



CAPSTONE DESIGN IN COMPANY PROJECT



AI-Based PMD failure prediction



TEAM





PARTICIPATING COMPANY



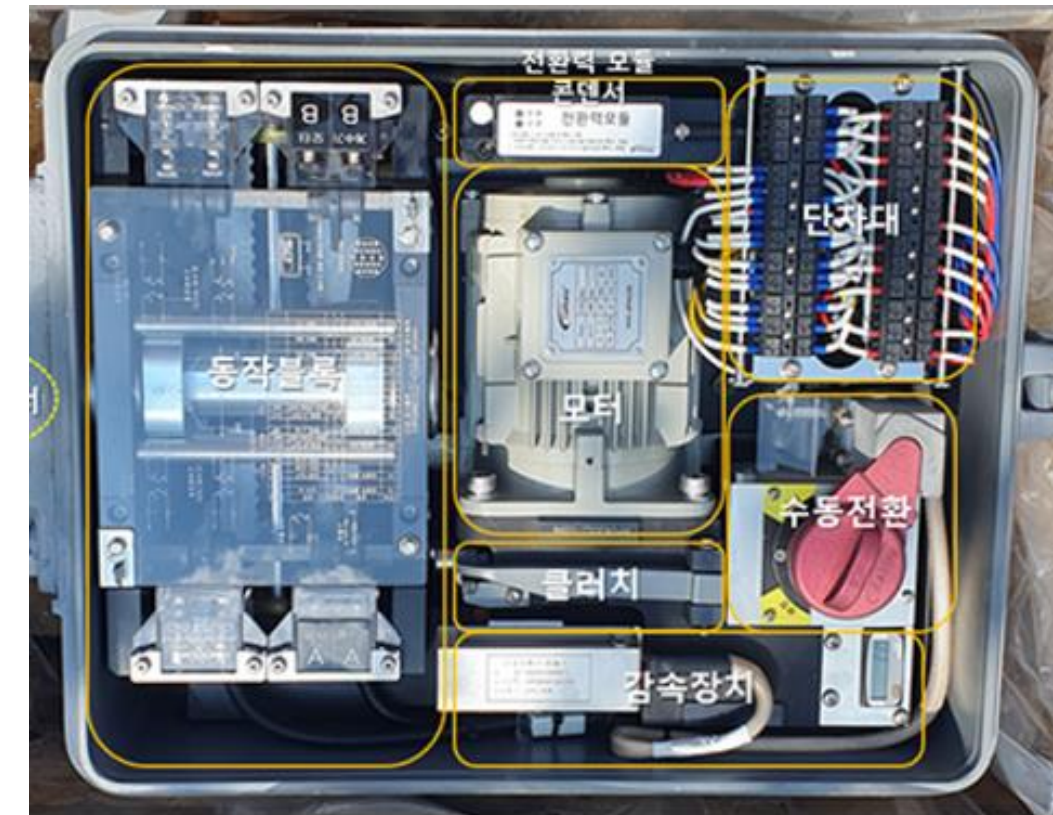
Sehwa Co., Ltd



철도 선로전환기



Team Visit to Sehwa



AGENDA

1. Motivation

2. Our Solution

3. AI Progress

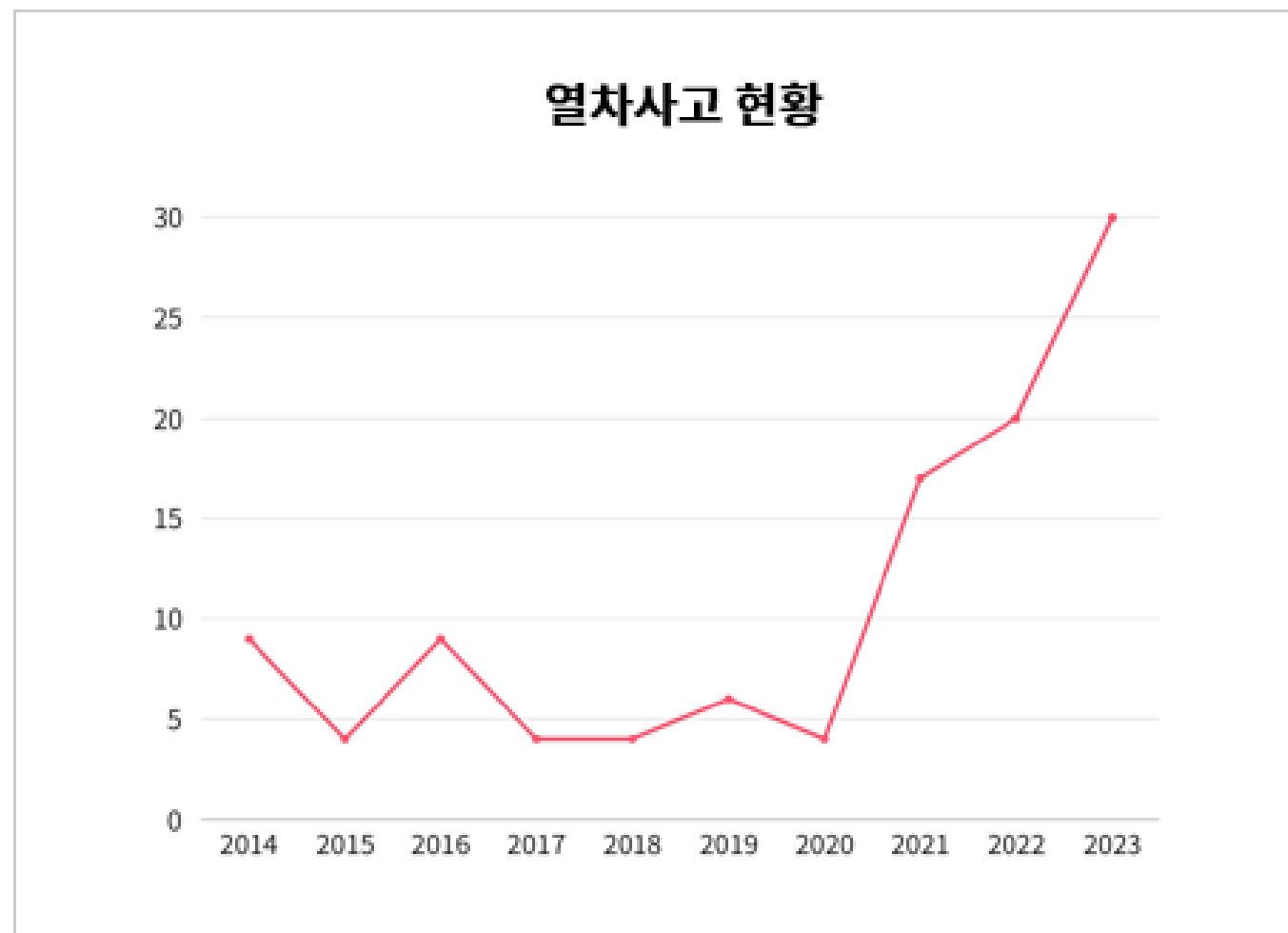
4. Result

5. Demo

6. Future Plan

Motivation

Current Status of Train Accidents on the Rise



출처 : 교통안전공단 철도안전정보포털(www.railsafety.or.kr)

Increase in train accident rates

- Train accidents surged
from 4 to 30 in 2020~23

The most accident is

- According to the analysis of railway signal control system failures from 2000 to the last 10 years, PMD failures accounted for 27%

Development of error prediction model

- Predicting the possibility of failure of the track converter to minimize train accidents

Our Solution

Sensor data

- Huge amount of Sensor data from Point Machine



- Pre-processing unnecessary parts and storing them in the database

AI Capabilities

- Training AI models for error detection and prediction



- Detect anomalies and predict potential failures with high accuracy

Our Solution

AI based Failure Prediction

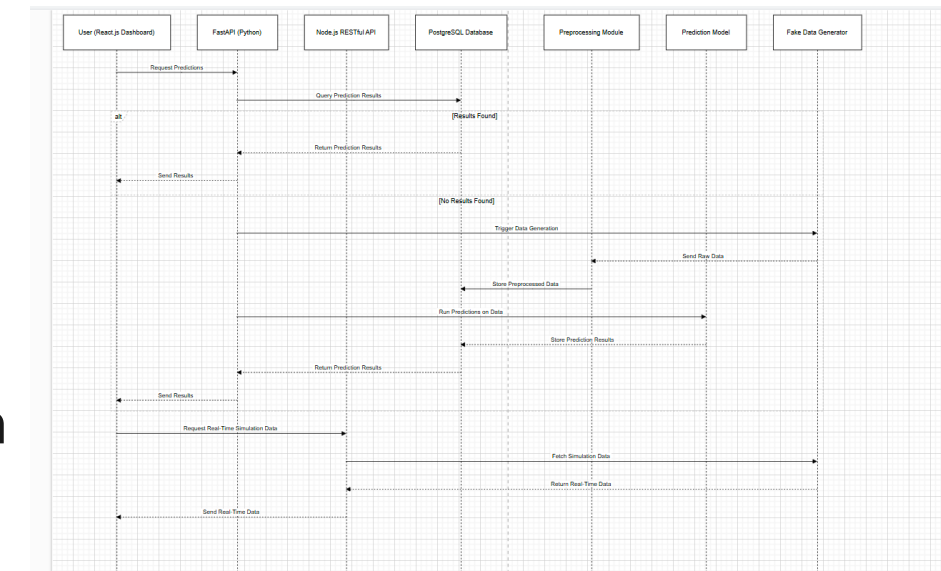
AI

- AI powered error detection
- AI powered failure prediction

```
+ Code + Text
[ ] 1 import pandas as pd
    2 import numpy as np
    3 import zipfile
    4 from sklearn.model_selection import train_test_split
    5 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
    6 from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
    7 from sklearn.preprocessing import MinMaxScaler
    8 from tensorflow.keras.models import Sequential
    9 from tensorflow.keras.layers import Dense, LSTM, Dropout
   10 from imblearn.combine import SMOTETomek
   11
   12 # Step 1: Load and Preprocess the Dataset
   13 def preprocess_data(file_path, num_rows=1000):
   14     data = pd.read_csv(file_path, nrows=num_rows)
   15     data_cleaned = data.dropna(axis=1, how='all').dropna()
   16     numerical_columns = data_cleaned.select_dtypes(include='number').columns
   17     data_cleaned = data_cleaned[numerical_columns]
   18     np.random.seed(42)
   19     data_cleaned['target'] = np.random.choice([0, 1], size=data_cleaned.shape[0], p=[0.9, 0.1])
   20     X = data_cleaned.drop('target', axis=1)
   21     y = data_cleaned['target']
   22     return X, y
   23
   24 # Step 2: Handle Class Imbalance and Split Data
   25 def balance_and_split_data(X, y):
   26     scaler = MinMaxScaler()
```

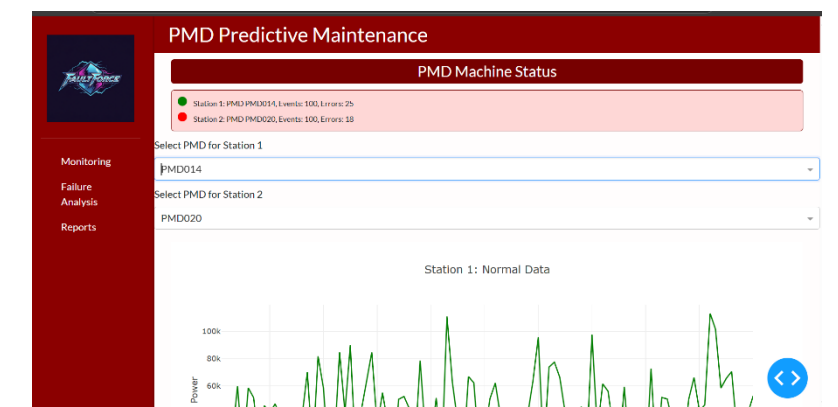
Backend

- Storing data
- Integrating AI models with Dashboard

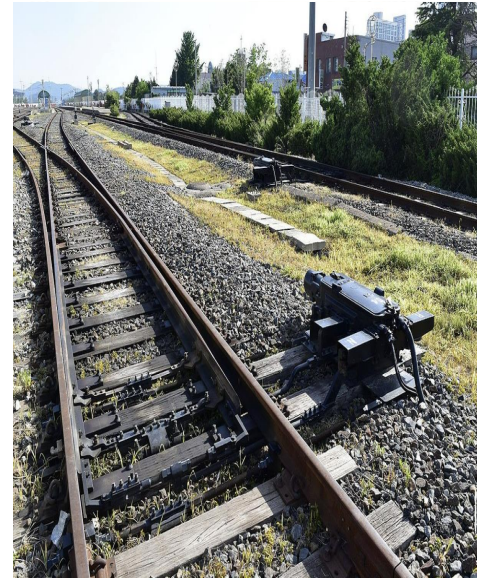


Frontend

- Maintenance dashboard



System Operation Process



Loaded the sensor data from PMD(선로전환기)



```

count    event_num_x  time_stamp_x  event_seq_x  as_volt_x
mean     9.784890e+03  2.023804e+16  1.306289e+02  3.882762e-17
std      8.850614e+03  3.106145e+12  7.996915e+01  1.000000e+00
min      1.000000e+00  2.022118e+16  1.000000e+00  -1.213207e+00
25%     3.255000e+03  2.023832e+16  6.400000e+01  -1.044030e+00
50%     5.885000e+03  2.023853e+16  1.200000e+02  -1.520451e-03
75%     1.414000e+04  2.023871e+16  1.920000e+02  1.036490e+00
max      3.237600e+04  2.024823e+16  6.430000e+02  8.655781e+01

count    output_n_volt_x  output_p_volt_x  ac_volt_x  ac_curr_x
mean     1.261920e-16     1.663509e-16     -6.793869e-14  -2.555805e-15
std      1.000000e+00     1.000000e+00     1.000000e+00     1.000000e+00
min      -3.748750e+00     -3.287985e+00     -4.351476e+00     -1.493761e+00
25%     -6.683620e-02     -1.739619e-01     -5.847566e-01     -1.138861e-01
50%     -4.892199e-02     -1.739619e-01     -3.284598e-02     6.548876e-02
75%     -1.730854e-02     -1.684533e-01     6.213222e-01     2.401433e-01
max      1.178197e+02     4.916917e+02     6.883285e+02     3.373299e+02

count    start_ts  end_ts  ...  event_seq_y  as_volt_y
mean     2.023137e+16  2.023137e+16  ...  NaN          NaN
std      2.556833e+12  2.556833e+12  ...  NaN          NaN
min      2.023012e+16  2.023012e+16  ...  NaN          NaN
25%     2.023081e+16  2.023081e+16  ...  NaN          NaN
50%     2.023081e+16  2.023081e+16  ...  NaN          NaN
75%     2.023081e+16  2.023081e+16  ...  NaN          NaN
max      2.023081e+16  2.023081e+16  ...  NaN          NaN
    
```

Cleaning and Preprocessing Sensor Data



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load data
data = pd.read_csv('data/normalized_data.csv')

# Basic Information
print('Dataset Information:')
data.info()

print('Summary Statistics:')
print(data.describe())

print('Missing values summary:')
print(data.isnull().sum())

# Data for unique values in categorical columns
cat_cols = ['cat_type', 'direction']
print('Unique values in (cat):')
for col in cat_cols:
    print(data[col].unique())

# Distribution of numerical columns
num_cols = ['as_volt_y', 'output_n_volt_x', 'output_p_volt_x', 'ac_volt_x', 'ac_curr_x']
print('Summary Statistics for Numerical Columns:')
    
```

AI model training for error detection and Prediction of PMD



Display on the Maintenance Worker's Screen

HTTP Req
(RESTful API)

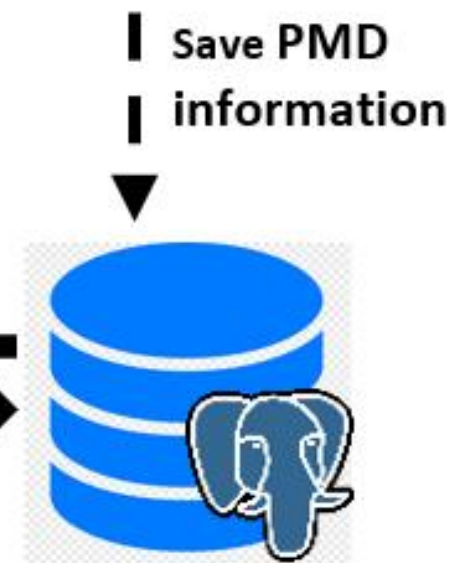
HTTP Res
(Restful API)



Backend Server

HTTP(Fast API, Flask)

HTTP Res
(FastAPI, Flask)



Save PMD information

Point Switch Machine Database Management System

Stored sensor data from the point switch machine

Stored the predicted and detected data from the frontend

Supports real-time tracking and historical data analysis.

The screenshot displays the pgAdmin 4 interface. The left sidebar shows the Object Explorer with the 'sensor_data' table selected. The main window shows a query window with the following SQL query:

```
1 SELECT * FROM public.sensor_data
2 ORDER BY event_num ASC, event_seq ASC
```

The Data Output pane shows a table with 25 rows of sensor data. The columns are: pmd_type, event_num, event_seq, as_volt, output_n_volt, output_r_volt, ac_volt, ac_curr, time_stemp, direction, start_ts, and end_ts. The data is sorted by event_num and event_seq.

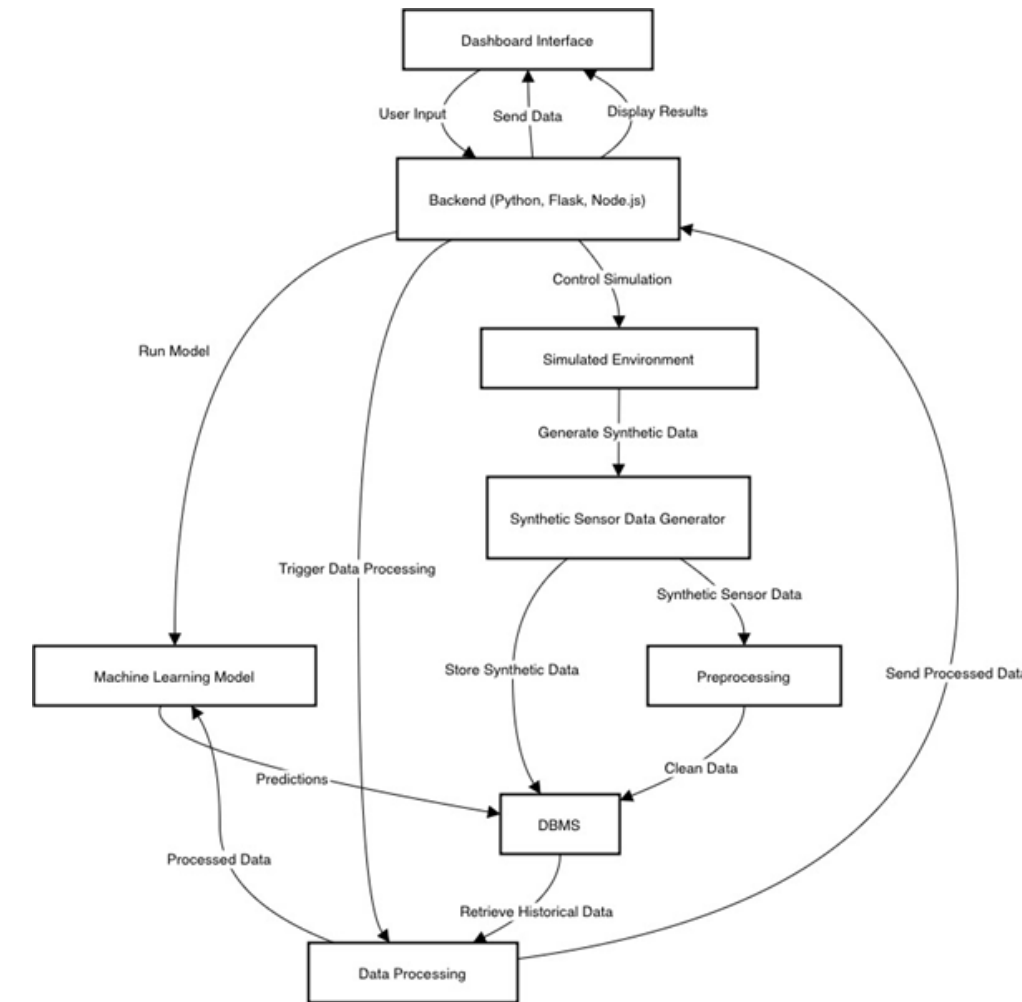
pmd_type	event_num	event_seq	as_volt	output_n_volt	output_r_volt	ac_volt	ac_curr	time_stemp	direction	start_ts	end_ts
PMD001	1	1	0.23	20.4	0.32	224.11	0.06	2.02311e+16	R	[null]	[null]
PMD001	2	2	0.23	20.4	0.35	222.64	0.05	2.02311e+16	R	[null]	[null]
PMD001	3	3	0.2	20.43	0.29	222.64	0.04	2.02311e+16	R	[null]	[null]
PMD001	4	4	0.23	20.4	0.38	223.01	0.04	2.02311e+16	R	[null]	[null]
PMD001	5	5	0.23	20.4	0.29	223.01	0.04	2.02311e+16	R	[null]	[null]
PMD001	6	6	0.23	20.4	0.32	224.11	0.05	2.02311e+16	R	[null]	[null]
PMD001	7	7	0.23	20.4	0.32	222.73	0.05	2.02311e+16	R	[null]	[null]
PMD001	8	8	0.23	20.4	0.29	222.64	0.06	2.02311e+16	R	[null]	[null]
PMD001	9	9	0.23	20.43	0.38	222.83	0.04	2.02311e+16	R	[null]	[null]
PMD001	10	10	6.73	20.43	0.29	222.83	0.05	2.02311e+16	R	[null]	[null]
PMD001	11	11	20	20.37	0.32	222.73	0.04	2.02311e+16	R	[null]	[null]
PMD001	12	12	19.94	20.37	0.29	222.83	0.04	2.02311e+16	R	[null]	[null]
PMD001	13	13	19.94	20.37	0.38	223.01	0.05	2.02311e+16	R	[null]	[null]
PMD001	14	14	19.94	20.34	0.32	223.19	0.05	2.02311e+16	R	[null]	[null]
PMD001	15	15	19.94	20.34	0.29	224.29	0.05	2.02311e+16	R	[null]	[null]
PMD001	16	16	19.94	20.37	0.29	222.83	0.05	2.02311e+16	R	[null]	[null]
PMD001	17	17	15.55	20.37	0.32	222.83	0.05	2.02311e+16	R	[null]	[null]
PMD001	18	18	-7.77	20.37	0.38	223.38	0.06	2.02311e+16	R	[null]	[null]
PMD001	19	19	-20.86	20.37	0.26	222.92	0.07	2.02311e+16	R	[null]	[null]
PMD001	20	20	-20.43	20.37	0.35	224.02	0.05	2.02311e+16	R	[null]	[null]
PMD001	21	21	-20.25	20.37	0.29	224.2	0.04	2.02311e+16	R	[null]	[null]
PMD001	22	22	-20.31	20.4	0.32	222.73	0.05	2.02311e+16	R	[null]	[null]
PMD001	23	23	-21.01	1.88	0.32	220.26	0.38	2.02311e+16	R	[null]	[null]
PMD001	24	24	-20.86	0.2	0.29	207.9	10.68	2.02311e+16	R	[null]	[null]
PMD001	25	25	-20.09	0.2	0.38	207.08	10.11	2.02311e+16	R	[null]	[null]

Total rows: 1000 of 497649 Query complete 00:00:01.008 Ln 1, Col 1

AI

- Process the sensor data
- Detect anomaly/abnormal data
- Predict the anomaly/abnormal data
- Capable of early fault detection.
- Store results in backend server.

High Level Design

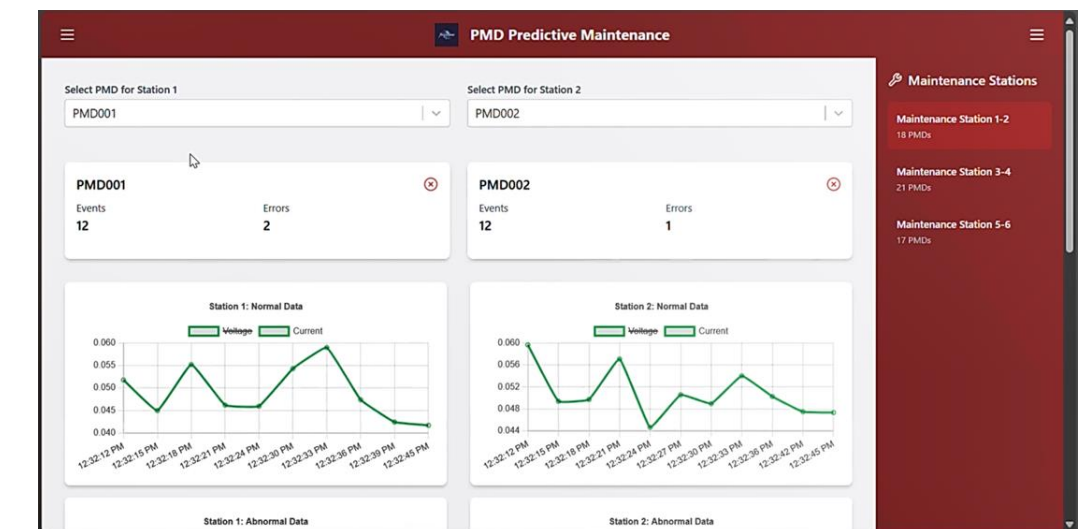


```

count    event_num_x  time_stemp_x  event_seq_x  as_volt_x  \
mean     9.704890e+03  2.023004e+16  1.300289e+02  3.082762e-17
std      8.850614e+03  3.106145e+12  7.996915e+01  1.000000e+00
min      1.000000e+00  2.022110e+16  1.000000e+00  -1.213207e+00
25%     3.255000e+03  2.023032e+16  6.400000e+01  -1.044030e+00
50%     5.885000e+03  2.023053e+16  1.280000e+02  -1.520451e-03
75%     1.414800e+04  2.023071e+16  1.920000e+02  1.036490e+00
max      3.237600e+04  2.024023e+16  6.430000e+02  8.655781e+01

count    output_n_volt_x  output_r_volt_x  ac_volt_x  ac_curr_x  \
mean     1.261920e-16  1.663569e-16  -6.793069e-14  -2.555865e-15
std      1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min      -3.748750e+00  -3.207985e+00  -4.351476e+00  -1.493761e+00
25%     -6.683628e-02  -1.739619e-01  -5.847566e-01  -1.138061e-01
50%     -4.892199e-02  -1.739619e-01  -3.284598e-02  6.540876e-02
75%     -1.730854e-02  -1.604533e-01  6.213222e-01  2.401433e-01
max      1.178197e+02  4.916917e+02  6.883285e+02  3.373299e+02

count    start_ts  end_ts  ...  event_seq_y  as_volt_y  \
mean     2.023137e+16  2.023137e+16  ...  NaN         NaN
std      2.556833e+12  2.556833e+12  ...  NaN         NaN
min      2.023012e+16  2.023012e+16  ...  NaN         NaN
25%     2.023081e+16  2.023081e+16  ...  NaN         NaN
50%     2.023081e+16  2.023081e+16  ...  NaN         NaN
75%     2.023081e+16  2.023081e+16  ...  NaN         NaN
  
```



```

# Step 3: Build LSTM Autoencoder
def build_lstm_autoencoder(input_dim):
    model = Sequential([
        LSTM(128, activation='relu', input_shape=(input_dim, 1), return_sequences=True),
        Dropout(0.3),
        LSTM(64, activation='relu', return_sequences=False),
        Dense(64, activation='relu'),
        Dense(input_dim, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

```

Random Forest Classifier

```

rf_model = RandomForestClassifier(random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict_proba(X_test)[:, 1]

```

Deep Neural Network (DNN)

```

def build_dnn(input_dim):
    model = Sequential([
        Dense(256, activation='relu', input_dim=input_dim),
        Dropout(0.4),
        Dense(128, activation='relu'),
        Dropout(0.4),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

```

Metadata not available for detected anomalies.

Model Performance Metrics:

Accuracy: 0.96290

Precision: 0.98178

Recall: 0.94328

F1 Score: 0.96214

Classification Report:

	precision	recall	f1-score	support		
0	0.94545	0.98251	0.96362	1429		
1	0.98178	0.94328	0.96214	1428		
accuracy			0.96290	2857		
macro avg			0.96362	0.96289	0.96288	2857
weighted avg			0.96361	0.96290	0.96288	2857

Abnormal Data Detected:

Number of Abnormal Data: 1372

Abnormal Event Numbers: [4, 8, 11, 13, 15, 16, 17, 19,

```
# Step 2: Function to prepare sequence data
def create_sequences(event_nums, window_size=3):
```

```
# Step 3: Prepare the data
window_size = 3 # Predict the next event based on the past 3 events
X, y = create_sequences(abnormal_event_numbers, window_size)
X = X.reshape((X.shape[0], X.shape[1], 1)) # Reshape to match LSTM input format
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step 4: Build the LSTM model
```

```
def build_lstm_model(input_shape):
    model = Sequential([
        LSTM(64, activation='relu', input_shape=input_shape),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1) # next event_num
    ])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model
```

```
# Create and train the model
```

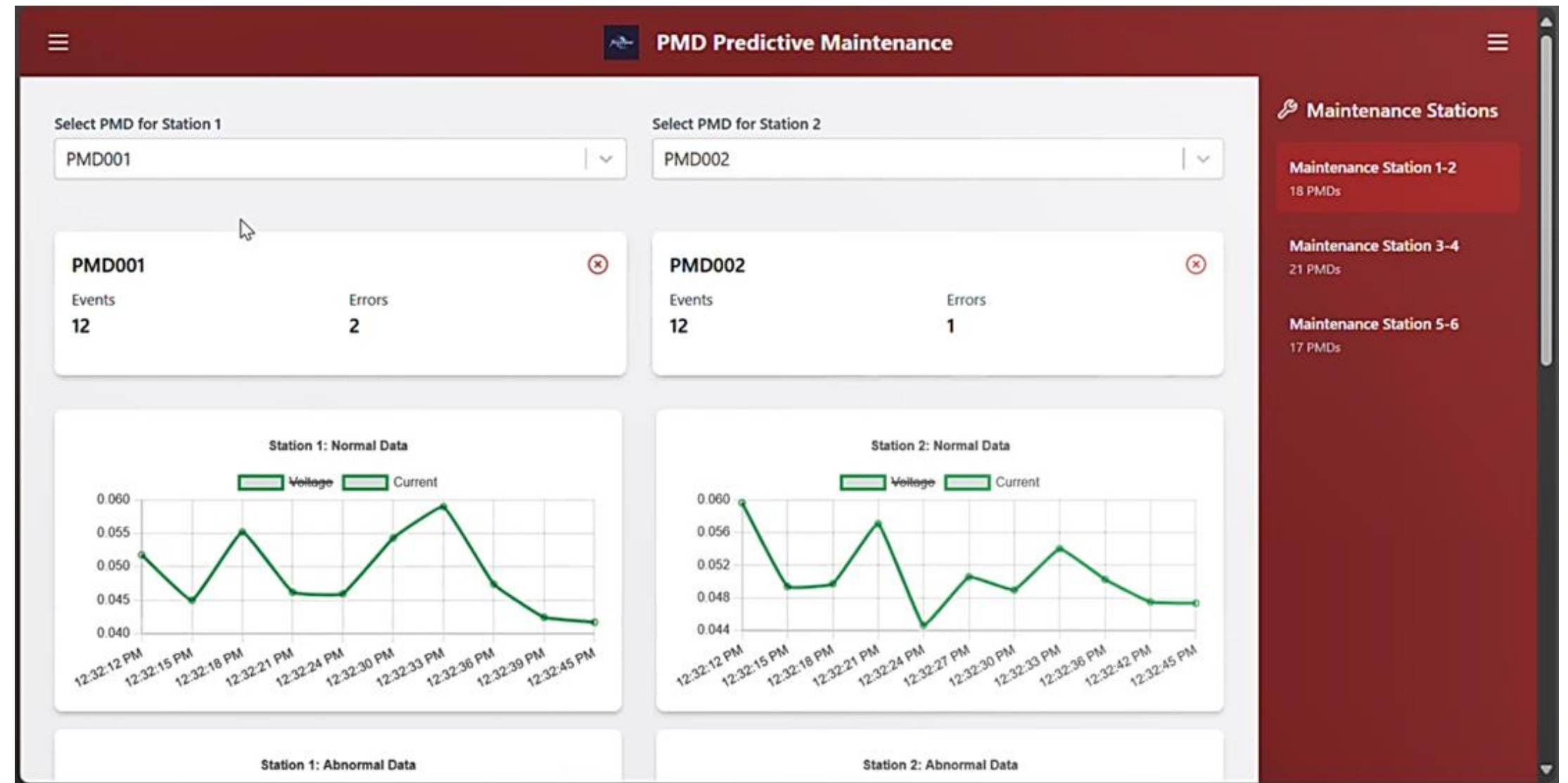
```
model = build_lstm_model((X.shape[1], 1))
print("start the learning model...")
history = model.fit(X_train, y_train, epochs=50, batch_size=128,
```

Top Predicted Error Events:

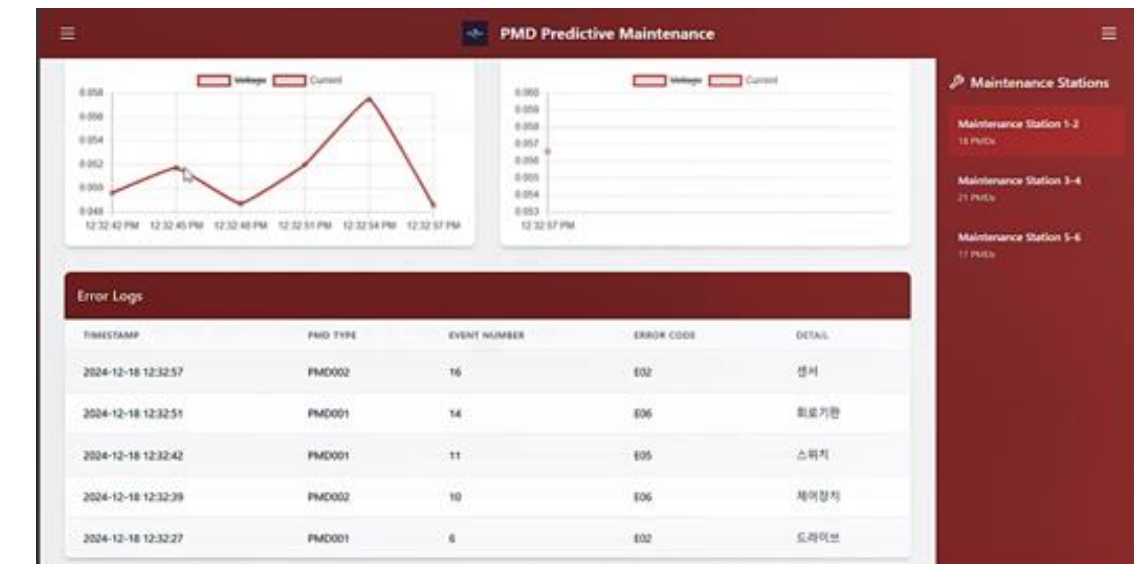
	event_num_x	error_prob
2281	2286	0.870000
1768	1773	0.870000
5105	5110	0.817952
2410	2415	0.802500
5397	5402	0.759667

Frontend

- User Interface
- Real-time monitoring
- Display detected errors.
- Alert maintenance worker about the error
- Visualizes predicted failure data and historical trends.

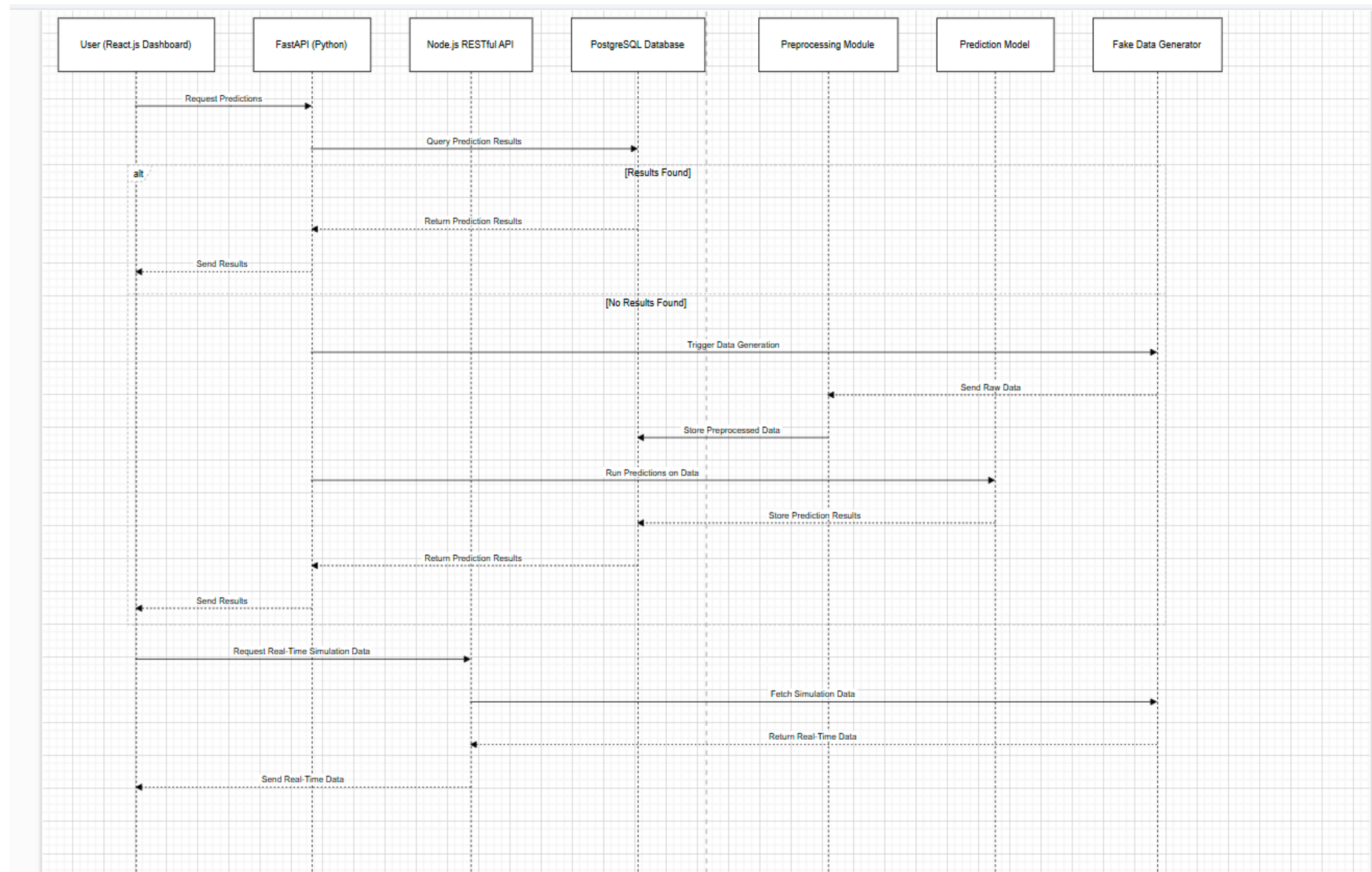


Error Logs				
TIMESTAMP	PMD TYPE	EVENT NUMBER	ERROR CODE	DETAIL
2024-12-18 12:32:57	PMD002	16	E02	센서
2024-12-18 12:32:51	PMD001	14	E06	회로기판
2024-12-18 12:32:42	PMD001	11	E05	스위치
2024-12-18 12:32:39	PMD002	10	E06	제어장치
2024-12-18 12:32:27	PMD001	6	E02	드라이브



Backend

- Receive request
- Handle the request
 - Data storage
 - Data retrieval
 - Other
- Storing AI prediction data
- Processing data
- Listen to AI request and update collection data
- Providing data to Frontend



Backend Code Files

The screenshot displays the Visual Studio Code interface for a project named 'FaultForce_Failure_prediction_dashboard'. The Explorer view on the left shows the project structure, with the 'backend' folder expanded to show 'app.py' selected. The main editor shows the code for 'app.py', which includes imports for Flask, SQLAlchemy, and TensorFlow, and defines a Flask application with a database connection and a sensor data model.

```
1 import express
2 import cors fro
3 import pg from
4
5 const { Pool }
6
7 // Initialize e
8 const app = exp
9 const PORT = 50
10
11 // Middleware
12 app.use(cors())
13 app.use(express
14
15 // PostgreSQL c
16 const pool = ne
17   user: "postgr
18   host: "localh
19   database: "Po
20   password: "00
21   port: 5432,
22 });
23
24 // Test Databas
25 pool.connect()
26   .then(() => c
27   .catch((err)
28
29 // Routes
30 app.get("/", (r
31   res.send("Hel
32 });
33
34 // Start Server
35 app.listen(PORT
36   console.log(`
37 });
38
```

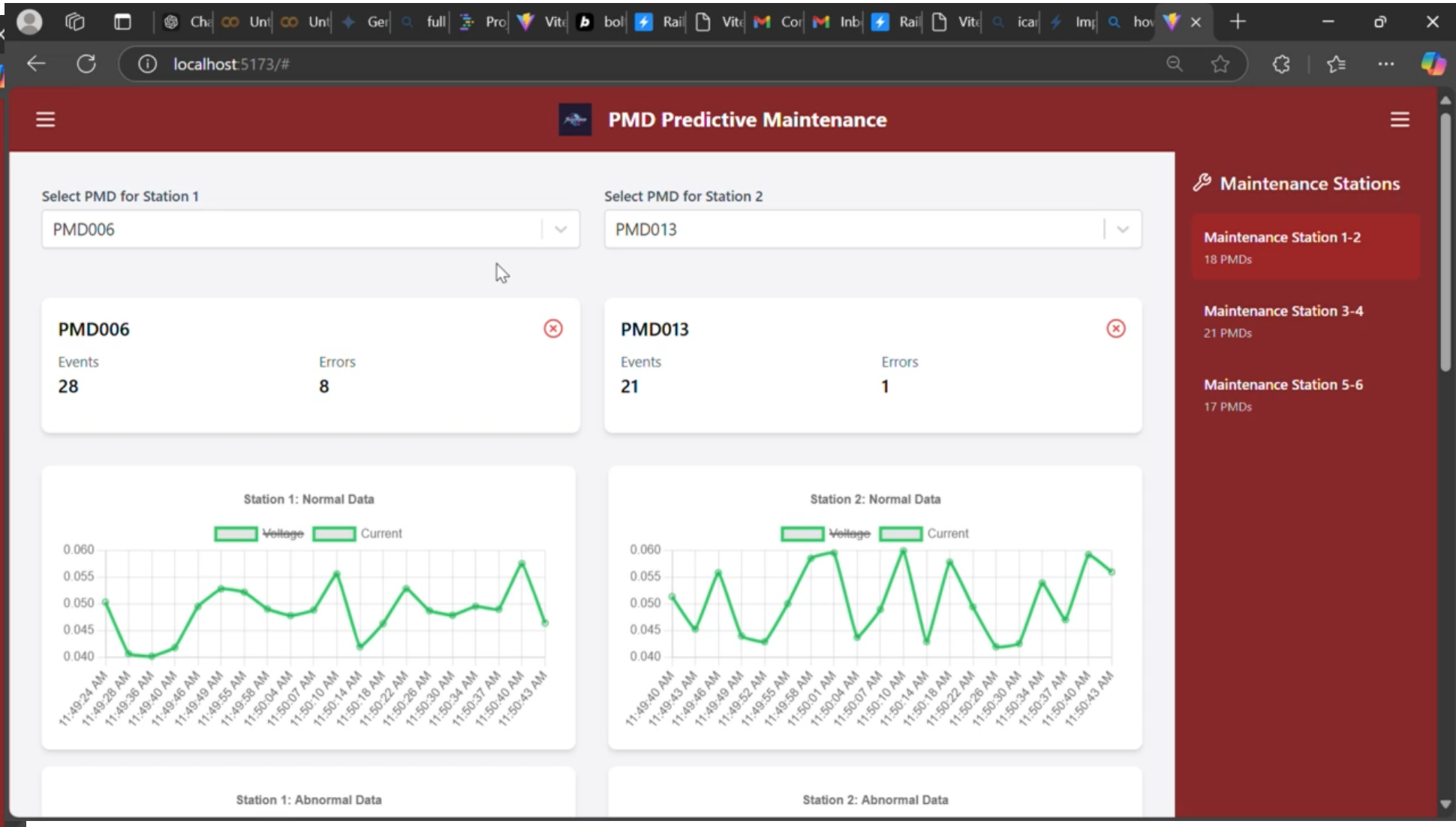
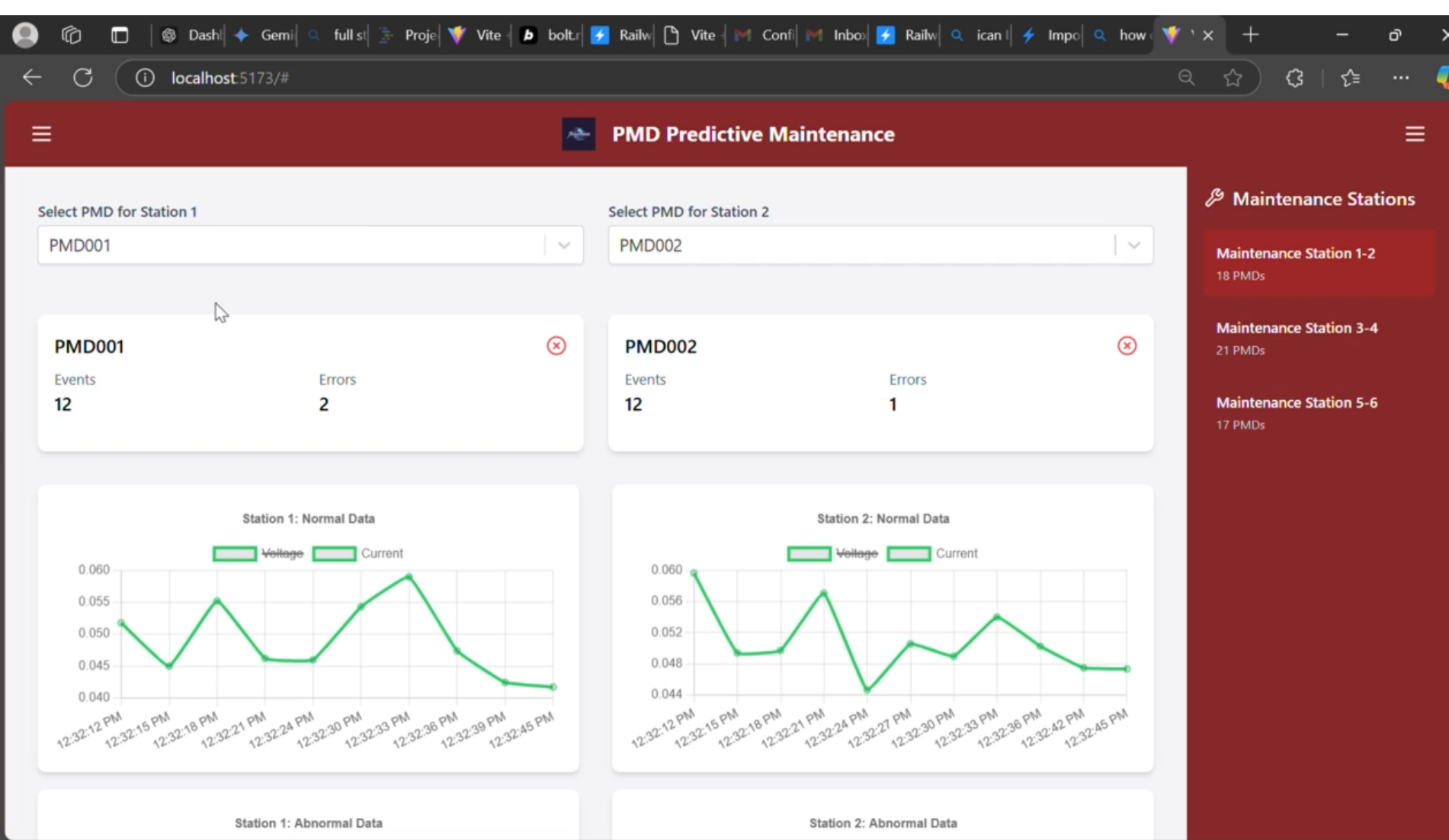
```
1 from flask import Flask, request, jsonify
2 from flask_sqlalchemy import SQLAlchemy
3 from sqlalchemy import create_engine, Column, Integer, String, Double, PrimaryKeyConstraint
4 from sqlalchemy.ext.declarative import declarative_base
5 from sqlalchemy.orm import sessionmaker
6 import joblib
7 import tensorflow as tf
8 import os
9 import numpy as np
10 from flask_cors import CORS
11
12 # Flask app and database setup
13 app = Flask(__name__)
14 CORS(app, supports_credentials=True)
15 app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:0000@localhost:5432/Point Switch Machine Databa
16 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
17
18 # Initialize Database
19 engine = create_engine(app.config['SQLALCHEMY_DATABASE_URI'])
20 Session = sessionmaker(bind=engine)
21 session = Session()
22
23 # Load Models
24 lstm_model_path = os.path.join('models', 'lstm_event_num_prediction_model1.h5')
25 lstm_model = tf.keras.models.load_model(lstm_model_path, compile=False)
26
27 rf_model_path = os.path.join('models', 'random_forest_model1.pkl')
28 rf_model = joblib.load(rf_model_path)
29
30 # SQLAlchemy Base
31 Base = declarative_base()
32
33 # SensorData Model with Composite Primary Key
34 class SensorData(Base):
35     __tablename__ = 'sensor_data'
36
37     pmd_type = Column(String(255), nullable=False)
38     event_num = Column(Integer, nullable=False)
39     event_seq = Column(Integer)
40     as_volt = Column(Double)
41     output_n_volt = Column(Double)
42     output_r_volt = Column(Double)
43     ac_volt = Column(Double)
44     ac_curr = Column(Double)
45     time_stemp = Column(Double)
46     direction = Column(String(255))
47     start_ts = Column(Double)
48     end_ts = Column(Double)
```

Activate Windows
Go to Settings to activate Windows.

RESULTS

- Successfully trained and validated AI models.
- Backend systems operational.
- Fake data simulation UI developed for real-time visualization.
- Early fault notifications reduce maintenance downtime and accident risks.

DEMO



NEXT SOLUTIONS

1

Real Time Monitoring:

- Integration with real-time sensor data for live updates.

2

Risk Level Development:

- Provides more granular notification to maintenance personnel by separating risk levels

3

Complete Database Integration:

- Integration with the bigger PMD sensor database

4

UI Enhancement:

- Add more functions and individual power graphs for Voltage and Current in the Prediction Dashboard

THANK YOU