In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
```

In [2]:
```
import os
import pandas as pd

# Specify folder path with spaces (either use quotes or raw string)
folder_path = r'D:\Training Dataset\Training Dataset\February 2024'
csv_files = [file for file in os.listdir(folder_path) if file.endswith('.csv')

# Combine CSV files into a single DataFrame
combined_data = pd.concat([pd.read_csv(os.path.join(folder_path, file)) for fi

# Display the combined DataFrame
print(combined_data.head())
```

```
                   ts  humidity1  humidity2  humidity3  humidity4  humidity5
\
0  2024-02-01 00:02:02      54.04      53.58      49.96      50.66      48.77
1  2024-02-01 00:04:07      52.74      52.43      49.59      50.33      48.95
2  2024-02-01 00:06:12      50.39      50.01      49.78      51.23      48.18
3  2024-02-01 00:08:17      48.73      48.31      49.61      51.67      47.29
4  2024-02-01 00:10:22      48.83      48.38      49.68      52.18      46.62

   humidity6  humidity7  humidity8  temperature1  temperature2  temperature3
\
0      49.92      42.50      48.98         15.57         15.65         17.12
1      49.52      42.65      48.73         16.16         16.23         17.17
2      48.07      42.66      48.41         16.81         16.92         17.01
3      46.64      42.19      47.82         17.17         17.27         16.95
4      46.23      41.71      47.52         16.92         17.02         16.77

   temperature4  temperature5  temperature6  temperature7  temperature8
0         16.79         16.80         16.66         19.10         18.66
1         16.73         16.88         16.95         19.05         18.78
2         16.29         17.05         17.34         18.90         18.80
3         16.04         17.25         17.68         19.02         18.93
4         15.73         17.35         17.65         19.10         18.90
```

In [3]:
```
len(folder_path)
```

Out[3]: 50

In [4]:
```
combined_data
```

Out[4]:

	ts	humidity1	humidity2	humidity3	humidity4	humidity5	humidity6	humidity7	h
0	2024-02-01 00:02:02	54.04	53.58	49.96	50.66	48.77	49.92	42.50	
1	2024-02-01 00:04:07	52.74	52.43	49.59	50.33	48.95	49.52	42.65	
2	2024-02-01 00:06:12	50.39	50.01	49.78	51.23	48.18	48.07	42.66	
3	2024-02-01 00:08:17	48.73	48.31	49.61	51.67	47.29	46.64	42.19	
4	2024-02-01 00:10:22	48.83	48.38	49.68	52.18	46.62	46.23	41.71	
...	...	...	...	...	...	...	...	...	
19593	2024-02-29 23:53:17	47.15	47.82	47.46	46.09	37.32	45.81	51.04	
19594	2024-02-29 23:55:22	46.56	47.16	47.18	45.75	36.98	45.32	51.20	
19595	2024-02-29 23:57:27	46.13	46.62	47.03	45.56	36.68	44.90	51.38	
19596	2024-02-29 23:59:32	46.07	46.44	48.51	46.42	36.62	44.91	53.53	
19597	2024-03-01 00:00:47	47.03	47.27	50.57	48.20	37.10	45.93	55.57	

19598 rows × 17 columns

In [5]:
```
combined_data.describe()
```

Out[5]:

	humidity1	humidity2	humidity3	humidity4	humidity5	humidity6	19
count	19598.000000	19598.000000	19598.000000	19598.000000	19598.000000	19598.000000	
mean	46.523609	46.150217	47.827360	50.271272	41.010618	39.214451	
std	5.064873	5.103563	3.890336	4.668937	5.493304	8.086799	
min	31.020000	32.620000	34.790000	35.450000	27.010000	22.010000	
25%	42.820000	42.282500	44.920000	46.520000	36.730000	31.720000	
50%	46.120000	45.830000	47.580000	49.810000	40.390000	39.840000	
75%	49.600000	49.450000	50.430000	53.700000	44.477500	45.600000	
max	69.740000	69.200000	65.070000	67.980000	63.550000	64.390000	

In [6]:
```
combined_data.isnull()
```

Out[6]:

	ts	humidity1	humidity2	humidity3	humidity4	humidity5	humidity6	humidity7	hum
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
19593	False	False	False	False	False	False	False	False	
19594	False	False	False	False	False	False	False	False	
19595	False	False	False	False	False	False	False	False	
19596	False	False	False	False	False	False	False	False	
19597	False	False	False	False	False	False	False	False	

19598 rows × 17 columns

In [7]:
```
combined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19598 entries, 0 to 19597
Data columns (total 17 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   ts            19598 non-null  object 
 1   humidity1     19598 non-null  float64
 2   humidity2     19598 non-null  float64
 3   humidity3     19598 non-null  float64
 4   humidity4     19598 non-null  float64
 5   humidity5     19598 non-null  float64
 6   humidity6     19598 non-null  float64
 7   humidity7     19598 non-null  float64
 8   humidity8     19598 non-null  float64
 9   temperature1  19598 non-null  float64
 10  temperature2  19598 non-null  float64
 11  temperature3  19598 non-null  float64
 12  temperature4  19598 non-null  float64
 13  temperature5  19598 non-null  float64
 14  temperature6  19598 non-null  float64
 15  temperature7  19598 non-null  float64
 16  temperature8  19598 non-null  float64
dtypes: float64(16), object(1)
memory usage: 2.5+ MB
```

In [8]:
```
combined_data.isna()
```

Out[8]:

	ts	humidity1	humidity2	humidity3	humidity4	humidity5	humidity6	humidity7	hum
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
19593	False	False	False	False	False	False	False	False	
19594	False	False	False	False	False	False	False	False	
19595	False	False	False	False	False	False	False	False	
19596	False	False	False	False	False	False	False	False	
19597	False	False	False	False	False	False	False	False	

19598 rows × 17 columns

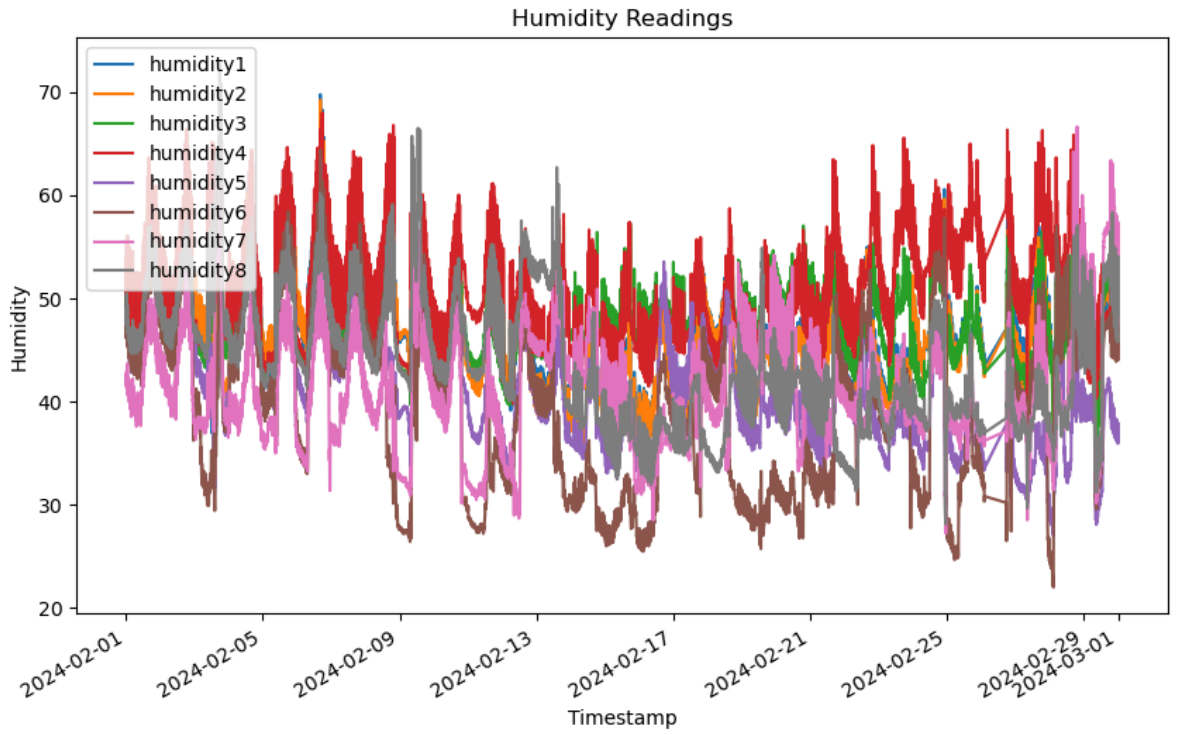
In [9]:
```
missing_values = combined_data.isnull()

# Count the number of missing values in each column
missing_counts = missing_values.sum()

# Print the number of missing values in each column
print("Missing values in each column:")
print(missing_counts)
```

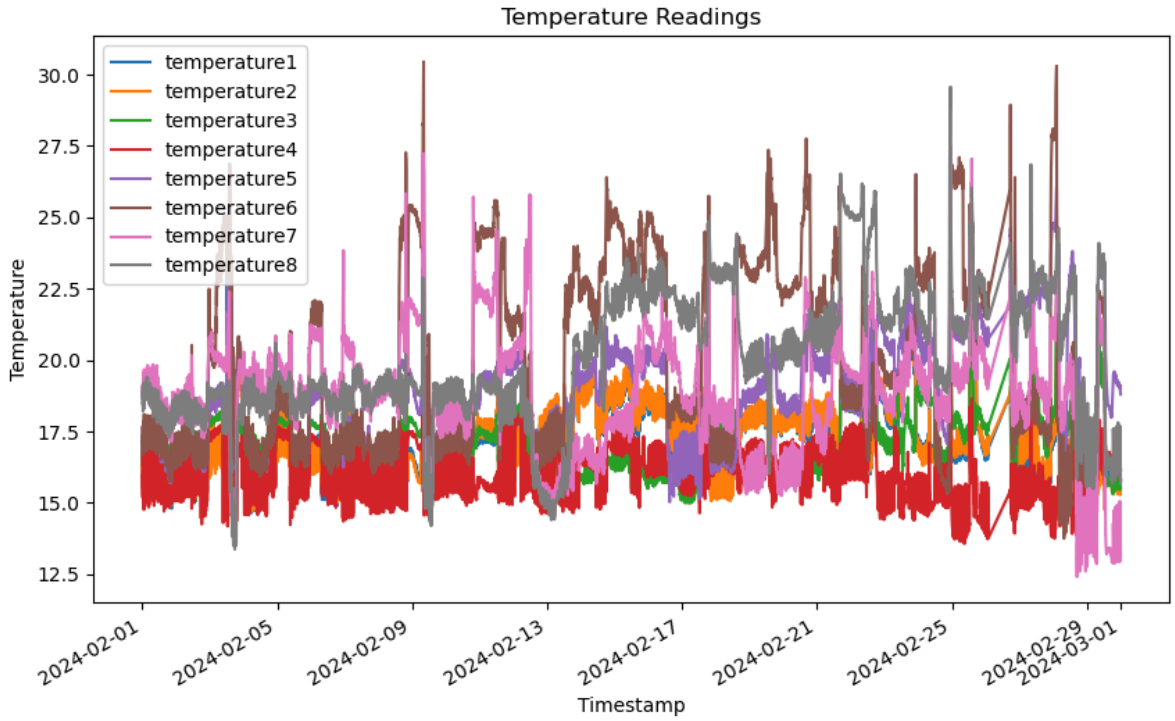
```
Missing values in each column:
ts              0
humidity1       0
humidity2       0
humidity3       0
humidity4       0
humidity5       0
humidity6       0
humidity7       0
humidity8       0
temperature1    0
temperature2    0
temperature3    0
temperature4    0
temperature5    0
temperature6    0
temperature7    0
temperature8    0
dtype: int64
```

In [10]:
```
dates = pd.date_range(start='2024-02-01', end='2024-02-29', freq='15T')

# Convert the 'Date' column to datetime
combined_data['ts'] = pd.to_datetime(combined_data['ts'])

# Set 'Date' column as the index
combined_data.set_index('ts', inplace=True)

# Resample the data to hourly intervals and calculate the mean
hourly_average = combined_data.resample('H').mean()

# Display the hourly averages
print(hourly_average.head())
```

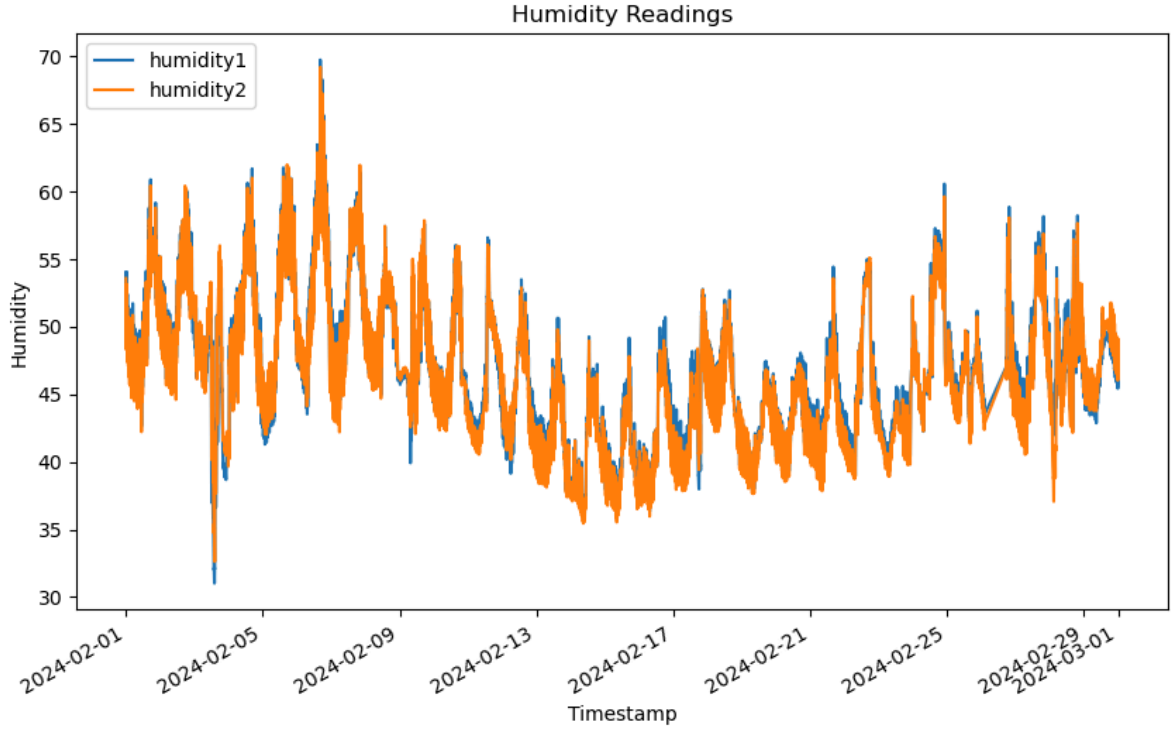
```
                     humidity1  humidity2  humidity3  humidity4  humidity5  \
ts
2024-02-01 00:00:00  50.822143  50.118571  50.302500  52.369643  46.929286
2024-02-01 01:00:00  49.774828  48.998621  49.567586  51.576207  45.660000
2024-02-01 02:00:00  49.128621  48.376897  48.728276  50.805517  44.840000
2024-02-01 03:00:00  48.492759  47.743103  47.750690  49.729310  44.209655
2024-02-01 04:00:00  48.346897  47.555172  48.798966  50.993793  44.341379

                     humidity6  humidity7  humidity8  temperature1  \
ts
2024-02-01 00:00:00  47.326429  42.003571  48.216429     16.282143
2024-02-01 01:00:00  46.199310  41.366897  47.053103     16.314828
2024-02-01 02:00:00  45.541724  40.655172  46.251724     16.301379
2024-02-01 03:00:00  44.888966  39.915172  45.553448     16.308276
2024-02-01 04:00:00  44.975862  40.617586  46.179655     16.473103

                     temperature2  temperature3  temperature4  temperature5
\
ts
2024-02-01 00:00:00     16.453929     16.688929     15.860357     17.235714
2024-02-01 01:00:00     16.516552     16.672069     15.861034     17.377241
2024-02-01 02:00:00     16.490345     16.731724     15.888276     17.438621
2024-02-01 03:00:00     16.503793     16.869310     16.033448     17.463103
2024-02-01 04:00:00     16.677931     16.598621     15.736207     17.504483

                     temperature6  temperature7  temperature8
ts
2024-02-01 00:00:00     17.272500     19.038214     18.692143
2024-02-01 01:00:00     17.350690     19.006207     18.840345
2024-02-01 02:00:00     17.352414     19.060000     18.907931
2024-02-01 03:00:00     17.375172     19.168621     18.976552
2024-02-01 04:00:00     17.443448     18.968621     18.823448
```

In [11]:
```
import matplotlib.pyplot as plt
# Assuming 'combined_data' is your DataFrame containing humidity and temperatu
# Replace 'combined_data' with the actual name of your DataFrame

# Plot humidity data
combined_data[['humidity1','humidity2','humidity3','humidity4','humidity5','hu
plt.title('Humidity Readings')
plt.xlabel('Timestamp')
plt.ylabel('Humidity')
plt.legend(loc='upper left')
plt.show()

# Plot temperature data
combined_data[['temperature1','temperature2','temperature3','temperature4','te
plt.title('Temperature Readings')
plt.xlabel('Timestamp')
plt.ylabel('Temperature')
plt.legend(loc='upper left')
plt.show()
```

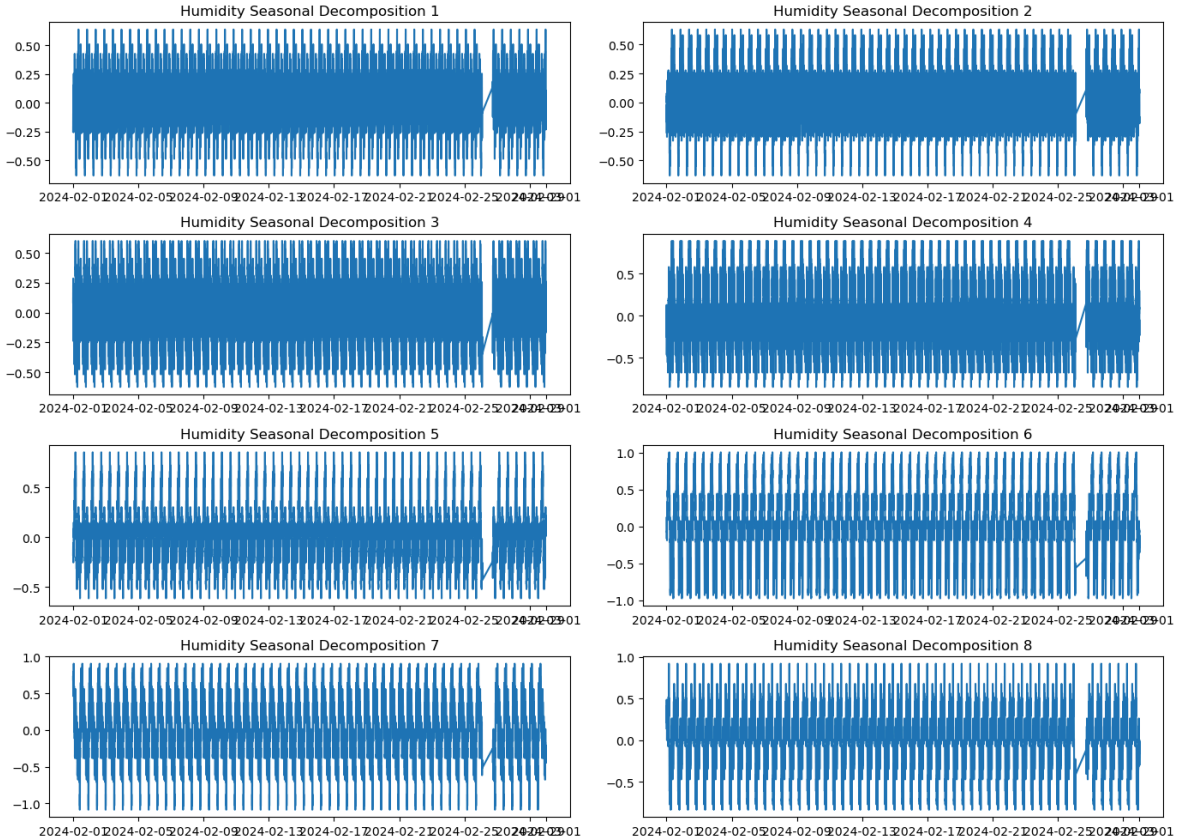


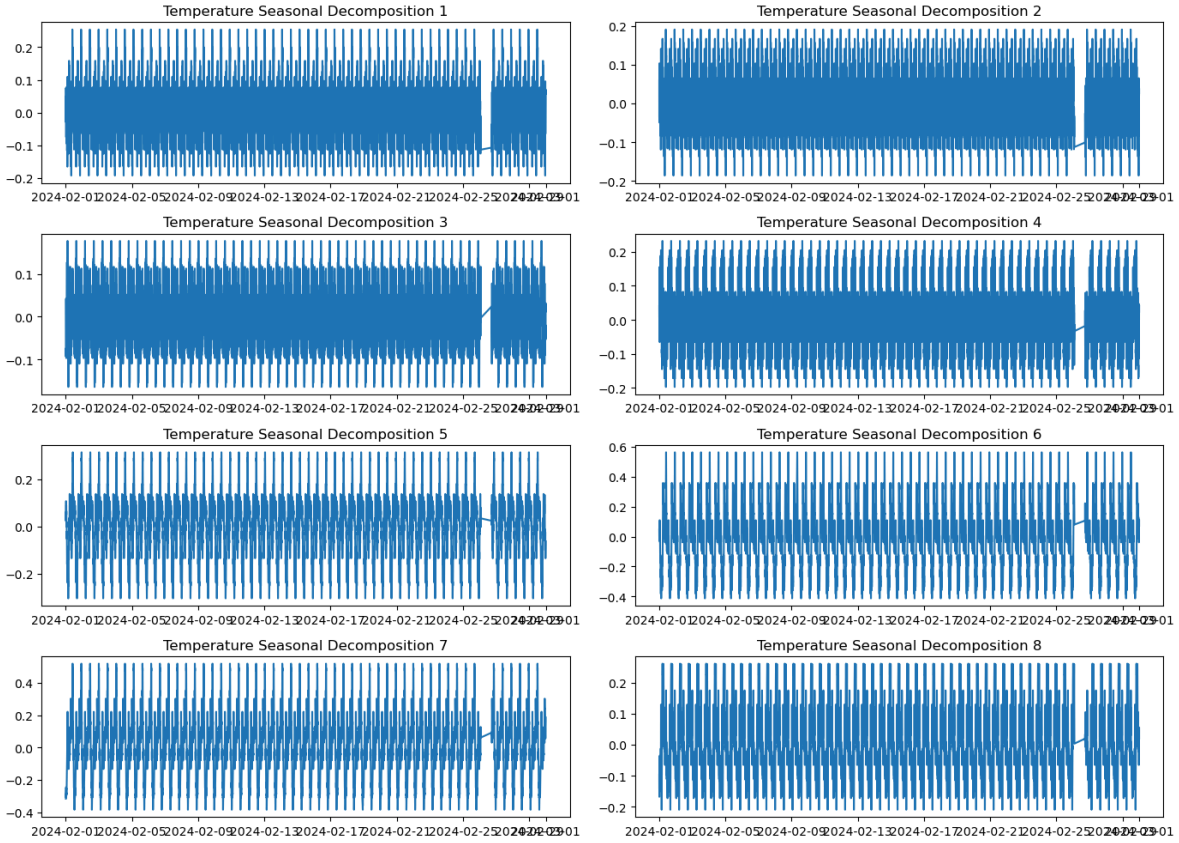


In [12]:
```
combined_data[['humidity1','humidity2']].plot(figsize=(10, 6))
plt.title('Humidity Readings')
plt.xlabel('Timestamp')
plt.ylabel('Humidity')
plt.legend(loc='upper left')
plt.show()
```



In [13]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Assuming 'combined_data' is your DataFrame containing humidity and temperatu
# Replace 'combined_data' with the actual name of your DataFrame

# Plot seasonal decomposition for humidity data
fig, axs = plt.subplots(4, 2, figsize=(14, 10))
for i in range(8):
    series = combined_data[f'humidity{i+1}']
    decomposition = seasonal_decompose(series, model='additive', period=365)
    axs[i//2, i%2].plot(decomposition.seasonal)
    axs[i//2, i%2].set_title(f'Humidity Seasonal Decomposition {i+1}')
plt.tight_layout()
plt.show()

# Plot seasonal decomposition for temperature data
fig, axs = plt.subplots(4, 2, figsize=(14, 10))
for i in range(8):
    series = combined_data[f'temperature{i+1}']
    decomposition = seasonal_decompose(series, model='additive', period=365)
    axs[i//2, i%2].plot(decomposition.seasonal)
    axs[i//2, i%2].set_title(f'Temperature Seasonal Decomposition {i+1}')
plt.tight_layout()
plt.show()
```



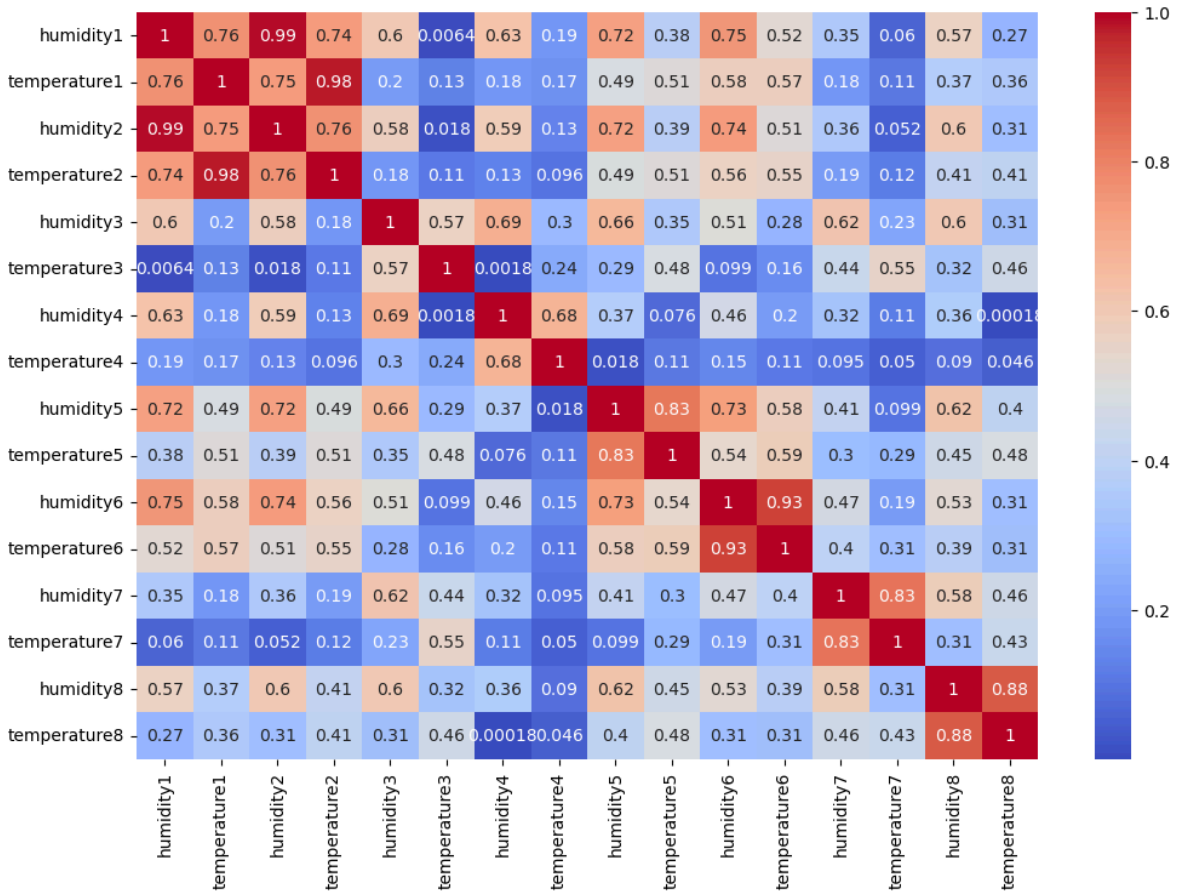

Temperature Seasonal Decomposition 1
Temperature Seasonal Decomposition 2
Temperature Seasonal Decomposition 3
Temperature Seasonal Decomposition 4
Temperature Seasonal Decomposition 5
Temperature Seasonal Decomposition 6
Temperature Seasonal Decomposition 7
Temperature Seasonal Decomposition 8

In [14]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler

# Assuming 'combined_data' is your DataFrame containing humidity and temperatu
# Replace 'combined_data' with the actual name of your DataFrame

# Concatenate humidity and temperature data into one DataFrame
data = combined_data[['humidity1', 'temperature1', 'humidity2', 'temperature2'

# Normalize the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Define the sequence length
seq_length = 30  # Example: using the past 30 days' data to predict the next d

# Function to create sequences for LSTM
def create_sequences(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

# Create sequences for LSTM
X, y = create_sequences(scaled_data, seq_length)

# Split data into training and testing sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Define the LSTM model
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(units=50, return_sequences=True, input_shape=(X_train
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(units=50, return_sequences=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(units=16),
    tf.keras.layers.Dense(units=len(data.columns))  # Output layer: same numbe
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_spl

# Evaluate the model
mse = model.evaluate(X_test, y_test)

# Plot loss during training
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
```

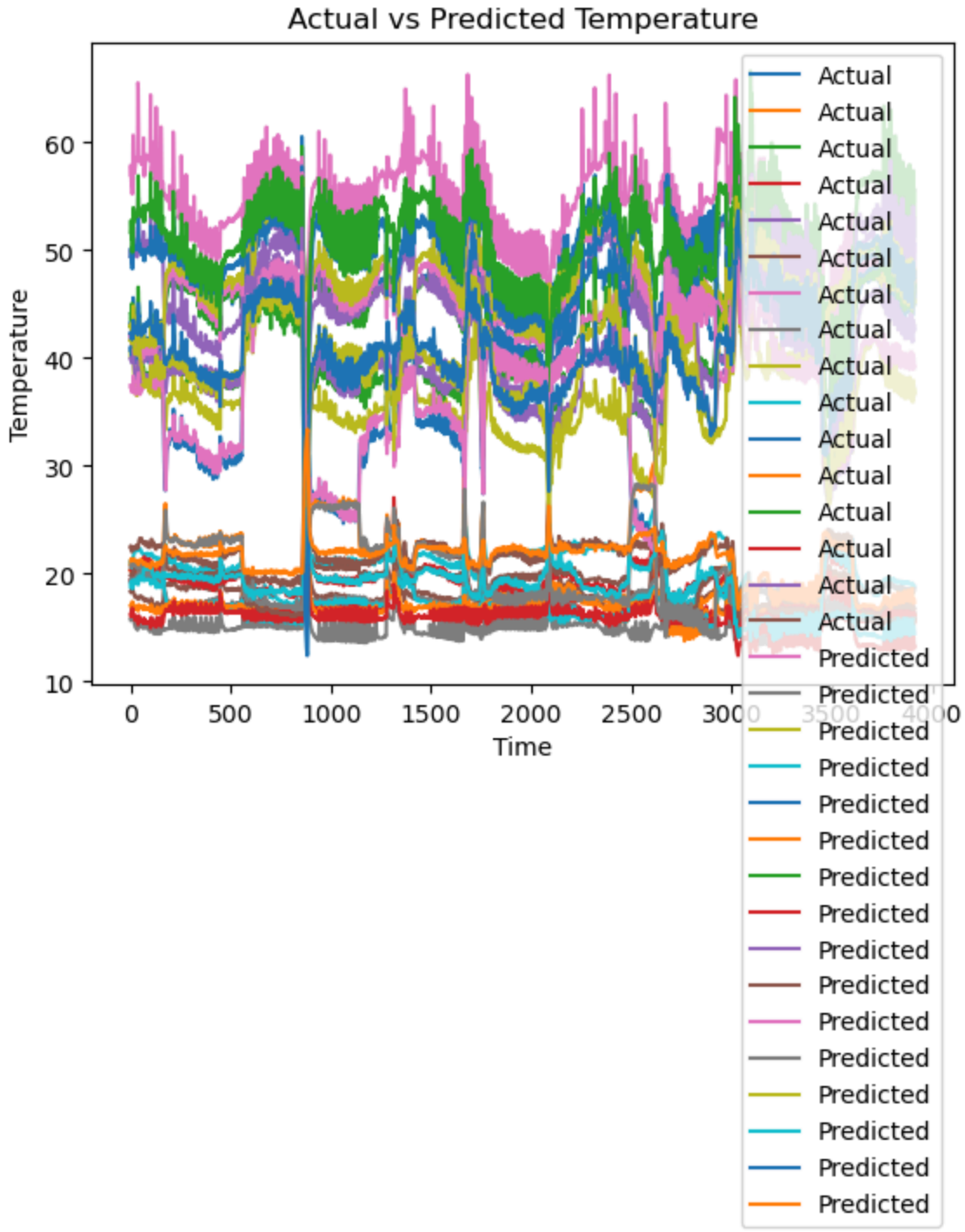
```
plt.legend()
plt.show()

# Make predictions
predictions = model.predict(X_test)

# Inverse scaling
predictions = scaler.inverse_transform(predictions)
y_test = scaler.inverse_transform(y_test)

# Evaluate predictions (e.g., using MAE, MSE, etc.)
# Example:
mae = np.mean(np.abs(predictions - y_test))
print("Mean Absolute Error:", mae)
```

```
Epoch 1/50

C:\Users\lenovo\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

392/392 ━━━━━━━━━━━━━━━━━━━━ 13s 27ms/step - loss: 0.0328 - val_loss: 0.0123
Epoch 2/50
392/392 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0089 - val_loss: 0.0107
Epoch 3/50
392/392 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0072 - val_loss: 0.0097
Epoch 4/50
392/392 ━━━━━━━━━━━━━━━━━━━━ 10s 27ms/step - loss: 0.0061 - val_loss: 0.0079
Epoch 5/50
```

In [15]:
```
plt.figure(figsize=(12,8))
sns.heatmap(data.corr().abs(), annot= True, cmap= 'coolwarm');
```



In [21]:
```
import matplotlib.pyplot as plt

# Visualize predictions vs actual values
plt.plot(y_test, label='Actual')
plt.plot(predictions, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('Actual vs Predicted Temperature')
plt.legend()
plt.show()

# Compute additional metrics
mse = np.mean((predictions - y_test)**2)
rmse = np.sqrt(mse)
r_squared = 1 - (np.sum((y_test - predictions)**2) / np.sum((y_test - np.mean(

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r_squared)
```




```
Mean Squared Error (MSE): 5.087653444278556
Root Mean Squared Error (RMSE): 2.255582728316245
R-squared: 0.9742197419799831
```