TEC | Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores
CE-3101 - Bases de Datos

Documentación Técnica

Profesor:

Marco Rivera Meneses

Estudiantes:

- David Alberto Robles Vargas
- Carlos Andrés Mata Calderon
- Luis Felipe Vargas Jiménez
- Jose Ignacio Calderón Diaz
- Jose Carlos Umaña Rivera

II Semestre, 2023

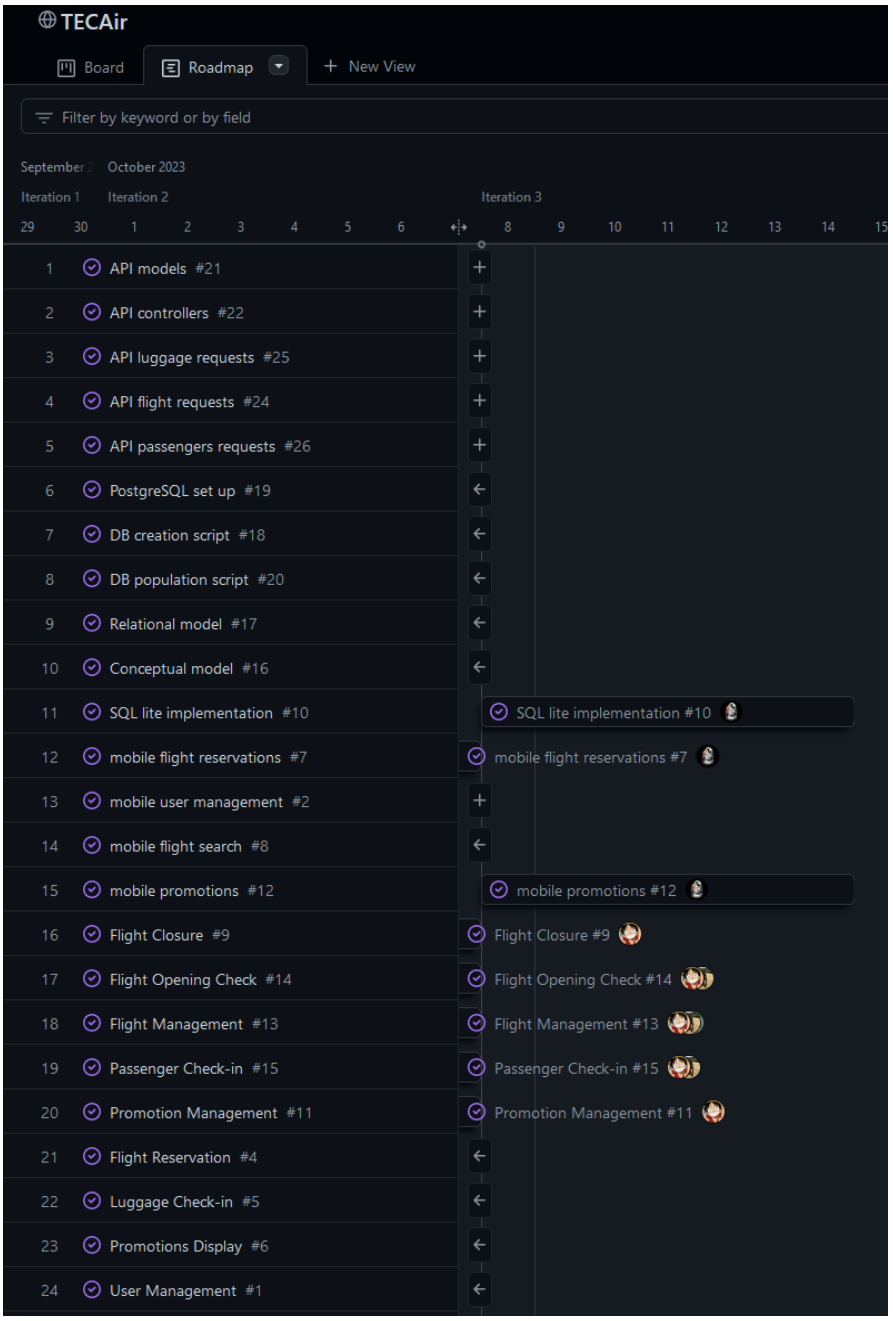
Repositorio de Github

https://github.com/Bases-Crew/TECAir

Plan de Trabajo en Github

Para más detalles visite la pestaña Board y Roadmap en el enlace a continuación:

https://github.com/orgs/Bases-Crew/projects/1



Estructuras de datos desarrolladas

Debido a la naturaleza y funcionalidad de los vuelos encontrada en el desarrollo del proyecto, la tabla “Flights” aporta información sobre el inicio y el final de un vuelo, pero en sí el viaje es delegado a la tabla “Stops”. Es decir, si un vuelo sale de San José, Costa Rica y termina en New York, USA; existirá una o varias filas en la tabla de escalas que harán referencia tanto al inicio como al final del vuelo. Con este peculiar enfoque se le permite a diferentes pasajeros el comprar “subvuelos” de manera individual, que están asociados a un vuelo principal (tabla Flight), en donde estos subvuelos corresponden a cada escala (tabla Stop). Volviendo al ejemplo anterior, si el vuelo empezaba en Costa Rica y terminaba en USA, si existe una escala que sale de Costa Rica y llega a Panamá, yo como usuario puedo comprar esta escala de forma individual, ignorando así el resto del trayecto.

Un usuario se almacena en la tabla Userw, sin embargo, si el usuario es admin o estudiante, ambas categorías tienen sus propias tablas que están relacionadas a la de usuario. Fue necesario crear un método en el API y una subclase en C# (UserDto) que permite recibir la información de un usuario junto a la de un estudiante e incluso la de un admin toda a la vez, para luego decidir si además de crear un nuevo usuario, también tendrá que crear un admin o estudiante relacionados.

Descripción de la arquitectura

El almacenamiento de los datos se da con el uso de PostgreSQL, posteriormente se desarrolla un API usando las tecnologías de ASP.NET y el Entity Framework en C# que permiten el acceso y la modificación de los datos almacenados en PostgreSQL mediante consultas escritas utilizando el lenguaje SQL. A su vez, el API permite definir rutas web que tanto la aplicación móvil con las vistas web pueden utilizar para acceder a la información necesaria. Las vistas web hacen uso de Angular, mientras que la aplicación móvil hace uso de Flutter y Dart. Además, entre la aplicación móvil y el API existe una capa adicional correspondiente al SQLite, la cual almacena la información proveniente del API de manera local, para su acceso sin necesidad de conexión activa a la red.

Problemas Conocidos

Al crear el script de población y luego de crear algunos vuelos y escalas, cuando llega el momento de agregar pasajeros, estos tienen un id que fue definido inicialmente con un serial, esto usualmente no sería un problema, pero debido a que los pasajeros son creados constantemente, es complicado el referenciarlos ya que una vez creado un pasajero, este aumenta el contador asociado al id serial, por lo que al crear un asiento y hacer la referencia a un pasajero en específico, no es extraño el encontrarse con ids poco intuitivas. Algo como que el primer pasajero de un nuevo vuelo contenga un id "546", aunque es el primer pasajero de este vuelo, no es el primer pasajero en la base de datos.

```
355   -- Insertar pasajeros
356   INSERT INTO PASSENGER (Uemail, Checked_in, Fno)
357   VALUES
358       ('klein@gmail.com', false, 1),
359       ('victor@gmail.com', false, 1);
360
361   -- Insert asientos
362
363   INSERT INTO SEAT (Snumber, Sclass, Pno)
364   VALUES
365       ('A1', 'Ejecutivo', 65),
366       ('C3', 'Turista', 66);
367
```

Los números 65 y 66 en la columna Pno de asientos reflejan los dos pasajeros creados arriba. Note que este vuelo únicamente tiene estos dos pasajeros, pero su "id" va por números elevados debido a los vuelos anteriores.

Durante la creación de tablas y la representación de estas en el API, así como su implementación en las vistas web se dieron errores asociados a las mayúsculas o minúsculas de cada variable o columna definida. PostgreSQL no es Case Sensitive, por lo que no afecta el uso de mayúsculas o minúsculas en el código o los datos descritos en este, sin embargo, lo mismo no aplica para el API o las vistas web. En el caso del API, es fácil reflejar algunos datos utilizando algunas mayúsculas, así como otros completamente en minúsculas, por lo que si no se mantiene el debido orden, es difícil saber la forma correcta de referenciar la información a la hora de su acceso. En las vistas web el problema persiste, ya que a la hora de hacer métodos post y enviar datos al API o recibirlos con un get o como resultado de algún otro método, es fácil que se mezcle el formato en el que se definen cada valor de las columnas.

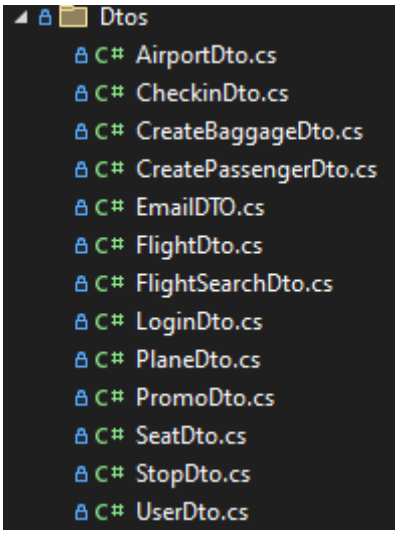
Por otra parte, se identifica un problema a la hora de obtener los datos de la aplicación móvil mediante un query directamente ejecutado en visual studio

code. Esto se debe a que a pesar de realizar alguna consulta a la base de datos, esta no devuelve los datos que fueron agregados directamente de la aplicación móvil. Para solucionar esto se debe obtener el archivo de la base de datos directamente del emulador o aplicación. Seguidamente, descarga el archivo .db y ahora sí podrá observar la información actual de la aplicación móvil. Note que esto debe realizarlo cada vez que quiera observar la nueva información de la aplicación.

Problemas Encontrados

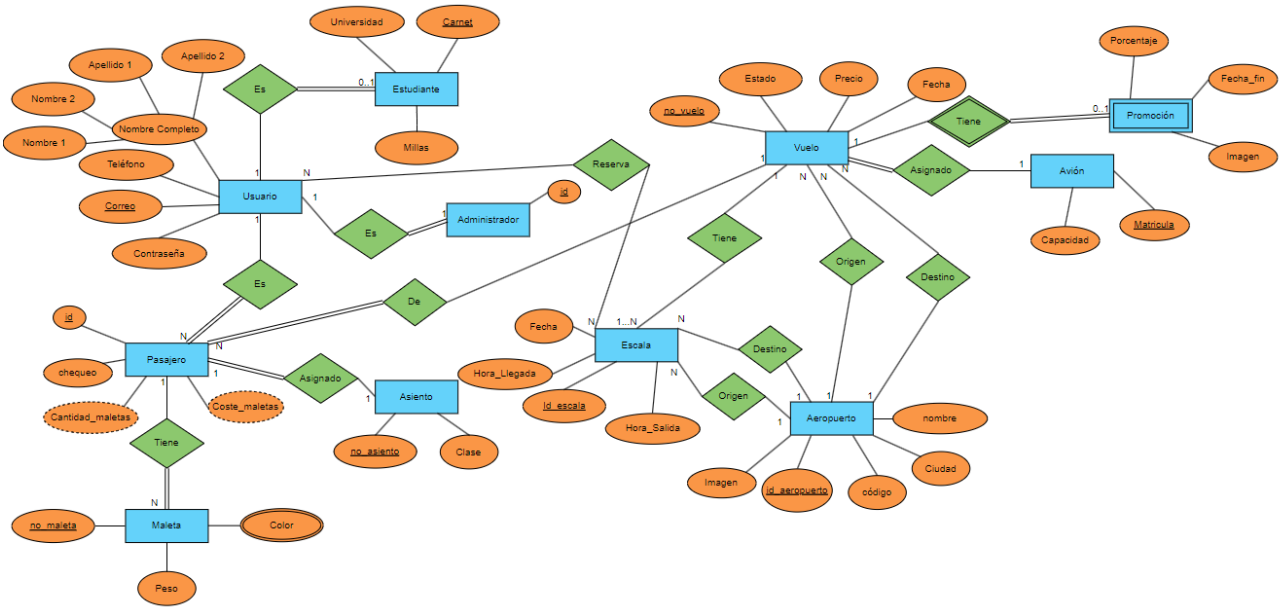
Al intentar enviar los datos a través de un formato JSON en el API, este necesitaba que se enviara información la cual era innecesaria o que causaba que se necesitara enviar JSON muy grandes o con información la cual aún no se conocía.

Para arreglar este problema se encontró que para poder enviar la información que se considerara necesaria se podrían agregar las clases “Dto” (Data Transfer Object) las cuales permiten realizar una conexión limpia entre el API y la base de datos.



Al intentar publicar las aplicaciones en el IIS, se producía un error al intentar utilizar el URL con los endpoints correspondientes al API, para solucionar el problema encontrado se procedió a eliminar lo que se había realizado para publicar y se comenzó de cero el proceso, donde los cambios que se hicieron para un funcionamiento correcto fue el cambio del path físico ofrecido, donde primeramente se había entregado uno en el apartado de documentos y en el intento exitoso se colocó un path directo al disco, así como un cambio en el puerto seleccionado; se considera que el error se debía al path dado en primera instancia, por lo que parece importante que la carpeta a utilizar se encuentre en un path casi en la raiz del equipo y así evitar problemas de permisos.

Diagrama Conceptual



Mapeo Relacional

Entidades Fuertes:

Usuario						
Email	**Contraseña**	**Number**	**Nombre1**	**Nombre2**	**Apellido1**	**Apellido2**

Estudiante		
Carné	**Universidad**	**Millas**

Administrador
IDAdmin

Pasajero	
No_Pasajero	**Chequeado**

Maleta	
No_maleta	**Peso**

Asiento	
No_asiento	**Clase**

Vuelo			
No_vuelo	**Estado**	**Precio**	**Fecha**

Escala			
IDEscala	**Hora_salida**	**Hora_llegada**	**Fecha**

Avión	
Matricula	**Capacidad**

Aeropuerto				
IDAeropuerto	**Código**	**Ciudad**	**Nombre**	**Imagen**

Entidades Débiles:

Promoción		
Imagen	**Porcentaje**	**Fecha_final**

Asociaciones Binarias [1:1]:

- **Usuario - Estudiante**

Se añade como Llave foránea, la llave primaria de usuario en estudiante.

Estudiante			
Carné	**Universidad**	**Millas**	**Uemail**

- **Usuario - Administrador**

Igualmente se añade como llave foránea, la llave primaria de usuario en administrador.

Administrador	
IDAdmin	**Uemail**

- **Pasajero - Asiento**

Se añade la llave primaria de Pasajero en Asiento.

Maleta		
No_maleta	**Peso**	**Pid**

- **Vuelo - Promoción**

Se añade la llave primaria de Vuelo en Promoción.

Promoción			
Imagen	**Porcentaje**	**Fecha_final**	**Vno**

Asociaciones Binarias [1:N]:

- **Usuario - Pasajero**

Se añade la llave primaria de Usuario en Pasajero.

Pasajero		
No_Pasajero	**Chequeado**	**Uemail**

- **Vuelo - Pasajero**

Se añade la llave primaria de Vuelo en Pasajero.

Pasajero			
No_Pasajero	**Chequeado**	**Uemail**	**Vno**

- **Pasajero - Maleta**

Se añade la llave primaria de Pasajero en Maleta.

Maleta		
No_maleta	**Peso**	**Pno**

- **Vuelo - Escala**

Se añade la llave primaria de Vuelo en Escala.

Escala				
IDEscala	**Hora_salida**	**Hora_llegada**	**Fecha**	**Vno**

- **Avión - Vuelo**

Se añade la llave primaria de Avión en Vuelo.

Vuelo				
No_vuelo	**Estado**	**Precio**	**Fecha**	**Pid**

- **Aeropuerto - Vuelo (Origen)**

Se añade la llave primaria de Aeropuerto en Vuelo.

Vuelo	

No_vuelo	Estado	Precio	Fecha	Pid	AOrigen

- **Aeropuerto - Vuelo (Destino)**

Se añade la llave primaria de Aeropuerto en Vuelo.

Vuelo						
No_vuelo	Estado	Precio	Fecha	Pid	VOrigen	VDestino

- **Aeropuerto - Escala (Origen)**

Se añade la llave primaria de Aeropuerto en Escala.

Escala				
IDEscala	Hora_salida	Hora_llegada	Fecha	EOrigen

- **Aeropuerto - Escala (Destino)**

Se añade la llave primaria de Aeropuerto en Escala .

Escala					
IDEscala	Hora_salida	Hora_llegada	Fecha	EOrigen	EDestino

Asociaciones Binarias [N:M]:

- **Usuario - Escala**

Se agregó una tabla de referencia cruzada con las llaves primarias de Usuario y Escala para el caso de esta relación.

Usuario_Escala	
Uemail	**Eid**

Atributos Multivaluados:

- **Colores de maleta**

Para tratar el caso de atributo multivaluado de los colores de una maleta se creó la nueva tabla que asocia la llave primaria de Maleta con el atributo color.

Color_maleta	
Mno	**Color**

Diagrama Relacional

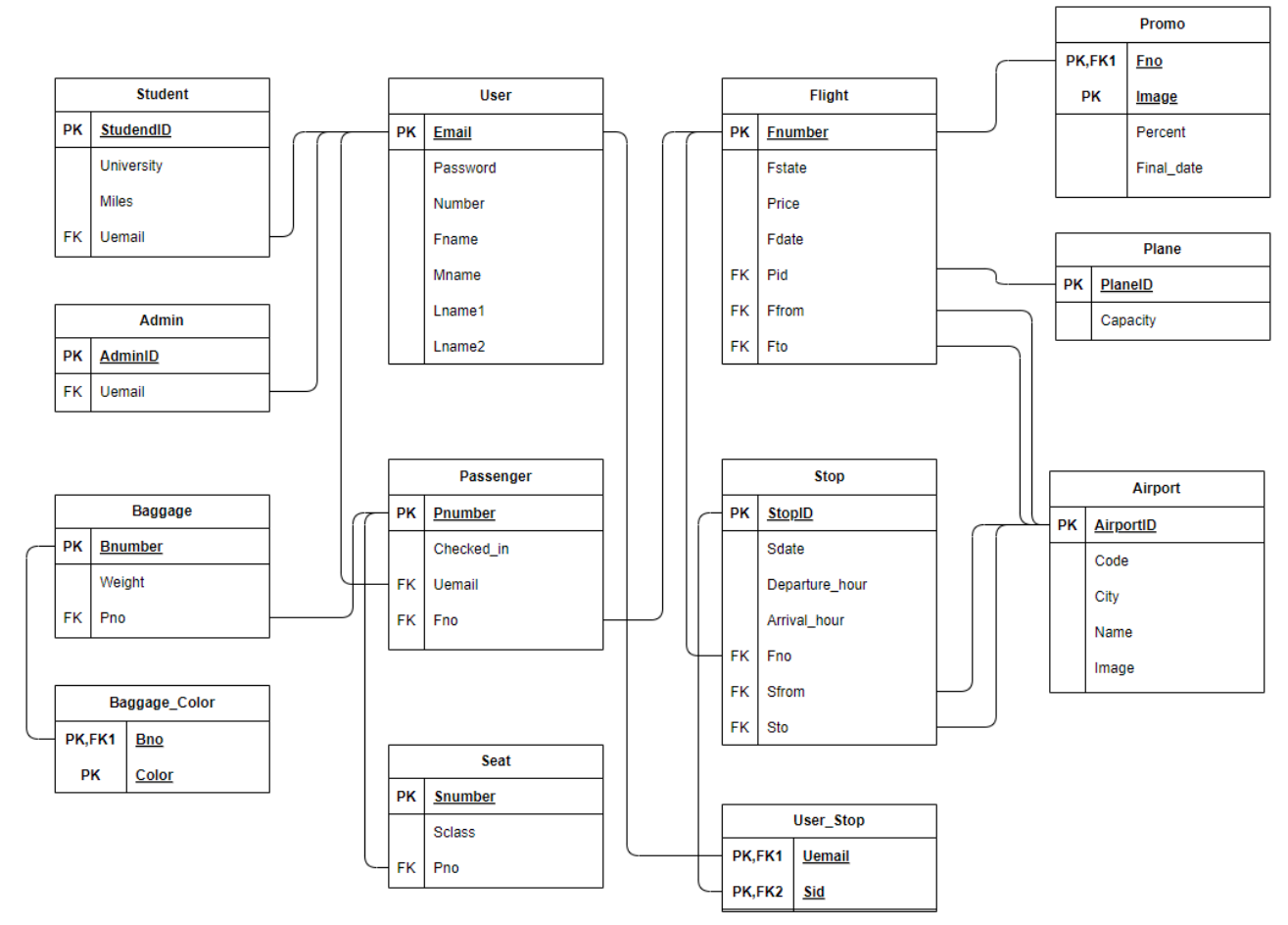
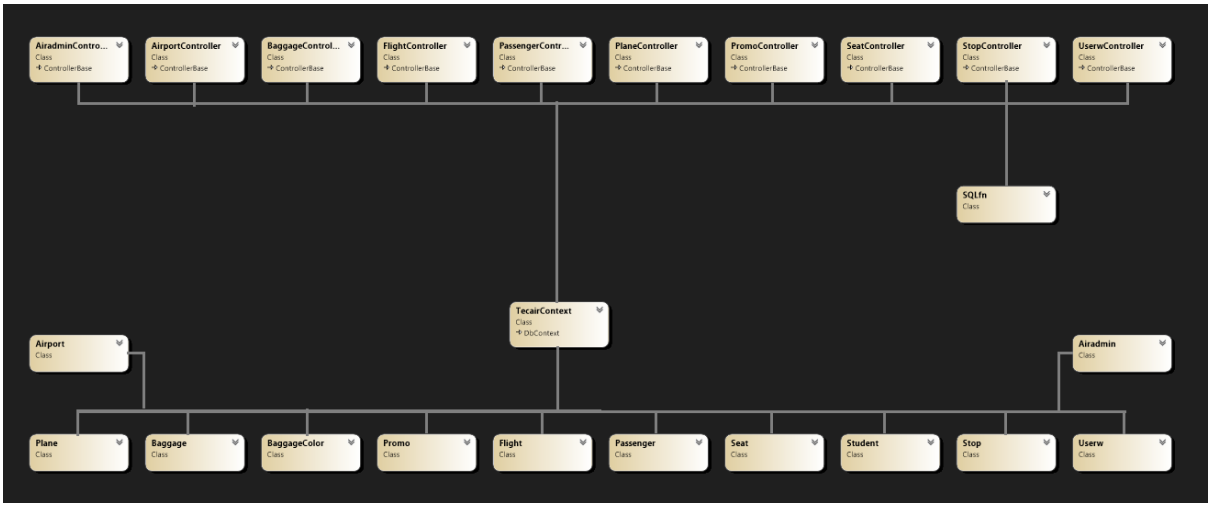


Diagrama de Clases

Para una mayor calidad, puede acceder a los diagramas en el siguiente enlace:

https://github.com/Bases-Crew/TECAir/tree/master/docs

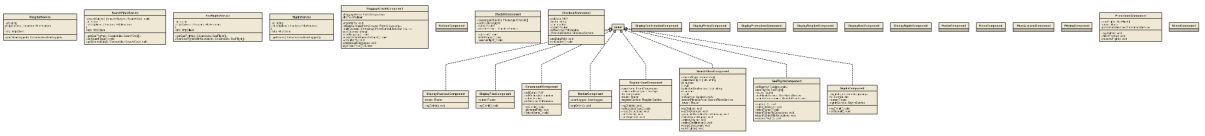


Nota: Este diagrama representa únicamente la conexión de algunas de las clases más importantes dentro del API (C#).

Diagram de clases generado por Angular

● Mitad izquierda:

● Mitad derecha:



Nota: Debido a la estructura del proyecto, no todas las clases se relacionan entre sí de forma directa.

Conclusiones

1. La mayoría de los sistemas de reporte necesitan de NuGet packages disponibles en las bibliotecas de .NET de Visual Studio.
2. Es posible desarrollar aplicaciones Web bastante robustas con Angular ya que posee una gran cantidad de funcionalidades.
3. El manejador de bases de datos de PostgreSQL es un excelente sistema en la que se pueden tener bases de datos extensas, y de una manera sencilla de entender

4. El entorno de desarrollo de Android para desarrollar una aplicación móvil permite crear aplicaciones móviles de manera sencilla y eficiente.
5. Es muy importante el entender el problema que se propone y así obtener un modelo conceptual y relacional de la base de datos, y de esta manera plasmar todas las necesidades antes de montar la base de datos.
6. Utilizar la herramienta de Entity Framework acelera el proceso de ya sea creación de la API o creación de la base de datos, dado que en el caso nuestro ya se contaba con la base de datos, se pudo utilizar el EF para la creación de la base de la API a través de un “Database First”

Recomendaciones

Como se refleja en la sección de problemas encontrados, el no definir y seguir una nomenclatura a la hora de nombrar o definir variables asociadas a cada columna de las tablas en la base de datos puede llevar al caos, por lo que se recomienda definir un orden y compartirlo con todos los desarrolladores para que la conexión entre las diferentes partes de la aplicación se dé con facilidad.

Se recomienda altamente el tomar en cuenta la mayor cantidad de factores posibles a la hora del mapeo de la base de datos, tanto del modelo conceptual como del relacional, para así evitar lo máximo posible la necesidad de hacer cambios a la hora de que la base de datos este creada, pues en caso de necesitar de añadir una nueva relación, o agregar o eliminar una columna de una tabla, puede causar muchos conflictos y retrasar el proyecto.

En el caso en los que ya se cuente con una base de datos montada antes de la creación de la aplicación Web de la API, es posible utilizar el método de “Database First” o “Reverse Engineering” para acelerar el proceso de creación a través de las herramientas con las que ya se cuentan en el proyecto.

Plan de Trabajo

Metas:

● Se espera un conjunto de vistas web desarrolladas en angular que permitan visualizar la información almacenada en la base de datos, así como su respectiva modificación o actualización.
● Permitir el acceso a algunas de las funciones de la aplicación a dispositivos móviles mediante una aplicación móvil.
● Almacenar de manera local los datos de la base de datos en el dispositivo móvil mediante SQLite.
● Sincronizar los datos en el SQLite de la aplicación móvil con los más recientes datos encontrados en la base de datos.
● Permitir el acceso y la modificación de la información de la base datos a través de un API desarrollado en C#, utilizando ASP.NET y el Entity Framework.
● Almacenar los datos en PostgreSQL de manera que mantengan la estructura especificada en el diagrama relacional y conceptual.

Roles:

Front End Manager: Se encarga de llevar un control de las tareas a realizar y posibles adiciones que se pueden necesitar a la hora del uso de la aplicación y la web. Además, coordina con los BackEnd Developer y el Mobile Developer acerca de la conexión.
Front End Designer: Crea mediante código la interfaz e implementa las funcionalidades necesarias para una sencilla utilización de la página.
Back End API Developer: Responsable de crear y mantener la interfaz de programación de la API, permitiendo una buena comunicación entre las aplicaciones.
Back End DataBase Designer: Se encarga de diseñar, implementar y mantener la estructura de la base de datos donde se almacenan y gestionan los datos del sistema.

Mobile Developer: Se encarga de diseñar y crear la interfaz gráfica para mostrar al cliente. Además, implementa toda funcionalidad necesaria con el fin de que el app sea amigable con el usuario.

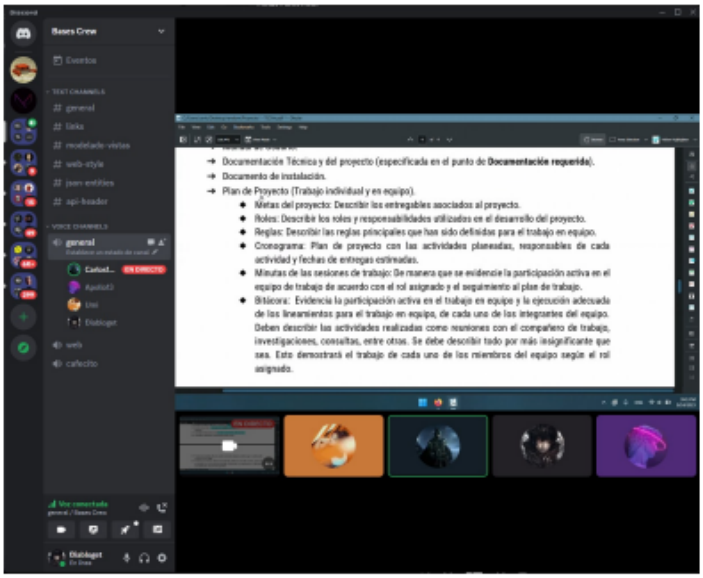
Reglas:

● Las reuniones sincrónicas se harán en su mayoría mediante discord, las cuales se emplearán para definir detalles específicos de las tareas a realizar. En el caso de ser posible o necesario, se harán reuniones presenciales.
● La comunicación restante se dará mediante un grupo de WhatsApp, en su mayoría para presentar actualizaciones en el desarrollo, y notificar de cualquier problema encontrado al resto del equipo.
● El control de versiones del código se hará mediante GitHub, con el fin de que todos los miembros del grupo tengan acceso al código actualizado por otros miembros.
● El código se realizará y comentará en el idioma inglés con el fin de mantener la coherencia del desarrollo.
● Los commits de GitHub se realizan usando la estandarización de Conventional Commits, por lo que también se comenta en inglés.
● Cada característica a realizar puede abrir un branch respectivo, el cual se debe de eliminar una vez la característica hace merge con el código fuente.
● Nunca comentar en el master en el GitHub.
● Comentar al estilo javadocs o utilizando XML.
● Nombrar funciones y variables lo más significativo y corto posible. (Al estilo clean code).

Minutas:

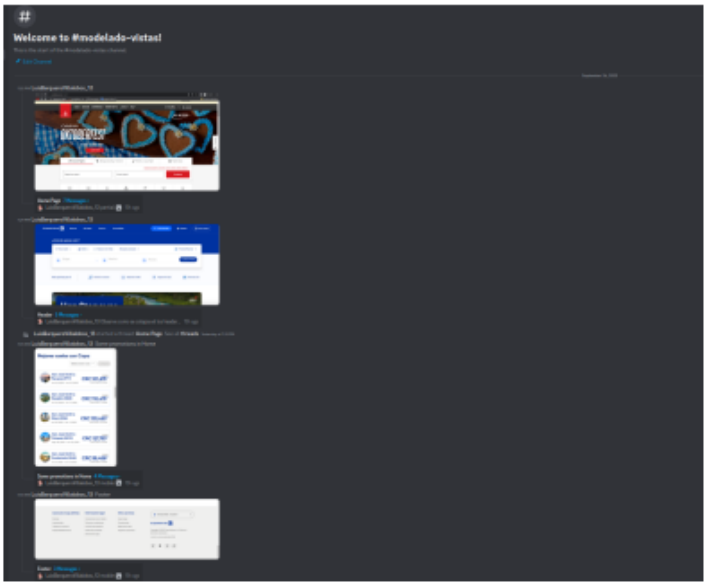
Minutas:

23-9: (Todos)



Se acordó el uso de GitHub projects para la planificación además de asignación de tareas y roles.

26-9: (Felipe - Carlos)



Se acordó el uso del canal de discord con threads para buscar el diseño de la página web.

29-9: (Todos)

Se llevó a cabo una reunión presencial en la que se debatió sobre el modelo conceptual propuesto. Se identificó un inconveniente en cómo se habían planteado las escalas, que originalmente presentaban una relación de vuelos con vuelos. Por ello, se decidió establecer una entidad denominada "escalas",

además de "tiquetes" y "factura", para abordar de manera adecuada el problema relacionado con las escalas.

Bitácora:

Carlos:
● 23-9: Reunión para planificación de plan de trabajo
● 26-9: Diseño de UI HomePage.
● 1-10: Diseño de UI toma de vuelos.
● 13-10: Diseño de Vista Reservaciones.
● 19-10: Implementación Vista Reservaciones.
● 20-10: Implementación Vista administrador**.**
● 21-10: Investigación sobre publicación del API en IIS.
● 22-10: Publicación del API en IIS.

Felipe:
● 23-9: Reunión para planificación de plan de trabajo.
● 26-9: Diseño de UI HomePage.
● 1-10: Diseño de UI toma de vuelos.
● 7-10: Diseño de Vista Reservaciones.
● 15-10: Diseño de Vista Administrador.
● 18-10: Implementación Vista administrador**.**

Jose:
● 23-9: Reunión para planificación de plan de trabajo.
● 25-9: Diseño de Mapa Conceptual .
● 3-10: Investigación e instalación de PostgreSQL.
● 7-10: Creación de la Base de datos.
● 16-10: Arreglos en la Base de datos.
● 21-10: Creación de Funciones de la API.
● 21-10: Investigación sobre publicación del API en IIS.

David
● 23-9: Reunión para planificación de plan de trabajo.
● 25-9: Investigación de PostgreSQL .

● 12-10: Creación de algunas vistas en la aplicación móvil.
● 17-10: Creación de Funciones de la API.
● 21-10: Población de la base de datos.

Ignacio
● 23-9: Reunión para planificación de plan de trabajo.
● 25-9: Investigación de SQLite.
● 2-10: Configuración de SQLite y desarrollo de prueba.
● 7-10: Creación de la aplicación móvil.
● 20-10: Conexión de la Base con la Aplicación móvil.

Commits Github
Felipe Vargas:
- feat: update code
- feat update
- feat:update
- update code
- update
- feat: register-user
- feat:view baggage final
- update code
- fix error
- update code
- update
- Merge branch 'web_reservations-advance' of https://github.com/Bases-Crew/TECAir into web_reservations-advance
- update code
- feat:update
- feat:update
- feat:update price add to baggage
- feat: update
- Merge branch 'web_reservations-advance' of https://github.com/Bases-Crew/TECAir into web_reservations-advance
- feat:create the baggage module
- feat: pdf generator
- fixed code
- feat:footer
- add:buttons
- apdate:fix the bugs
- feat:advance
- create:the checkout page for client
- update

- feat: menu client created(the dropdown for client is having a problem)
- Merge branch 'web_reservations_initit' of https://github.com/Bases-Crew/TECAir into web_reservations_initit
- create:plane-airplane
- feat: create the component navbar
- Fix: Sign-in
- upload: pricing component
- update view
- feat: slidebar and rtl module for view client
- fix: boostrap and sign in
- feat: carousel component
- feat: home component added
- build: "setup project"
- docs: specification
- docs: add readme
- feat: initalize project

Carlos12001:
- Merge branch 'sqlite-mobile' into web_reservations_models_and_pages
- docs: api documentacion interna
- docs: internal coding
- Merge branch 'WebApi' into web_reservations_models_and_pages
- feat: baggage generator
- feat: services place airplane
- feat: component almost ready
- feat: checkin
- chore:felipe
- Merge branch 'WebApi' into web_reservations_models_and_pages
- feat: check in init
- fix: search correct the flies
- Merge branch 'web_reservations_models_and_pages' into WebApi
- fix: search flies services
- fix: checkout services
- fix: sevice register
- Merge branch 'web_reservations_models_and_pages' into WebApi
- feat: checkout
- Merge branch 'WebApi' into web_reservations_models_and_pages
- feat; services checkout
- feat; services checkout
- feat: add examples
- feat: compile
- Merge branch 'web_reservations_models_and_pages' into web_reservations-advance
- feat: always show somethings
- feat: promos

- feat: login api
- feat: login api
- feat: login api
- feat:information seugre to papges
- feat: app conectting
- Merge branch 'web_reservations-advance' into web_reservations_models_and_pages
- Merge branch 'WebApi' into web_reservations_models_and_pages
- Merge branch 'web_reservations_models_and_pages' into web_reservations-advance
- feat: fdkaslfjsa
- feat: add usser api
- Merge branch 'WebApi' into web_reservations_models_and_pages
- fix: name
- Merge branch 'web_reservations_models_and_pages' into web_reservations-advance
- chore: cleaning things
- feat: finish comunication beetween components
- Merge branch 'web_reservations_models_and_pages' into web_reservations-advance
- feat: almost comunication search with see
- fix: comppenet correct
- feat: almost done search
- feat: search-stop model
- feat: add routing
- fix: see flights
- feat: add seats
- feat: add models
- fix
- Merge branch 'web_reservations-advance' of https://github.com/Bases-Crew/TECAir into web_reservations-advance
- chore
- fix: delete main etiqueted
- fix: position buttons
- Merge branch 'web_reservations-advance' into web_reservations_models_and_pages
- Merge branch 'web_reservations_routing' into merge_routing_and_advanaced
- chore: cleaning bav bar
- chore: cleaning the house
- feat: add routing sing in
- feat: SIGIN IN WORKING WITH ANGULAR FORMS
- feat: sign in workikng almost
- feat: add routing
- build: change

Ignacio Calderón:
- sync
- docu interna
- fix promo

- reservation complete
- reservacion basica
- user verification and flights
- login completo
- add sign in
- input login form
- responsive login
- get db
- sqlite
- change type of seat
- add execute sqlite
- Merge remote-tracking branch 'origin/UI-profile-mobile' into merge-login-mobile
- add seat type
- proceso hasta reservar
- fix search bar
- reservation page
- reservation ui complete
- format code
- basic view flight search
- promotions listView
- objetos

jcur02:
- feat: added endpoints for editor
- Merge branch 'WebApi' of https://github.com/Bases-Crew/TECAir into WebApi
- feat: create baggages endpoint
- Merge branch 'WebApi' of https://github.com/Bases-Crew/TECAir into WebApi
- feat: more endpoints
- feat: create passengers
- feat: added some needed gets for the frontend
- fix: User now can be passenger of multiple flights
- feat: API controllers

David Robles:
- feat: added method to consult flights with their discount percentage attached
- feat: Post method to receive every flight assigned to passenger with f.fstate = false
- Merge branch 'WebApi' of https://github.com/Bases-Crew/TECAir into WebApi
- feat: add new user (user/student/admin)
- fix: enabled Cors attributes to allow any origin in the post request bodies
- fix: changed max values and other details
- feat: SQL Function mixed with a get in the API
- feat: scheduled flights screen added
- feat: added login and profile screens

Bibliografía (Enlaces)

Installing Entity Framework Core - EF Core | Microsoft Learn
PostgreSQL: Documentation: 16: 1.3. Creating a Database
sqflite | Flutter Package