7° INFORMÁTICA 2023

Informe

Proyecto Fiscalberry

Colegio Florentino Ameghino

Empresa: Paxapos

Profesor: Tomas Arce

ÍNDICE

¿De qué trata el proyecto?

Este proyecto es de código abierto y está hecho por la empresa Paxapos, la cual nos brindó diferentes opciones por grupos para poder llevar a cabo nuestras pasantías.
Se debe desarrollar una aplicación o página web la cual envíe un documento hacia el servidor de Web Socket que abre Fiscalberry para que este pueda traducirlo y convertirlo en una factura electrónica impresa en papel.
Está desarrollado para poder conectarse y utilizar 2 tipos de impresoras: Fiscales y Receipt. Está pensado para cualquier usuario que desee enviar a imprimir, un recibo o una factura, mediante la misma red pero también para que funcione con cualquier impresora, marca o modelo.

Herramientas que utilizamos

Nuestro trabajo es traducir algunos de los archivos ya creados al lenguaje DART, ya que, como dijimos anteriormente, este proyecto está hecho en el lenguaje Python y se desea actualizar.

- Conocimiento del lenguaje Python.
- Conocimiento del lenguaje Dart.
- Github en donde tenemos los archivos base.
- Impresoras enviadas por PAXAPOS.

Primera parte

Traducciones

EscPConstants.dart
Archivo con constantes, algunas se utilizaran en la función "printer.set()", y otras para formato de texto individual "individual text formatting"

Este archivo está relacionado a las impresoras térmicas que el proyecto utiliza, el cual se encargará de controlar y darle formato al texto a imprimir.
Empieza importando variables de otro archivo llamado "escpos.constants".
Luego declara distintas constantes que serán utilizadas más adelante en otros archivos. Algunas serán usadas en "printer.set()":

- Las primeras 3 constantes son para definir el alineamiento del texto (CENTER, LEFT, RIGHT).
- Luego le siguen 2 constantes para definir el tipo de la fuente (A, B).
- Las 4 siguientes definen el estilo (NORMAL, BOLD, UNDERLINED, BOLDUNDERLINED). y otras en formatos de textos individuales:
- La mayoría de constantes se utilizarán para controlar las constantes anteriores (BOLD and UNDERLINED).
- La última constante se utilizará para cortar el papel (CUT).

```dart
import 'package:escpos.constants.dart/EscPConstants.dart';

// For printer.set()
const String CENTER = 'CENTER';
const String LEFT = 'LEFT';
const String RIGHT = 'RIGHT';

const String FONT_A = 'A';
const String FONT_B = 'B';

const String NORMAL = 'NORMAL';
const String BOLD = 'B';
const String UNDERLINED = 'U';
const String BOLD_UNDERLINED = 'BU';

// For individual text formatting
const String UNDERLINED_ON = '\x1B\x2D\x01';
const String UNDERLINED2_ON = '\x1B\x2D\x02';
const String UNDERLINED_OFF = '\x1B\x2D\x00';

const String BOLD_ON = '\x1B\x45\x01';
const String BOLD_OFF = '\x1B\x45\x00';

const String PARTIAL_CUT = 'PART';

```

FiscalberryComandosDart.dart

Este archivo es parte del sistema de manejo de las impresoras. Proporciona funcionalidades para enviar comandos y para manejar las excepciones de la conexión.

Empieza importando librerías y variables de otro archivo llamado "**ComandoInterface.dart**". Luego declara la clase **PrinterException** que utilizará aparte atributos del archivo **Exception**.

La siguiente clase, **FiscalberryComandos**, utilizará parte del archivo ComandoInterface para poder funcionar. Primero se necesita indicar el módulo de traducción que utilizará la clase (**traductorModule**), además de una constante estática (**DEAFULT_DRIVER**) que se representa el controlador predeterminado.

_sendCommand se encarga de enviar un comando a la impresora por medio de la conexión (**cthis.conector.sendCommand(comando, skipStatusErrors**).

Por último, en caso de que existan excepciones (**PrinterException**), se registrará y mostrará el error (**CommandException**) utilizando una de las bibliotecas anteriormente exportadas.

```dart
import 'package:http/http.dart' as http;
import 'package:logging/logging.dart';
import 'ComandoInterface.dart';


class PrinterException implements Exception {
}

class FiscalberryComandos implements ComandoInterface {
  String traductorModule = "Traductores.TraductorFiscalberry";
  static const String DEFAULT_DRIVER = "Fiscalberry";

  Future<dynamic> _sendCommand(String comando, {bool skipStatusErrors = false}) async {
    try {
      var ret = await this.conector.sendCommand(comando, skipStatusErrors);
      return ret;
    } on PrinterException catch (e) {
      Logger("ProxyComandos").error("PrinterException: ${e.toString()}");
      throw ComandoException("Error de la impresora: ${e.toString()}.\nComando enviado: $comando");
    }
  }
}
```

EscPComandos.dart

Este archivo es el que contiene las funcionalidades específicas de la impresora, como por ejemplo el formato en que saldrá el texto, los datos que se escribirán, en qué momento se imprimirá el ticket, en qué momento se cortará el ticket, etc.

Comienza importando comandos de códigos anteriores, como así también librerías que adaptamos de Python a Dart, ya que varían sus nombres, su funcionalidad e incluso la necesidad de importar.

Luego tenemos una función definida que se llama **floatToString**. Ésta toma un valor numérico o cadena, lo formatea como una cadena con dos decimales, y luego elimina los ceros finales y el punto decimal innecesario. Esto para presentar números en un formato más limpio.

La segunda función definida se llama **pad**. La misma tiene como objetivo formatear una cadena de texto para que tenga un tamaño específico, agregando caracteres de relleno a la izquierda o a la derecha según la indicación de alineación.

```dart
import 'package:logging/logging.dart';
import 'package:ComandoInterface/ComandoInterface.dart';
import 'package:ComandoInterface/ComandoException.dart';
import 'package:intl/intl.dart';

String floatToString(dynamic inputValue) {
  if (inputValue is! double) {
    inputValue = double.parse(inputValue.toString());
  }
  return inputValue.toStringAsFixed(2).replaceAll(RegExp(r"0*$"), "").replaceAll(RegExp(r"\.$"), "");
}

void main() {
  dynamic value = 123.456;
  String result = floatToString(value);
  print('Resultado: $result');
}

String pad(dynamic texto, int size, String relleno, {String float = 'l'}) {
  String text = texto.toString();
  if (float.toLowerCase() == 'l') {
    return text.substring(0, size).padRight(size, relleno);
  } else {
    return text.substring(0, size).padLeft(size, relleno);
  }
}

void main() {
  String resultado1 = pad("Hola", 10, "-", float: "l");
  String resultado2 = pad("Mundo", 10, "*", float: "r");

  print('Resultado 1: $resultado1');
  print('Resultado 2: $resultado2');
}
```

Segunda parte

En esta parte del año finalizamos el proyecto creamos una réplica del proyecto en app. Creamos una app por medio de flutter para poder imprimir los tickets por medio de un dispositivo móvil y una impresora térmica.
El código está hecho con la estructura del archivo flutter el cual cuenta con 3 archivos:

main.dart

Este código representa la entrada principal (main) de una aplicación Flutter y define la estructura básica de la aplicación.

```dart
import 'package:flutter/material.dart';

import 'home_page.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Paxapos',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: HomePage(),
    );
  }
}

```

- Se importan las bibliotecas necesarias para construir la interfaz de usuario con Flutter. En particular, se importa material.dart para los widgets de Material Design y **home_page.dart**, que probablemente contiene la lógica y la interfaz de usuario de la página principal de la aplicación.

- La función main es la entrada principal de la aplicación Flutter. En este caso, se llama a **runApp** con una instancia de **MyApp**, que es el widget principal de la aplicación.
- Se define una clase **MyApp** que extiende de **StatelessWidget**. Esta clase representa el widget principal de la aplicación y es un widget sin estado, lo que significa que su representación visual no cambia durante el ciclo de vida de la aplicación.
- Se implementa el método build que devuelve la interfaz de usuario de la aplicación. En este caso, se utiliza un widget **MaterialApp** que configura aspectos globales de la aplicación, como el título y el tema visual. El widget principal de la aplicación se establece como una instancia de **HomePage**, que es la página principal de la aplicación.
- **title:** Es el título de la aplicación que se puede mostrar en la barra de aplicaciones o en otros lugares según el diseño de la plataforma.
- **theme:** Define el tema visual de la aplicación. En este caso, se utiliza un tema con un tono de color principal azul (**primarySwatch: Colors.blue**).
- **home:** Especifica el widget que se mostrará como la página principal de la aplicación. En este caso, es una instancia de la clase HomePage.

home_page.dart

Este código Flutter representa una pantalla de la interfaz de usuario para una aplicación que genera un ticket de compra.

```dart
import 'package:flutter/material.dart';
import 'package:flutter_bluetooth/print_page.dart';
import 'package:intl/intl.dart';

class HomePage extends StatelessWidget {
  final List<Map<String, dynamic>> data = [
    {'title': 'Agua Gasificada ', 'price': 165, 'qty': 3}
  ];

  final f = NumberFormat("\$###,###.00", "en_ARS");

  @override
  Widget build(BuildContext context) {
    int _total = 0;
    _total = data.map((e) => e['price'] * e['qty']).reduce(
          (value, element) => value + element,
        );

    return Scaffold(
      appBar: AppBar(
        title: Text('Paxapos ticket'),
        backgroundColor: Colors.redAccent,
      ),
      body: Column(
        children: [
          Expanded(
```

- Se importan las bibliotecas necesarias para construir la interfaz de usuario, incluyendo **flutter/material.dart** para los widgets de Flutter, **flutter_bluetooth/print_page.dart** para la página de impresión (que probablemente contenga la lógica de impresión), y **intl.dart** para formatear números.
- Se define una clase HomePage que extiende de **StatelessWidget**, lo que significa que esta clase no tiene estado mutable.
- Se define una lista **data** que contiene información sobre los elementos que se mostrarán en el ticket. Cada elemento es un mapa con claves como 'title', 'price' y 'qty' para el título, el precio y la cantidad respectivamente.
- Se crea una instancia de **NumberFormat** para formatear los números en formato de moneda argentina.
- Se implementa el método **build** que devuelve la interfaz de usuario de la página.
- Se calcula el total sumando el producto de 'price' y 'qty' para cada elemento en la lista **data**.
- Se crea la estructura básica de la pantalla utilizando un **Scaffold** con una **AppBar** y un **Column** que contiene los elementos de la pantalla.
- Se utiliza un **ListView.builder** para mostrar la lista de elementos en un formato de lista. Cada elemento se representa como un **ListTile** que muestra el título, el subtotal y el total para cada elemento.
- Se muestra la sección del total junto con un botón de impresión. El total se muestra con el formato de moneda argentina. Al hacer clic en el botón de impresión, se navega a la página de impresión (**PrintPage**) pasando los datos de la lista. El botón está estilizado con colores y un ícono de impresión.

print_page.dart

Este código Flutter representa una página de impresión en la que se puede seleccionar una impresora Bluetooth y enviar un ticket de compra para ser impreso.

```dart
import 'package:bluetooth_print/bluetooth_print.dart';
import 'package:bluetooth_print/bluetooth_print_model.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';

class PrintPage extends StatefulWidget {
  final List<Map<String, dynamic>> data;
  PrintPage(this.data);

  @override
  _PrintPageState createState() => _PrintPageState();
}

class _PrintPageState extends State<PrintPage> {
  BluetoothPrint bluetoothPrint = BluetoothPrint.instance;
  List<BluetoothDevice> _devices = [];
  String _devicesMsg = "";
  final f = NumberFormat("\$###,###.00", "en_ARS");

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) => {initPrinter()});
  }

  Future<void> initPrinter() async {
    bluetoothPrint.startScan(timeout: Duration(seconds: 2));
```

- Se importan las bibliotecas necesarias para interactuar con la impresión Bluetooth, construir la interfaz de usuario con Flutter y formatear números.
- Se define la clase PrintPage que es un widget de estado (**StatefulWidget**). Recibe una lista de datos (**data**) que representa el contenido que se imprimirá en el ticket.
- Se define la clase de estado **_PrintPageState** que extiende State y se encarga de manejar el estado interno de la página de impresión.
- Se crea una instancia de **BluetoothPrint** para interactuar con la impresión Bluetooth. Se definen variables para almacenar la lista de dispositivos Bluetooth, un mensaje de estado y un formateador de números.
- Se sobrescribe el método **initState** para ejecutar la inicialización de la impresora después de que el marco de la interfaz de usuario haya sido renderizado.
- Este método (**initPrinter**) inicia la búsqueda de dispositivos Bluetooth y actualiza la lista de dispositivos cuando se encuentran disponibles.
- Este método (**build**) construye la interfaz de usuario de la página de impresión. Si no se encuentran dispositivos, se muestra un mensaje en el

centro de la pantalla. Si hay dispositivos disponibles, se muestra una lista de dispositivos Bluetooth y permite al usuario seleccionar uno.

- Este método (**_startPrint**) se llama cuando un dispositivo Bluetooth se selecciona en la lista. Conecta la impresora Bluetooth seleccionada y crea una lista de **LineText** que contiene el contenido del ticket. Cada línea de texto puede tener diferentes propiedades como peso, ancho, altura y alineación.