In [59]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as web
import datetime as dt

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM


stock1 = pd.read_csv('/Users/ashiyabhandari/Desktop/apple_stock.csv', index_
```

In [60]:
```
stock = stock.loc['2020-01-01':]
stock.head()
```

Out[60]:

	Adj Close	Close	High	Low	Open	Volume
2020-01-02	72.796043	75.087502	75.150002	73.797501	74.059998	135480400
2020-01-03	72.088303	74.357498	75.144997	74.125000	74.287498	146322800
2020-01-06	72.662712	74.949997	74.989998	73.187500	73.447502	118387200
2020-01-07	72.320969	74.597504	75.224998	74.370003	74.959999	108872000
2020-01-08	73.484375	75.797501	76.110001	74.290001	74.290001	132079200

In [61]:
```
plt.title("Close Price of Apple from 2020 to 2024", size = 11)
plt.plot(stock['Close'])
plt.xlabel("Date", size = 10)
plt.ylabel("Closing Price", size = 10)
plt.show()
```

In [62]:
```
#creating min max scaler
scaler = MinMaxScaler(feature_range=(0,1))

#fit the scaler to stock dataset, convert it into numpy using .values and th
scaled_data = scaler.fit_transform(stock['Close'].values.reshape(-1,1))
```

In [63]:
```
#number of days you want the prediction for to see what the price is going t
#advise to not have only few days because model will only gather few informa
days_prediction = 60

train_size = int(len(scaled_data) * 0.8)
#first 80% for training
train_data = scaled_data[:train_size]
#20% for testing
test_data = scaled_data[train_size:]

x_train = []
y_train = []

x_test = []
y_test = []
```

```
#loops from the position of scaled_data starting from days_prediction
for x in range(days_prediction, len(train_data)):
    #creating a set of past data points that will be used to make prediction
    #collects the past data and adds it to the list, x-train. This helps mod
    x_train.append(train_data[x - days_prediction: x])
    #y takes the value for today's price. (what we want to predict for today
    y_train.append(train_data[x, 0])


for x in range(days_prediction, len(test_data)):
    x_test.append(test_data[x - days_prediction:x])
    y_test.append(test_data[x, 0])

#conversiting list to arrays as machine learning language mostly work with n
x_train = np.array(x_train)
y_train = np.array(y_train)

x_test = np.array(x_test)
y_test = np.array(y_test)


#using np.reshape() function to change the shape of x_train which is require
#ex: if you had 100 samples and each sample used 5 past days,
#the reshaped x_train would have the shape (100, 5, 1).
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Build a Model

In [65]:

```
model = Sequential()

#more layers you add more sophisticated
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1
#drops at 20% randomly during training to prevent the model from memorizing
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
#return_sequence = False by default, learns from all the previous layers
model.add(LSTM(units=50))
model.add(Dropout(0.2))

#outputs the final stock price prediction
model.add(Dense(units=1))
```

In [66]:

```
#Compiling the model. Need to specify optimizer, loss function, and any metr
model.compile(optimizer='adam', loss='mean_squared_error')
```

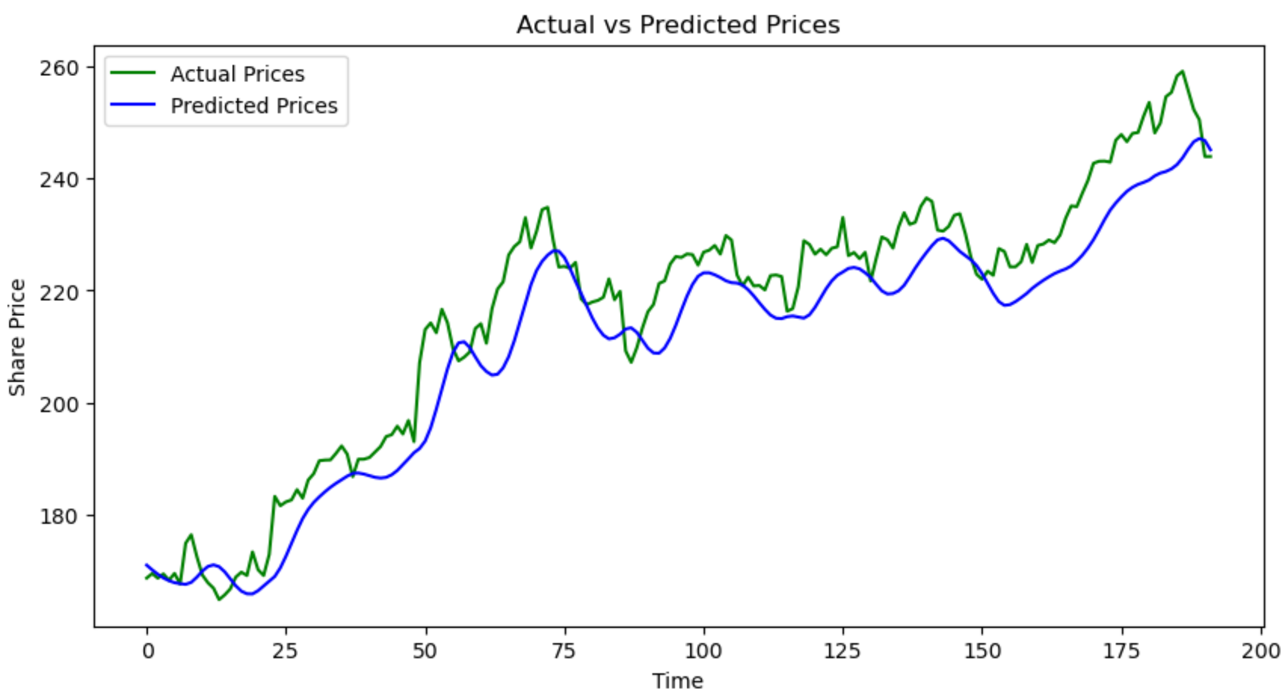
```
#train the model on training data.
#model is going to see 32 batch at the same time.
model.fit(x_train, y_train, epochs=50, batch_size=32)

#make prediction
predicted_prices = model.predict(x_test)

#reverse the scaling to get the actual values
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test.reshape(-1, 1))  # Convert a
```

```
Epoch 1/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - loss: 0.0902
Epoch 2/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0051
Epoch 3/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0035
Epoch 4/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0031
Epoch 5/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 0.0036
Epoch 6/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 0.0031
Epoch 7/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 0.0029
Epoch 8/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - loss: 0.0025
Epoch 9/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - loss: 0.0028
Epoch 10/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0029
Epoch 11/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0026
Epoch 12/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0025
Epoch 13/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0028
Epoch 14/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 0.0022
Epoch 15/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0024
Epoch 16/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 0.0020
Epoch 17/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0025
Epoch 18/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0024
Epoch 19/50
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0024
Epoch 20/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0021
Epoch 21/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0026
Epoch 22/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0021
Epoch 23/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0022
Epoch 24/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0022
Epoch 25/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - loss: 0.0020
Epoch 26/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0018
Epoch 27/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0019
Epoch 28/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0017
Epoch 29/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0019
Epoch 30/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 0.0016
Epoch 31/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 0.0016
Epoch 32/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 0.0017
Epoch 33/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 0.0019
Epoch 34/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 0.0018
Epoch 35/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 0.0016
Epoch 36/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0016
Epoch 37/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0016
Epoch 38/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0016
Epoch 39/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0016
Epoch 40/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0016
Epoch 41/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0017
Epoch 42/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0016
Epoch 43/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - loss: 0.0016
Epoch 44/50
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0016
Epoch 45/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0016
Epoch 46/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 0.0016
Epoch 47/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0016
Epoch 48/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 0.0014
Epoch 49/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 0.0014
Epoch 50/50
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.0013
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

In [67]:
```python
plt.figure(figsize=(10, 5))
plt.plot(actual_prices, color = 'green', label='Actual Prices')
plt.plot(predicted_prices, color = 'blue', label='Predicted Prices')
plt.title('Actual vs Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```



In [68]:
```python
print("Actual Price | Predicted Price")
print("-" * 30)

for x, y in zip(actual_prices.flatten(), predicted_prices.flatten()):
    print(f"   { float(x):.3f}            {float(y):.3f}")
```

```
Actual Price | Predicted Price
------------------------------
   168.840          171.140
   169.650          170.290
   168.820          169.524
   169.580          168.860
   168.450          168.375
   169.670          168.018
   167.780          167.847
   175.040          167.729
   176.550          168.075
   172.690          168.993
   169.380          170.094
   168.000          170.910
   167.040          171.187
   165.000          170.870
   165.840          169.964
   166.900          168.723
   169.020          167.460
   169.890          166.511
   169.300          166.047
   173.500          166.023
   170.330          166.592
   169.300          167.435
   173.030          168.256
   183.380          169.129
   181.710          170.666
   182.400          172.722
   182.740          175.044
   184.570          177.330
   183.050          179.427
   186.280          181.054
   187.430          182.315
   189.720          183.301
   189.840          184.190
   189.870          185.002
   191.040          185.705
   192.350          186.338
   190.900          186.971
   186.880          187.477
   189.980          187.532
   189.990          187.345
   190.290          187.049
   191.290          186.768
   192.250          186.639
   194.030          186.759
   194.350          187.225
   195.870          187.987
   194.480          189.016
   196.890          190.067
```

193.120	191.147
207.150	191.896
213.070	193.218
214.240	195.601
212.490	198.897
216.670	202.460
214.290	206.027
209.680	208.996
207.490	210.693
208.140	210.885
209.070	209.879
213.250	208.202
214.100	206.633
210.620	205.596
216.750	204.956
220.270	205.110
221.550	206.250
226.340	208.237
227.820	211.049
228.680	214.364
232.980	217.743
227.570	221.072
230.540	223.567
234.400	225.205
234.820	226.320
228.880	227.057
224.180	227.008
224.310	225.881
223.960	223.957
225.010	221.651
218.540	219.475
217.490	217.270
217.960	215.144
218.240	213.341
218.800	212.055
222.080	211.401
218.360	211.600
219.860	212.237
209.270	213.167
207.230	213.376
209.820	212.570
213.310	211.145
216.240	209.740
217.530	208.899
221.270	208.850
221.720	209.785
224.720	211.498
226.050	213.830
225.890	216.472
226.510	218.992

226.400	221.090
224.530	222.551
226.840	223.159
227.180	223.169
228.030	222.804
226.490	222.319
229.790	221.738
229.000	221.408
222.770	221.333
220.850	220.978
222.380	220.142
220.820	219.059
220.910	217.832
220.110	216.659
222.660	215.642
222.770	215.075
222.500	215.012
216.320	215.353
216.790	215.476
220.690	215.275
228.870	215.096
228.200	215.686
226.470	217.007
227.370	218.652
226.370	220.340
227.520	221.722
227.790	222.718
233.000	223.309
226.210	223.940
226.780	224.122
225.670	223.860
226.800	223.176
221.690	222.322
225.770	221.088
229.540	219.944
229.040	219.395
227.550	219.480
231.300	219.961
233.850	220.933
231.780	222.419
232.150	224.009
235.000	225.443
236.480	226.767
235.860	228.007
230.760	229.023
230.570	229.315
231.410	228.866
233.400	227.938
233.670	226.964
230.100	226.199

```
225.910          225.467
222.910          224.458
222.010          223.003
223.450          221.220
222.720          219.495
227.480          218.039
226.960          217.369
224.230          217.495
224.230          218.035
225.120          218.735
228.220          219.480
225.000          220.396
228.020          221.139
228.280          221.828
229.000          222.453
228.520          223.025
229.870          223.465
232.870          223.841
235.060          224.385
234.930          225.244
237.330          226.300
239.590          227.565
242.650          229.049
243.010          230.809
243.040          232.650
242.840          234.317
246.750          235.585
247.770          236.676
246.490          237.658
247.960          238.374
248.130          238.896
251.040          239.244
253.480          239.678
248.050          240.412
249.790          240.907
254.490          241.184
255.270          241.656
258.200          242.414
259.020          243.615
255.590          245.157
252.200          246.480
250.420          247.057
243.850          246.698
243.860          245.044
```

Prediction for Future Day

In [70]:
```
model_inputs = scaled_data
```

```
real_data = model_inputs[len(model_inputs) - days_prediction : len(model_inp
real_data = np.array(real_data)
real_data = np.reshape(real_data, (1, real_data.shape[0], 1))  # Reshape for


prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
print(f"Prediction: {prediction}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
Prediction: [[242.4975]]
```