🧠 SafeSchool Software Architecture (Full Overview)

The software stack has **4 major layers**:

```
[1] Firmware (Device Level) →  [2] Cloud Backend (API + DB)
          ↓                              ↓
[3] AI/Analytics Engine          [4] Frontend Dashboard
```

Each layer has its own purpose and technologies 👇

🧩 1. Firmware Layer (On ESP32 Watch)

🧰 Language & Tools

- **Language:** C / C++ (Arduino framework or ESP-IDF)
- **IDE:** Arduino IDE / PlatformIO
- **Libraries used:**
  - `Wire.h` → I²C communication (MAX30102, MPU6050)
  - `Adafruit_Sensor.h` → for generic sensor interface
  - `MAX30102.h` → heart rate readings
  - `MPU6050.h` → motion data
  - `TinyGPS++` → GPS parsing
  - `WiFi.h` → network connectivity
  - `HTTPClient.h` → to POST data to backend

⚙️ Firmware Functionality

Sensor Initialization

```
setup() {
    initWiFi();
    initMAX30102();
    initMPU6050();
    initGSR();
    initGPS();
}
```

1.
2. **Continuous Data Collection**
    - Reads HR, HRV, GSR, motion, temperature, GPS.
    - Calculates running average & deviation.

Stress Detection Logic

```
if (HR > baselineHR + 20 && HRV < baselineHRV * 0.7 && GSR > threshold && motion < low_motion) {
    sendAlert("Distress");
}
```

3.

Fall Detection

```
if (accelMagnitude > 2.0 && motion == 0) {
    sendAlert("Fall Detected");
}
```

4.
5. **Data Transmission**

Sends JSON packets to the backend every 10–30 seconds:

```
{
  "device_id": "child_001",
  "heart_rate": 94,
  "hrv": 28,
  "gsr": 0.42,
```

```
  "temperature": 36.5,
  "motion": "still",
  "lat": 26.1435,
  "lon": 91.7898,
  "alert": "none"
}
```

-

Uses HTTPS POST:

```
POST /api/v1/ingest
Content-Type: application/json
```

-

6. **Local Feedback**
    - Vibration Motor → activated if alert
    - SOS Button → manual trigger
    - Sleep mode → saves power

🧠 Firmware Output

- Stress events
- Fall detection events
- Route tracking coordinates
- Battery and uptime info
- All data → pushed to cloud backend

☁️ 2. Cloud Backend Layer

This is the **bridge** between IoT devices and the dashboard.

🧰 Technology Stack

Component	Option 1	Option 2
Backend Framework	Node.js + Express.js	Python + Flask/FastAPI
Database	PostgreSQL + TimescaleDB (for time-series)	MongoDB Atlas
Deployment	Render / AWS EC2 / Railway	Firebase / Heroku
Authentication	JWT (JSON Web Token)	
Notification Service	Twilio (SMS), SendGrid (Email), Firebase Push	
Storage	AWS S3 / Google Cloud Storage (for logs or backups)	

⚙️ Key API Endpoints

Endpoint	Method	Purpose
`/api/v1/ingest`	POST	Receive live sensor data
`/api/v1/alerts`	GET	Fetch all alerts (SOS/distress/fall)
`/api/v1/dashboard`	GET	Data for dashboard visualization
`/api/v1/login`	POST	User authentication (JWT)
`/api/v1/children/:id`	GET	Retrieve individual child data

🧮 Backend Responsibilities

1. **Data Ingestion**
    - Receives data packets from ESP32.
    - Validates & stores into PostgreSQL (time-series schema).

Example table:

```
CREATE TABLE sensor_data (
  id SERIAL PRIMARY KEY,
  device_id TEXT,
  heart_rate FLOAT,
  hrv FLOAT,
  gsr FLOAT,
  temperature FLOAT,
  motion TEXT,
  lat FLOAT,
  lon FLOAT,
  timestamp TIMESTAMP DEFAULT NOW()
);
```

    -
2. **Alert Management**
    - Evaluates thresholds server-side.
    - If condition → triggers notification (SMS/email).
3. **API for Dashboard**
    - Aggregates last 24-hour data for each user.
    - Provides averages, graphs, and trends.
4. **Authentication & Security**
    - JWT tokens for user login.
    - HTTPS + CORS + rate limiting (`helmet`, `express-rate-limit`).
5. **Notification Engine**

- Twilio for SMS → “Child distress detected near school gate.”
- SendGrid for email → “Alert: Route deviation detected.”
- Firebase for push → app notifications.

🤖 3. AI / Analytics Layer

This is the **intelligence** that converts sensor data → insights.

🧰 Technology Stack

- **Language:** Python
- **Libraries:** NumPy, Pandas, Scikit-learn, TensorFlow/Keras
- **Data Pipeline:** Cron jobs or background tasks using Celery

---

⚙️ Core Logic

1. **Data Preprocessing**
   - Filter noise from HR/GSR readings.
   - Calculate **HRV** (time-domain method using R–R intervals).
   - Normalize values by child’s baseline.
2. **Feature Engineering**
   - ΔHR, ΔHRV, ΔGSR, ΔTemp, motion variance
   - Derived stress index:
     $S = w_1(\Delta HR) + w_2(\Delta HRV) + w_3(GSR) + w_4(Motion)$
   - Score scaled to 0–100.
3. **Anomaly Detection**

Uses rolling mean + z-score threshold:

```
if zscore(stress_score) > 2.5:
    alert = "stress_spike"
```

4. **Learning Component**
    - Model learns individual baselines per child.
    - Adaptive thresholds: less false alarms.
5. **Analytics Outputs**
    - Daily stress trend graphs.
    - Alert frequency per time of day.
    - Route deviation patterns.
6. **Optional ML Upgrade**
    - Deep learning (LSTM) for sequence analysis of HRV & motion.
    - CNN-based classification if you ever add audio or image sensors later.

📊 Example AI Output

Time	HR	HRV	GSR	Stress Score	Label
09:30	82	45	0.15	32	Normal
10:15	101	22	0.41	76	Distress
11:05	92	35	0.25	40	Calm

🖥️ 4. Dashboard Frontend Layer

Your **React.js** dashboard is what parents, teachers, and admins see.

🧰 Technology Stack

Component	Tool
Frontend Framework	React.js (with Hooks)
Data Visualization	Chart.js / Recharts
Mapping	Leaflet.js / Mapbox
HTTP Client	Axios
Auth Management	JWT + Context API
Deployment	Netlify / Vercel

⚙️ Frontend Features

1. **Login & Auth**
    - JWT authentication for role-based access (Parent / Teacher / Admin).
2. **Live Status Panel**
    - Real-time child state: 🟢 Safe | 🟠 Alert | 🔴 SOS
    - Uses WebSocket or polling API.
3. **Stress Trends**
    - Line chart of stress score vs time.
    - Color-coded for normal/stress/distress zones.
4. **Route Map**
    - Leaflet map shows:
        - Current location pin
        - Expected route (polyline)

        - Deviation markers

5. **Alert History**
    - Table of alerts with timestamp, type, GPS, and resolved status.
6. **Analytics Dashboard**
    - Weekly and monthly reports.
    - Top “stress hours”, daily averages, etc.
7. **Admin View**
    - Monitor all children in a school simultaneously.
    - Filter by class, grade, or location.

🧠 Example React Component Snippet

```
useEffect(() => {
  axios.get("/api/v1/dashboard")
    .then(res => setData(res.data))
    .catch(err => console.error(err));
}, []);

return (
  <div>
    <h2>Child Stress Trend</h2>
    <LineChart data={data.stressScores} />
  </div>
);
```

🔐 5. Security & Privacy

Layer	Measures
Device	Encrypted Wi-Fi communication, token auth
Cloud	HTTPS, rate-limiting, hashed credentials

Database	Anonymized IDs, no personal data beyond parent link
Frontend	JWT storage in HttpOnly cookies

🔁 6. Data Flow Summary

```
Sensors → ESP32 (C++) → Wi-Fi → Cloud API (Express/Flask)
          ↓                                   ↓
       Local Alerts                  PostgreSQL + AI Model
          ↓                                   ↓
       Dashboard (React) ← API / WebSocket ← Analytics Layer
```

💡 7. Example Tech Stack Summary

Layer	Technology	Purpose
Hardware	ESP32, MAX30102, MPU6050, GPS	Sensing & data collection
Firmware	Arduino C++	Data processing, alerts
Cloud	Node.js + Express + PostgreSQL	Data storage, API, alerting
AI Layer	Python + Scikit-learn	Stress scoring, anomaly detection
Frontend	React.js + Chart.js + Leaflet	Visualization & alerts
Deployment	Render + Netlify	Cloud hosting

🌍 8. Future Enhancements

- Replace HTTP with MQTT for continuous data streaming.
- Add **mobile app (React Native)** for parents.
- Implement **Edge AI** model directly on ESP32.
- Use **Firebase Realtime DB** for live dashboards.
- Integrate **Google Maps Distance Matrix API** for smarter route tracking.