PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

Nicolás Camacho Plazas

Mateo Florido Sanchez

DOCUMENTO DE DISEÑO

SISTEMA DE AGENCIA DE VIAJES

ESTRUCTURAS DE DATOS

2018

Descripción General del Diseño

Las agencias de viaje generan gran cantidad de datos que necesitan ser manejados por un sistema. Estas, generan ventas y cotizaciones de tiquetes y paquetes aéreos a destinos nacionales. Del mismo modo, pueden modificar todas sus ventas; es decir, realizar cambios y cancelaciones de los vuelos actuales. Además, estas necesitan mantener organizada y segura la información para poder acceder a asientos disponibles en aerolíneas, sus ingresos y el rendimiento de la agencia.

Herramientas y Servicios Usados

g++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0

GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git

git version 2.17.1

Travis CI Ubuntu 14.04 | 16.04 LTS

Diseño – TAD

- TAD Venta

La clase Venta modela una adquisición de un tiquete de avión.

Atributos:

- M_Agency: Cadena de Caracteres. Representa el identificador de una agencia.
- M_ID: Cadena de Caracteres Representa el Identificador único de la venta.
- M_Flight: Cadena de Caracteres. Representa el vuelo correspondiente al tiquete.
- M_CustomerID: Cadena de Caracteres. Representa la identificación del comprador.
- M_Customer: Cadena de Caracteres. Representa el nombre

del cliente.

- M_FlightDate: Cadena de Caracteres. Representa la fecha del tiquete aéreo adquirido.
- M_BuyDate: Cadena de Caracteres. Representa la fecha de compra del tiquete aéreo.
- M_BuyHour: Cadena de Caracteres. Representa la hora de compra del tiquete aéreo.

TAD Ruta

La clase ruta representa las vías aéreas definidas por la aeronáutica civil.

Atributos:

- M_Code: Cadena de Caracteres. Representa el identificador único de la ruta.
- M_Weekday: Cadena de Caracteres. Representa el día de la semana en que opera el vuelo.
- M_Origin: Cadena de Caracteres. Representa la ciudad de origen de la ruta.
- M_Destination: Cadena de Caracteres. Representa la ciudad de destino de la ruta.
- M_Hour: Cadena de Caracteres. Representa la hora en que opera el vuelo.
- M_FlightDuration: Entero. Representa la duración del vuelo.
- M_Capacity: Entero. Representa la capacidad máxima del avión que opera la ruta.
- M_Price: Entero: Representa el precio de cada tiquete para la ruta.

TAD Archivo

La clase Archivo representa únicamente la lectura y persistencia de datos.

Atributos:

Ninguno

Interface:

ReadFlights

Carga la lista de vuelos programados por la Aeronáutica Civil.

Parámetros:

Aeronáutica: Representa a la aeronáutica civil principal que contiene las rutas programadas y agencias de venta de tiquetes aéreos.

Filename: Representa el nombre del archivo que contiene la información de los vuelos programados por la aeronáutica civil.

Retorna:

No retorna nada.

Precondición:

Filename representa un archivo válido.

Postcondición:

Vuelos programados son cargados al sistema.

ReadAgencies

Carga la lista de Agencias registradas.

Parámetros:

Aeronáutica: Representa a la aeronáutica civil principal que contiene las rutas programadas y agencias de venta de tiquetes aéreos.

Filename: Representa el nombre del archivo que contiene la información de las agencias registradas en el sistema.

Retorno:

No realiza retorno.

Precondición:

Filename es un nombre de archivo válido

Postcondición:

Se cargan en el sistema las agencias registradas.

ReadSales

Carga las ventas realizadas por todas las agencias registradas.

Parámetros:

Aeronáutica: Representa a la aeronáutica civil principal que contiene las rutas programadas y agencias de venta de tiquetes aéreos.

Filename: Representa el nombre del archivo que contiene la información de las ventas realizadas para las agencias.

Retorna:

No retorna nada.

Precondición:

Filename representa un archivo válido.

Postcondición:

Se cargan en el sistema las ventas realizadas.

TAD Agencia

La clase Agencia representa un establecimiento que comercializa tiquetes aéreos entre dos ciudades.

Atributos:

- M_AgencyID: Cadena de Caracteres. Representa el Identificador único de la agencia.
- M_Password: Cadena de Caracteres. Representa la contraseña asociada a cada una de las agencias.
- M_Sales: Colección de TAD Venta. Representa una colección de todas las ventas realizadas por la agencia.

Interface:

NewSale

Realiza una venta en una agencia específica de un tiquete de vuelo.

Parámetros:

M_Agency: Representa la agencia que expide la venta del tiquete aéreo.

M_ID: Representa el identificador único de la venta.

M_Flight: Es el identificador de vuelo del tiquete aéreo.

M_CustomerID: Es el número de identificación del comprador.

M_Customer: Contiene los nombres y apellidos completos del comprador.

M_FlightDate: Es la fecha del vuelo.

M_BuyDate: Representa la fecha de compra del tiquete aéreo.

M_BuyHour: Representa la hora de compra del tiquete aéreo.

Retorna:

No realiza retorno.

Precondición:

M_Agency es una agencia válida.

M_ID: Representa el identificador único de la venta.

M_Flight válido.

M_CustomerID no es vacío

M_Customer no es vacío.

M_FlightDate es válida para el vuelo dado.

M_BuyDate >= Fecha actual.

M_BuyHour<= Hora actual.

Postcondición:

Ninguna.

SalesReport

Realiza un reporte de todas las ventas de una agencia.

Parámetros:

Ninguno.

Retorno:

Cadena de Caracteres con toda la información de las ventas realizadas por la agencia.

Precondición:

Ninguna.

Postcondición:

Ninguna.

TAD Aeronautica

La clase Aeronáutica representa a la autoridad civil que reglamenta las leyes aeronáuticas y controla el tráfico aéreo.

Atributos:

M_Agencies: Colección de TAD Agencia. Representa todas las agencias registradas.

M_Routes: Colección de TAD Ruta. Representa todas las rutas aprobadas por la aeronáutica civil.

Interface:

NewAgency

Crea una nueva agencia.

Parámetros:

Name: Representa el nombre de la agencia.

Password: Representa la contraseña de una agencia.

Retorno:

No tiene retorno.

Precondición:

Ninguna.

Postcondición:

Se ingresa una nueva agencia en la Colección de Agencias M_Agencies.

NewRoute

Crea una nueva ruta reglamentada.

Parámetros:

M_Code: Representa el código de la nueva ruta.

M_Weekday: Representa el día de la semana en el que opera el vuelo.

M_Origin: Representa la ciudad de origen del vuelo.

M_Destination: Representa la ciudad de destino del vuelo.

M_Hour: Es la hora en la que el vuelo comienza a operar.

M_FlightDuration: Es la duración total del vuelo entre las dos ciudades.

M_Capacity: Representa la capacidad total del avión que opera la ruta.

M_Price: Es el precio de cada tiquete de la ruta.

Retorno:

No tiene retorno.

Precondición:

Ninguna.

Postcondición:

Se ingresa una nueva ruta en la Colección de Rutas M_Routes.

Sell

Comprueba que un vuelo pueda ser vendido

Parámetros:

IdVuelo: Representa el identificador del vuelo que se va a vender.

fecha: Representa la fecha en la que el vuelo va a operar.

currentAgency: Representa la agencia que vende el tiquete aereo.

customerID: Representa el número de identificación del comprador.

buyDate: Es la fecha de compra del tiquete aéreo.

buyHour: Es la hora de compra del tiquete aéreo.

Retorno:

Booleano.

Precondición:

IdVuelo es un vuelo válido.

fecha >= fecha Actual

currentAgency es una agencia válida.

customerID no vacío.

Postcondición:

Se llama al método NewSale para confimar la venta del tiquete aéreo.

NewSale

Realiza una venta de un tiquete aéreo en una agencia.

Parámetros:

M_Agency: Representa la agencia en la que se vende el vuelo.

M_ID: Representa el identificador único de venta.

M_Flight: Representa el identificador del vuelo.

M_CustomerID: Representa la identificación del comprador.

M_Customer: Representa los nombres completos del comprador.

M_FlightDate: Representa una fecha válida donde opera el vuelo.

M_BuyDate: Representa la fecha de compra del tiquete.

M_BuyHour: Representa la hora de compra del tiquete.

Retorno:

No realiza retorno.

Precondición:

IdVuelo es un vuelo válido.

fecha >= fecha Actual

currentAgency es una agencia válida.

customerID no vacío.

Postcondición:

Se realiza la venta del vuelo requerido y se almacena en la Colección de Ventas de la agencia.

ContarVentas

Verifica el número de sillas vendidas de un vuelo.

Parámetros:

IdVuelo: Representa el ID de un vuelo a verificar.

Retorno:

Retorna el número de sillas.

Precondición:

IdVuelo es un vuelo válido.

Postcondición:

Ninguno.

VerificarFechas

Verifica el día de una fecha dada.

Parámetros:

Fecha: Fecha en formato añomesdía.

Día: Día entre lunes y domingo a comparar.

Retorno:

Booleano si la fecha coincide con el día.

Precondición:

Fecha tiene un formato válido

Día es un día de la semana válido.

Postcondición:

Ninguna.

SalesReport

Concatena el reporte de ventas de cada agencia.

Parámetros:

Ninguno.

Retorno:

Cadena de Caracteres con el reporte de las ventas de todas las agencias.

Precondición:

Ninguno.

Postcondición:

Ninguno.

ReportFlights

Realiza un reporte de los vuelos que operan en un día determinado o en general los vuelos disponibles.

Parámetros:

origen: Lugar de origen para ver los vuelos que salen de esta ciudad.

Fecha: fecha en la que se quiere conocer que vuelos operan.

Retorno:

Retorna una colección de Rutas.

Precondición:

Origen es una ciudad de origen válida.

Fecha está en un formato de fecha válido.

Postcondición:

Ninguna.

ReportInventory

Realiza un reporte de los vuelos vendidos, cambiados o cancelados de una agencia.

Parámetros:

AgencyID: Representa el ID de la agencia actual.

Retorno:

Retorna una cadena de caracteres con el reporte de todos los vuelos vendidos, cambiados o cancelados.

Precondición:

AgencyID es un identificador de vuelo válido.

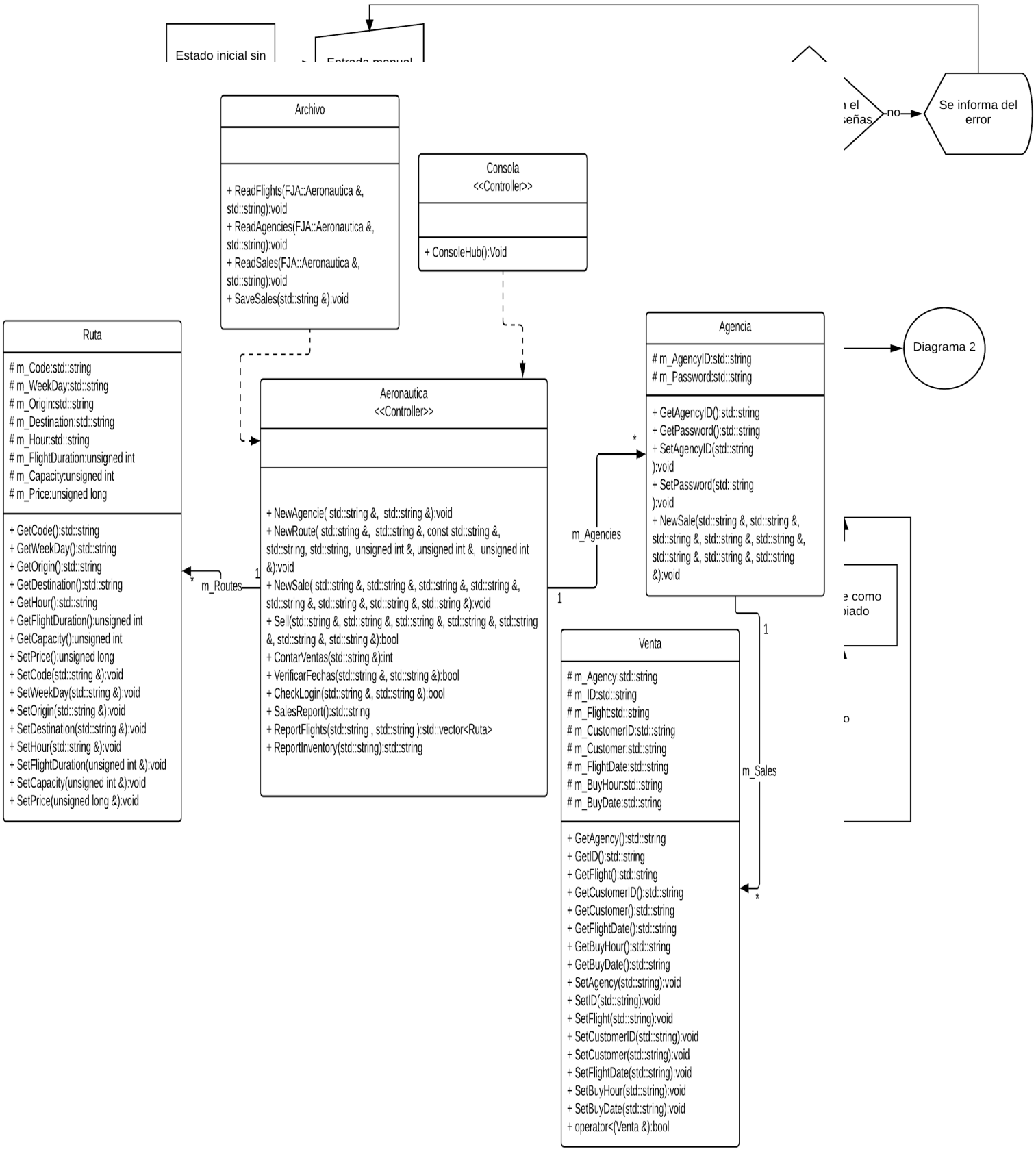
Postcondición:

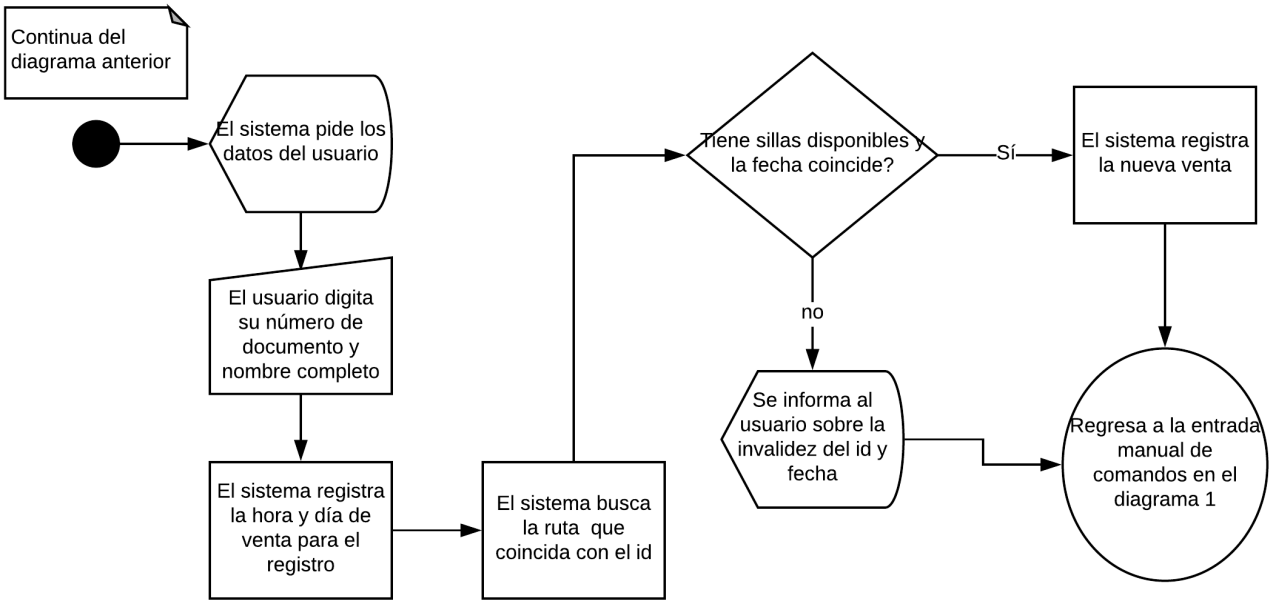
Ninguna.

Diagramas y Esquemáticos

Diagrama de Clases

Diagramas de Flujo





Acta de corrección

- **Entrega 1:** El problema es una diferencia en la forma de hacer la captura del comando login. Se supone que se debía escribir login seguido del id de la agencia (login <id_afencia>), pero se solicitaba el id después de haber escrito login. La corrección se realizó tomando el comando junto con el id, y eliminando la captura segmentada.

SEGUNDA ENTREGA

Diseño - TAD

A continuación estarán listados las nuevas clases del proyecto para la segunda entrega.

-TAD Trie

La clase Trie modela un árbol para la recuperación de la información necesaria para realizar el auto-completado con la tecla TAB.

Atributos:

- m_Root : Tiene la raíz del arbol Trie

Interface:

Insert:

Codifica una palabra nueva en el Trie.

Parámetros:

- v: Cadena de caracteres que representa la palabra u oración que quiero codificar.

Retorno:

No retorna nada.

Precondición:

Cadena de caracteres no vacía, y un árbol con raíz.

Postcondición:

Un árbol Trie con la palabra u oración codificada.

Search:

Indica si existe o una palabra codificada en el árbol Trie.

Parámetros:

- v: Cadena de caracteres que representa la palabra u oración que quiero codificar.

Retorno:

Un booleano que representa si se encontró o no la palabra.

Precondición:

Cadena de caracteres no vacía, y un árbol con raíz.

Postcondición:

Un booleano que indica la existencia de la palabra u oración indicada.

Coincidence:

Indica cual es la primera coincidencia o palabra, dado un prefijo (parte de la palabra).

Parámetros:

- query: Una cadena de caracteres que representa el prefijo de la palabra u oración a buscar.
- R: Una cadena de caracteres que representa la respuesta.

Retorno:

Una cadena de caracteres con la primera palabra que coincide con el prefijo indicado.

Precondición:

Cadena de caracteres no vacía, un árbol con raíz y con una palabra codificada que coincida con el prefijo especificado.

Postcondición:

Una cadena de caracteres con la primera palabra que coincidió con el prefijo especificado.

-TAD Console

El controlador Console representa la interfaz con el usuario donde se ejecutan todos los comandos especificados por el mismo.

Esta utiliza la clase con el mismo nombre en el paquete PUJ creado por Leonardo Florez-Valencia.

Atributos:

- m_Trie: Un árbol que representa el Trie con los comandos codificados.
- m_Aero: Una clase Aeronautica que representa el controlador

de negocio del sistema.

- m_Agencie: Una cadena de caracteres que representa el ID de la agencia con sesión iniciada.

Interface:

Greet:

Imprime en la consola los datos de inicio para el usuario.

Retorno:

No tiene retorno.

Precondición:

Una Clase Console con instancia.

Postcondición:

No tiene pos-condiciones.

Trigger:

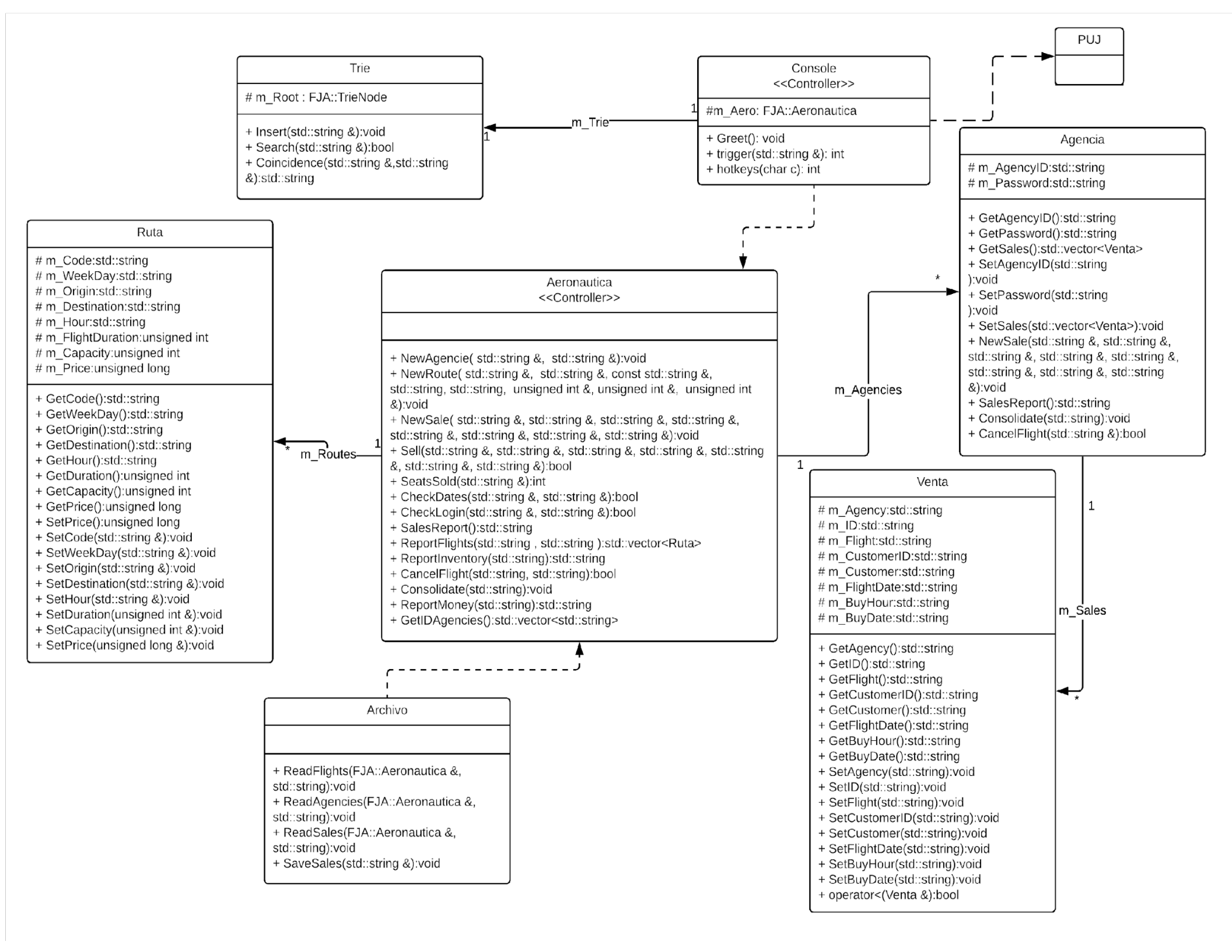
Acciona los comandos digitados por el usuario.

Hotkeys:

Detecta el comando digitado por el usuario.

Diagramas y Esquemáticos

Diagrama de Clases


PUJ
Trie
m_Root : FJA::TrieNode
+ Insert(std::string &):void
+ Search(std::string &):bool
+ Coincidence(std::string &,std::string &):std::string
m_Trie
1
1
Console
<<Controller>>
#m_Aero: FJA::Aeronautica
+ Greet(): void
+ trigger(std::string &): int
+ hotkeys(char c): int
Agencia
m_AgencyID:std::string
m_Password:std::string
+ GetAgencyID():std::string
+ GetPassword():std::string
+ GetSales():std::vector<Venta>
+ SetAgencyID(std::string):void
+ SetPassword(std::string):void
+ SetSales(std::vector<Venta>):void
+ NewSale(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):void
+ SalesReport():std::string
+ Consolidate(std::string):void
+ CancelFlight(std::string &):bool
Ruta
m_Code:std::string
m_WeekDay:std::string
m_Origin:std::string
m_Destination:std::string
m_Hour:std::string
m_FlightDuration:unsigned int
m_Capacity:unsigned int
m_Price:unsigned long
+ GetCode():std::string
+ GetWeekDay():std::string
+ GetOrigin():std::string
+ GetDestination():std::string
+ GetHour():std::string
+ GetDuration():unsigned int
+ GetCapacity():unsigned int
+ GetPrice():unsigned long
+ SetPrice():unsigned long
+ SetCode(std::string &):void
+ SetWeekDay(std::string &):void
+ SetOrigin(std::string &):void
+ SetDestination(std::string &):void
+ SetHour(std::string &):void
+ SetDuration(unsigned int &):void
+ SetCapacity(unsigned int &):void
+ SetPrice(unsigned long &):void
Aeronautica
<<Controller>>
+ NewAgencie(std::string &, std::string &):void
+ NewRoute(std::string &, std::string &, const std::string &, std::string, std::string, unsigned int &, unsigned int &, unsigned int &):void
+ NewSale(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):void
+ Sell(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):bool
+ SeatsSold(std::string &):int
+ CheckDates(std::string &, std::string &):bool
+ CheckLogin(std::string &, std::string &):bool
+ SalesReport():std::string
+ ReportFlights(std::string , std::string):std::vector<Ruta>
+ ReportInventory(std::string):std::string
+ CancelFlight(std::string, std::string):bool
+ Consolidate(std::string):void
+ ReportMoney(std::string):std::string
+ GetIDAgencies():std::vector<std::string>
m_Routes
*
1
m_Agencies
*
1
Venta
m_Agency:std::string
m_ID:std::string
m_Flight:std::string
m_CustomerID:std::string
m_Customer:std::string
m_FlightDate:std::string
m_BuyHour:std::string
m_BuyDate:std::string
+ GetAgency():std::string
+ GetID():std::string
+ GetFlight():std::string
+ GetCustomerID():std::string
+ GetCustomer():std::string
+ GetFlightDate():std::string
+ GetBuyHour():std::string
+ GetBuyDate():std::string
+ SetAgency(std::string):void
+ SetID(std::string):void
+ SetFlight(std::string):void
+ SetCustomerID(std::string):void
+ SetCustomer(std::string):void
+ SetFlightDate(std::string):void
+ SetBuyHour(std::string):void
+ SetBuyDate(std::string):void
+ operator<(Venta &):bool
1
m_Sales
*
Archivo
+ ReadFlights(FJA::Aeronautica &, std::string):void
+ ReadAgencies(FJA::Aeronautica &, std::string):void
+ ReadSales(FJA::Aeronautica &, std::string):void
+ SaveSales(std::string &):void

Acta Correcciones Segunda Entrega

En la segunda entrega se encontraron dos problemas:

- En primer lugar, en el diagrama de clases, no se estaba representado adecuadamente el patrón de diseño MVC (Model View Controller). Ya se realizaron los respectivos ajustes que se pueden evidenciar en el diagrama de la tercera entrega.
- Por último, la consola no realizaba de forma completa la función de autocompletar, específicamente, esta no mostraba todas las posibles opciones, se limitaba a autocompletar por la coincidencia más cercana. Ahora al presionar dos veces la tecla TAB imprime todas las posibles coincidencias.

TERCERA ENTREGA

Diseño - TAD

A continuación estarán listados las nuevas clases del proyecto para la tercera entrega.

-TAD Graph

La clase Graph modela un grafo de rutas aéreas con costos basados en tiempo.

Atributos:

- m_Vertices : Arreglo con las ciudades incluidas en el grafo.
- m_Matrix : Matriz de adyacencia que representa las conexiones de los vértices (ciudades) del grafo con el tiempo de viajar entre esas conexiones.

Interface:

AddNode:

Agrega un nuevo nodo en el grafo.

Parámetros:

- v: Cadena de caracteres que representa el nombre del nodo a agregar.

Retorno:

No retorna nada.

Precondición:

Cadena de caracteres no vacía.

Postcondición:

Nuevo elemento en el arreglo de vértices.

AddArc:

Agrega una conexión entre dos nodos del grafo.

Parámetros:

- a: Índice del nodo de origen.
- b: Índice del nodo de destino.

Retorno:

Sin retorno.

Precondición:

Ambos índices son válidos.

Postcondición:

Ninguno.

GetIndex:

Recupera el índice de un nodo en el arreglo de vértices.

Parámetros:

- a: Representa una cadena de caracteres con el nombre de un nodo del grafo.

Retorno:

Un entero con el índice que tiene asignado la cadena en el arreglo de vértices.

Precondición:

Cadena de caracteres no vacía.

Postcondición:

Ninguna.

Dijkstra:

Realiza el algoritmo de Dijkstra para el grafo.

Parámetros:

- seed: Representa el índice semilla en el vector de vértices.

Retorno:

Arreglo con el árbol de recubrimiento mínimo con inicio en la semilla.

Precondición:

Existe el grafo y la semilla.

Postcondición:

Árbol de recubrimiento mínimo en la semilla.

FloydWarshall:

Realiza el algoritmo de Floyd-Warshall en el grafo.

Parámetros:

- origen: Cadena de caracteres con la ciudad de origen.
- destino: Cadena de caracteres con la ciudad de destino.

Retorno:

Arreglo con el camino óptimo desde origen al destino.

Precondición:

Cadena de caracteres origen y destino no vacías.

Postcondición:

Ninguna.

Prim:

Realiza el algoritmo de Prim en el grafo.

Parámetros:

- seed: Representa el índice semilla en el vector de vértices.

Retorno:

Arreglo con el árbol de recubrimiento mínimo con inicio en la semilla.

Precondición:

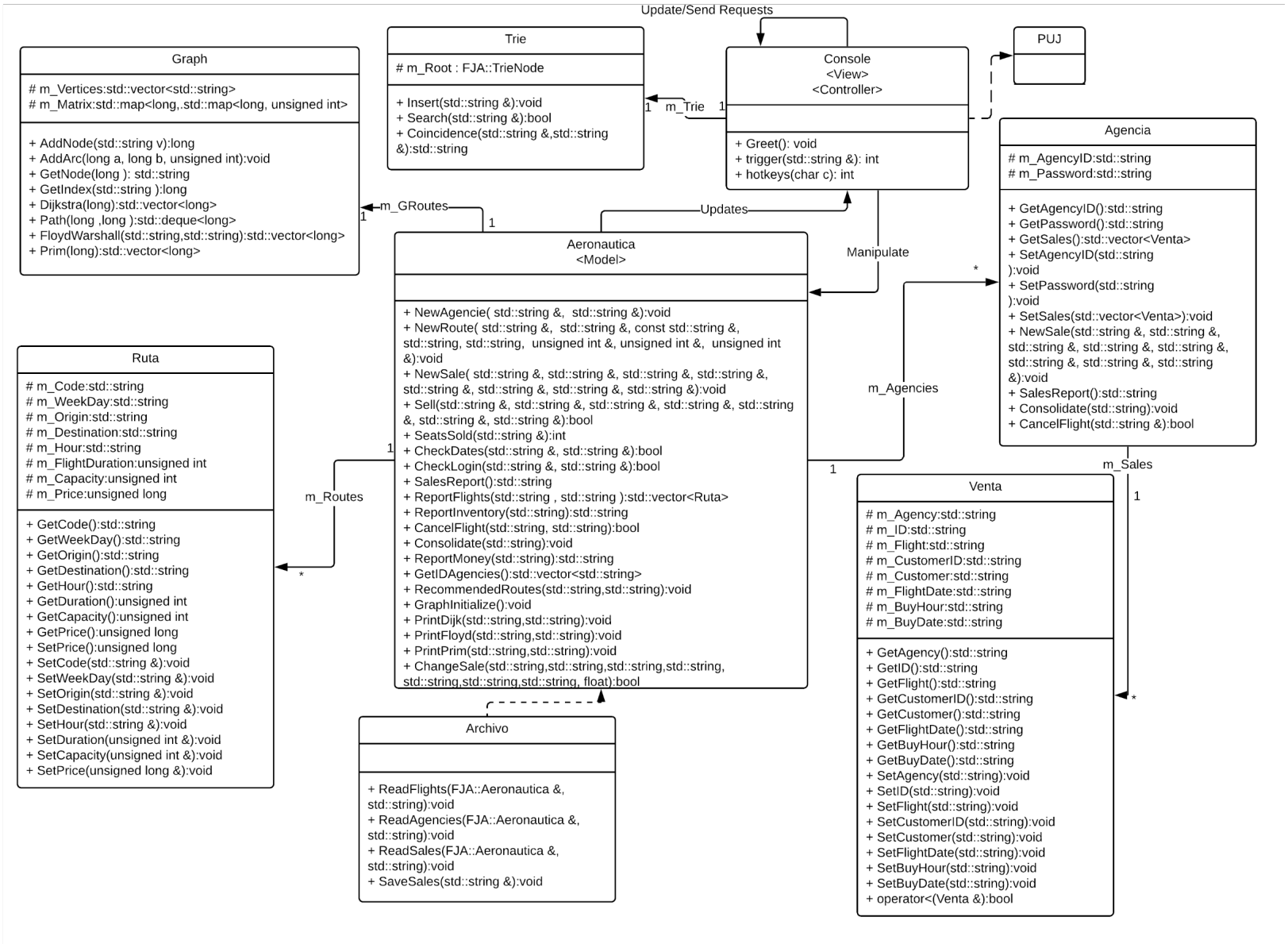
Existe el grafo y la semilla.

Postcondición:

Árbol de recubrimiento mínimo en la semilla.

Diagramas y Esquemáticos

Diagrama de Clases


Update/Send Requests
Graph
m_Vertices:std::vector<std::string>
m_Matrix:std::map<long,.std::map<long, unsigned int>
+ AddNode(std::string v):long
+ AddArc(long a, long b, unsigned int):void
+ GetNode(long): std::string
+ GetIndex(std::string):long
+ Dijkstra(long):std::vector<long>
+ Path(long ,long):std::deque<long>
+ FloydWarshall(std::string,std::string):std::vector<long>
+ Prim(long):std::vector<long>
Trie
m_Root : FJA::TrieNode
+ Insert(std::string &):void
+ Search(std::string &):bool
+ Coincidence(std::string &,std::string &):std::string
1
m_Trie
1
Console
<View>
<Controller>
+ Greet(): void
+ trigger(std::string &): int
+ hotkeys(char c): int
PUJ
Agencia
m_AgencyID:std::string
m_Password:std::string
+ GetAgencyID():std::string
+ GetPassword():std::string
+ GetSales():std::vector<Venta>
+ SetAgencyID(std::string
):void
+ SetPassword(std::string
):void
+ SetSales(std::vector<Venta>):void
+ NewSale(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):void
+ SalesReport():std::string
+ Consolidate(std::string):void
+ CancelFlight(std::string &):bool
m_GRoutes
1
1
Updates
Manipulate
*
Aeronautica
<Model>
+ NewAgencie(std::string &, std::string &):void
+ NewRoute(std::string &, std::string &, const std::string &, std::string, std::string, unsigned int &, unsigned int &, unsigned int &):void
+ NewSale(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):void
+ Sell(std::string &, std::string &, std::string &, std::string &, std::string &, std::string &, std::string &):bool
+ SeatsSold(std::string &):int
+ CheckDates(std::string &, std::string &):bool
+ CheckLogin(std::string &, std::string &):bool
+ SalesReport():std::string
+ ReportFlights(std::string , std::string):std::vector<Ruta>
+ ReportInventory(std::string):std::string
+ CancelFlight(std::string, std::string):bool
+ Consolidate(std::string):void
+ ReportMoney(std::string):std::string
+ GetIDAgencies():std::vector<std::string>
+ RecommendedRoutes(std::string,std::string):void
+ GraphInitialize():void
+ PrintDijk(std::string,std::string):void
+ PrintFloyd(std::string,std::string):void
+ PrintPrim(std::string,std::string):void
+ ChangeSale(std::string,std::string,std::string,std::string, std::string,std::string,std::string, float):bool
m_Agencies
1
m_Sales
1
Ruta
m_Code:std::string
m_WeekDay:std::string
m_Origin:std::string
m_Destination:std::string
m_Hour:std::string
m_FlightDuration:unsigned int
m_Capacity:unsigned int
m_Price:unsigned long
+ GetCode():std::string
+ GetWeekDay():std::string
+ GetOrigin():std::string
+ GetDestination():std::string
+ GetHour():std::string
+ GetDuration():unsigned int
+ GetCapacity():unsigned int
+ GetPrice():unsigned long
+ SetPrice():unsigned long
+ SetCode(std::string &):void
+ SetWeekDay(std::string &):void
+ SetOrigin(std::string &):void
+ SetDestination(std::string &):void
+ SetHour(std::string &):void
+ SetDuration(unsigned int &):void
+ SetCapacity(unsigned int &):void
+ SetPrice(unsigned long &):void
1
m_Routes
*
Venta
m_Agency:std::string
m_ID:std::string
m_Flight:std::string
m_CustomerID:std::string
m_Customer:std::string
m_FlightDate:std::string
m_BuyHour:std::string
m_BuyDate:std::string
+ GetAgency():std::string
+ GetID():std::string
+ GetFlight():std::string
+ GetCustomerID():std::string
+ GetCustomer():std::string
+ GetFlightDate():std::string
+ GetBuyHour():std::string
+ GetBuyDate():std::string
+ SetAgency(std::string):void
+ SetID(std::string):void
+ SetFlight(std::string):void
+ SetCustomerID(std::string):void
+ SetCustomer(std::string):void
+ SetFlightDate(std::string):void
+ SetBuyHour(std::string):void
+ SetBuyDate(std::string):void
+ operator<(Venta &):bool
*
Archivo
+ ReadFlights(FJA::Aeronautica &, std::string):void
+ ReadAgencies(FJA::Aeronautica &, std::string):void
+ ReadSales(FJA::Aeronautica &, std::string):void
+ SaveSales(std::string &):void