



Escuela Técnica
Superior
de Ingeniería de
Telecomunicación



Universidad
Politécnica
de Cartagena



Escuela
Técnica
Superior

**Ingeniería de
Telecomunicación**

Grado en Ingeniería Telemática

TRABAJO DE FIN DE GRADO

Desarrollo de aplicaciones multiplataforma usando Frameworks y plataformas Low-code: Un caso de estudio

Autor:

Cristian Alexander Tapia Macías

Directora:

María Francisca Rosique Contreras

Cartagena, a 23 de septiembre de 2021



Universidad
Politécnica
de Cartagena

1. ÍNDICE

2.	Introducción.....	6
2.1.	Motivación y objetivos	7
3.	Estado del arte.....	8
3.1.	Comparativa de tecnologías	8
3.1.1	React Native.....	8
3.1.2	Ionic	8
3.1.3	Xamarin	9
3.1.4	Por qué Flutter	9
3.2.	Node.js.....	11
3.3.	XAMPP	11
3.4.	Postman.....	12
3.5.	Visual Studio Code.....	12
3.6.	Android Studio.....	12
3.7.	Git	12
3.8.	GitHub.....	13
3.9.	Dart.....	13
3.10.	Flutter	13
3.10.1	Widgets	13
4.	Caso de estudio	14
4.1.	Contexto	14
4.2.	Planteamiento del problema.....	14
4.3.	Propuesta de solución.....	14
5.	Desarrollo de caso de estudio	15
5.1.	Base de datos	15
5.1.1	Tabla "cash_counts"	15
5.1.2	Tabla "products"	16
5.1.3	Tabla "tables"	16
5.1.4	Tabla "tbills"	17
5.1.5	Tabla "users"	18
5.2.	RESTful CRUD API server	18
5.2.1	Config.....	20
5.2.2	Rutas	20

5.2.3	Controladores	20
5.2.4	Modelos	21
5.3.	Aplicación	22
5.3.1	data.dart	23
5.3.2	server_request.dart.....	24
5.3.3	route_generator.dart	25
5.3.4	main.dart	25
5.3.5	pages.dart.....	26
5.3.6	first_page.dart.....	26
5.3.7	second_page.dart.....	27
5.3.8	third_page.dart	27
5.3.9	fourth_page.dart	28
5.3.10	fifth_page.dart.....	28
6.	Conclusiones	35
7.	Líneas Futuras.....	35
8.	Bibliografía.....	36
9.	Anexos	38
9.1.	Anexo I – Código <i>backend</i>	38
9.1.1	db.config.js	38
9.1.2	cash.count.controller.js	38
9.1.3	product.controller.js	40
9.1.4	table.controller.js.....	43
9.1.5	tbill.controller.js.....	46
9.1.6	user.controller.js.....	49
9.1.7	cash_count.model.js.....	52
9.1.8	product.model.js.....	54
9.1.9	table.model.js	57
9.1.10	tbill.model.js.....	60
9.1.11	user.model.js.....	62
9.1.12	db.js.....	65
9.1.13	cash_count.routes.js.....	65
9.1.14	product.routes.js.....	66
9.1.15	table.routes.js	67
9.1.16	tbill.routes.js	67

9.1.17	user.routes.js	68
9.1.18	server.js.....	68
9.2.	Anexo II – Código de aplicación.....	69
9.2.1	data.dart	69
9.2.2	server_request.dart.....	71
9.2.3	route_generator.dart	75
9.2.4	main.dart	77
9.2.5	pages.dart.....	78
9.2.6	first_page.dart.....	78
9.2.7	second_page.dart.....	79
9.2.8	third_page.dart	80
9.2.9	fourth_page.dart	82
9.2.10	fifth_page.dart.....	85
9.2.11	sixth_page.dart.....	104
9.2.12	edit_bill_product.dart.....	108
9.2.13	edit_price.dart	112
9.2.14	hero_dialog_route.dart [14].....	114

2. Introducción

Hasta ahora cuando queríamos descargar una aplicación en nuestro Smartphone, alguna vez hemos encontrado que no estaba disponible para nuestro entorno (Android o iOS). Este problema tiene su origen en que cuando el individuo como desarrollador tenía que construir algo, tenía que plantearse si quería proyectarse en desarrollo para Android o iOS.

Un ejemplo real de este planteamiento es Calcy IV, una aplicación para Android que realiza análisis de Pokémon en Pokémon Go, sin embargo, no está disponible para iOS, por lo que los usuarios de esta plataforma tienen que usar aplicaciones alternativas.

Esto resultaba, para empresas o proyectos pequeños de pocos recursos (como el expuesto en el ejemplo anterior) en que la aplicación finalmente únicamente estuviese disponible para una de estas plataformas. Esto supone tener un menor alcance en número de usuarios, que repercutirá en la propia evolución del proyecto.

Con la llegada y desarrollo de Flutter, el paradigma afronta una nueva solución para este inconveniente. Teniendo como base el lenguaje de programación Dart, y sus propios Widgets, Flutter cuenta con una fácil curva de aprendizaje.

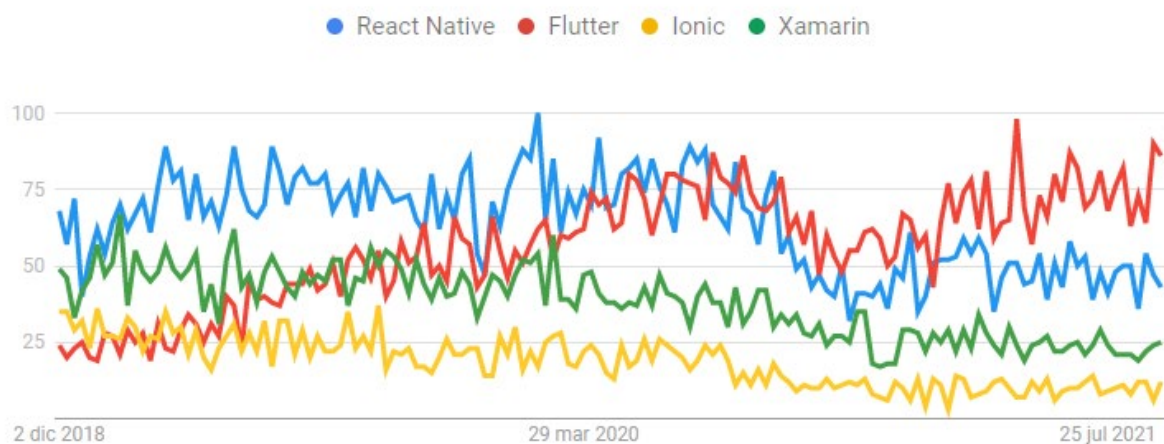


Figura: 2-1: Popularidad de Flutter respecto a competidores

Pese a ser el framework que se ha lanzado más tarde entre sus competidores, es ya el que más ha crecido en popularidad. Gran parte del éxito reside en la propia comunidad de desarrolladores detrás de Flutter que apuestan y confían en la solución de Google para el desarrollo de aplicaciones multiplataforma.

2.1. Motivación y objetivos

En el campo de las TIC el entorno evoluciona rápidamente, y te da una oportunidad de aprendizaje continuo. Con las herramientas de aprendizaje que brinda la UPCT se puede seguir indagando en nuevas tecnologías como en este caso.

Se presenta la oportunidad de poder desarrollar e implementar una aplicación para un negocio de hostelería con el fin de mejorar el sistema que utilizan actualmente y desarrollarlo para incrementar la productividad.

Algunos objetivos marcados para realizar dicha aplicación es recopilar el estado del arte en el desarrollo de aplicaciones multiplataforma, un análisis entre los principales frameworks, plataformas e IDEs existentes y finalmente comenzar el caso de estudio con la opción más adecuada.

3. Estado del arte

En la actualidad hay unas cuantas tecnologías que podrían ser solución para lograr hacer llegar una aplicación a distintos entornos con una única base de código. Siendo las más populares React Native, Flutter, Ionic y Xamarin.

3.1. Comparativa de tecnologías

3.1.1 React Native

React Native, creado por Facebook cuyo lanzamiento fue en 2015, es un framework JavaScript para crear aplicaciones reales nativas para iOS y Android, basado en la librería de JavaScript React para la creación de componentes visuales, cambiando el propósito de los mismos para, en lugar de ser ejecutados en navegador, funcionar directamente sobre las plataformas móviles nativas.



Figura: 3.1-1: Logo de React Native

Para lograrlo React Native usa el mismo paradigma fundamental de construcción de bloques de UI que las aplicaciones nativas reales de Android e iOS, pero gestiona la interacción entre los mismos utilizando las capacidades de JavaScript y React[1].

3.1.2 Ionic

Ionic Framework es un SDK de *frontend* de código abierto creado en 2013 por Drifty Co. Originalmente basado en Angular, para desarrollar aplicaciones híbridas basado en tecnologías web (HTML, CSS y JS). Es decir, un framework que permite desarrollar aplicaciones para iOS nativo, Android y la web, desde una única base de código.



Figura: 3.1-2: Logo de Ionic

Por su compatibilidad e integración de Cordova e Ionic Native, hacen posible trabajar con componentes híbridos. Se integra con los principales frameworks de frontend, como Angular, React o Vue, aunque también se puede usar Vanilla JavaScript [2].

3.1.3 Xamarin

Siendo fundada originalmente en 2011 y adquirida por Microsoft desde 2016, Xamarin es una plataforma de desarrollo de código abierto basada en .NET FRAMEWORK.

Permite el desarrollo de aplicaciones móviles con una compatibilidad de hasta el 90% del código entre distintas plataformas, utilizando C# como único lenguaje de programación.



Figura: 3.1-3: Logo Xamarin

Compilado de forma nativa, Xamarin es una buena herramienta para crear aplicaciones de alto rendimiento con aspecto nativo. Cuenta con Xamarin.iOS y Xamarin.Android como principales clientes, que compilan de forma automática el código fuente con el método Ahead-of-time en caso de iOS y con lenguaje intermedio y posteriormente en AOT para Android, con la posibilidad de adaptarse para solucionar posibles problemas como la asignación de memoria[3].

3.1.4 Por qué Flutter

Flutter es un framework de código abierto desarrollado por Google para crear aplicaciones nativas de forma fácil, rápida y sencilla. Su principal ventaja radica en que genera código 100% nativo para cada plataforma, con lo que el rendimiento y la UX es totalmente idéntico a las aplicaciones nativas tradicionales.



Figura: 3.1-4: Logo Flutter

Como hemos expuesto en los apartados anteriores hay alternativas en cuanto a framework o herramientas que permiten utilizar un mismo código fuente para iOS o Android, pero ninguna genera código nativo en su totalidad.

La principal y más importante ventaja de Flutter es que desarrollas un solo proyecto para todos los sistemas operativos, lo que significa una reducción de costes y tiempo de producción[4].

Pese a que comparte muchas funcionalidades con sus competidores, Flutter dispone de algunas únicas que lo hacen la opción acertada para realizar la aplicación en nuestro caso de estudio.

- Calidad nativa *cross-platform*.
- Soporte para aplicaciones de escritorio: Uno de los puntos clave de Flutter es que una de sus plataformas destino que soporta es para *desktop*. Ya que este será el núcleo de nuestro sistema de gestión, siendo la versión móvil una herramienta para acompañar y flexibilizar el trabajo.
- Experiencia de usuario: Flutter incluye Material Design de Google y Cupertino de Apple, con lo que la experiencia de usuario es óptima. Con la llegada inminente de Android 12 también se está incorporando el nuevo estándar de UI, *Material You*.
- Desarrollo ágil y rápido: Gracias a la característica *hot-reload* y *hot-restart*, se puede programar y ver los cambios en tiempo real en el dispositivo de prueba o emuladores.
- Curva de aprendizaje sencilla.
- Confianza: La seguridad de trabajar con un framework respaldado por un gigante como Google y que se ve reflejado en la popularidad como se ve en *Figura: 2-1*.
- Gran comunidad de desarrolladores colaborando con el proyecto.

Para un programa de gestión es necesario una base de datos, por tanto, el proyecto contará con dos partes, una de *backend* que será el servidor, y otra que será la propia app con la que interactuará el cliente.

Para la creación del servidor *RESTful CRUD API* que la aplicación utilizará, se empleará Node.js, con la librería ExpressJS, siguiendo el patrón de desarrollo *Modelo-Controlador-Ruta* para definir el conjunto de servicios necesarios para el funcionamiento del proyecto. Con XAMPP se enlazará una base de datos MySQL la cual tiene la opción de ser gestionada con una interfaz gráfica phpMyAdmin. Para esta parte de *backend* se usará Visual Studio Code como editor de código y Postman como primera base de pruebas para comprobar su correcto funcionamiento.

En cuanto al desarrollo de la propia aplicación, se usará Flutter & Android Studio. Y tanto el código escrito en este apartado como para la parte del servidor será versionado usando Git & GitHub.

A continuación, se procederá a exponer las tecnologías empleadas en este proyecto.

3.2. Node.js



Node.js [5], [6] es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript, asíncrono, con I/O de datos de una arquitectura orientada a eventos.

Utiliza un sistema escalable y por cada conexión bien realiza su *callback*, o si no hay trabajo que hacer se pondrá en reposo. Esto contrasta con el modelo de concurrencia más común hoy en día, en el que se emplean hilos del sistema operativo.

Al no realizar I/O directamente, el proceso casi nunca se bloquea. Por ello, es muy propicio desarrollar sistemas escalables en Node.js.

3.3. XAMPP



XAMPP [7] es una distribución de Apache que contiene las tecnologías de desarrollo web más comunes en un solo paquete. Es una herramienta ideal para estudiantes para desarrollar y testear aplicaciones en PHP y MySQL.

Los principales servicios de los que haremos uso de XAMPP son MySQL, para la base de datos, y Apache, necesario para poder tener una representación visual de la base de datos con phpMyAdmin.

3.4. Postman



[8], [9] Herramienta para construcción y uso de APIs. Simplifica cada paso del ciclo de vida de la API. Principalmente nos permite crear peticiones sobre APIs de forma sencilla, lo que facilita su testeo.

Todo basado en una extensión de *Google Chrome* que es el que realizará la función de Cliente http.

3.5. Visual Studio Code



Visual Studio Code es un editor de código desarrollado por Microsoft en colaboración con la comunidad programado en *TypeScript*, *JavaScript*, *CSS*, entre otros.

Combina la simplicidad de un editor de código con las funcionalidades centrales del ciclo *edit-build-debug* de un desarrollador.

3.6. Android Studio



[10] IDE (*Integrated Development Environment*) oficial para Desarrollo de apps Android, basado en *IntelliJ IDEA*. Además del potente editor de código de *IntelliJ*. Android Studio ofrece otras características que incrementan la productividad en la construcción de aplicaciones Android, tales como:

- Emulador rápido, fluido y con diversas funcionalidades.
- Entorno unificado para desarrollo de todos los dispositivos Android.
- Aplicar cambios en el código con la app en ejecución sin necesidad de reiniciar.

3.7. Git



Git [11], [12] es una herramienta para el control de versiones de código libre más utilizado en todo el mundo. Presenta una arquitectura distribuida, es un ejemplo de DVCS (*Distributed Version Constrol Sistem*).

En Git la copia de trabajo de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios. Sus tres principales focos son:

- Rendimiento.
- Seguridad.
- Flexibilidad.

3.8. GitHub



GitHub es un servicio de hosting para repositorios Git en la nube que ofrece múltiples herramientas y funciones extra. Su principal ventaja es que hace más sencilla el desarrollo de código colaborativo.

3.9. Dart



Lenguaje de programación de código abierto desarrollado por Google. Es un lenguaje optimizado para el cliente para el desarrollo rápido de aplicaciones en cualquier plataforma. Su objetivo es ofrecer el lenguaje más productivo para desarrollo multiplataforma.

Una gran característica de Dart es que ofrece *null safety*, que se traduce a que una variable solo puede ser nula si se especifica. Con esto será capaz de transformar los errores en ejecución, a errores en análisis en la sintaxis.

3.10. Flutter



Kit de herramientas de UI de Google para realizar aplicaciones compiladas nativamente para diversas plataformas desde una única base de código.

3.10.1 Widgets

En Flutter todos los elementos de UI son llamados widgets, la idea es que se construye la interfaz de usuario a base de estos. Los widgets describen la como debería ser la vista dado su configuración actual y su estado. Cuando el estado de un widget cambia, el widget reconstruye su estructura.

4. Caso de estudio

4.1. Contexto

Este proyecto ha sido realizado para el restaurante Brasería Medieval, situado en Alicante, en el municipio de Monforte del Cid.

Con el objetivo de actualizar y mejorar, el programa de gestión implementado en su apertura en 2006.

Para obtener mejor información sobre las necesidades y posibles mejoras del sistema establecido en el restaurante tuve un acercamiento con el programa y reuniones con el jefe de sala para examinar los puntos fuertes y débiles. Así como para escuchar cualquier petición de la que fuese necesario darle especial importancia.

4.2. Planteamiento del problema

Actualmente el negocio es gestionado por un único dispositivo, que es el ordenador principal que se encuentra localizado en la zona de bar. Los trabajadores tienen que anotar a mano las comandas, llevar una copia a cocina, otra a la brasa y otra a la barra, donde finalmente se introducen los datos en el ordenador.

Esto supone que el jefe de sala tenga que recorrer varias veces el restaurante para llevar la información a todos los puntos. El problema que este método presenta es que en las horas punta el estrés es muy alto y hay productos que no se anotan, se olvidan y finalmente no se cobra al cliente.

4.3. Propuesta de solución

En vista de la segmentación del negocio, se ha visto que es necesario que la propuesta de solución sea capaz de ser ejecutada en distintos tipos de dispositivos. Escritorio, para el ordenador centrar en la zona bar, *smartphone* o similar para el jefe de sala que tiene que recorrer las mesas, y *tablet* para las zonas de cocina y brasa.

Para incrementar el rendimiento y realizar un trabajo más eficaz, se propone el desarrollo una aplicación que podrá ser ejecutada en un *Smartphone*, *Tablet* de la que disponga el trabajador para poder añadir los productos con un solo *tap*. Estos productos se verán reflejados automáticamente en el ordenador principal de la zona bar.

Principalmente el programa a desarrollar debe cumplir una serie de funcionalidades fundamentales para el correcto funcionamiento del restaurante. Las principales deben ser:

- Contar con un catálogo de productos.
- Añadir / Modificar / Eliminar productos.
- Disponer de gestión de mesas

- Contar con una calculadora automática que maneje el sistema de cobros y cambios dependiendo de los productos que consuma una mesa.
- Apartado para consultar los movimientos y ver el resumen del día para el conteo de caja.

5. Desarrollo de caso de estudio

5.1. Base de datos

Para comenzar esta parte primero tenemos que establecer una base de datos sobre la que se realizarán las peticiones. Se ha optado por una base de datos relacional SQL (*Structured Query Language*)

A continuación, se presenta la estructura planteada.

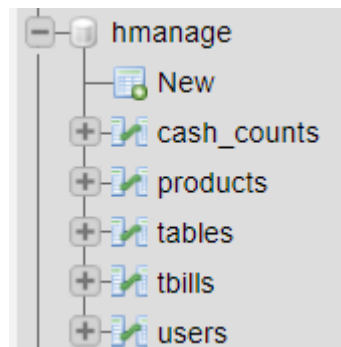


Figura: 5.1-1: Tablas de la base de datos

5.1.1 Tabla “cash_counts”

Esta tabla guardará un histórico económico por días del resumen financiero del restaurante, se dividirá en:

- **Id:** Identificador de la entrada en tabla.
- **Day:** Día natural sobre el que se guardan los datos.
- **Net_sale:** Cobro total, que suma cobros al contado y tarjeta.
- **Card_payments:** Cobros con tarjeta.
- **Cash_payments:** Cobros en efectivo.
- **Number_sales:** Número de ventas en ese día.
- **Average_ticket:** Cálculo medio del total vendido entre el número de ventas.
- **Income:** El total de entradas de dinero que se hace a caja.
- **Outflow:** El total de dinero que ha salido de caja.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(10)			No	None		AUTO_INCREMENT
2	day 	date			Yes	NULL		
3	net_sale	decimal(10,2)			Yes	NULL		
4	card_payments	decimal(10,2)			Yes	NULL		
5	cash_payments	decimal(10,2)			Yes	NULL		
6	number_sales	int(10)			Yes	NULL		
7	average_ticket	decimal(10,2)			Yes	NULL		
8	income	decimal(10,2)			Yes	NULL		
9	outflow	decimal(10,2)			Yes	NULL		

Figura: 5.1-2: Tabla "cash_counts"

5.1.2 Tabla "products"

Aquí se recogen todos los productos que oferta el restaurante:

- **Id:** Identificador de producto.
- **Name:** Nombre del producto.
- **Price:** Precio del producto.
- **Category:** Categoría del producto.


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(10)		UNSIGNED	No	None		AUTO_INCREMENT
2	name	varchar(40)	utf8mb4_general_ci		No	None		
3	price	decimal(10,2)			Yes	NULL		
4	category	text	utf8mb4_general_ci		Yes	NULL		

Figura: 5.1-3: Tabla "products"

5.1.3 Tabla "tables"

Esta tabla contiene las mesas a las que se les añadirán los productos.

- **Id:** Identificador de mesa
- **Number:** Numeración de la mesa que indica el usuario, variará dependiendo de la zona de trabajo.
- **Total:** El total de factura a abonar por el cliente.


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(11)			No	None		AUTO_INCREMENT
2	number	int(11)			No	None		
3	total	decimal(10,2)			Yes	NULL		

Figura: 5.1-4: Tabla "tables"

5.1.4 Tabla "tbills"

Esta tabla contiene una colección de productos consumidos por las mesas que aún están pendiente de pago. El nombre de *tbill* se refiere a table bill y detallamos su construcción a continuación:

- **Id:** Identificador de tbill.
- **Tnumber:** Número de mesa a la que corresponde el producto.
- **Item:** Nombre del producto.
- **Units:** Número de unidades consumidas del producto.
- **Iprice:** Precio del producto.
- **Total:** Precio total del tbill calculado en base a *iprice* y *units*





#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(11)			No	None		AUTO_INCREMENT
2	tnumber 	int(11)			No	None		
3	item 	varchar(40)	utf8mb4_general_ci		Yes	NULL		
4	units	int(11)			Yes	NULL		
5	iprice 	decimal(10,2)			Yes	NULL		
6	total	decimal(10,2)			Yes	NULL		

Figura: 5.1-5: Tabla "tbills"

Además, para esta tabla se ha creado un clave triple para poder gestionar duplicados en una mesa, que estará compuesta por las columnas *tnumber*, *item* e *iprice*.

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null
				tnumber	6	A	No
triple_index	BTREE	Yes	No	item	6	A	Yes
				iprice	6	A	Yes

Figura: 5.1-6: Clave índice triple

5.1.5 Tabla “users”

Esta tabla contiene los usuarios registrados en la aplicación, que serán utilizados en un futuro para el manejo y registro de personas.

- **Id:** Identificador de usuario.
- **Login:** Campo con el que iniciará sesión.
- **Name:** Nombre de usuario.
- **Email:** Correo electrónico.
- **Passwd:** Contraseña del usuario.


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id 	int(11)			No	None		AUTO_INCREMENT
2	login	text	utf8mb4_general_ci		No	None		
3	name	text	utf8mb4_general_ci		No	None		
4	email	text	utf8mb4_general_ci		No	None		
5	passwd	text	utf8mb4_general_ci		No	None		

Figura: 5.1-7: Tabla “users”

5.2.RESTful CRUD API server

REST es un acrónimo de *REpresentational State Transfer*, y se denomina REST a cualquier interfaz que se comunique entre sistemas mediante HTTP para obtener o generar, y en definitiva tratar datos en todos los formatos posibles.

Por otro lado, API (Application Programming Interface), es un programa que permite la comunicación entre aplicaciones. No tiene la imposición de comunicarse en una red, puede ser entre dos aplicaciones de un mismo equipo.

CRUD es acrónimo de *Create, Read, Update, Delete*. Que se usa para referirse a las funciones básicas que realizará el servidor.

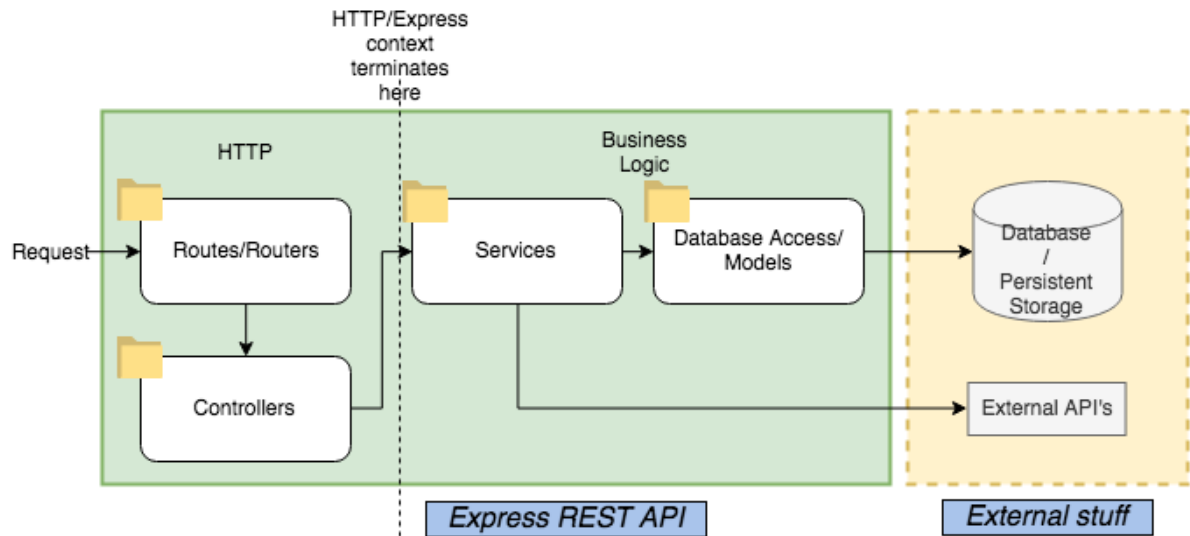


Figura: 5.2-1: Diagrama de arquitectura a seguir

Se ha seguido una estructura muy común en cuanto a la estructura del servidor usando Node.js y Express. Con un patrón Ruta-Controlador-Modelo que tiene una estrecha relación con el modelo de la base de datos descrita en el apartado anterior. El resultado de estructura se representa a continuación y se detallarán las funciones de sus partes en detalle.

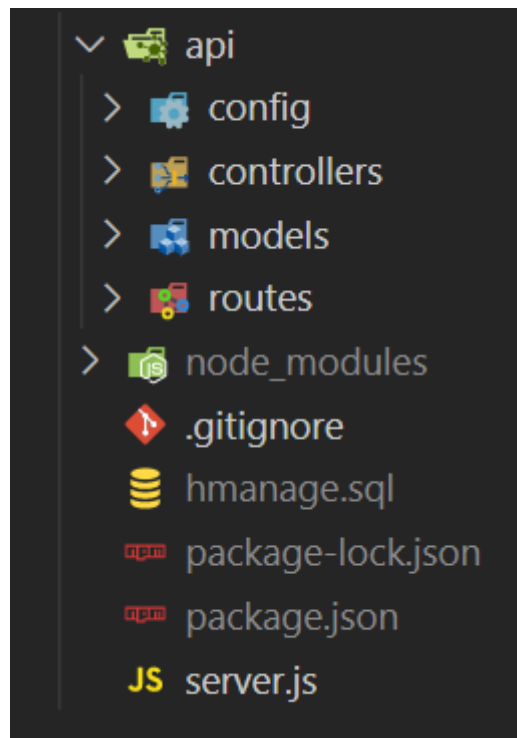


Figura: 5.2-2: Estructura del servidor

5.2.1 Config

Únicamente se encuentra el fichero en el cual se definen los parámetros para posteriormente establecer la conexión a la base de datos. Estos son:

- **Host:** Ubicación de la base de datos
- **User:** Nombre de usuario.
- **Password:** Contraseña.
- **Db:** Nombre de la base de datos a la que se accederá.

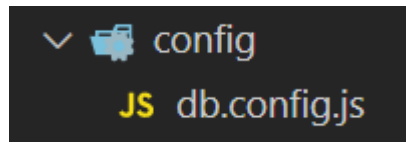


Figura: 5.2-3: Contenido de "config"

5.2.2 Rutas

Las peticiones llegarán mediante una URL que variará dependiendo del servicio que se quiera utilizar, y será encadenada directamente con su controlador correspondiente asignándole la función que deberá ejecutar, ninguna lógica en esta parte.

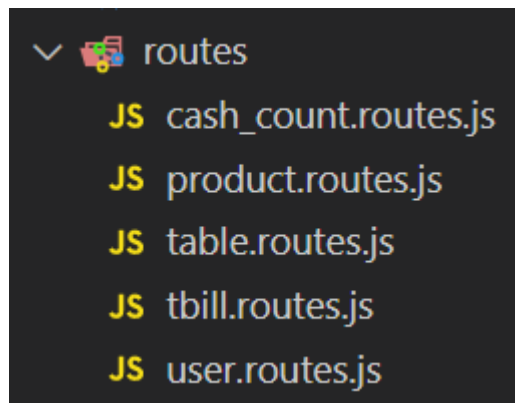


Figura: 5.2-4: Contenido de "routes"

5.2.3 Controladores

Una vez la petición es dirigida al controlador, y es el encargado de devolver datos o error en función del resultado de la petición. Los controladores establecen el código HTTP de la respuesta. Para obtener respuesta, los controladores reenvían la petición HTTP hacia las rutas.

Es aquí, como se ver representado en *Figura: 5.2-1*, donde termina la capa de Express/HTTP para poder obtener escalabilidad, de lo contrario si se tuviese que cambiar frameworks, sería más trabajo iterar por todas las instancias del objeto sobre el que se trabaja y modificarlas[13].

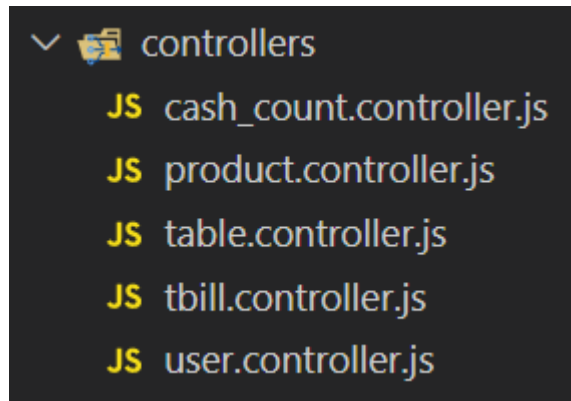


Figura: 5.2-5: Contenido de "controllers"

5.2.4 Modelos

Es donde se desarrollará la mayoría de la lógica del servidor. Concretamente es donde se definen las consultas SQL que se realizarán a la base de datos. Al separar esta lógica de los controladores se proporciona rapidez cuando se requiera testear esta lógica.

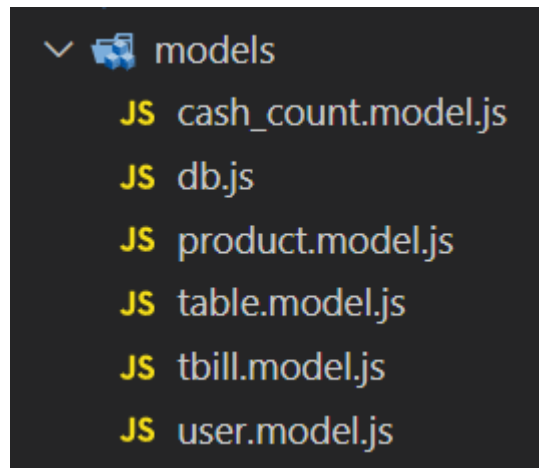


Figura: 5.2-6: Contenido de "models"

En esta carpeta también se encuentra el archivo *db.js*, que se encargará de crear la conexión a la base de datos utilizando los parámetros especificados en 5.2.1

Finalmente, en *Figura: 5.2-2* podemos ver el archivo *server.js* que tendrá la función de importar todas las rutas y arrancar el propio servidor que en nuestro caso escuchará peticiones en el puerto 8888.

5.3. Aplicación

Para el desarrollo de la aplicación usaremos Android Studio con Flutter que cuenta con Dart como lenguaje de programación base. Nuestra aplicación cuenta con la siguiente estructura:

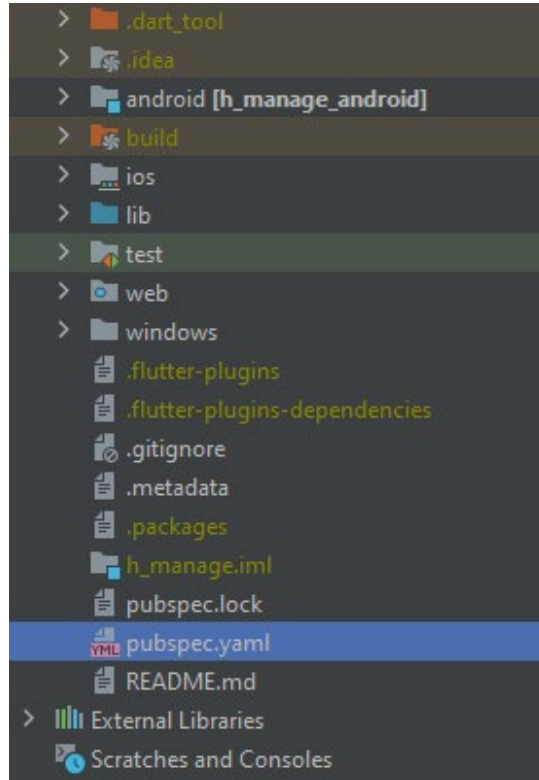


Figura: 5.3-1: Estructura del proyecto

Pero previo a proceder a comentar los archivos donde hemos realizado todo el desarrollo, haremos un inciso para hablar de *pubspec.yaml*, lugar donde declararemos, en dependencias, todos paquetes externos que necesitemos importar para el proyecto.

```
23 dependencies:
24   flutter:
25     sdk: flutter
26
27   cupertino_icons: ^1.0.2
28   google_fonts: ^2.1.0
29   math_expressions: ^2.1.1
30   http: ^0.13.3
31   badges: ^2.0.1
32   escpos: ^5.0.2
33   qr_flutter: ^4.0.0
```

Figura: 5.3-2: Dependencias del proyecto

Los principales ficheros con los que trabajaremos se encuentran en la carpeta lib, en la raíz se encuentran archivos que no tendrán vistas al usuario con excepción de main.dart que es con el que arrancará el programa.

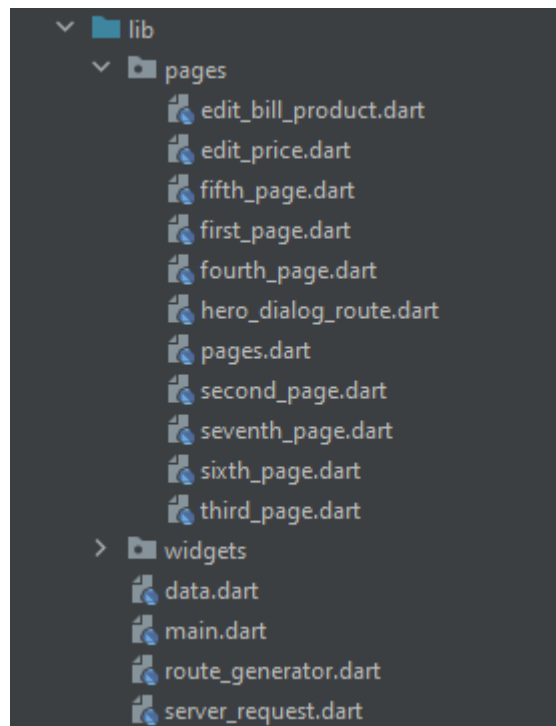


Figura: 5.3-3: Archivos de la aplicación

5.3.1 data.dart

Dart es un lenguaje que tiene soporte para la orientación a objetos, y aquí es dónde definiremos todos los objetos que vamos a utilizar, que se corresponderán y seguirán la estructura de las tablas de la base de datos detalladas en la colección de figuras del apartado 5.1. Estos serán construidos a partir de un JSON que se reciba al hacer una petición http.get al servidor. A continuación, se listan los objetos:

```
// Creating a User Object
class User {
  final int id;
  final String login;
  final String name;
  final String email;

  const User({
    required this.id,
    required this.login,
    required this.name,
    required this.email,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] as int,
      login: json['login'] as String,
      name: json['name'] as String,
      email: json['email'] as String,
    );
  }
}
```

Figura: 5.3-4: Estructura ejemplo de un objeto

- User: Objeto asociado a una fila de la tabla *users*.
- Product: Asociado a una fila de la tabla *products*.
- Tablee: Con dos "e" por ser *Table* una palabra reservada de Dart, asociado a una fila de la tabla *tables*.
- Tbill: Asociado a una fila de la tabla *tbill*.
- CashCount: Asociado a una fila de la tabla *cash_counts*.

5.3.2 server_request.dart

En este archivo se localizan las peticiones hacia nuestro RESTful API server. Requiere las siguientes importaciones:

- *data.dart*: Una vez realizada la petición requerirá de la estructura de los objetos para transformar la respuesta recibida por el servidor en objetos con los que tratar en Dart.
- *foundation.dart*: Requerido para la función *compute* que se encargará en paralelo de hacer un *parse* de la respuesta del servidor en objetos definidos.
- *async.dart*: Para que las peticiones del servidor sean asíncronas, evitando así el bloqueo del programa esperando la respuesta.
- *convert.dart*: Convertirá el cuerpo de la respuesta en un *Map* que servirá como intermediario para transformarlo finalmente en un objeto.
- *http.dart*: Librería basada en futuros para hacer peticiones HTTP.


```

static Future createCashCount() async {
  final response = await http.post(Uri.parse(_host + 'cash_counts'));

  if (response.statusCode != 200)
    throw Exception('Failed to create cash count');
}

```

Figura: 5.3-5: Ejemplo de función

En el ejemplo de la figura de arriba realizamos una petición que no recibe argumentos, mediante un `http.post` a la url compuesta por la dirección del servidor y `'cash_counts'`, en este ejemplo específico sería `http://192.168.1.134:8888/cash_counts`. Si el servidor no responde con un código de 200 OK se lanzará una excepción.

5.3.3 route_generator.dart

Aquí se encuentra la lógica de navegación entre páginas. Lo haremos con *named routes* (rutas nombradas), esto permitirá individualizar cada página, ahorrando así duplicidad en el código a la misma vez que proporcionando una clara estructura compartimentada para cada página. Otra de las ventajas que proporciona es que podremos pasar argumentos entre páginas, lo que es fundamental y necesario.

```

case '/sixth':
  if(args is List<num>) {
    return MaterialPageRoute(
      builder: (_) =>
        SixthPage(
          total: args[0],
          tnumber: args[1],
        )); // SixthPage, MaterialPageRoute
  }
  return _errorRoute();

```

Figura: 5.3-6: Ejemplo de navegación con parámetros de entrada

El caso expuesto en la figura inmediatamente anterior corresponde a la navegación de la página para cobrar en efectivo, requiere de un argumento de entrada que es una lista de números, que serán el total a cobrar, para calcular el cambio a devolver, y el número de mesa a la que realiza el cobro, para, mediante una petición al servidor, limpiar los productos asociado a la mesa. En caso de no recibir los argumentos correctos nos llevaría a una página genérica de error.

5.3.4 main.dart

Es el punto de arranque de la aplicación, mostrando directamente la página principal, se usa rutas nombradas, es necesario importar *route_generator.dart* así como la librería de *material.dart*, que es la que contiene lo elementos UI de Material Design o Cupertino.

El resto de páginas están contenidas en la carpeta lib, siendo las cuatro primeras un *sandbox* para experimentar con las funcionalidades de Flutter:

5.3.5 pages.dart

En java si se quisiera importar todos los archivos dentro de una carpeta podríamos usar `***`. Pero Dart no contempla esto. Para solventar este problema se ha creado *pages.dart*.

```
export 'first_page.dart';  
export 'second_page.dart';  
export 'third_page.dart';  
export 'fourth_page.dart';  
export 'fifth_page.dart';  
export 'sixth_page.dart';  
export 'seventh_page.dart';  
export 'edit_bill_product.dart';
```

Figura: 5.3-7: Código de *pages.dart*

Es una sencilla exportación de las páginas que se verían afectadas por `***`, de esta manera cuando queremos importar las páginas para la navegación en *route_generator.dart* en lugar de importar las páginas una a una, solo importamos *pages.dart*

5.3.6 first_page.dart

Creada con la intención de probar cómo funciona el sistema de rutas nombradas, esta página contiene una fila de botones centrados con los que nos moveremos por distintas páginas de la aplicación.

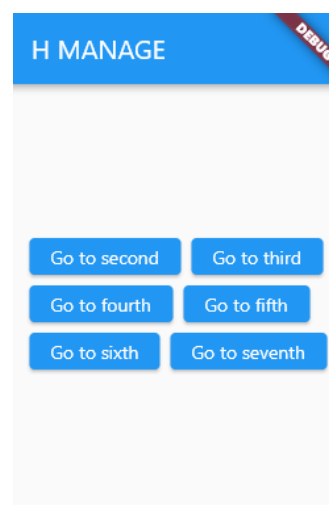


Figura: 5.3-8: Vista de *first_page.dart*

5.3.7 second_page.dart

Aquí se puso a prueba el intercambio de argumentos entre páginas, siendo necesario un argumento para acceder, se muestra en pantalla dicho argumento "Hello there from the first page". Por último, tiene un botón para ir a la página tres.

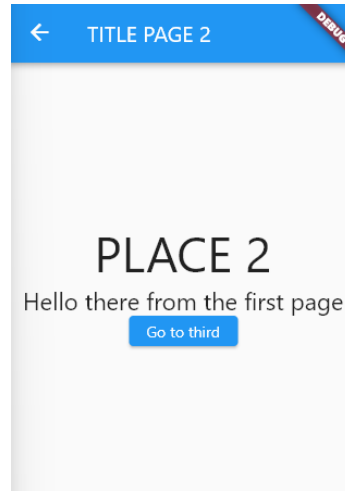


Figura: 5.3-9: Vista de second_page.dart

5.3.8 third_page.dart

Página planteada para comprobar cómo se comporta cuando se imponen más de una ruta en el *stack* de navegación. También se aprovechó para realizar la primera muestra en pantalla de datos recibidos tras una petición al servidor.

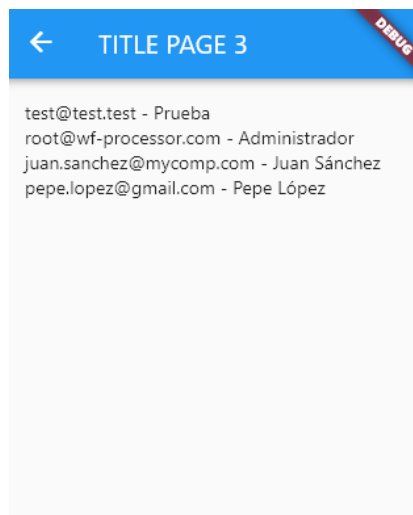


Figura: 5.3-10: Vista de third_page.dart

Esta muestra la lista de usuarios en la base de datos si la petición se completa con éxito, o la página genérica de error si hay error.

5.3.9 fourth_page.dart

En esta ruta se siguen probando funcionalidades sobre las peticiones al servidor. Concretamente se muestra un formulario para la creación de un nuevo usuario. Si es registrado en la base de datos este se muestra en pantalla.

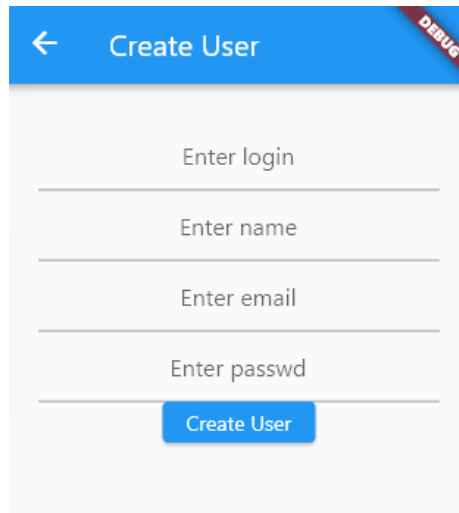


Figura: 5.3-11: Vista de fourth_page.dart

5.3.10 fifth_page.dart

Aquí es donde se ha escrito la mayoría del código propiamente funcional para la aplicación del caso de estudio. La vista principal consta de cuatro pestañas, pero dejaremos apartada la primera ya que su cometido es sencillamente de probar funcionalidades conforme se requiera.

5.3.10.1 Pestaña Tables

En esta pestaña se generará una vista de botones de todas las mesas que hay en la base de datos. Estos botones mostrarán el número de la mesa y el total a cobrar pendiente de esta. Y serán generados con el Widget *FutureBuilder*.

En la parte superior encontraremos un texto que nos pide seleccionar mesa, pues si no hay mesa seleccionada el programa no te dejará desplazarte por el resto de pestañas. Si lo intentas saldrá en la parte inferior un *SnackBar* (mensaje que desaparece a los pocos segundos) indicándote qué hay que hacer para poder realizar la acción.



Figura: 5.3-12: Vista de la pestaña Tables con Snackbar

Una vez seleccionada la mesa, la vista será desplazada automáticamente a la pestaña productos, y el mensaje de la parte superior pasará a mostrar el número de la mesa seleccionada.

5.3.10.2 Pestaña Products

Esta guarda mucha similitud con la del apartado anterior, al igual que en la pestaña tablas, la vista de esta será construida a partir de *FutureBuilder*. Que mostrará todos los productos que haya en la base de datos. Y no nos dejará acceder a la siguiente parte hasta que hayamos añadido algún producto a la mesa. Si lo intentamos aparece nuevamente un *Snackbar* en la parte inferior solicitando al usuario que introduzca algún producto.

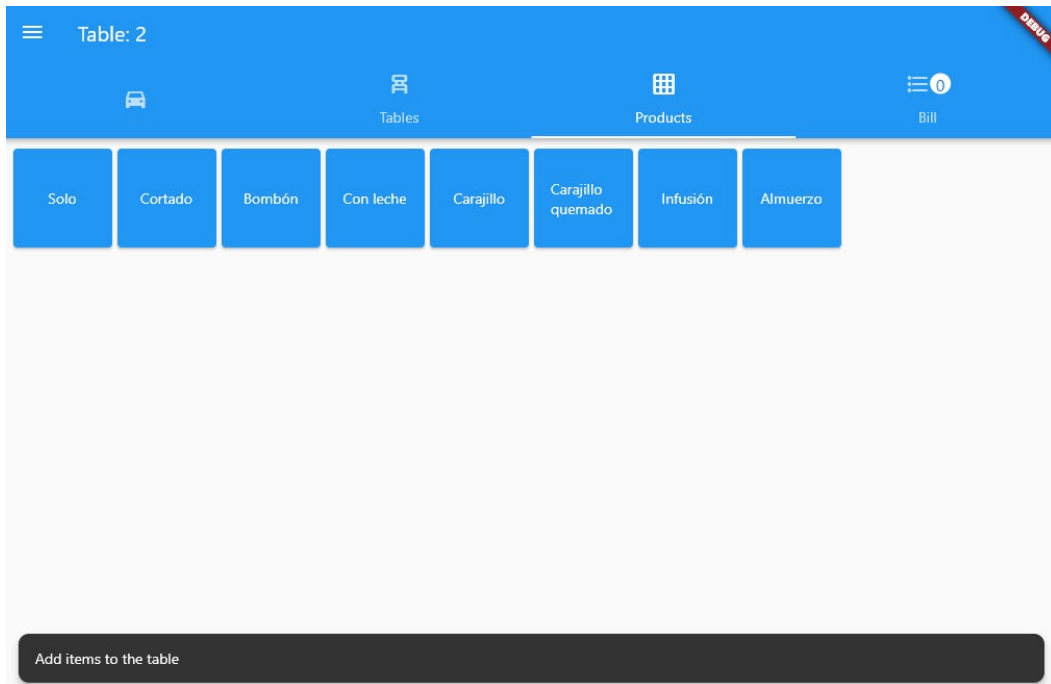


Figura: 5.3-13: Vista de la pestaña products con SnackBar

En la parte siguiente pestaña podemos ver un *badge*, este indica el número de productos que tiene la mesa seleccionada y va aumentando conforme se añadan más. Una vez haya entradas, se podrá proceder al siguiente apartado.

5.3.10.3 Pestaña Bill

Aquí aparecerá un resumen de los productos consumidos por una mesa, nuevamente se utiliza el Widget *FutureBuilder* para obtener la lista de productos asociados a dicha mesa.

Al final de cada entrada hay un botón que llevará a *edit_bill_product.dart*.

A la derecha de la lista, o debajo si el dispositivo no lo permite, se encuentra el total a pagar por la mesa, "Money" corresponde al dinero entregado por el cliente y "Change" que es el cambio que le corresponde. Esto se manejará en *sixth_page.dart*

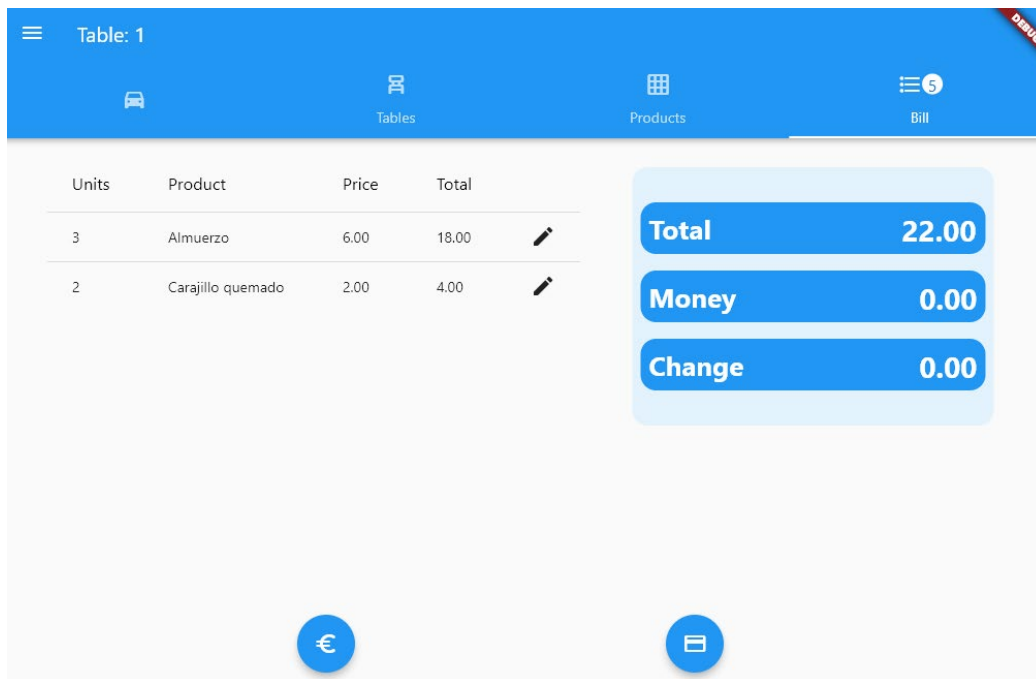


Figura: 5.3-14: Vista de la pestaña Bill

En la parte inferior se encuentran dos botones:

- El botón con el símbolo de euro lleva a `sixth_page.dart`
- El que tiene una tarjeta registrará directamente el cobro y limpiará la mesa.

5.3.10.4 `edit_bill_product.dart`

Esta página tomará como datos la fila desde la que fue pulsado el botón editar, y permite modificar todos los campos de la entrada en la pestaña Bill, unidades, nombre del producto y precio. Mostrando como último campo una previsualización del total de esa entrada.

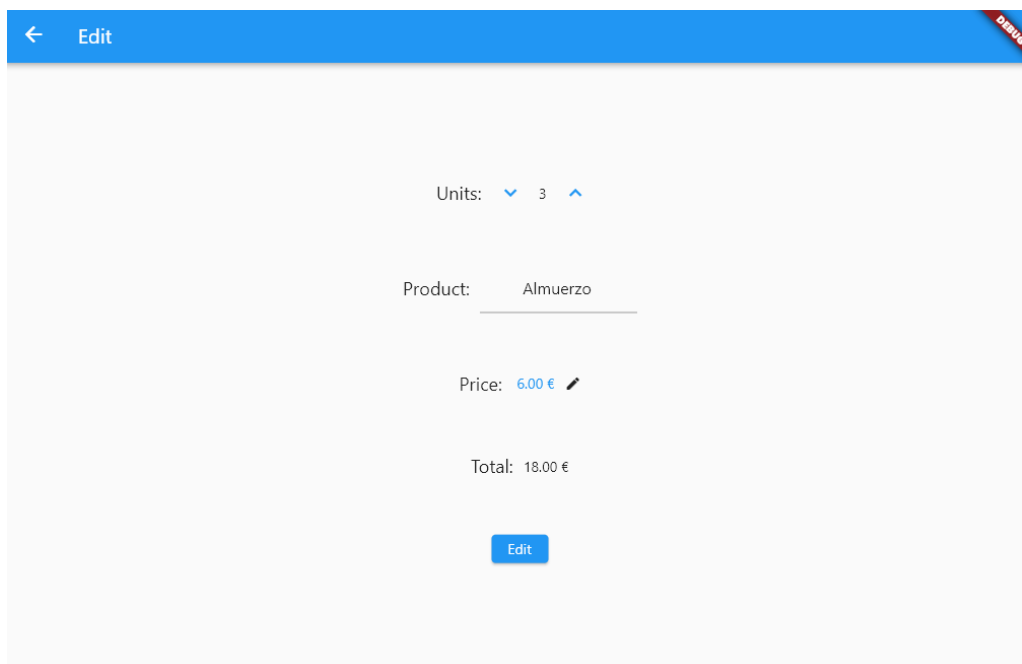


Figura: 5.3-15: Vista de `edit_bill_product.dart`

Para editar el precio del producto se emplearán *hero_dialog_route.dart* y *edit_price.dart*

Una vez realizada las modificaciones que se desea, al pulsar el botón "Edit" se realizará la petición al servidor y volveremos a la vista de la pestaña Bill, devolviéndole un booleano para notificarle si es necesario o no de que realice una reconstrucción del Widget.

5.3.10.5 *hero_dialog_route.dart* y *edit_price.dart*

Para esta parte se ha hecho uso del código por *funwithflutter* en su repositorio público *flutter_ui_tips*[14].

hero_dialog_route.dart define un tipo de navegación mediante PopUp Cards, que a la vista del usuario parece que seguimos en la misma página. Modifica algunos parámetros de *PageRoute* para conseguir este efecto.

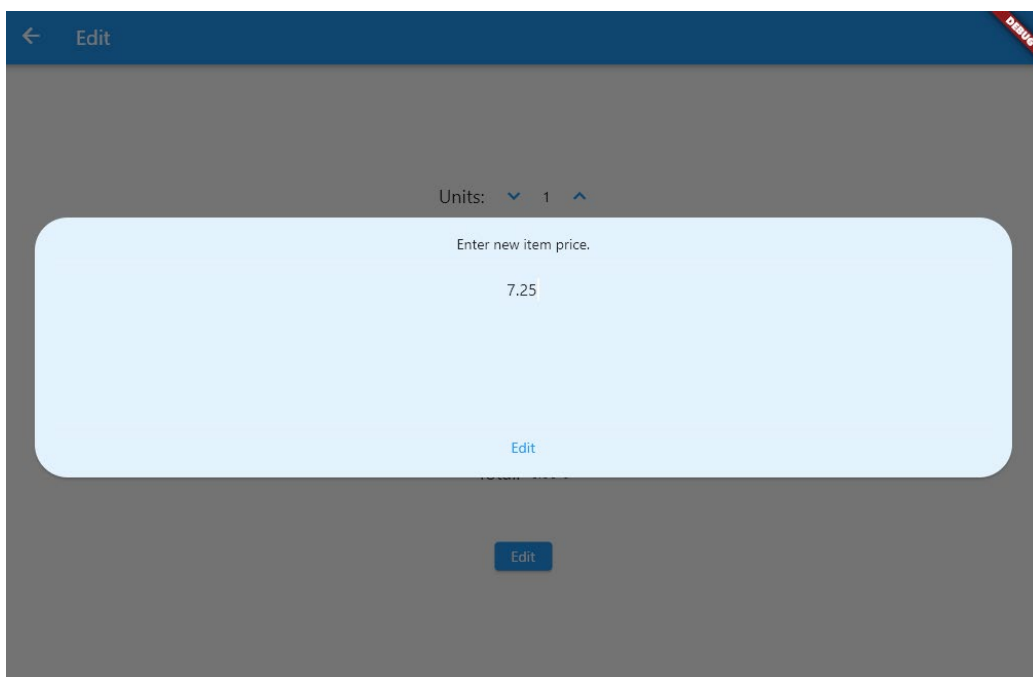


Figura: 5.3-16: Navegacion usando efecto PopUp card

edit_price.dart es la página que se sobrepondrá a la principal, y haciendo uso de *hero_dialog_route.dart* a la vista resulta como un PopUp

5.3.10.6 *sixth_page.dart*

Aquí se maneja el cobro en efectivo de una mesa, se llega desde el botón con icono de euro en la pestaña Bill, toma los argumentos explicados en el ejemplo de *Figura: 5.3-6*



Figura: 5.3-17: Vista de sixth_page.dart

En la parte superior se muestra a modo informativo la mesa sobre la que se está realizando el cobro.

En la parte central se dispone de "Total" que muestra el total a cobrar, y en "Cash" se muestra el dinero entregado por el cliente, que se podrá ir agregando pulsando los botones en pantalla. Si la cantidad que da el cliente es menor a la total a cobrar, se mostrará un dialogo de alerta informando al usuario y se le pedirá que vuelva a introducir la cantidad nuevamente.

Con el botón "Clear" ponemos a cero el parámetro "Cash" y con el botón de "OK" si todo es correcto, realizará la llamada al servidor para registrar la transacción, limpiar la mesa de productos y volverá a *fifth_page.dart* en la pestaña Bill.

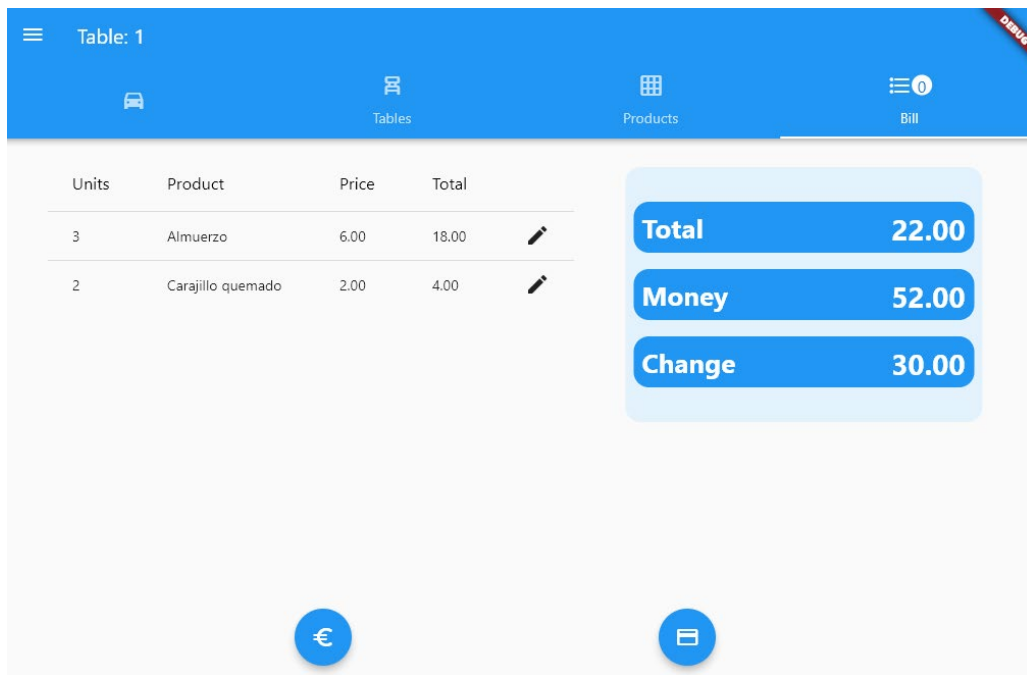


Figura: 5.3-18: Redirección a `fifth_page.dart` con parámetros devueltos

Al volver a `fifth_page.dart` devolvemos una lista que contiene el dinero entregado por el cliente y el cambio que le corresponde, actualizando así los Widget pertinentes.

6. Conclusiones

En este trabajo se presenta el desarrollo en progreso de una aplicación para la gestión y funcionamiento de un restaurante mediante el uso de SDK para desarrollar aplicaciones multiplataforma partiendo de una única base de código. Empleando como entorno el IDE Android Studio y el SDK Flutter. Además, se ha requerido el desarrollo de una RESTful API, utilizando Node.js y la librería ExpressJS, que utiliza peticiones HTTP para manejar los datos de una base de datos SQL que también se ha creado utilizando MySQLS.

A lo largo del desarrollo se ha podido comprobar que cuando ejecutamos la aplicación en alguna plataforma, ya sea Android o Desktop, esta no tiene ningún problema de rendimiento si se comparase con una hecha específicamente para la plataforma destino. Comprobando de primera mano todo el potencial que ofrecen los SDKs para desarrollo multiplataforma. Que comprime de manera drástica las líneas de código y todo el aprendizaje anterior en contraste a si se hubiese optado por la manera tradicional de tener un proyecto para cada plataforma.

Por tanto, se pueden dar por satisfechos los objetivos planteados en el Trabajo fin de Grado dado que a raíz del análisis de los frameworks más populares en la actualidad, se ha hecho la elección de Flutter como opción a seguir. Pues dispone de un amplísimo repertorio de contenido, tanto por documentación como por contenido audiovisual que crean desde el equipo de desarrollo, lo que resulta en una experiencia de programación con una curva de aprendizaje muy satisfactoria.

7. Líneas Futuras.

Una vez terminado este trabajo fin de grado que ha servido para sentar una base para el caso de estudio, se seguirá con el desarrollo continuo de la aplicación hasta su implementación en la Brasería Medieval. Para ello a continuación se listan una serie de funcionalidades que formarán parte del producto final:

- Complementar los futuros cuando se completan con error añadiendo botón para reintentar la petición.
- No mostrar el total de una mesa cuando no hay ningún producto asociado a ella. Esto facilitará al usuario el identificar qué tiene pendiente.
- No mostrar el botón de editar el producto cuando ya se ha cobrado la mesa.
- Implementar sistema de tickets por impresora térmica o de manera digital mediante escaneo de código QR.
- Implementar sistema de clientes para las personas que necesiten Factura.
- Categorizar las mesas en las tres secciones del negocio: Terraza, bar, sala.
- Hacer uso de la categoría de los productos.
- Opción de traspasar mesa. Cuando unos clientes se mueven de una mesa a otra.

- Cuando se va a cobrar una mesa añadir un botón que permita dividir la cuenta según el número que se indique.
- Introducir cantidad personalizada de dinero cuando se cobra.
- Implementar tres nuevas rutas, que permitan añadir, modificar o eliminar productos, mesas y clientes respectivamente.
- En lo referente a productos, también implementar una opción para subir una foto, esto hará más sencillo para el usuario el localizar productos.
- Implementar sistema distribuido para tomar nota: El jefe de sala tomará la comanda a una mesa y será notificado en barra (para la bebida), en cocina (para los platos de cocina) y en la brasa (para los platos de carne a la brasa).
- Línea de comunicación entre los cocineros de brasa y cocina para la coordinación en los platos.
- Añadir rutas que permitan hacer entrada o salida de caja, así como la vista del resumen del día para el conteo.
- Web simple que muestre el menú del día mediante el escaneo de código QR.
- Lista de la compra / Inventario (pendiente consultar con el jefe de sala para obtener los detalles).
- Histórico de tickets.
- Histórico de cajas.
- Autenticación por usuarios.

8. Bibliografía

[1] "¿Qué es React Native?," *Deloitte Spain*. <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html> (accessed Sep. 22, 2021).

[2] "Qué es Ionic: ventajas y desventajas de usarlo en apps móviles híbridas," *Profile Software Services*, Feb. 22, 2021. <https://profile.es/blog/que-es-ionic/> (accessed Sep. 22, 2021).

[3] Andrés Peña, "Xamarin: ¿Qué es y para qué sirve?," *Formadores IT*, Jun. 22, 2020. <http://www.formadoresit.es/xamarin-que-es-y-para-que-sirve/> (accessed Sep. 22, 2021).

[4] "✓ Qué es Flutter y por qué utilizarlo en la creación de tus apps," *Quality Devs*, Jul. 05, 2019. <https://www.qualitydevs.com/2019/07/05/que-es-flutter/> (accessed Sep. 22, 2021).

[5] "Node.js," *Wikipedia, la enciclopedia libre*. Aug. 05, 2021. Accessed: Sep. 19, 2021. [Online]. Available: <https://es.wikipedia.org/w/index.php?title=Node.js&oldid=137486573>

[6] Node.js, "Acerca," *Node.js*. <https://nodejs.org/es/about/> (accessed Sep. 22, 2021).

[7] D. Dvorski, "Installing, Configuring, and Developing with XAMPP," p. 10.

[8] "Postman API Platform," *Postman*. <https://www.postman.com/product/what-is-postman/> (accessed Sep. 19, 2021).

[9] P. V. Cuervo, "¿Qué es Postman?," *Arquitecto IT*. <https://www.arquitectoit.com/postman/que-es-postman/> (accessed Sep. 22, 2021).

[10] "Meet Android Studio," *Android Developers*. <https://developer.android.com/studio/intro> (accessed Sep. 19, 2021).

[11] "Git." <https://git-scm.com/> (accessed Sep. 21, 2021).

[12] Atlassian, "Qué es Git: conviértete en todo un experto en Git con esta guía," *Atlassian*. <https://www.atlassian.com/es/git/tutorials/what-is-git> (accessed Sep. 22, 2021).

[13] "Project structure for an Express REST API when there is no 'standard way' – Corey Cleary." <https://www.coreycleary.me/project-structure-for-an-express-rest-api-when-there-is-no-standard-way> (accessed Sep. 22, 2021).

[14] G. Hayes, *funwithflutter/flutter_ui_tips*. 2021. Accessed: Sep. 23, 2021. [Online]. Available: https://github.com/funwithflutter/flutter_ui_tips

9. Anexos

9.1. Anexo I – Código *backend*

9.1.1 db.config.js

```
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "",
  DB: "hmanage"
}
```

9.1.2 cash.count.controller.js

```
const CashCount = require("../models/cash_count.model.js");

// Create and Save a new CashCount
exports.create = (req, res) => {
  CashCount.init((err, data) => {
    if (err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the CashCount."
      });
    else res.send(data);
  });
};

// Retrieve all CashCounts from the database
exports.findAll = (req, res) => {
  CashCount.getAll((err, data) => {
    if (err){
      res.status(500).send({
        message:
          err.message || "Some error occurred while retrieving cash counts."
      });
    } else {
      res.send(data);
    }
  });
};
```

```

}

// Find a single Cash count with a cashCountDate
exports.findOne = (req, res) => {
  CashCount.findByDate(req.params.cashCountDate, (err, data) => {
    if (err) {
      if (err.kind === "not_found") {
        res.status(404).send({
          message: `No Cash count found with id ${req.params.cashCountDate}`
        });
      } else {
        res.status(500).send({
          message: "Error retrieving Cash count with id " +
            req.params.cashCountDate
        });
      }
    } else res.send(data);
  });
}

// Update a Cash count identified by the cashCountDate in the request
exports.update = (req, res) => {
  // Validate request
  if(!req.body){
    res.status(400).send({
      message: "Content cannot be empty!"
    });
  }

  CashCount.updateByDate(
    req.params.cashCountDate,
    new CashCount(req.body),
    (err, data) => {
      if(err) {
        if (err.kind === "not_found") {
          res.status(404).send({
            message: `No Cash count found with Date ${req.params.cashCountDate}`
          });
        } else {

```

```

        res.status(500).send({
            message: "Error updating Cash count with id
" + req.params.cashCountDate
        });
    }
    } else res.send(data);

}

);
}

// Delete a Cash count with the specified cashCountDate in the request
exports.delete = (req, res) => {
    CashCount.remove(req.params.cashCountId, (err, data) => {
        if (err) {
            if (err.kind === "not_found") {
                res.status(404).send({
                    message: `No user found with id ${req.params.cashCountId}`
                });
            } else {
                res.status(500).send({
                    message: "Could not delete User with id " + req.params.cashCountId
                });
            }
        } else res.send({ message: "User was deleted successfully!" });
    });
}
}

```

9.1.3 product.controller.js

```

const Product = require("../models/product.model.js");

// Create and Save a new Product
exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content cannot be empty!"
        });
    }
}

```



```

}

//Create a Product
const product = new Product({
  name: req.body.name,
  price: req.body.price,
  category: req.body.category
});

// Save Product in the database
Product.create(product, (err, data) => {
  if (err) {
    res.status(500).send({
      message:
        err.message || "Some error occurred while creating t
he Product."
    });
  } else {
    res.send(data);
  }
});
}

// Retrieve all Products from the database
exports.findAll = (req, res) => {
  Product.getAll((err, data) => {
    if (err) {
      res.status(500).send({
        message:
          err.message || "Some error occurred while retrieving
products."
      });
    } else {
      res.send(data);
    }
  });
}

// Find a single Product with the productId
exports.findOne = (req, res) => {
  Product.findById(req.params.productId, (err, data) => {
    if (err) {

```

```

        if (err.kind == "not_found") {
            res.status(404).send({
                message: `No Product found with id ${req.params.
productId}`
            });
        } else {
            res.status(500).send({
                message: `Error retrieving Product with id ${req
.params.productId}`
            });
        }
    } else {
        res.send(data);
    }
});
}

// Update a Product identified by the productId
exports.update = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content cannot be empty"
        });
    }

    Product.updateById(
        req.params.productId,
        new Product(req.body),
        (err, data) => {
            if (err) {
                if (err.kind === "not_found") {
                    res.status(404).send({
                        message: `No user found with id ${req.params
.productId}`
                    });
                } else {
                    res.status(500).send({
                        message: `Error updating Product with id ${r
eq.params.productId}`
                    });
                }
            }
        }
    )
}

```

```

        } else {
            res.send(data);
        }
    }
});
}

// Delete a Product with the provided productId
exports.delete = (req, res) => {
    Product.remove(req.params.productId, (err, data) => {
        if (err) {
            if (err.kind === "not_found") {
                res.status(404).send({
                    message: `No product found with id ${req.params.productId}`
                });
            } else {
                res.status(500).send({
                    message: `Could not delete Product with id ${req.params.productId}`
                });
            }
        } else {
            res.send({ message: "Product was deleted!" });
        }
    });
}
}

```

9.1.4 table.controller.js

```

const Table = require("../models/table.model.js");

// Create and Save a new Table
exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content cannot be empty"
        });
    }

    // Create a Table
    const table = new Table({

```

```

        number: req.body.number,
        total: req.body.total,
    });

    //Save Table in the DB
    Table.create(table, (err, data) => {
        if (err) {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating t
he Table."
            });
        } else {
            res.send(data);
        }
    });
}

// Retrieve all Tables from the DB
exports.findAll = (req, res) => {
    Table.getAll((err, data) => {
        if (err) {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while retrieving
tables."
            });
        } else {
            res.send(data);
        }
    });
}

// Find a single Table with the number
exports.findOne = (req, res) => {
    Table.findByNumber(req.params.tableNumber, (err, data) => {
        if (err) {
            if (err.king == "not_found") {
                res.status(404).send({
                    message: `No table found with number ${req.param
s.tableNumber}`
                });
            }
        }
    });
}

```

```

        } else {
            res.status(500).send({
                message: `Error retrieveing Table with number ${
req.params.tableNumber}`
            });
        }
    } else {
        res.send(data);
    }
});
}

// Updates the total money to pay in a table
exports.update = (req, res) => {
    // Validate request
    if(!req.body){
        res.status(400).send({
            message: "Content cannot be empty!"
        });
    }
    Table.updateByTableNumber(new Table(req.body), (err, data) => {
        if (err) {
            if (err.kind == "not_found") {
                res.status(404).send({
                    message: `No Tablefound with number ${req.params
.number}`
                });
            } else {
                res.status(500).send({
                    message: "Error updating Table with number " + r
eq.params.number
                });
            }
        } else res.send(data);
    });
}

// Delete a Table with the provided tableNumber
exports.delete = (req, res) => {
    Table.remove(req.params.tableNumber, (err, data) => {
        if (err){

```

```

        if(err.kind === "not_found"){
            res.status(404).send({
                message: `No table found with number ${req.param
s.tableNumber}`
            });
        } else {
            res.status(500).send({
                message: `Could not delete Table with number ${r
eq.params.tableNumber}`
            });
        }
    } else {
        res.send({ message: "Table was deleted!"});
    }
});
}

```

9.1.5 tbill.controller.js

```

const Tbill = require("../models/tbill.model.js");

// Create and Save a new Tbill
exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content cannot be empty!"
        });
    }

    // Create a Tbill
    const tbill = new Tbill({
        tnumber: req.body.tnumber,
        item: req.body.item,
        units: req.body.units,
        iprice: req.body.iprice,
        total: req.body.total,
    });

    // Save Tbill in the DB
    Tbill.create(tbill, (err, data) => {
        if (err)
            res.status(500).send({

```

```

        message:
            err.message || "Some error occurred while creatin
g the Tbill."
    });
    else res.send(data);
});
}

// Retrieve all Tbills from the given table
exports.findOne = (req, res) => {
    Tbill.findTableBill(req.params.tnumber, (err, data) => {
        if (err) {
            if (err.kind == "not_found") {
                res.status(404).send({
                    message: `Not Tbills found with table number: ${
req.params.tnumber}`.
                });
            } else {
                res.status(500).send({
                    message: "Error retrieving Tbills with table numb
er: " + req.params.tnumber
                });
            }
        } else res.send(data);
    });
}

// Retrieve all Tbills from all DB
exports.findAll = (req, res) => {
    Tbill.getAll((err, data) => {
        if (err){
            res.status(500).send({
                message:
                    err.message || "Some error occurred while retriee
ving Tbills."
            });
        } else {
            res.send(data);
        }
    });
}
}

```

```

// Update a Tbill identified by the id in the request
exports.update = (req, res) => {
  // Validate request
  if (!req.body) {
    res.status(400).send({
      message: "Content cannot be empty!"
    });
  }

  Tbill.updateById(
    req.params.tbillId,
    new Tbill(req.body),
    (err, data) => {
      if (err) {
        if (err.kind === "not_found") {
          res.status(404).send({
            message: `No Tbill found with id ${req.param
s.tbillId}`
          });
        } else {
          res.status(500).send({
            message: "Error updating Tbill with id " + r
eq.params.tbillId
          });
        }
      } else res.send(data);
    }
  );
}

// Delete a Tbill with the specified tbillId in the request
exports.delete = (req, res) => {
  Tbill.remove(req.params.tnumber, (err, data) => {
    if (err) {
      if (err.kind === "not_found") {
        res.status(404).send({
          message: `No tbills found with number ${req.para
ms.tnumber}`
        });
      } else {
        res.status(500).send({

```



```

                message: "Could not delete Tbills with tnumber "
+ req.params.tnumber
                });
            }
        } else res.send({ message: "Tbills were deleted successfully
!"});
    });
}

```

9.1.6 user.controller.js

```

const User = require("../models/user.model.js");

// Create and Save a new User
exports.create = (req, res) => {
    // Validate request
    if (!req.body) {
        res.status(400).send({
            message: "Content cannot be empty!"
        });
    }

    // Create a User
    const user = new User({
        login: req.body.login,
        name: req.body.name,
        email: req.body.email,
        passwd: req.body.passwd
    });

    // Save User in the database
    User.create(user, (err, data) => {
        if (err)
            res.status(500).send({
                message:
                    err.message || "Some error occured while creatin
g the User."
            });
        else res.send(data);
    });
}

// Retrieve all Users from the database

```

```

exports.findAll = (req, res) => {
  User.getAll((err, data) => {
    if (err){
      res.status(500).send({
        message:
          err.message || "Some error occurred while retrieving users."
      });
    } else {
      res.send(data);
    }
  });
}

// Find a single User with a userId
exports.findOne = (req, res) => {
  User.findById(req.params.userId, (err, data) => {
    if (err) {
      if (err.kind == "not_found") {
        res.status(404).send({
          message: `Not User found with id ${req.params.userId}.`
        });
      } else {
        res.status(500).send({
          message: "Error retrieving User with id " + req.params.userId
        });
      }
    } else res.send(data);
  });
}

// Update a User indentified by the userId in the request
exports.update = (req, res) => {
  // Validate Request
  if (!req.body) {
    res.status(400).send({
      message: "Content cannot be empty!"
    });
  }
}

```

```

    User.updateById(
      req.params.userId,
      new User(req.body),
      (err, data) => {
        if(err) {
          if (err.kind === "not_found") {
            res.status(404).send({
              message: `No User found with id ${req.params
                .userId}`
            });
          } else {
            res.status(500).send({
              message: "Error updating User with id " + req
                .params.userId
            });
          }
        } else res.send(data);
      }
    );
  }
}

// Delete a User with the specified userId in the request
exports.delete = (req, res) => {
  User.remove(req.params.userId, (err, data) => {
    if (err) {
      if (err.kind === "not_found") {
        res.status(404).send({
          message: `No user found with id ${req.params.use
            rId}`
        });
      } else {
        res.status(500).send({
          message: "Could not delete User with id " + req.
            params.userId
        });
      }
    } else res.send({ message: "User was deleted successfully!"}
    );
  });
}
}

```

9.1.7 cash_count.model.js

```
const { json } = require("body-parser");
const sql = require("../db.js");

// Constructor
const CashCount = function(cashCount) {
  this.day = cashCount.day;
  this.netSale = cashCount.netSale;
  this.cardPayments = cashCount.cardPayments;
  this.cashPayments = cashCount.cashPayments;
  this.numberSales = cashCount.numberSales;
  this.averageTicket = cashCount.averageTicket;
  this.income = cashCount.income;
  this.outflow = cashCount.outflow;
}

CashCount.init = result => {
  sql.query(`INSERT INTO cash_counts (day, net_sale, card_payments
, cash_payments, number_sales, average_ticket, income, outflow)
VALUES (curdate(), 0, 0, 0, 0, 0, 0, 0)
ON DUPLICATE KEY UPDATE id=id`, (err, res) => {
    if(err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }
    console.log("Cash count created: ", {id: res.insertId});
    result(null, {id: res.insertId});
  });
}

CashCount.findByDate = (cashCountDate, result) => {
  sql.query(`SELECT * FROM cash_counts WHERE day = ${cashCountDate}
`, (err, res) => {
    if(err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }
  })
}
```

```

    if (res.length) {
      console.log("Found product: ", res[0]);
      result(null, res[0]);
      return;
    }

    // No entry found
    result({ kind: "not_found" }, null);
  });
}

CashCount.getAll = result => {
  sql.query("SELECT * FROM cash_counts", (err, res) => {
    if (err) {
      console.log("error: ", err);
      result(null, err);
      return;
    }
    console.log("cash_count: ", res);
    result(null, res);
  });
}

CashCount.updateByDate = (cashCountDate, cashCount, result) => {
  sql.query(
    `UPDATE cash_counts SET
    net_sale = COALESCE(net_sale + ?, net_sale),
    card_payments = COALESCE(card_payments + ?, card_payments),
    cash_payments = COALESCE(cash_payments + ?, cash_payments),
    number_sales = COALESCE(number_sales + ?, number_sales),
    average_ticket = COALESCE(?, average_ticket),
    income = COALESCE(income + ?, income),
    outflow = COALESCE(outflow + ?, outflow)
    WHERE day = ?`,
    [cashCount.netSale, cashCount.cardPayments, cashCount.cashPayments, cashCount.numberSales, cashCount.averageTicket, cashCount.income, cashCount.outflow, cashCountDate],
    (err, res) => {
      if (err) {
        console.log("error: ", err);
        result(null, err);
        return;
      }
    }
  );
}

```

```

    }

    if (res.affectedRows == 0) {
        //No Cash count found with the id provided
        result({ kind: "not_found"}, null);
        return;
    }

    console.log("Cash count updated: ", {id: res.insertId, .
    ..cashCount});
    result(null, {id: res.insertId, ...cashCount});

    });
}

CashCount.remove = (id, result) => {
    sql.query("DELETE FROM cash_counts WHERE id = ?", id, (err, res)
    => {
        if (err) {
            console.log("error: ", err);
            result(null, err);
            return;
        }

        if (res.affectedRows == 0) {
            // no Cash count found with that id
            result({ kind: "not_found" }, null);
            return;
        }

        console.log("deleted Cash count with id: ", id);
        result(null, res);
    });
}

module.exports = CashCount;

```

9.1.8 product.model.js

```

const { json } = require("body-parser");
const sql = require("../db.js");

// Constructor

```

```

const Product = function(product) {
  this.name = product.name;
  this.price = product.price;
  this.category = product.category;
}

Product.create = (newProduct, result) => {
  var name = newProduct.name;
  var price = newProduct.price;
  var category = newProduct.category;

  sql.query(`INSERT INTO products (name, price, category)
    VALUES ('${name}', ${price}, '${category}')`, (err, res) =>
  {
    if (err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }
    console.log("Producto creado: ", {id: res.insertId, ...n
ewProduct});
    result(null, {id: res.insertId, ...newProduct});
  });
}

Product.findById = (productId, result) =>{
  sql.query(`SELECT * FROM products WHERE id = ${productId}`, (err
, res) => {
    if (err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }

    if (res.length) {
      console.log("Found product: ", res[0]);
      result(null, res[0]);
      return;
    }

    // No Product found
    result({ kind: "not_found" }, null);
  });
}

```

```

    });
}

Product.getAll = result => {
    sql.query("SELECT id, name, FORMAT(price,2) AS price, category FROM products", (err, res) => {
        if (err) {
            console.log("error: ", err);
            result(null, err);
            return;
        }
        console.log("products: ", res);
        result(null, res);
    });
}

Product.updateById = (id, product, result) => {
    sql.query(
        "UPDATE tbills SET name = ?, price = ?, category = ? WHERE id = ?",
        [product.name, product.price, product.category],
        (err, res) => {
            if (err) {
                console.log("error: ", err);
                result(null, err);
                return;
            }

            if (res.affectedRows == 0) {
                //No product found with the id provided
                result({ kind: "not_found"}, null);
                return;
            }

            console.log("product updated: ", {id: id, ...product});
            result(null, {id: id, ...product});
        }
    );
}

Product.remove = (id, result) => {

```



```

    sql.query("DELETE FROM products WHERE id = ?", id, (err, res) =>
    {
        if (err) {
            console.log("error: ", err);
            result(null, err);
            return;
        }

        if (res.affectedRows == 0) {
            // no product found with that id
            result({ kind: "not_found" }, null);
            return;
        }

        console.log("deleted user with id: ", id);
        result(null, res);
    });
}

module.exports = Product;

```

9.1.9 table.model.js

```

const { json } = require("body-parser");
const sql = require("../db.js");

// Constructor
const Table = function(table) {
    this.number = table.number;
    this.total = table.total;
}

Table.create = (newTable, result) => {
    var number = newTable.number;

    sql.query(`INSERT INTO tables (number, total) VALUES (${number},
    ${total})`, (err, res) => {
        if (err) {
            console.log("error: ", err);
            result(err, null);
            return;
        }
    })
}

```

```

        console.log("Mesa creada: ", {id: res.insertId, ...newTable}
    );
    result(null, {id: res.insertId, ...newTable});
  });
}

Table.findByNumber = (tableNumber, result) => {
  sql.query(`SELECT * FROM tables WHERE number = ${tableNumber}`,
    (err, res) => {
      if (err) {
        console.log("error: ", err);
        result(err, null);
        return;
      }

      if (res.length) {
        console.log("Found table: ", res[0]);
        result(null, res[0]);
        return;
      }

      // No Table found
      result({ kind: "not_found" }, null);
    });
}

Table.getAll = result => {
  sql.query("SELECT id, number, FORMAT(total,2) AS total FROM tabl
es", (err, res) => {
    if (err) {
      console.log("error: ", err);
      result(null, err);
      return;
    }
    console.log("tables: ", res);
    result(null, res);
  });
}

Table.updateByTableNumber = (newTable, result) => {
  sql.query(
    `UPDATE tables SET total = ? WHERE number = ?`,

```

```

    [newTable.total, newTable.number],
    (err, res) => {
      if (err) {
        console.log("error: ", err);
        result(null, err);
        return;
      }

      if (res.affectedRows == 0) {
        //No Table count found with the number provided
        result({ kind: "not_found"}, null);
        return;
      }

      console.log("Table total updated: ", {id: res.insertId,
total: newTable.total});
      result(null, {id: res.insertId, total: newTable.total});

    });
  }

Table.remove = (number, result) => {
  sql.query("DELETE FROM tables WHERE number = ?", number, (err, r
es) => {
    if (err) {
      console.log("error: ", err);
      result(null, err);
      return;
    }

    if (res.affectedRows == 0) {
      // no table found with tat number
      result ({ kind: "not_found" }, null);
      return;
    }

    console.log("deleted table with number: ", number);
    result(null, res);
  });
}

module.exports = Table

```

```

const { json } = require("body-parser");
const sql = require("../db.js");

// Constructor
const Tbill = function(tbill) {
  this.tnumber = tbill.tnumber;
  this.item = tbill.item;
  this.units = tbill.units;
  this.iprice = tbill.iprice;
  this.total = tbill.total;
}

Tbill.create = (newTbill, result) => {
  var tnumber = newTbill.tnumber;
  var item = newTbill.item;
  var units = newTbill.units;
  var iprice = newTbill.iprice;
  var total = newTbill.total;

  sql.query(`INSERT INTO tbills (tnumber, item, units, iprice, total)
VALUES (${tnumber}, '${item}', ${units}, ${iprice}, ${total})
ON DUPLICATE KEY UPDATE units = (units + 1), total = units*iprice`, (err, res) => {
    if (err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }
    console.log("Tbill created: ", {id: res.insertId, ...newTbill});
    result(null, {id: res.insertId, ...newTbill});
  });
}

Tbill.findTableBill = (tableNumber, result) => {
  sql.query(`SELECT id, tnumber, item, units, FORMAT(iprice,2) AS iprice, FORMAT(total,2) AS total FROM tbills WHERE tnumber = ${tableNumber}`, (err, res) => {
    if (err) {

```

```

        console.log("error: ", err);
        result(err, null);
        result;
    }

    if (res.length) {
        console.log("found product(s): ", res);
        result(null, res);
        return;
    }

    result({ kind: "not_found"}, null);
});
}

Tbill.getAll = result => {
    sql.query("SELECT id, tnumber, item, units, FORMAT(iprice,2) AS
iprice, FORMAT(total,2) AS total FROM tbills", (err, res) => {
        if (err) {
            console.log("error: ", err);
            result(null, err);
            return;
        }

        console.log("tbills: ", res);
        result(null, res);
    });
}

Tbill.updateById = (id, tbill, result) => {
    sql.query(
        `UPDATE tbills SET
        tnumber = COALESCE(?, tnumber),
        item = COALESCE(?, item),
        units = COALESCE(?, units),
        iprice = COALESCE(?, iprice),
        total = COALESCE(?, total)
        WHERE id = ?`,
        [tbill.tnumber, tbill.item, tbill.units, tbill.iprice, tbill
.total, id],
        (err, res) => {
            if (err) {

```

```

        console.log("error: ", err);
        result(null, err);
        return;
    }

    if (res.affectedRows == 0) {
        // no tbill found with that Id
        result({ kind: "not_found" }, null);
        return;
    }

    console.log("Updated tbill: ", { id: id, ...tbill });
    result(null, {id: id, ...tbill});
}
);
}

Tbill.remove = (tnumber, result) => {
    sql.query("DELETE FROM tbills WHERE tnumber = ?", tnumber, (err,
    res) => {
        if (err) {
            console.log("error: ", err);
            result(null, err);
            return;
        }

        if (res.affectedRows == 0) {
            result({ kind: "not_found" }, null);
            return;
        }

        console.log("deleted tbills with Table number: ", tnumber);
        result(null, res);
    });
}

module.exports = Tbill;

```

9.1.11 user.model.js

```

const { json } = require("body-parser");
const sql = require("../db.js");

```

```

//constructor
const User = function(user) {
  this.login = user.login;
  this.name = user.name;
  this.email = user.email;
  this.passwd = user.passwd;
}

User.create = (newUser, result) => {
  var login = newUser.login;
  var name = newUser.name;
  var email = newUser.email;
  var passwd = newUser.passwd;
  sql.query(`INSERT INTO users (login, name, email, passwd)
    VALUES ('${login}', '${name}', '${email}', '${passwd}')`, (err
, res) => {
    if (err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }
    console.log("User created: ", {id: res.insertId, ...newUser}
);
    result(null, { id: res.insertId, ...newUser});
  });
}

User.findById = (userId, result) => {
  sql.query(`SELECT * FROM users WHERE id = ${userId}`, (err, res) =
> {
    if (err) {
      console.log("error: ", err);
      result(err, null);
      return;
    }

    if (res.length) {
      console.log("found user: ", res[0]);
      result(null, res[0]);
      return;
    }
  }
}

```

```

    // no User found with the id
    result({ kind: "not_found" }, null);
  });
}

User.getAll = result => {
  sql.query("SELECT * FROM users", (err, res) => {
    if (err) {
      console.log("error: ", err);
      result(null, err);
      return;
    }

    console.log("users: ", res);
    result(null, res);
  });
}

User.updateById = (id, user, result) => {
  sql.query(
    "UPDATE users SET login = ?, name = ?, passwd = ? WHERE id = ?",
    [user.login, user.name, user.passwd, id],
    (err, res) => {
      if (err) {
        console.log("error: ", err);
        result(null, err);
        return;
      }

      if (res.affectedRows == 0) {
        // not found User with the id
        result({ kind: "not_found" }, null);
        return;
      }

      console.log("updated user: ", { id: id, ...user });
      result(null, { id: id, ...user });
    }
  );
}

User.remove = (id, result) => {

```



```

sql.query("DELETE FROM users WHERE id = ?", id, (err, res) => {
  if (err) {
    console.log("error: ", err);
    result(null, err);
    return;
  }

  if (res.affectedRows == 0) {
    // not found User with the id
    result({ kind: "not_found" }, null);
    return;
  }

  console.log("deleted user with id: ", id);
  result(null, res);
});
}

module.exports = User;

```

9.1.12 db.js

```

const mysql = require("mysql");
const dbConfig = require("../config/db.config.js");

//Create a connection to the database
const connection = mysql.createConnection({
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  database: dbConfig.DB
});

//open the MySQL connection
connection.connect(error => {
  if (error) throw error;
  console.log("Successfully connected to the databse.");
});

module.exports = connection;

```

9.1.13 cash_count.routes.js

```

module.exports = app => {

```

```

    const cash_counts = require("../controllers/cash_count.controller.js");

    // Create a new Cash count
    app.post("/cash_counts", cash_counts.create);

    // Retrieve all Cash counts
    app.get("/cash_counts", cash_counts.findAll);

    // Retrieve a single Cash count with Date
    app.get("/cash_counts/:cashCountDate", cash_counts.findOne);

    // Update a Cash count with cashCountDate
    app.put("/cash_counts/:cashCountDate", cash_counts.update);

    // Delete a Cash count with cashCountId
    app.delete("/cash_counts/:cashCountId", cash_counts.delete);
}

```

9.1.14 product.routes.js

```

module.exports = app => {
  const products = require("../controllers/product.controller.js");
;

  //Create a new Product
  app.post("/products", products.create);

  //Retrieve all Products
  app.get("/products", products.findAll);

  //Retrieve a single User with userId
  app.get("/products/:productId", products.findOne);

  //Update a User with userId
  app.put("/products/:productId", products.update);

  //Delete a User with userId
  app.delete("/products/:productId", products.delete);
}

```

9.1.15 table.routes.js

```
module.exports = app => {
  const tables = require("../controllers/table.controller.js");

  // Create a new Table
  app.post("/tables", tables.create);

  // Retrieve all Table
  app.get("/tables", tables.findAll);

  // Retrieve a single Table with userId
  app.get("/tables/:tableNumber", tables.findOne);

  // Update a Table total with provided tableNumber
  app.put("/tables/:number", tables.update);

  // Delete a Table with userId
  app.delete("/tables/:tableNumber", tables.delete);
}
```

9.1.16 tbill.routes.js

```
module.exports = app => {
  const tbills = require("../controllers/tbill.controller.js");

  // Create a new Tbill
  app.post("/tbills", tbills.create);

  // Retrieve all Tbills
  app.get("/tbills", tbills.findAll);

  // Retrieve Tbills from a table with tableNumber
  app.get("/tbills/:tnumber", tbills.findOne);

  // Update a Tbill with tbillId
  app.put("/tbills/:tbillId", tbills.update);

  // Delete a Tbill with tnumber
  app.delete("/tbills/:tnumber", tbills.delete);
}
```

9.1.17 user.routes.js

```
module.exports = app => {
  const users = require("../controllers/user.controller.js");

  //Create a new User
  app.post("/users", users.create);

  //Retrieve all Users
  app.get("/users", users.findAll);

  //Retrieve a single User with userId
  app.get("/users/:userId", users.findOne);

  //Update a User with userId
  app.put("/users/:userId", users.update);

  //Delete a User with userId
  app.delete("/users/:userId", users.delete);
}
```

9.1.18 server.js

```
const express = require('express');

const app = express();

//parse request of content-type: application/json
app.use(express.json());

//parse requests of content-type: application/x-www-form-urlencoded
app.use(express.urlencoded({
  extended: true
}));

//Includes
require("../api/routes/user.routes.js")(app);
require("../api/routes/product.routes.js")(app);
require("../api/routes/table.routes.js")(app);
require("../api/routes/tbill.routes.js")(app);
require("../api/routes/cash_count.routes.js")(app);

//set port, listen for requests
```

```
app.listen(8888, () => {
  console.log("Server is running on port 8888");
});
```

9.2. Anexo II – Código de aplicación

9.2.1 data.dart

```
// Creating a User Object
class User {
  final int id;
  final String login;
  final String name;
  final String email;

  const User({
    required this.id,
    required this.login,
    required this.name,
    required this.email,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] as int,
      login: json['login'] as String,
      name: json['name'] as String,
      email: json['email'] as String,
    );
  }
}

// Creating a Product object
class Product {
  final int id;
  final String name;
  final String price;
  final String category;

  const Product({
    required this.id,
    required this.name,
    required this.price,
    required this.category,
  });

  factory Product.fromJson(Map<String, dynamic> json) {
    return Product(
```

```

        id: json['id'] as int,
        name: json['name'] as String,
        price: json['price'] as String,
        category: json['category'] as String,
    );
}
}

// Create a Tablee object
class Tablee {
    final int id;
    final int number;
    final String total;

    const Tablee({
        required this.id,
        required this.number,
        required this.total,
    });

    factory Tablee.fromJson(Map<String, dynamic> json) {
        return Tablee(
            id: json['id'] as int,
            number: json['number'] as int,
            total: json['total'] as String,
        );
    }
}

// Create a Tbill object
class Tbill {
    final int id;
    final int? tnumber;
    final String item;
    final int units;
    final String iprice;
    final String total;

    const Tbill({
        required this.id,
        required this.tnumber,
        required this.item,
        required this.units,
        required this.iprice,
        required this.total,
    });

    factory Tbill.fromJson(Map<String, dynamic> json) {
        return Tbill(
            id: json['id'] as int,
            tnumber: json['tnumber'] as int?,

```

```

        item: json['item'] as String,
        units: json['units'] as int,
        iprice: json['iprice'] as String,
        total: json['total'] as String,
    );
}
}

// Create a CashCount object
class CashCount {
    final String day;
    final String? netSale;
    final String? cardPayments;
    final String? cashPayments;
    final int? numberSales;
    final String? averageTicket;
    final String? income;
    final String? outflow;

    const CashCount({
        required this.day,
        required this.netSale,
        required this.cardPayments,
        required this.cashPayments,
        required this.numberSales,
        required this.averageTicket,
        required this.income,
        required this.outflow,
    });

    factory CashCount.fromJson(Map<String, dynamic> json) {
        return CashCount(
            day: json['day'] as String,
            netSale: json['net_sale'] as String?,
            cardPayments: json['card_payments'] as String?,
            cashPayments: json['cash_payments'] as String?,
            numberSales: json['number_sales'] as int?,
            averageTicket: json['average_ticket'] as String?,
            income: json['income'] as String?,
            outflow: json['outflow'] as String?,
        );
    }
}
}

```

9.2.2 server_request.dart

```

import 'package:h_manage/data.dart';
import 'package:flutter/foundation.dart';
import 'dart:async';
import 'dart:convert';
import 'package:http/http.dart' as http;

```

```

// TODO: Update server model request regarding format
(DECIMAL(10,2))

class ServerRequest {
    static String _host = 'http://192.168.1.134:8888/';
    // Request to obtain all tables
    static Future<List<Tablee>> fetchTables(http.Client client)
    async {
        print('Doing the table fetchFunction');
        final response = await client.get(Uri.parse(_host +
'tables'));

        // Using the compute function to run parseTables in a
separate isolate
        return compute(parseTables, response.body);
    }

    static List<Tablee> parseTables(String responseBody) {
        final parsed = jsonDecode(responseBody).cast<Map<String,
dynamic>>();

        return parsed.map<Tablee>((json) =>
Tablee.fromJson(json)).toList();
    }

    // Function that retrieves all the products from the server
    static Future<List<Product>> fetchProducts(http.Client
client) async {
        print('Doing a products fetchFunction');
        final response = await client.get(Uri.parse(_host +
'products'));

        // Using the compute function to run parseProducts in a
separate isolate
        return compute(parseProducts, response.body);
    }

    // Function that converts a response body into a
List<Product>
    static List<Product> parseProducts(String responseBody) {
        final parsed = jsonDecode(responseBody).cast<Map<String,
dynamic>>();

        return parsed.map<Product>((json) =>
Product.fromJson(json)).toList();
    }

    static Future<Tbill> createTbill(
        int tnumber, String item, int units, String iprice,
String total) async {
        final response = await http.post(

```



```

    Uri.parse(_host + 'tbills'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, dynamic>{
      'tnumber': tnumber,
      'item': item,
      'units': units,
      'iprice': iprice,
      'total': total,
    })),
  );

  if (response.statusCode == 200) {
    return Tbill.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to create tbill');
  }
}

static Future<List<Tbill>> fetchTbills(http.Client client)
async {
  final response = await client.get(Uri.parse(_host +
'tbills/'));
  return compute(parseTbills, response.body);
}

static List<Tbill> parseTbills(String responseBody) {
  final parsed = jsonDecode(responseBody).cast<Map<String,
dynamic>>();
  print(responseBody.toString());
  return parsed.map<Tbill>((json) =>
Tbill.fromJson(json)).toList();
}

static updateTbill(
  int tbillId,
  int? tnumber,
  String item,
  int units,
  String iprice,
  String total) async {
  final response = await http.put(
    Uri.parse(_host + "tbills/" + tbillId.toString()),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, dynamic>{
      'tbillId': tbillId,
      'tnumber': tnumber,
      'item': item,

```

```

        'units': units,
        'iprice': iprice,
        'total': total,
    })),
);

if (response.statusCode != 200)
    throw Exception('Failed to update tbill');
}

static Future<List<Tbill>> fetchTableTbills(
    http.Client client, String table) async {
    final response = await client.get(Uri.parse(_host +
'tbills/' + table));
    if (response.statusCode == 404) {
        return [];
    }
    return compute(parseTbills, response.body);
}

static Future<List<Tbill>> newfetchTableTbills(String table)
async {
    final response = await http.get(Uri.parse(_host +
'tbills/' + table));
    if (response.statusCode == 404) {
        return [];
    }
    return compute(parseTbills, response.body);
}

static Future createCashCount() async {
    final response = await http.post(Uri.parse(_host +
'cash_counts'));

    if (response.statusCode != 200)
        throw Exception('Failed to create cash count');
}

static Future<CashCount> updateCashCount(
    String? netSale,
    String? cardPayments,
    String? cashPayments,
    int? numberSales,
    String? averageTicket,
    String? income,
    String? outflow,
    String day) async {
    final response = await http.put(
        Uri.parse(_host + 'cash_counts/' + day),
        headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',

```

```

    },
    body: jsonEncode(<String, dynamic>{
      'netSale': netSale,
      'cardPayments': cardPayments,
      'cashPayments': cashPayments,
      'numberSales': numberSales,
      'averageTicket': averageTicket,
      'income': income,
      'outflow': outflow,
      'day': day,
    })),
  );

  if (response.statusCode == 200) {
    return CashCount.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to update CashCount');
  }
}

static Future deleteTbill(String tnumber) async{
  final response = await http.delete(Uri.parse(_host +
'tbills/' + tnumber));
  if (response.statusCode != 200)
    throw Exception('Failed to Delete tbillls');
}

static Future updateTable(int number, String total,) async {
  final response = await http.put(
    Uri.parse(_host + 'tables/' + number.toString()),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, dynamic>{
      'number': number,
      'total': total,
    })),
  );

  if (response.statusCode != 200) {
    throw Exception('Failed to update Table');
  }
}
}

```

9.2.3 route_generator.dart

```

import 'package:flutter/material.dart';
import 'package:h_manage/pages/pages.dart';
import 'package:h_manage/pages/sixth_page.dart';

```

```

class RouteGenerator {
    static Route<dynamic> generateRoute(RouteSettings settings)
    {
        // Getting arguments passed in while calling
Navigator.pushNamed
        final args = settings.arguments;

        switch (settings.name) {
            case '/':
                return MaterialPageRoute(builder: (_) => FirstPage());

            case '/second':
                // Validation of correct data type
                if (args is String) {
                    return MaterialPageRoute(
                        builder: (_) => SecondPage(
                            data: args,
                        ));
                }
                // If args is not of the correct type, return an error
page.
                // While in development you can also throw an
exception
                return _errorRoute();

            case '/third':
                return MaterialPageRoute(
                    builder: (_) => ThirdPage(
                        title: "TITLE PAGE 3",
                    ));

            case '/fourth':
                return MaterialPageRoute(
                    builder: (_) => FourthPage(
                        title: "title fourth",
                    ));

            case '/fifth':
                return MaterialPageRoute(
                    builder: (_) => FifthPage(
                        data: "TITLE PAGE 5",
                    ));

            case '/sixth':
                if (args is List<num>) {
                    return MaterialPageRoute(
                        builder: (_) =>
                            SixthPage(
                                total: args[0],
                                tnumber: args[1],
                            ));
                }
            }
        }
    }
}

```

```

    }
    return _errorRoute();

    case '/seventh':
      return MaterialPageRoute(
        builder: (_) => SeventhPage(
          title: "title for seventh",
        ));

    case '/fifth/edit_bill_product':
      if (args is List<dynamic>) {
        return MaterialPageRoute(builder: (_) =>
          EditBillProduct(
            id: args[0],
            units: args[1],
            product: args[2],
            price: args[3],
            total: args[4],
          ));
      }
      return _errorRoute();

    default:
      // If there is no such named route in the switch
statement, eg /third
      return _errorRoute();
  }
}

static Route<dynamic> _errorRoute() {
  return MaterialPageRoute(builder: (_) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Error'),
      ),
      body: Center(
        child: Text('ERROR'),
      ),
    );
  });
}
}

```

9.2.4 main.dart

```

import 'package:flutter/material.dart';

import 'package:h_manage/route_generator.dart';

void main() => runApp(MyApp());

```

```

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      initialRoute: '/',
      onGenerateRoute: RouteGenerator.generateRoute,
    );
  }
}

```

9.2.5 pages.dart

```

export 'first_page.dart';
export 'second_page.dart';
export 'third_page.dart';
export 'fourth_page.dart';
export 'fifth_page.dart';
export 'sixth_page.dart';
export 'seventh_page.dart';
export 'edit_bill_product.dart';

```

9.2.6 first_page.dart

```

import 'package:flutter/material.dart';

class FirstPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('H MANAGE'),
      ),
      body: Center(
        child: Wrap(
          spacing: 8.0,
          runSpacing: 8.0,
          children: <Widget>[
            ElevatedButton(
              child: Text('Go to second'),
              onPressed: () {
                Navigator.of(context).pushNamed('/second',
                  arguments: 'Hello there from the first
page');
              },
            ),
          ],
        ),
      ),
    );
  }
}

```

```

        ElevatedButton(
          child: Text('Go to third'),
          onPressed: () {
            Navigator.of(context).pushNamed('/third',
              arguments: 'You are now on third page, hi
from page One');
          },
        ),
        ElevatedButton(
          child: Text('Go to fourth'),
          onPressed: () {
            Navigator.of(context).pushNamed('/fourth');
          },
        ),
        ElevatedButton(
          child: Text('Go to fifth'),
          onPressed: () {
            Navigator.of(context).pushNamed('/fifth');
          },
        ),
        ElevatedButton(
          child: Text('Go to sixth'),
          onPressed: () {
            Navigator.of(context).pushNamed('/sixth');
          },
        ),
        ElevatedButton(
          child: Text('Go to seventh'),
          onPressed: () {
            Navigator.of(context).pushNamed('/seventh');
          },
        ),
      ],
    ),
  ),
);
}
}

```

9.2.7 second_page.dart

```

import 'package:flutter/material.dart';

class SecondPage extends StatelessWidget {
  final String data;

  SecondPage({Key? key, required this.data}) : super(key:
key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(

```

```

    appBar: AppBar(
      title: Text('TITLE PAGE 2'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: <Widget>[
          Text(
            'PLACE 2',
            style: TextStyle(fontSize: 45),
          ),
          Text(
            data,
            style: TextStyle(fontSize: 22),
          ),
          ElevatedButton(
            child: Text('Go to third'),
            onPressed: () {
              Navigator.of(context).pushNamed('/third',
                arguments: 'You are now on third page, hi
from page One');
            },
          ),
        ],
      ),
    ),
  );
}

```

9.2.8 third_page.dart

```

import 'dart:async';
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:flutter/foundation.dart';
import 'package:http/http.dart' as http;

import 'package:h_manage/data.dart';

// A function that retrieves the users from the server
Future<List<User>> fetchUsers(http.Client client) async {
  final response =
    await
client.get(Uri.parse('http://192.168.1.134:8888/users'));

  // Using the compute function to run parseUsers in a
  separate isolate
  return compute(parseUsers, response.body);
}

```



```

// A function that converts a response body into a List<User>
List<User> parseUsers(String responseBody) {
    final parsed = jsonDecode(responseBody).cast<Map<String,
dynamic>>();

    return parsed.map<User>((json) =>
User.fromJson(json)).toList();
}

class ThirdPage extends StatelessWidget {
    const ThirdPage({Key? key, required this.title}) :
super(key: key);

    final String title;

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text(title),
            ),
            body: FutureBuilder<List<User>>(
                future: fetchUsers(http.Client()),
                builder: (context, snapshot) {
                    if (snapshot.hasError) {
                        return const Center(
                            child: Text('An error has occurred!'),
                        );
                    } else if (snapshot.hasData) {
                        return UsersList(users: snapshot.data!);
                    } else {
                        return const Center(
                            child: CircularProgressIndicator(),
                        );
                    }
                },
            ),
        );
    }
}

class UsersList extends StatelessWidget {
    const UsersList({Key? key, required this.users}) :
super(key: key);

    final List<User> users;

    @override
    Widget build(BuildContext context) {
        return ListView.builder(
            padding: const EdgeInsets.all(16),

```

```

        itemCount: users.length,
        itemBuilder: (context, index) {
            return Text(users[index].email + ' - ' +
users[index].name);
        },
    );
}
}

```

9.2.9 fourth_page.dart

```

import 'dart:async';
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

import 'package:h_manage/data.dart';

// A function that creates a new user in the database
Future<User> createUser(
    String login, String name, String email, String passwd)
async {
    final response = await http.post(
        Uri.parse('http://192.168.1.134:8888/users'),
        headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
        },
        body: jsonEncode(<String, String>{
            'login': login,
            'name': name,
            'email': email,
            'passwd': passwd,
        })),
    );

    if (response.statusCode == 200) {
        // If the server did return a 200 OK response, then parse
the JSON
        return User.fromJson(jsonDecode(response.body));
    } else {
        // If not, then throw an exception
        throw Exception('Failed to create user');
    }
}

class FourthPage extends StatefulWidget {
    const FourthPage({Key? key, required this.title}) :
super(key: key);

    final String title;

```

```

@override
_FourthPageState createState() {
  return _FourthPageState();
}
}

class _FourthPageState extends State<FourthPage> {
  final TextEditingController _controllerLogin =
  TextEditingController();
  final TextEditingController _controllerName =
  TextEditingController();
  final TextEditingController _controllerEmail =
  TextEditingController();
  final TextEditingController _controllerPasswd =
  TextEditingController();
  Future<User>? _futureUser;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Create User '),
      ),
      body: Container(
        alignment: Alignment.center,
        padding: const EdgeInsets.all(8.0),
        margin: EdgeInsets.symmetric(
          horizontal: MediaQuery.of(context).size.width / 20,
          vertical: MediaQuery.of(context).size.width / 20,
        ),
        child: (_futureUser == null) ? buildColumn() :
buildFutureBuilder(),
      ),
    );
  }

  Column buildColumn() {
    return Column(
      //mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      mainAxisAlignment: MainAxisAlignment.max,
      children: <Widget>[
        TextField(
          textAlign: TextAlign.center,
          textCapitalization: TextCapitalization.sentences,
          controller: _controllerLogin,
          decoration: const InputDecoration(hintText: 'Enter
login'),
        ),
        TextField(
          textAlign: TextAlign.center,
          textCapitalization: TextCapitalization.sentences,

```

```

        controller: _controllerName,
        decoration: const InputDecoration(hintText: 'Enter
name'),
    ),
    TextField(
        textAlign: TextAlign.center,
        controller: _controllerEmail,
        decoration: const InputDecoration(hintText: 'Enter
email'),
    ),
    TextField(
        textAlign: TextAlign.center,
        controller: _controllerPasswd,
        decoration: const InputDecoration(hintText: 'Enter
passwd'),
    ),
    ElevatedButton(
        onPressed: () {
            setState(() {
                _futureUser = createUser(
                    _controllerLogin.text,
                    _controllerName.text,
                    _controllerEmail.text,
                    _controllerPasswd.text,
                );
            });
        },
        child: const Text('Create User'),
    ),
],
);
}

FutureBuilder<User> buildFutureBuilder() {
    return FutureBuilder<User>(
        future: _futureUser,
        builder: (context, snapshot) {
            if (snapshot.hasData) {
                return Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                        Text(snapshot.data!.login),
                        Text(snapshot.data!.name),
                        Text(snapshot.data!.email),
                    ],
                );
            } else if (snapshot.hasError) {
                return Text('${snapshot.error}');
            }
        },
    );
}

return const CircularProgressIndicator();

```

```

    },
  );
}
}

```

9.2.10 fifth_page.dart

```

import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:http/http.dart' as http;

import 'package:h_manage/data.dart';
import 'package:h_manage/route_generator.dart';
import 'package:h_manage/server_request.dart';

// TODO: Do something when Futures completes with error
// TODO: Center the text in the CircleAvatar (number of items
// badge)
// TODO: Prevent further calls when tapping the same table
// over and over
// TODO: Add a refresh button on server connection fail
// TODO: Remove the 1st tab (index 0) and move that to a
// drawer maybe
// TODO: Remove TotalTableBill in the Tables TAB when total is
// 0
// TODO: Disable edit button (tbills items) when the Table has
// paid
// TODO: Maybe add a transition to tbill-tab to prevent the
// abrupt refresh

class FifthPage extends StatefulWidget {
  final String data;

  FifthPage({Key? key, required this.data}) : super(key: key);

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

class _FifthPageState extends State<FifthPage>
  with SingleTickerProviderStateMixin {
  List<bool> _isDisabled = [false, false, true, true];
  late TabController _tabController;
  int _selectedTable = 0;
  int _itemsInTable = 0;
  num _totalTableBill = 0;
  num _money = 0;
  num _change = 0;
  bool _clearMoneyChange = false;
  late Future<List<Product>> listOfDBProducts;
  late Future<List<Tablee>> listOfDBTables;
  late Future<List<Tbill>> listOfTbills;

```

```

late Future<List<Tbill>> futureTableBill;
final GlobalKey<ScaffoldMessengerState> scaffoldMessengerKey
=
    GlobalKey<ScaffoldMessengerState>();

onTap() {
    if (!_isEnabled[_tabController.index]) {
        if (!_isEnabled[2] == true) {
            final snackBar = SnackBar(
                content: Text('Select a table'),
                duration: const Duration(milliseconds: 1500),
                behavior: SnackBarBehavior.floating,
                shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10.0),
                ),
            );

scaffoldMessengerKey.currentState!.showSnackBar(snackBar);
        }
        if (!_isEnabled[2] == false && !_isEnabled[3] == true) {
            final snackBar = SnackBar(
                content: Text('Add items to the table'),
                duration: const Duration(milliseconds: 1500),
                behavior: SnackBarBehavior.floating,
                shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10.0),
                ),
            );

scaffoldMessengerKey.currentState!.showSnackBar(snackBar);
        }

        setState(() {
            if (!_isEnabled[2] == true) _tabController.index = 1;
            if (!_isEnabled[2] == false && !_isEnabled[3] == true)
                _tabController.index = 2;
        });
    }
    if (_clearMoneyChange == true) {
        clearMoneyAndChange();
    }

    if (_tabController.index == 3) {
        refreshFuture();
        updateBadge();
    }
}

@override
void initState() {
    super.initState();
}

```

```

        futureTableBill =
ServerRequest.newfetchTableTbills(1.toString());
        listOfDBProducts =
ServerRequest.fetchProducts(http.Client());
        listOfDBTables = ServerRequest.fetchTables(http.Client());
        _tabController = TabController(length: 4, vsync: this);
        _tabController.addListener(onTap);
        ServerRequest.createCashCount();
    }

    @override
    void dispose() {
        _tabController.dispose();
        super.dispose();
    }

    // Keeps updating the selected table at the top (title)
    Widget _resolveSelectedTable() {
        if (_selectedTable == 0) {
            _isDisabled = [false, false, true, true];
            return Text('Table: Please select a table');
        }
        return Text('Table: ' + _selectedTable.toString());
    }

    // Keeps updating the n° of items on the selected table
    (Tab(Row(Text)))
    Widget _resolveNumberOfItemsInTable() {
        if (_itemsInTable == 0) {
            return Text('0', textAlign: TextAlign.center);
        }
        return Text(_itemsInTable.toString(), textAlign:
TextAlign.center);
    }

    void _changeSelectedTable(int number) {
        setState(() {
            _selectedTable = number;
        });

        refreshFuture();
        updateBadge();

        _isDisabled = [false, false, false, true];
        if (_itemsInTable != 0) _isDisabled[3] = false;
        _tabController.index = 2;
    }

    // New function that updates the future and updates the n°
of items badge
    void _newUpdateFuture(int number) {

```

```

    refreshFuture();
    updateBadge();
}

// Updates money and change
void updateMoneyAndChange(List<num> money) {
    setState(() {
        _money = money[0];
        _change = money[1];
    });
}

void clearMoneyAndChange() {
    _clearMoneyChange = false;

    _isDisabled = [false, false, true, true];
    setState(() {
        _money = 0;
        _change = 0;
        _selectedTable = 0;
        if (_isDisabled[2] == true) _tabController.index = 1;
        if (_isDisabled[2] == false && _isDisabled[3] == true)
            _tabController.index = 2;
    });
    _resolveSelectedTable();
}

void cardPaymentRoutine() {
    _isDisabled = [false, false, true, true];
    ServerRequest.deleteTbill(_selectedTable.toString());
    ServerRequest.updateCashCount(_totalTableBill.toString(),
        _totalTableBill.toString(), null, 1, null, null, null,
        getDate());
    ServerRequest.updateTable(_selectedTable, 0.toString());

    setState(() {
        _selectedTable = 0;
        _itemsInTable = 0;
        if (_isDisabled[2] == true) _tabController.index = 1;
        if (_isDisabled[2] == false && _isDisabled[3] == true)
            _tabController.index = 2;
        listOfDBTables =
ServerRequest.fetchTables(http.Client());
    });
    _resolveSelectedTable();
}

void refreshFuture() {
    setState(() {
        futureTableBill =

```



```

ServerRequest.newfetchTableTbills(_selectedTable.toString());
    });
}

void updateBadge() {
    int count = 0;
    num total = 0;
    futureTableBill.then((value) {
        for (var item in value) {
            count = count + item.units;
            total = total + num.parse(item.total);
        }
        ServerRequest.updateTable(_selectedTable,
total.toString());

        setState(() {
            _itemsInTable = count;
            _totalTableBill = total;
            listOfDBTables =
ServerRequest.fetchTables(http.Client());
        });

        _isDisabled = [false, false, false, true];
        if (_itemsInTable != 0) _isDisabled[3] = false;
    });
}

// Goes to the sixth page and expects a return of money and
change
void askForMoney(BuildContext context) async {
    final moneyChange = await Navigator.of(context)
        .pushNamed('/sixth', arguments: <num>[_totalTableBill,
_selectedTable]);
    moneyChange as List<num>;
    print(moneyChange);
    if (moneyChange != [0, 0]) {
        setState(() {
            listOfDBTables =
ServerRequest.fetchTables(http.Client());
            _money = moneyChange[0];
            _change = moneyChange[1];
            _itemsInTable = 0;
        });
        _clearMoneyChange = true;
    }
}

// Goes to the edit_bill_product and edit the tbill entry
void editBillProduct(BuildContext context, List<dynamic>
tbillEntry) async {
    print(tbillEntry.toString());
}

```

```

    final tbillBody = await Navigator.of(context)
      .pushNamed('/fifth/edit_bill_product', arguments:
tbillEntry);
    if (tbillBody != null) tbillBody as bool;
    if (tbillBody == true) {
      refreshFuture();
      updateBadge();
    }
  }
}

// Retrieves the current date in format yyyy-mm-dd
String getDate() {
  DateTime now = DateTime.now();
  DateTime date = DateTime(now.year, now.month, now.day);
  final List<String> finalDate = date.toString().split(" ");
  return finalDate[0];
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    onGenerateRoute: RouteGenerator.generateRoute,
    home: DefaultTabController(
      length: 4,
      child: ScaffoldMessenger(
        key: scaffoldMessengerKey,
        child: Scaffold(
          drawer: Drawer(
            child: ListView(
              children: [
                ListTile(
                  title: Text(
                    'HManage',
                    style: TextStyle(
                      fontSize: 23,
                      color: Colors.blue,
                    ),
                  ),
                  textAlign: TextAlign.center,
                ),
                ListTile(
                  title: Text('This is the title'),
                  leading: Icon(Icons.add),
                  subtitle: Text('This is the subtitle'),
                  onTap: () => print("handsome"),
                  onLongPress: () => print("ugly"),
                ),
                ListTile(
                  title: Text('This is the title'),
                  leading: Icon(Icons.print),
                  subtitle: Text('This is the subtitle'),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  );
}

```

```

    ),
    ListTile(
      title: Text('This is the title'),
      leading: Icon(Icons.keyboard),
      subtitle: Text('This is the subtitle'),
    ),
    ListTile(
      title: Text('This is the title'),
      leading: Icon(Icons.account_balance),
      subtitle: Text('This is the subtitle'),
    ),
  ],
),
),
),
appBar: AppBar(
  bottom: TabBar(
    controller: _tabController,
    tabs: [
      Tab(icon: Icon(Icons.directions_car)),
      Tab(child: Text('Tables'), icon:
Icon(Icons.chair_alt)),
      Tab(
        child: Text('Products'),
        icon: Icon(Icons.grid_on_rounded)),
      Tab(
        child: Text('Bill'),
        icon: Row(
          mainAxisAlignment:
MainAxisAlignment.center,
          children: [
            Icon(Icons.format_list_bulleted_rounded),
            CircleAvatar(
              backgroundColor: Colors.white,
              radius: 10,
              child:
_resolveNumberOfItemsInTable(),
            ),
          ],
        ),
      ),
    ],
  ),
),
title: _resolveSelectedTable(),
),
body: TabBarView(
  physics: NeverScrollableScrollPhysics(),
  controller: _tabController,
  children: [
    SingleChildScrollView(
      child: Column(

```



```

        child: Text('TEST PLACEHOLDER')),
    ],
  ),
),
FutureBuilder<List<Tablee>>(
  future: listOfDBTables,
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return const Center(
        child: Text('An error has occurred!'),
      );
    } else if (snapshot.hasData) {
      return TableView(
        tables: snapshot.data!,
        changeTable: (int val) =>
_changeSelectedTable(val),
        //updateFuture: (int val) =>
_newUpdateFuture(val),
      );
    } else {
      return const Center(
        child: CircularProgressIndicator(),
      );
    }
  },
),
FutureBuilder<List<Product>>(
  future: listOfDBProducts,
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return const Center(
        child: Text('An error has occurred!'),
      );
    } else if (snapshot.hasData) {
      return ProductsView(
        tnumber: _selectedTable,
        products: snapshot.data!,
        updateFuture: (int val) =>
_newUpdateFuture(val),
      );
    } else {
      return const Center(
        child: CircularProgressIndicator(),
      );
    }
  },
),
FutureBuilder<List<Tbill>>(
  future: futureTableBill,
  builder: (context, snapshot) {
    if (snapshot.connectionState ==

```

```

ConnectionState.done) {
    if (snapshot.hasData) {
        if (snapshot.requireData.length != 0)
        {
            return Scaffold(
                body: Align(
                    alignment: Alignment.topCenter,
                    child: ListView(
                        children: [
                            Wrap(
                                alignment:
WrapAlignment.center,
                                spacing: 8.0,
                                runSpacing: 8.0,
                                children: [
                                    PopulateDataTable(
                                        tbills:
snapshot.data!,
                                        updateTbill:
(List<dynamic> val) =>
editBillProduct(context, val),
                                    ),
                                    Padding(
                                        padding:
EdgeInsets.all(28.0),
                                        child: Container(
                                            height: 250,
                                            width: 350,
                                            decoration:
BoxDecoration(
                                                color:
Colors.blue.shade50,
                                                borderRadius:
BorderRadius.circular(15),
                                            ),
                                        child: Column(
                                            mainAxisAlignment:
MainAxisAlignment.center,
                                            children: [
                                                Padding(
                                                    padding:
const
EdgeInsets.all(8.0),
                                                    child:
Container(
                                                        height: 50,
                                                        width: 350,
                                                        decoration:

```

```

BoxDecoration(
    color:
Colors.blue,
borderRadius:
BorderRadius.circular(
15),
),
child: Row(
mainAxisAlignment:
MainAxisAlignment
.spaceBetween,
children:
[
padding:
const EdgeInsets
.all(8.0),
child:
Text(
'Total',
style: TextStyle(
fontSize: 26,
color: Colors.white,
fontWeight:
FontWeight.bold,
),
),
padding:
const EdgeInsets
.all(8.0),
child:

```



```

    ],
  ),
),
),
),
padding:
  const
EdgeInsets.all(8.0),
child:
Container(
  height: 50,
  width: 350,
  decoration:
    BoxDecoration(
      color:
Colors.blue,
borderRadius:
BorderRadius.circular(
15),
),
  child: Row(
mainAxisAlignment:
MainAxisAlignment
.spaceBetween,
children:
[
padding:
  Padding(
const EdgeInsets
.all(8.0),
child:
Text(
  'Change',
style: TextStyle(
fontSize: 26,
color: Colors.white,

```



```

    );
  }
}

class PopulateDataTable extends StatelessWidget {
  final List<Tbill> tbills;
  final List<DataRow> itemRow = [];
  final Function(List<dynamic>) updateTbill;

  PopulateDataTable({
    Key? key,
    required this.tbills,
    required this.updateTbill,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    for (var item in tbills) {
      itemRow.add(
        DataRow(
          cells: [
            DataCell(Text(item.units.toString())),
            DataCell(Text(item.item)),
            DataCell(Text(item.iprice)),
            DataCell(Text(item.total)),
            DataCell(Icon(Icons.edit), onTap: () {
              updateTbill([
                item.id,
                item.units,
                item.item,
                item.iprice,
                item.total,
              ]);
            })),
          ],
        ),
      );
    }
    return Padding(
      padding: EdgeInsets.all(15.0),
      child: SingleChildScrollView(
        scrollDirection: Axis.horizontal,
        child: DataTable(
          columns: [
            const DataColumn(
              label: Text(
                'Units',
                style: TextStyle(
                  fontWeight: FontWeight.w500,
                  fontSize: 16,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}

```

```

    ),
  ),
  const DataColumn(
    label: Text(
      'Product',
      style: TextStyle(
        fontWeight: FontWeight.w500,
        fontSize: 16,
      ),
    ),
  ),
  const DataColumn(
    label: Text(
      'Price',
      style: TextStyle(
        fontWeight: FontWeight.w500,
        fontSize: 16,
      ),
    ),
  ),
  const DataColumn(
    label: Text(
      'Total',
      style: TextStyle(
        fontWeight: FontWeight.w500,
        fontSize: 16,
      ),
    ),
  ),
  const DataColumn(
    label: Text(
      '',
      style: TextStyle(
        fontWeight: FontWeight.w500,
        fontSize: 1,
      ),
    ),
  ),
],
rows: itemRow,
),
),
);
}
}

```

```

class ProductsView extends StatelessWidget {
  final List<Product> products;
  final int tnumber;
  final Function(int) updateFuture;

```

```

ProductsView({
  Key? key,
  required this.tnumber,
  required this.updateFuture,
  required this.products,
  // required this.addItem,
}) : super(key: key);

@override
Widget build(BuildContext context) {
  return GridView.builder(
    scrollDirection: Axis.vertical,
    gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(
      maxCrossAxisExtent: 100,
      crossAxisSpacing: 5,
      mainAxisSpacing: 5,
    ),
    padding: const EdgeInsets.all(10),
    itemCount: products.length,
    itemBuilder: (context, index) {
      return ElevatedButton(
        onPressed: () {
          ServerRequest.createTbill(
            tnumber,
            products[index].name,
            1,
            products[index].price,
            products[index].price,
          );
          updateFuture(tnumber);
        },
        child: Text(products[index].name));
    },
  );
}

class TableView extends StatelessWidget {
  final List<Tablee> tables;
  final Function(int) changeTable;
  TableView({
    Key? key,
    required this.tables,
    required this.changeTable,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return GridView.builder(
      scrollDirection: Axis.vertical,
      gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(

```

```

        maxCrossAxisExtent: 100,
        crossAxisSpacing: 5,
        mainAxisSpacing: 5,
      ),
      padding: const EdgeInsets.all(10),
      itemCount: tables.length,
      itemBuilder: (context, index) {
        return ElevatedButton(
          onPressed: () {
            changeTable(tables[index].number);
          },
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(tables[index].number.toString()),
              Text(tables[index].total),
            ],
          ),
        );
      },
    );
  }
}

```

9.2.11 sixth_page.dart

```

import 'package:flutter/material.dart';
import 'package:h_manage/server_request.dart';

//TODO: Create bills and coins icons

class SixthPage extends StatefulWidget {
  final num total;
  final num tnumber;
  SixthPage({
    Key? key,
    required this.total,
    required this.tnumber
  }) : super(key: key);

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

class _SixthPageState extends State<SixthPage> {
  String getDate() {
    DateTime now = DateTime.now();
    DateTime date = DateTime(now.year, now.month, now.day);
    final List<String> finalDate = date.toString().split(" ");
    return finalDate[0];
  }
}

```



```

@override SixthPage get widget => super.widget;
num _counter = 0;
List<int> listInt = [200,100,50,20,10,5];
List<double> listDouble = [2,1,0.5,0.2,0.1,0.05,0.02,0.01];

List<num> moneyChange = [0,0];

void _addMoney(num number) {
  setState(() {
    _counter = _counter + number;
  });
}

void _clearMoney() {
  setState(() {
    _counter = 0;
  });
}

void _clearTbills(){
  ServerRequest.updateCashCount(
    widget.total.toString(),
    null,
    widget.total.toString(),
    1,
    null,
    null,
    null,
    getDate()
  );
  ServerRequest.deleteTbill(widget.tnumber.toString());
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Payment table: ' +
widget.tnumber.toString()),
    ),
    body: Center(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(
                'Total:',
              ),
              Text(

```

```

        widget.total.toStringAsFixed(2),
        style:
Theme.of(context).textTheme.headline4,
    ),
    Text(
        'Cash:',
    ),
    Text(
        _counter.toStringAsFixed(2),
        style:
Theme.of(context).textTheme.headline4,
    ),
    ...tableView2(listInt),
    Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
            ...tableView2(listDouble),
        ],
    ),
    Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
            ElevatedButton(onPressed: () =>
_clearMoney(), child: Text('Clear')),
            ElevatedButton(
                child: Text('OK'),
                onPressed: () {
                    moneyChange[0] = _counter;
                    moneyChange[1] = _counter -
widget.total;

                    if (moneyChange[1] < 0) {
                        _clearMoney();
                        showDialog<String>(
                            context: context,
                            builder: (BuildContext context) =>
                                // TODO: Handle iOS Dialog
                                AlertDialog(
                                    title: const Text('Not
enough money'),
                                    content: const Text(
                                        'Please introduce
quantity again'),
                                    actions: <Widget>[
                                        TextButton(
                                            onPressed: () =>
Navigator.pop(
                                context, 'Cancel'),
                                            child: const
Text('Cancel'),
                                        ),
                                        TextButton(

```

```

                                onPressed: () =>
Navigator.pop(context, 'OK'),
                                child: const Text('OK'),
                                ),
                                ],
                                ),
                                );
                                } else {
                                ServerRequest.updateTable(
int.parse(widget.number.toString()), 0.toString()
                                );
                                _clearTbills();
                                Navigator.pop(context, moneyChange);
                                }
                                }
                                ),
                                ],
                                ),
                                ],
                                ),
                                ],
                                ),
                                ),
                                ),
                                // floatingActionButton: FloatingActionButton(
                                //   heroTag: "sum",
                                //   onPressed: _incrementCounter,
                                //   tooltip: 'Increment',
                                //   child: Icon(Icons.add),
                                // ),
                                );
                                }

List<Widget> tableView2 (List<num> listOfInts){
  List<Widget> buttons = [];
  for(var individual in listOfInts) {
    // buttons.add(Text(individual.number.toString()));
    buttons.add(
      ElevatedButton(
        child: Text(individual.toString() + '€'),
        onPressed: () => _addMoney(individual),
      )
    );
  }
  return buttons;
}

class ButtonWidget extends StatelessWidget {

```

```

    final num foreignCount;
    final Function(int) onCountChange;
    ButtonWidget({required this.foreignCount, required
this.onCountChange });

    @override
    Widget build(BuildContext context) {
        return ElevatedButton(
            onPressed: () => onCountChange(13),
            child: Text('13')
        );
    }
}

```

9.2.12 edit_bill_product.dart

```

import 'package:flutter/material.dart';
import 'package:h_manage/server_request.dart';

import 'package:h_manage/pages/hero_dialog_route.dart';
import 'package:h_manage/pages/edit_price.dart';

class EditBillProduct extends StatefulWidget {
    final int id;
    final int units;
    final String product;
    final String price;
    final String total;
    EditBillProduct({
        Key? key,
        required this.id,
        required this.units,
        required this.product,
        required this.price,
        required this.total,
    }) : super(key: key);

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

class _EditBillProductState extends State<EditBillProduct> {
    int count = 0;
    num total = 0;
    num itemPrice = 0;
    bool justOnce = true;
    bool productError = false;

    TextEditingController _controllerProduct =
    TextEditingController();
}

```

```

String getDate() {
    DateTime now = DateTime.now();
    DateTime date = DateTime(now.year, now.month, now.day);
    final List<String> finalDate = date.toString().split(" ");
    return finalDate[0];
}

@override
EditBillProduct get widget => super.widget;

@override
void initState() {
    super.initState();
    count = widget.units;
    total = num.parse(widget.total);
    itemPrice = num.parse(widget.price);

    _controllerProduct.text = widget.product;
    _controllerProduct.selection = TextSelection(
        baseOffset: 0, extentOffset:
        _controllerProduct.text.length);
}

void decreaseUnits() {
    if (count > 1) {
        setState(() {
            count--;
        });
        updateTotal();
    }
}

void increaseUnits() {
    setState(() {
        count++;
    });
    updateTotal();
}

void updateTotal() {
    setState(() {
        total = count * itemPrice;
    });
}

// Goes to the ..itemPrice and edit the product price
void editPrice(BuildContext context, num price) async {
    final returnedPrice = await Navigator.of(context)
        .pushNamed('/fifth/edit_bill_product/price',
arguments: price);
    returnedPrice as num;
}

```

```

void editPrice2(BuildContext context, num price) async {
  final returnedPrice = await
Navigator.of(context).push(HeroDialogRoute(builder: (context)
{
  return AddPricePopupCard(price: price);
})));
if (returnedPrice != null) returnedPrice as num;
print(returnedPrice);
if (returnedPrice is num) {
  setState(() {
    itemPrice = returnedPrice;
  });
  updateTotal();
}
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Edit'),
    ),
    body: Container(
      alignment: Alignment.center,
      padding: const EdgeInsets.all(8.0),
      margin: EdgeInsets.symmetric(
        horizontal: MediaQuery.of(context).size.width / 20,
        vertical: MediaQuery.of(context).size.width / 20,
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        mainAxisAlignment: MainAxisAlignment.max,
        children: <Widget>[
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text('Units:', style: TextStyle(fontSize:
18)),
              TextButton(
                onPressed: () => decreaseUnits(),
                child: Icon(Icons.keyboard_arrow_down)
              ),
              Text(count.toString()),
              TextButton(
                onPressed: () => increaseUnits(),
                child: Icon(Icons.keyboard_arrow_up))
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,

```

```

        children: [
          Text('Product: ', style: TextStyle(fontSize:
18)),
          Container(
            width: 155,
            child: TextField(
              textAlign: TextAlign.center,
              textCapitalization:
TextCapitalization.sentences,
              controller: _controllerProduct,
              decoration: InputDecoration(
                hintText: 'Product name',
                errorText: productError ? "Value Can't
Be Empty" : null,
              ),
              onTap: () {
                if (justOnce == true) {
                  justOnce = false;
                  _controllerProduct.selection =
TextSelection(
                    baseOffset: 0,
                    extentOffset:
_controllerProduct.text.length);
                }
              },
            ),
          ],
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Price: ', style: TextStyle(fontSize:
18)),
            TextButton(
              onPressed: () {
                editPrice2(context, itemPrice);
              },
              child: Text(itemPrice.toStringAsFixed(2) +
' €')),
            Icon(Icons.edit, size: 16),
          ],
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Total: ', style: TextStyle(fontSize:
18)),
            Text(total.toStringAsFixed(2) + ' €'),
          ],
        ),
        ElevatedButton(

```

```

        onPressed: () {
          setState(() {
            _controllerProduct.text.isEmpty
              ? productError = true
              : productError = false;
          });
          if (!productError) {
            ServerRequest.updateTbill(
              widget.id,
              null,
              _controllerProduct.text,
              count,
              itemPrice.toString(),
              total.toString());
            ScaffoldMessenger.of(context).showSnackBar(
              const SnackBar(content: Text('Processing
Data'))),
            );
            Navigator.pop(context, true);
          }

        },
        child: const Text('Edit'),
      ),
      // ElevatedButton(
      //   onPressed: () {
      //     print(widget.id);
      //     print(widget.units);
      //     print(widget.product);
      //     print(widget.price);
      //     print(widget.total);
      //   },
      //   child: const Text('show'),
      // ),
    ],
  ),
),
);
}
}

```

9.2.13 edit_price.dart

```

import 'package:flutter/material.dart';

class AddPricePopupCard extends StatefulWidget {
  final num price;

  AddPricePopupCard({Key? key, required this.price}) :
super(key: key);

  @override

```



```

    _AddPricePopupCardState createState() =>
    _AddPricePopupCardState();
}

class _AddPricePopupCardState extends State<AddPricePopupCard>
{
    @override AddPricePopupCard get widget => super.widget;

    TextEditingController _controllerPrice =
    TextEditingController();

    @override
    Widget build(BuildContext context) {
        return Center(
            child: Padding(
                padding: const EdgeInsets.all(32.0),
                child: Material(
                    color: Colors.blue.shade50,
                    elevation: 2,
                    shape:
                    RoundedRectangleBorder(borderRadius:
                    BorderRadius.circular(32)),
                    child: SingleChildScrollView(
                        child: Padding(
                            padding: const EdgeInsets.all(16.0),
                            child: Column(
                                mainAxisAlignment: MainAxisAlignment.min,
                                children: [
                                    const Text('Enter new item price.'
                                    ),
                                    const Divider(
                                        color: Colors.white,
                                        thickness: 0.2,
                                    ),
                                    TextField(
                                        keyboardType: TextInputType.number,
                                        controller: _controllerPrice,
                                        textAlign: TextAlign.center,
                                        decoration: InputDecoration(
                                            hintText: '0.00',
                                            border: InputBorder.none,
                                        ),
                                        cursorColor: Colors.white,
                                        maxLines: 6,
                                    ),
                                    const Divider(
                                        color: Colors.white,
                                        thickness: 0.2,
                                    ),
                                    TextButton(
                                        onPressed: () {

```

```

        if (_controllerPrice.text.isEmpty) {
          Navigator.pop(
            context, widget.price);
        } else {
          Navigator.pop(
            context,
num.parse(_controllerPrice.text));
        }
      },
      child: const Text('Edit'),
    ),
  ],
),
),
),
),
),
),
);
}
}

```

9.2.14 hero_dialog_route.dart [14]

```

import 'package:flutter/material.dart';

class AddPricePopupCard extends StatefulWidget {
  final num price;

  AddPricePopupCard({Key? key, required this.price}) :
super(key: key);

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

class _AddPricePopupCardState extends State<AddPricePopupCard>
{
  @override AddPricePopupCard get widget => super.widget;

  TextEditingController _controllerPrice =
  TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Padding(
        padding: const EdgeInsets.all(32.0),
        child: Material(
          color: Colors.blue.shade50,
          elevation: 2,
          shape:

```

```

        RoundedRectangleBorder(borderRadius:
BorderRadius.circular(32)),
        child: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.min,
              children: [
                const Text('Enter new item price.'
                ),
                const Divider(
                  color: Colors.white,
                  thickness: 0.2,
                ),
                TextField(
                  keyboardType: TextInputType.number,
                  controller: _controllerPrice,
                  textAlign: TextAlign.center,
                  decoration: InputDecoration(
                    hintText: '0.00',
                    border: InputBorder.none,
                  ),
                  cursorColor: Colors.white,
                  maxLines: 6,
                ),
                const Divider(
                  color: Colors.white,
                  thickness: 0.2,
                ),
                TextButton(
                  onPressed: () {
                    if (_controllerPrice.text.isEmpty) {
                      Navigator.pop(
                        context, widget.price);
                    } else {
                      Navigator.pop(
                        context,
num.parse(_controllerPrice.text));
                    }
                  },
                  child: const Text('Edit'),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  );
}
}

```

