Coding Challenge: Deep Work Session Tracker

Objective

Design and build a **Deep Work Session Tracker** system that allows users to plan, start, pause, resume, and complete focused work sessions. The system must log session activity, track interruptions, and provide summaries for productivity reflection.

The system should include:

- A **FastAPI backend** with an **SQLite database**
- A **ReactJS frontend** to interact with the API
- A **Python SDK**, generated using **OpenAPI Generator CLI**
- Automation scripts (`setupdev.bat` , `runapplication.bat`)

Tasks & Requirements

1. Backend Development (FastAPI & SQLite)

Implement the following endpoints:

1. **POST /sessions/** → Schedule a new work session (title, duration in minutes, goal).
2. **PATCH /sessions/{session_id}/start** → Start the session.
3. **PATCH /sessions/{session_id}/pause** → Pause the session (log the interruption reason).
4. **PATCH /sessions/{session_id}/resume** → Resume a paused session.
5. **PATCH /sessions/{session_id}/complete** → Mark session as completed (store completion time).
6. **GET /sessions/history** → Get a summary of past sessions with durations, pauses, and completion ratio.

🚨 Trick Logic in Backend (Hidden Complexity)

- A session can only be paused if it's currently active.
- If the user **pauses more than 3 times**, mark the session as “interrupted”.
- A session must be completed within **10% of its original duration**, otherwise it’s “overdue”.
- If a session was never resumed after a pause, it should be marked as “abandoned”.
- The system should store **pause reasons** (e.g., phone call, Slack, distraction).

📌 Examples:

✅ Session A: 50 minutes, paused 2× → status: completed
❌ Session B: paused 4× → status: interrupted
❌ Session C: not resumed after pause → status: abandoned
✅ Session D: completed 60 minutes after 50-minute duration → status: overdue

---

2. Database Schema (SQLite)

```
CREATE TABLE sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    goal TEXT,
    scheduled_duration INTEGER NOT NULL, -- in minutes
```

```sql
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    status TEXT CHECK (status IN ('scheduled', 'active', 'paused', 'completed', 'interrupted', 'abandoned', 'overdue')) DEFAULT 'scheduled',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE interruptions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id INTEGER NOT NULL,
    reason TEXT NOT NULL,
    pause_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (session_id) REFERENCES sessions(id) ON DELETE CASCADE
);

```

- Use **Alembic** for schema migrations
- Include a sample SQL (`seed_data.sql`) with pre-scheduled and completed sessions

---

3. Frontend (ReactJS)

Your frontend should allow users to:

1. **Schedule a session** (enter title, goal, and duration)
2. **Start, pause, resume, and complete** a session
3. **Enter reason when pausing**
4. **View session history**, including status and stats
5. **(Bonus)** Timer countdown while active

Use **Axios** to call the API.
Minimal design is fine — usability is the focus.

---

4. Python SDK (OpenAPI Generator)

Generate a Python SDK:

```
openapi-generator-cli generate -i http://localhost:8000/openapi.json -g python -o deepwork_sdk

```

Sample Usage:

```python
from deepwork_sdk.api.sessions_api import SessionsApi
from deepwork_sdk import ApiClient
client = ApiClient()
api = SessionsApi(client)
# Schedule a session
api.create_session(title="Write project doc", goal="Finish outline", scheduled_duration=45)

```

Write a **sample script** using the SDK.

---

5. Setup Script (`setupdev.bat`)

```bat
@echo off
echo Setting up backend...
```

```
python -m venv env
call env\Scripts\activate
pip install -r requirements.txt
alembic upgrade head
echo Setting up frontend...
cd frontend
npm install
cd ..

```

---

6. Run Script (`runapplication.bat`)

```
@echo off
echo Starting backend...
call env\Scripts\activate
python main.py
echo Starting frontend...
cd frontend
npm start

```

Evaluation Criteria

✅ Session state transitions handled correctly
✅ Robust validation (e.g., cannot pause before starting)
✅ Summary history reflects accurate status
✅ SDK works with sample script
✅ Clean frontend UX for managing sessions
✅ Setup scripts run smoothly
✅ README with instructions and insights
✅ Unit tests for session and interruption logic

Deliverables

- FastAPI backend
- SQLite DB & Alembic migrations
- ReactJS frontend
- Python SDK
- Sample Python script
- Setup and run scripts
- Unit tests
- README

💡 Bonus Features

- Real-time session timer
- Weekly productivity reports
- CSV export of sessions

- “Focus score” based on interruption ratio

Good luck!