**Project Documentation**

INSIGHT STREAM

**1. Introduction**

• Project Title: INSIGHT STREAM

• Team ID : NM2025TMID37020

• Team Leader: SANDHIYA G & [sandhiya2ndnov@gmail.com](mailto:sandhiya2ndnov@gmail.com)

• Team Members: REBECCA R & [rebe8195@gmail.com](mailto:rebe8195@gmail.com)

REVATHY J & [revathyjeyachandran1612@gmail.com](mailto:revathyjeyachandran1612@gmail.com)

ROJA J & [rojaroja6324@gmail.com](mailto:rojaroja6324@gmail.com)

2 **Project** **Overview**

• Purpose: Insight Stream leverages AI to analyse large volumes of customer conversations (support chats, calls, reviews, feedback) and automatically generate actionable insights. It helps businesses discover key themes, identify trends, and empower leaders to make data-driven decisions in real time.

Extended Purposes:

. Bridge Communication Gaps: Ensure that feedback from frontline teams and customers directly reaches management in a structured, actionable format.

. Improve Customer Experience: Detect recurring pain points, compliance issues, or product improvement needs quickly, so businesses can act before problems escalate

. Enable Personalization: Allow businesses to set up custom “streams” for topics like customer satisfaction, retention, or product features, tailoring insights to their goals.

. Promote Transparency: Keep stakeholders at all levels aligned with clear visibility into customer feedback and team performance.

. Support Scalability: As organizations grow, ensure they can still listen to the “voice of the customer” without being overwhelmed by data volume.

. Drive Continuous Improvement: Provide feedback loops that guide product enhancements, training, and customer service upgrades.

real-time communication.

• Features:

. Executive-level summaries with drill-down into themes or individual conversations

. Customizable streams (retention, compliance, product issues, etc.)

. Real-time tracking of customer voice and sentiment

. Feedback and reporting system for decision-makers

. Admin control panel for configuration and user management

3. **Architecture**

• Frontend: React.js with Bootstrap and Material UI

• Backend: Node.js and Express.js managing server logic and API endpoints

• Database: MongoDB stores user data, project information, applications, and chat messages

**4. Setup Instructions**

Node.js (v16 or later)

MongoDB (local or MongoDB Atlas for cloud database)

Git (for cloning repository)

React.js (comes with create-react-app)

Express.js & Mongoose (installed via npm)

Visual Studio Code (or any IDE of choice)

Installation steps:

Step 1: Clone the Repository

git clone <repository-url>

cd Insight-Stream

Step 2: Install Frontend Dependencies

cd client

npm install

This installs all required React.js packages including Material UI, Bootstrap, and Axios.

Step 3: Install Backend Dependencies

cd ../server

npm install

This installs Express.js, Mongoose, JWT, bcrypt, and other backend dependencies.

Step 4: Configure Environment Variables

Create a .env file inside the server/ directory with the following variables:

PORT=5000

MONGO\_URI=mongodb://localhost:27017/insightstream # or MongoDB Atlas URI

JWT\_SECRET=your\_jwt\_secret\_key

Step 5: Setup MongoDB

If running locally, make sure MongoDB service is active:

mongodb

If using MongoDB Atlas, replace MONGO\_URI with your connection string.

Step 6: Run the Application

Frontend (React App):

cd client

npm start

Backend (Node.js Server):

cd ../server

npm start

step 7: Access the Application

Open http://localhost:3000 in your browser for the frontend

Backend runs on `http://localhost:5000` by default

../server npm install

5. Folder Structure

**Insight-stream/**

│── README.md # Project overview & instructions

│── requirements.txt # Python dependencies (if Python project)

│── environment.yml # (optional) for conda environment

│── setup.py # (if packaging project)

│── .gitignore # Ignore unnecessary files

├── data/ # Raw & processed data

│ ├── raw/ # Original datasets (unmodified)

│ ├── processed/ # Cleaned/transformed datasets

│ └── streaming/ # Real-time incoming data sample

├── notebooks/ # Jupyter/Colab notebooks for exploration

│ ├── EDA.ipynb # Exploratory Data Analysis

│ ├── preprocessing.ipynb # Data cleaning steps

│ └── model\_training.ipynb # ML model experiments

├── src/ # Main project source code

│ ├── \_\_init\_\_.py

│ ├── config/ # Config files (YAML/JSON)

│ ├── data\_pipeline/ # Scripts for data collection & processing

│ ├── features/ # Feature engineering scripts

│ ├── models/ # ML/AI models

│ ├── visualization/ # Dashboard / plotting scripts

│ └── utils/ # Helper functions

├── tests/ # Unit and integration tests

│ ├── test\_data\_pipeline.py

│ ├── test\_models.py

│ └── test\_utils.py

├── reports/ # Documentation & results

│ ├── figures/ # Graphs and plots

│ ├── presentations/ # PPTs or PDFs

│ └── final\_report.pdf

├── dashboards/ # Frontend / visualization apps

│ ├── streamlit\_app.py # Example Streamlit app

│ ├── flask\_app.py # Example Flask/Django app

│ └── templates/ # HTML templates if using Flask

└── logs/ # Logging files (pipeline runs, errors)

**6. Running the Application**

Running the Application – Insight Stream

Follow these steps to set up and run the project locally:

1 Clone the Repository

Bash

Git clone <https://github.com/REBECCA351/insightstream.git>

Cd insightstream

2 Create a Virtual Environment

Bash

Python -m venv venv

Activate it:

Windows

Bash

Venv\Scripts\activate

Linux / Mac

Bash

Source venv/bin/activate

3 Install Dependencies

Bash

Pip install -r requirements.txt

4 Set Environment Variables (if required)

If your project uses API keys, database URLs, or other configs:

Bash

Cp .env.example .env

Then edit .env with your values.

5 Run Database Migrations (if backend is used)

For Django:

Bash

Python manage.py migrate

For Flask with SQLAlchemy:

Bash

Flask db upgrade

6 Start the Application

Streamlit Dashboard

Bash

Streamlit run dashboards/streamlit\_app.py

Flask App

Bash

Python app.py

Django App

Bash

Python manage.py runserver

Command-line Execution

Bash

Python main.py

7 Open in Browser

Streamlit → <http://localhost:8501>

Flask → <http://127.0.0.1:5000>

Django → http://127.0.0.1:8000

**7. API Documentation**

Insight Stream – API Documentation

🔹 Base URL

<http://localhost:5000/api/v1>

(Change host/port if deployed on server)

Authentication

Currently, no authentication is required (add JWT/Auth later if needed).

Endpoints

1. Health Check

Endpoint:

GET /health

Description: Check if the API server is running.

Response:

Json

{

“status”: “ok”,

“message”: “Insight Stream API is running”

}

1. Ingest Data (Streaming Input)

Endpoint:

POST /data/ingest

Description: Accepts raw data (IoT/Real-time inputs).

Request Body:

Json

{

“device\_id”: “sensor\_01”,

“timestamp”: “2025-09-17T10:30:00Z”,

“value”: 42.5

}

Response:

Json

{

“status”: “success”,

“message”: “Data ingested successfully”

}

1. Get Processed Insights

Endpoint:

GET /data/insights

Description: Returns processed analytics from streaming data.

Query Params:

Device\_id (optional) → Filter by device

Limit (optional) → Number of records to fetch

Response:

Json

{

“device\_id”: “sensor\_01”,

“avg\_value”: 41.8,

“max\_value”: 56.2,

“min\_value”: 35.4,

“last\_updated”: “2025-09-17T10:35:00Z”

}

1. List Devices

Endpoint:

GET /devices

Description: Get list of all devices/sensors sending data.

Response:

Json

{

“devices”: [

{“device\_id”: “sensor\_01”, “status”: “active”},

{“device\_id”: “sensor\_02”, “status”: “inactive”}

]

}

1. Visualize Data (Optional)

Endpoint:

GET /visualize/{device\_id}

Description: Returns a chart or JSON data for visualization.

Response (JSON):

Json

{

“device\_id”: “sensor\_01”,

“time\_series”: [

{“time”: “2025-09-17T10:00:00Z”, “value”: 40.5},

{“time”: “2025-09-17T10:05:00Z”, “value”: 42.1}

]

}

🔹 Error Responses

Status Code

Meaning

Example Response

400

Bad Request

{ “error”: “Invalid input data” }

404

Not Found

{ “error”: “Device not found” }

500

Server Error

{ “error”: “Internal server error” }

**8. User Interface**

User Interface for Insight Stream

1 Login / Home Page

If authentication is required → Login screen with username & password.

Otherwise, a landing page with project title:

“Insight Stream – Real-Time Data Insights”

2 Dashboard Page

Main screen showing real-time insights.

Components:

Header → Project name + user profile icon.

Sidebar (Navigation)

Dashboard

Devices

Insights

Reports

Settings

Main Area (Cards + Charts)

KPI Cards (Top row):

Total Devices Active

Last Updated Time

Avg Value Today

Alerts (if any abnormal data)

Real-Time Graphs

Line chart → Streaming data over time

Bar/Donut chart → Device status (active/inactive)

Heatmap → Usage patterns

3 Devices Page

Table showing:

| Device ID | Status | Last Seen | Avg Value | Actions |

Buttons: Add Device, Remove Device, View Logs

4 Insights Page

Processed insights in text + visuals:

Example: “Sensor\_01 shows an increasing trend of 12% in the last hour.”

Download option for CSV/PDF report.

5 Reports Page

Generate reports (daily/weekly/monthly).

Export → CSV / PDF.

6 Settings Page

Manage environment variables, thresholds, and alert notifications.

UI Implementation Options

Streamlit ( easiest for data dashboards)

Streamlit\_app.py → real-time charts + insights.

Flask/Django + HTML/CSS/JS ( full web app).

React + Chart.js/Recharts ( modern UI with dynamic visuals).

Example Streamlit UI Layout (Python snippet)

Python

Import streamlit as st

Import pandas as pd

Import time

St.title(“ Insight Stream Dashboard”)

# KPI Cards

Col1, col2, col3 = st.columns(3)

Col1.metric(“Active Devices”, 5)

Col2.metric(“Average Value”, “42.5”)

Col3.metric(“Last Updated”, “10:35 AM”)

# Real-time Chart

Chart = st.line\_chart()

Data = []

For I in range(50):

New\_value = {“time”: I, “value”: i\*2}

Data.append(new\_value)

Chart.add\_rows(pd.DataFrame([new\_value]))

Time.sleep(0.2)

This would give you a live updating chart+KPIs

**9. Testing**

Testing for Insight Stream

Testing ensures that your application runs without errors, APIs return correct results, and the UI behaves as expected.

1 Unit Testing

Tests for individual functions and modules.

Example files:

Tests/

├── test\_data\_pipeline.py

├── test\_models.py

├── test\_utils.py

Example (Pytest)

Python

# tests/test\_utils.py

From src.utils.data\_cleaner import clean\_data

Def test\_clean\_data\_removes\_nulls():

Raw\_data = [{“device”: “sensor1”, “value”: None}, {“device”: “sensor2”, “value”: 42}]

Cleaned = clean\_data(raw\_data)

Assert all(d[“value”] is not None for d in cleaned)

Run tests:

Bash

Pytest -v

2 Integration Testing

Tests how different modules work together (e.g., data ingestion + processing).

Example:

Python

Def test\_ingest\_and\_insight(client):

# POST data

Response = client.post(“/data/ingest”, json={

“device\_id”: “sensor\_01”,

“timestamp”: “2025-09-17T10:30:00Z”,

“value”: 42.5

})

Assert response.status\_code == 200

# GET insights

Insights = client.get(“/data/insights?device\_id=sensor\_01”)

Assert “avg\_value” in insights.json

3 API Testing

Use Postman / Thunder Client or pytest + requests to test endpoints.

Example (using requests):

Python

Import requests

BASE\_URL = <http://127.0.0.1:5000/api/v1>

Def test\_health():

R = requests.get(f”{BASE\_URL}/health”)

Assert r.status\_code == 200

Assert r.json()[“status”] == “ok”

4 UI Testing

If Streamlit → manually verify KPIs, graphs, and updates.

If React/Flask UI → use Selenium / Playwright to automate browser clicks.

Example (Selenium):

Python

From selenium import webdriver

Driver = webdriver.Chrome()

Driver.get(<http://127.0.0.1:8501>)

Assert “Insight Stream” in driver.title

Driver.quit()

5 Performance Testing

Simulate high data load (many devices sending data).

Use Locust or JMeter.

Example (Locust snippet):

Python

From locust import HttpUser, task

Class InsightUser(HttpUser):

@task

Def send\_data(self):

Self.client.post(“/data/ingest”, json={

“device\_id”: “load\_test”,

“timestamp”: “2025-09-17T12:00:00Z”,

“value”: 55.1

})

Run:

Bash

Locust -f load\_test.p

checklist

[ ] Unit tests for data pipeline

[ ] API endpoint tests (Postman/pytest)

[ ] Integration tests for ingestion → insights

[ ] UI validation (manual/automated)

[ ] Load testing for performance



