TEC Bank: Documentación Técnica

Bases de Datos

Instituto Tecnológico de Costa Rica
Tarea corta #1

Carlos Adrián Araya Ramírez	**2018319701**
José Alejandro Chavarría Madriz	**2019067306**
Sebastián Mora Godínez	**2019227554**
Michael Shakime Richards Sparks	**2018170667**

Prof . Marco Rivera Meneses

Índice

Diagramas

Diagrama de Arquitectura

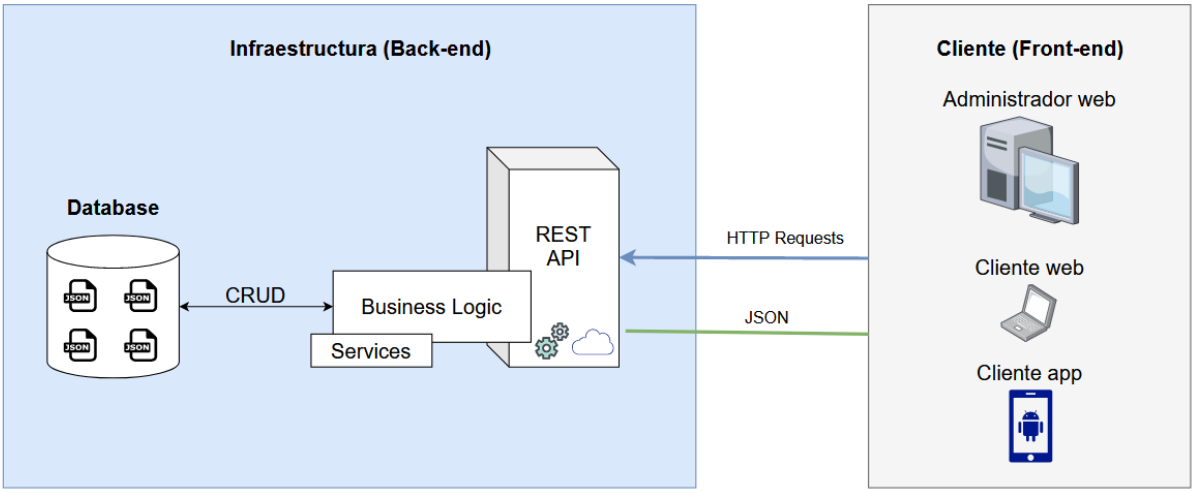


Diagrama de Clases

Descargar de https://github.com/sebas-mora28/TecBank/wiki

Diagrama Entidad-relación

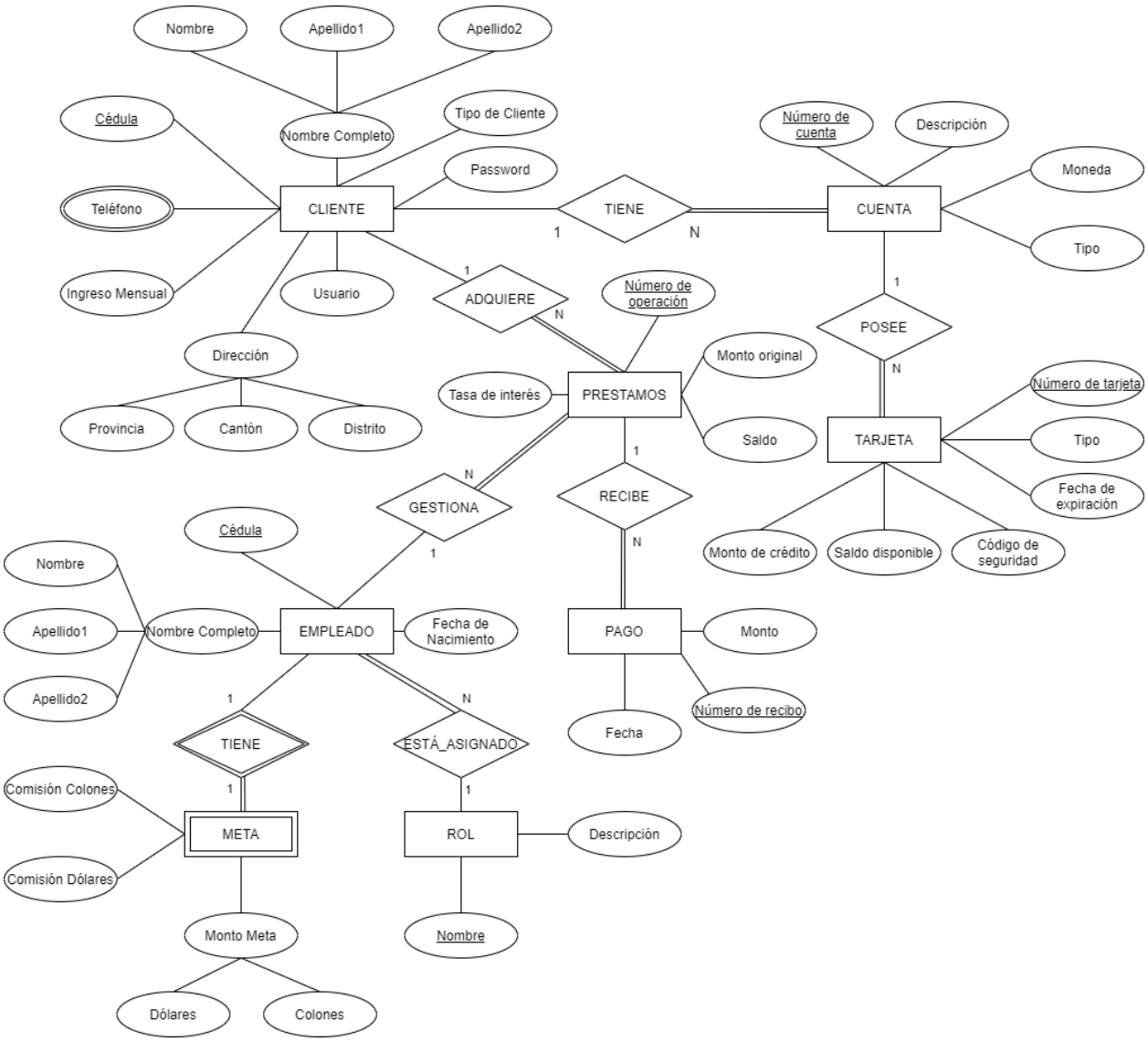
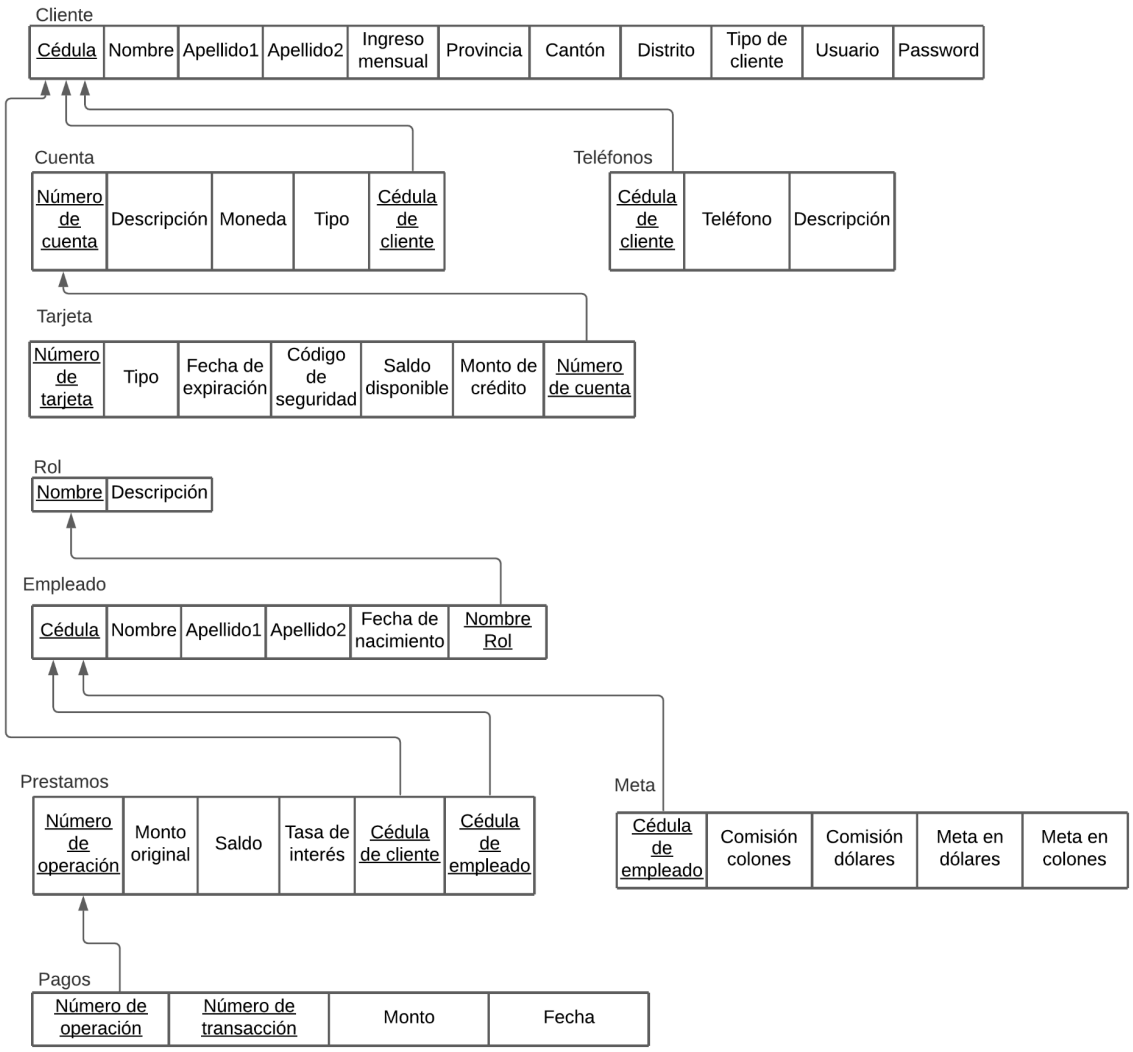


Diagrama Relacional



Métodos Implementados

REST API

Para la implementación de la API se utilizó documentación proporcionada por Microsoft [1] para crear una web API en C# con ASP.NET en la cual se crean tres módulos principales:

- **Models:** en este módulo se encuentran los modelos de las entidades que utiliza la API, es decir aquí se encuentran las clases que modelan las entidades con sus atributos, un ejemplo de un modelo para esta implementación es el de Rol, el cual posee dos atributos, Nombre y Descripción ambos strings, estas clases constan únicamente de atributos y metodos get y set para cada uno de ellos.
- **Services:** este módulo se encarga de manejar todos los servicios de cada entidad, es decir en este módulo se encuentran todos los métodos relacionados al CRUD de cada entidad para luego ser utilizados por lo controladores, algunos de los métodos que se encuentran en las clases de este módulo son el Add, Delete, Update.
- **Controllers:** en este módulo se encuentran los controladores de la API, estos son los responsables de manejar las requests que hace el admin y los clientes haciendo uso de los servicios de cada modelo de entidad, es aquí donde se atrapan y manejan las solicitudes REST como los GET, POST, PUT y las demás

solicitudes necesarias para la implementación, para este caso solo fue necesaria la implementación de una solicitud GET y un POST.

Un método muy importante en la implementación de la API es el método controlador de JSON "`JSONManager`", encargado de la serialización de los datos en memoria para ser guardados y cargados. Para la implementación de este método se utilizó la librería Newtonsoft [2], la cual ya posee cada uno de los métodos necesarios para serializar y deserializar los datos, el método implementado por el equipo utiliza esta librería y una la ruta de los archivos JSON para cargar y guardar los datos de cada entidad.

Aplicación web

La aplicación web fue desarrollada en Angular, por lo que fue necesario implementar una serie de componentes y servicios cada uno con sus respectivas funcionalidades.

Cada componente posee un método que le permite conocer en qué url se encuentra y mostrar así cierto comportamiento requerido.

Add-item posee un método onSubmit() que ejecuta cuando el botón de enviar a base de datos es oprimido envía la información que el usuario haya introducido, con el formato correcto dependiendo del url donde se encuentre.

Header posee la función is_client() que indica si el usuario está o no en alguno de los urls asociados a un cliente. Además modifica los parámetros para manejar el color de los botones en el header.

Cada componente item tiene la funcion onEdit() que lo comunica directamente con el servicio de UI.

Item holder tiene varios métodos importantes ya que es la piedra angular de la implementación de los distintos servicios de gestión de la aplicación. Algunos de ellos son: toogleAddItem() y add_item() las que se comunican con los servicios de UI y API respectivamente al querer agregar un item. editItemClicked() utiliza toogleAddItem() y prepara la interfaz de edición con el servicio de UI. editItem() envía al API la petición de put para un ítem. La función es llamada con un diferente atributo dependiendo el url del usuario y la llave primaria del objeto. DeleteItem() Funcion que envía al API la petición de deletreo para un ítem. La función es llamada con un diferente atributo dependiendo el url del usuario y la llave primaria del objeto.

El servicio de API tiene funciones GET, POST, PUT, DELETE generales que evalúan la URL donde el usuario se encuentra y dependiendo de esto ejecuta una función auxiliar de su tipo, para realizar la acción con el tipo de objeto que sea necesario, así el API service abstrae la necesidad del componente de saber en qué URL se encuentra y tan solo debe ejecutar una única llamada al api.

El UI service se encarga de controlar aspectos de interfaz cuando 3 o más componentes deben interactuar entre sí. Este servicio almacena variables de estado que permiten a los demás componentes conocer el estado de otros, modificarlo o ser agnósticos a su propio estado y tan solo realizar llamadas al servicio.

Aplicación móvil

La aplicación móvil fue desarrollada en el entorno de desarrollo Android, donde fue necesario implementar una serie de fragmentos que representan las vistas de cada una de las partes de la aplicación.

Cada uno de los fragmentos tiene un método llamado onCreateview() donde se detallan todas las instrucciones que se deben ejecutar cuando una de las vistas es creada. Además, existe otro método llamado onViewCreated() donde se detallan cada una de las funcionalidades que tendrá la vista una vez que fue creada, dentro de este método, se realizan las consultas al API, se asignan las funcionalidades a los botones, etc.

Además se implementó el método getClients() que se encarga de realizar la consulta al API para obtener todos los clientes disponibles para visualizarlas en las vistas correspondientes.

Estructuras desarrolladas

Para el almacenamiento y organización de los datos se utilizaron archivos JSON los cuales tienen la forma de estructuras desarrolladas para cada tipo de entidad del mini mundo las cuales tienen la siguiente forma:

- **Cliente:**

  Nombre_Completo: String
  Cedula: String
  Dirección: String
  Telefonos: Lista String
  Ingreso_Mensual: Double
  Tipo_de_cliente: String
  Prestamos: Lista Prestamo

- **Rol:**

  Nombre: String
  Descripcion: String

- **Cuenta:**

  Numero_Cuenta: String
  Descripcion: String
  Moneda: String
  Tipo: String
  Cedula_Propietario: String
  Saldo: double
  Movimientos: Lista Movimiento

- **Tarjeta:**

  Numero_de_tarjeta: String

  Fecha_de_experacion: String

  Codigo_de_Seguridad: String

  Tipo: String

  Saldo_o_Credito: double

  Numero_Cuenta: String

- **Pago:**

  Monto: long

  Fecha: String

- **Movimiento:**

  Fecha: String

  Monto: double

  Tipo: String

- **Prestamo:**

  Numero_de_prestamo: int

  Tasa_de_interes: float

  Monto_original: long

  Saldo: int

  Pagos: Lista Pago

- **Transferencia:**

  Receptor: String

  Monto: double

Algoritmos Desarrollados

Algoritmo de login

Del lado del cliente web para verificar el logueo de un cliente primero es llamado un GET al API con todos los clientes registrados. Cuando el usuario da click al ingresar el comparado el valor de su nombre de usuario y contraseña contra el de cada uno de los clientes adquiridos, si se logra tener una congruencia entre nombre y contraseña se le permite al cliente ingresar a las demás vistas de la aplicación.

Por otra parte si no es encontrado un match pero el nombre de usuario es admin y la contraseña, se le dejará ingresar a las vistas de administrador.

Algoritmo de creación de tablas de ítems

Para llenar el componente item-holder con cada uno de los ítems para cierta categoría (cuentas,tarjetas,etc) se hace una llamada de GET según el URL. La respuesta después es introducida en un ciclo que genera un componente ítem dentro de item holder con los datos de cada ítem.

El componente ítem posee una variable que identifica a cada posible item que puede contener, sin embargo sólo mostrará aquellas que estén condicionadas para el URL donde se encuentre el usuario.

Problemas conocidos

Todo lo requerido en la especificación es solucionado correctamente.

Problemas encontrados

Post utilizando la biblioteca Volley

Se encontró un error al intentar realizar un POST desde la aplicación de android mediante la biblioteca de Volley. Al momento de realizar la consulta, se devolvía un error.

El problema era que en la forma en que se estaba enviado la información era incorrecta, pues se estaba enviando por los parámetros y no por el cuerpo. Además, se debería especificar a través de los headers que el formato era JSON.

Desplazamiento de vistas en aplicación móvil

Al momento de querer navegar entre las diferentes vistas de la aplicación no se desplazaba correctamente. La causa del problema era la forma en que se estaba implementando esta acción, pues no era la correcta.

Se descubrió que android studio ofrece otra forma para navegar entre las diferentes vistas y se procedió a cambiar la implementación.

Uso de archivos JSON como base de datos

En el backend se necesitaba una forma de almacenar los datos sin utilizar base de datos, por lo tanto se tomó la decisión de guardar los datos en archivos JSON, sin embargo, surgió el problema de que la mayoría de documentaciones de

implementación de un API en C# con .NET contemplaban únicamente el caso de utilizar una base de datos como SQL y NOSQL.

En un principio se estaba siguiendo una guía de implementación proporcionada por Microsoft [1], en la cual se utiliza una lista estática en memoria para guardar los datos de la API, sin embargo, esto implicaba que cada vez que el API se reiniciaba los datos se perdían. Para solucionar esto se mantuvo la idea de utilizar una lista para almacenar los datos pero además se implementó una clase estática llamada "JSONManager" la cual posee los métodos responsables de cargar y guardar los datos de la API en archivos JSON esto fue posible con la librería Newtonsoft [2] , de esta manera cada vez que se corre la API se llaman los métodos correspondientes para recuperar los datos almacenados en los archivos JSON.

Problemas de CORS al conectar Angular con otro servidor

Para el desarrollo más ágil del proyecto se decidió subir el API a un servidor web gratuito de Azure. Sin embargo, al intentar hacer requests HTTP el servidor se negaba a responder por CORS.

Para solucionar este problema fue necesario implementar un proxy del lado de Angular que permitiera realizar estos requests a server. De igual forma el mismo proxy fue necesario al intentar utilizar el API en otro servidor localhost.

Angular con Bold reports

Para generar los reportes en angular se intentó utilizar una prueba de 15 días del software de pago Bold reports, y aunque se siguió la documentación proveída por los desarrolladores no se logró hacer funcionar por razones aún desconocidas.

Finalmente, fue desechado y se decidió utilizar la versión gratuita de reporting services de stimulsoft. Para ambos también fue necesario hacer un proyecto en Visual Studio pues solo allí se podían conseguir los NuGet packages requeridos.

Plan de actividades

Área	Actividad	Duración Estimada [h]	Responsable	Fecha de entrega
Web App	Adaptación de Visual Studio para Angular	6	Jose Alejandro	19/08/21
	Implementación de Login	12	Jose Alejandro	24/08/21
	Implementación de CRUD de roles	24	Jose Alejandro	28/08/21
	Implementación de CRUD de clientes	12	Jose Alejandro	28/08/21
	Implementación de visualización de cuentas	6	Jose Alejandro	31/08/21
	Implementación de visualización de tarjetas	6	Jose Alejandro	31/08/21
	Implementación de conexión con el API desarrollado	12	Jose Alejandro	1/9/21
	Implementacion de páginas mock para la vista cliente	6	Jose Alejandro	1/9/21
	Implementación de un sistema de reporte (cliente)	6	Jose Alejandro	5/9/21
	Implementación de un sistema de reporte (servidor)	6	Jose Alejandro	5/9/21
	Implementación de estética de la aplicación con CSS	6	Jose Alejandro	5/9/21
API	Investigación, instalación paquetes y creación del proyecto en C# con .NET	12	Shakime Richards	21/08/21
	Implementación de los modelos de las entidades	6	Adrian Araya	24/08/21
	Implementación de la base de datos utilizando JSON	12	Adrian Araya	28/08/21
	Implementación de los services (CRUD) para cada modelo de entidad	6	Adrian Araya	28/08/21

	Implementación de los controladores para las solicitudes REST (GET, POST)	12	Shakime Richards	03/09/21
	Implementación de las restricciones de cada protocolo de request	4	Shakime Richards	03/09/21
Movil App	Investigación del entorno de desarrollo android	18	Sebastián Mora	18/08/21
	Creación de la vista para login	24	Sebastián Mora	24/08/21
	Creación de la vista del menú principal	24	Sebastián Mora	24/08/21
	Creación de la vista para las cuentas	12	Sebastián Mora	26/08/21
	Creación de la vista del menú de cuentas	12	Sebastián Mora	26/08/21
	Creación de la vista para las transferencias	24	Sebastián Mora	26/08/21
	Implementación de la conexión de la aplicación con el API	48	Sebastián Mora	07/09/21

Minutas

Minutas de Reunión				
#	**Asunto a tratar**	**Participantes**	**Actividades y acuerdos**	**Fecha**
1	Planeación y división del proyecto	Todos	● Sebas se encargará de la app móvil. ● José se encargará de la app web. ● Shakime y Adrián se encargarán de la API. ● Se acordó finalizar el esqueleto de todas las secciones del proyecto y revisarlos en una posterior reunión.	18/8/2021
2	Gestión de URLs	Todos	● Se expusieron los avances de cada sección entre los miembros del grupo. ● Se acordaron las URLs que se iban a utilizar para el muestreo de la información. ● Modificación al diagrama conceptual. ● Se plantearon las dudas sobre el funcionamiento de las tarjetas de crédito y los pagos de préstamos.	26/8/2021
3	Control y funcionalidad	Todos	● Se mostró la etapa casi final de cada sección a los demás integrantes. ● Se plantearon algunos cambios estéticos en la interfaz del web app.	31/8/2021
4	Testeo	Todos	● Pruebas de la funcionalidad de todas las partes juntas.	5/9/2021

Bitácora

Fecha	Estudiante(s)	Actividad
18/8/2021	Todos	Planeamiento y distribución de tareas.
19/8/2021		Se instalaron todos los recursos a utilizar en la tarea.
	José Chavarría	Creación e implementación del proyecto y esqueleto de la aplicación web.
21/8/2021	Sebastián Mora	Creación e implementación del proyecto y esqueleto de la aplicación móvil.
23/8/2021	Shakime Richards	Creación del proyecto y template para realizar CRUD en la API.
24/8/2021	José Chavarría	Implementación de la vista para el login.
	Sebastián Mora	Creación de la vista para login y menú principal
25/8/2021	José Chavarría	Creación de todas las vistas restantes (menú principal y cuentas, transferencias).
	Adrián Araya	Implementación de la estructura inicial (entidades - comunicación) en la API.
26/8/2021	Adrián Araya Shakime Richards	Corrección de errores en el gitignore para universalidad a la hora de ejecutar la API.
	Adrián Araya	Establecimiento de la comunicación Json para el almacenamiento de datos.
	Sebastián Mora	Implementación de la vista para cuentas, tarjetas y menú de cuentas.
	Adrián Araya	Actualización de la vista básica del cliente.
29/8/2021	Sebastián Mora	Actualización de las vistas para cuentas y menú de cuentas.
		Implementación de las vistas para transferencias.
	Adrián Araya	Establecimiento de la comunicación Json para el almacenamiento de datos.
	Shakime Richards	Implementación de depósitos, retiros, transferencias y seguimiento de los movimientos.

30/8/2021	Shakime Richards	Implementación del login en la API.
31/8/2021	Todos	Reunión de control.
	Sebastián Mora	Cambios menores
1/9/2021	Shakime Richards	Corrección en la estructura de la entidad pagos. (ID de entidad fuerte)
	José Chavarría	Funcionalidad CRUD del cliente completa con conexión al API (azure).
2/9/2021	Sebastián Mora	Finalizada la aplicación móvil.
4/9/2021	José Chavarría	Estética de la página utilizando CSS.
5/9/2021	Sebastián Mora	Cambios en la interfaz de la aplicación móvil.
	José Chavarría	Funcionalidad del reporte implementada.

Conclusiones

- Se utiliza el entorno de desarrollo de Android para desarrollar una aplicación móvil.
- Se utiliza la biblioteca Volley para realizar las consultas al API desde la aplicación móvil.
- Es posible desarrollar aplicaciones Web bastante robustas con Angular ya que posee una gran cantidad de funcionalidades.
- Angular tiene políticas de CORS que no permiten la interacción entre servidores si no son contempladas.
- Muchos sistemas de reporte necesitan de NuGet packages disponibles en las bibliotecas de .NET de Visual Studio.
- Para agilizar el desarrollo sin necesidad de instalar ambientes u otras aplicaciones es posible trabajar con Azure y otras plataformas que permiten subir sitios web gratuitos
- Los archivos JSON, C# y TypeScript son case sensitive, por lo que pueden causar muchos problemas de integración si no hubo planificación previa.

Recomendaciones

- Se recomienda utilizar el entorno de desarrollo Android para desarrollar aplicaciones móviles por las diferentes facilidades y funcionalidades que ofrece.
- Se recomienda utilizar la biblioteca Volley para realizar consultas a un API por su facilidad e integración con el entorno de desarrollo de Android.
- Antes de comenzar un proyecto con Angular es importante hacer una investigación inicial puesto que es una plataforma con un grado de complejidad importante, producto de sus muchas funcionalidades.
- Utilizar un proxy del lado de Angular para comunicaciones inter servidores o bien desactivar CORS del lado del servidor API.
- Se va a desarrollar un API en .NET, utilizar Visual Studio para tener acceso a mayor funcionalidad como sistemas de reporte.
- Utilizar sitios web gratuitos para hospedar APIs se puede permitir un desarrollo en paralelo de una aplicación web que lo requiera, sin la necesidad de instalar los ambientes necesarios para correr el API.
- Es muy importante definir convenciones para el nombramiento de variables en las estructuras que se van a definir en los archivos JSON, C# y TypeScript pues son case sensitive.

Bibliografía

[1] "Create a web API with ASP.NET Core - Learn", *Docs.microsoft.com*, 2021. [Online]. Available: https://docs.microsoft.com/en-us/learn/modules/build-web-api-aspnet-core/?WT.mc_id=dotnet-35129-website. [Accessed: 04- Sep- 2021].

[2] "Introduction", *Newtonsoft.com*, 2021. [Online]. Available: https://www.newtonsoft.com/json/help/html/Introduction.htm. [Accessed: 04- Sep- 2021].