1. 파일 헤더 · 이벤트릿 패치 / 기본 import

* eventlet.monkey\_patch() 를 다른 모듈 import 전에 호출(윈도우 greendns 회피 env 설정 포함).
* 표준 라이브러리 + flask, flask\_socketio, requests, pymongo(선택) 등 불러옵니다.
* 목적: 비동기 소켓 + HTTP 서버 동작을 위한 준비.

1. 환경변수 로드 / 상수 설정

* .env 또는 .env.example에서 OPENAI\_API\_KEY, OPENAI\_MODEL, PORT, MONGO\_URI, STABILITY\_SECONDS, COOLDOWN\_SECONDS, REPEATER\_INTERVAL 등을 읽어옵니다.
* 기본값이 없으면 안전한 기본값을 사용(예: 포트 8000).

1. Flask + SocketIO 설정

* app = Flask(...) 및 socketio = SocketIO(app, async\_mode=...) 생성.
* async\_mode로 eventlet 사용을 시도하고, 실패 시 threading으로 폴백.

1. MongoDB 연결(선택) 및 로컬 저장 대체

* MONGO\_URI가 있으면 pymongo로 접속 시도. 실패하거나 없으면 로컬 JSON(chats\_local.json)에 메시지 저장.
* 함수: save\_chat(doc) — Mongo가 있으면 DB에, 없으면 로컬 파일에 append.

1. 텍스트 정리 유틸리티들

* strip\_code\_fence, extract\_json\_from\_text, clean\_llm\_text 등: LLM (OpenAI) 응답에서 코드 펜스/불필요문자 제거, 텍스트 내 JSON 추출 등을 담당.
* 목적: LLM 응답을 안정적으로 파싱해서 TTS/질문 텍스트로 사용하기 위함.

1. 졸음지수 → 단계 변환 (stage mapping)

* stage\_from\_D(D) : D값(0~100)을 받아서 단계 문자열 반환 (정상, 의심경고, 집중모니터링, 개선, L1, L2, L3, FAILSAFE).
* 임계값 매핑은 코드 내부에 하드코딩되어 있음(필요시 수정 가능).

1. LLM(REST) 헬퍼: 단계별 알림/질문 생성

* generate\_stage\_message(stage\_name) : OpenAI API 호출해서 {announcement, question} 형태의 JSON을 리턴하려 시도.
* OpenAI 키가 없거나 호출 실패 시 fallback\_map에 정의된 기본 문구를 반환.
* 호출 실패/응답 파싱을 안전하게 처리하도록 다양한 예외·정리 로직 포함.

1. 안정화 / 쿨다운 로직 전역 변수

* \_last\_stage, \_pending\_stage, \_pending\_since, \_last\_emit\_ts 등으로 전이(히스테리시스)와 재발 방지(쿨다운) 관리.
* STABILITY\_SECONDS(관찰이 안정화되어야 emit), COOLDOWN\_SECONDS(같은 단계 재발송 방지).

1. 반복 전송(Repeater) — 경고를 주기적으로 반복 송출

* start\_stage\_repeater(stage\_name, ann\_text, q\_text, interval, max\_repeats) : 스레드로 즉시 한 번 emit하고 주기적으로 socketio.emit("state\_prompt", ...) 및 chat\_message로 반복 전송.
* stop\_stage\_repeater() 로 기존 반복 스레드를 안전히 종료.
* 각 emit은 고유 msg\_id 생성(sha1)하여 중복 처리/클라이언트에서 별도 취급 가능.

1. / (index) 엔드포인트

* templates/index.html 또는 static/index.html을 찾아서 반환(없으면 404). UI 파일을 템플릿/정적에 두면 브라우저에서 확인 가능.

1. /ingest (핵심: D값 수신 및 상태 전이)

* 동작 흐름 요약:
  1. POST로 JSON 수신 (D, ts) → 클라이언트에 d\_update 이벤트 바로 emit(숫자 실시간 표시용).
  2. stage = stage\_from\_D(D) 계산.
  3. 만약 stage가 \_last\_stage와 같다면 pending 리셋 후 아무 작업 안 함.
  4. \_pending\_stage가 달라지면 pending 시작(시간 기록) — 안정화 대기(STABILITY\_SECONDS).
  5. pending이 안정화되면 emit\_lock으로 동시성 보호 후 다시 확인(중복/쿨다운/텍스트 중복 체크) → generate\_stage\_message() 호출 → 기존 repeater 중단 후 새로운 repeater 시작.
* 반환값에 pending, emitted, reason(중복·쿨다운 등) 정보를 줌.

1. 테스트/보조 엔드포인트

* /test\_emit?stage=L2 : 수동으로 특정 단계에 대해 반복(최대 3회) 시작 — 디버깅/테스트용.
* /save\_msg : 메시지를 DB/파일에 저장하고 chat\_message로 emit.
* /chat\_send : 사용자가 보낸 텍스트를 받아 OpenAI로 보내 LLM 응답을 받고 chat\_message로 emit 및 저장(OPENAI\_API\_KEY 없을 땐 에코형 응답 제공).
* /health : 헬스체크.

1. SocketIO 이벤트 핸들러

* connect, disconnect 로그 출력만 있음. 클라이언트는 d\_update, state\_prompt, chat\_message 이벤트를 받도록 구현하면 됨.

1. \_\_main\_\_ 실행

* socketio.run(app, host="0.0.0.0", port=PORT, ...) 로 서버 실행. 콘솔에 stability/async\_mode 정보 출력.