configurable).

AI Models, Thresholds, and Detection Logic

Face Verification and Proxy Detection

- Main class: `FaceVerificationService`
- Embedding model: `InceptionResnetV1` from `facenet-pytorch` (used via `FaceRecognizer` in `backend/recognition.py`)

Reference Image Loading

- `FaceVerificationService.load_reference_images(student_id)`
  Loads and embeds 1 to 3 reference images per student.

Verification Method

- `FaceVerificationService.verify_face(student_id, live_image_base64, threshold=None)`
  - Loads reference embeddings if not already loaded
  - Detects face in the live image and computes its embedding
  - Compares live embedding to each reference embedding using Euclidean distance
  - Uses adaptive thresholding based on the number of reference images
  - Supports enhanced logic for 3-angle verification:

        - Checks average and standard deviation of distances
        - Requires at least 2 of 3 views to be close
    - Returns a result dictionary containing:
        - `verified`
        - `best_distance`
        - `threshold`
        - Per-view distances

Verification Logic (Simplified Snippet)

```
if distance < threshold:
    best_match = True

if not best_match and len(distances) >= 3:
    avg_distance = np.mean(list(distances.values()))
    std_distance = np.std(list(distances.values()))
    if avg_distance < 0.88 and std_distance < 0.12:
        best_match = True
    close_views = sum(1 for d in distances.values() if d < 0.90)
    if close_views >= 2:
        best_match = True

verified = best_match
```

Proxy Detection

- If `verified` is `False`, the system logs a proxy violation indicating that a different person was detected.

Face and Behavior Detection - Hybrid Verification

Main class: `HybridVerificationService` (located in `backend/hybrid_verification.py`)

Object detection:

- Uses `YOLOv8n` (Ultralytics) for detecting people, faces, and electronic devices
- Initialized as: `self.yolo_detector = YOLO('yolov8n.pt')`

MediaPipe Face Mesh:

- Used for head pose and gaze estimation
- Initialized as: `self.face_mesh = mp.solutions.face_mesh.FaceMesh(...)`

Detection steps (within `process_frame` method):

1. **Person and Face Detection**
   Method: `_detect_person_and_face(frame)`
   - Runs YOLO on the input frame
   - Filters detections based on object class and confidence score
2. **Multiple People Detection**
   Method: `_detect_multiple_people_with_persistence(persons)`
   - Checks if more than one person is detected with confidence above 0.3
   - Ensures that detected faces are at least 50 pixels apart
   - Requires that the condition persists for at least 1.0 second
3. **Comprehensive Violation Detection**
   Method: `_detect_comprehensive_violations(frame, violations)`
   - Detects multiple types of behavioral violations including:
     - **Head turning (pose):**
       - Uses six key landmarks to compute yaw and pitch angles
       - Flags a violation if `abs(yaw) > 30` degrees or `abs(pitch) > 20` degrees

- **Gaze/Eye Aspect Ratio (EAR):**
  - Calculates EAR for both eyes
  - Triggers violations if `avg_ear < 0.2` (eyes closed) or `avg_ear > 0.5` (abnormal wide eyes)
- **Device Detection:**
  - Monitors for presence of device classes (YOLO classes 67, 73, 62)
  - Requires detection confidence greater than 0.5
  - Device must be continuously visible for at least 1.0 second

Key Code Snippet – Hybrid Verification

```
backend/hybrid_verification.py
def _detect_comprehensive_violations(self, frame, violations):
    if abs(yaw) > 30 or abs(pitch) > 20:
        violations['head_turning'] = True
    if avg_ear < 0.2 or avg_ear > 0.5:
        violations['looking_away'] = True
    if detection_duration >= self.device_min_duration:
        violations['device_detected'] = True
    if face_count > 1 and faces_are_apart:
        violations['multiple_faces'] = True
```

This logic checks for behavioral violations like head turning, abnormal eye aspect ratio, visible devices, and multiple faces in the frame.

Duplicate Prevention

Before saving a violation in `main.py`, the system checks if a similar violation already exists within a recent time window (2 seconds):

```
exists = db.query(Violation).filter(
    Violation.student_id == data.student_id,
    Violation.exam_id == data.exam_id,
```

```
    Violation.violation_type == v_type,
    Violation.timestamp >= time_window_start
).first()

if not exists:
    db.add(Violation(...))
```

This prevents redundant database entries and ensures that the same violation type isn't logged repeatedly for the same student and exam within a short interval.

Class and Method Reference

FaceVerificationService (`face_verification.py`)

- `load_reference_images(student_id)`
- `verify_face(student_id, live_image_base64, threshold=None)`

FaceRecognizer (`recognition.py`)

- `get_embedding(face_img)`

HybridVerificationService (`hybrid_verification.py`)

- `process_frame(frame, student_id)`
- `_detect_person_and_face(frame)`
- `_detect_multiple_people_with_persistence(persons)`
- `_detect_comprehensive_violations(frame, violations)`

Example: Full Face Verification Flow

1. Student registers by uploading 1 to 3 reference images (front, left, and right profiles).

2. The backend embeds these reference images and stores them in memory.

3. During the exam, the frontend captures a live frame and sends it to the backend.

4. The backend extracts the face from the frame, computes its embedding, and compares it to the stored reference embeddings.

5. If the comparison passes threshold and logic (e.g., multi-angle matching), the student is verified. Otherwise, a proxy violation is logged.

Example: Full Violation Detection Flow The frontend captures a webcam frame and sends it to the `/hybrid_analyze` API endpoint.

1. The backend processes the frame using YOLO (for object detection) and MediaPipe (for face mesh and pose estimation).

2. It checks for behavioral and visual violations such as:

    - Multiple faces
    - Head turning
    - Looking away
    - Device detection

3. If a violation is detected and hasn't already been logged in the last 2 seconds, it is saved to the database.

4. The frontend retrieves and displays these violations in a results table in real time or at the end of the exam.

Environment Variables

- `DATABASE_URL` – PostgreSQL connection string (used in `backend/models.py`)
- Other variables as needed for cloud services, API keys, or model paths

How to Run

1. Start the backend server (FastAPI)
2. Start the frontend server (React)
3. Open the app in your browser (usually at `http://localhost:5173`)
4. Use the application to take an exam, and view violations on the results page

Troubleshooting

- **Database connection issues:**
    - Ensure the `DATABASE_URL` is correct
    - Check your network and PostgreSQL SSL settings
- **Violations not appearing:**
    - Check backend logs for detection or saving errors
    - Confirm violations are being saved in the database
    - Verify the frontend is grouping and displaying them correctly
- **Face/Hybrid detection errors:**
    - Ensure all required model weights (YOLO, face mesh, etc.) are downloaded and accessible
    - Verify all dependencies are installed as per `requirements.txt`