In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
```

In [2]:
```
# Downloading stock data for Apple (AAPL)
data = yf.download('AAPL', start='2015-01-01', end='2022-12-31')
data = data[['Close']]  # using the closing price
data.head()
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
```

Out[2]:

Price	Close
Ticker	AAPL
Date	
2015-01-02	24.320435
2015-01-05	23.635279
2015-01-06	23.637510
2015-01-07	23.968956
2015-01-08	24.889904

In [3]:
```
plt.figure(figsize=(10, 4))
plt.plot(data['Close'])
plt.title("AAPL Stock Price")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.grid()
plt.show()
```

In [4]:
```python
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

X = []
y = []

time_step = 60  # Using 60 days to predict the 61st

for i in range(time_step, len(scaled_data)):
    X.append(scaled_data[i-time_step:i, 0])
    y.append(scaled_data[i, 0])

X, y = np.array(X), np.array(y)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

In [5]:
```python
train_size = int(len(X) * 0.8)

X_train = X[:train_size]
y_train = y[:train_size]
X_test = X[train_size:]
y_test = y[train_size:]
```

In [6]:
```python
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(1))

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400

 dropout (Dropout)           (None, 60, 50)            0

 lstm_1 (LSTM)               (None, 50)                20200

 dropout_1 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```python
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test)
```

```
Epoch 1/10
49/49 [==============================] - 7s 72ms/step - loss: 0.0079 - val_loss: 0.0018
Epoch 2/10
49/49 [==============================] - 3s 53ms/step - loss: 0.0011 - val_loss: 0.0023
Epoch 3/10
49/49 [==============================] - 2s 51ms/step - loss: 0.0011 - val_loss: 0.0017
Epoch 4/10
49/49 [==============================] - 2s 50ms/step - loss: 0.0010 - val_loss: 0.0020
Epoch 5/10
49/49 [==============================] - 2s 50ms/step - loss: 8.7880e-04 - val_loss: 0.0016
Epoch 6/10
49/49 [==============================] - 2s 50ms/step - loss: 8.7068e-04 - val_loss: 0.0032
Epoch 7/10
49/49 [==============================] - 2s 51ms/step - loss: 8.8790e-04 - val_loss: 0.0033
Epoch 8/10
49/49 [==============================] - 3s 55ms/step - loss: 9.8547e-04 - val_loss: 0.0014
Epoch 9/10
49/49 [==============================] - 3s 52ms/step - loss: 8.2853e-04 - val_loss: 0.0023
Epoch 10/10
49/49 [==============================] - 2s 50ms/step - loss: 8.3099e-04 - val_loss: 0.0021
```

In [8]:

```python
predicted = model.predict(X_test)
predicted = scaler.inverse_transform(predicted.reshape(-1, 1))
actual = scaler.inverse_transform(y_test.reshape(-1, 1))

plt.figure(figsize=(10, 4))
plt.plot(actual, label='Actual Price')
plt.plot(predicted, label='Predicted Price')
plt.title("AAPL Price Prediction")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.show()
```

```
13/13 [==============================] - 1s 13ms/step
```

In [9]:

```python
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(actual, predicted))
print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 7.220755809081442
```

In [ ]:

```
```