Load and preprocess the MNIST dataset

In [1]:

```
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten the input images from 28x28 pixels to 784-dimensional vectors and normalize
def preprocess_data(X):
    X = X.reshape(X.shape[0], 784).astype('float32') / 255
    return X

X_train = preprocess_data(X_train)
X_test = preprocess_data(X_test)

# Convert the target labels to categorical one-hot encoding
def one_hot_encode_labels(y):
    return np_utils.to_categorical(y, 10)

y_train = one_hot_encode_labels(y_train)
y_test = one_hot_encode_labels(y_test)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.
npz
11490434/11490434 [==============================] - 1s 0us/step
```

Build the neural network model

In [2]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

def create_mnist_model():
    model = Sequential()
    model.add(Dense(512, input_shape=(784,), activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    return model

def compile_model(model):
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'

# Create the model
model = create_mnist_model()

# Compile the model
compile_model(model)
```

Train the model

In [3]:

```
def train_model(model, X_train, y_train, X_test, y_test, epochs=10, batch_size=128):
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epoch
    return history

# Train the model
history = train_model(model, X_train, y_train, X_test, y_test, epochs=10, batch_size=128
```

```
Epoch 1/10
469/469 [==============================] - 13s 25ms/step - loss: 0.2507 - accuracy: 0.92
43 - val_loss: 0.1230 - val_accuracy: 0.9608
```

```
Epoch 2/10
469/469 [==============================] - 12s 26ms/step - loss: 0.1020 - accuracy: 0.96
83 - val_loss: 0.0787 - val_accuracy: 0.9756
Epoch 3/10
469/469 [==============================] - 12s 25ms/step - loss: 0.0731 - accuracy: 0.97
70 - val_loss: 0.0827 - val_accuracy: 0.9727
Epoch 4/10
469/469 [==============================] - 12s 25ms/step - loss: 0.0555 - accuracy: 0.98
24 - val_loss: 0.0691 - val_accuracy: 0.9775
Epoch 5/10
469/469 [==============================] - 12s 25ms/step - loss: 0.0457 - accuracy: 0.98
50 - val_loss: 0.0744 - val_accuracy: 0.9779
Epoch 6/10
469/469 [==============================] - 12s 25ms/step - loss: 0.0396 - accuracy: 0.98
70 - val_loss: 0.0710 - val_accuracy: 0.9803
Epoch 7/10
469/469 [==============================] - 12s 25ms/step - loss: 0.0342 - accuracy: 0.98
93 - val_loss: 0.0793 - val_accuracy: 0.9758
Epoch 8/10
469/469 [==============================] - 11s 24ms/step - loss: 0.0294 - accuracy: 0.99
01 - val_loss: 0.0654 - val_accuracy: 0.9823
Epoch 9/10
469/469 [==============================] - 11s 24ms/step - loss: 0.0276 - accuracy: 0.99
08 - val_loss: 0.0717 - val_accuracy: 0.9804
Epoch 10/10
469/469 [==============================] - 11s 24ms/step - loss: 0.0261 - accuracy: 0.99
12 - val_loss: 0.0677 - val_accuracy: 0.9832
```

Evaluate the model

In [4]:

```python
def evaluate_and_print_accuracy(model, X_test, y_test):
    accuracy = model.evaluate(X_test, y_test)[1]
    return accuracy

# Evaluate and print the accuracy
accuracy = evaluate_and_print_accuracy(model, X_test, y_test)
print("Accuracy:", accuracy)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0677 - accuracy: 0.9832
Accuracy: 0.9832000136375427
```

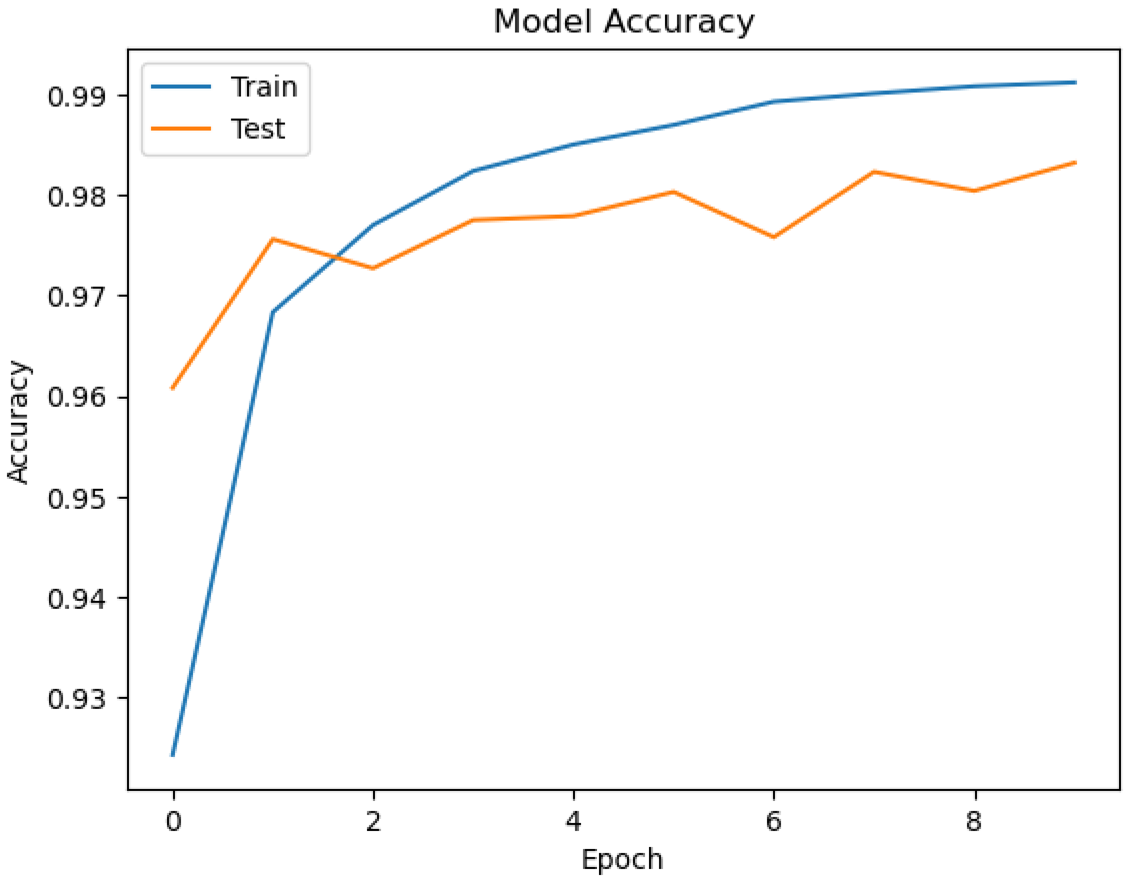
Plot the training history

In [5]:

```python
import matplotlib.pyplot as plt

def plot_accuracy(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

# Plot the training and validation accuracy
plot_accuracy(history)
```



Make predictions

In [6]:

```
def predict_single_image(model, image):
    # Expand the dimensions of the image to match the model's input shape
    image = np.expand_dims(image, axis=0)

    # Use the model to make predictions
    prediction = model.predict(image)

    # Get the predicted class
    predicted_class = np.argmax(prediction)

    return predicted_class

# Predict on a single image
digit = X_test[0]
predicted_digit = predict_single_image(model, digit)
print("Predicted digit:", predicted_digit)
```

```
1/1 [==============================] - 0s 205ms/step
Predicted digit: 7
```