perplexity

what are the API's we are using the how the deep clear backend works ?

Here's a **deep, clear breakdown of the APIs used and how the backend of Flix works**:

APIs Used (Core & External):

- **Internal Backend APIs** (built with FastAPI/Node.js):
  - **Authentication API:** User registration, login, and secure access
  - **User Profile API:** Get/update driver info (experience, confidence, vehicle, etc.)
  - **Route Planning API:** Accepts source, destination; returns safest and most adaptive route (calling AI engine)
  - **Hazard/SOS Reporting API:** Users or mesh nodes send alerts; backend stores and relays to affected routes
  - **Feedback/Trip Sync API:** Uploads finished trip data for AI retraining and analytics
  - **Mesh Network API:** Manages how device-to-device messages are handled and relayed, supports real-time mesh updates and syncs when back online
- **External/Third-party APIs:**
  - **Mapbox API:** For map tiles, offline map downloads, and live traffic overlays when online
  - **Weather API:** (e.g., OpenWeatherMap or IMD) For real-time weather by location—fetched on route planning and periodically during trip
  - **Traffic API:** (Mapbox Traffic, HERE, or TomTom) Gets current congestion, accident, and road closure data for routing

Backend Data Flow (Step-by-Step):

1. **User logs in:** Calls Authentication API → user is returned a token.
2. **Profile fetched:** Calls User Profile API to get skill, preferences, and vehicle info.
3. **Route planned:** Calls Route Planning API.
   - The backend fetches traffic and weather from external APIs, merges with hazard history and mesh data.
   - AI engine (TensorFlow Lite model) predicts safest, fastest personalized route.
   - Returns the route, hazard points, and map download links to the app.
4. **Trip starts:** Backend logs trip start; triggers trip-based offline map and hazard data download.

5. **During trip:**
    - Hazard or SOS events are sent/received via Mesh API (Bluetooth/Wi-Fi Direct); backend sync happens when user is back online.
    - If connected, live route updates, traffic/weather changes, and crowd alerts come via real-time backend push notifications.
6. **Trip ends:** Trip and feedback data is posted to Feedback/Trip Sync API; AI training and analytics updated on backend.

Special Tech Stack Points:

- **TensorFlow Lite:** AI routing happens mostly on-device but can be coordinated via backend for new data and retraining.
- **MongoDB Atlas:** Stores all user, route, hazard, and usage analytics, built for scale.
- **Mesh relays:** Edge ESP32 modules interface with backend and help bridge data when phones see intermittent connectivity.
- **AWS/Firebase:** General cloud sync, user authentication, and push notification infrastructure.

Summary
Your backend is modular, cloud-native, built for the Indian context, and combines:

- High-speed APIs for the app,
- Real-time traffic and weather from global APIs,
- Crowd-sourced and mesh-powered data for "always-on" safety and navigation—even when offline.[1]

This covers the API stack and deep backend logic you'll need for any Q&A or technical grilling!

⁂

1. Flix_Project_Context.pdf