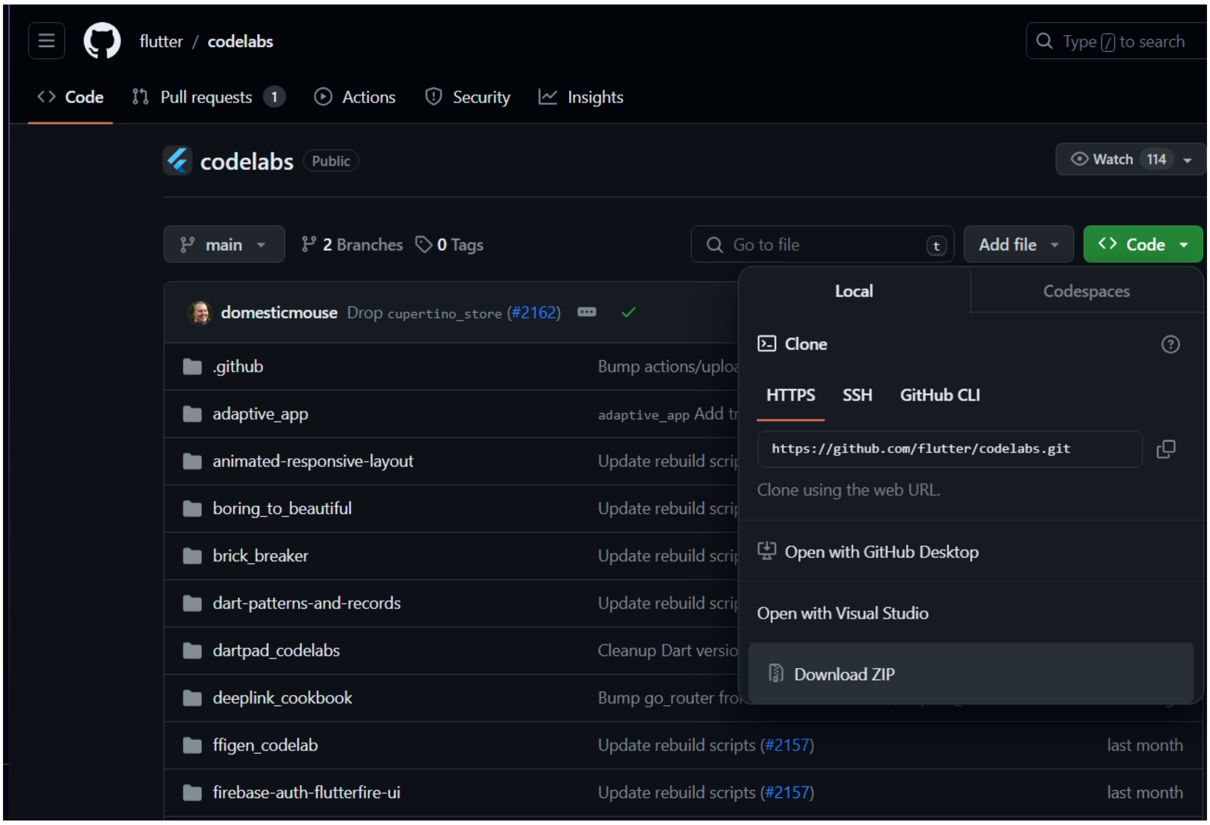
Lab session 6: Integrating Machine Learning Model in a Flutter application

Task-3 Create a Flutter app to classify texts

In this documentation, we will learn how to run a text-classification inference from a Flutter app with **TensorFlow Serving** through **REST** and **gRPC**.

To download the code for this codelab:

1. Navigate to the GitHub repository for this codelab.
2. Click Code > Download zip to download all the code for this codelab.



domesticmouse Drop cupertino_store (#2162)

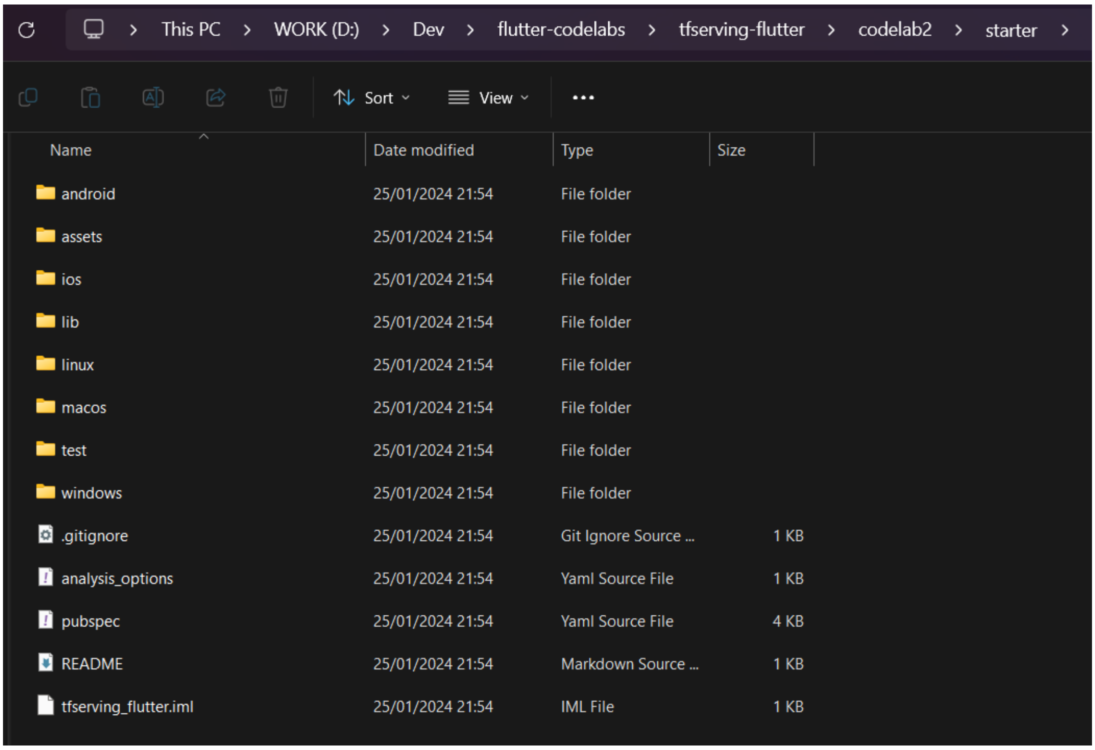
.github	Bump actions/uploa	
adaptive_app	adaptive_app Add tr	
animated-responsive-layout	Update rebuild scrip	
boring_to_beautiful	Update rebuild scrip	
brick_breaker	Update rebuild scrip	
dart-patterns-and-records	Update rebuild scrip	
dartpad_codelabs	Cleanup Dart versio	
deeplink_cookbook	Bump go_router fro	
ffigen_codelab	Update rebuild scripts (#2157)	last month
firebase-auth-flutterfire-ui	Update rebuild scripts (#2157)	last month

```
https://github.com/flutter/codelabs.git
```

Clone using the web URL.

3. Unzip the downloaded zip file to unpack a codelabs-main root folder with all the resources that you need.

For this codelab, we only need the files in the tfserving-flutter/codelab2 subdirectory in the repository, which contains the starter folder with the starter code that we will build upon for this codelab.

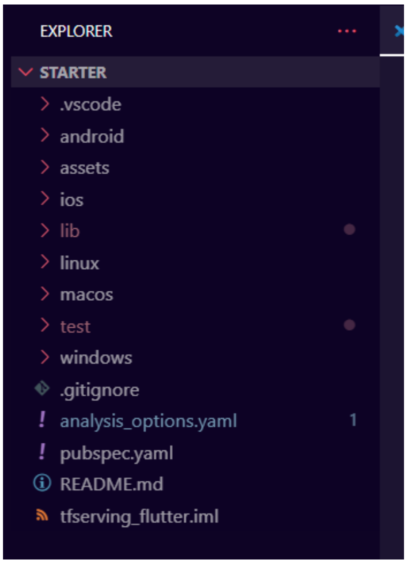


This PC > WORK (D:) > Dev > flutter-codelabs > tfserving-flutter > codelab2 > starter >

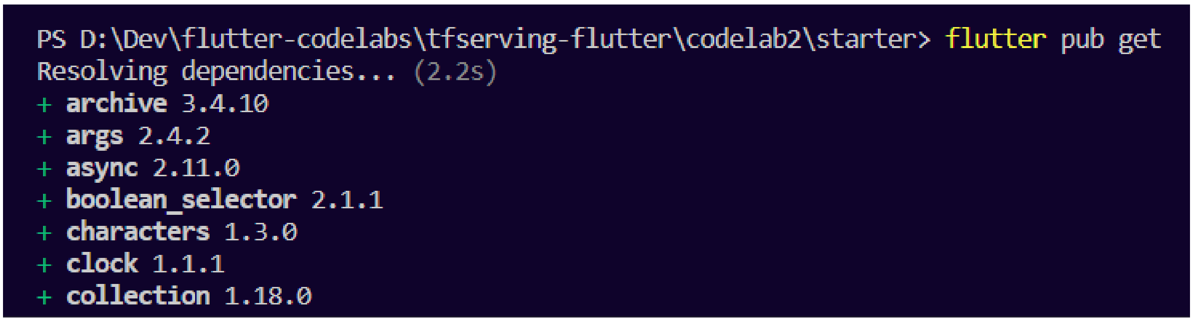
Name	Date modified	Type	Size
android	25/01/2024 21:54	File folder	
assets	25/01/2024 21:54	File folder	
ios	25/01/2024 21:54	File folder	
lib	25/01/2024 21:54	File folder	
linux	25/01/2024 21:54	File folder	
macos	25/01/2024 21:54	File folder	
test	25/01/2024 21:54	File folder	
windows	25/01/2024 21:54	File folder	
.gitignore	25/01/2024 21:54	Git Ignore Source ...	1 KB
analysis_options	25/01/2024 21:54	Yaml Source File	1 KB
pubspec	25/01/2024 21:54	Yaml Source File	4 KB
README	25/01/2024 21:54	Markdown Source ...	1 KB
tfserving_flutter.iml	25/01/2024 21:54	IML File	1 KB

Download the dependencies for the project

In the first place, we will open the code in vs code and get all the dependancies



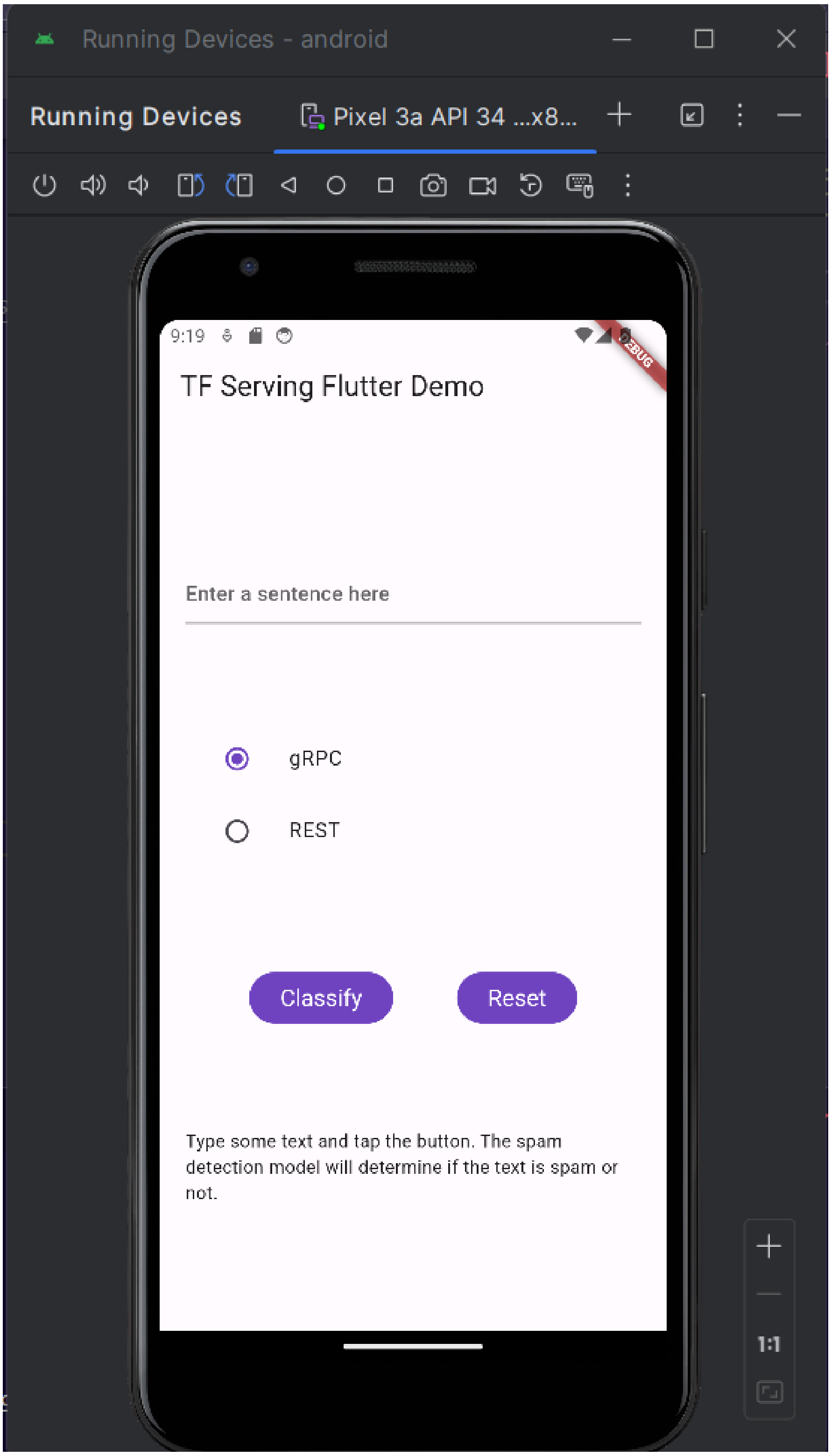
```
STARTER
  .vscode
  android
  assets
  ios
  lib
  linux
  macos
  test
  windows
  .gitignore
  analysis_options.yaml   1
  pubspec.yaml
  README.md
  tfserving_flutter.iml
```



```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> flutter pub get
Resolving dependencies... (2.2s)
+ archive 3.4.10
+ args 2.4.2
+ async 2.11.0
+ boolean_selector 2.1.1
+ characters 1.3.0
+ clock 1.1.1
+ collection 1.18.0
```

Run and explore the app

Once we have all the necessary dependancies, we will run the app to get an overview of all the features we can see in it and how it's displayed:



TF Serving Flutter Demo

Enter a sentence here

● gRPC

○ REST

Classify

Reset

Type some text and tap the button. The spam detection model will determine if the text is spam or not.

Deploy a text-classification model with TensorFlow Serving

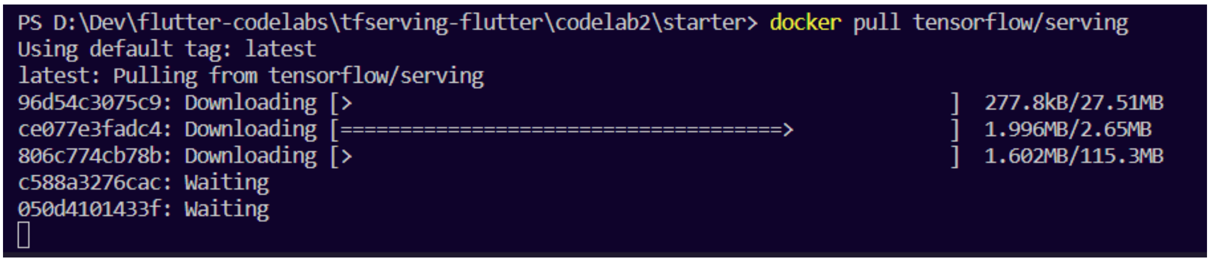
Text classification is a very common machine learning task that classifies texts into predefined categories.

In this codelab, we will deploy a pretrained model from the Train a comment-spam detection model with TensorFlow Lite Model Maker codelab with TensorFlow Serving and call the backend from our Flutter frontend to classify the input text as ***spam* or *not spam***.

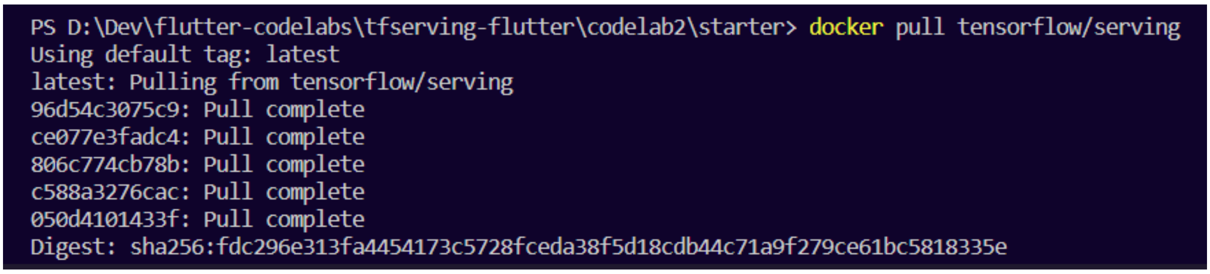
Note: A pretrained SavedModel along with the vocabulary and label files is provided in the *tfserving-flutter/codelab1/mm_spam_savedmodel* folder.

Start TensorFlow Serving

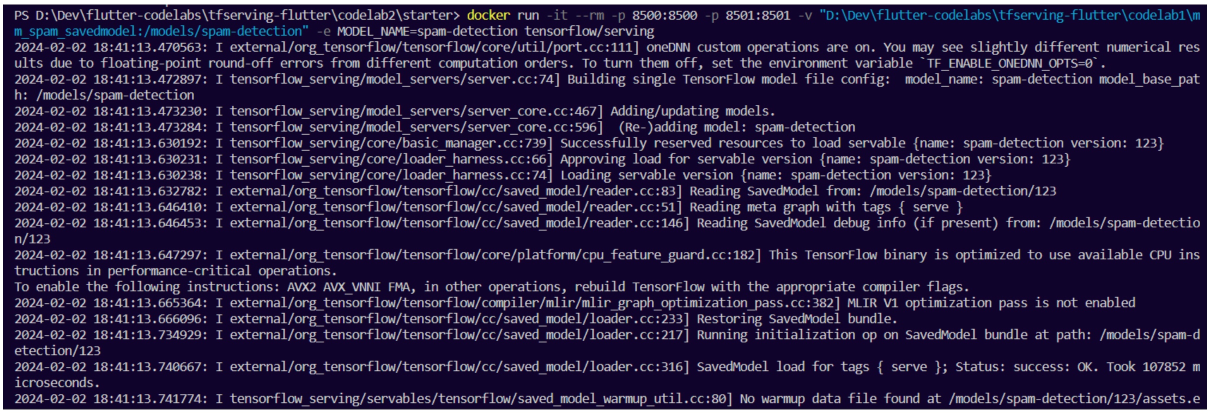
- In a terminal, start TensorFlow Serving with Docker, but replace the *PATH/TO/SAVEDMODEL* placeholder with the absolute path of the mm_spam_savedmodel folder on the computer.



```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
96d54c3075c9: Downloading [>                                                  ]  277.8kB/27.51MB
ce077e3fadc4: Downloading [====================================>              ]  1.996MB/2.65MB
806c774cb78b: Downloading [>                                                  ]  1.602MB/115.3MB
c588a3276cac: Waiting
050d4101433f: Waiting
```

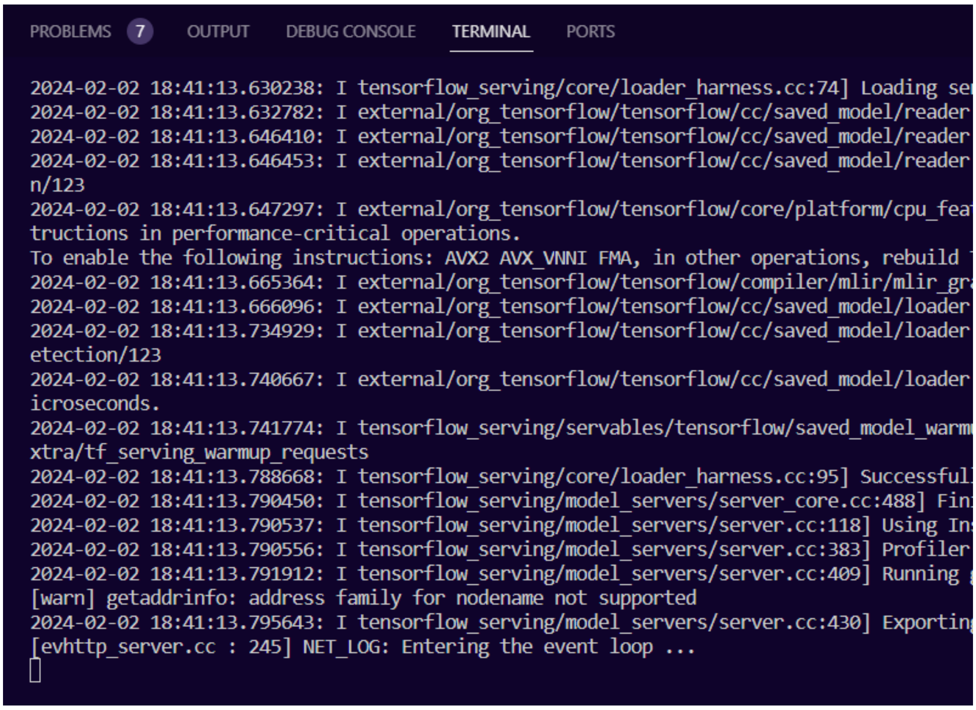


```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
96d54c3075c9: Pull complete
ce077e3fadc4: Pull complete
806c774cb78b: Pull complete
c588a3276cac: Pull complete
050d4101433f: Pull complete
Digest: sha256:fdc296e313fa4454173c5728fceda38f5d18cdb44c71a9f279ce61bc5818335e
```



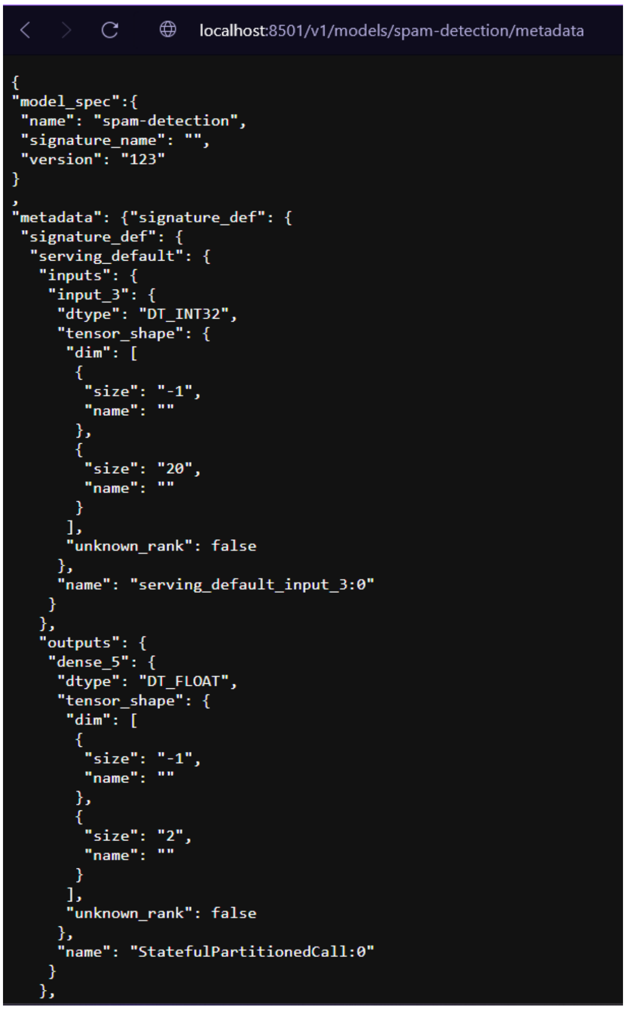
```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker run -it --rm -p 8500:8500 -p 8501:8501 -v "D:\Dev\flutter-codelabs\tfserving-flutter\codelab1\m
m_spam_savedmodel:/models/spam-detection" -e MODEL_NAME=spam-detection tensorflow/serving
2024-02-02 18:41:13.470563: I external/org_tensorflow/tensorflow/core/util/port.cc:111] oneDNN custom operations are on. You may see slightly different numerical res
ults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-02-02 18:41:13.472897: I tensorflow_serving/model_servers/server.cc:74] Building single TensorFlow model file config:  model_name: spam-detection model_base_pat
h: /models/spam-detection
2024-02-02 18:41:13.473230: I tensorflow_serving/model_servers/server_core.cc:467] Adding/updating models.
2024-02-02 18:41:13.473284: I tensorflow_serving/model_servers/server_core.cc:596]  (Re-)adding model: spam-detection
2024-02-02 18:41:13.630192: I tensorflow_serving/core/basic_manager.cc:739] Successfully reserved resources to load servable {name: spam-detection version: 123}
2024-02-02 18:41:13.630231: I tensorflow_serving/core/loader_harness.cc:66] Approving load for servable version {name: spam-detection version: 123}
2024-02-02 18:41:13.630238: I tensorflow_serving/core/loader_harness.cc:74] Loading servable version {name: spam-detection version: 123}
2024-02-02 18:41:13.632782: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:83] Reading SavedModel from: /models/spam-detection/123
2024-02-02 18:41:13.646410: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:51] Reading meta graph with tags { serve }
2024-02-02 18:41:13.646453: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:146] Reading SavedModel debug info (if present) from: /models/spam-detectio
n/123
2024-02-02 18:41:13.647297: I external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU ins
tructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-02-02 18:41:13.665364: I external/org_tensorflow/tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:382] MLIR V1 optimization pass is not enabled
2024-02-02 18:41:13.666096: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:233] Restoring SavedModel bundle.
2024-02-02 18:41:13.734929: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:217] Running initialization op on SavedModel bundle at path: /models/spam-d
etection/123
2024-02-02 18:41:13.740667: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:316] SavedModel load for tags { serve }; Status: success: OK. Took 107852 m
icroseconds.
2024-02-02 18:41:13.741774: I tensorflow_serving/servables/tensorflow/saved_model_warmup_util.cc:80] No warmup data file found at /models/spam-detection/123/assets.e
```

Docker automatically downloads the TensorFlow Serving image first, which takes a minute. Afterward, TensorFlow Serving should start. The log should look like this code snippet:



```
2024-02-02 18:41:13.630238: I tensorflow_serving/core/loader_harness.cc:74] Loading se
2024-02-02 18:41:13.632782: I external/org_tensorflow/tensorflow/cc/saved_model/reader
2024-02-02 18:41:13.646410: I external/org_tensorflow/tensorflow/cc/saved_model/reader
2024-02-02 18:41:13.646453: I external/org_tensorflow/tensorflow/cc/saved_model/reader
n/123
2024-02-02 18:41:13.647297: I external/org_tensorflow/tensorflow/core/platform/cpu_fea
tructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild
2024-02-02 18:41:13.665364: I external/org_tensorflow/tensorflow/compiler/mlir/mlir_gr
2024-02-02 18:41:13.666096: I external/org_tensorflow/tensorflow/cc/saved_model/loader
2024-02-02 18:41:13.734929: I external/org_tensorflow/tensorflow/cc/saved_model/loader
etection/123
2024-02-02 18:41:13.740667: I external/org_tensorflow/tensorflow/cc/saved_model/loader
icroseconds.
2024-02-02 18:41:13.741774: I tensorflow_serving/servables/tensorflow/saved_model_warm
xtra/tf_serving_warmup_requests
2024-02-02 18:41:13.788668: I tensorflow_serving/core/loader_harness.cc:95] Successful
2024-02-02 18:41:13.790450: I tensorflow_serving/model_servers/server_core.cc:488] Fin
2024-02-02 18:41:13.790537: I tensorflow_serving/model_servers/server.cc:118] Using In
2024-02-02 18:41:13.790556: I tensorflow_serving/model_servers/server.cc:383] Profiler
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:409] Running
[warn] getaddrinfo: address family for nodename not supported
2024-02-02 18:41:13.795643: I tensorflow_serving/model_servers/server.cc:430] Exportin
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
```

After starting the TensorFlow Serving docker image, we can visit *http://localhost:8501/v1/models/spam-detection/metadata* to inspect the details of the input and output tensors.



```
{
"model_spec":{
 "name": "spam-detection",
 "signature_name": "",
 "version": "123"
}
,
"metadata": {"signature_def": {
 "signature_def": {
  "serving_default": {
   "inputs": {
    "input_3": {
     "dtype": "DT_INT32",
     "tensor_shape": {
      "dim": [
       {
        "size": "-1",
        "name": ""
       },
       {
        "size": "20",
        "name": ""
       }
      ],
      "unknown_rank": false
     },
     "name": "serving_default_input_3:0"
    }
   },
   "outputs": {
    "dense_5": {
     "dtype": "DT_FLOAT",
     "tensor_shape": {
      "dim": [
       {
        "size": "-1",
        "name": ""
       },
       {
        "size": "2",
        "name": ""
       }
      ],
      "unknown_rank": false
     },
     "name": "StatefulPartitionedCall:0"
    }
   },
```

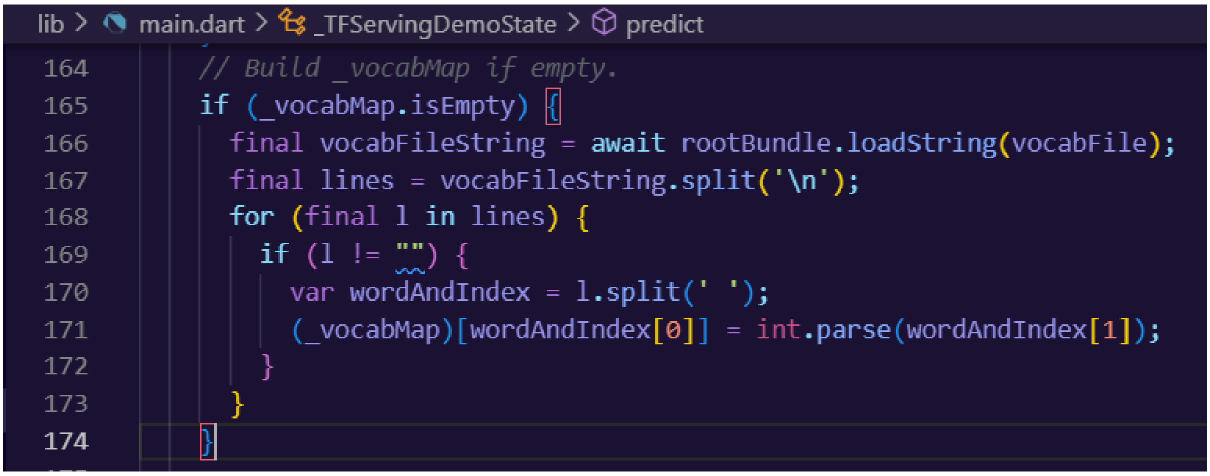
Tokenize input sentence

Once the backend ready, we need to tokenize the input sentence to be able to send client requests to TensorFlow Serving.

If we inspect the input tensor of the model, we can see that it expects a list of 20 integer numbers instead of raw strings.

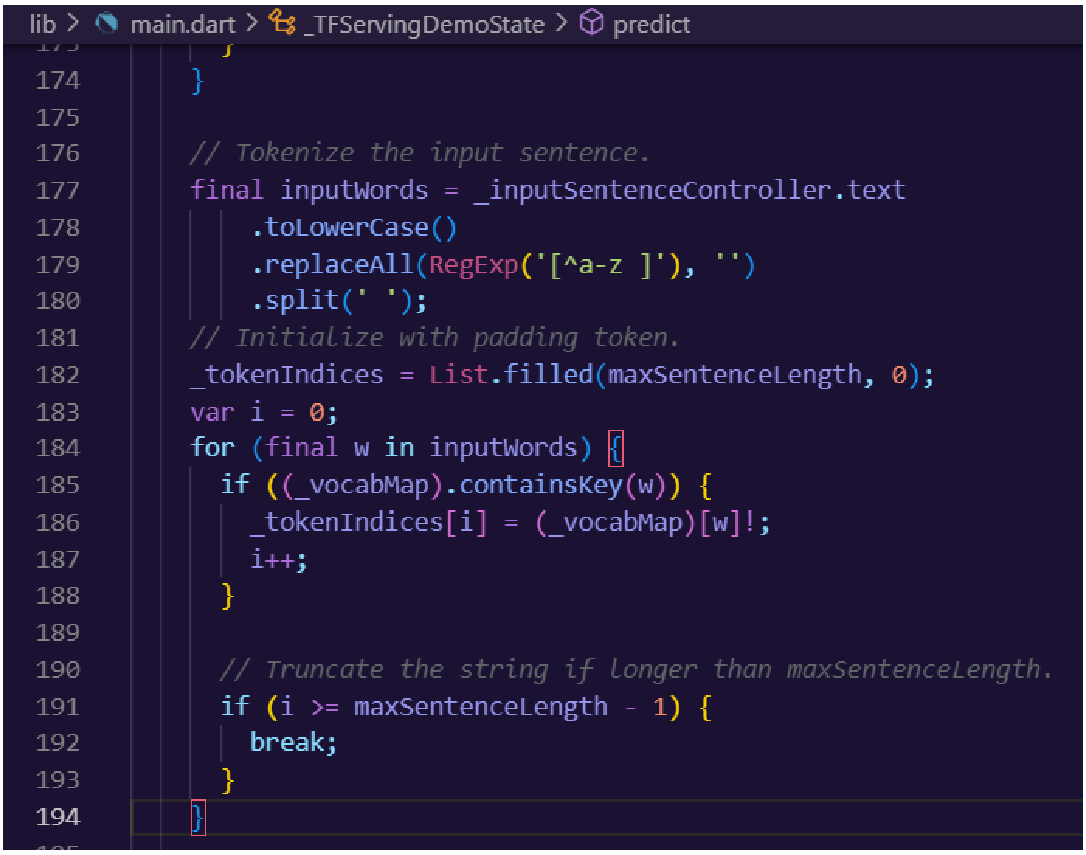
Tokenization is when we map the individual words typed in the app to a list of integers based on a vocabulary dictionary before sending them to the backend for classification.

1. In the lib/main.dart file, add this code to the predict() method to build the _vocabMap vocabulary dictionary.



```
// Build _vocabMap if empty.
if (_vocabMap.isEmpty) {
  final vocabFileString = await rootBundle.loadString(vocabFile);
  final lines = vocabFileString.split('\n');
  for (final l in lines) {
    if (l != "") {
      var wordAndIndex = l.split(' ');
      (_vocabMap)[wordAndIndex[0]] = int.parse(wordAndIndex[1]);
    }
  }
}
```

Immediately after the previous code snippet, add this code to implement tokenization:



```
}

// Tokenize the input sentence.
final inputWords = _inputSentenceController.text
    .toLowerCase()
    .replaceAll(RegExp('[^a-z ]'), '')
    .split(' ');
// Initialize with padding token.
_tokenIndices = List.filled(maxSentenceLength, 0);
var i = 0;
for (final w in inputWords) {
  if ((_vocabMap).containsKey(w)) {
    _tokenIndices[i] = (_vocabMap)[w]!;
    i++;
  }

  // Truncate the string if longer than maxSentenceLength.
  if (i >= maxSentenceLength - 1) {
    break;
  }
}
```

This code lowercases the sentence string, removes non-alphabet characters, and maps the words to 20 integer indices based on the vocabulary table.

Connect the Flutter app with TensorFlow Serving through REST

There are two ways to send requests to TensorFlow Serving:

- REST
- gRPC

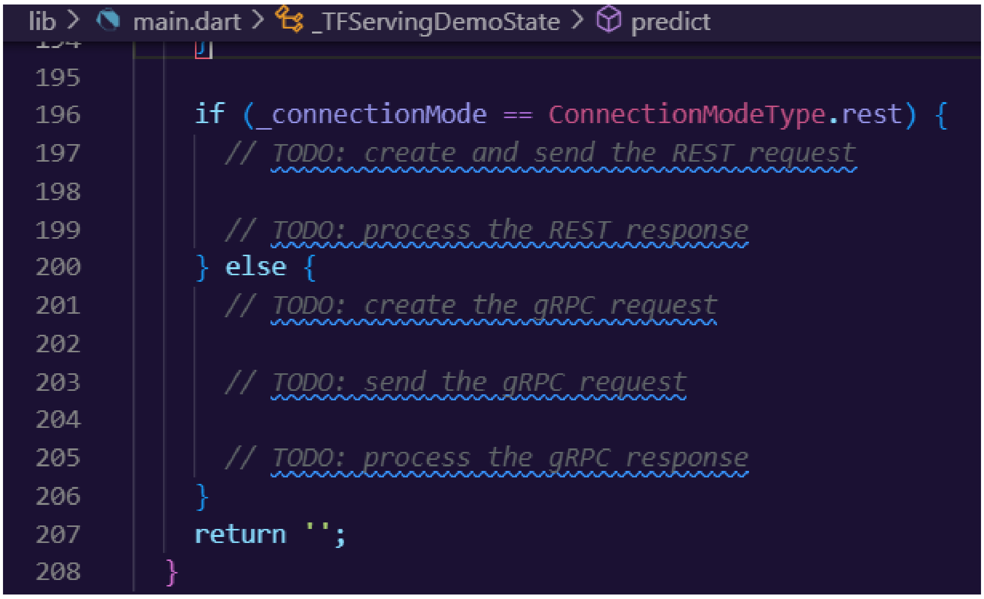
Send requests and receive responses through REST

There are three simple steps to send requests and receive responses through REST:

1. Create the REST request.
2. Send the REST request to TensorFlow Serving.
3. Extract the predicted result from the REST response and render the UI.

Create and send the REST request to TensorFlow Serving

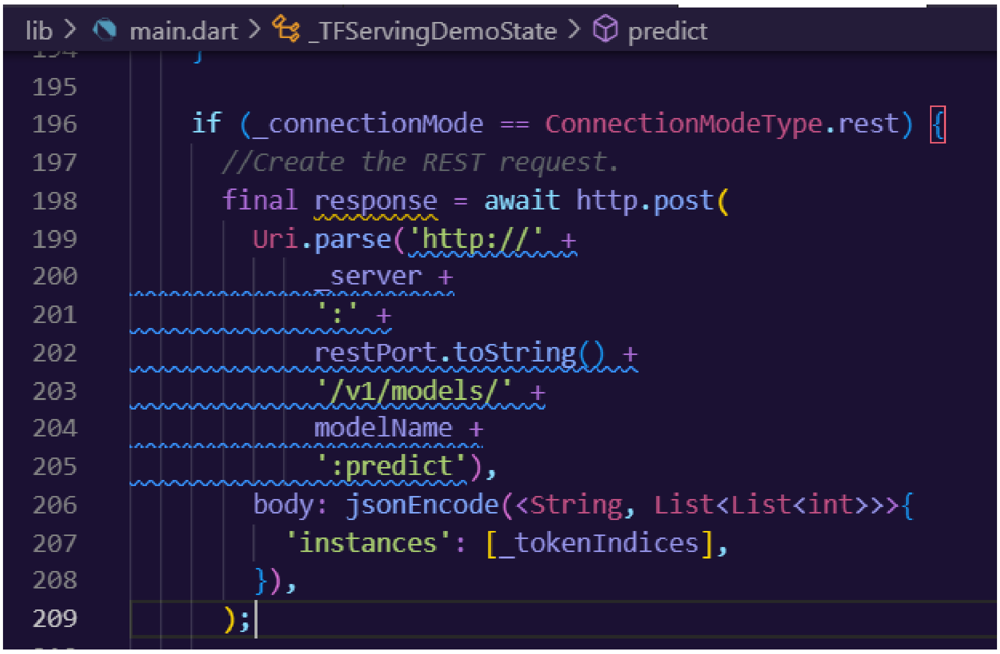
1. Right now, the predict() function doesn't send the REST request to TensorFlow Serving. You need to implement the REST branch to create a REST request:



lib > main.dart > _TFServingDemoState > predict

```
    if (_connectionMode == ConnectionModeType.rest) {
      // TODO: create and send the REST request

      // TODO: process the REST response
    } else {
      // TODO: create the gRPC request

      // TODO: send the gRPC request

      // TODO: process the gRPC response
    }
    return '';
  }
```

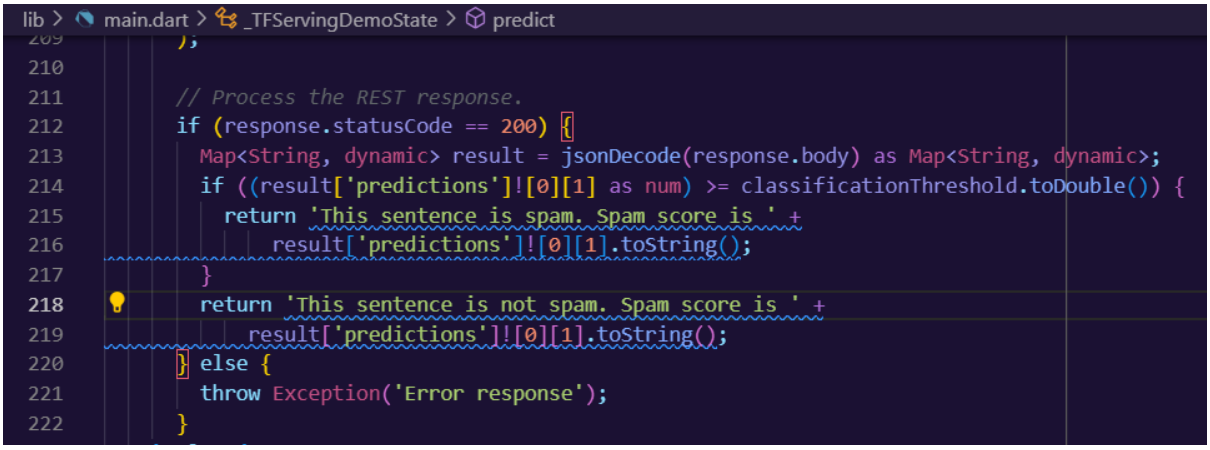
2. Add this code to the REST branch:



```dart
    if (_connectionMode == ConnectionModeType.rest) {
      //Create the REST request.
      final response = await http.post(
        Uri.parse('http://' +
            _server +
            ':' +
            restPort.toString() +
            '/v1/models/' +
            modelName +
            ':predict'),
        body: jsonEncode(<String, List<List<int>>>{
          'instances': [_tokenIndices],
        }),
      );
```

Process the REST response from TensorFlow Serving

- Add this code right after the previous code snippet to handle the REST response:

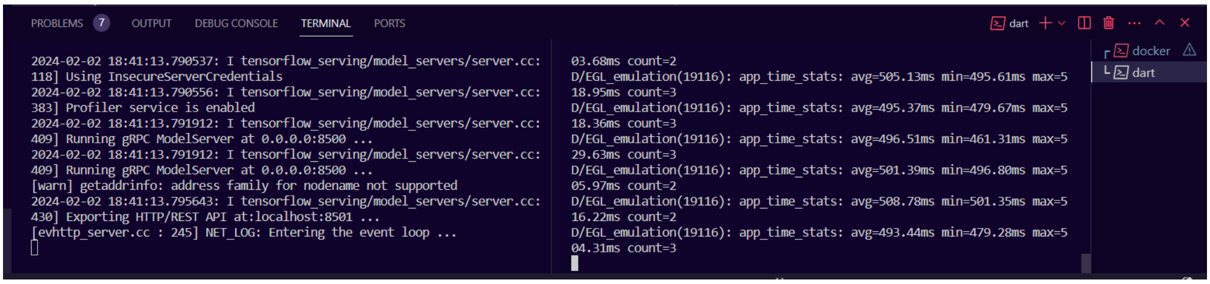


```dart
      // Process the REST response.
      if (response.statusCode == 200) {
        Map<String, dynamic> result = jsonDecode(response.body) as Map<String, dynamic>;
        if ((result['predictions']![0][1] as num) >= classificationThreshold.toDouble()) {
          return 'This sentence is spam. Spam score is ' +
              result['predictions']![0][1].toString();
        }
        return 'This sentence is not spam. Spam score is ' +
            result['predictions']![0][1].toString();
      } else {
        throw Exception('Error response');
      }
```

The postprocessing code extracts the probability that the input sentence is a spam message from the response and displays the classification result in the UI.

Run it

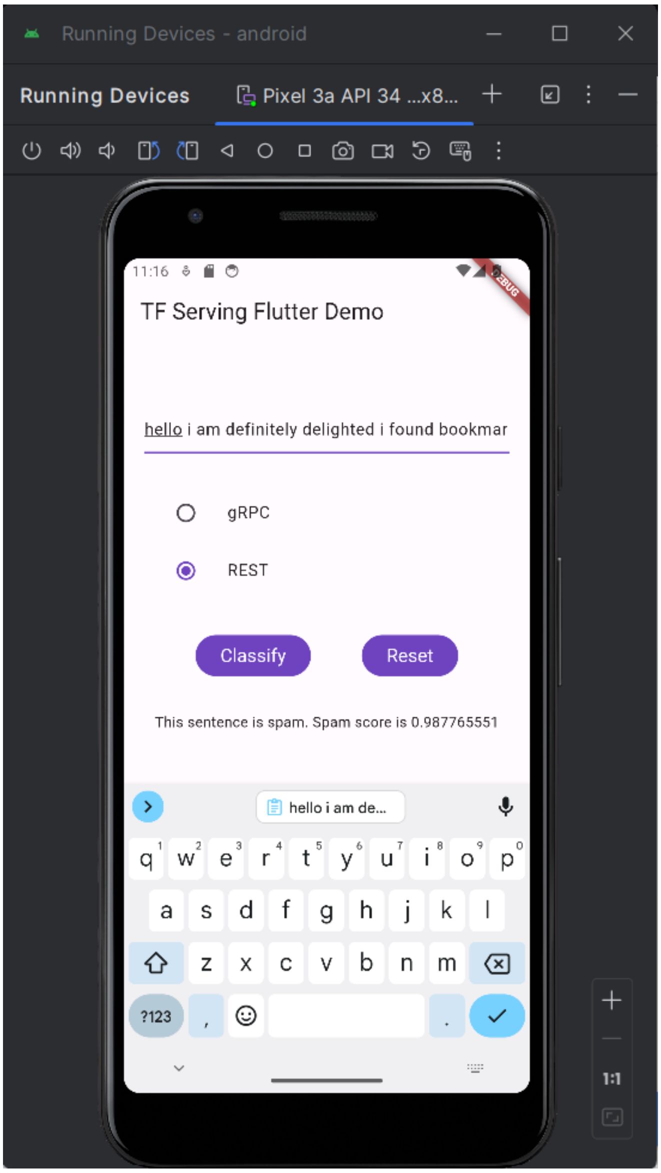
1. Type flutter run in a new terminal. Make sure docker is running on the same time.



```
2024-02-02 18:41:13.790537: I tensorflow_serving/model_servers/server.cc:
118] Using InsecureServerCredentials
2024-02-02 18:41:13.790556: I tensorflow_serving/model_servers/server.cc:
383] Profiler service is enabled
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:
409] Running gRPC ModelServer at 0.0.0.0:8500 ...
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:
409] Running gRPC ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2024-02-02 18:41:13.795643: I tensorflow_serving/model_servers/server.cc:
430] Exporting HTTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
```

```
03.68ms count=2
D/EGL_emulation(19116): app_time_stats: avg=505.13ms min=495.61ms max=5
18.95ms count=3
D/EGL_emulation(19116): app_time_stats: avg=495.37ms min=479.67ms max=5
18.36ms count=3
D/EGL_emulation(19116): app_time_stats: avg=496.51ms min=461.31ms max=5
29.63ms count=3
D/EGL_emulation(19116): app_time_stats: avg=501.39ms min=496.80ms max=5
05.97ms count=2
D/EGL_emulation(19116): app_time_stats: avg=508.78ms min=501.35ms max=5
16.22ms count=2
D/EGL_emulation(19116): app_time_stats: avg=493.44ms min=479.28ms max=5
04.31ms count=3
```

2. Enter some text and then select **REST > Classify**.



TF Serving Flutter Demo

hello i am definitely delighted i found bookmar

○ gRPC

● REST

This sentence is spam. Spam score is 0.987765551

Note: Depending on your training procedure and the input sentence, the model may predict a different spam score. For example, "hello i am definitely delighted i found bookmarking site www winbig com very profitable" is an example sentence that generally receives a high spam score.

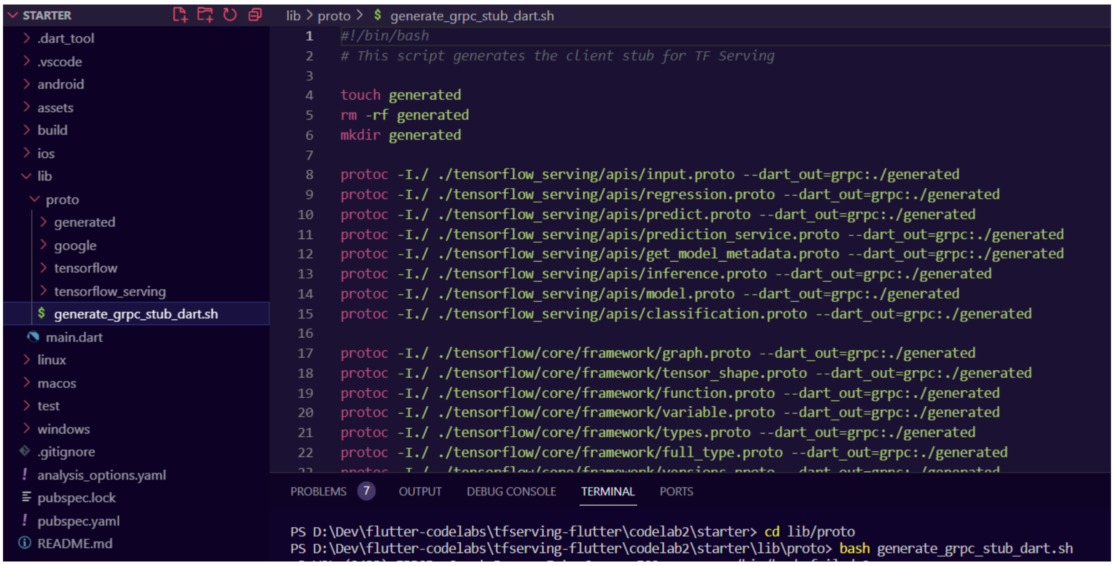
Connect the Flutter app with TensorFlow Serving through gRPC

In addition to REST, TensorFlow Serving also supports gRPC.

gRPC is a modern, open source, high-performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in, and across, data centers with pluggable support for load balancing, tracing, health checking, and authentication. It's been observed that gRPC is more performant than REST in practice.

- In the terminal, navigate to the starter/lib/proto/ folder and generate the stub:

bash generate_grpc_stub_dart.sh

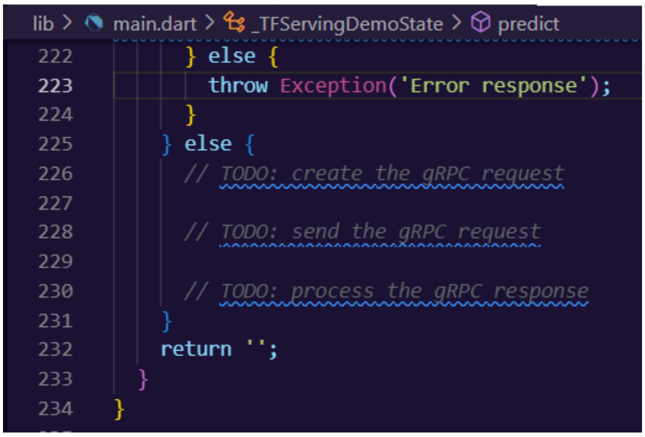


```bash
#!/bin/bash
# This script generates the client stub for TF Serving

touch generated
rm -rf generated
mkdir generated

protoc -I./ ./tensorflow_serving/apis/input.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/regression.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/predict.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/prediction_service.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/get_model_metadata.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/inference.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/model.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/classification.proto --dart_out=grpc:./generated

protoc -I./ ./tensorflow/core/framework/graph.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/tensor_shape.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/function.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/variable.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/types.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/full_type.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/versions.proto --dart_out=grpc:./generated
```

```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> cd lib/proto
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter\lib\proto> bash generate_grpc_stub_dart.sh
```

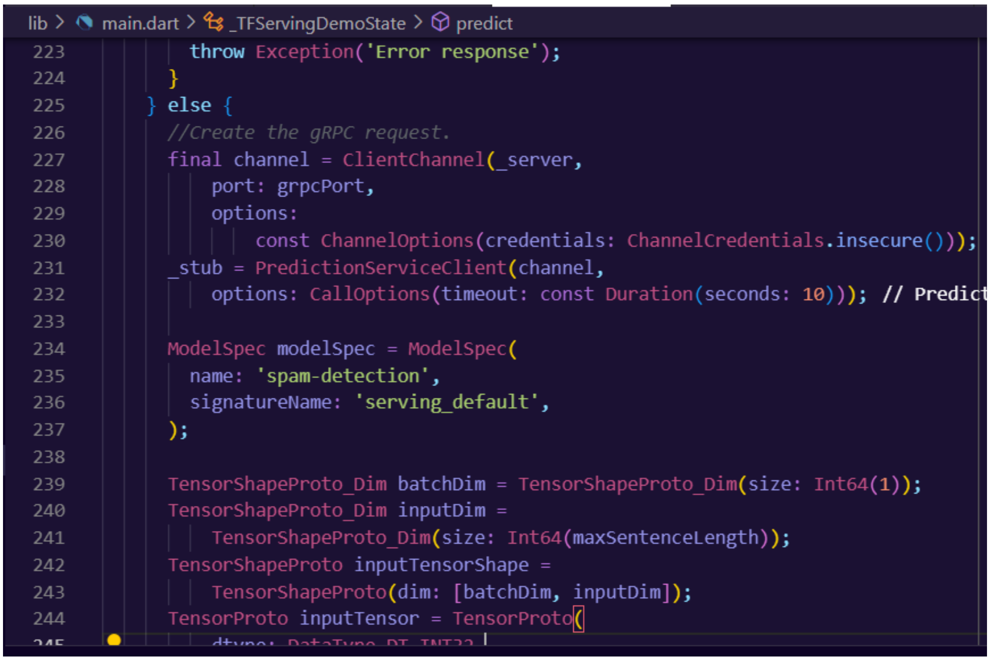
Create the gRPC request

Similar to the REST request, you create the gRPC request in the gRPC branch.



```dart
      } else {
        throw Exception('Error response');
      }
    } else {
      // TODO: create the gRPC request

      // TODO: send the gRPC request

      // TODO: process the gRPC response
    }
    return '';
  }
}
```

Add this code to create the gRPC request:

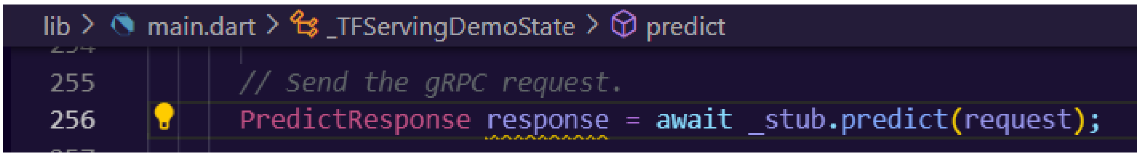


```dart
        throw Exception('Error response');
      }
    } else {
      //Create the gRPC request.
      final channel = ClientChannel(_server,
          port: grpcPort,
          options:
              const ChannelOptions(credentials: ChannelCredentials.insecure()));
      _stub = PredictionServiceClient(channel,
          options: CallOptions(timeout: const Duration(seconds: 10))); // Predict

      ModelSpec modelSpec = ModelSpec(
        name: 'spam-detection',
        signatureName: 'serving_default',
      );

      TensorShapeProto_Dim batchDim = TensorShapeProto_Dim(size: Int64(1));
      TensorShapeProto_Dim inputDim =
          TensorShapeProto_Dim(size: Int64(maxSentenceLength));
      TensorShapeProto inputTensorShape =
          TensorShapeProto(dim: [batchDim, inputDim]);
      TensorProto inputTensor = TensorProto(
          dtype: DataType.DT_INT32
```

The input and output tensor names could differ from model to model, even if the model architectures are the same. Make sure to update them if you train your own model.

Send the gRPC request to TensorFlow Serving

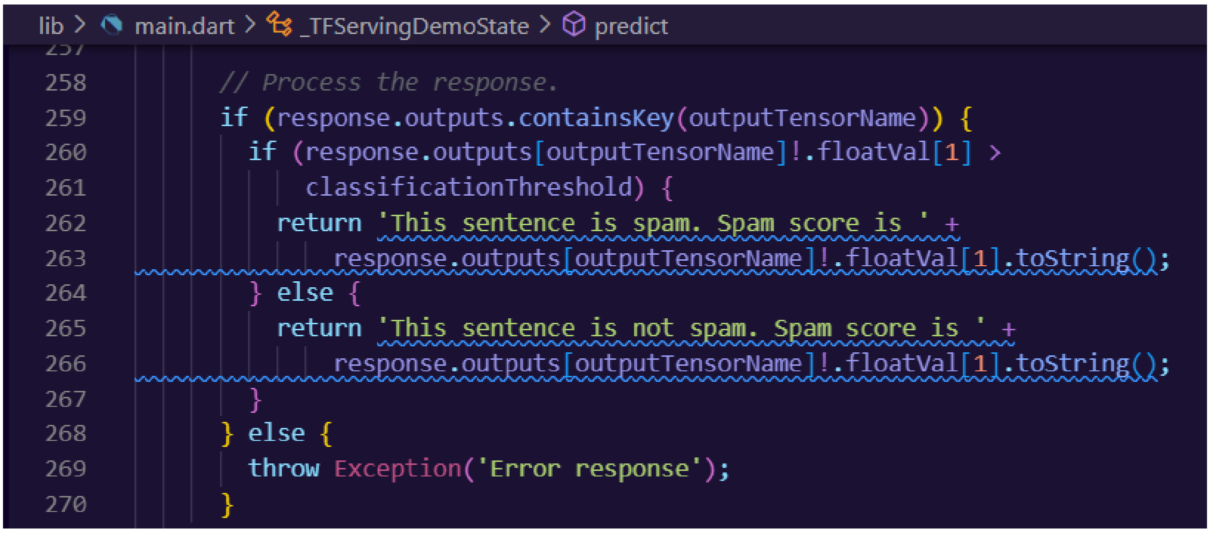
Add this code after the previous code snippet to send the gRPC request to TensorFlow Serving:



```
    // Send the gRPC request.
    PredictResponse response = await _stub.predict(request);
```

Process the gRPC response from TensorFlow Serving

Add this code after the previous code snippet to implement the callback functions to handle the response:

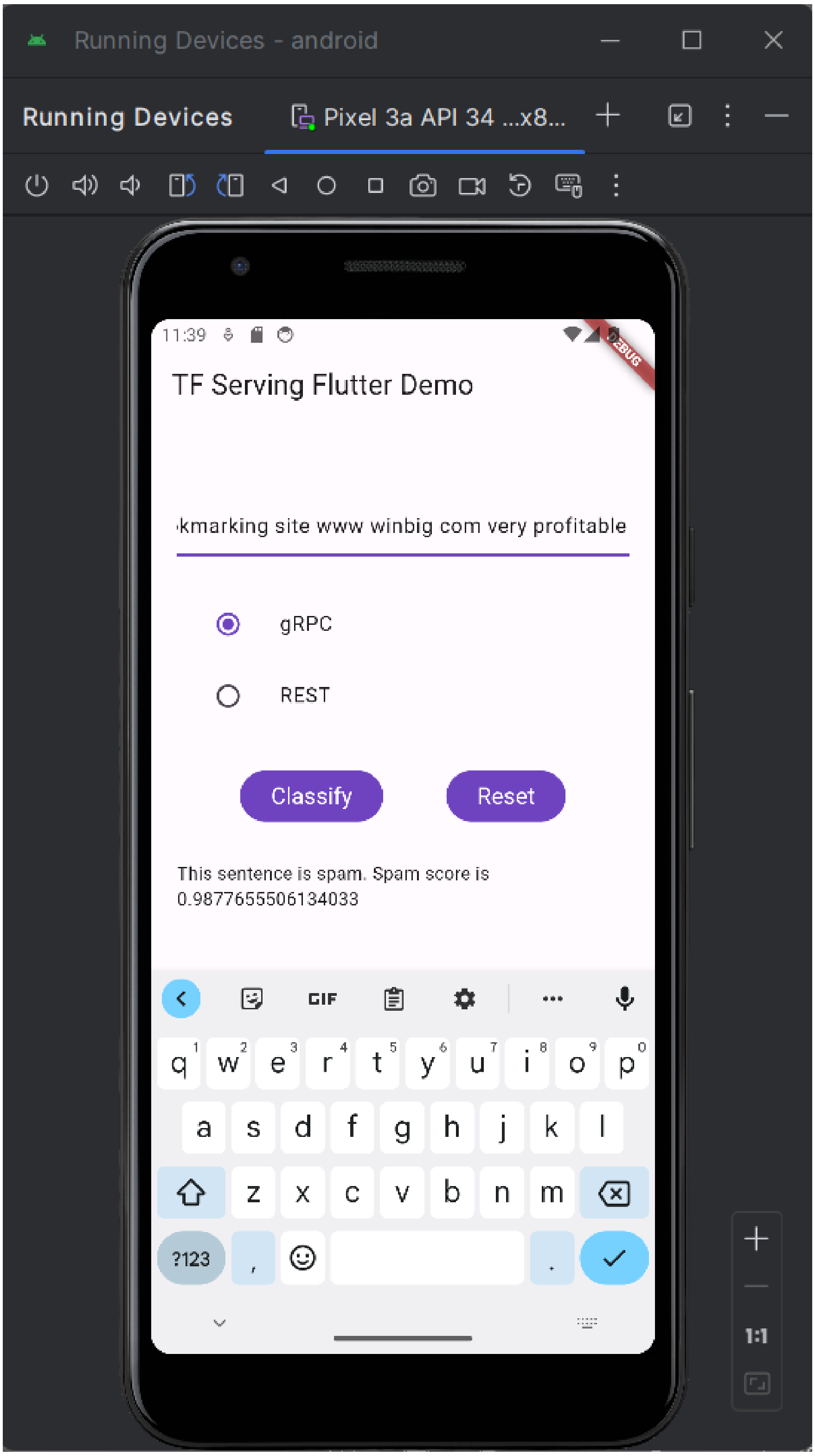


```
    // Process the response.
    if (response.outputs.containsKey(outputTensorName)) {
      if (response.outputs[outputTensorName]!.floatVal[1] >
          classificationThreshold) {
        return 'This sentence is spam. Spam score is ' +
            response.outputs[outputTensorName]!.floatVal[1].toString();
      } else {
        return 'This sentence is not spam. Spam score is ' +
            response.outputs[outputTensorName]!.floatVal[1].toString();
      }
    } else {
      throw Exception('Error response');
    }
```

Now the postprocessing code extracts the classification result from the response and displays it in the UI.

Run it

1. Type flutter run in a new terminal. Make sure docker is running on the same time.
2. Enter some text and then select **gRPC > Classify**.



TF Serving Flutter Demo

kmarking site www winbig com very profitable

● gRPC

○ REST

Classify

Reset

This sentence is spam. Spam score is 0.9877655506134033

Conclusion

This lab helped us discover new features of TensorFlow. It made the text-classification to a quite simple process.