Sensor Monitoring System – Documentation

Overview  
  
This project is a real-time sensor data monitoring and visualization system. It features:  
- Real-time data ingestion via WebSocket  
- Signal decoding, filtering, and feature extraction  
- Machine learning-based classification  
- Storage in a PostgreSQL/TimescaleDB database  
- A Tkinter-based GUI for live visualization and analysis

# Project Structure

sensor\_monitoring\_system/  
├── app/  
│ ├── ws/  
│ ├── db/  
│ ├── processing/  
│ ├── ml/  
│ ├── utils/  
│ └── auth/  
├── main.py  
├── run\_all.py  
├── tkinter\_ui.py  
├── test.py  
├── requirements.txt  
├── readme.md

# Top-Level Files

main.py  
- Purpose: Main backend entry point.  
- Functionality: Loads environment variables, trains the ML model from CSV, starts the WebSocket server, and logs startup/shutdown events.  
  
run\_all.py  
- Purpose: Unified launcher for backend and UI.  
- Functionality: Initializes the database schema, starts the backend in a separate thread, and launches the Tkinter UI in the main thread.  
  
tkinter\_ui.py  
- Purpose: Main GUI for live data visualization.  
- Functionality:   
 - Fetches latest samples from the database.  
 - Plots raw and filtered signals.  
 - Displays classification results.  
 - Allows filtering by client/channel.  
 - Provides refresh and auto-refresh controls.  
  
test.py  
- Purpose: Test client for simulating sensor data.  
- Functionality:   
 - Connects to the WebSocket server.  
 - Sends random float arrays as encoded hex payloads.  
 - Receives and prints classification results.  
  
requirements.txt  
- Purpose: Lists all Python dependencies.  
  
readme.md  
- Purpose: Project overview, setup instructions, and usage guide.

# app/ Subfolders

app/ws/websocket\_server.py  
- Purpose: WebSocket server for real-time data ingestion.  
- Key Functions:  
 - handle\_client: Handles incoming WebSocket connections, decodes and processes data, extracts features, classifies with ML, filters the signal, and stores results.  
 - moving\_average: Simple moving average filter for signal smoothing.  
 - start\_server: Starts the WebSocket server.  
  
app/db/  
db\_handler.py  
- Purpose: Database access and manipulation.  
- Key Functions:  
 - get\_connection: Connects to PostgreSQL.  
 - get\_kanal\_id\_by\_name: Gets channel ID by name.  
 - get\_channels\_for\_client: Lists all channels.  
 - initialize\_schema: Loads and applies the schema from schema.sql.  
 - insert\_sample\_data: Inserts raw/filtered signals, features, and classification into the database.  
 - fetch\_latest\_samples: Retrieves recent samples with features and classification.  
 - authenticate\_user: Authenticates users (for future use).  
  
schema.sql  
- Purpose: Defines the database schema.  
- Tables:  
 - messplattform: Sensor platforms.  
 - kanal: Sensor channels.  
 - datenpunkt: Stores raw and filtered signal arrays.  
 - merkmale: Extracted features.  
 - proben: Classification results.  
- Indexes: For efficient time-based and relational queries.  
  
app/processing/  
decoder.py  
- Purpose: Decodes hex-encoded signal data from microcontrollers.  
- Key Function:   
 - decode\_signal: Converts hex string to a list of floats or uint16s.  
  
feature\_extractor.py  
- Purpose: Extracts statistical features from a signal array.  
- Key Function:   
 - extract\_features: Computes mean, median, std, variance, range, count above mean, quantiles, mean abs, and form factor.  
  
app/ml/  
model.py  
- Purpose: Machine learning model for classification.  
- Key Functions:  
 - load\_model: Loads and trains a KNN classifier from CSV.  
 - classify\_sample: Predicts the class for a given feature vector.  
  
trainer.py  
- Purpose: Trains the ML model from a CSV file.  
- Key Function:   
 - train\_from\_csv: Loads data and trains the model.  
  
Trainingsdaten\_Timmi.csv  
- Purpose: Training data for the ML model.  
- Format: CSV with label and 9 feature columns.  
  
app/utils/logger.py  
- Purpose: Simple logging utility.  
- Key Function:   
 - log: Prints timestamped log messages with levels (INFO, ERROR, etc.).  
  
app/auth/jwt\_handler.py  
- Purpose: JWT-like token authentication for WebSocket clients.  
- Key Function:   
 - verify\_jwt: Checks if the provided token matches the server secret.

# Data Flow

1. Sensor/Test Client sends encoded signal data via WebSocket.  
2. WebSocket Server decodes, filters, extracts features, classifies, and stores the data.  
3. Database stores all raw/filtered signals, features, and classification results.  
4. Tkinter UI fetches and visualizes the latest data, showing both raw and filtered signals and the predicted class.

# Setup & Usage

1. Install dependencies: pip install -r requirements.txt  
2. Configure .env file with DB credentials and JWT secret.  
3. Run the application: python run\_all.py  
4. (Optional) Use test.py to simulate sensor data.

# Security

- Uses a simple token-based authentication for WebSocket clients.  
- User authentication logic is present for future expansion.

# Extensibility

- Easily add new feature extraction methods in feature\_extractor.py.  
- Swap out the ML model in model.py for more advanced algorithms.  
- Add more UI features in tkinter\_ui.py.