Instituto Tecnológico de Costa Rica
Área Académica de Administración de Tecnologías de Información
Curso: Bases de Datos Avanzados
Profesor: María José Artavia Jiménez
Grupo: 02

TECPlane

Estudiantes:
José Pablo Esquivel Morales
Randy Alejandro Martínez Sandí
Luis José Martínez Ramírez
Gustavo Alonso Fallas Carrera

Cartago, - de octubre

Tabla de Contenidos

Resumen **2**

Completitud del Proyecto **2**

Arquitectura General **3**

Base de Datos **4**

Herramientas y Equipo 4

Documentos y Colecciones 4

Replicación y Distribución 10

Cliente **14**

Herramientas y Equipo 14

Funcionamiento 14

Reportes 17

Administradores 17

Empleados 20

Pasajeros 23

Aplicación **26**

Herramientas y Equipo 26

Funcionamiento 26

Conclusiones **27**

Bibliografía **29**

Resumen

El presente documento es una ficha técnica del proyecto programado llamado TECPlane, el cual consiste en desarrollar un sistema de bases distribuidas (replicación únicamente) utilizando un motor de base de dato no relacional (NoSQL). El proyecto es un sistema de manejo de aerolíneas, aeropuertos y aeropuertos. Asimismo, permite la creación de cuentas para que pasajeros puedan registrarse, crear cuentas, ver vuelos y comprar tiquetes, así como realizar operaciones básicas a la hora de viajar. El objetivo es aprender y aplicar el uso de un motor no relacional para lograr un sistema balanceado y siempre disponible.

Completitud del Proyecto

El proyecto TECPlane se completó en su totalidad (100%) según lo acordado y entendido en las especificaciones del proyecto.

1. Arquitectura General

El sistema TECPlane está compuesto por tres capas a gran escala: el Cliente, la Aplicación y los Datos o Persistencia. El Cliente es punto de entrada donde interactúan los usuarios con el programa. La Aplicación es la fracción donde se maneja las solicitudes, conexiones y controles. Los Datos es donde guarda toda la información que se consume. En la Figura 1 se muestra un diagrama de arquitectura que lo visualiza.

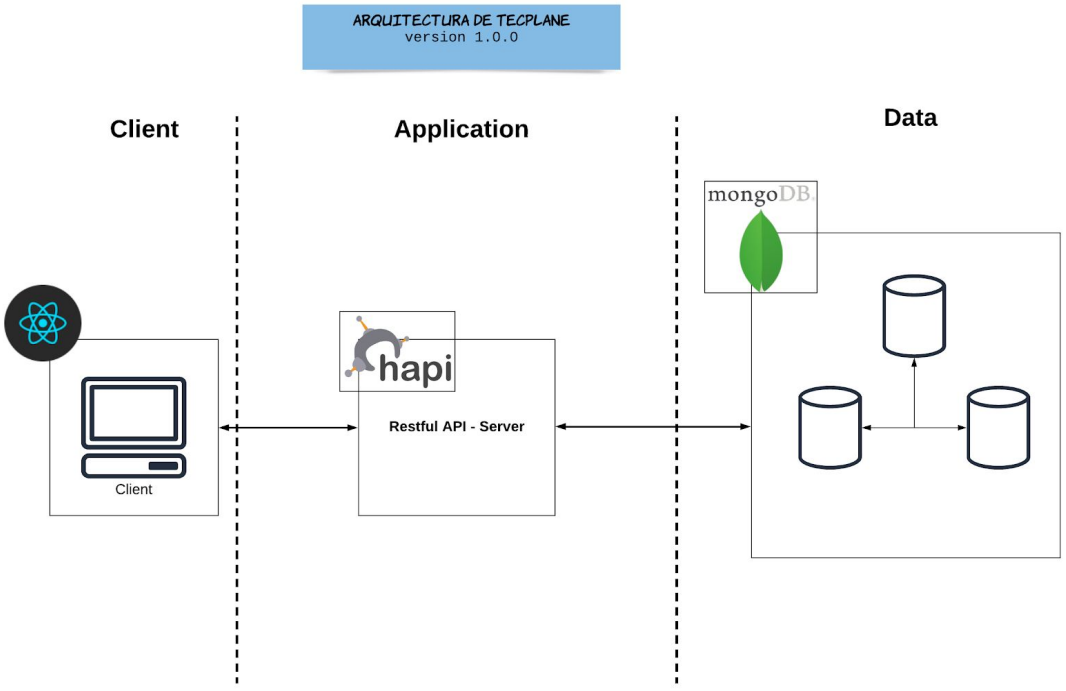

Figura 1. Diagrama de Arquitectura de TECPlane.

Como se puede observar anteriormente, el Cliente está construido usando la biblioteca de Interfaz de Usuario (*Front-End*) de JavaScript llamada ReactJS. La Aplicación es un servidor en NodeJS usando la biblioteca de HapiJS, este se comporta como un RESTful API para dirigir las peticiones del Cliente a hacia los Datos (*Back-End*). Por último, las Bases de Datos están basadas en el motor no relacional de MongoDB. En las siguientes secciones se entra en detalle de cómo está compuesta cada sección.

2. Base de Datos

2.1. Herramientas y Equipo

La base de datos utilizada fue MongoDB 4.2 que se corrieron en Windows 10 Pro, Windows 10, Windows 8.1 y Mac OS X 10.14.5. Para la manipulación de los datos aparte de la aplicación se usó únicamente el Mongo Shell y sus respectivos comandos. El sistema operativo es indiferente siempre y cuando la versión de la utilizada sea la misma.

2.2. Documentos y Colecciones

Se entiende que MongoDB es una base de datos no relacional basada en documentos y colecciones. Estos objetos utilizan el mecanismo de BSON, que es una forma binaria de JSON (*JavaScript Object Notation*). Acá Mongo tiene su propio procesamiento de datos utilizando la estructura de JSON, lo cual lo vuelve mi flexible y adaptativa para cualquier necesidad. Dentro de la jerarquía de este motor, la unidad más básica es un documento, luego, un conjunto de documentos forman una colección. Eventualmente, un conjunto de colecciones forman la base de datos total.

Lo útil de estas bases de datos es que no hay restricción en cantidad de documentos y colecciones, es más los documentos dentro de una colección dada no deben tener la misma estructura o datos. No obstante, en una colección específica lo más coherente sería que haya datos relacionados entre sí.

Para la base de datos de TECPlane, se usaron las siguientes colecciones:

- ***Airline***
- ***Airport***
- ***Employees***
- ***Flights***
- ***Passengers***
- ***Tickets***

La primera colección es ***Airline.*** Esta contiene la información de todas las aerolíneas asociadas a TECPlane. Los datos que contiene son los siguientes:

```
{
  _id: '',
  name: ',
  countries: [ ],
  airport_id: ''
}
```

El *_id* es un identificador único para cada aerolínea, está basado el código IATA de tres caracteres únicos para cada aerolínea. *name* corresponde al nombre, *countries* es un arreglo de los países a cuál esta aerolínea viaja. El *airport_id* es el identificador único del aeropuerto donde tiene la base central.

La segunda colección es ***Airport.*** Esta contiene la información de todos los aeropuertos asociadas a TECPlane. Los datos que contiene son los siguientes:

```
{
  _id: '',
  name: '',
  city: '',
  country: '',
  number: '',
  webPage: ''
}
```

El *_id* es un identificador único para cada aeropuerto, está basado el código IATA de tres caracteres únicos para cada aeropuerto. *name* corresponde al nombre, *city* y *country* son la cuidad y el país donde están ubicados. El *number* y *la webPage* son el número telefónico y la página web de cada aeropuerto para el contacto.

La tercera colección es ***Employee.*** Esta contiene la información de todos los distintos empleados registrados a TECPlane. Los datos que contiene son los siguientes:

```
{
    _id:'',
    firstName: '',
    lastName: '',
    username: '',
    password: '',
    role: '',
    area: '',
    initalDate: ''
}
```

El *_id* es un identificador único para cada empleado, el cual es la identificación del registro nacional. *firstName* y *lastName* corresponde al nombre y apellido, respectivamente. *username* y *password* son el usuario y la contraseña que se usa para acceder al sistema. El *role* y *la area* son el rol y la sección donde se desempeña actualmente dentro de TECPlane. *initialDate* es la fecha donde comenzó a laborar.

La cuarta colección es ***Flights.*** Esta contiene la información de todos los vuelos registrados a TECPlane. Los datos que contiene son los siguientes:

```
{
  _id:'' ,
  name: '',
  departure: '',
  arrives: '',
  takeOff: '',
  landing: '',
  origin: '',
  destination: '',
  price: '',
  restrictions: [],
  services: [],
  state: '',
  capacityPlane: '',
  ticketsSold: '',
  airline_id: ''
}
```

El *_id* es un identificador único para cada vuelo, es una cadena *random* de letras. *departure* y *arrives* son las horas de salida y llegada en notación de 24h, así como *takeOff* y *landing* son las fechas de salida y llegada. *origin* y *destination* son los países de donde sale a donde llega. *price* el valor del vuelo, restricción es un arreglo de las cosas no permitida en este vuelo y *services* un arreglo de que sí. El *state* es el estado actual del vuelo, puede ser *OnTime, Delayed* o *Cancelled. capacityPlane* es la cantidad de personas que pueden viajar y *ticketsSold* los boletos vendidos de este vuelo. Por último, está *airline_id* el cual es el identificador a la aerolínea que está dando este servicio.

La quinta colección es ***Passengers.*** Esta contiene la información de todos los pasajeros registrados a TECPlane. Los datos que contiene son los siguientes:

```
{
  _id: '' ,
  firstName: '',
  lastName: '',
  username: '',
  email: '',
  password:'',
  birthday:'',
  country: '',
  address: '',
  phone: []
}
```

El *_id* es un identificador único para cada pasajero, el cual es la identificación del registro nacional. *firstName* y *lastName* corresponde al nombre y apellido, *birthday* la fecha de cumpleaños en el formato: yyyy-dd-mm. *country* y *address* son el país y la dirección exacta de residencia, respectivamente. *username* y *password* son el usuario y la contraseña que se usa para acceder al sistema. *phone* es el número de teléfono.

La sexta colección es ***Tickets.*** Esta contiene la información de los vuelos y de los pasajeros, ya que, es la herramienta de acceso al vuelo por parte de los pasajeros. Como se explica a continuación, contiene *id's* externos siendo una especie de relación entre estas dos colecciones. Los datos que contiene son los siguientes:

```
{
  _id:'',
  amount:'' ,
  seats: [],
  baggage: '',
  carryOn: '',
  flight_id:'',
  passenger_id: '',
  checked: '',
  boarded: '',
  dateBought: ''
}
```

El *_id* es un identificador único para cada tiquete, el cual es una cadena de caracteres aleatorios. *passenger_id* es el identificador del pasajero que adquirió, por lo tanto, *amount* es la cantidad de espacios que compró y *seats* es un arreglo con los asientos que le corresponde, lo cual debe ser coherente con *amount*. *baggage* y *carryOn* son la cantidad de maletas de viaje y equipo de mano que van a llevar consigo. *flight_id* pertenece al identificador del vuelo, como se puede deducir, acá está la relación.

mounts y *lastName* corresponde al nombre y apellido, *birthday* la fecha de cumpleaños en el formato: yyyy-dd-mm. *country* y *address* son el país y la dirección exacta de residencia, respectivamente. *username* y *password* son el usuario y la contraseña que se usa para acceder al sistema. *phone* es el número de teléfono.

2.3. Replicación y Distribución

TECPlane emplea un sistema de replicación de toda la información en la base de datos en tres distintos nodos. El objetivo de esto es lograr tener toda la información siempre disponible en caso de que haya una caída. Esto permite que el sistema siempre esté disponible.

La replicación funciona de la siguiente manera: un nodo maestro y dos nodos esclavos, en caso de una caída, el sistema de MongoDB encuentra el próximo nodo y lo asume como el nuevo maestro. Los pasos para lograr esto fueron los siguientes:

1. Crear una carpeta en C:\. Dentro de esta debe haber una subcarpeta llamada ***data***. Dentro de la carpeta ***data*** crear las carpetas ***db*** y ***log***. Realizar esto para cada nodo deseado del clúster.

2. Dentro de la carpeta **log**, crear el archivo ***mongod.log.***

3. Abrir una terminal de comando:

   ```
   mongod --bind_ip_all --port 27017 --dbpath C:\mongodb\data\db --replSet "replSet"
   ```

   --port: indica el Puerto. El Puerto que MongoDB utiliza por defecto es el 27017.
   --dbpath: representa la ubicación donde se creará la base. La mostrada en el comando sirve de ejemplo.
   --replSet será el nombre de la replicación. En este ejemplo se usa “replSet” como nombre.

   Repetir este paso en cada nodo para para “encenderlo”.

4. Abrir otra terminal y ejecutar el commando:

   `mongo --port 27017 --replSet "replSet"`

   --port: especifica el puerto.

   --replSet: es el nombre del replica set al que se quiere conectar.

5. Abrir una terminal de comando:

   `rs.initiate( { _id : "replSet", members: [{ _id: 0, host: "Hostname:Port" }, { _id: 1, host: " Hostname:Port " }, { _id: 2, host: " Hostname:Port " }]})`

   Este comando inicializa todos los nodos del clúster. Es posible ejecutarlo con solo un nodo principal. Es posible ejecutar este comando sin parámetros para usar la configuración por defecto. Es importante notar que si se utiliza en *Hostname* "*localhost*" (127.0.0.1) esto evita que otros IP diferentes de *localhost* se puedan conectar al clúster.

   Para ver la configuración del *set*, ejecutar: `rs.conf()`. Para ver el *status* del *replica set*, ejecutar: `rs.status()`.

6. Para agregar otro nodo al set: `rs.add("Hostname:Port").`

7. Para darle acceso *Read* a un nodo:
   a. Conectarse a dicho nodo.
   b. Ejecutar: `rs.SlaveOk()`

8. Para cambiar la prioridad de cada nodo:
   a. Convertir la configuración en una variable: `cfg = rs.conf().`
   b. Cambiar la prioridad: `cfg.members[0].priority = <número>` (por ejemplo: 1, 2, 3, 0.5)
   c. Ejecutar: `rs.reconfig(cfg).`

9. Para cambiar la versión de protocolo:
    a. Convertir la configuración en una variable: `cfg = rs.conf()`.
    b. `cfg.protocolVersion = 1`
    c. Ejecutar: `rs.reconfig(cfg)`.

10. Para conectarse desde una aplicación externa al nodo maestro:

```
mongo --host
replicaSetName/host1[:porthost1],host2[:porthost1],host3[:por
thost3],etc
```

3. Cliente

3.1. Herramientas y Equipo

El Cliente es la sección donde el usuario interactúa con el sistema, visto en la Figura 1, página 2. Para desarrollarlo se utilizó ReactJS v16.9, usando NodeJS v10.16 y NPM v6.10. Es una aplicación web, un página dinámica, corre en todo los principales navegadores (Mozilla Firefox, Google Chrome y Apple Safari) y todo computador sin importar el sistema operativo.

3.2. Funcionamiento

El Cliente está compuesto por una serie componentes usando la premisa nativa de ReactJS, un conjunto de componentes trabajando juntos que reaccionan según cada evento, haciendo el código reutilizable en múltiples partes.

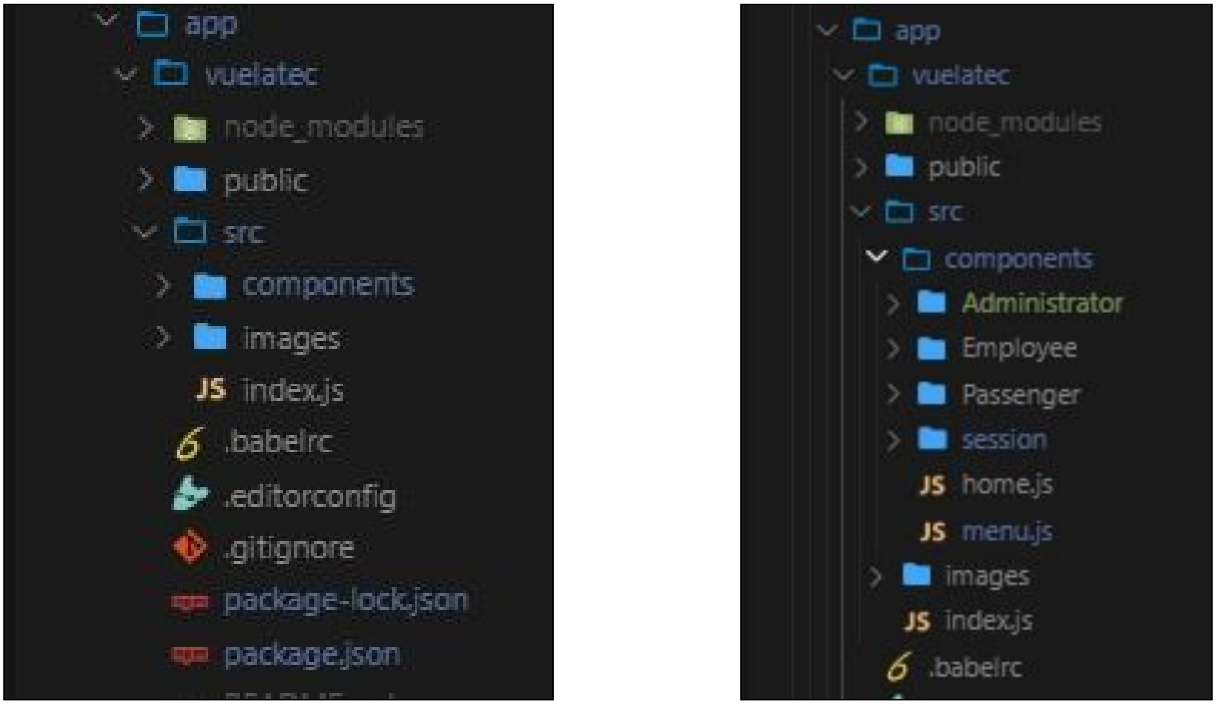

Figura 1. Anatomía de la solución del cliente usando ReactJS.

a) Captura derecha: completa de la solución. **b)** Captura izquierda: de los principales componentes.

La carpeta de *node_modules* son las dependencias propias del proyecto instaladas y usadas por medio de NPM y NodeJS. La carpeta *src* es donde está el código: las vistas de las páginas corresponden a los componentes que ya se mencionaron. El resto de los archivos son meramente de configuración y de mantener corriendo la aplicación. Es importante notar que se requiere conocimiento en JavaScript ya que estas son bibliotecas basada en este lenguaje.

3.3. Reportes

A continuación, se muestran las capturas de pantallas de los reportes solicitados. Estos están organizados por el usuario que los ejecuta: administrador, empleado o pasajero.

3.3.1. Administradores

Los siguientes reportes son efectuados únicamente por un empleado del rol de administrador. Esto quiere decir que son exclusivos y ningún otro tipo de empleado puede realizarlos. Estos son reportes del tipo administrativo y gerencial.

Código Aerolinea	Boletos Vendidos	Monto Total
AAL	4	$400
SHT	6	$210
AAL	0	$0
UAE	120	$16800

Figura 2. Reporte de la ganancia total de todas las aerolíneas.

Identificación	Minimo	Maximo
123454325	1	5

Figura 3. Reporte de los boletos comprado por pasajero.
La búsqueda se realiza por el número de identificación.

Reportes Administrativos

Ganancia Total de Vuelos | Boletos Comprados por Pasajero | Destinos más Visitados | Boletos Registrados en el Sistema | Top 3 Pasajeros

Destino	Cantidad de pasajeros con boleto
Atlanta	120
Frankfurt (Order)	6
Tokyo	4

Figura 4. Reporte de los destinos más visitados.

Reportes Administrativos

Ganancia Total de Vuelos | Boletos Comprados por Pasajero | Destinos más Visitados | Boletos Registrados en el Sistema | Top 3 Pasajeros

Identificación: 157483924

Estado: Seleccionar un estado de vue

Desde: mm/dd/yyyy

Hasta: mm/dd/yyyy

Cantidad Operaciones	Identificador Vuelo	Nombre	Origen	Destino	Estado	Código de Aerolinea	Boletos
11	AA4573	AtlTok	Atlanta	Tokyo	OnTime	AAL	Código: ktl123 | Pasajero: Giovanni - 123454325 | Cantidad: 1. Código: sht123 | Pasajero: Giovanni - 123454325 | Cantidad: 5. Código: sht125 | Pasajero: Laurel - 157483924 | Cantidad: 3. Código: hgn135 | Pasajero: Laurel - 157483924 | Cantidad: 2.
1	DU1289	DuUs	Dubai	Atlanta	OnTime	UAE	Código: dua432 | Pasajero: Laurel - 157483924 | Cantidad: 1.

Figura 5. Reporte de los boletos registrados.

La búsqueda se realiza por identificación del pasajero, un rango de fechas o estado de un vuelo.

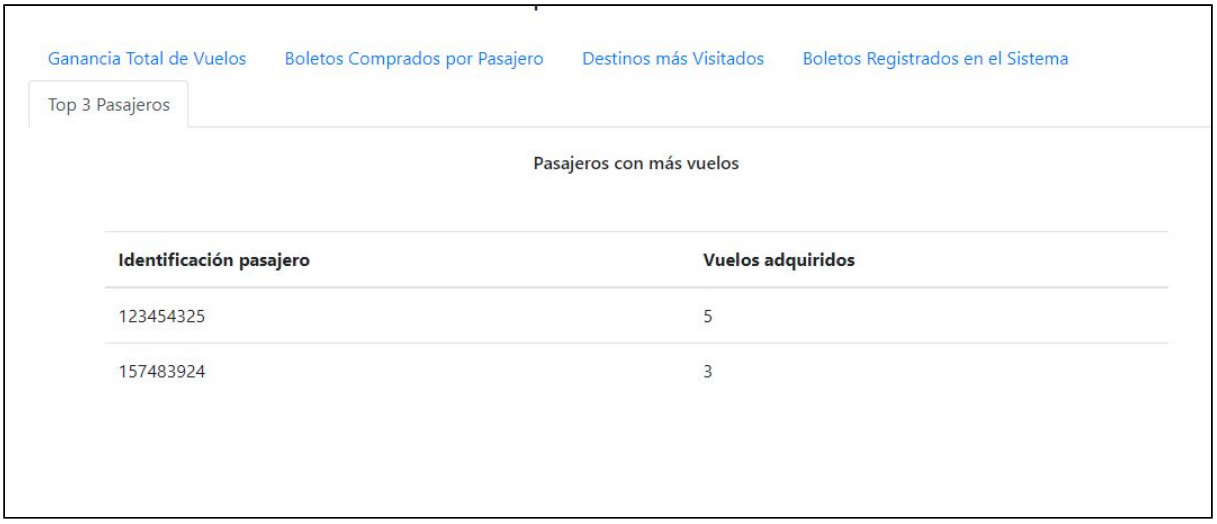

Figura 6. Reporte de los tres pasajeros con más vuelos.

3.3.2. Empleados

Los siguientes reportes se asocian a los empleados de TECPlane. Estos se encargan de procesar información de los vuelos, tiquetes y de los pasajeros. Estos reportes son exclusivos a este rol.

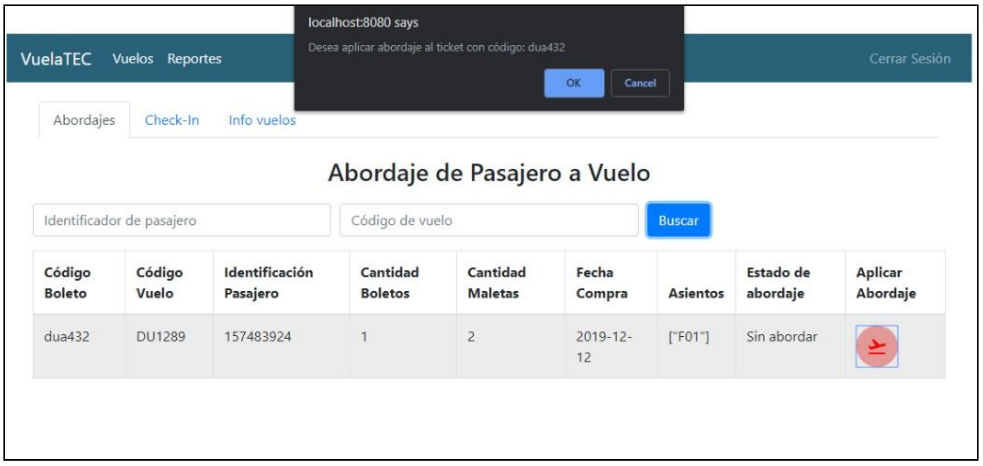

Figura 7. Reporte de la ejecución de un abordaje de uno o varios pasajeros asociados a un tiquetes. La búsqueda se realiza por la identificación del pasajero o el código de vuelo.

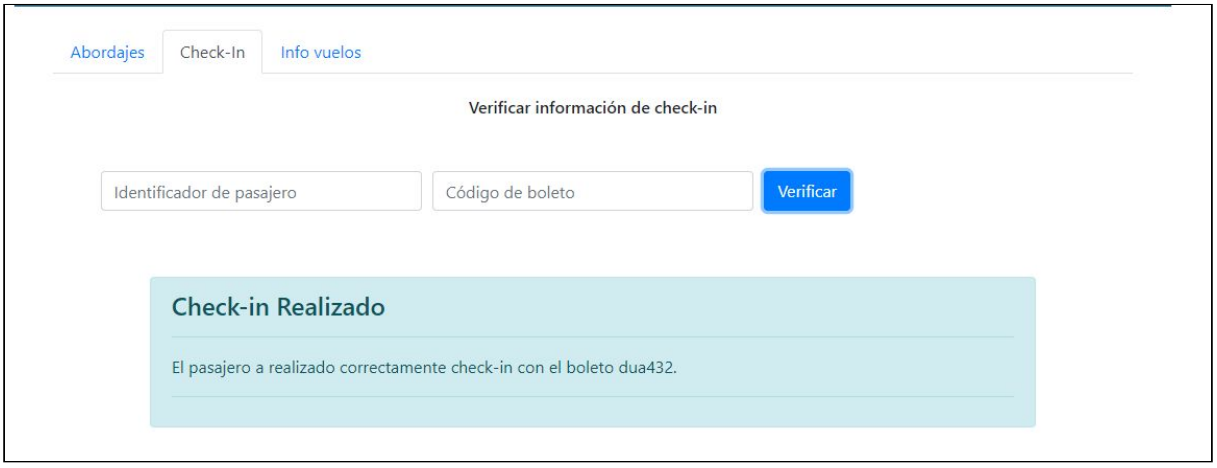

Figura 8. Reporte de verificación si un usuario haya realizado *check-in.*

Abordajes | Check-In | Info vuelos

Ver Vuelos

Identificador	Nombre	Origen	Destino	Itinerario	Precio	Estado	Capacidad permitida	Código de Aerolinea	Restricciones
AA4573	AtlTok	Atlanta	Tokio	Salida: 2019-12-2 [19:00] - Llegada: 2019-13-2 [09:00]	$ 100	OnTime	250	AAL	["weapons","foods","ani
LLU8965	SmFr	London	Frankfurt	Salida: 2019-3-2 [18:00] - Llegada: 2019-3-2 [21:00]	$ 35	OnTime	150	SHT	["weapons","foods","ani
DU1289	DuUs	Dubai	Atlanta	Salida: 2019-5-6 [10:00] -	$ 140	OnTime	450	UAE	["weapons","foods","ani

Figura 9. Reporte de la información de todos los vuelos.

Reportes para Funcionarios

Información de pasajero | Vuelos Registrados en el Sistema

Consultar información de un pasajero

Numero de cedula

Cedula de pasajero

Buscar

Identificación	Nombre	Usuario	Correo electrónico	País	Dirección	Telefonos	Fecha Nacimiento	Tickets-Info
987654321	prueba test1	test	test@email.com	Armenia	Ashtarak	[8899775566]	1990-11-30	

Figura 10. Reporte de la la información de todos los pasajeros.
La búsqueda se realiza por la identificación del pasajero.

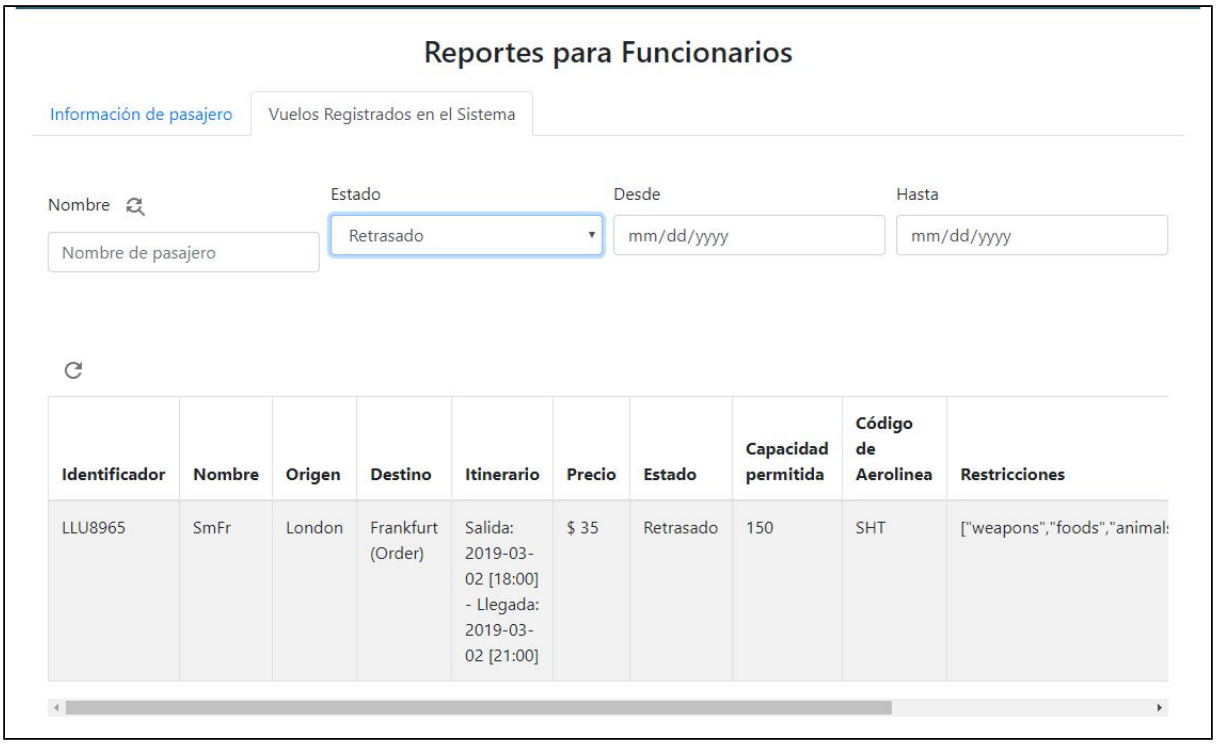

Figura 11. Reporte de los vuelos realizados por los pasajeros.
La búsqueda se realiza por nombre del pasajero, un rango de fechas o el estado del vuelo.

3.3.3. Pasajeros

Los siguientes reportes pertenecen únicamente a los pasajeros, ya que se asocian a las acciones que estos pueden realizar en el sistema, por lo tanto también son exclusivos.

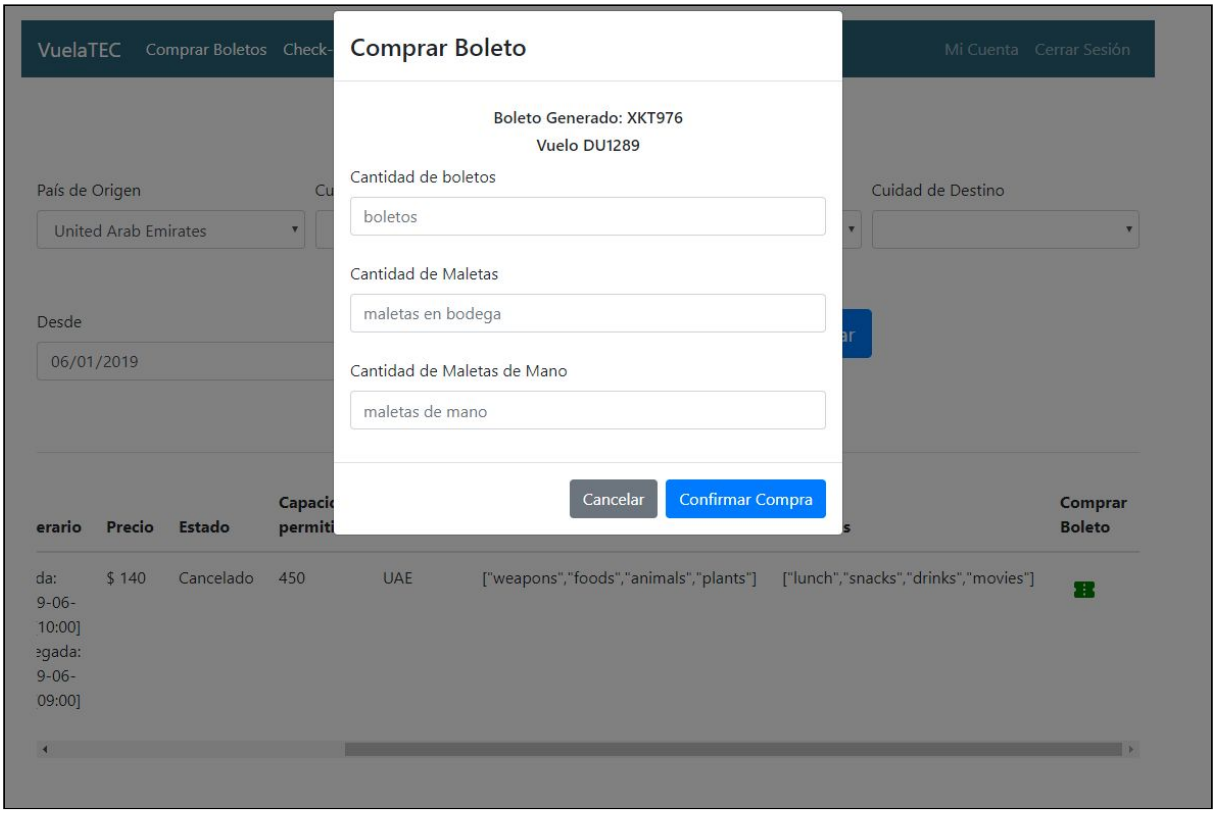

Figura 12. Reporte de compra de un boleto disponibles.
La búsqueda se realiza por ciudad y país de destino y origen, y un rango de fechas.

Check-In | Info vuelos

Ver Vuelos

Identificador	Nombre	Origen	Destino	Itinerario	Precio	Estado	Capacidad permitida	Código de Aerolinea	Restricciones
AA4573	AtlTok	Atlanta	Tokio	Salida: 2019-12-2 [19:00] - Llegada: 2019-13-2 [09:00]	$ 100	OnTime	250	AAL	["weapons","foods","ani
LLU8965	SmFr	London	Frankfurt	Salida: 2019-3-2 [18:00] - Llegada: 2019-3-2 [21:00]	$ 35	OnTime	150	SHT	["weapons","foods","ani
DU1289	DuUs	Dubai	Atlanta	Salida: 2019-5-6 [10:00] -	$ 140	OnTime	450	UAE	["weapons","foods","ani

Figura 13. Reporte de ver todos los vuelos disponibles.

Vuelos | -

Consultar mis vuelos

Estado: Seleccionar un estado de vuelo

Desde: mm/dd/yyyy

Hasta: mm/dd/yyyy

Código Boleto	Código Vuelo	Identificación Pasajero	Cantidad Boletos	Cantidad Maletas	Fecha Compra	Asientos	Estado de abordaje	Estado de check-in
sht125	AA4573	157483924	3	3	2019-15-03	["E01","E02","E03"]	Sin abordar	Sin chequear
hgn135	AA4573	157483924	2	2	2019-18-06	["G02","G03"]	Sin abordar	Sin chequear
dua432	DU1289	157483924	1	2	2019-12-12	["F01"]	Sin abordar	Sin chequear

Figura 14. Reporte de ver todos todo los vuelos que un dado pasajero ha comprado.

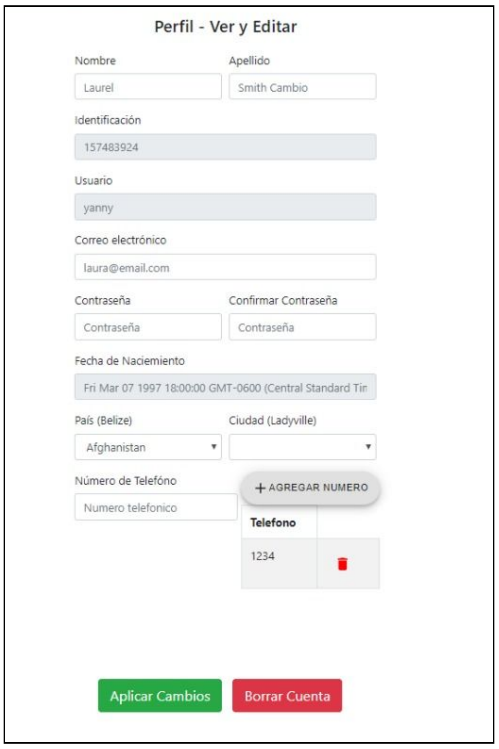

Figura 15. Reporte de edición de la información de una cuenta de un usuario.

3.3.4. Funcionalidades Otras

Los reportes anteriores no son las únicas que acciones que se pueden ejecutar en el el sistema de TECPlane. Los empleados y pasajeros sí tienen las funciones limitadas de las acciones que pueden hacer, pero el administrador no. Como su nombre lo dice debe administrar toda la información que entra y sale del sistema. Por ende, puede realizar un *CRUD*, es decir *Create Review Update Delete*, de las instancias en el sistema, estas incluyen aerolíneas, aeropuertos, empleados y vuelos.

A continuación, se muestran capturas de pantalla con su respectiva descripción de estas las acciones mencionadas.

3.3.4.1. Aerolíneas

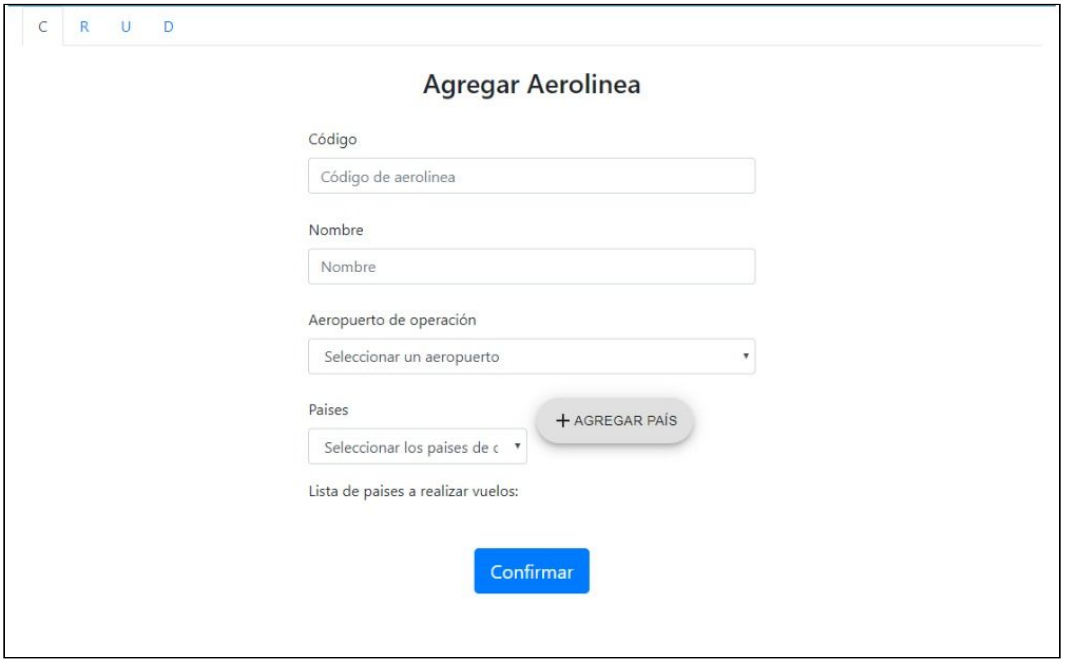

Figura 16. Creación de una aerolínea.

C R U D

Ver Aerolineas

Identificador	Nombre	Aeropuerto de operación	Países
AAL	American Airlines	ATL	["Australia","Aruba","Brasil","United Arab Emirates","Costa Rica","Egypt","United Kingdom","United States","Germany","Japan"]
CLH	Lufthansa	FRA	["Australia","Brasil","United Arab Emirates","Costa Rica","Egypt","United Kingdom","United States","Germany","Japan"]
SHT	British Airways	LHR	["Australia","Aruba","Brasil","United Arab Emirates","Costa Rica","Egypt","United Kingdom","United States","Germany","Japan"]
UAE	Emirates	DXB	["Australia","Brasil","United Arab Emirates","Egypt","United Kingdom","United States","Germany","Japan"]
JBU	JetBlue	ATL	["Brasil","Costa Rica","United Kingdom","United States","Germany","Japan"]
QTR	Qatar Airways	DXB	["Australia","United Arab Emirates","Egypt","United Kingdom","United States","Germany","Japan"]

Figura 17. Consulta de todas las aerolíneas.
La búsqueda no tiene filtro ya que muestra todas las aerolíneas registradas y activas a la fecha.

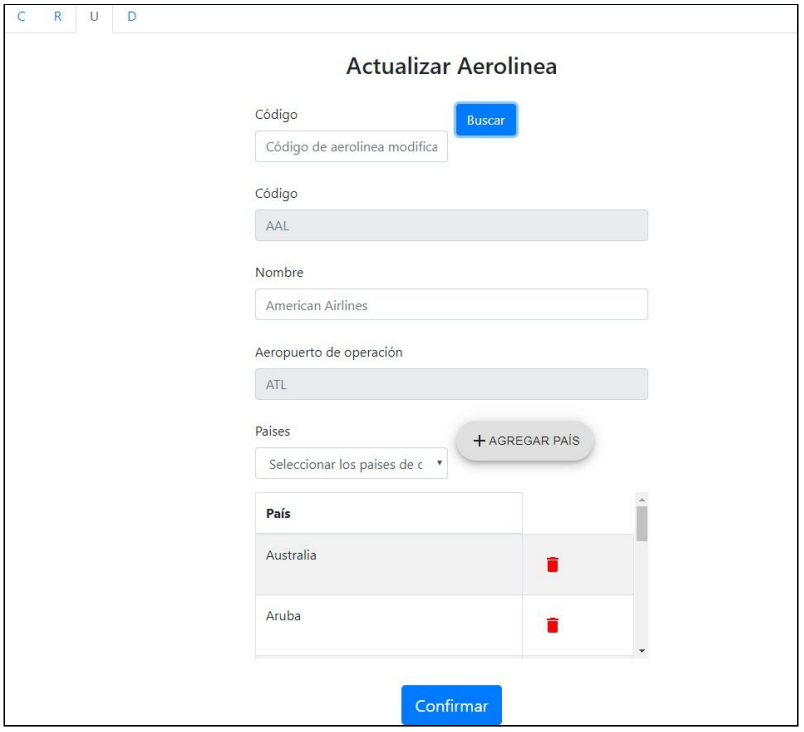

Figura 18. Edición de una aerolínea.
Se editan por medio de su identificador único.

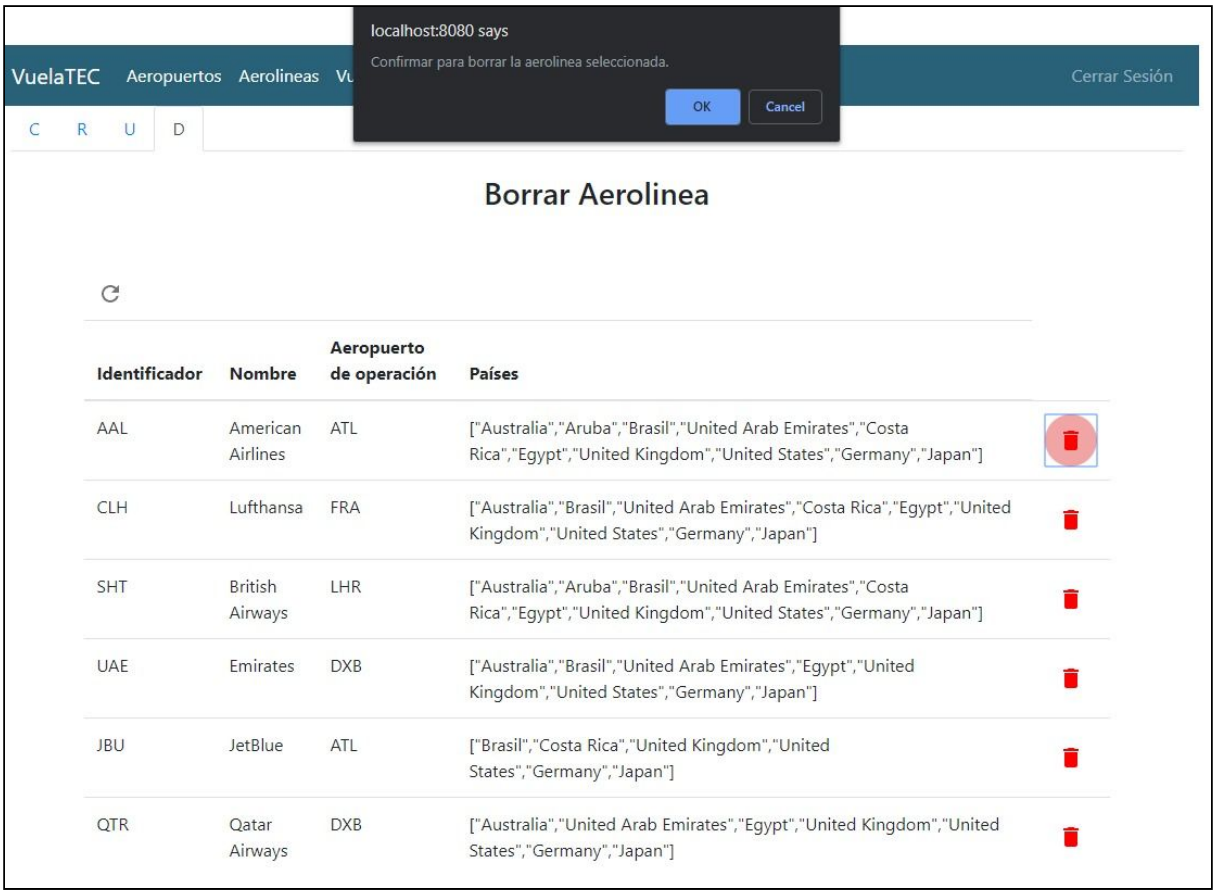

Figura 19. Eliminación de una aerolíneas.
Se hace una consulta de todas las aerolíneas y escoge cuál se quiere eliminar.

3.3.4.2. Aeropuertos

Figura 20. Creación de un aeropuerto.

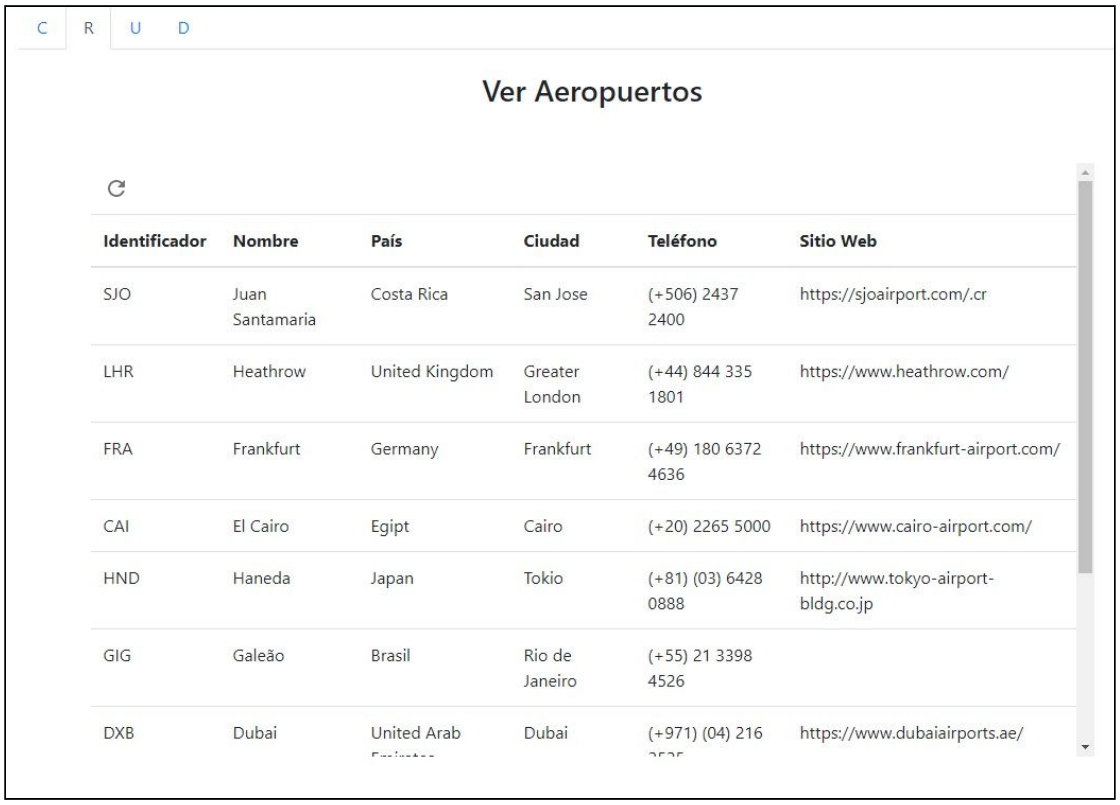

Figura 22. Consulta de todos los aeropuertos.

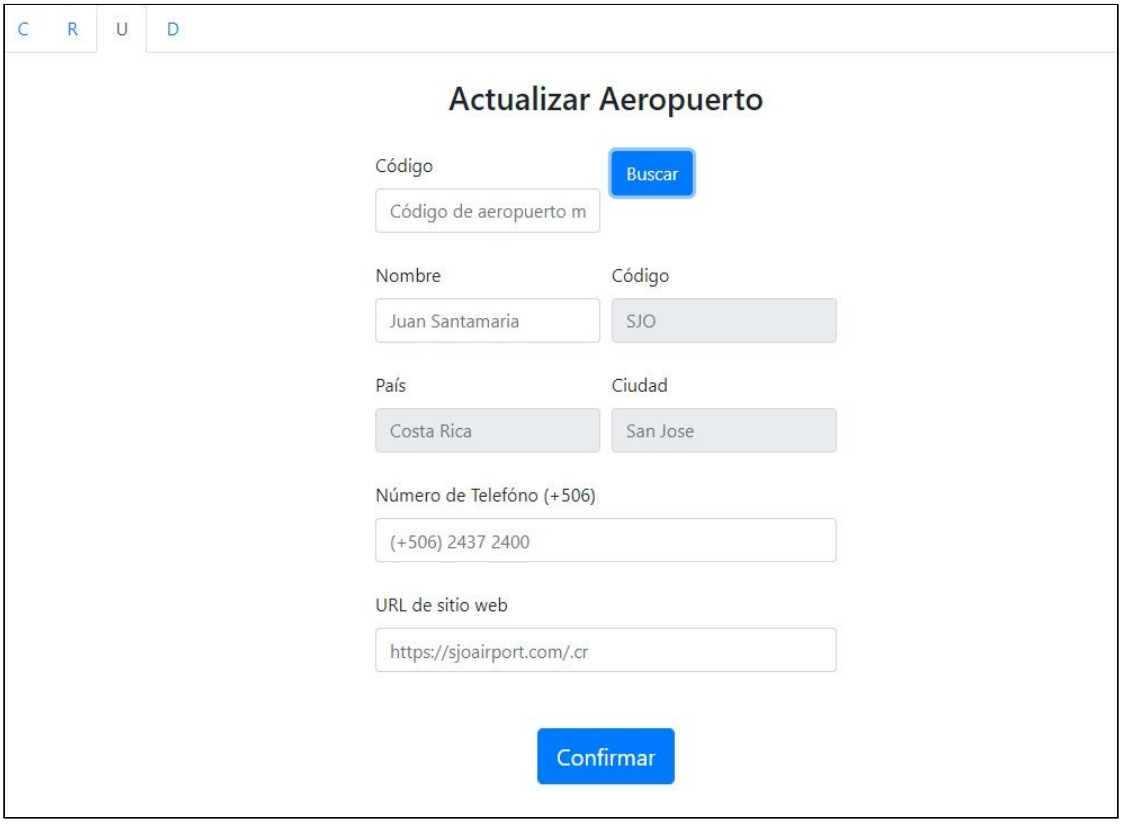

Figura 23. Edición de una aeropuerto.
Se editan por medio de su identificador único.

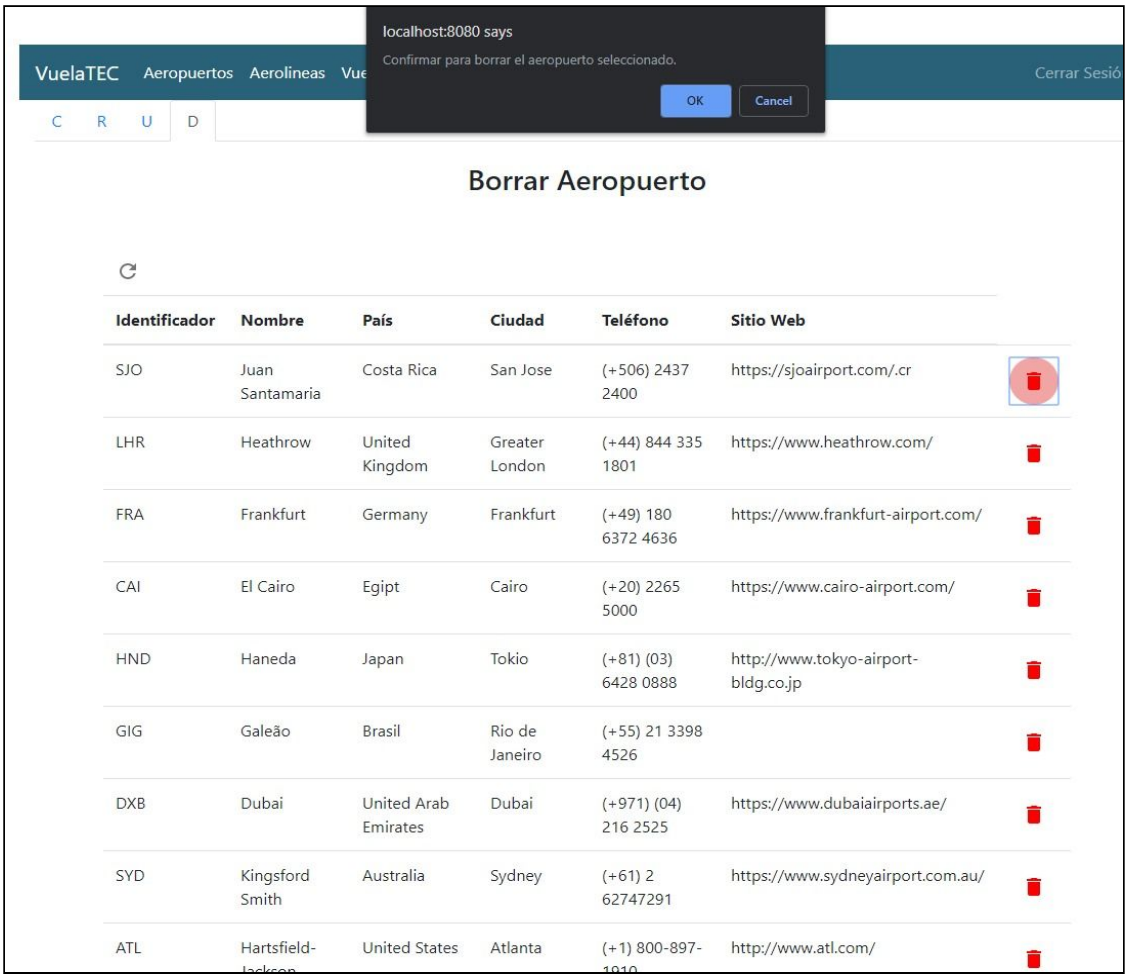

Figura 24. Eliminación de un aeropuerto.
Se hace una consulta de todos los aeropuertos y escoge cuál se quiere eliminar.

3.3.4.3. Empleados

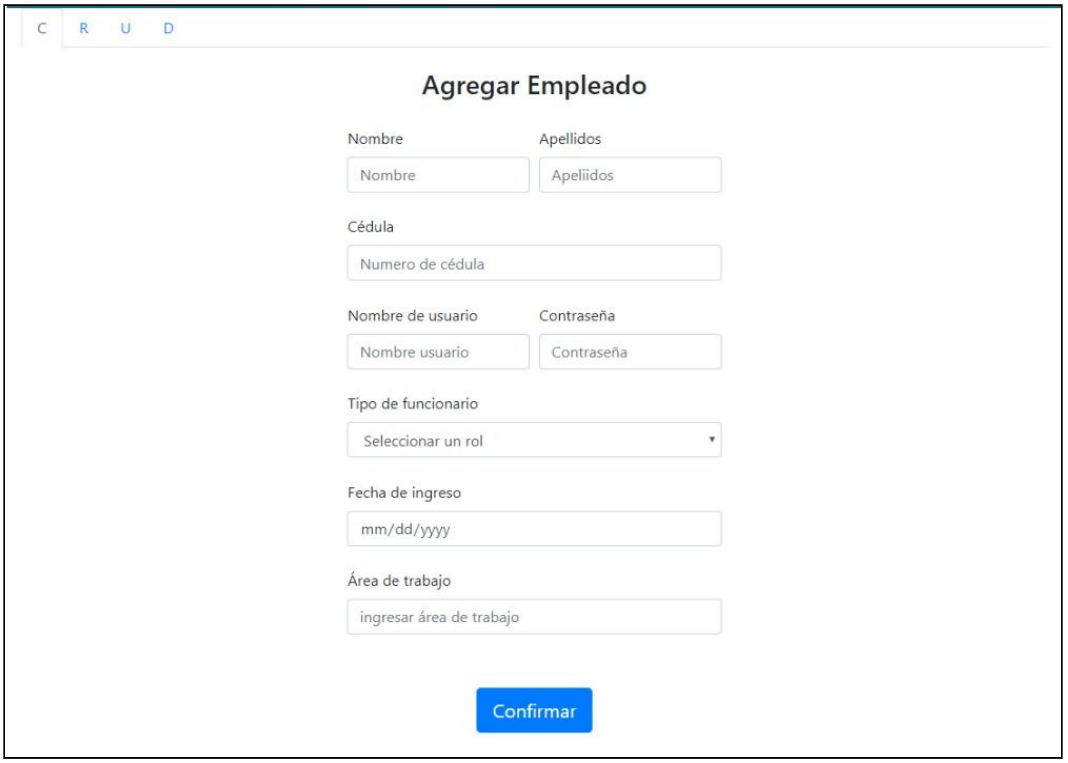

Figura 25. Creación de un empleado.

Identificador	Nombre	Usuario	Rol	Fecha de ingreso
657483056	Camila Murrillo	cammu	administrator	2010-04-03
2483429504	Andrea Retana	andret	operator	2011-14-07
1163428574	Ilana Chavarria	ilchav	operator	2015-23-05
3547318409	Jorge Lizano	jgliz	technician	2008-03-12
45647329	Paul Greenwood	plfw	technician	2013-07-02

Figura 26. Consulta de todos los empleados.

La búsqueda no tiene filtro ya que muestran todos las empleados registrados y activos a la fecha.

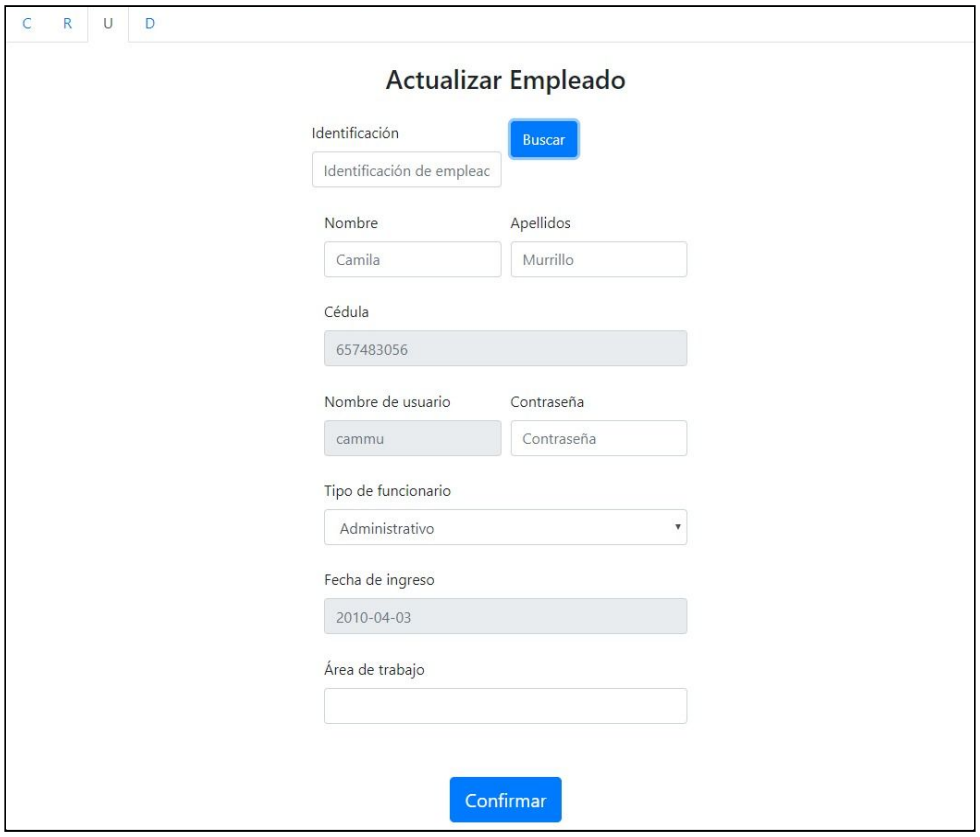

Figura 27. Edición de un empleado.
Se editan por medio de su identificador único.

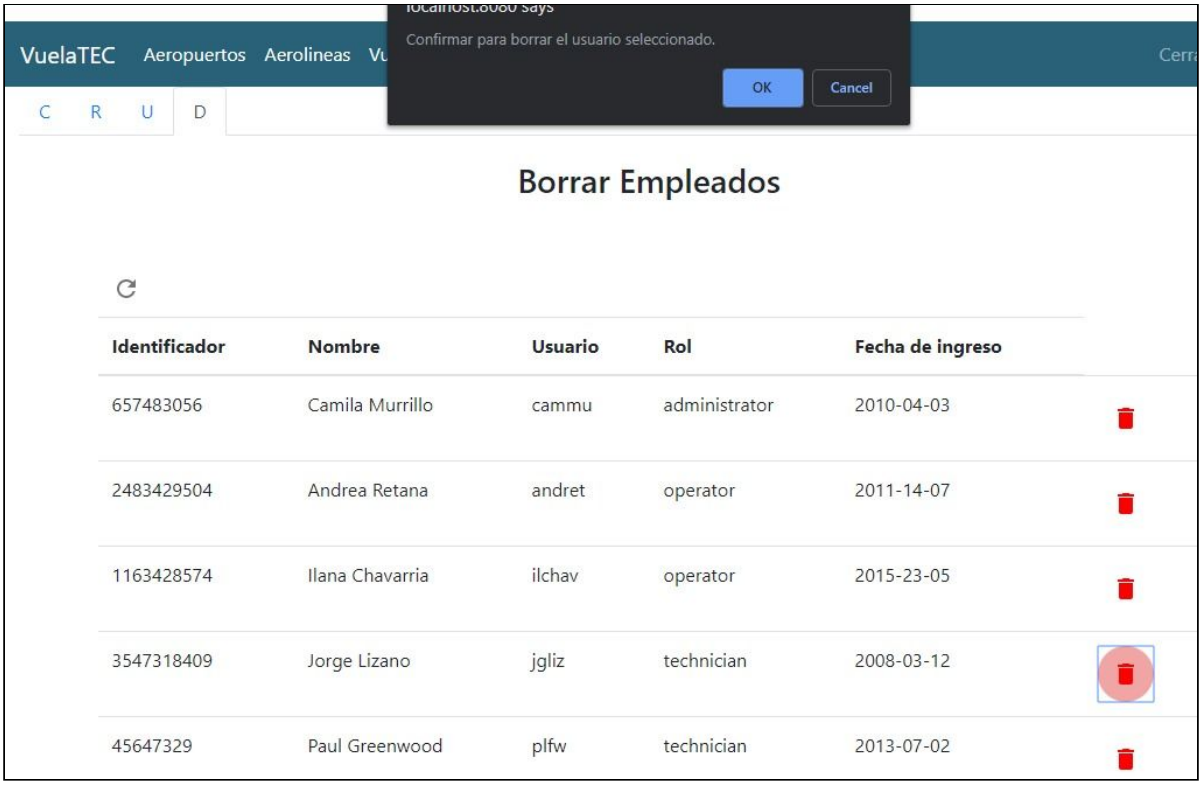

Figura 28. Eliminación de un empleado.
Se hace una consulta de todos los aeropuertos y escoge cuál se quiere eliminar.

3.3.4.4. Vuelos

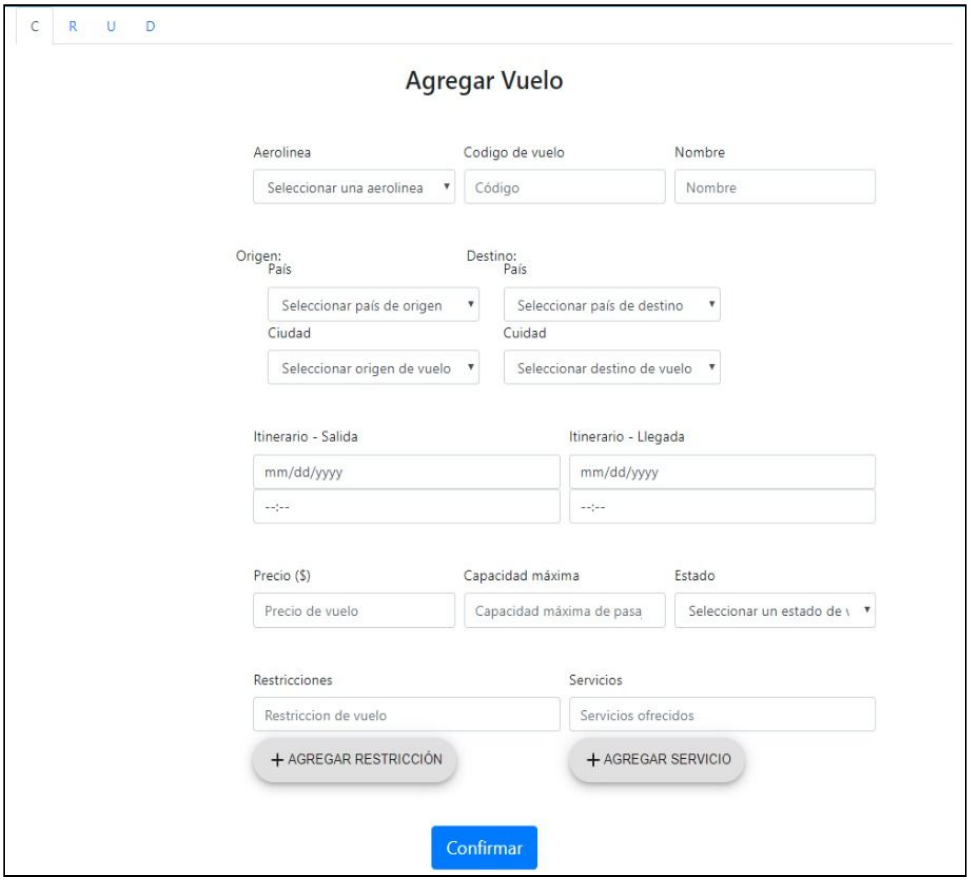

Figura 29. Creación de un vuelo.

Identificador	Nombre	Origen	Destino	Itinerario	Precio	Estado	Capacidad permitida	Código de Aerolinea	Restricciones
AA4573	AtlTok	Atlanta	Tokio	Salida: 2019-12-2 [19:00] - Llegada: 2019-13-2 [09:00]	$ 100	OnTime	250	AAL	["weapons","foods","a
LLU8965	SmFr	London	Frankfurt	Salida: 2019-3-2 [18:00] - Llegada: 2019-3-2 [21:00]	$ 35	OnTime	150	SHT	["weapons","foods","a
DU1289	DuUs	Dubai	Atlanta	Salida: 2019-5-6 [10:00] -	$ 140	OnTime	450	UAE	["weapons","foods","a

Figura 30. Consulta de todos los vuelos.
La búsqueda no tiene filtro ya que muestran todos los vuelos registrados y activos a la fecha.

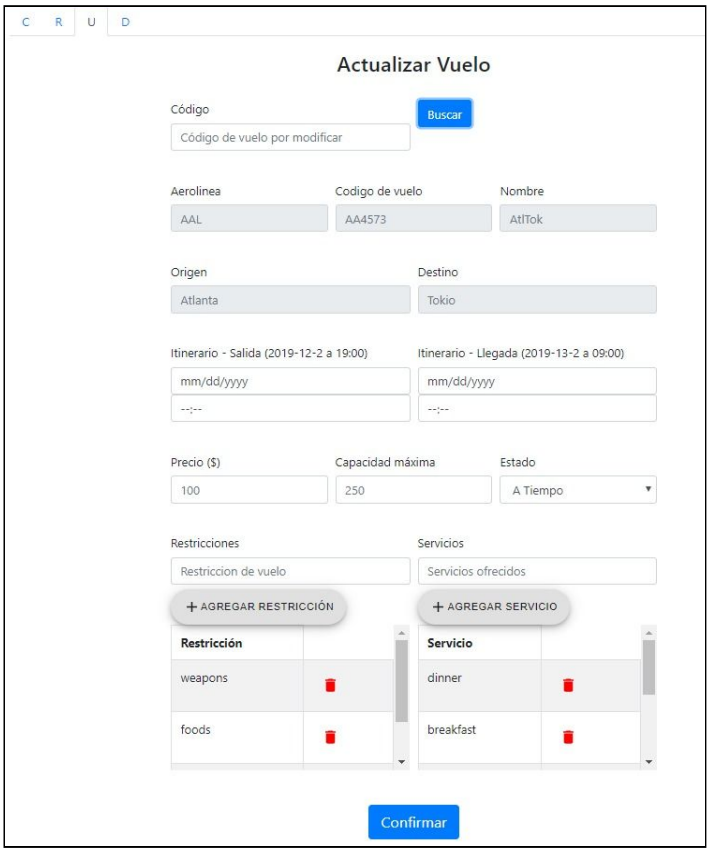

Figura 31. Edición de un vuelo.
Se editan por medio de su identificador uno.

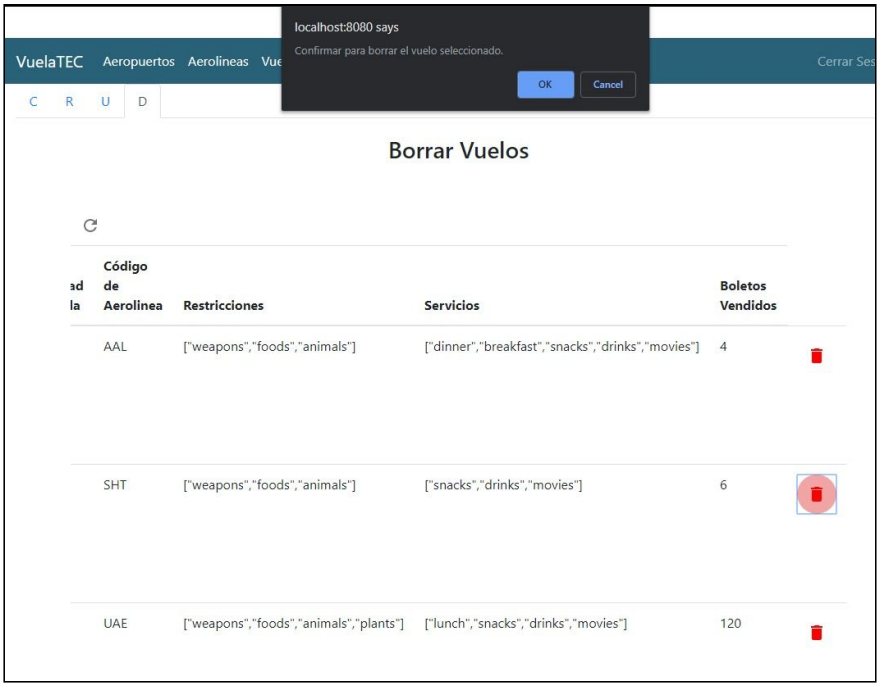

Figura 32. Eliminación de un vuelo.
Se hace una consulta de todos los vuelos y escoge cuál se quiere eliminar.

4. Aplicación

4.1. Herramientas y Equipo

La Aplicación es la sección que actúa como intermediaria entre el Cliente y los Datos, como está ilustrado en cada en la Figura 1 en la página 2. Es un servidor RESTful API para poder conectar y. realizar las consultas hacia la base de datos. Está construido usando HapiJS v18.4, que es una biblioteca de NodeJS y NPM, utilizando las mismas versiones que se usaron para el Cliente.

4.2. Funcionamiento

El RESTful API es un servidor que redirige las consultas a la base de datos, utilizando métodos de HTTPS, se hacen las consultas del CRUD por medio de las acciones POST, GET, PUT, DELETE, UPDATE. Asimismo, en caso de que la conexión se pierda debido a algún fallo de la base de datos, éste redirige al nodo central. Para el este caso el nodo principal no presenta fallas por lo cual siempre está corriendo.

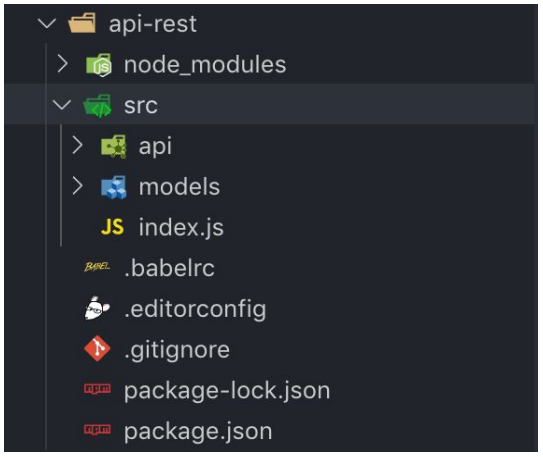

Figura 33. Anatomía de la solución del servidor usando HapiJS.

Como se puede observar, la estructura del servidor es muy parecida al del Cliente puesto que están construidos usando soluciones de NodeJS y NPM. La diferencia radica en que en *src* hay dos carpetas *models* y *api*. La primera presenta la conexión y la reconexión a las bases de datos iniciales y de recuperación en caso de fallo. La segunda carpeta, tiene una subcarpeta *v1* que contiene las rutas y *paths* para efectuar las consultas respectivas.

5. Conclusiones

El uso de la base de datos no relacional, en este caso MongoDB, resultó muy sencillo porque el equipo ya ha utilizado esta herramienta en proyectos pasados. El reto acá fue se basó en el modelaje de los datos, ya que, tales niveles de *queries* e información que se relacionaba era un poco más complejo a lo que se había hecho con anterioridad.

La que pueda ser la relación más importante del sistema es la información de los pasajeros, los tiquetes y los vuelos, ya que, estos se dependen los tiquetes dependen de un pasajero y de un vuelo existente. A la hora de encontrar como asociar tal información hay múltiples formas, la primera a probar fue la de crear documentos embebidos, es decir, crear un subdocumento dentro de otro. Pero a la hora de personalizar las consultas se hacían muy complejas. La solución fue usar las propuestas de la documentación de MongoDB Docs la cual es usar identificadores por documentos y colocarlos en otros documentos aparte, creando relaciones de uno a uno y de uno a muchos, en caso de ser necesario.

Los *queries* o consultas son sencillas, es saber cuál operador utilizar y cómo aplicarlo según lo que se quiere. Es como crear funciones de JavaScript basado en parámetros de JSON, entonces resultado fácil como se quiere la búsqueda. La única contraparte es que no siempre se obtiene resultados tan “limpios” como en SQL normal, a veces se obtiene parámetros extras que añaden más cosas al resultado de la consulta final.

No obstante, usar MongoDB es fácil, hay mucha libertad a la hora de diseñar las colecciones y documentos. No hay reglas ni restricciones que seguir, los tipos de datos son muy variados y se pueden crear tantos documentos variados como se requieran.

La replicación es muy sencilla, mucho más que en una base de datos relacional. La ventaja es que hay no acá fragmentación entonces todos los datos se replican y se evitan ese trabajo. Como se mostró en la sección de la replicación, los pasos son muy simple y todos son en la terminal de comando. El hecho que no tenga alguna ventana visual o un *wizard* de instalación lo hace más ligero y rápido en cuestiones de procesamiento.

Se tienen disponible las computadoras las cuales se van a usar como nodos, se conectan en la misma red, se crea el set de replicación y se coloca un maestro, luego se crean y se unen los nodos esclavos al conjunto y listo. MongoDB tiene un conjunto de configuraciones ya listas en caso de que se pierda un nodo, se pierda la conexión y también en el manejo y distribución de la información. Esto permite que uno se enfoque en la información.

El cliente, la página web, es producto sencillo, al tener experiencia usando esta herramienta es simple de utilizar, no toma tiempo por la práctica, es simplemente ajustar las necesidades a lo que se especifica. Asimismo, la parte del servidor RESTful y la API ya se tenía práctica de proyectos anteriores por lo que se recicló código útil y se ajustó.

El problema de usar herramientas con NodeJS es lo pesado que puede llegar a ser por las dependencias de *node_modules.* Entre más dependencias y requerimientos llega a ocupar más espacio y un poco más complicado de manejar. Otro detalle que ya se tiene contemplado es que se requiere una curva de aprendizaje empezando desde JavaScript, ya que, puede ser difícil de entender o seguir debido a la complejidad que tiene. Sin embargo, es una solución sencilla, rápida y fácil una vez que se entiende.

Bibliografía

MongoDB. (s.f). *Welcome to the MongoDB Docs.* [Blog]. Tomado de: https://docs.mongodb.com/