In [2]:
```python
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
```

In [4]:
```python
# Downloading Ethereum Data From Yahoo Finance
eth_data = yf.download('ETH-USD', start='2018-01-01', end='2021-01-01')

# Processing the data
data = eth_data['Close'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data)

# Split data into training and testing sets
train_size = int(len(scaled_data) * 0.8)
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]
```

```
[*********************100%%**********************]  1 of 1 completed
```

In [6]:
```python
# Create sequences for time series prediction

def create_sequences(data, sequence_length):
  X, y = [], []
  for i in range(len(data) - sequence_length - 1):
    X.append(data[i:(i + sequence_length), 0])
    y.append(data[i + sequence_length, 0])
  return np.array(X), np.array(y)
```

In [9]:
```python
sequence_length = 10 # adjust the sequence length as needed
X_train, y_train = create_sequences(train_data, sequence_length)
X_test, y_test = create_sequences(test_data, sequence_length)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [11]:
```python
# Build and train the LSTM model

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32)

# Make the Prediction
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

# Inverse transform predictions to original scale
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

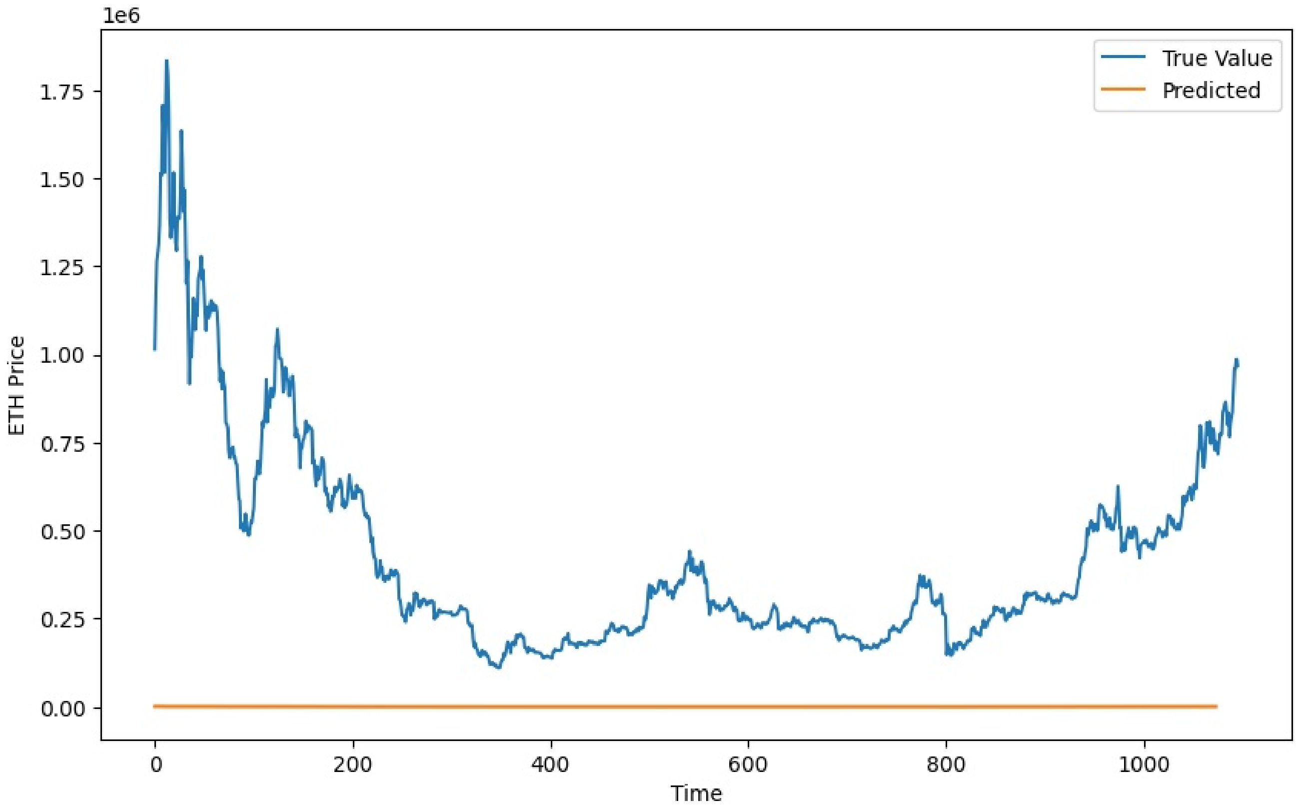
```
Epoch 1/100
28/28 [==============================] - 9s 6ms/step - loss: 0.0123
Epoch 2/100
28/28 [==============================] - 0s 7ms/step - loss: 0.0019
Epoch 3/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0013
Epoch 4/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0013
Epoch 5/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0013
Epoch 6/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0013
Epoch 7/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0012
Epoch 8/100
28/28 [==============================] - 0s 7ms/step - loss: 0.0012
Epoch 9/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0011
Epoch 10/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0016
Epoch 11/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0011
Epoch 12/100
28/28 [==============================] - 0s 5ms/step - loss: 0.0011
Epoch 13/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0018
Epoch 14/100
```

```
28/28 [==============================] - 0s 6ms/step - loss: 0.0014
Epoch 15/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0010
Epoch 16/100
28/28 [==============================] - 0s 6ms/step - loss: 9.4809e-04
Epoch 17/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0010
Epoch 18/100
28/28 [==============================] - 0s 6ms/step - loss: 8.8750e-04
Epoch 19/100
28/28 [==============================] - 0s 6ms/step - loss: 8.9576e-04
Epoch 20/100
28/28 [==============================] - 0s 6ms/step - loss: 8.0271e-04
Epoch 21/100
28/28 [==============================] - 0s 6ms/step - loss: 8.5560e-04
Epoch 22/100
28/28 [==============================] - 0s 6ms/step - loss: 8.4768e-04
Epoch 23/100
28/28 [==============================] - 0s 6ms/step - loss: 9.0988e-04
Epoch 24/100
28/28 [==============================] - 0s 6ms/step - loss: 7.6974e-04
Epoch 25/100
28/28 [==============================] - 0s 6ms/step - loss: 7.3346e-04
Epoch 26/100
28/28 [==============================] - 0s 6ms/step - loss: 7.4072e-04
Epoch 27/100
28/28 [==============================] - 0s 6ms/step - loss: 7.0458e-04
Epoch 28/100
28/28 [==============================] - 0s 6ms/step - loss: 7.0068e-04
Epoch 29/100
28/28 [==============================] - 0s 7ms/step - loss: 6.7667e-04
Epoch 30/100
28/28 [==============================] - 0s 6ms/step - loss: 6.8898e-04
Epoch 31/100
28/28 [==============================] - 0s 6ms/step - loss: 6.8625e-04
Epoch 32/100
28/28 [==============================] - 0s 6ms/step - loss: 7.3622e-04
Epoch 33/100
28/28 [==============================] - 0s 9ms/step - loss: 6.2441e-04
Epoch 34/100
28/28 [==============================] - 0s 12ms/step - loss: 6.2787e-04
Epoch 35/100
28/28 [==============================] - 0s 9ms/step - loss: 6.0586e-04
Epoch 36/100
28/28 [==============================] - 0s 6ms/step - loss: 5.8176e-04
Epoch 37/100
28/28 [==============================] - 0s 6ms/step - loss: 6.6488e-04
Epoch 38/100
28/28 [==============================] - 0s 7ms/step - loss: 5.9169e-04
Epoch 39/100
28/28 [==============================] - 0s 8ms/step - loss: 5.4045e-04
Epoch 40/100
28/28 [==============================] - 0s 9ms/step - loss: 6.1624e-04
Epoch 41/100
28/28 [==============================] - 0s 8ms/step - loss: 5.7730e-04
Epoch 42/100
28/28 [==============================] - 0s 9ms/step - loss: 5.2712e-04
Epoch 43/100
28/28 [==============================] - 0s 8ms/step - loss: 5.2127e-04
Epoch 44/100
28/28 [==============================] - 0s 8ms/step - loss: 5.0688e-04
Epoch 45/100
28/28 [==============================] - 0s 8ms/step - loss: 5.0071e-04
Epoch 46/100
28/28 [==============================] - 0s 8ms/step - loss: 5.2598e-04
Epoch 47/100
28/28 [==============================] - 0s 8ms/step - loss: 6.6213e-04
Epoch 48/100
28/28 [==============================] - 0s 8ms/step - loss: 6.3248e-04
Epoch 49/100
28/28 [==============================] - 0s 8ms/step - loss: 4.6108e-04
Epoch 50/100
28/28 [==============================] - 0s 8ms/step - loss: 4.5460e-04
Epoch 51/100
28/28 [==============================] - 0s 8ms/step - loss: 4.4319e-04
Epoch 52/100
28/28 [==============================] - 0s 9ms/step - loss: 4.9935e-04
Epoch 53/100
28/28 [==============================] - 0s 9ms/step - loss: 4.4537e-04
Epoch 54/100
28/28 [==============================] - 0s 9ms/step - loss: 5.0914e-04
Epoch 55/100
28/28 [==============================] - 0s 7ms/step - loss: 4.7274e-04
Epoch 56/100
28/28 [==============================] - 0s 6ms/step - loss: 4.1486e-04
Epoch 57/100
28/28 [==============================] - 0s 6ms/step - loss: 4.2405e-04
Epoch 58/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0049
```

```
Epoch 59/100
28/28 [==============================] - 0s 6ms/step - loss: 6.9758e-04
Epoch 60/100
28/28 [==============================] - 0s 6ms/step - loss: 5.2112e-04
Epoch 61/100
28/28 [==============================] - 0s 6ms/step - loss: 4.6914e-04
Epoch 62/100
28/28 [==============================] - 0s 6ms/step - loss: 5.2609e-04
Epoch 63/100
28/28 [==============================] - 0s 6ms/step - loss: 4.4716e-04
Epoch 64/100
28/28 [==============================] - 0s 6ms/step - loss: 4.5407e-04
Epoch 65/100
28/28 [==============================] - 0s 6ms/step - loss: 5.5943e-04
Epoch 66/100
28/28 [==============================] - 0s 6ms/step - loss: 4.4751e-04
Epoch 67/100
28/28 [==============================] - 0s 6ms/step - loss: 4.1178e-04
Epoch 68/100
28/28 [==============================] - 0s 6ms/step - loss: 4.0069e-04
Epoch 69/100
28/28 [==============================] - 0s 6ms/step - loss: 4.0532e-04
Epoch 70/100
28/28 [==============================] - 0s 6ms/step - loss: 4.1844e-04
Epoch 71/100
28/28 [==============================] - 0s 6ms/step - loss: 3.6760e-04
Epoch 72/100
28/28 [==============================] - 0s 6ms/step - loss: 3.8918e-04
Epoch 73/100
28/28 [==============================] - 0s 6ms/step - loss: 4.2587e-04
Epoch 74/100
28/28 [==============================] - 0s 6ms/step - loss: 3.7470e-04
Epoch 75/100
28/28 [==============================] - 0s 6ms/step - loss: 3.8743e-04
Epoch 76/100
28/28 [==============================] - 0s 6ms/step - loss: 4.1344e-04
Epoch 77/100
28/28 [==============================] - 0s 6ms/step - loss: 3.6134e-04
Epoch 78/100
28/28 [==============================] - 0s 6ms/step - loss: 3.6631e-04
Epoch 79/100
28/28 [==============================] - 0s 6ms/step - loss: 3.5267e-04
Epoch 80/100
28/28 [==============================] - 0s 6ms/step - loss: 3.4659e-04
Epoch 81/100
28/28 [==============================] - 0s 6ms/step - loss: 3.5300e-04
Epoch 82/100
28/28 [==============================] - 0s 6ms/step - loss: 3.6680e-04
Epoch 83/100
28/28 [==============================] - 0s 6ms/step - loss: 3.4381e-04
Epoch 84/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0012
Epoch 85/100
28/28 [==============================] - 0s 6ms/step - loss: 3.7275e-04
Epoch 86/100
28/28 [==============================] - 0s 6ms/step - loss: 0.0019
Epoch 87/100
28/28 [==============================] - 0s 6ms/step - loss: 4.0897e-04
Epoch 88/100
28/28 [==============================] - 0s 6ms/step - loss: 3.3010e-04
Epoch 89/100
28/28 [==============================] - 0s 6ms/step - loss: 3.5274e-04
Epoch 90/100
28/28 [==============================] - 0s 6ms/step - loss: 4.9342e-04
Epoch 91/100
28/28 [==============================] - 0s 6ms/step - loss: 3.3107e-04
Epoch 92/100
28/28 [==============================] - 0s 6ms/step - loss: 3.2033e-04
Epoch 93/100
28/28 [==============================] - 0s 6ms/step - loss: 3.2039e-04
Epoch 94/100
28/28 [==============================] - 0s 6ms/step - loss: 3.2762e-04
Epoch 95/100
28/28 [==============================] - 0s 6ms/step - loss: 3.7336e-04
Epoch 96/100
28/28 [==============================] - 0s 6ms/step - loss: 3.2383e-04
Epoch 97/100
28/28 [==============================] - 0s 6ms/step - loss: 3.3090e-04
Epoch 98/100
28/28 [==============================] - 0s 6ms/step - loss: 3.3542e-04
Epoch 99/100
28/28 [==============================] - 0s 7ms/step - loss: 3.0527e-04
Epoch 100/100
28/28 [==============================] - 0s 6ms/step - loss: 3.3310e-04
28/28 [==============================] - 1s 3ms/step
7/7 [==============================] - 0s 5ms/step
```

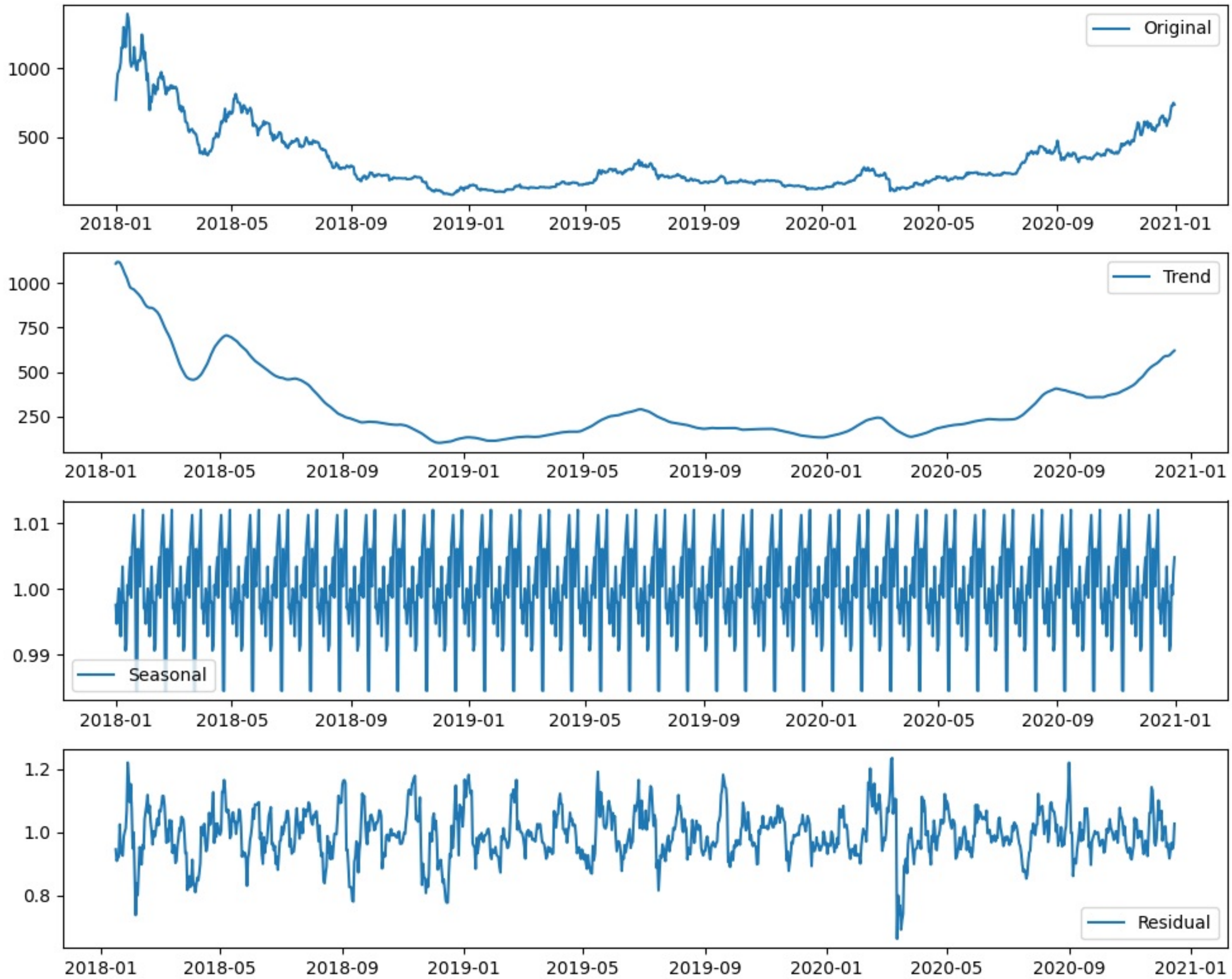
In [12]:
```
# Plotting predictions
plt.figure(figsize=(10, 6))
```

```
plt.plot(scaler.inverse_transform(data), label='True Value')
plt.plot(np.concatenate([train_predict, test_predict]), label='Predicted')
plt.xlabel('Time')
plt.ylabel('ETH Price')
plt.legend()
plt.show()
```



In [15]:
```
eth_close = eth_data['Close']
result = seasonal_decompose(eth_close, model='multiplicative', period=30) # Adjust the period as needed


# Plot the decomposition
plt.figure(figsize=(10, 8))

# Plot the decomposition
plt.figure(figsize=(10, 8))

plt.subplot(411)
plt.plot(eth_close, label='Original')
plt.legend()

plt.subplot(412)
plt.plot(result.trend, label='Trend')
plt.legend()

plt.subplot(413)
plt.plot(result.seasonal, label='Seasonal')
plt.legend()

plt.subplot(414)
plt.plot(result.resid, label='Residual')
plt.legend()

plt.tight_layout()
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



1. **Original Data:** The first subplot (`subplot(411)`) displays the original Ethereum closing prices. This plot shows the raw data without any decomposition.

2. **Trend Component:** The second subplot (`subplot(412)`) shows the trend component obtained from the decomposition. It represents the underlying trend or pattern in the data, abstracting from short-term fluctuations and seasonality.

3. **Seasonal Component:** The third subplot (`subplot(413)`) displays the seasonal component extracted from the time series. It represents the periodic fluctuations or seasonal patterns present in the data.

4. **Residual Component:** The fourth subplot (`subplot(414)`) shows the residual component, which is the remainder after removing the trend and seasonal components. It represents the random or irregular fluctuations that are not captured by the trend or seasonality.

The `seasonal_decompose` function allows for the investigation of different components of a time series, aiding in understanding underlying patterns, seasonal behavior, and irregularities.

This kind of analysis is valuable for:

- Understanding underlying trends and patterns in the data.
- Identifying seasonality or periodic fluctuations.
- Investigating irregularities or unexpected behavior in the time series.

The period parameter in `seasonal_decompose` specifies the length of the seasonal component and might need adjustment based on the frequency or periodicity of the seasonality present in the data. Adjusting this parameter can help in obtaining a more accurate decomposition of the time series.

In [16]:

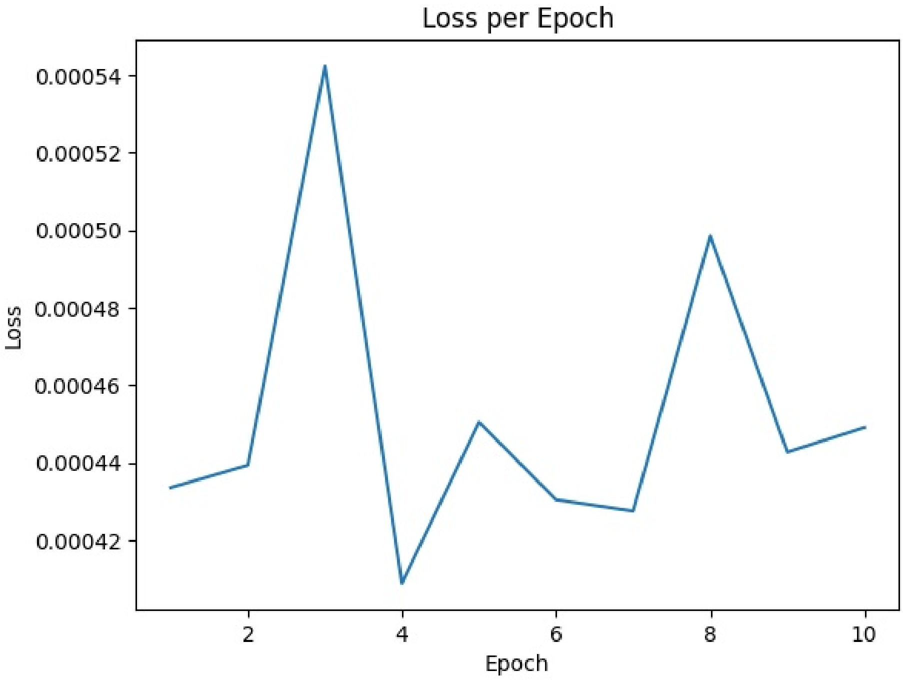
```
# Define the sequence length
sequence_length = 10  # Adjust the sequence length as needed

# Create TimeseriesGenerator for train and test sets
train_generator = TimeseriesGenerator(scaled_data, scaled_data, length=sequence_length, batch_size=1)
test_generator = TimeseriesGenerator(scaled_data, scaled_data, length=sequence_length, batch_size=1)

# Build an LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model using generator
model.fit(train_generator, epochs=10)

# Predictions using generator
predicted_values = model.predict(test_generator)

# Inverse transform predictions to original scale
predicted_values = scaler.inverse_transform(predicted_values)
```

```
Epoch 1/10
1086/1086 [==============================] - 10s 7ms/step - loss: 0.0029
Epoch 2/10
1086/1086 [==============================] - 7s 6ms/step - loss: 0.0013
Epoch 3/10
1086/1086 [==============================] - 8s 7ms/step - loss: 0.0011
Epoch 4/10
1086/1086 [==============================] - 7s 6ms/step - loss: 9.7039e-04
Epoch 5/10
1086/1086 [==============================] - 7s 7ms/step - loss: 7.9590e-04
Epoch 6/10
1086/1086 [==============================] - 7s 6ms/step - loss: 5.9195e-04
Epoch 7/10
1086/1086 [==============================] - 6s 6ms/step - loss: 5.3202e-04
Epoch 8/10
1086/1086 [==============================] - 7s 6ms/step - loss: 5.0702e-04
Epoch 9/10
1086/1086 [==============================] - 8s 7ms/step - loss: 4.7096e-04
Epoch 10/10
1086/1086 [==============================] - 6s 6ms/step - loss: 4.4949e-04
1086/1086 [==============================] - 5s 4ms/step
```

In [18]:

```
# Store loss history per epoch
history = model.fit(train_generator, epochs=10, verbose=1)

# Extracting loss values
loss_per_epoch = history.history['loss']

# Plotting the loss per epoch
plt.plot(range(1, len(loss_per_epoch) + 1), loss_per_epoch)
plt.title('Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 1/10
1086/1086 [==============================] - 7s 6ms/step - loss: 4.3353e-04
Epoch 2/10
1086/1086 [==============================] - 7s 7ms/step - loss: 4.3933e-04
Epoch 3/10
1086/1086 [==============================] - 7s 7ms/step - loss: 5.4236e-04
Epoch 4/10
1086/1086 [==============================] - 6s 6ms/step - loss: 4.0891e-04
Epoch 5/10
1086/1086 [==============================] - 7s 7ms/step - loss: 4.5043e-04
Epoch 6/10
1086/1086 [==============================] - 6s 6ms/step - loss: 4.3040e-04
Epoch 7/10
1086/1086 [==============================] - 8s 7ms/step - loss: 4.2751e-04
Epoch 8/10
1086/1086 [==============================] - 7s 6ms/step - loss: 4.9849e-04
Epoch 9/10
1086/1086 [==============================] - 7s 6ms/step - loss: 4.4269e-04
Epoch 10/10
1086/1086 [==============================] - 8s 7ms/step - loss: 4.4905e-04
```



Fluctuations: The loss values show fluctuations across epochs, not consistently decreasing or increasing. Some epochs have slightly higher losses than others.

Stability: The losses appear to be relatively close to each other, indicating that the model might have reached a certain stability in training.

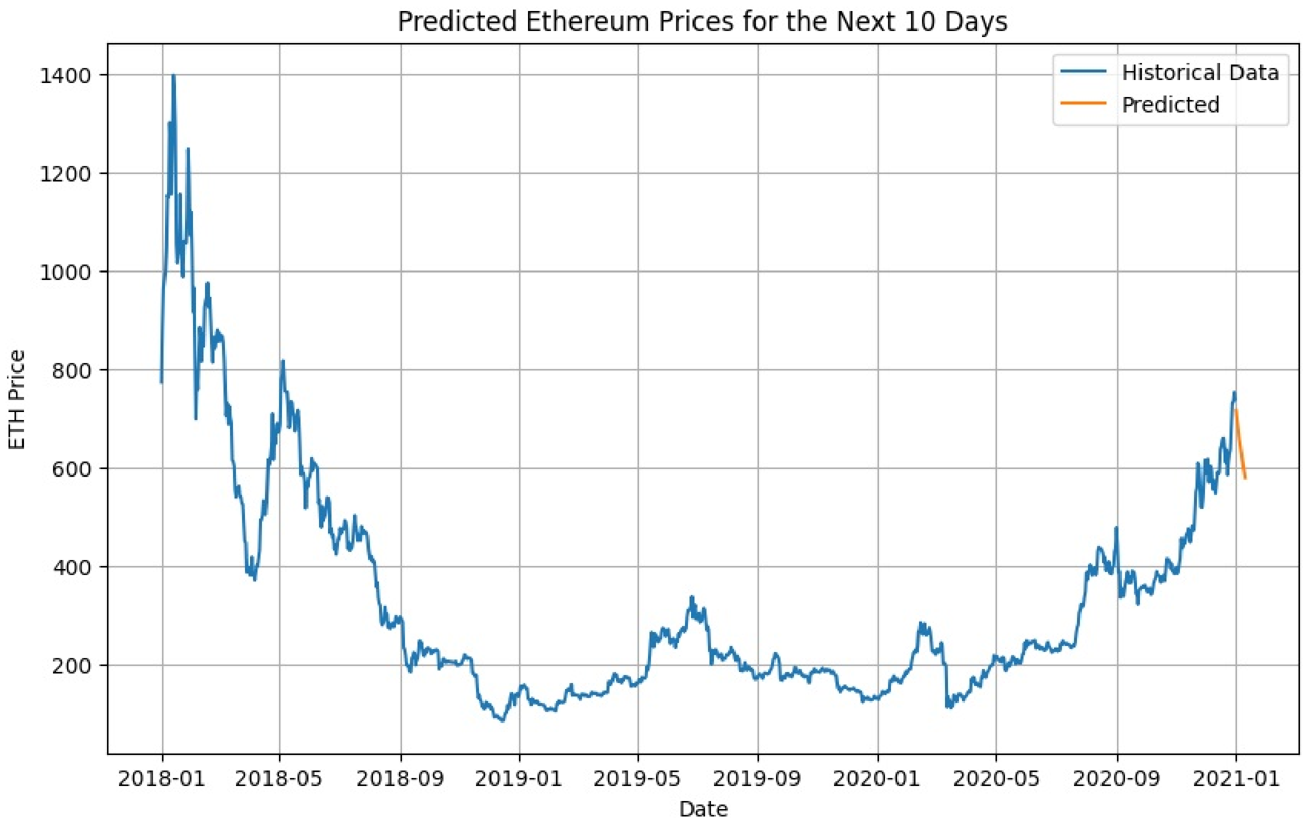
Overall Trend: While there isn't a clear decreasing trend in losses, they seem to hover around a similar range across epochs.

Evaluation: Typically, it's ideal to see a decreasing trend in loss values across epochs, signifying that the model is learning and improving its predictions. However, the interpretation of loss values heavily depends on the specific context of the problem and the dataset.

In [21]:

```
# Get the last sequence from the training data
last_sequence = scaled_data[-sequence_length:]

# Generate predictions for the next 10 days
predictions = []

for _ in range(10):
  current_prediction = model.predict(last_sequence.reshape(1, sequence_length, 1))[0, 0]
  predictions.append(current_prediction)
  last_sequence = np.append(last_sequence[1:], current_prediction).reshape(-1, 1)


# Inverse transform predictions to original scale
predicted_values = scaler.inverse_transform(np.array(predictions).reshape(-1, 1))

# Generate dates for the next 10 days
last_date = eth_data.index[-1]
dates = pd.date_range(start=last_date, periods=11)[1:]  # 11 because the first date is already in the data

# Plotting the predicted values for the next 10 days
plt.figure(figsize=(10, 6))
plt.plot(eth_data.index, eth_data['Close'], label='Historical Data')
plt.plot(dates, predicted_values, label='Predicted')
plt.xlabel('Date')
plt.ylabel('ETH Price')
plt.title('Predicted Ethereum Prices for the Next 10 Days')
plt.legend()
plt.grid(True)
plt.show()
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
```



The `reshape` function is used to adjust the shape of the `last_sequence` array before predicting the next value in the time series.

In time series forecasting using recurrent neural networks like LSTM, the input data needs to be structured in a specific format that includes:

1. **Samples:** Each sample represents a sequence of data points. For instance, a sequence of past closing prices for Ethereum.
2. **Time Steps:** Each time step within a sample represents an individual data point within the sequence.
3. **Features:** These are the different dimensions or variables present in each time step.

In the context of LSTM models in Keras, the input data needs to be 3-dimensional `(batch_size, time_steps, features)`. When you reshape the `last_sequence`, you're essentially formatting it into the shape that the LSTM model expects: `(1, sequence_length, 1)`.

Here's why `reshape` is important in time series data:

1. **Model Input Shape:** Neural networks expect data in specific shapes. Reshaping allows you to structure your data correctly so that it fits the input requirements of your model.

2. **Sequence Handling:** For LSTM models, reshaping helps organize the time series data into sequences with appropriate steps. It's crucial for preserving the temporal aspect of the data.

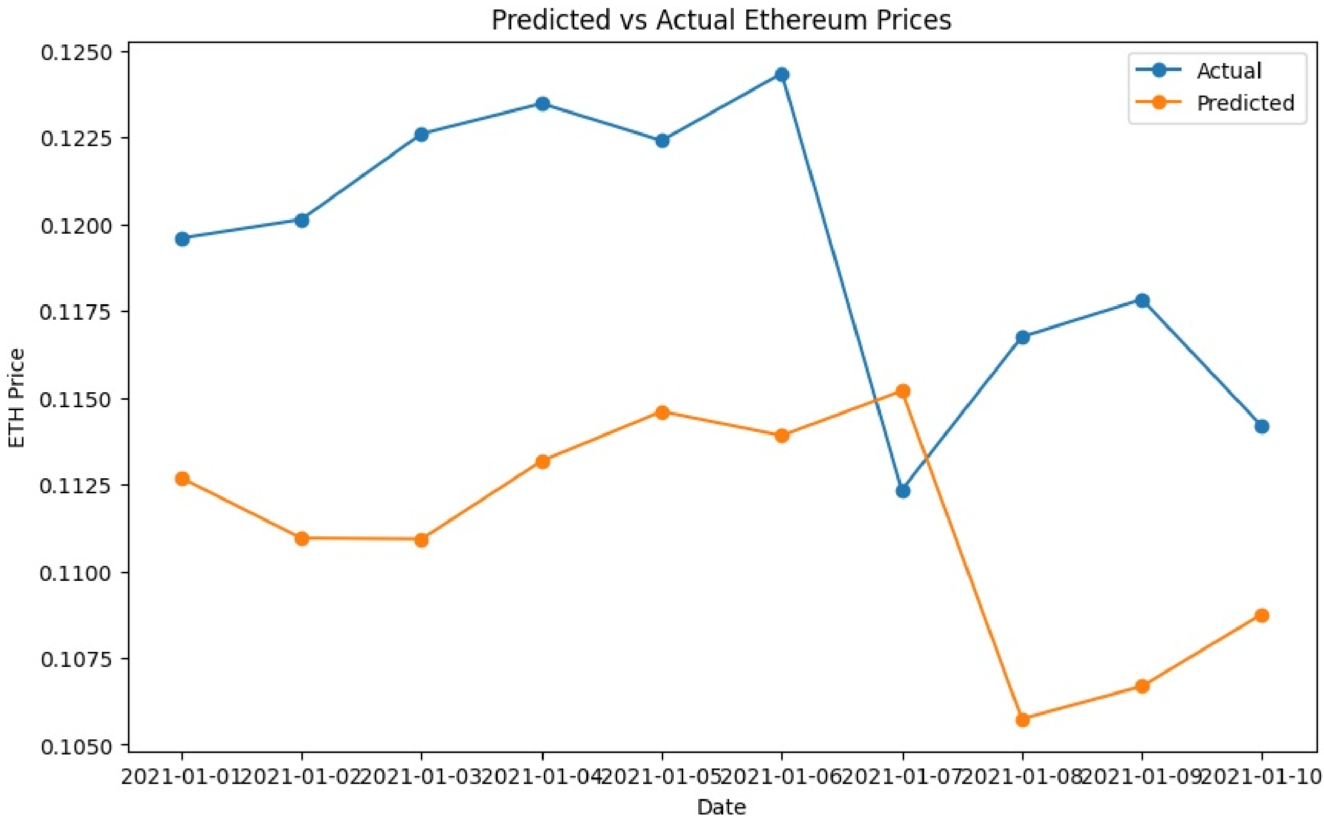
3. **Feature Engineering:** Reshaping might be used when you have multiple features in your time series data. It helps organize these features in a way that the model can learn effectively.

In the given code, `last_sequence.reshape(1, sequence_length, 1)` transforms the data into a shape that the LSTM model understands: a single sample with `sequence_length` time steps and one feature per step. This reshaping is vital to maintain the sequence structure and ensure the model can make predictions based on the historical sequence data.

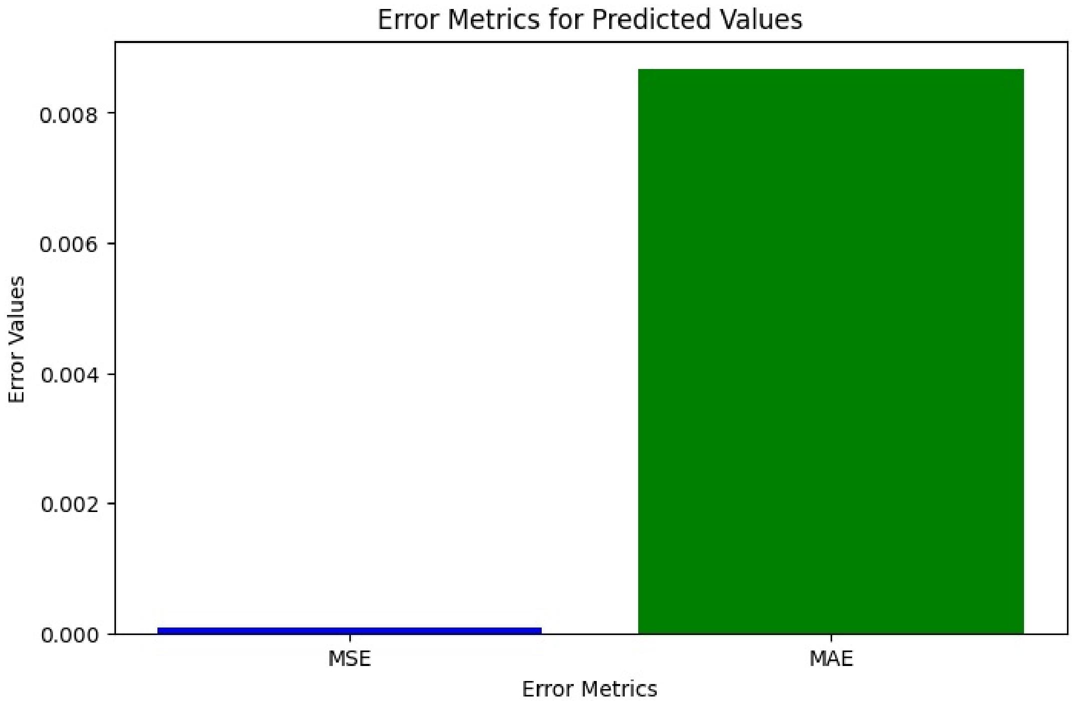
In [25]:
```
y_pred = model.predict(X_test)

# Taking the first 10 dates to align with the first 10 predictions
dates_to_plot = dates[:10]

plt.figure(figsize=(10, 6))
plt.plot(dates_to_plot, y_test[:10], label='Actual', marker='o')
plt.plot(dates_to_plot, y_pred[:10], label='Predicted', marker='o')
plt.xlabel('Date')
plt.ylabel('ETH Price')
plt.title('Predicted vs Actual Ethereum Prices')
plt.legend()
plt.show()



# Select the first 10 predicted values for comparison
y_pred_subset = y_pred[:10].squeeze()  # Remove the extra dimension if present
```

```
# Calculate error metrics
mse = mean_squared_error(y_test[:10], y_pred_subset)
mae = mean_absolute_error(y_test[:10], y_pred_subset)

# Print the calculated error metrics
print(f"MSE: {mse}")
print(f"MAE: {mae}")

# Plotting error metrics
plt.figure(figsize=(8, 5))
plt.bar(['MSE', 'MAE'], [mse, mae], color=['blue', 'green'])
plt.xlabel('Error Metrics')
plt.ylabel('Error Values')
plt.title('Error Metrics for Predicted Values')
plt.show()
```

```
7/7 [==============================] - 0s 3ms/step
```




```
MSE: 8.26733330445227e-05
MAE: 0.008671124264126742
```



In [ ]:

- **Mean Squared Error (MSE)** measures the average of the squares of the errors or deviations, which gives more weight to large errors. A lower MSE indicates a better fit between predicted and actual values. In this case, the MSE of approximately 8.27e-05 suggests that, on average, the squared differences between predicted and actual values are very small.

- **Mean Absolute Error (MAE)** measures the average absolute differences between predicted and actual values. It provides a more straightforward understanding of the average magnitude of errors. An MAE of around 0.00867 indicates that, on average, the model's predictions deviate by approximately 0.00867 units from the actual values.

In summary, both metrics indicate that the model's predictions are relatively close to the actual values, with small average deviations between predicted and actual values for the given subset of data. Lower values for these metrics generally signify better model performance. However, to assess the model comprehensively, it's essential to consider these metrics along with other evaluation techniques and on larger datasets if available.

In [ ]: