Grupo: LOS\_BARONES\_DE\_LA\_CERVEZA (31)

*Integrantes:  
Nicolás Patronelli – Legajo: 152.367-3 (REPRESENTANTE)  
Leonardo Maestri – Legajo: 155.993-0  
Martín Acerboni – Legajo: 159.057-1*

TP Gestión de datos

FRBA-CRUCERO

Contenido

[Consideraciones generales 3](#_Toc12815531)

[Sobre la solución propuesta 3](#_Toc12815532)

[Archivo de configuración 3](#_Toc12815533)

[Manejo de fechas 3](#_Toc12815534)

[Seguridad 3](#_Toc12815535)

[Convención de nombres elegida 4](#_Toc12815536)

[Login 4](#_Toc12815537)

[Validaciones en el proceso de Login 5](#_Toc12815538)

[Roles 6](#_Toc12815539)

[ABM de Roles 7](#_Toc12815540)

[Crear Rol 7](#_Toc12815541)

[Modificar Rol 7](#_Toc12815542)

[Habilitar/Deshabilitar Rol 8](#_Toc12815543)

[ABM de Puertos 9](#_Toc12815544)

[ABM de Cruceros 10](#_Toc12815545)

[Alta de un nuevo crucero 10](#_Toc12815546)

[Modificación de un crucero existente 12](#_Toc12815547)

[Baja por Servicio Técnico 13](#_Toc12815548)

[Baja Definitiva 14](#_Toc12815549)

[ABM de Recorridos 15](#_Toc12815550)

[Alta de nuevo recorrido 15](#_Toc12815551)

[Editar Recorrido 20](#_Toc12815552)

[Alto de nuevo tramo 21](#_Toc12815553)

[Modificar el precio de un tramo 22](#_Toc12815554)

[Deshabilitar recorrido 23](#_Toc12815555)

[Habilitar recorrido 23](#_Toc12815556)

[Generar Viaje 23](#_Toc12815557)

[Comprar/Reservar Viaje 24](#_Toc12815558)

[Pagar Reserva 24](#_Toc12815559)

[Listados Estadísticos 25](#_Toc12815560)

[Otra información 26](#_Toc12815561)

[Datos personales de los integrantes 26](#_Toc12815562)

[Corrección recibida en la entrega preliminar del DER 26](#_Toc12815563)

# Consideraciones generales

## Sobre la solución propuesta

Las decisiones de diseño de la aplicación (tanto de usuario en C# como de Base de Datos en SQL Server) que tomamos son justificadas según nuestro criterio, que claramente es particular y seguramente sesgado por nuestra experiencia y conocimiento. Por lo tanto, es evidente que deben existir soluciones mejores a las que nosotros fuimos proponiendo e implementado en la realización del TP. En otras palabras, nuestra solución es sólo una de las posibles.

## Archivo de configuración

El archivo de configuración se encuentra dentro de la carpeta *src* y se llama archivoConfig.txt

Su contenido inicial es:

fecha config=2018/06/01  
Data Source=localhost\SQLSERVER2012  
Initial Catalog=GD1C2019  
User Id=gdCruceros2019  
Password=gd2019

El campo ‘fecha config’ contiene la fecha que la aplicación utiliza como fecha actual.

## Manejo de fechas

Tal como pide el enunciado, levantamos mediante un archivo de configuración la fecha del sistema. Dicha fecha (denominada *fechaConfig*) se emplea como fecha actual en todas las funcionalidades de la aplicación.

## Seguridad

Si bien puede exceder un poco los límites del TP, durante su desarrollo no utilizamos consultas SQL armadas con string donde los parámetros se concatenan. Hacer eso puede conllevar a sufrir ataques de tipo [SQL Injection](https://es.wikipedia.org/wiki/Inyecci%C3%B3n_SQL). En cambio, utilizamos la clase *SQLParameter* que provee C# para cargar parámetros en las consultas. En particular, diseñamos una clase wrapper llamada **Parametro[[1]](#footnote-1)**, que puede encontrarse en *FbraCrucero.Utils.Parametro,* para facilitar la creación de parámetros.

**Ejemplo de lo que NO hacemos:**

string miConsulta = "SELECT \* FROM Tabla WHERE nombre = " + nombre " AND edad IS BETWEEN " + edad1 + " AND " + edad2;

**Ejemplo de lo que SI hacemos:**

Parametro paramNombre = new Parametro("@nombre", SqlDbType.NVarChar, nombre, 255);

Parametro paramEdadA = new Parametro("@edad\_menor", SqlDbType.Int, edadMenor);

Parametro paramEdadB = new Parametro("@edad\_mayor", SqlDbType.Int, edadMayor);

Luego, la consulta anterior nos queda:

string miConsulta = "SELECT \* FROM Tabla WHERE nombre = @nombre AND edad IS BETWEEN @edad\_menor AND @edad\_mayor”;

Esta solución también favorece la mitigación de errores en la construcción de las consultas (errores que en general serían difíciles de encontrar) al no tener que hacer ningún tipo de concatenación. Además, mejora la legibilidad de la consulta, al estar compuesta por un único string y no ser producto de la concatenación de varios.

## Convención de nombres elegida

La convención o estrategia de nombres (named strategy) elegida fue la siguiente:

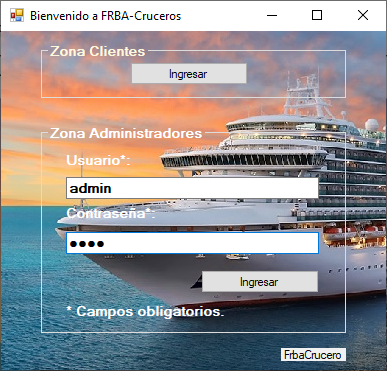
* **[LOS\_BARONES\_DE\_LA\_CERVEZA].[USP\_nombre\_procedimiento\_almacenado]** para nombrar a los procedimientos almacenados que creamos para la aplicación (USP significa *User Stored Procedure*).
* **[LOS\_BARONES\_DE\_LA\_CERVEZA].[UTR\_nombre\_trigger]** para nombrar a los triggers que creamos para la aplicación.
* **[LOS\_BARONES\_DE\_LA\_CERVEZA].[UF\_nombre\_funcion]** para nombrar a las funciones que creamos para la aplicación.
* **FK\_nombre\_foreign\_key** para nombrar a las claves foráneas que vinculan las tablas.
* LowerCamelCase para las variables en C#.
* UpperCamelCase para las clases en C#.

# Login

La primera pantalla de la aplicación es la de **Login**. La misma se distribuye en dos secciones:

* Zona Clientes
* Zona Administradores

En la siguiente imagen podemos apreciarla:



Dado que los clientes no realizan login para ingresar a la aplicación, su acceso se lleva adelante simplemente pulsando el botón *Ingresar* de Zona Clientes.

En el caso de los administradores, estos sí deben realizar el proceso de login, ingresando su usuario y contraseña. Dado que por el alcance del TP no hay proceso de registro de usuarios, nosotros incluimos de forma predetermina el siguiente set de usuarios administradores (como solicitaba el enunciado):

|  |  |
| --- | --- |
| **Usuario** | **Contraseña** |
| **admin** | **w23e** |
| nico | w23e |
| leo | w23e |
| martin | w23e |

Al intentar loguearse con cualquier otro nombre de usuario que no sea uno de los detallados, el sistema informará que el nombre de usuario no existe.

## Validaciones en el proceso de Login

Para validar el proceso de login de un usuario construimos un procedimiento almacenado llamado *USP\_Login*, cuya implementación puede encontrarse en el archivo *script\_creacion\_inicial.sql.*

El mismo consiste en:

1. Validar si el nombre de usuario ingresado existe. Si no existe el login falla.
2. Si el nombre de usuario existe entonces validar que la contraseña ingresada sea la correcta. Si no lo es, el usuario existe pero la contraseña es incorrecta y el login falla. En esta situación se incrementa en una unidad el campo *cantidad\_intentos\_fallidos* del usuario en la tabla *Usuarios*.
3. Si el nombre de usuario existe y la contraseña es correcta, entonces validar que el usuario se encuentre habilitado. Si no lo está, el login falla.
4. Si el nombre de usuario existe, la contraseña es correcta y el usuario está habilitado entonces el proceso de Login es exitoso.

En todos los casos donde se produce una falla en el login se abre un cuadro de diálogo informando el motivo.

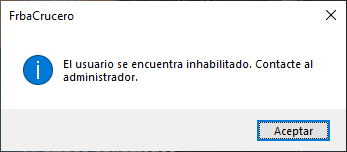
El procedimiento almacenado cargará un valor en un parámetro de salida llamado @resultado, siendo:

|  |  |
| --- | --- |
| **@resultado** | **Evento** |
| 1 | El usuario ingresado no existe. |
| 2 | Contraseña incorrecta. |
| 3 | Usuario inhabilitado. |
| 4 | Ingreso correcto. |

Luego, desde la aplicación procederemos simplemente a capturar ese valor y mostrar el mensaje de error oportuno o brindarle acceso al usuario en caso de un login correcto.

El sistema lleva la cuenta de la cantidad de intentos de ingreso fallidos por cada usuario en el campo *cantidad\_intentos\_fallidos* de la tabla *Usuarios*. Cuando dicha cantidad es 3, el campo *habilitado* de Usuario pasa de 1 a 0 para indicar que el mismo está deshabilitado. Esta operatoria se realiza de forma automática mediante un trigger (sobre el evento UPDATE en la tabla *Usuarios*) denominado *UTR\_inhabilitar\_intentos\_fallidos*.

Si un usuario se encuentra inhabilitado no podrá ingresar al sistema aun cuando ingrese su contraseña correctamente. Recibirá un aviso como el siguiente:



La única forma de que pueda volver a ingresar al sistema es habilitándolo manualmente mediante una query desde SQL Server Management. Si un usuario acumula una cantidad de intentos de ingreso fallidos de 1 o 2 e ingresa satisfactoriamente, entonces el campo *cantidad\_intentos\_fallidos* se reinicia a 0 (operatoria que es parte del procedimiento *USP\_Login* antes mencionado).

## Encriptación de contraseñas

Para cumplir este requerimiento utilizamos la función **HASHBYTES()** que provee SQL Server. En particular, usamos el parámetro ‘SHA2\_256’ que permite obtener el hash de un string empleando el algoritmo SHA256, tal como solicitaba el enunciado.

Ejemplo:

SET @hash\_pass\_ingresada = HASHBYTES('SHA2\_256',@pass\_ingresada)

Usamos esta función dentro del procedimiento almacenado *USP\_Login* ya mencionado, que es el encargo de procesar el login de los usuarios.

# Roles

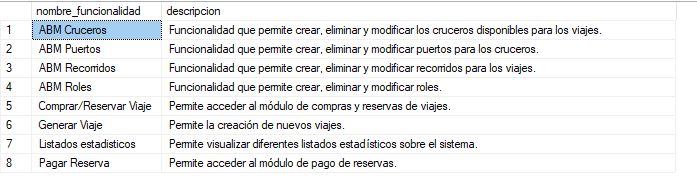
La aplicación inicia con los roles predeterminados *Rol\_Cliente* y *Rol\_Admin*, cuyas funcionalidades se detallan a continuación:

|  |  |  |
| --- | --- | --- |
| **Funcionalidades/Roles** | **Rol\_Cliente** | **Rol\_Admin** |
| ABM Cruceros |  | ✓ |
| ABM Puertos |  | ✓ |
| ABM Recorridos |  | ✓ |
| ABM Roles |  | ✓ |
| Comprar/Reservar Viaje | ✓ | ✓ |
| Generar Viaje |  | ✓ |
| Listados estadísticos |  | ✓ |
| Pagar Reserva | ✓ | ✓ |

El listado de funcionalidades es fijo para la aplicación, no varía. Las mismas se encuentran precargadas en la tabla *Funcionalidades*. Mediante la siguiente query pueden verse el listado de todas ellas junto a su descripción:

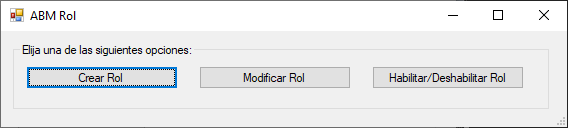
SELECT \*

FROM LOS\_BARONES\_DE\_LA\_CERVEZA.Funcionalidades



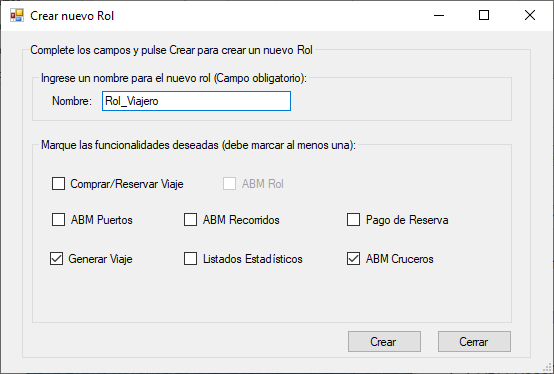
## ABM de Roles

Esta funcionalidad permite crear, modificar o deshabilitar un rol. Debemos seleccionar a cuál de estas subfuncionalidades queremos acceder mediante una pantalla como la siguiente:



### Crear Rol

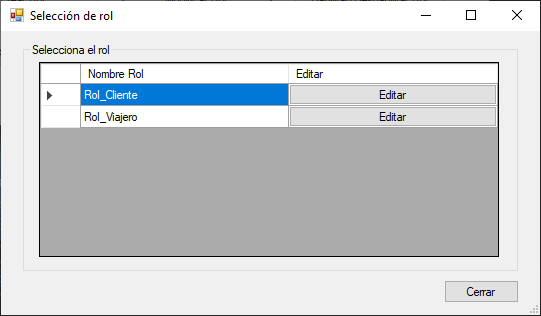
Para crear un nuevo rol debe ingresarse el nombre del rol (el sistema verificará que el mismo no esté en uso, y de estarlo, avisará oportunamente) y marcar por lo menos una funcionalidad. Una vez ingresado el nombre del nuevo rol y haber marcado las funcionalidades, se debe presionar el botón *Crear* para dar de alta el nuevo rol. Si el alta es satisfactoria se avisará mediante un cuadro de diálogo. La creación de un nuevo rol impacta en las tablas *Roles* y *Funcionalidades\_Por\_Roles* de la base de datos.



No permitimos que se creen roles nuevos que incluyan la funcionalidad *ABM Rol*, ya que la consideramos una funcionalidad de alto nivel de privilegio, cuya habilitación para otro rol que no sea el *Administrador* puede conllevar a situaciones peligrosas (¿Qué sucede si un usuario de tipo cliente tiene acceso al *ABM Rol* y decide deshabilitar el Rol *Administrador*? Los usuarios de tipo administrador no podrán ingresar al sistema).

### Modificar Rol

El usuario podrá seleccionar el rol a editar mediante un listado como el siguiente:



Al presionar el botón *Editar* se disparará un formulario similar al de creación de un nuevo rol, con su nombre y funcionalidades ya precargadas.

Decidimos no permitir la edición del rol *Administrador* para conservar la integridad del sistema; por tal motivo, directamente no lo mostramos en el listado de los roles a editar.

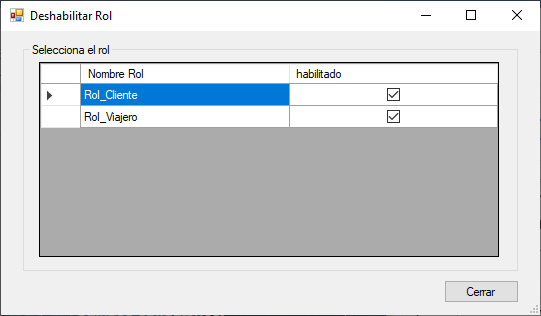
Si puede modificarse el rol *Cliente*, pero no su nombre, ya que se perdería la semántica del rol *Cliente* (fundamental en el sistema). Pueden agregarse y quitarse funcionalidades del rol *Cliente* pero, como ya mencionamos, no puede adicionársele la funcionalidad *ABM Rol*. Por ejemplo, puede agregarse la funcionalidad *Listados Estadísticos* al rol *Cliente* para que este tipo de usuarios pueda consultar dichos listados.

**Importante:** Las restricciones implementadas no son tecnológicas o de implementación (podríamos permitir el agregado de la funcionalidad *ABM Rol* al rol *Cliente* modificando unas pocas líneas de código) sino de negocio y según nuestra propia perspectiva.

Por supuesto, pueden editarse todos los otros roles creados, siempre que el nombre del rol éste disponible y se seleccione al menos una funcionalidad

### Habilitar/Deshabilitar Rol

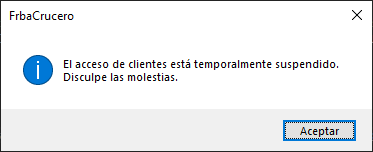
Se puede habilitar y deshabilitar un rol seleccionado y deseleccionando el checkbox asociado al nombre del rol que queremos alterar, como se muestra en la siguiente pantalla:



Cuando un rol se deshabilita se marca a 0 el campo *habilitado* en su registro de la tabla *Roles.* Análogamente, cuando se lo vuelve a habilitar se marca dicho campo a 1. Por defecto, todos los roles empiezan habilitados.

Decidimos que no pueda deshabilitarse el rol *Administrador*, dado que son el único tipo de usuario que tiene acceso al ABM de roles. Por tal motivo, directamente no mostramos el rol *Administrador* en la pantalla de habilitar/deshabilitar roles.

Si se deshabilita el rol *Cliente*, cuando se quiera ingresar desde la pantalla inicial a la *Zona Clientes* se obtendrá el siguiente mensaje:



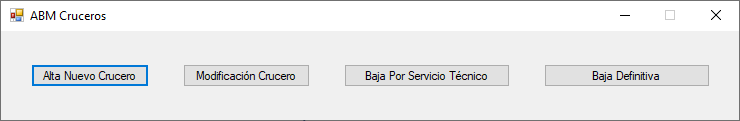
Deberá ingresar un administrador al sistema y habilitar el rol *Cliente* nuevamente desde la funcionalidad *ABM Rol* para que los usuarios clientes puedan ganar acceso a la aplicación nuevamente. En la práctica, este enfoque puede resultar útil para cuando se llevan adelante tareas de mantenimiento en el sistema y no se permite, momentáneamente, el acceso de usuarios clientes al mismo.

# ABM de Puertos

Pese a que el enunciado solicita no implementar dicha funcionalidad, si debía tenerse en cuenta su impacto en el modelo de datos (creamos la tabla *Puertos* y la completamos con los puertos de la migración, además de relacionarla con la tabla *Tramos*). Nosotros lo incluimos como una funcionalidad más (ver [listado de funcionalidades](#_Roles)). Cuando se intenta ingresar a dicho ABM se muestra un cuadro de diálogo mencionando que no está disponible aún. Sin embargo, dado el diseño de nuestra solución, su implementación no impactaría al modelo ni la aplicación en cuanto a cambios en otras funcionalidades: simplemente debería introducirse el código del ABM de Puertos en el form ya reservado para el mismo, y el resto de las funcionalidades deberían seguir funcionando de igual manera (es transparente para ellas).

# ABM de Cruceros

Esta funcionalidad permite dar de alta un nuevo crucero, modificar alguno existente, dar de baja un crucero por servicio técnico o darlo de baja en forma definitiva, tal como se ve en la siguiente imagen:



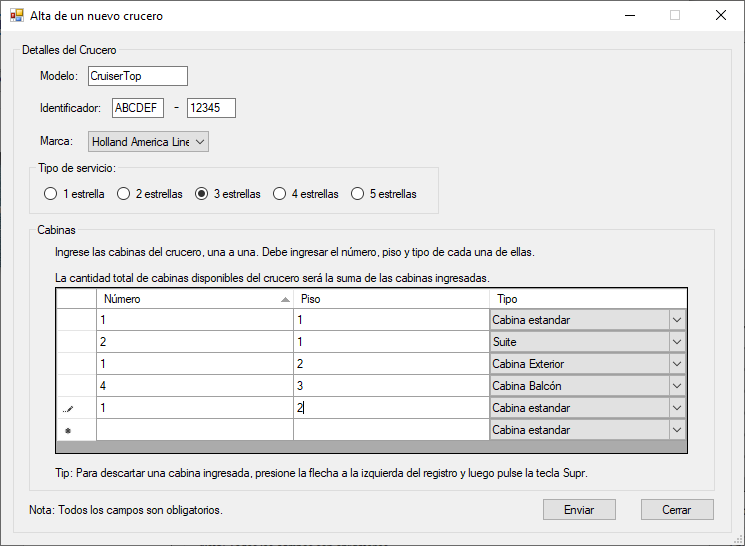
## Alta de un nuevo crucero

Estudiando los registros de la tabla maestra, llegamos a la conclusión que el campo *identificador* de un crucero es de negocio (viene a ser una especie de patente o dominio del crucero, como el que tiene un auto, por ejemplo). Por este motivo, no permitimos que existan dos cruceros con igual identificador (esta restricción ya se cumple en los cruceros migrados). Si se intenta dar de alta o modificar un crucero usando un identificador ya existente la operación se cancela y se informa mediante un cuadro de diálogo.

Vimos también que estos identificadores se componen de hasta 6 letras en mayúsculas, seguidas de un guion, y un número de hasta 5 dígitos al final. Por lo tanto, adoptamos que todos los identificadores nuevos que se ingresen respeten ese formato. Para ello, en el textbox identificador creamos los eventos necesarios para asegurar que el usuario solo pueda ingresar hasta 6 letras mayúsculas como primer parte del identificador, y hasta 5 dígitos como segunda parte de este.

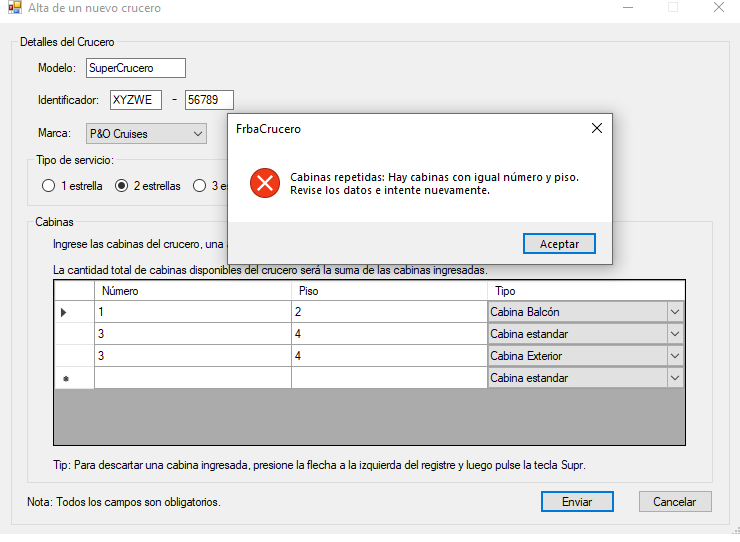
En cuanto a la *marca*, dado que el enunciado solicitaba que dicho campo pudiese ser cambiado “de la forma más fácil posible”, decidimos normalizarla creando una entidad (tabla en la BD) llamada *Marcas*. En el formulario, el usuario puede seleccionar una marca de un combobox con todas las provenientes de la migración. Además, agregamos la opción *Otra* por sí el crucero que quiere dar de alta no es de ninguna de las marcas disponibles.

En cuanto al tipo de servicio, dado que un mismo crucero puede tener cabinas de distintos tipos (Estándar, Suite, Exterior, Balcón o Ejecutivo), no nos pareció correcto que el crucero tenga como tipo de servicio un tipo de cabina. En cambio, pensamos en el tipo de servicio de un crucero como el nivel de sus prestaciones generales, no propio de sus cabinas. Por ejemplo, hay cruceros que cuentan con piscina a bordo, bar, salones de fiestas, juegos acuáticos, etc. independientemente del tipo de sus cabinas. Así que decidimos que el tipo de servicio sea un valor de 1 a 5 estrellas, representando el nivel de las comodidades que presta (a mayor cantidad de estrellas mayor es el nivel del crucero). Para el caso de los cruceros provenientes de la migración, al no contar con información al respecto, decidimos que todos tengan el valor por default de 4 estrellas en su tipo de servicio.



Para agregar las cabinas del crucero utilizamos un datagriview que permite el ingreso de cada una de ellas detallando su número, piso y tipo de cabina. Una validación que hacemos es que no se permita el ingreso de dos cabinas con igual número y piso (aunque tengan distinto tipo), ya que la ubicación dentro del crucero físicamente debe ser única.

Todos los campos mencionados son obligatorios para el alta del crucero (se hacen las validaciones correspondientes y se informa al usuario en caso de faltantes). Además, decidimos que todo crucero nuevo debe registrar al menos una cabina (sino se ingresa al menos una cabina también se informa del error).

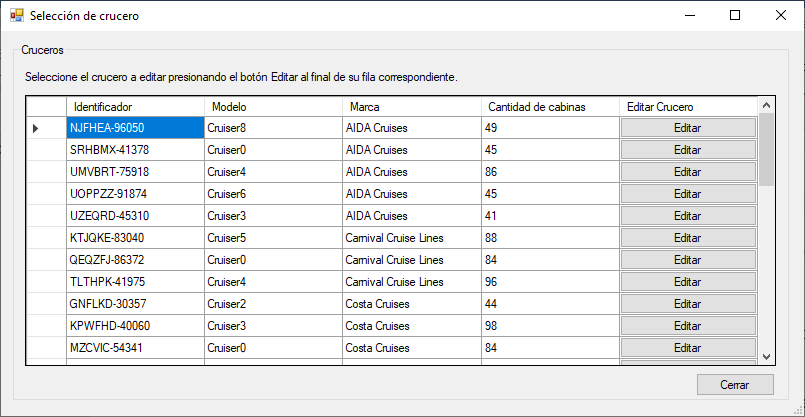


Para esta sección utilizamos los procedimientos almacenados *USP\_insertar\_crucero* y *USP\_insertar\_cabina* desde la aplicación en C#, cuyas implementaciones se pueden encontrar en el archivo de migración.

## Modificación de un crucero existente

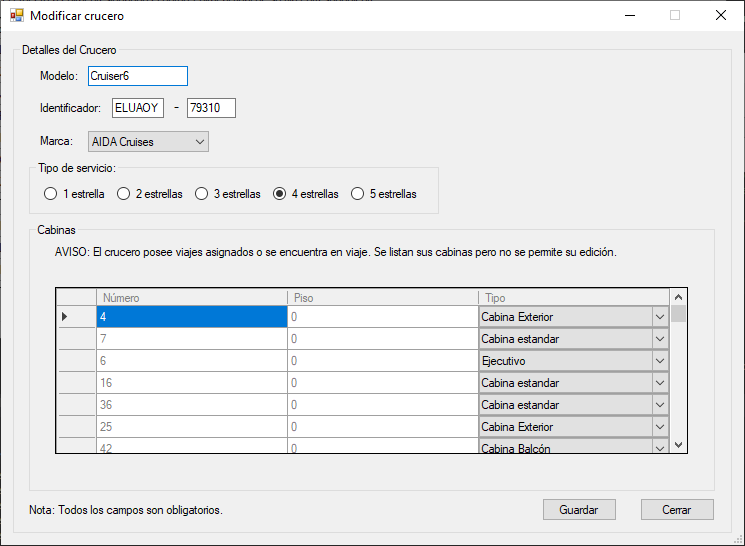
Para modificar un crucero primero mostramos una pantalla con un datagridview que carga los cruceros de la tabla *Cruceros* para seleccionar cual modificar.

El usuario puede visualizar los datos principales de cada crucero e incluso ordenar el listado según cada campo presionando la cabecera de las columnas.



Pulsando el botón *Editar* se pasa al formulario de modificación del crucero seleccionado. Como restricción, no permitimos editar un crucero que se haya dado de baja en forma definitiva.

Por razones de consistencia, tampoco permitimos modificar las cabinas de un crucero que se encuentra en viaje o tiene un viaje planificado con compras y reservas. Se listan sus cabinas, pero no se permite su edición, como se ve en la siguiente imagen.

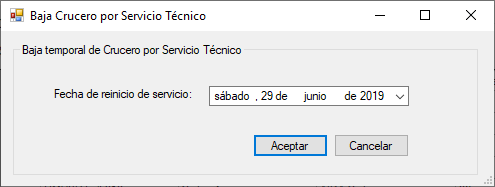


Si se pueden editar el resto de campos del crucero. Dado que todos los cruceros de la migración tienen viajes, para poder editar las cabinas de un crucero (agregar nuevas, modificar las existentes o eliminarlas) se debe dar de alta un nuevo crucero.

A nivel objetos de base de datos, para modificar un crucero creamos el procedimiento almacenado *USP\_actualizar\_crucero*, llamándolo desde la aplicación C#.

## Baja por Servicio Técnico

Para dar de baja un crucero por servicio técnico se listan todos los cruceros activos (no mostramos los que ya están fuera de servicio ni tampoco los que fueron dados de baja en forma definitiva). El usuario selecciona el crucero y debe ingresar, en un nuevo cuadro de diálogo, la fecha en que el crucero volverá a estar en funcionamiento (haciendo uso de un datetimepicker):



Para permitir la baja por servicio técnico validamos primero que la fecha elegida para el reinicio del servicio sea posterior a la actual (la fecha tomada del archivo de configuración). Caso contrario, se muestra un mensaje de error y se aborta la operación.

A nivel base de datos, se establece en 1 el campo *baja\_fuera\_servicio* en el registro correspondiente al crucero seleccionado (en la tabla *Cruceros*) y se inserta un registro en la tabla *Cruceros\_Fuera\_Servicio* (que tiene un campo *id\_crucero* que es FK a la PK de *Cruceros*), cargando en el campo *fecha\_inicio\_fuera\_servicio* la fecha actual (la del archivo de configuración) y en el campo *fecha\_fin\_fuera\_servicio* la fecha que el usuario administrador escogió.

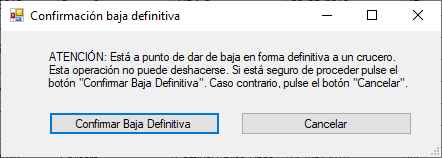
Una vez dado de baja el crucero seleccionado por servicio técnico, recargamos el datagridview con el listado de cruceros excluyéndolo, para que el usuario administrador pueda, en forma inmediata, elegir otro, favoreciendo la experiencia de usuario.

## Baja Definitiva

De forma análoga a dar de baja un crucero por servicio técnico, primero se muestra un listado con aquellos cruceros activos o que estén dados de baja por servicio técnico, para que el administrador seleccione aquel que quiere dar de baja en forma definitiva. Permitimos la baja definitiva de cruceros en reparación porque, por ejemplo, puede darse la situación que un crucero no pueda repararse (ponerse en funcionamiento nuevamente) y entonces se decide eliminarlo de la flota activa de la empresa. Por supuesto, no listamos aquellos cruceros que ya fueron dados de baja en forma definitiva.

**Importante:** Dado que ya existe una baja temporal (la baja por servicio técnico), decidimos que la baja definitiva de un crucero no pueda deshacerse, es decir, si un crucero es dado de baja en forma definitiva no podrá activarse nuevamente.

Por este motivo, primero mostramos un cuadro de diálogo para que el administrador confirme realmente el proceso de baja definitiva, como se ve en la siguiente imagen:

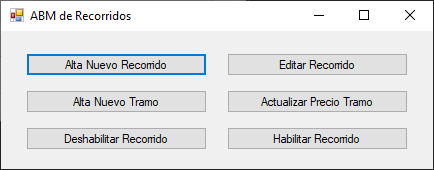


A nivel impacto en el modelo de datos, cuando un crucero es dado de baja en forma definitiva se marca a 1 el campo *baja\_vida\_util* y se actualiza el campo *fecha\_baja\_vida\_util* con la fecha actual (nuevamente, la del archivo de configuración) en el registro perteneciente al crucero seleccionado de la tabla *Cruceros*.

En este caso también recargamos el datagridview con el listado de cruceros en forma dinámica, excluyendo el que se acaba de dar de baja en forma definitiva. De esta manera, el usuario no tiene que volver a entrar a la funcionalidad desde el principio, sino que puede dar de baja otro crucero en forma inmediata y sin inconsistencias (dar de baja en forma definitiva dos veces un mismo crucero).

# ABM de Recorridos

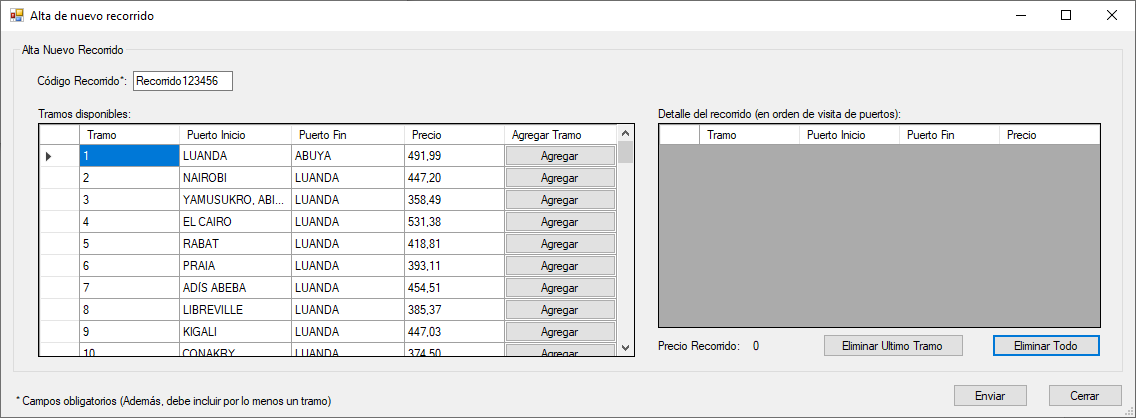
El ABM de Recorridos agrupa una serie de funcionalidades, como se muestra en la siguiente imagen:



El usuario puede dar de alta un nuevo recorrido, editar uno existente, además de deshabilitar y volver a habilitar un recorrido. También incluimos algunas funcionalidades pertenecientes a lo que sería un ABM de tramos, de manera tal que puedan darse de alta nuevos tramos y actualizar el precio de los existentes (ya que el enunciado aclaraba que al crearse un recorrido debía poder fijarse el precio de los tramos). Dado que se brindaba libertad para modelar esta parte de la solución, nuestra decisión fue tener cada funcionalidad modularizada por separado, lo que creemos favorece la experiencia de usuario y también mejora la mantenibilidad del código.

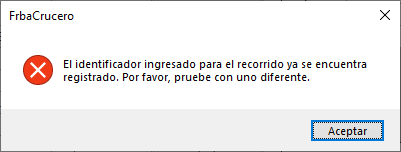
## Alta de nuevo recorrido

Este formulario es el que permite el alta de un nuevo recorrido. Recordemos que un recorrido se compone de n tramos que deben cumplir ciertas condiciones: el puerto de fin del tramo i debe ser el mismo que el puerto de inicio del tramo i+1.



Para dar de alta un nuevo recorrido debemos ingresar un código de recorrido (campo *recorrido\_codigo* en la tabla *Recorrido*). Dado que el enunciado solicitaba que el usuario debía ingresar un código de recorrido alfanumérico, nosotros asumimos que dicho código es de negocio. El mismo debe ser único por recorrido, pero no lo empleados como PK en la tabla *Recorrido*, sino que usamos un campo IDENTITY denominado *id\_recorrido* que es interno (clave subrogada).

La primera validación que hacemos es que el código de recorrido ingresado por el usuario este disponible, es decir, no exista otro recorrido en la base de datos con el código de recorrido ingresado. En caso de que así sea, informamos al usuario de tal situación para que elija uno distinto:



El siguiente paso consiste en el armado del recorrido, es decir, seleccionar los tramos que lo conformarán. Para tal fin, mostramos en un datagridview el listado de todos los tramos disponibles en la tabla *Tramo*, enseñando de cada uno su puerto de inicio, puerto de fin y precio, de tal manera que el usuario tenga toda la información relevante a disposición.

Dentro del armado del recorrido, el primer paso consiste en seleccionar el tramo inicial, pulsando el botón *Agregar* que se encuentra al final de cada registro tramo. Al hacerlo, **el datagridview de tramos se recargará de forma automática mostrándonos solamente los posibles tramos que pueden seguir al inicial**, es decir, aquellos cuyo puerto de origen coincide con el puerto de destino del tramo inicial.

El usuario podrá seleccionar el segundo tramo que desee para la conformación del nuevo recorrido, de vuelta, pulsando el botón *Agregar* al final del mismo, y entonces el datagridview de tramos se volverá a cargar, esta vez, mostrando aquellos tramos que puedan ser continuación del segundo tramo seleccionado (aquellos cuyo puerto de inicio coincide con el puerto de fin del segundo tramo). Este paso se puede repetir de forma indefinida, es decir, es iterativo y no tiene un tope predefinido, tal como solicitaba el enunciado. De esta manera, se puede crear un nuevo recorrido compuesto por uno, dos, tres o n tramos. Si el puerto de origen es A se puede volver a pasar por el mismo las veces que se desee, no hay límites al respecto.

Es importante aclarar que el usuario puede, en cualquier momento, dar de alta el nuevo recorrido: es el quién decide cuando está conforme con el recorrido generado. Para ello, debe pulsar el botón *Enviar*. Una validación que efectuamos es que se debe haber seleccionado al menos un tramo, pues no tiene sentido que exista un recorrido sin ningún tramo.

Para facilitar la experiencia de usuario, a medida que éste va agregando los tramos a su nuevo recorrido, mostramos en un datagridview ubicado a la derecha del formulario el estado actual del mismo, es decir, como se va formando el recorrido. Además, el usuario puede eliminar el último tramo agregado con tan solo pulsar el botón *Eliminar Último Tramo*. Así, se pueden agregar n tramos, eliminar el último tramo, eliminar el ahora último tramo, agregar uno nuevo, etc… El recorrido se irá armando de forma dinámica según la necesidad del usuario.

Incluimos también un botón *Eliminar Todo* que al pulsarlo limpia el recorrido actual eliminando todos los tramos seleccionados.

Para que el usuario pueda saber el precio base del recorrido calculamos de forma dinámica dicho valor, a medida que el usuario agregue o elimine tramos y lo mostramos en una etiqueta. Así, el usuario puede saber si el precio base del recorrido es muy excesivo (porque ha incluido muchos tramos o tramos muy costosos) y entonces eliminar algunos tramos para disminuir el costo base del recorrido.

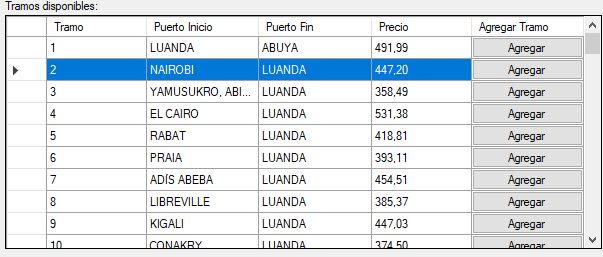
Creemos un nuevo recorrido utilizando los tramos provenientes de la migración solamente. Por ejemplo, creemos el recorrido:

**NAIROBI-LUANDA**

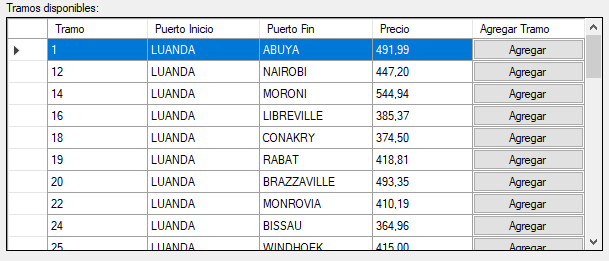
**LUANDA-ABUYA**

**ABUYA-LUANDA**

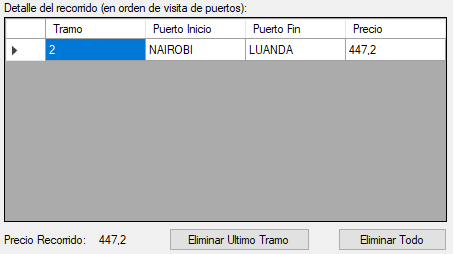
El primer paso, entonces, es seleccionar el tramo **NAIROBI-LUANDA** del datagridview izquierdo (el de tramos):



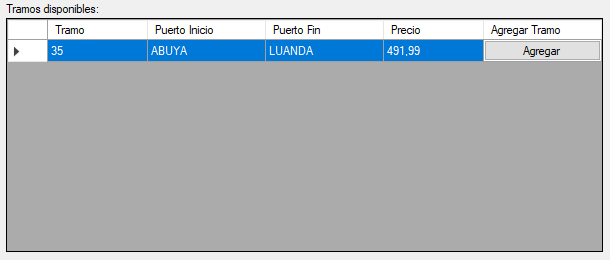
Al hacerlo, el datagridview de tramos se actualizará automáticamente para mostrarnos los posibles tramos que pueden seguir al elegido:



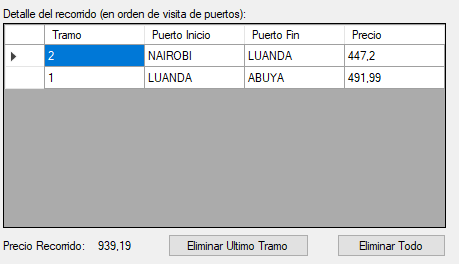
Vemos que se cargan todos los tramos cuyo puerto de inicio es LUANDA. Al mismo tiempo, en el datagridview derecho veremos el estado actual de nuestro recorrido, con el detalle de los tramos que fuimos agregando y el precio base actual de este:



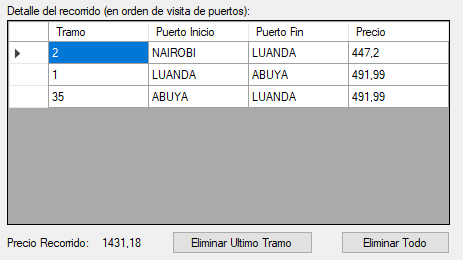
Seleccionemos el segundo tramo de nuestro nuevo recorrido, que era **LUANDA-ABUYA**. Al hacerlo, el datagridview de tramos se recargará, otra vez, con los posibles tramos que pueden seguir. En este caso, según los tramos de la migración, vemos que el único tramo disponible es ABUYA-LUANDA:



Nuevamente, en el datagridview derecho podemos ver como se va confeccionando nuestro tramo en orden (el primer tramo es el tramo inicial de nuestro recorrido, el segundo el que le sigue, etc.). Vemos como el precio se recalculo de forma automática:



Seleccionemos el tramo **ABUYA-LUANDA** para terminar la confección de nuestro nuevo recorrido, que quedará de la siguiente manera:



El precio base final de nuestro recorrido es $1431,18. Ingresamos un código de recorrido de no haberlo hecho antes (en este ejemplo ingresamos “Recorrido123456”) y pulsamos *Enviar*. Un cuadro de diálogo emergente nos confirmará el éxito del alta del nuevo recorrido.

El alta de un nuevo recorrido impacta en las tablas *Recorrido* y *Tramos\_por\_Recorrido* de nuestro modelo de datos. La tabla *Tramos\_por\_Recorrido* es la que permite enlazar los diferentes tramos que conforman un recorrido a través de los campos *tramo\_anterior* y *tramo\_siguiente*, que son FK a la misma tabla (por ende, es una tabla recursiva). Esto permite recuperar los tramos de un recorrido en su orden correcto. Podemos identificar el primer tramo de un recorrido porque el registro *tramo\_anterior* está en NULL. Análogamente, el último tramo de un recorrido tendrá el campo *tramo\_siguiente* en NULL.

Para consultar la conformación de nuestro nuevo recorrido en la BD empleamos la siguiente query:

SELECT

r.recorrido\_codigo 'Identificador Recorrido',

tpr.tramo\_anterior 'Tpr anterior',

tpr.tramo\_siguiente 'Tpr siguiente',

p1.puerto\_nombre 'Puerto Inicio',

p2.puerto\_nombre 'Puerto Fin',

t.tramo\_precio 'Precio tramo'

FROM LOS\_BARONES\_DE\_LA\_CERVEZA.Recorrido r

JOIN LOS\_BARONES\_DE\_LA\_CERVEZA.Tramos\_por\_Recorrido tpr

ON r.id\_recorrido = tpr.id\_recorrido

JOIN LOS\_BARONES\_DE\_LA\_CERVEZA.Tramo t

ON tpr.id\_tramo = t.id\_tramo

JOIN LOS\_BARONES\_DE\_LA\_CERVEZA.Puerto p1

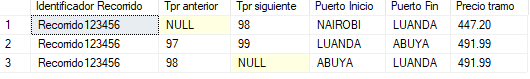
ON t.tramo\_puerto\_inicio = p1.id\_puerto

JOIN LOS\_BARONES\_DE\_LA\_CERVEZA.Puerto p2

ON t.tramo\_puerto\_destino = p2.id\_puerto

WHERE r.recorrido\_codigo = 'Recorrido123456'

En nuestro caso, arrojará el siguiente resultado:



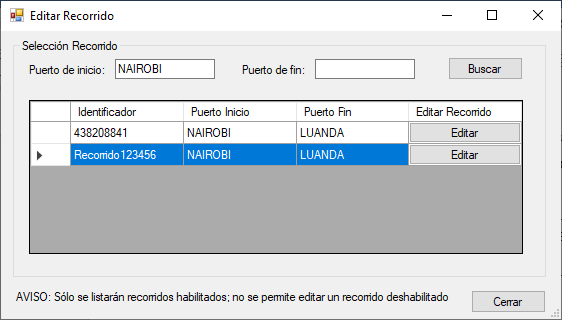
Para dar de alta un nuevo recorrido empleamos los procedimientos almacenados *USP\_insertar\_recorrido* y *USP\_insertar\_tramo\_por\_recorrido*, cuya implementación puede verse en el script de migración, como la de todos los procedimientos almacenados mencionados en este documento.

## Editar Recorrido

Esta funcionalidad permite editar un recorrido existente, lo que implica cambiar su código de recorrido y los tramos que lo conforman. Para elegir el recorrido a editar incluimos primero un formulario de búsqueda que permite buscar recorridos por puerto de inicio y fin. Si dejamos ambos campos vacíos y pulsamos el botón *Buscar*, la búsqueda se hará libre y se mostrarán todos los recorridos disponibles.

**Importante:** Sólo permitimos editar recorridos habilitados. Aquellos recorridos que hayan sido deshabilitados no podrán editarse; deberán volver a habilitarse para que puedan ser editados.

Podemos buscar el recorrido “Recorrido123456” que acabamos de dar de alta:



Al pulsar el botón *Editar* se mostrará un formulario equivalente al de alta, donde el usuario podrá agregar nuevos tramos, eliminar existentes, hasta cambiar por completo el recorrido. También podrá cambiar el código del recorrido por otro que éste disponible.

## Alto de nuevo tramo

Los tramos iniciales con los que contamos son los provenientes de la migración. Sin embargo, podemos dar de alta nuevos tramos.

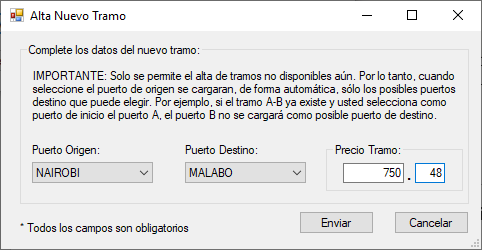
Dado que el ABM de Puertos no es una funcionalidad disponible para esta versión del sistema, no podemos dar de alta nuevas ciudades (puertos). Por lo tanto, todos los tramos que podemos generar quedan supeditados a los puertos provenientes de la migración. Producto de la migración existen 88 tramos, que pueden consultarse mediante la siguiente query:

SELECT \*

FROM LOS\_BARONES\_DE\_LA\_CERVEZA.Tramo

Como la cantidad de puertos de la migración es 45, entonces podemos generar 45C2 = 990 tramos, es decir, se pueden dar de alta 990-88=902 tramos nuevos en la situación actual.

Para el alta de un nuevo tramo se debe seleccionar el puerto de origen (de todos los disponibles) de un primer combobox. Automáticamente, se cargarán los posibles puertos destino en un segundo combobox, que son aquellos para los cuales todavía no existe el tramo con el puerto de origen seleccionado:



En este ejemplo, creamos un nuevo tramo que une los puertos **NAIROBI-MALABO.**

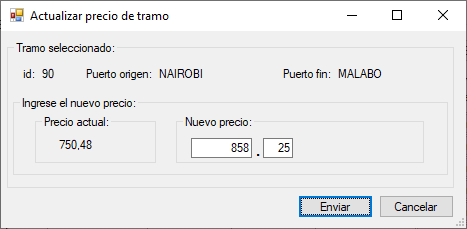
Por último, se debe ingresar un precio para el tramo (validamos que sea correcto) y pulsar el botón *Enviar*.

Para mejorar la experiencia de usuario, los combobox se recargarán de forma automática al dar de alta un nuevo tramo, no permitiendo que se intente dar de alta nuevamente el tramo recién ingresado.

## Modificar el precio de un tramo

Dado que si se cambia el puerto de origen o fin de un tramo éste ya deja de ser el mismo tramo, el único campo que tiene sentido editar en un tramo es su precio. Para ello, creamos esta funcionalidad.

Podremos buscar el tramo a editar ingresando el puerto de inicio o fin. Como siempre, sino ingresamos ningún campo y pulsamos *Buscar* se nos mostrarán todos los tramos disponibles. Al presionar *Editar* en la fila correspondiente al tramo deseado se mostrará una pantalla como la siguiente:



En este ejemplo cambiamos el precio del tramo recién creado y lo incrementamos (por ejemplo, producto de la inflación).

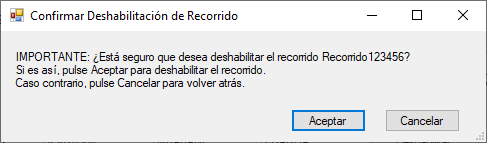
Tanto el alta como la modificación de un tramo impactan en la tabla *Tramo* de la base de datos.

## Deshabilitar recorrido

Esta funcionalidad permite deshabilitar un recorrido. Al deshabilitar un recorrido el mismo no se mostrará más como recorrido disponible para la creación de nuevos viajes.

Podemos buscar recorridos ingresando su puerto de inicio y fin. Solo se mostrarán los recorridos habilitados.

Como ejemplo, podemos deshabilitar el recorrido “Recorrido123456” creado antes. Debemos confirmar dicha acción mediante un cuadro de diálogo.



Cuando deshabilitamos un recorrido se actualiza el campo *recorrido\_estado* del recorrido en cuestión (en la tabla *Recorrido*), pasando de 0 (habilitado) a 1 (deshabilitado).

## Habilitar recorrido

Es la funcionalidad inversa a la anterior. Sólo se mostrarán los recorridos deshabilitados. Dado que por defecto todos los recorridos de la migración están habilitados, el único recorrido deshabilitado actualmente es el recorrido “Recorrido123456” que acabamos de deshabilitar.

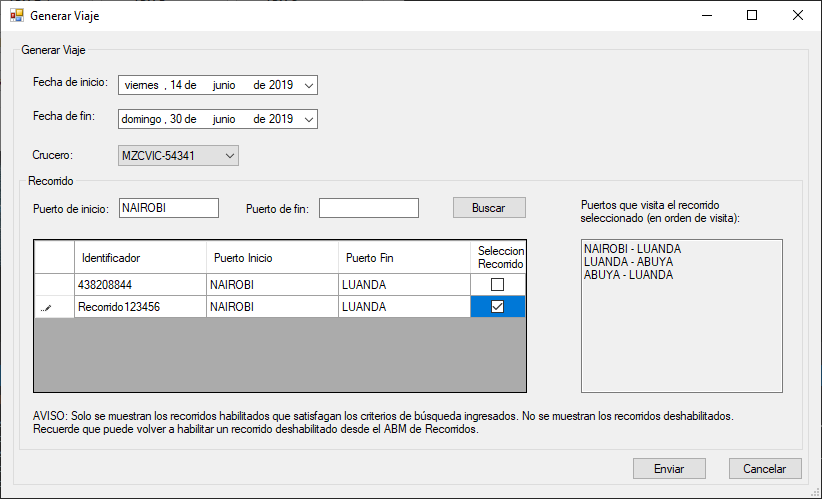
Al habilitar un recorrido, el campo *recorrido\_estado* de su registro pasa de 1 a 0 nuevamente. Como ejemplo, podemos volver a habilitar el recorrido “Recorrido123456”.

# Generar Viaje

Esta funcionalidad es la que permite la creación de nuevos viajes. Para ello, debemos vincular un crucero con un recorrido y determinar una fecha de inicio y fin para el viaje.

El primer paso consiste en ingresar la fecha de inicio y fin de nuestro viaje. Al hacerlo, se cargará de forma dinámica un combobox con los cruceros disponibles en esas fechas. Luego seleccionamos un crucero (por medio de su identificador). Por último, debemos elegir el recorrido que hará el viaje. Para ello se nos mostrarán en un datagridview aquellos recorridos habilitados. Podemos buscar por puerto de inicio y fin.

En el siguiente ejemplo seleccionamos el recorrido “Recorrido123456” que creamos con anterioridad:



Vemos como en un textbox a la derecha se nos muestra el detalle del recorrido, incluyendo todos los puertos por los que pasa, de manera que el administrador que da de alta un nuevo viaje pueda saber las ciudades que contemplará éste, lo que es de especial interés sabiendo que los pasajeros pueden bajarse en un puerto intermedio (aunque deban pagar el total del viaje).

Para dar de alta pulsamos el botón *Enviar* y la aplicación nos confirmará el éxito o no de la operación.

Una validación importante que hacemos es que la fecha de fin del viaje sea posterior a la de inicio.

Algunas validaciones adicionales son la no selección de un crucero y la no selección de un viaje. En ambos casos se informará al usuario cuando éste pulse *Enviar*.

Una vez que el viaje ha sido creado exitosamente pasar a estar disponible para ser comprado.

Para la implementación de esta funcionalidad hacemos uso del procedimiento almacenado *USP\_insertar\_viaje*, invocándolo desde C# con todos los parámetros cargados por el usuario.

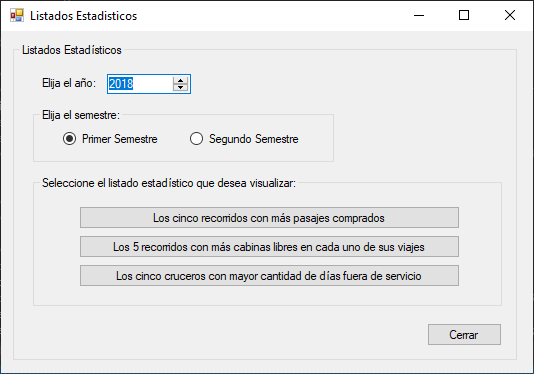
El alta de un nuevo viaje crea un nuevo registro en la tabla *Viaje* de nuestro esquema.

# Comprar/Reservar Viaje

# Pagar Reserva

# Listados Estadísticos

La última funcionalidad de la aplicación permite consultar tres listados estadísticos diferentes. En forma común a todos ellos, el usuario deberá ingresar el año y el semestre para el cuál desea visualizar los resultados, como se ve en la siguiente imagen:



Luego, debe pulsar el botón correspondiente al listado que desea visualizar.

El año se ingresa mediante un datetimepicker cuyo formato se fijo a año solamente. Por defecto se carga el año 2018. Esto asegura que el usuario no pueda ingresar un valor incorrecto y ahorra validaciones manuales.

Como un año tiene sólo dos semestres, optamos por incluir dos radiobutton en un groupbox. Si el usuario selecciona uno, el otro automáticamente se desmarcará. Además, el primer semestre se selecciona por defecto, por lo que de vuelta, no hay que realizar validaciones manuales.

Dado que en este caso no hay posibilidad de sufrir un ataque de SQL Injection, no parametrizamos las consultas en la implementación de esta funcionalidad. ¿Por qué decimos que no se puede presentar este problema? Como el usuario que consulta los listados no tiene que ingresar ningún parámetro, sino simplemente elegir un año y trimestre desde un datetimepicker, las consultas no utilizan ningún dato externo para generarse.

El código de las consultas se encuentra dentro de la clase *FrbaCrucero.ListadoEstadistico.Consultas*.

# Otra información

## Datos personales de los integrantes

|  |  |  |  |
| --- | --- | --- | --- |
| **Alumno** | **Legajo** | **Curso** | **Email** |
| Nicolás Patronelli **(Representante)** | 152.367-3 | K3014 (Sábados mañana) | [nicopatronelli@gmail.com](mailto:nicopatronelli@gmail.com) |
| Leonardo Maestri | 155.993-0 | K3014 |  |
| Martín Acerboni | 159.057-1 | K3014 |  |

## Corrección recibida en la entrega preliminar del DER

Hola.

Su Trabajo de Gestión de Datos ha sido corregido.

Su tp esta correcto.

Saludos,

Corrección Gestión de Datos

1. Dentro del namespace FrbaCrucero.Utils incluimos varias clases de utilidad adicionales. [↑](#footnote-ref-1)