Documentación Taco Paco

Santi Martínez

13 de noviembre de 2025

lstsetlanguaje=Javascript

Índice

1. Introducción

Taco Paco es una aplicación de comida rápida en la que el cliente puede reservar su mesa, y a partir de ahí seleccionar sus productos favoritos.

La app de escritorio forma parte únicamente del establecimiento, ya que su funcionalidad radica en cobrar pedidos y liberar las mesas. Gracias a ello podrá ver también las mesas libres y las ocupadas.

Ambas se conectan entre sí a través de una API Rest elaborada con express y mongoose, el último para guardar y editar los cambios en Mongo Atlas, la base de datos de la aplicación.

2. Arquitectura de la app

Diagrama de flujo

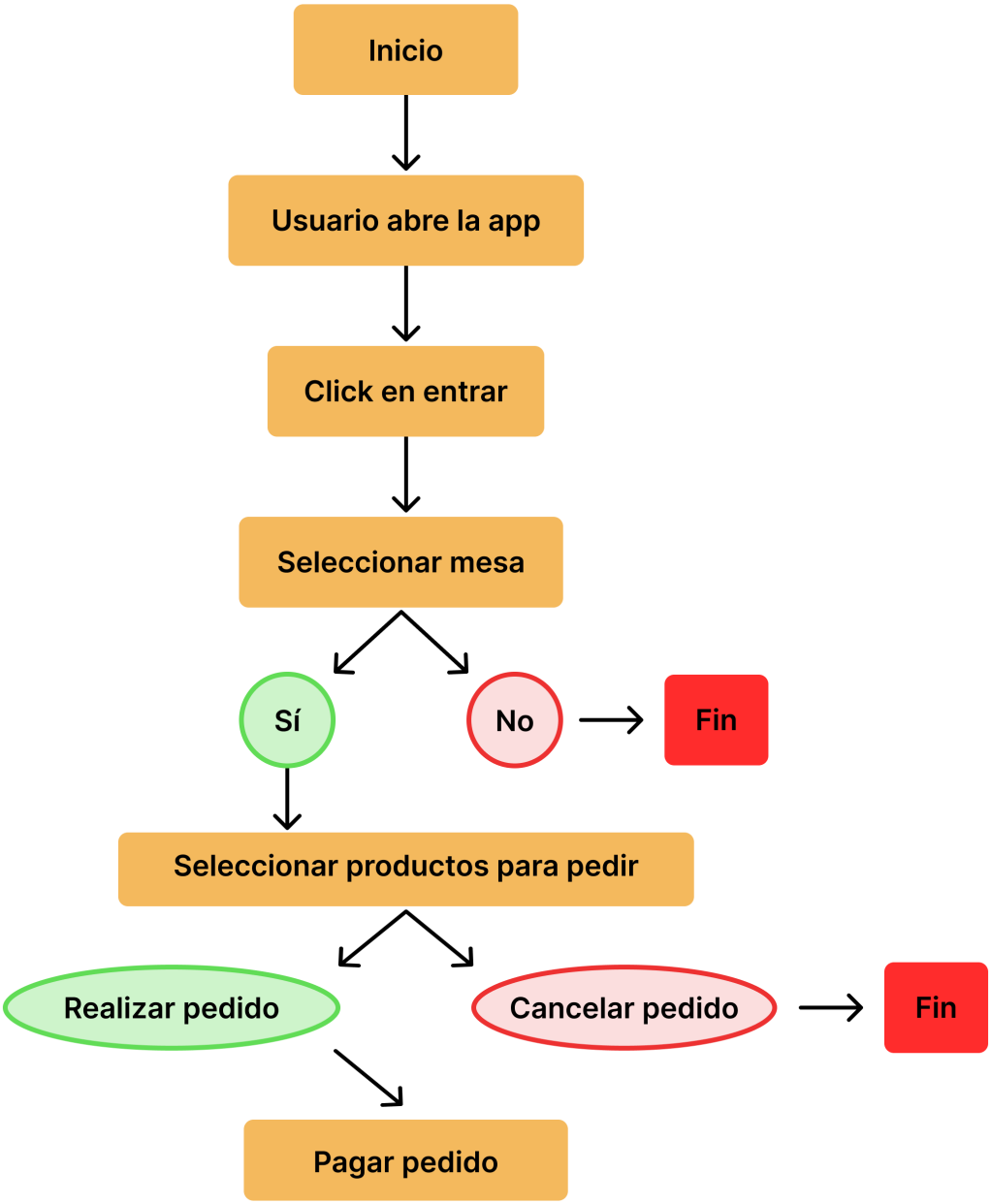


Figura 1: Diagrama de flujo

Diagrama de grafo

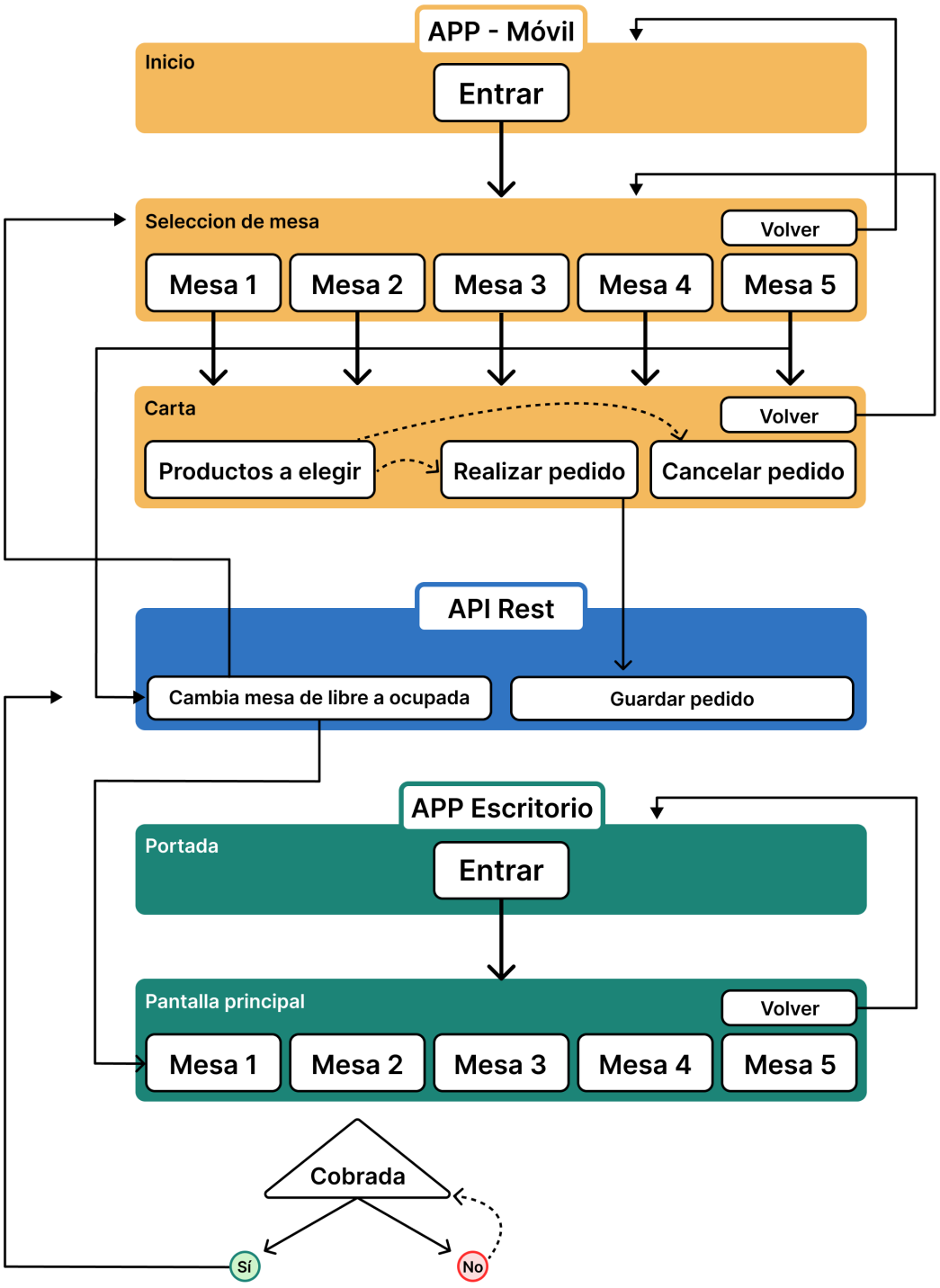


Figura 2: Diagrama de grafo

Diagrama de casos de uso

El **Diagrama de casos de uso** muestra las posibles utilizaciones de la aplicación por todos los actores*.

*Actor: Es el cliente y el servidor, los entes que pueden darle usos a la app.

En este caso el diagrama contiene dos **includes** por parte del cliente, que vienen a partir de la selección de productos dentro de la carta.

- Realizar Pedido
- Cancelar Pedido

Mientras que **extends** contiene uno solo por parte del Servidor:

- Librar mesas

Includes y extends

definiciones

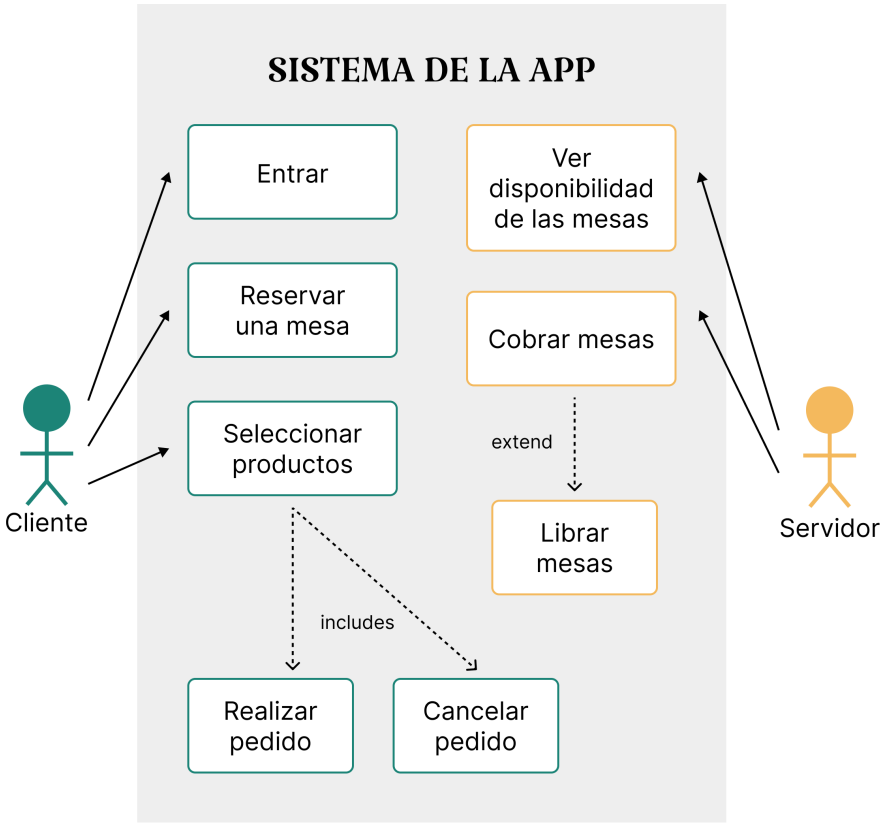


Figura 3: Diagrama de casos de uso

3. APP Móvil

Estructura de la app

- app/
  - sampledata/
  - manifest/
    - `AndroidManifest`
  - kotlin+java/
    - **activities/**
      - `Carta`
      - `EleccionMesa`
      - `MainActivity`
    - **clases/**
      - `Mesa`
      - `Pedido`
    - **servicios/**
      - `Api`
      - `RetrofitClient`
  - java (generated)/
  - resources/
    - **drawable/**
    - **layout/**
      - `carta`
      - `eleccion mesa`
      - `inicio`
    - **mipmap/**
    - **values/**
    - **xml/**
  - resources (generated)/
- Gradle Scripts/

Imágenes de la app

Pantalla principal

Primera pantalla que se encuentra el usuario al iniciarl la app, la única funcionalidad que tiene es la que le da el botón **Entrar**, el cual como su nombre indica, sirve para entrar en el siguiente menú de la app.

En la pantalla se encuentra el nombre de la empresa junto a su año en la que se fundó. También el botón de entrar que es el encargado de cambiar entre activities.

CASO 1: Todas las mesas libres

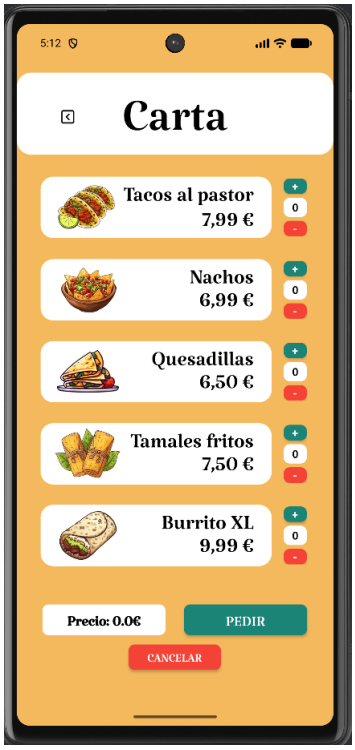
Esta es la pantalla en la que se muestran las mesas disponibles del establecimiento. Se trata de un menú con 5 mesas:

- 3 interiores
- 2 terraza

Esta diferenciación se encuentra en la parte baja de la pantalla, la cual se destaca con dos colores diferentes.
En pantalla se encuentran los botones que llevan a los activities de sus respectivas mesas y un botón para volver al activity anterior.

CASO 2: Mesas ocupadas

Mismo funcionamiento que en el caso anterior, pero sin embargo en esta pantalla se pueden apreciar que tanto la **Mesa 3** como la **Mesa 4**, se encuentran ocupadas.
La app no va a permitir su ocupación ni utilización de las mismas, es decir, quedan bloqueadas hasta que desde la APP-Escritorio se libreren cobrando el pedido a los clientes.



Carta

Esta es la pantalla en la que se muestran las mesas disponibles del establecimiento. Se trata de un menú con 5 mesas:

- 3 interiores
- 2 terraza

Esta diferenciación se encuentra en la parte baja de la pantalla, la cual se destaca con dos colores diferentes.
En pantalla se encuentran los botones que llevan a los activities de sus respectivas mesas y un botón para volver al activity anterior.

4. APP Escritorio

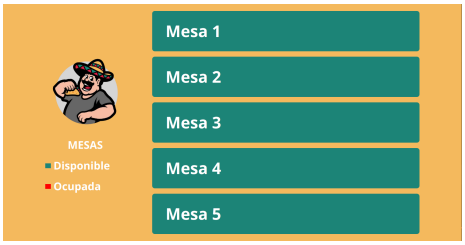
Inicio app

Esta es la pantalla inicial de la app escritorio, en la que el pulsar en **Entrar**, abre el siguiente activity. Cabe decir que es una app únicamente para el establecimiento de Taco Paco, ya que sus posibles usos son:

- Establecer un control de qué mesas están o no disponibles
- En caso de que no estén disponibles, se les puede cobrar su pedido desde esta app, lo que libera a conticuación la mesa

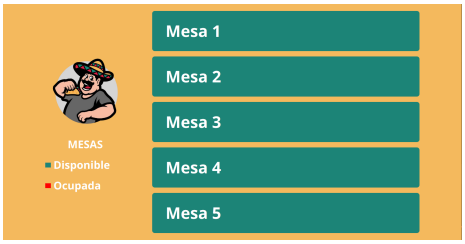
Control de mesas. CASO 1: Todas libres

Esta pantalla es la principal de la app. Como se puede observar contiene como contenido prioritario las 5 mesas des establecimiento. En este caso no hay ninguna mesa con un pedido pendiente.



Control de mesas. CASO 2: Mesas ocupadas

Las mesas ocupadas se pueden apreciar por su fondo en rojo y el pedido que se debe en su interior. Para liberarla solo hay que clicar en el **botón azul** de cobrar



5. API Rest

5.1. Flujo de la API con las APPs

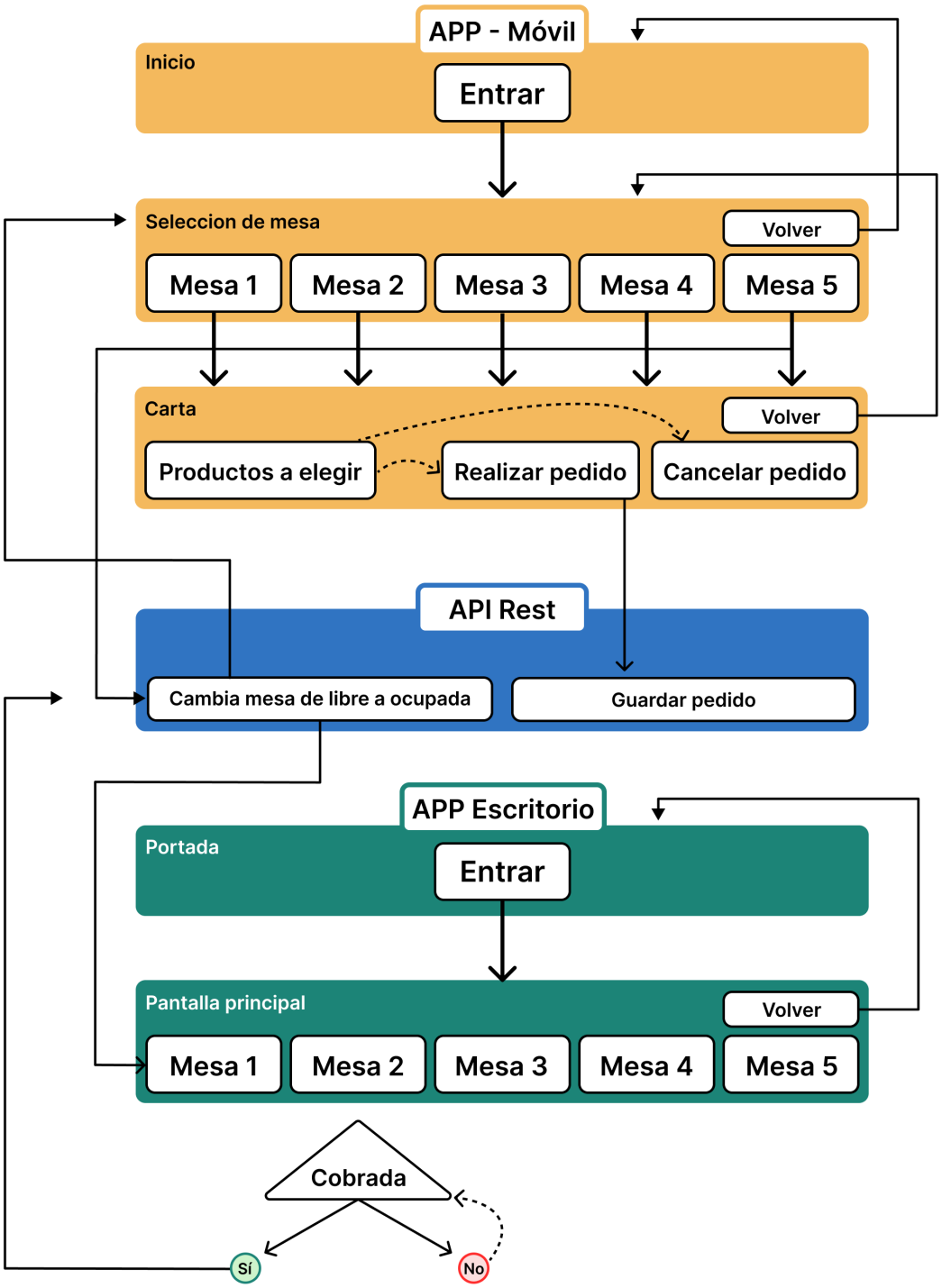


Figura 4: Diagrama de grafo

5.2. Componentes

Figura 5: Estructura de la API

5.2.1. models/

Explicación de lo que son los models

Endpoints

```
const mongoose = require('mongoose')
const mesaSchema = new mongoose.Schema(
    {
        nombre: { type: String, required: true, unique: true },
        ocupada: { type: Boolean, default: false }
    },
    { versionKey: false }
);
const Mesa = mongoose.model("Mesas", mesaSchema);
```

Endpoints

```
const mongoose = require('mongoose')
const pedidoSchema = new mongoose.Schema(
    {
        precioTotal: { type: Number, required: true }
    },
    { versionKey: false }
);
const Pedido = mongoose.model("Pedidos", pedidoSchema);
```

5.2.2. node_modules/

Conjunto de bibliotecas de node. Cada una de ellas cumplen una función en la API:

- **pnpm**: Paquete instalador de node elegido
- **dotenv**: Para la utilización de las variables de entorno guardadas en el .env
- **express**: Módulo con el que se construye la API
- **mongoose**: Módulo de Mongo que sirve para conectarse de forma sencilla a la base de datos de Mongo Atlas. A traves de los modelos creados anteriormente, se pueden guardar, actualizar y eliminar datos en la base de datos a través de los endpoints.

5.2.3. server (Endpoints)

Endpoints

```
app.get('/mesas', async (req, res) => {
    try {
        const mesas = await Mesa.find().sort({ nombre: 1 });
        res.json(mesas);
    } catch (err) {
        res.json({ error: err.message });
    }
});


app.put('/mesas/:nombre', async (req, res) => {
    try {
        console.log("Mesa ocupada:", req.body);

        const nombreMesa = req.params.nombre;
        const { ocupada } = req.body;

        const mesa = await Mesa.findOneAndUpdate(
            { nombre: nombreMesa },
            { ocupada },
            { new: true, upsert: true }
        );
    }
});


app.post('/pedidos', async (req, res) => {
    try {
        const { precioTotal } = req.body;
        const nuevoPedido = new Pedido({ precioTotal });
        const pedidoGuardado = await nuevoPedido.save();

        console.log("Pedido guardado:", pedidoGuardado);
        res.json(pedidoGuardado);
    } catch (err) {
        console.error("Error al guardar pedido:", err);
        res.json({ error: err.message });
    }
});
```

6. Conclusión

7. Bibliografía

- **Formatos de imagen de Latex**: https://manualdelatex.com/tutoriales/figuras
- **Imagen al lado de texto en Latex**: https://foro.rinconmatematico.com/index.php?topic=114810.0
- **Texto al rededor de figuras en Lated**: https://es.wikibooks.org/wiki/Manual_de_LaTeX/Inclusi%C3%B3n_de_gr%C3%A1ficos/Texto_alrededor_de_figuras
- **Insertar código en el pdf**: https://trspos.com/insertar-bloque-de-codigo-latex/