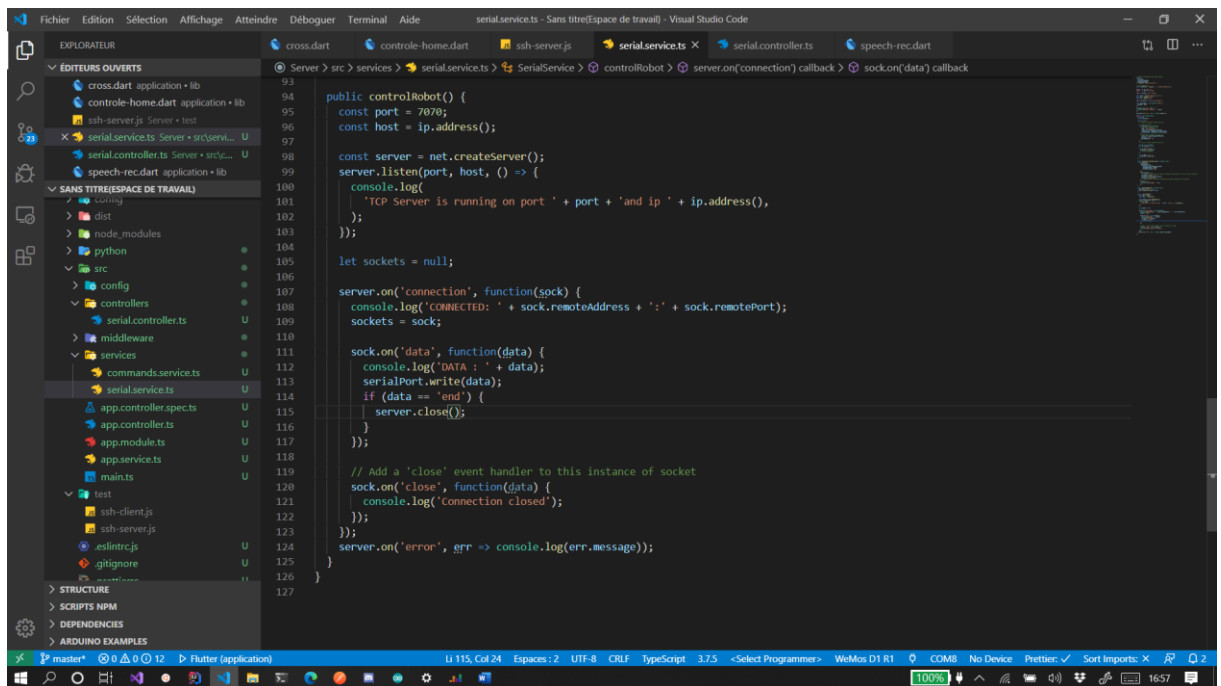


# Rapport séance 02/03/2020

- Durant cette séance et chez moi, je me suis penché sur le contrôle à distance du robot. Pour contrôler à distance, je vais utiliser un Protocol TCP à la place du Bluetooth trop instable. Lorsque je me lance le contrôle automatique du robot depuis mon application. Elle envoie une requête à mon serveur principal qui va ensuite lancer un serveur TCP secondaire pour pouvoir communiquer en direct avec le téléphone. Ensuite lorsque le téléphone envoie des données de déplacement, mon serveur secondaire les récupère et les envoie par la connexion série à la carte Arduino qui traite ensuite les données et déplace le robot.



The screenshot shows the Visual Studio Code editor with a Dart file named `serial.service.ts`. The code defines a `controlRobot()` function that sets up a TCP server. The server listens on a specified port and host. When a connection is established, it logs the connection details and sets up a handler for incoming data. The data handler logs the received data and writes it to a serial port. The server also has a 'close' event handler and an error handler.

```
public controlRobot() {
  const port = 7070;
  const host = ip.address();

  const server = net.createServer();
  server.listen(port, host, () => {
    console.log(
      'TCP Server is running on port ' + port + ' and ip ' + ip.address(),
    );
  });

  let sockets = null;

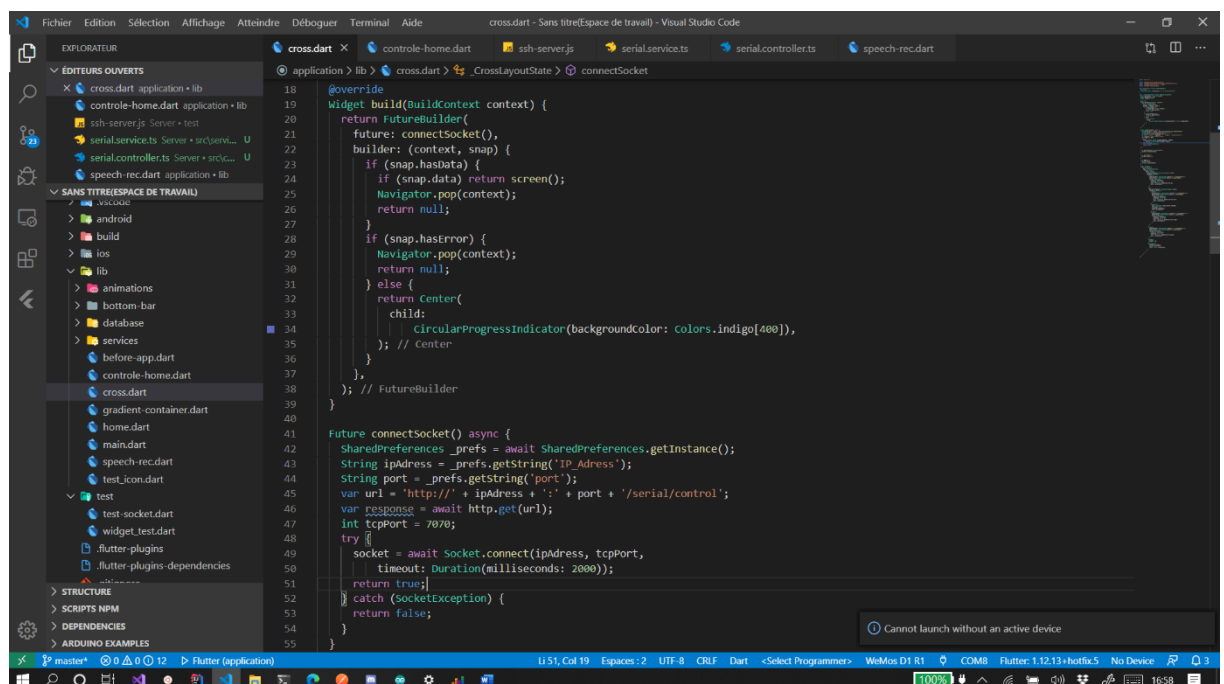
  server.on('connection', function(sock) {
    console.log('CONNECTED: ' + sock.remoteAddress + ' : ' + sock.remotePort);
    sockets = sock;

    sock.on('data', function(data) {
      console.log('DATA: ' + data);
      serialPort.write(data);
      if (data == 'end') {
        server.close();
      }
    });

    // Add a 'close' event handler to this instance of socket
    sock.on('close', function(data) {
      console.log('Connection closed');
    });
  });

  server.on('error', err => console.log(err.message));
}
```

Figure 1 : code qui lance le serveur TCP et envoi les données à la carte Arduino



The screenshot shows the Visual Studio Code editor with a Dart file named `cross.dart`. The code defines a `@override` widget `build(BuildContext context)` that returns a `FutureBuilder`. The `future` property is set to `connectSocket()`. The `builder` property is a function that takes `context` and `snapshot` as arguments. It checks if the snapshot has data or an error and updates the UI accordingly. The `connectSocket()` function is an async function that uses `SharedPreferences` to get the IP address and port, and then attempts to connect to the server.

```
@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: connectSocket(),
    builder: (context, snap) {
      if (snap.hasData) {
        if (snap.data) return screen();
        Navigator.pop(context);
        return null;
      }
      if (snap.hasError) {
        Navigator.pop(context);
        return null;
      }
      else {
        return Center(
          child:
            CircularProgressIndicator(backgroundColor: colors.indigo[400]),
        ); // Center
      }
    },
  ); // FutureBuilder
}

Future connectSocket() async {
  SharedPreferences _prefs = await SharedPreferences.getInstance();
  String ipAddress = _prefs.getString('IP_Address');
  String port = _prefs.getString('port');
  var url = 'http://' + ipAddress + ':' + port + '/serial/control';
  var response = await http.get(url);
  int tcpPort = 7070;
  try {
    socket = await Socket.connect(ipAddress, tcpPort,
      timeout: Duration(milliseconds: 2000));
    return true;
  } catch (socketException) {
    return false;
  }
}
```

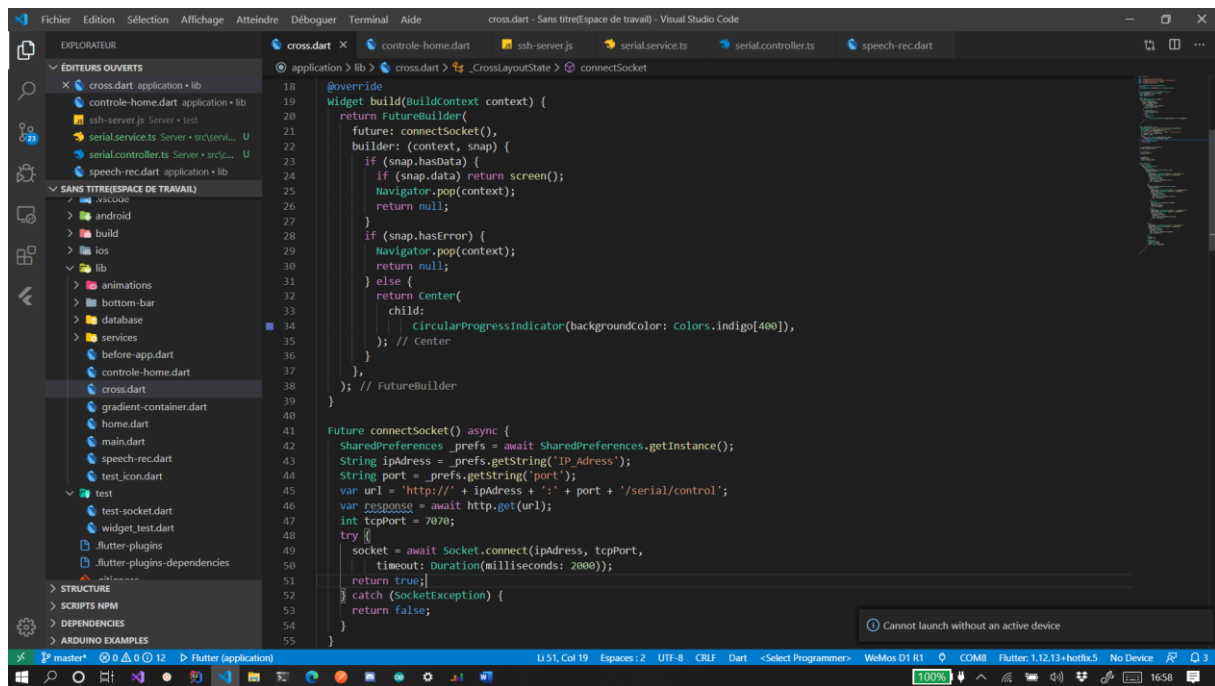


Figure 2 et Figure 3 : code de l'application mobile de la partie controle du robot