In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

```
C:\Users\Tahir\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' o
r newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
# Some functions to help out with
def plot_predictions(test,predicted):
    plt.plot(test, color='red',label='Real IBM Stock Price')
    plt.plot(predicted, color='blue',label='Predicted IBM Stock Price')
    plt.title('IBM Stock Price Prediction')
    plt.xlabel('Time')
    plt.ylabel('IBM Stock Price')
    plt.legend()
    plt.show()

def return_rmse(test,predicted):
    rmse = math.sqrt(mean_squared_error(test, predicted))
    print("The root mean squared error is {}.".format(rmse))
```

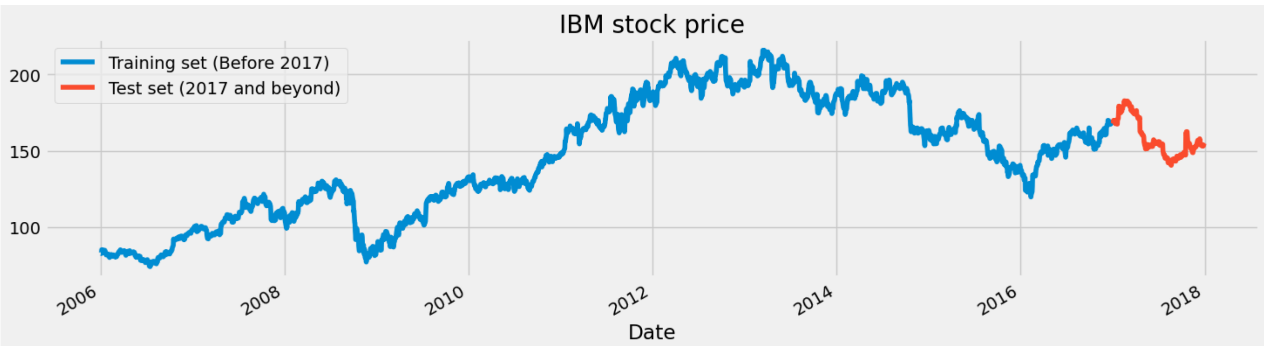
In [4]:
```
# First, we get the data
dataset = pd.read_csv('IBM_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
dataset.head()
```

Out[4]:

	Open	High	Low	Close	Volume	Name
Date						
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

In [5]:
```
# Checking for missing values
training_set = dataset[:'2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

In [6]:
```
# We have chosen 'High' attribute for prices. Let's see what it looks like
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)
dataset["High"]['2017':].plot(figsize=(16,4),legend=True)
plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])
plt.title('IBM stock price')
plt.show()
```



In [7]:
```
# Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

In [8]:
```python
# Since LSTMs store long term memory state, we create a data structure with 60 timesteps and 1 output
# So for each element of training set, we have 60 previous training set elements
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(training_set_scaled[i-60:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

In [9]:
```python
# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

In [10]:
```python
# The LSTM architecture
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fourth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
regressor.fit(X_train,y_train,epochs=50,batch_size=32)
```

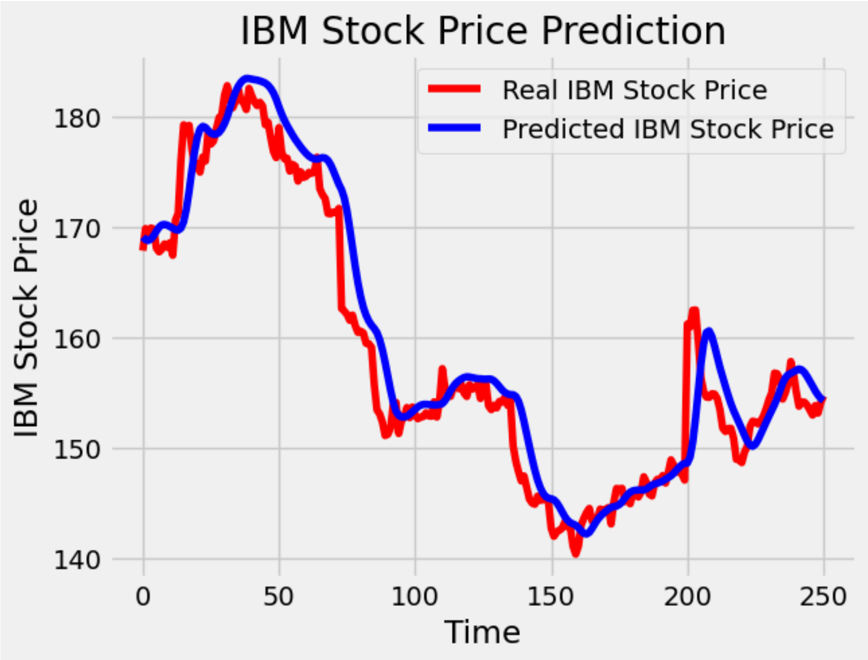
```
C:\Users\Tahir\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 15s 78ms/step - loss: 0.0423
Epoch 2/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 77ms/step - loss: 0.0110
Epoch 3/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 79ms/step - loss: 0.0082
Epoch 4/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 78ms/step - loss: 0.0065
Epoch 5/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 76ms/step - loss: 0.0067
Epoch 6/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 77ms/step - loss: 0.0055
Epoch 7/50
85/85 ━━━━━━━━━━━━━━━━━━━━ 7s 77ms/step - loss: 0.0057
Epoch 8/50
```

In [11]:
```python
# Now to get the test set ready in a similar way as the training set.
# The following has been done so forst 60 entires of test set have 60 previous values which is impossible to get unless we
# 'High' attribute data for processing
dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

In [12]:
```python
# Preparing X_test and predicting the prices
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 154ms/step
```

In [13]:
```python
# Visualizing the results for LSTM
plot_predictions(test_set,predicted_stock_price)
```



In [14]:
```python
# Evaluating our model
return_rmse(test_set,predicted_stock_price)
```

```
The root mean squared error is 3.2608720502921993.
```

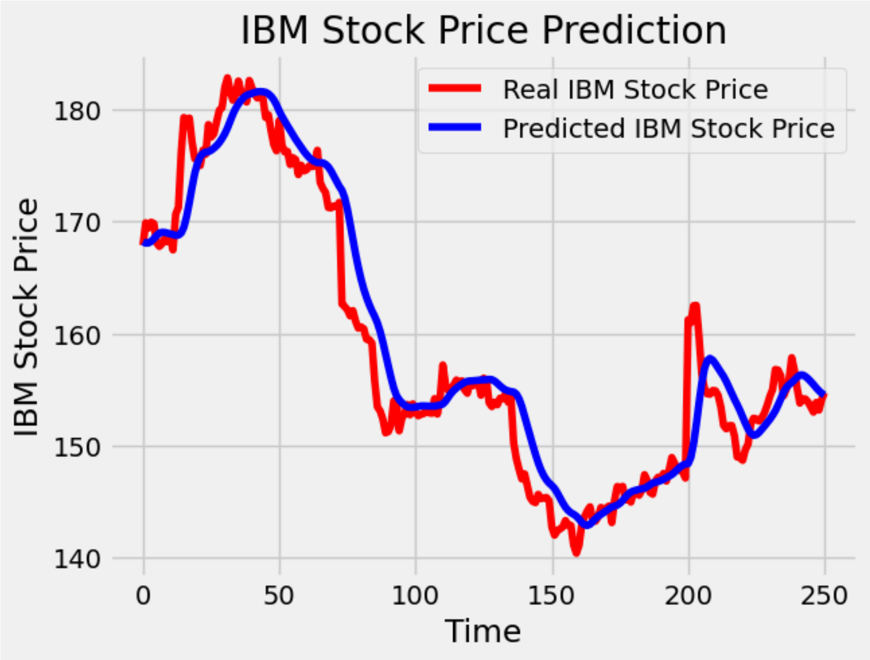
In [22]:
```python
# The GRU architecture
regressorGRU = Sequential()
# First GRU layer with Dropout regularisation
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Second GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Third GRU layer
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],1), activation='tanh'))
regressorGRU.add(Dropout(0.2))
# Fourth GRU layer
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dropout(0.2))
# The output layer
regressorGRU.add(Dense(units=1))
# Compiling the RNN with updated learning rate syntax
regressorGRU.compile(optimizer=SGD(learning_rate=0.01, decay=1e-7, momentum=0.9, nesterov=False), loss='mean_squared_error'
# Fitting to the training set
regressorGRU.fit(X_train,y_train,epochs=50,batch_size=150)
```

```
Epoch 21/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 202ms/step - loss: 0.0025
Epoch 22/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 199ms/step - loss: 0.0025
Epoch 23/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 199ms/step - loss: 0.0024
Epoch 24/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 201ms/step - loss: 0.0026
Epoch 25/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 199ms/step - loss: 0.0023
Epoch 26/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 197ms/step - loss: 0.0023
Epoch 27/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 207ms/step - loss: 0.0023
Epoch 28/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 200ms/step - loss: 0.0024
Epoch 29/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 197ms/step - loss: 0.0021
Epoch 30/50
```

In [23]:
```python
# Preparing X_test and predicting the prices
X_test = []
for i in range(60,311):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
GRU_predicted_stock_price = regressorGRU.predict(X_test)
GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted_stock_price)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 185ms/step
```

In [24]:
```python
# Visualizing the results for GRU
plot_predictions(test_set,GRU_predicted_stock_price)
```



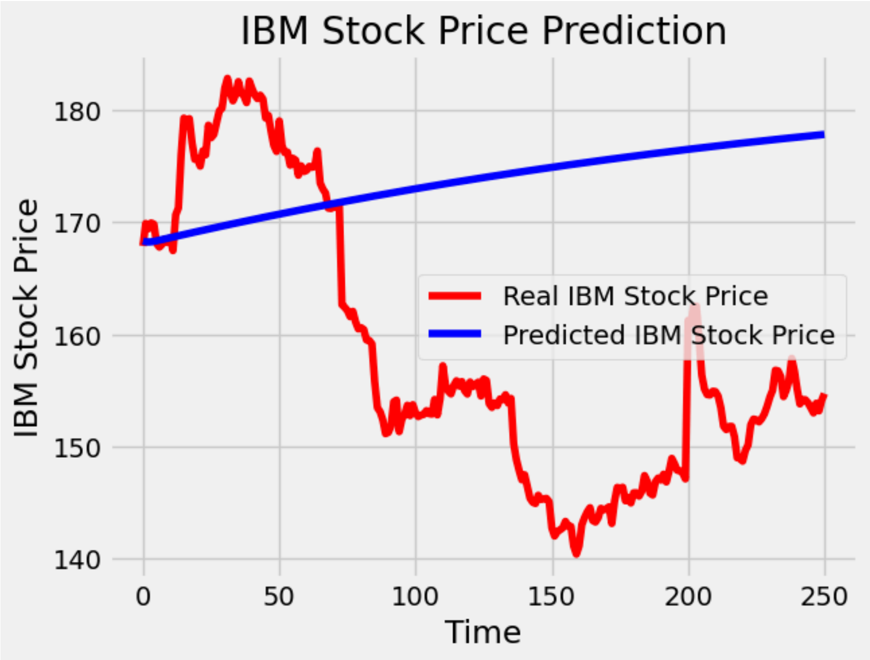
In [25]:
```python
# Evaluating GRU
return_rmse(test_set,GRU_predicted_stock_price)
```

```
The root mean squared error is 3.274031065295633.
```

In [26]:
```python
# Preparing sequence data
initial_sequence = X_train[2708,:]
sequence = []
for i in range(251):
    new_prediction = regressorGRU.predict(initial_sequence.reshape(initial_sequence.shape[1],initial_sequence.shape[0],1))
    initial_sequence = initial_sequence[1:]
    initial_sequence = np.append(initial_sequence,new_prediction,axis=0)
    sequence.append(new_prediction)
sequence = sc.inverse_transform(np.array(sequence).reshape(251,1))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
```

In [27]:
```
# Visualizing the sequence
plot_predictions(test_set,sequence)
```



In [28]:
```
# Evaluating the sequence
return_rmse(test_set,sequence)
```

The root mean squared error is 20.915662509996988.

In [ ]: