Universitat
Autònoma
de Barcelona

5250: Agenda Deportiva

Memoria del Proyecto Final de Carrera
de Ingeniería Informática
realizado por
Julián Ruiz Burgos
y dirigido por
Xavier Roca Marva
Bellaterra 12 de Septiembre de 2013

Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

El sotasignat, Xavier Roca Marva

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

Julián Ruiz Burgos

I per tal que consti firma la present.

Signat: Xavier Roca Marva

Bellaterra, 13 de Septembre de 20013

Tabla de contenido

1. Introducción

1.1 Motivaciones del proyecto y temática del proyecto

Este PFC nace de la idea de desarrollar un proyecto que una formación y deporte —ya que este último es un aspecto muy importante para mí— y de la intención de que el resultado tenga una utilidad real, que cubra una necesidad real y resulte en una herramienta que otras personas puedan aprovechar.

Con estas dos ideas en mente se describe el proceso de desarrollo de una aplicación de escritorio que permite a los usuarios de crear y administrar sus propias dietas, rutinas de entrenamiento y crear calendarios donde organizar estas dietas y rutinas así como otros eventos personalizados.

Inicialmente estaba previsto que la aplicación además permitiera a los usuarios tener registros de resultados de pruebas de rendimiento y un registro del riesgo de lesión, dependiendo del tipo de ejercicios realizados así como un módulo "inteligente" que mediante algoritmos de inteligencia artificial y redes neuronales aconsejara al usuario los cambios a realizar en su rutina y/o dieta para mejorar sus resultados. Por motivos que se detallarán más adelante estas últimas características descritas no han sido desarrolladas.

El motivo académicamente relevante, que es el que se trata en la memoria, es el proceso de desarrollo de la aplicación: todos los pasos y procedimientos llevados a cabo para el diseño y creación de dicha aplicación.

1.2 Estado del arte

Aunque hoy en día existen muchos programas y aplicaciones relacionadas con el deporte —especialmente con auge de los *smartphones* y *tablets*— la mayoría de estas aplicaciones únicamente ofrecen un registro de la sesión concreta y algunos aspectos de dicha sesión (tiempo, recorrido…) que en muchos casos carecen de rigurosidad o son meras aproximaciones dados unos valores fijos prefijados (como podría ser el consumo calórico).

Otro aspecto a destacar es que la mayoría de estas aplicaciones están ideadas para dispositivos móviles y esto implica ciertas limitaciones en su diseño y uso, como por ejemplo tener abiertos diferentes módulos de una misma aplicación y navegar rápida y fácilmente entre estos, limitaciones que con una aplicación de escritorio no existen.

Algunos ejemplos de aplicaciones de este tipo que podemos encontrar en la actualidad son:

Endomondo: esta aplicación diseñada para dispositivos móviles permite hacer "tracking" del recorrido durante una sesión y lo almacena en su base de datos que los usuarios pueden consultar más tarde en su web. *Endomondo* permite elegir qué clase de deporte o actividad se va a realizar y el cálculo de calorías varía según la elección, aunque este cálculo es una estimación según un estándar, no un cálculo real ya que no tiene en cuenta factores como la intensidad del ejercicio.

MapMyRun/Walk: Similar a *Endomondo*, orientada también a dispositivos móviles. Igual que el anterior recoge con la aplicación móvil el registro de la sesión realizada y permite ver la información en su web.

Adidas MiCoach: Aplicación muy completa. Además de las funciones de las aplicaciones anteriores *Adidas MiCoach* propone diferentes ejercicios a realizar según el deporte practicado y el objetivo. Estos ejercicios son predeterminados para el deporte en sí, sin tener en cuenta las características personales del usuario. Permite usar dispositivos externos como *Kinect* de Xbox, *PSMove* y otros para registrar en tiempo real los ejercicios. El problema de esto último es que se necesitan estos dispositivos extra para sacar el máximo partido.

Esta es una selección de entre un gran número de aplicaciones existentes (las más conocidas o utilizadas). Como ya se ha comentado, la diferencia principal entre estas aplicaciones y la aplicación que se desarrolla en este proyecto es que éste último no pretende centrarse en sesiones individuales de entrenamiento sino más bien en un planning global de los entrenos y la/s dietas.

Otra diferencia substancial que la aplicación propuesta tiene respecto a las anteriores es que la aplicación desarrollada en el proyecto no está orientada a dispositivos móviles. Se podría plantear, sin embargo, una aplicación móvil que hiciera el *tracking* en tiempo real del ejercicio y después se conectara con la aplicación de escritorio. Así se conseguirían registros más rigurosos. Debido al coste temporal y de desarrollo que esta extensión supondría quedará en una de las posibles líneas de continuación del proyecto.

1.3 Descripción de la aplicación

Como ya se ha comentado en el punto 1.1 de la memoria la aplicación permite al usuario crear y administrar entrenamientos y dietas, así como calendarios para realizar el seguimiento de ambas. A continuación se presenta una descripción más detallada de la aplicación.

Cada usuario dispone de un perfil personalizado que contiene información personal del usuario.

El usuario puede crear y editar calendarios que utiliza para realizar el seguimiento —si lo desea— de los ejercicios a realizar cada día de la semana y las diferentes comidas que tomar, según la dieta que haya añadido —si es que lo ha hecho. Este seguimiento se realiza mediante eventos que el usuario puede añadir al calendario. Estos eventos son de diferente tipo, dependiendo de si son eventos que representan un ejercicio, una comida, una dieta o rutina entera o bien un evento de texto simple. Todos los eventos contienen la información correspondiente a la fecha y hora del evento, la duración, la frecuencia de repetición, si es un evento editable o no, si se debe ignorar la información horaria, si es visible o no y opciones de personalización de la apariencia: color, estilo de fuente, tamaño, etc. Además, los eventos de tipo ejercicio contienen información específica del ejercicio a realizar: número de repeticiones, series, peso, etc.

Los ejercicios que el usuario puede utilizar se cargan de una base de datos que los clasifica y organiza por categorías según diferentes aspectos (ya sean aspectos por defecto o categorías creadas por el propio usuario). Cada ejercicio se compone de una descripción, de la lista de categorías a las que pertenece y los músculos que ejercita. De la misma manera las categorías también se componen de una descripción y la lista de ejercicios que son contenidos en dicha categoría.

De forma similar a los ejercicios, de la base de datos se cargan los alimentos y comidas. Los alimentos son clasificados según su tipo: cereales, carnes, etc. y las comidas según si son comidas por defecto o bien si son comidas creadas por el usuario. Los alimentos, además del tipo, se componen de descripción y aporte calórico. Las comidas se componen de una lista de los alimentos que las forman, una descripción y el aporte energético total de la comida.

Las rutinas y dietas también se cargan de la base de datos y poseen una estructura análoga: ambas se componen del nombre, la descripción y una tabla de los ejercicios o comidas a realizar o tomar cada día. Además tanto rutinas como dietas almacenan el consumo o aporte calórico total respectivamente.

1.4 Planificación temporal

La planificación temporal inicial —descrita en el informe previo— hablaba de cuatro vías de trabajo en paralelo: desarrollo de la aplicación, documentación de ámbito deportivo, documentación de los conocimientos necesarios para la creación del *Módulo Inteligente* y finalmente obtención de muestras empíricas reales que luego se utilizarían para evaluar dicho módulo. Esta planificación se describe en el Diagrama de Gantt del informe previo —ILUSTRACIÓN 43.

Conforme el desarrollo del proyecto iba evolucionando en el tiempo esta planificación se vio alterada por los distintos cambios en éste, así como en el diseño de la aplicación.

Al realizar el análisis de requerimientos de la futura aplicación se llegó a la conclusión que era inviable el diseño del módulo *Inteligente* ya que para poder crear este módulo era necesaria tener casi la totalidad de la aplicación diseñada con el resto de módulos totalmente funcionales y así poder utilizarlos para representar los resultados obtenidos en las muestras empíricas dentro de la aplicación —era necesaria toda la estructura de ejercicios, rutinas, comidas y dietas. En consecuencia, las tareas de documentación para la creación de dicho módulo y la obtención de muestras empíricas desaparecen y el tiempo previsto para su realización es absorbido por el resto de tareas. Del mismo modo, la tarea de documentación deportiva, en especial la documentación referente a la parte de dietética, también desaparece ya que al desaparecer el módulo *inteligente* desaparece en el proyecto actual la necesidad de estudiar este apartado, porque el módulo de dietas no tendrá una opción de creación automática de dietas ni rutinas.

Debido a estos cambios el proyecto, que inicialmente iba a centrarse en el mencionado módulo *inteligente* y por tanto en el ámbito de la inteligencia artificial, pasa a centrarse en el desarrollo de la aplicación, perteneciente al ámbito de la Ingeniería del Software.

La planificación final del proyecto queda dividida, finalmente, en cuatro bloques: Análisis de requerimientos, Diseño de clases, Diseño de la aplicación e Implementación y pruebas: ILUSTRACIÓN 44, ILUSTRACIÓN 45

En el diagrama se observa que la mayoría de tareas se realizan de forma secuencial y esto es debido a que el proyecto ha sido llevado a cabo por una sola persona y por lo tanto no se podían realizar al mismo tiempo.

1.5 Herramientas utilizadas

La herramienta principal para desarrollar el proyecto ha sido *Microsoft Visual Studio 2012*. Ésta no ha sido elegida de forma trivial, sino que fue después de estudiar diversos aspectos del desarrollo de la aplicación que se decidió usar dicha herramienta. Los criterios o aspectos estudiados fueron el lenguaje de programación y en especial la inclusión de las herramientas necesarias para desarrollar la interfaz gráfica de la aplicación.

En lo que respecta a las herramientas de desarrollo de la interfaz, *Visual Studio 2012* contiene todos los elementos necesarios para desarrollar una aplicación, incluyendo su interfaz gráfica, basada en formularios —Windows Forms— que el desarrollador tiene a su disposición. *Visual Studio 2012* permite además el uso del framework *.NET* y enlazar de forma sencilla con bases de datos SQL.

Otra herramienta utilizada ha sido Internet. Pese a ser una fuente de información más que una herramienta propiamente dicha, es necesario destacar la importancia que ésta ha tenido durante todo el desarrollo del proyecto. Además del desarrollo de la aplicación en sí Internet ha sido clave a la hora de solucionar problemas relacionados con la herramienta anteriormente descrita. La cantidad inmensurable de sitios web, foros, tutoriales… que pueblan Internet ponen a disposición soluciones para cualquier problema o dificultad que pueda surgir durante el desarrollo del proyecto y por eso es considerada como una herramienta más, quizá la más importante de todas.

2. Análisis de requerimientos

2.1 Identificación de los interesados

Cuando se habla de interesados —*stakeholders*— se piensa en todas aquellas personas u organizaciones que pueden afectar de forma positiva o negativa al desarrollo del proyecto. En el caso de este proyecto los principales interesados serían el desarrollador del proyecto y los usuarios finales de la aplicación, entre los que también se incluye el desarrollador del proyecto.

Formar parte de los interesados por partida doble —como creador y usuario final— permite enfocar el desarrollo del proyecto de manera que a la hora de pensar y decidir qué se va a hacer y cómo se va a hacer se tienen en cuenta directamente las necesidades del usuario final. Por ejemplo: la idea es que la aplicación final la puedan usar todo tipo de usuarios independientemente de su habilidad con el uso de un ordenador y así llegar al máximo número de usuarios posibles. Teniendo esto en mente es *relativamente* sencillo diseñar qué tipo de interfaz conviene que la aplicación tenga para facilitar el que el uso de la aplicación sea sencillo y fluido, con una curva de aprendizaje poco pronunciada.

Más allá del usuario final podríamos pensar en interesados de carácter intermedio, es decir, no usuarios de la aplicación en sí, pero si organismos o personas interesadas en adquirir la aplicación para dar un servicio a través de ella. Algunos interesados de este tipo podrían ser entrenadores personales —que quieran utilizar la aplicación para que sus clientes la usen y así administrar los calendarios de estos—, Jefes de equipos deportivos, etc.

2.2 Diagrama Entidad/Relación

Los diagramas de Entidad/Relación permiten representar las entidades relevantes de un sistema de información y las interrelaciones entre dichas entidades.

En este proyecto, nuestro diagrama de Entidad/Relación está formado por las entidades *Usuario, Alimento, Comida, Dieta, Ejercicio, Categoría, Rutina, evento y Calendario*:

- **Usuario**: Representa al individuo que utiliza la aplicación.
- **Alimento**: Representa un alimento, como su nombre indica. *Arroz Blanco, Atún fresco,* etc. Son ejemplos de alimentos.
- **Comida**: Representa un número determinado de alimentos agrupados en una sola entidad. *Macarrones con tomate y queso, Huevos fritos con patatas*, etc. Son ejemplos de comidas.
- **Dieta**: Representa un número determinado de comidas agrupadas por días para formar lo que comúnmente se conoce como una dieta. La imagen adjuntada a continuación es un ejemplo de dieta.

	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO
Desayuno	Un vaso de leche con un puñado de cereales (maíz) y una o dos galletas tipo "María" o "Digestive". Medio kiwi	Un vaso de leche con media tostada integral, un poco de aceite de oliva y jamón cocido. Un zumo de naranja natural.	Un vaso de leche con un puñado de cereales (avena, trigo y arroz) y una o dos galletas. Medio kiwi.	Un vaso de leche con media tostada integral, un poco de aceite de oliva y jamón cocido. Un zumo de naranja natural.	Un vaso de leche con un puñado de cereales y una magdalena integral. Una mandarina.	Un vaso de leche con un puñado de cereales y una o dos galletas. Medio melocotón.	Un vaso de leche con media tostada integral, un poco de aceite de oliva y jamón cocido. Un zumo de naranja natural.
Almuerzo	Una manzana	Una pera	Un plátano	Una pera	Un melocotón	Un kiwi	Macedonia de frutas y nueces
Comida	Ternera a la plancha con verduras (patatas, guisantes y judías) hervidas. Una pera.	Crema de verduras (calabacín, champiñones, zanahoria). Merluza hervida o a la plancha. Medio yogurt con media fruta.	Pasta con pollo y verduras salteados (calabacín, guisantes, tomate, champiñones...). Una manzana.	Ensalada. Salteado de verduras (calabacín y patata) con lomo de cerdo a la plancha. Medio yogurt natural con media fruta.	Lentejas casi vegetarianas (con judías, patata, zanahoria y taquitos de jamón). Un yogurt natural.	Ensalada de arroz. Pez espada a la plancha. Manzana.	Estofado de verduras (patata, champiñones, puerro y zanahoria) con albóndigas caseras de pollo y ternera. Un flan casero.
Merienda	Un bocadillo pequeño (sandwich) de jamón cocido y queso fresco, con aceite y tomate. Una mandarina	Una tortilla a la francesa con pan y tomate. Un kiwi.	Escalopín casero de pollo y pavo cocido con queso y tomate. Un yogurt de frutas.	Un bocadillo pequeño de jamón con aceite y tomate. Un vasito (un bol pequeño) de queso fresco con fruta (kiwi, melocotón, uva, pera...).	Una tortilla a la francesa con pan y tomate. Uva.	Un bocadillo pequeño de lomo con aceite y tomate. Un vasito (un bol pequeño) de queso fresco con fruta (kiwi, melocotón, uva, pera...).	Una crema de champiñones con atún. Un batido casero de leche y frutas.
Cena	Un vaso de leche con cereales (arroz inflado)	Un vaso de leche con una galleta.	Un vaso de leche con una magdalena integral.	Un yogurt con cereales.	Un vaso de leche con cereales.	Un yogurt con cereales integrales variados.	Un yogurt con una galleta.

Ilustración 1: Dieta, ejemplo extraído de www.saludymedicina.org

- **Ejercicio:** Representa un ejercicio o actividad física concreta, realizada de una manera determinada con objetivo de ejercitar una o varias partes del cuerpo. *Flexiones*, *Dominadas*, *Carrera corta*, etc. Son ejemplos de ejercicios.

- **Categoría:** Una categoría es una entidad abstracta en la que se agrupan diferentes ejercicios, habitualmente por una o varias características comunes como por ejemplo el grupo muscular que trabajan, si son individuales o en grupo, si son para mejorar uno u otro atributo, etc. *Tren superior, Pal Chagui, Respiraciones,* etc. Son ejemplos de categorías.

- **Rutina:** Una rutina es un conjunto de ejercicios agrupados en uno o varios días o sesiones. La imagen adjuntada a continuación es un ejemplo de rutina.

LUNES	MIERCOLES	VIERNES
ESPALDA	**ESPALDA**	**ESPALDA**
Remo con mancuernas 3x15	Jalones tras nuca 3x15	Jalones frontales 3x15
PECHO	**PECHO**	**PECHO**
Press plano mancuernas3x10	Aperturas planas 3x10	Pullover 3x10
HOMBRO	**HOMBRO**	**HOMBRO**
Press con mancuernas 3x15	Elevaciones laterales 3x15	Elevaciones frontales 3x15
BICEPS	**BICEPS**	**BICEPS**
curl alterno 3x15	Curl martillo 3x15	Curl concentrado 3x15
TRICEPS	**TRICEPS**	**TRICEPS**
extensiones en polea 3x15	extensiones en polea 3x15	extensiones en polea 3x15
patadas 3x15	Press Francés 3x15	patadas 3x15
PIERNA	**PIERNA**	**PIERNA**
Sentadilla 3x15	Extensión cuádriceps 3x15	Sentadilla 3x15
Peso muerto 3x15	Aductor/Abductor 3x20	Peso muerto 3x15
ABDOMEN	**ABDOMEN**	**ABDOMEN**
Elevación de rodillas 3x20	Elevación de rodillas 3x20	Elevación de rodillas 3x20
Oblicuos con peso 3x20	Oblicuos con peso 3x20	Oblicuos con peso 3x20
Encogimientos 3x20	Encogimientos 3x20	Encogimientos 3x20

Ilustración 2: "Rutina Vane", ejemplo extraído de www.musculacionparaprincipiantes.com

- **Evento:** Un evento representa un acontecimiento puntual de duración finita programado por el usuario. Un evento puede ser de tipos diferentes: un ejercicio, una dieta, una rutina, etc. Son ejemplos de eventos posibles.
- **Calendario:** Representa lo que comúnmente es conocido como calendario. En calendario se pueden incluir eventos creados por el usuario.

Las relaciones entre estas entidades se ven representadas en el diagrama de Entidad/Relación adjuntado en el anexo siguiente (ILUSTRACIÓN 46, página 90). Este diagrama será utilizado más adelante para el diseño del sistema de clases y el consiguiente diagrama de clases.

2.3 Requerimientos funcionales

A la hora de determinar los requerimientos funcionales de la aplicación se han documentado todas aquellas funcionalidades que la aplicación debería tener. Esta documentación se realiza mediante casos de uso, describiendo así los pasos para llevar a cabo dicha funcionalidad y su efecto en la aplicación.

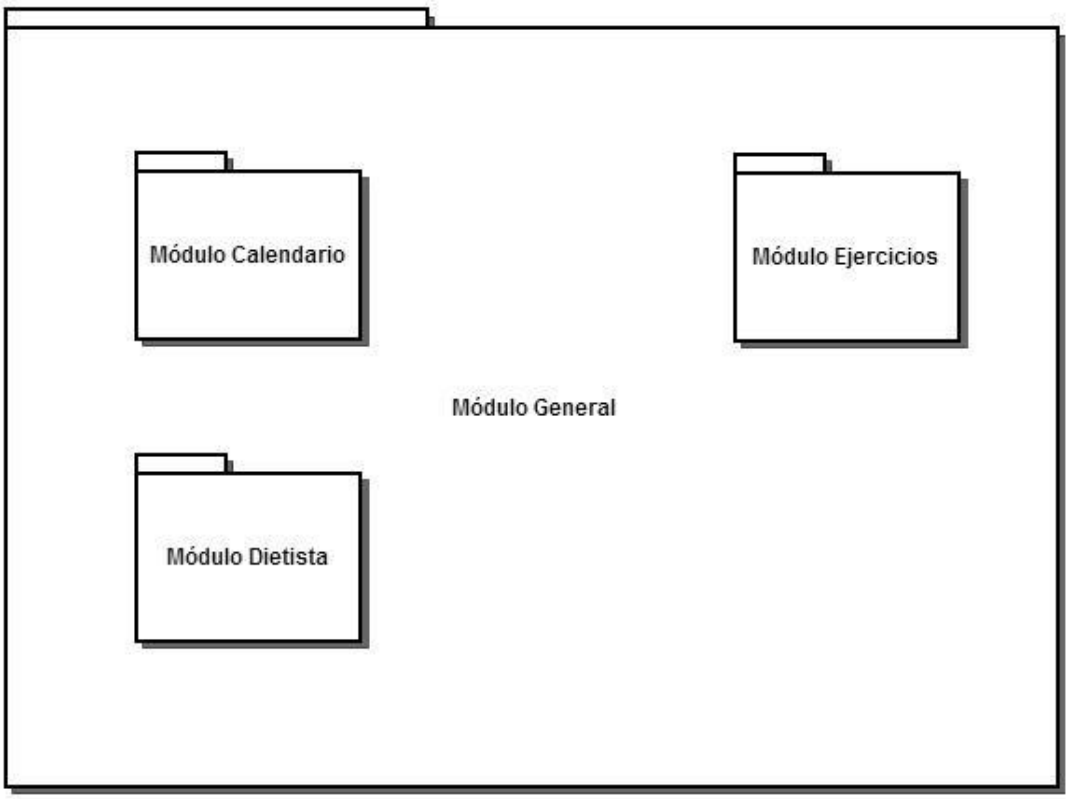
La lista de casos de uso se encuentra en el documento de Modelo de casos de uso —el próximo apartado de la memoria—, que como bien indica su nombre contiene todos los casos de usos de la aplicación, separados por módulos.

2.3.1 Modelo de Casos de uso

El propósito de este apartado es definir las funcionalidades del sistema y el contexto de este —interacción con entidades externes o actores. Se presentan los diferentes diagramas de casos de uso para los módulos en que se divide el sistema y las descripciones de cada actor y caso de uso.

El modelo de casos de uso que se presenta consta de cuatro módulos y los diagramas de casos de uso correspondientes a cada módulo: *General, Calendario, Ejercicios, Dietista*.

El siguiente diagrama muestra la arquitectura inicial en que se divide el programa, en cuanto a módulos se refiere.



2.3.1.1 Actores

Usuario

Este actor representa al usuario final del programa. El usuario interactúa con el programa mediante la instalación de este en su ordenador.

Administrador

Este actor representa al usuario de tipo administrador. Este usuario puede modificar todos los perfiles de usuario (del resto de usuarios) así como configuraciones del programa que no están disponibles para los usuarios convencionales.

2.3.1.2 Casos de Uso

2.3.1.2.1 Módulo General

Crear Nuevo Usuario

Como indica el nombre este caso de uso describe los pasos para crear un nuevo usuario, así como las restricciones a tener en cuenta al hacerlo.

Ver Información de Usuario

El objetivo del caso de uso es que el usuario pueda ver la información correspondiente al usuario —nombre, apellidos, correo electrónico, etc.

Eliminar Usuario

Este caso de uso describe los pasos a seguir para eliminar un usuario existente. Implicará eliminar la entrada de la base de datos así como las entradas correspondientes a los otros módulos y otras entradas derivadas.

Loguear Usuario

Este caso de uso describe el procedimiento para conectarse como usuario de la aplicación.

Desconectar Usuario

Este caso de uso describe el procedimiento para desconectarse como usuario de la aplicación.

Cargar Módulo

Este caso de uso describe los pasos a seguir para cargar un módulo del programa. Cada módulo a utilizar debe de ser cargado de forma independiente siguiendo los pasos descritos en este caso de uso.

Cerrar Módulo

Este caso de uso describe la funcionalidad de cerrar un módulo abierto. Esto afecta únicamente al módulo en cuestión y a la base de datos, ya que se guardará el estado en que se encuentra el módulo antes de cerrar.

Redimensionar Ventana

Este caso de uso describe la funcionalidad de redimensionar el tamaño de la ventana.

2.3.1.2.2 Módulo Calendario

Crear Nuevo Calendario

En este caso de uso se describe la funcionalidad de crear un nuevo calendario y los pasos para llevarla a cabo.

Abrir Calendario

Este caso de uso describe la funcionalidad de abrir un nuevo calendario y el procedimiento para llevar a cabo dicha acción.

Cerrar Calendario

En este caso de uso se describe la funcionalidad de cerrar un calendario abierto, los pasos a seguir y el alcance de esta acción.

Editar Opciones de Calendario

Este caso de uso describe la funcionalidad de modificar las opciones de un calendario existente como pueden ser la fecha de inicio y fin de este, como se representa dicho calendario, etc.

Eliminar Calendario

Este caso de uso describe el procedimiento para borrar un calendario de un usuario y los efectos que esta acción tiene sobre el programa.

Guardar Calendario Como Imagen

Este caso de uso describe la funcionalidad de guardar un calendario existente como una imagen y los pasos para realizar dicha acción.

Imprimir Calendario Existente

Este caso de uso describe la funcionalidad de imprimir un calendario existente y los pasos para realizar dicha tarea.

Cambiar tipo de vista

Este caso de uso describe la funcionalidad de cambiar de tipo de visa de un calendario (diaria, semanal, mensual, etc.)

Añadir Nuevo Evento

Este caso de uso describe la funcionalidad de añadir un nuevo evento a un calendario existente y los pasos para llevar a cabo dicha acción.

Modificar Evento

Este caso de uso describe la funcionalidad de modificar un evento existente de un calendario.

Replicar Evento

Este caso de uso describe la funcionalidad de replicar un evento existente como un nuevo evento de un calendario. Este evento no tiene por qué ser una copia exacta del original, sino que se puede personalizar como si de un nuevo evento se tratara.

Eliminar Evento

Este caso de uso describe la funcionalidad de eliminar un evento existente de un calendario y los efectos de dicha acción.

Modificar Opciones de Módulo

El objetivo del caso de uso es describir la funcionalidad que permite al usuario configurar una serie de opciones o parámetros del módulo *Calendario*. Esta configuración afectara únicamente al módulo *Calendario* y al usuario logueado.

2.3.1.2.3 Módulo Ejercicios

Modificar Opciones de Módulo

Este caso de uso describe la funcionalidad de modificar las opciones del *Módulo Ejercicios* — las opciones personalizables de cada módulo varían según qué módulo sea.

Cerrar Módulo

Este caso de uso describe la funcionalidad de cerrar el módulo abierto.

Crear Ejercicio

Este caso de uso describe la funcionalidad de crear un nuevo tipo de ejercicio.

Abrir Ejercicio

Este caso de uso describe la funcionalidad de abrir un ejercicio existente en la lista de ejercicios.

Modificar Ejercicio

Este caso de uso describe la funcionalidad de modificar un tipo de ejercicio ya existente.

Eliminar Ejercicio

Este caso de uso describe la funcionalidad de borrar un ejercicio existente de la lista de ejercicios o de una rutina y los efectos que esta acción tiene.

Añadir Ejercicio a Rutina

Este caso de uso describe la funcionalidad de añadir un tipo de ejercicio existente a una rutina.

Añadir Ejercicio como Evento a Calendario

Este caso de uso describe la funcionalidad de añadir un ejercicio a un calendario existente como evento de este.

Crear Rutina

Este caso de uso describe la funcionalidad de crear una nueva rutina.

Abrir Rutina

Este caso de uso describe la funcionalidad de abrir una nueva rutina existente.

Cerrar Rutina

Este caso de uso describe la funcionalidad de cerrar una rutina abierta.

Editar Opciones de Rutina

Este caso de uso describe la funcionalidad de modificar las opciones de una rutina existente.

Eliminar Rutina

Este caso de uso describe la funcionalidad de eliminar una rutina existente y los efectos de dicha acción.

Replicar Rutina

Este caso de uso describe la funcionalidad de crear una copia de una rutina existente.

Imprimir Rutina

Este caso de uso describe la funcionalidad de imprimir una rutina.

Guardar Rutina como Imagen.

Este caso de uso describe la funcionalidad de guardar una rutina en formato de imagen.

Crear Categoría

Este caso de uso describe la funcionalidad de crear una nueva categoría de ejercicios.

Abrir Categoría

Este caso de uso describe la funcionalidad de abrir una categoría existente.

Modificar Categoría

Este caso de uso describe la funcionalidad de modificar una categoría existente.

Añadir Ejercicio a Categoría

Este caso de uso describe la funcionalidad de añadir un tipo de ejercicio existente a una categoría.

Eliminar Categoría

Este caso de uso describe la funcionalidad de eliminar una categoría existente.

2.3.1.2.4 Módulo Dietista

Modificar Opciones de Módulo

Este caso de uso describe la funcionalidad de modificar las opciones del módulo Dietista (las opciones personalizables de cada módulo varían según qué módulo sea).

Cerrar Módulo

Este caso de uso describe la funcionalidad de cerrar el módulo abierto.

Crear Alimento

Este caso de uso describe la funcionalidad de crear un nuevo tipo de alimento.

Modificar Alimento

Este caso de uso describe la funcionalidad de modificar las propiedades de un alimento ya creado.

Abrir Alimento

Este caso de uso describe la funcionalidad de abrir un alimento ya existente en la lista de alimentos.

Eliminar Alimento

Este caso de uso describe la funcionalidad de eliminar un alimento ya existente en la lista de alimentos.

Añadir Alimento a Comida

Este caso de uso describe la funcionalidad de añadir un alimento existente a una comida.

Crear Comida

Este caso de uso describe la funcionalidad de crear una nueva comida —las comidas posteriormente se añaden a las dietas, o a los calendarios como eventos.

Modificar Comida

Este caso de uso describe la funcionalidad de modificar las propiedades de una comida existente en la lista de comidas.

Abrir Comida

Este caso de uso describe la funcionalidad de abrir una comida existente en la lista de comidas.

Añadir Comida como evento

Este caso de uso describe la funcionalidad de añadir una comida como un evento independiente —no como parte de una dieta— a un calendario existente.

Añadir Comida a Dieta

Este caso de uso describe la funcionalidad de añadir una comida a una dieta ya existente.

Eliminar Comida

Este caso de uso describe la funcionalidad de eliminar una comida existente de la lista de comidas o de una dieta concreta.

Crear Dieta

Este caso de uso describe la funcionalidad de crear una nueva dieta.

Abrir Dieta

Este caso de uso describe la funcionalidad de abrir una dieta ya existente en la lista de dietas.

Modificar Dieta

Este caso de uso describe la funcionalidad de modificar una dieta existente en la lista de dietas.

Cerrar Dieta

Este caso de uso describe la funcionalidad de cerrar una dieta abierta.

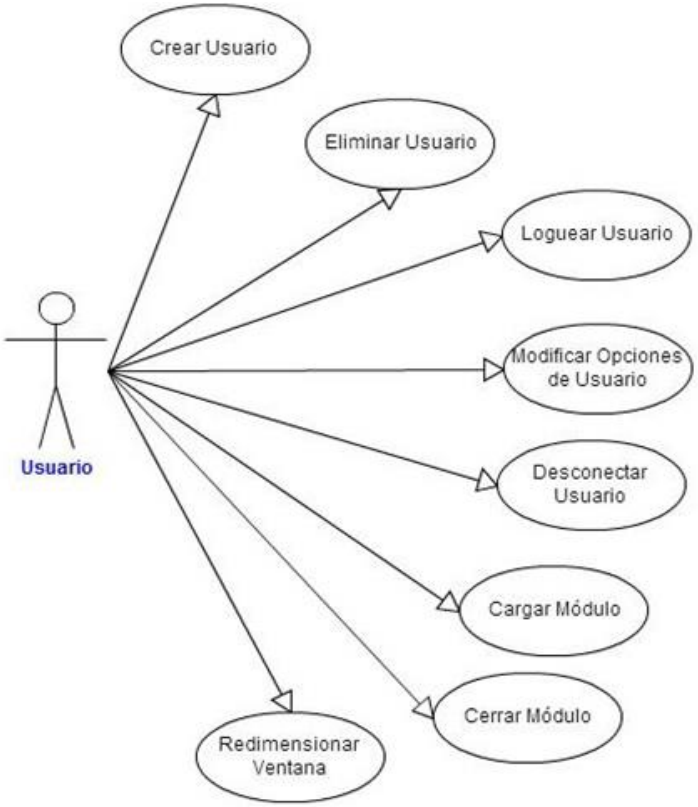
Eliminar Dieta

Este caso de uso describe la funcionalidad de eliminar una dieta existente en la lista de dietas.

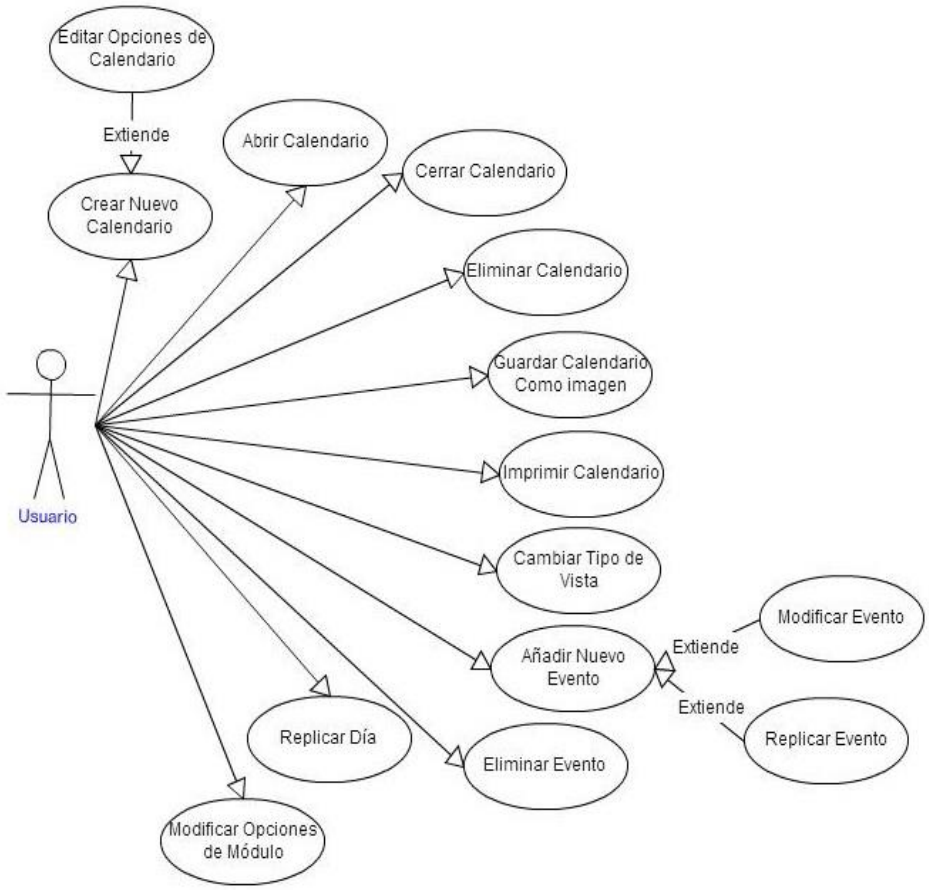
2.3.1.3 Vistas

A continuación se muestran las diferentes vistas funcionales del programa, separadas por módulos, cada una con sus casos de uso.

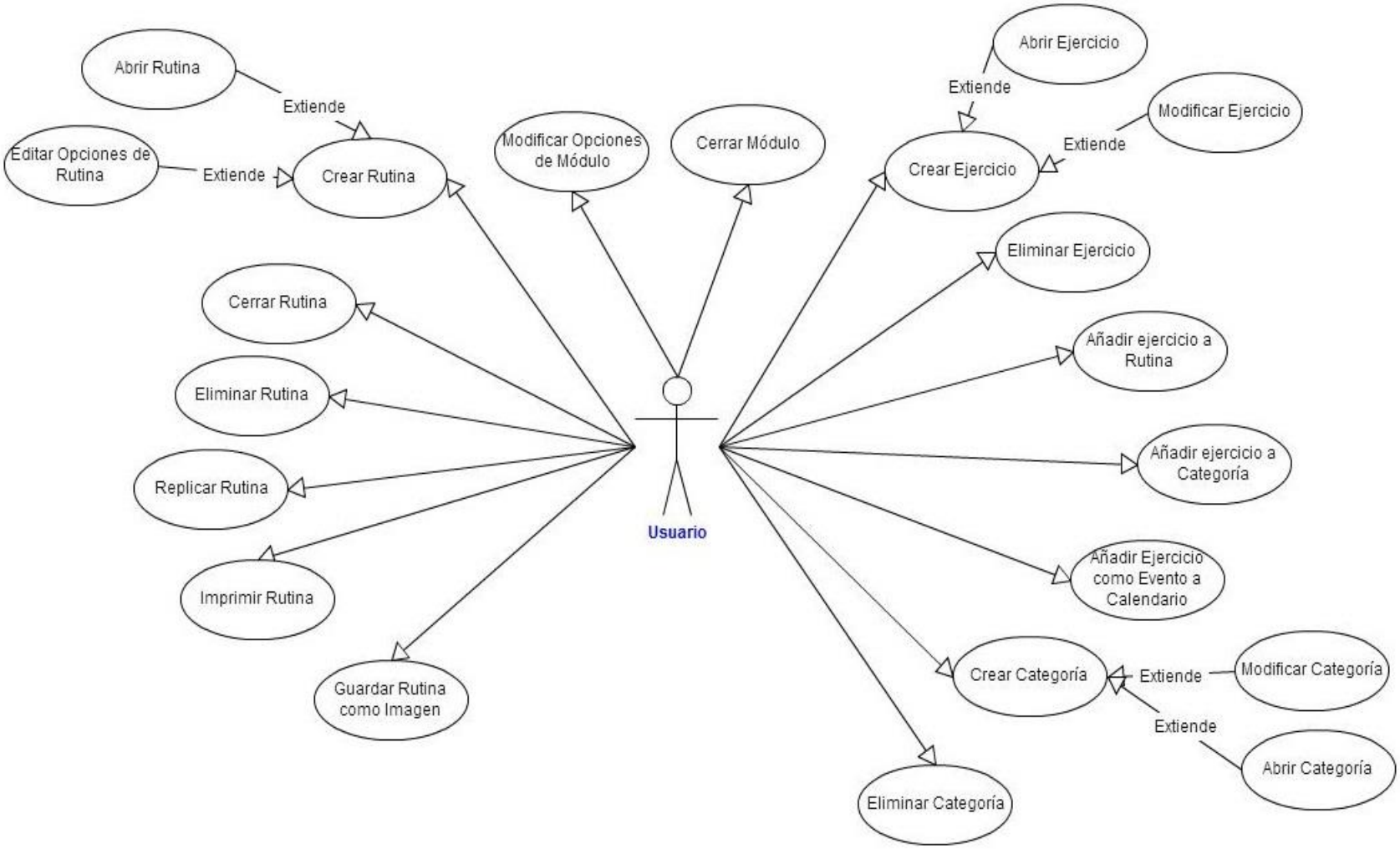
Módulo General



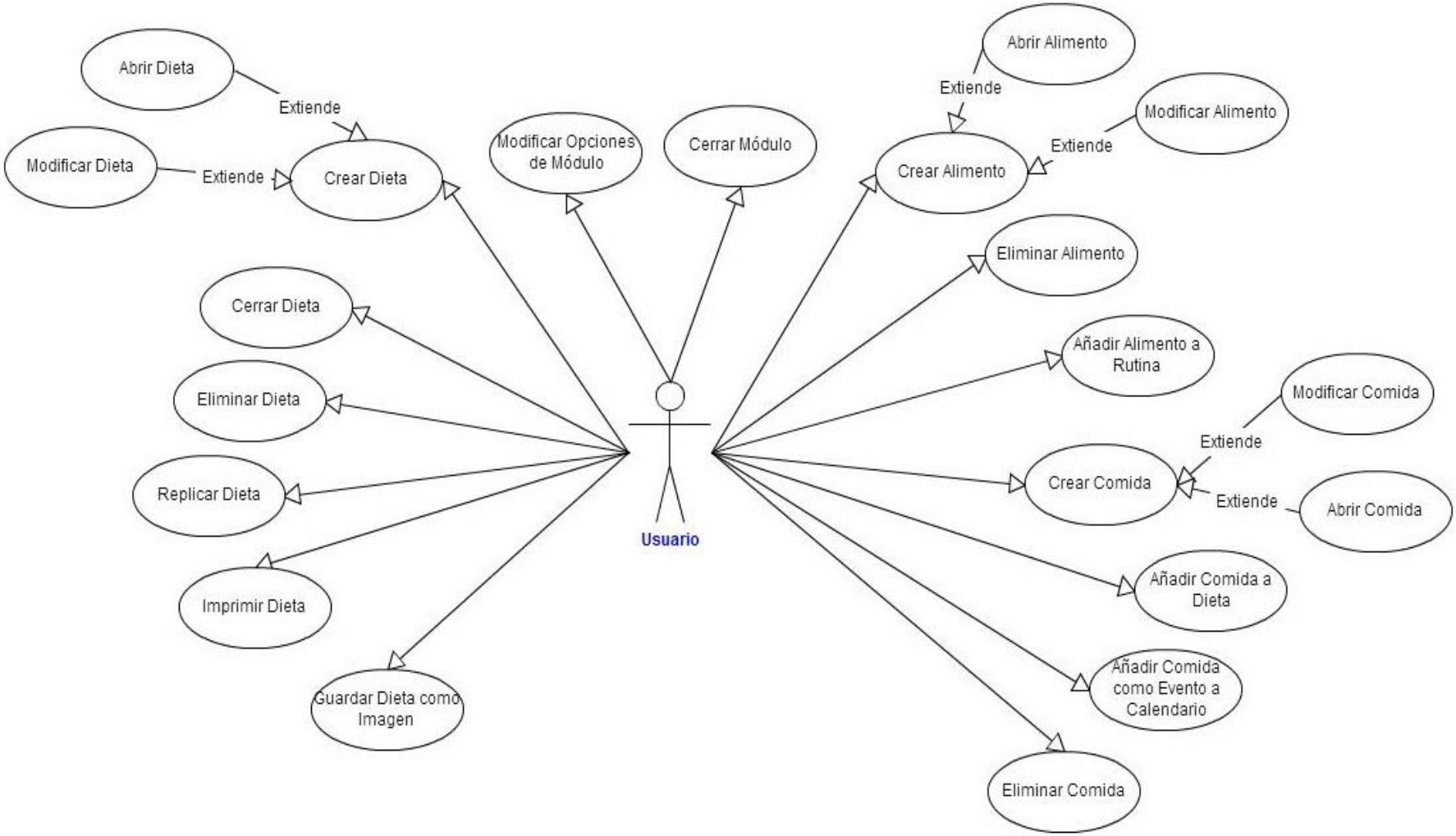
Módulo Calendario



Módulo Ejercicios



Módulo Dietista



2.3.2 Diagramas de secuencia

Los diagramas de secuencia son una herramienta muy útil a la hora de modelar la interacción entre los diferentes objetos de la aplicación y la implementación de dicho escenario. Se utiliza la descripción de los casos de uso para determinar qué objetos son necesarios para la implementación de cada escenario. Las líneas verticales discontinuas muestran la activación de los objetos que intervienen en el escenario y los mensajes pasados entre objetos se muestran mediante flechas horizontales.

Los diagramas de secuencia están separados por módulos:

2.3.2.1 Módulo General

- Crear Usuario (ILUSTRACIÓN 47)
- Conectar y desconectar Usuario (ILUSTRACIÓN 48)
- Ver información de Usuario (ILUSTRACIÓN 49)
- Cargar Módulo (ILUSTRACIÓN 50)
- Cerrar Aplicación (ILUSTRACIÓN 51)

2.3.2.2 Módulo Ejercicios

- Crear Ejercicio (ILUSTRACIÓN 52)
- Abrir Ejercicio (ILUSTRACIÓN 53)
- Modificar Ejercicio (ILUSTRACIÓN 54)
- Eliminar Ejercicio (ILUSTRACIÓN 55)
- Añadir Ejercicio a Rutina (ILUSTRACIÓN 56)
- Crear o Modificar Rutina (ILUSTRACIÓN 57)
- Replicar Rutina (ILUSTRACIÓN 58)
- Guardar Rutina como Imagen (ILUSTRACIÓN 59)
- Cerrar Módulo (ILUSTRACIÓN 60)

Algunos diagramas de secuencia se han omitido ya que al pensar en la estructura y forma de interactuar de los elementos ya se hizo de tal manera que algunos casos de uso funcionaran de manera análoga a otros. Este es el caso de los casos de uso de *Crear Categoría*, *Abrir Categoría*, *Modificar Categoría* y *Eliminar Categoría*; análogos a *Crear, Abrir, Modificar* y *Eliminar Ejercicio respectivamente.*

2.3.2.3 Módulo Dietista

- Crear Alimento (ILUSTRACIÓN 61)
- Modificar Alimento (ILUSTRACIÓN 62)
- Eliminar Alimento (ILUSTRACIÓN 63)
- Añadir Alimento a Comida (ILUSTRACIÓN 64)
- Crear/Modificar Dieta (ILUSTRACIÓN 65)
- Añadir Comida a Dieta (ILUSTRACIÓN 66)

Como sucedía con el módulo anterior algunos diagramas de secuencia se han omitido ya que su funcionamiento es análogo al de otros casos ya descritos anteriormente:

Crear, Abrir, Modificar y Eliminar Comida son análogos respectivamente a *Crear, Abrir, Modificar y Eliminar Ejercicio* (del módulo *Ejercicios*) respectivamente; *Replicar Dieta* es análoga a *Replicar Rutina* del módulo *Ejercicios* y finalmente *Cerrar Módulo* sigue el mismo procedimiento en todos los módulos.

2.3.2.4 Módulo Calendario

- Crear Calendario (ILUSTRACIÓN 67)
- Añadir evento de tipo Normal (ILUSTRACIÓN 68)
- Añadir evento de tipo Comida (ILUSTRACIÓN 69)
- Añadir evento de tipo Ejercicio (ILUSTRACIÓN 70)
- Añadir evento de tipo Dieta (ILUSTRACIÓN 71)
- Añadir evento de tipo Rutina (ILUSTRACIÓN 72)
- Eliminar un evento (ILUSTRACIÓN 73)
- Replicar un evento (ILUSTRACIÓN 74)
- Editar evento (ILUSTRACIÓN 75)

Los diagramas de secuencia de *Eliminar* y *Abrir Calendario* no se han representado ya que su funcionamiento es análogo al de *Eliminar* y *Abrir Ejercicio, Comida,* Etc.

Una vez terminado el análisis de requerimientos podemos pasar a la siguiente fase de desarrollo: el diseño.

3. Diseño

3.1 Método de desarrollo

El método de desarrollo empleado para llevar a cabo el proyecto ha sido un método iterativo e incremental. Este método consiste en dividir el proceso de desarrollo en iteraciones, desarrollando el software de forma incremental para así poder aprovechar lo aprendido y realizado en la iteración anterior. El aprendizaje proviene del desarrollo propiamente dicho y de las pruebas realizadas en cada iteración del desarrollo que comienza por una implementación simple de los requerimientos del sistema. Esta implementación se irá mejorando hasta que la aplicación esté completa. En cada iteración se realizan cambios en el diseño, se agregan nuevas funcionalidades y capacidades al sistema.

Las ventajas de usar este método de desarrollo saltan a la vista, ya que además permite decidir el rumbo que tomará la aplicación teniendo en cuenta el trabajo realizado hasta el momento y por encima de todo, gracias al factor aprendizaje cada nueva etapa de desarrollo permite más optimización que la etapa anterior ya que como se ha comentado se pueden aplicar todas las mejoras y nuevos conocimientos adquiridos en las iteraciones previas. Además, en este caso el cliente y desarrollador son la misma persona, lo que facilita trabajar directamente con el cliente que suele ser uno de los principales problemas con este método de desarrollo.

Las etapas en las que se divide el método de desarrollo descrito son: *Análisis de requerimientos, Diseño, Implementación y Test.*

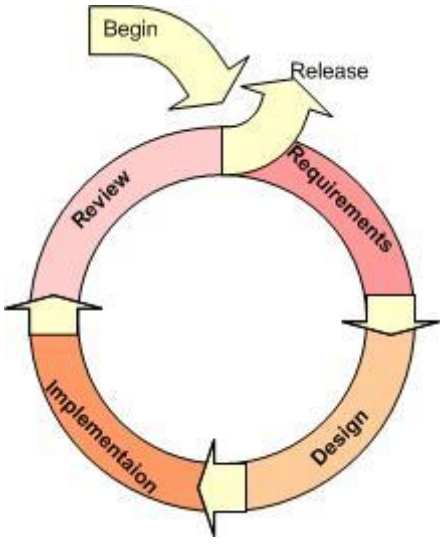


Ilustración 3: Representación gráfica del método Iterativo incremental. Ejemplo extraído de www.synders.us

3.2 Arquitectura de la aplicación

La arquitectura de la aplicación continúa con la línea descrita en apartados anteriores de la memoria: la división en módulos es trasladada al ámbito del desarrollo.

Aprovechando esta división la aplicación queda constituida por cuatro módulos funcionales, parcialmente independientes unos de otros — parcialmente porque aunque operan de forma independiente es necesario en algunos casos haber cargado previamente un módulo concreto para cargar otro. Esto sucede con el *Módulo General*, que se carga al ejecutar la aplicación. Este módulo debe estar abierto siempre, ya que es desde este módulo desde el cual se abren el resto de módulos. Así mismo, para poder cargar el *Módulo Calendario* es necesario cargar también los módulos *Ejercicios* y *Dietista*.

Esta división, sin embargo, no se ve directamente reflejada en la estructura interna de las clases de la aplicación, sino que se podría decir que es más una división lógica que se verá *físicamente* plasmada en la fase de implementación.

3.2.1 Diseño de clases

Partiendo del diagrama Entidad/Relación creado en la fase de Análisis de requerimientos y teniendo en cuenta las funcionalidades que se pensaron para la aplicación se llevó a cabo el diseño de las clases y la estructura de datos interna de la aplicación. En lo que respecta a las clases nos encontramos con las siguientes clases —algunas de ellas en correspondencia directa con las entidades del diagrama E/R mientras que otras entidades se corresponden con más de una clase. Para que resulte más sencillo ver la relación entre las entidades teóricas —por llamarlas de alguna manera— de las clases internas se presentan las entidades y se describe/n la/s clase/s a las que equivale/n.

3.2.1.1 Módulo General

No hay ninguna clase que pertenezca a este módulo. La entidad *Usuario*, cuya clase pertenecería a este módulo, no tiene correspondencia como clase. La información referente al usuario es tratada por las estructuras internas que se explicarán en el siguiente apartado y por la base de datos directamente, sin estar *unidas* en una clase.

3.2.1.2 Módulo Ejercicios

- **Ejercicio:** La entidad *Ejercicio* corresponde con tres clases distintas: la clase abstracta *Ejercicio*, y las clases *Ejercicio_Tipo* y *Ejercicio_Implementación* que heredan de la primera. Esta separación existe porque es necesaria la separación entre lo que llamamos el ejercicio *genérico* y lo que llamamos una *ejecución concreta* de ese mismo ejercicio. Tomamos como ejemplo un ejercicio: *Flexiones comunes*. *Ejercicio_Tipo* se corresponde a la idea del ejercicio en sí, el ente abstracto —en el sentido más platónico de la palabra— mientras que *Ejercicio_Implementación* representa la realización de dicho ejercicio, es decir, una ejecución real del ejercicio, determinado por atributos como el número de repeticiones, el número de series, etc.

- **Categoría**: La clase *Categoría* tiene correspondencia directa con la entidad del mismo nombre. La clase *Categoría* interactúa con la clase *Ejercicio_Tipo*. La clase está formada por un nombre, descripción, una lista de *Ejercicio_Tipo* y los métodos para manipular dicha lista.

- **Rutina**: Esta clase tiene correspondencia directa con la entidad del mismo nombre. La clase *Rutina* interactúa con la clase *Ejercicio_Implementación*. La clase está formada por toda la información correspondiente a la rutina —Nombre, Descripción, Lista de Ejercicio...—, los métodos para manipular dicha rutina—*Añadir* y *Eliminar Ejercicio.* —, y los métodos para calcular el consumo energético de dicha rutina.

En la página siguiente se adjunta el diagrama de clases correspondiente al módulo *Ejercicios*.

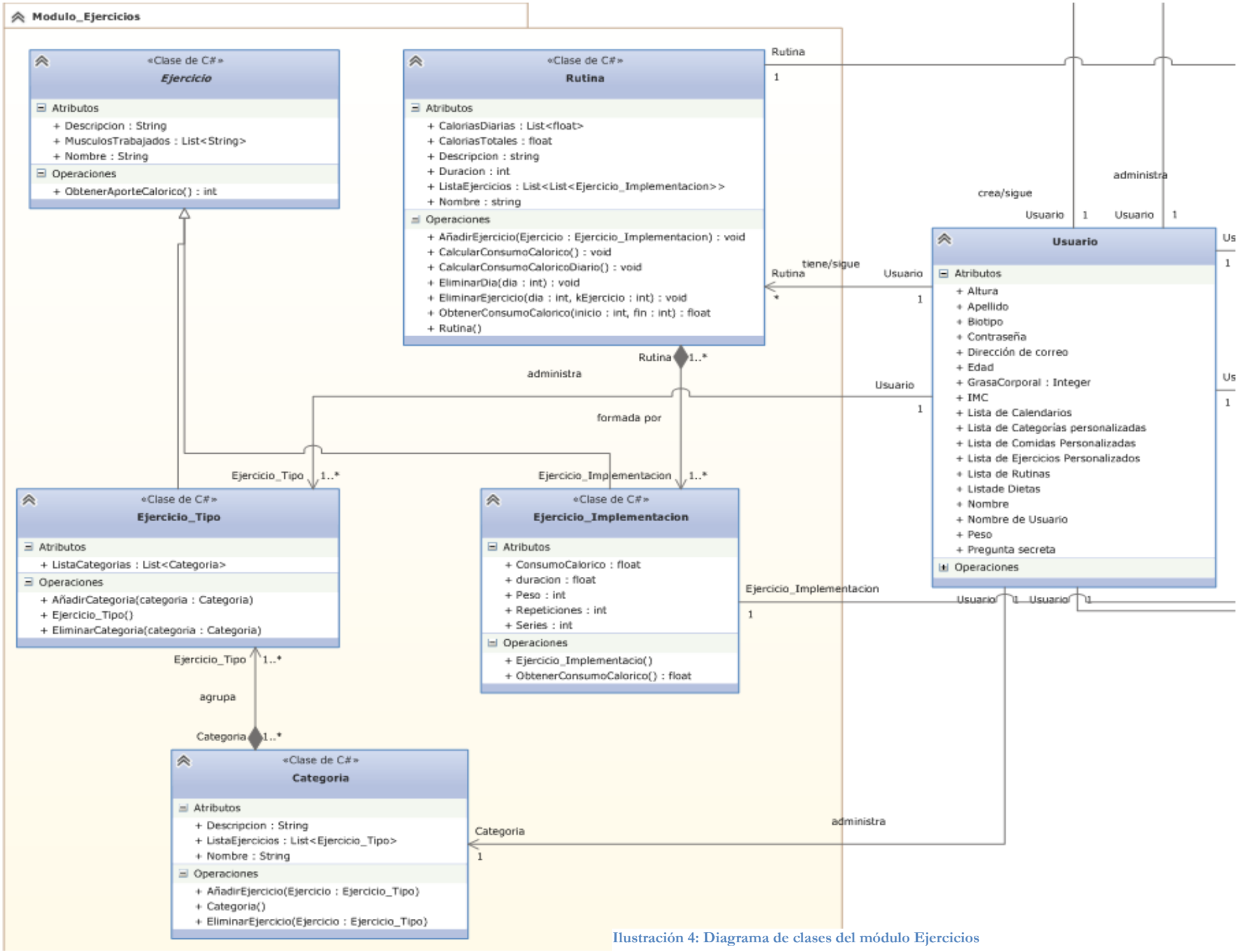


Ilustración 4: Diagrama de clases del módulo Ejercicios

3.2.1.3 Módulo Dietista

- **Alimento:** Esta clase tiene correspondencia directa con la entidad del mismo nombre. La clase está formada por la información correspondiente a un alimento —*Nombre, Descripción, Tipo, Cantidad, etc.*

- **Comida**: Esta clase tiene correspondencia directa con la entidad del mismo nombre. La clase está formada por el nombre, descripción y una lista de los alimentos que la forman, los métodos para manipular dicha lista y los métodos para calcular el aporte energético de la comida.

- **Dieta:** Esta clase tiene correspondencia directa con la entidad del mismo nombre. La clase está formada por toda la información correspondiente a la dieta que representa —*Nombre, descripción, Tabla de comidas, etc.* — y los métodos para manipular dicha dieta —*Añadir* y *Eliminar Comida,* etc.

En la página siguiente se adjunta el diagrama de clases correspondiente al módulo *Dietista.*

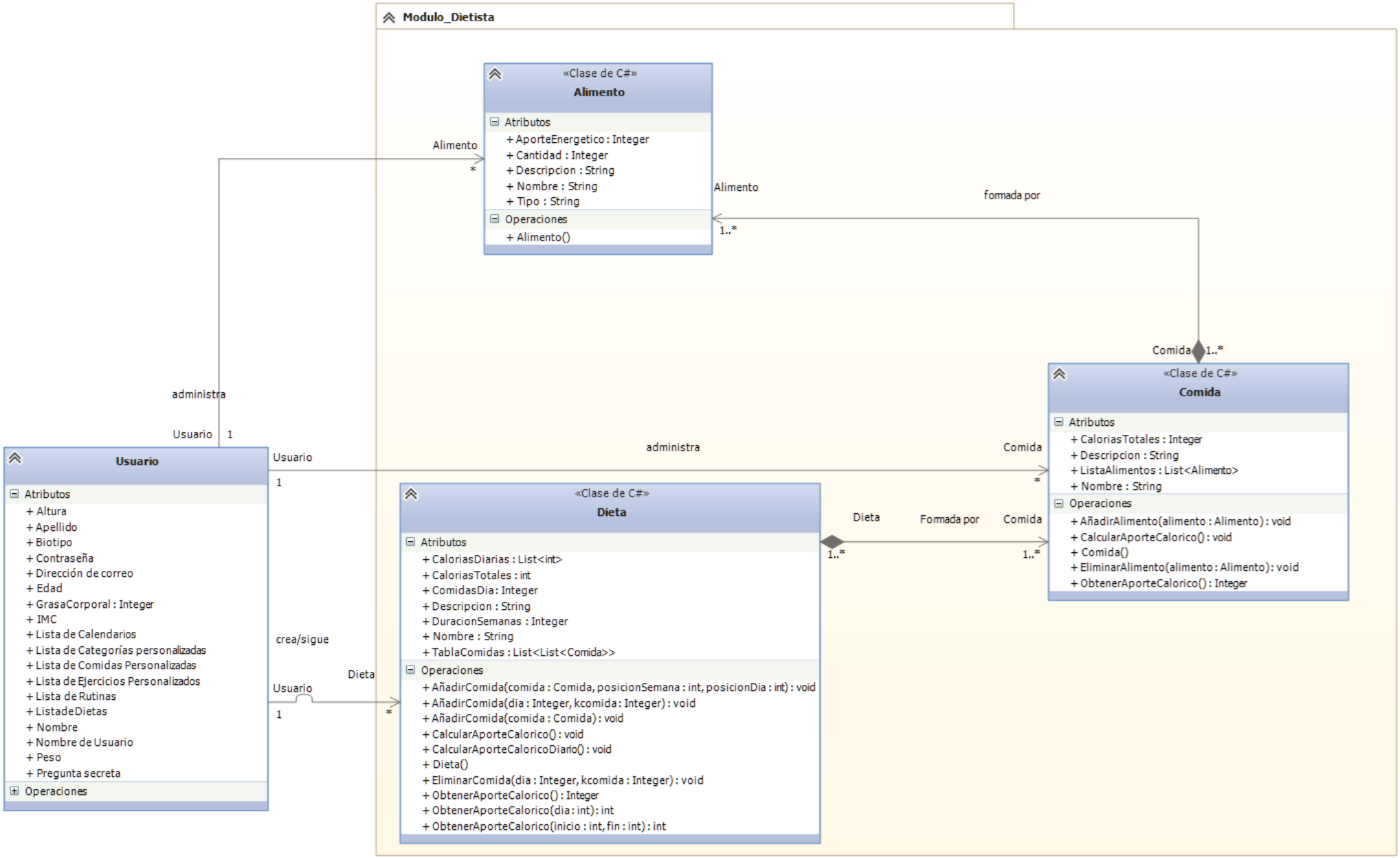


Ilustración 5: Diagrama de clases del módulo Dietista

3.2.1.4 Módulo Calendario

Para el módulo calendario se han utilizado varias clases pertenecientes a un proyecto de código libre llamado *Calendar.NET* [1] y modificado según las necesidades de la aplicación ya que las funcionalidades que permite dicho proyecto son muy limitadas y en ocasiones funcionaban de forma errónea.

De serie, el proyecto *Calendar.NET* ofrecía un *control* calendario que mostraba diferentes eventos y disponía de dos tipos de vista distintos: vista mensual y vista diaria. A partir de ahí hubieron de añadirse todo el resto de funcionalidades modificando las clases de dicho proyecto. Se ha añadido la posibilidad de añadir eventos mediante la interfaz —antes solamente se podían añadir mediante código—, que estos eventos sean de diferentes tipos, se han corregido multitud de errores en lo que se refiere al comportamiento de dicho *control*— errores a la hora de mostrar los eventos por pantalla, solapamiento de eventos, excepciones provocadas por malfuncionamiento a la hora de representar los meses por pantalla, malfuncionamiento en la frecuencia de repetición de los eventos, etc.—, se ha añadido la posibilidad de modificar y eliminar eventos,etc. —algunos de estos cambios se mencionan en el anexo de incidencias correspondiente al *Módulo Calendario*

133.

Para personalizar dicho *control* para las necesidades de nuestro proyecto hubo que editar exhaustivamente las clases que lo formaban: *Calendar, IEvent, CustomEvent, CustomRecurringFunctionAttribute, EventComparer, EventComparerDuracion* —nueva—, *EventTooltip y EventDetails.* Además se añadieron nuevas clases para los diferentes tipos de evento: *EventoDieta, EventoEjercicio* y *EventoRutina.*

- **Calendar:** Esta clase implementa la entidad del mismo nombre. La clase está formada por la información esencial del calendario —*fecha, lista de eventos, etc.* —, propiedades que controlan el funcionamiento interno de la clase y métodos tanto para manipular el calendario como para manipular las propiedades internas de la clase —*Añadir, modificar y eliminar eventos, cambiar vista, etc.*.

- **IEvent**: Esta clase es la interfaz que sirve de base para el resto de clases que representan los tipos de evento posibles —*CustomEvent*, *EventoDieta, EventoRutina* y *EventoEjercicio.* Fue necesario modificarla para añadir el campo *tipo*, que identifica numéricamente qué tipo de evento es el evento. Sus campos incluyen la duración, fecha del evento, la frecuencia de repetición, etc.

- **CustomEvent:** Esta clase implementa la interfaz *IEvent* y representa los eventos de tipo *normal* —no son ni ejercicios, ni rutinas, etc. — y los eventos de tipo *Comida.*

[1] http://www.codeproject.com/Articles/378900/Calendar-NET

- **EventoEjercicio:** Esta clase implementa también la interfaz *IEvent* y representa los eventos de tipo *Ejercicio*. Además de los atributos y métodos base esta implementación añade los campos específicos de un *Ejercicio_Implementación —Series, Repeticiones, Duración, Peso y ConsumoCalórico*.

- **EventoRutina:** Esta clase implementa la interfaz *IEvent* y representa los eventos de tipo *Rutina*. Además de los atributos y métodos base esta implementación incluye una lista de los eventos ejercicio que componen la rutina.

- **EventoDieta:** Análoga a la clase *EventoRutina* pero para los eventos de tipo *Dieta*.

- **EventComparer** *y* **EventComparerDuracion:** Ambas clases implementan la interfaz *IComparer*. La primera compara dos eventos según la hora a la que se produce el evento y la segunda los compara según la duración de dichos eventos.

- **EventTooltip:** Esta clase hereda de la clase *UserControl* y representa la etiqueta que aparece al pasar el cursor por encima de un evento. Sus atributos principales son el color de la etiqueta, color del texto, la fuente, los márgenes y el texto en sí.

- **EventDetails:** Esta clase hereda de la clase *Form*. Es la encargada de mostrar la información de los eventos por pantalla y guardar las modificaciones que el usuario realice en el evento.

En el diagrama de clases final se han omitido algunas clases del módulo calendario para evitar sobrecargar el diagrama. En el diagrama aparecen únicamente las clases principales de cada módulo.

En la página siguiente se encuentra adjunto el diagrama de clases del módulo C*alendario*.

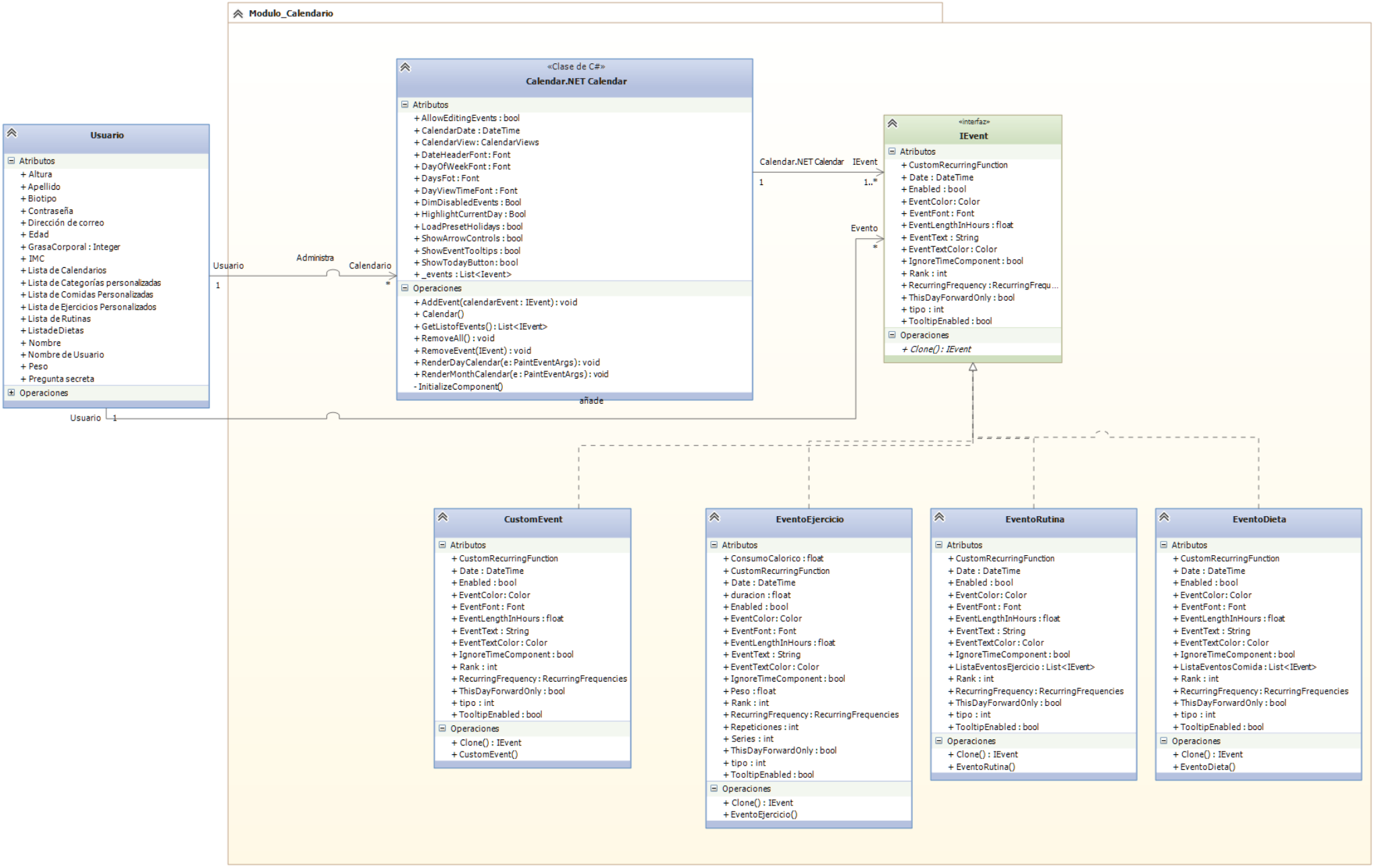


Ilustración 6: Diagrama de clases del módulo Calendario

3.3 Diseño de la base de datos

A la hora de diseñar la base de datos de la aplicación se tomaron en cuenta diferentes criterios y aspectos para valorar qué tipo de base de datos era más conveniente. Pese a que lo más habitual hubiera sido crear una base de datos relacional en SQL para aprovechar las ventajas que este tipo de base de datos ofrecen —una sola consulta puede rescatar mucha información, utilizan un lenguaje estandarizado, son seguras, *liberan* al usuario de ciertas responsabilidades de coherencia y control al manejar la información de la base de datos, una vez finalizadas están dotadas de una gran robustez…— se tomó la decisión de crear una base de datos de las denominadas *NoSQL*, concretamente una base de datos basada en documentos.

Por qué usar una base de datos de este tipo después de haber mencionado las bondades de una base de datos en SQL? Este tipo de bases de datos son muy amigables con el programador ya que ofrecen un margen de libertad muy amplio a la hora de decidir la estructura de cualquier aspecto de la base de datos —por ejemplo cómo separar cada campo de las muestras, la jerarquía de los documentos…—, su desarrollo es más rápido que el de las bases de dato de otro tipo y no es necesario ninguna herramienta ni software para implementarlas a parte de un editor de texto.

Por otro lado —y como consecuencia de la mencionada libertad que ofrecen— sus mayores inconvenientes son que la coherencia e integración de los datos depende totalmente del programador y que para que alguien externo pueda trabajar con ella debe de aprender exhaustivamente la estructura de dicha base de datos —este último aspecto menos relevante en nuestro caso ya que todo el proyecto lo realiza una sola persona.

Valorados estos aspectos se decidió dar preferencia a la libertad y sencillez para manipular la información frente a mayor seguridad y/o robustez.

3.3.1 Estructura

La estructura de la Base de datos está divida en dos elementos principales: directorios y ficheros.

Los ficheros almacenan información concreta de diferentes entidades —ejercicios, alimentos, comidas, usuarios...— y los directorios organizan dichos ficheros según el módulo y usuario al que pertenecen. Esta estructura queda reflejada en los diagramas siguientes:

3.3.1.1 Módulo General

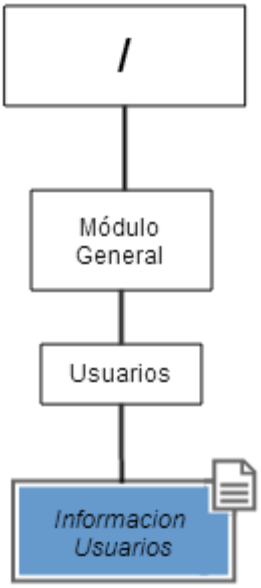


Ilustración 7: Estructura de la base de datos, Módulo General

3.3.1.2 Módulo Calendario

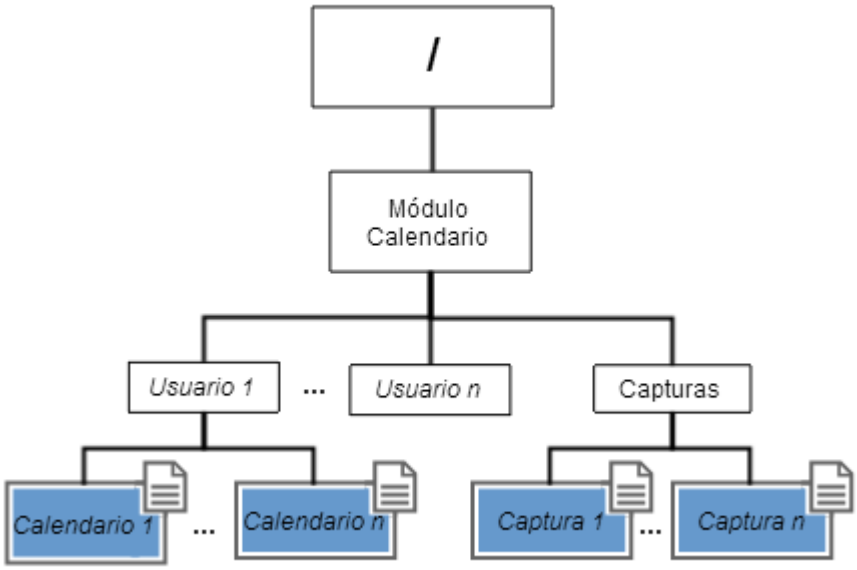


Ilustración 8: Estructura de la base de datos, Módulo Calendario

3.3.1.3 Módulo Ejercicios

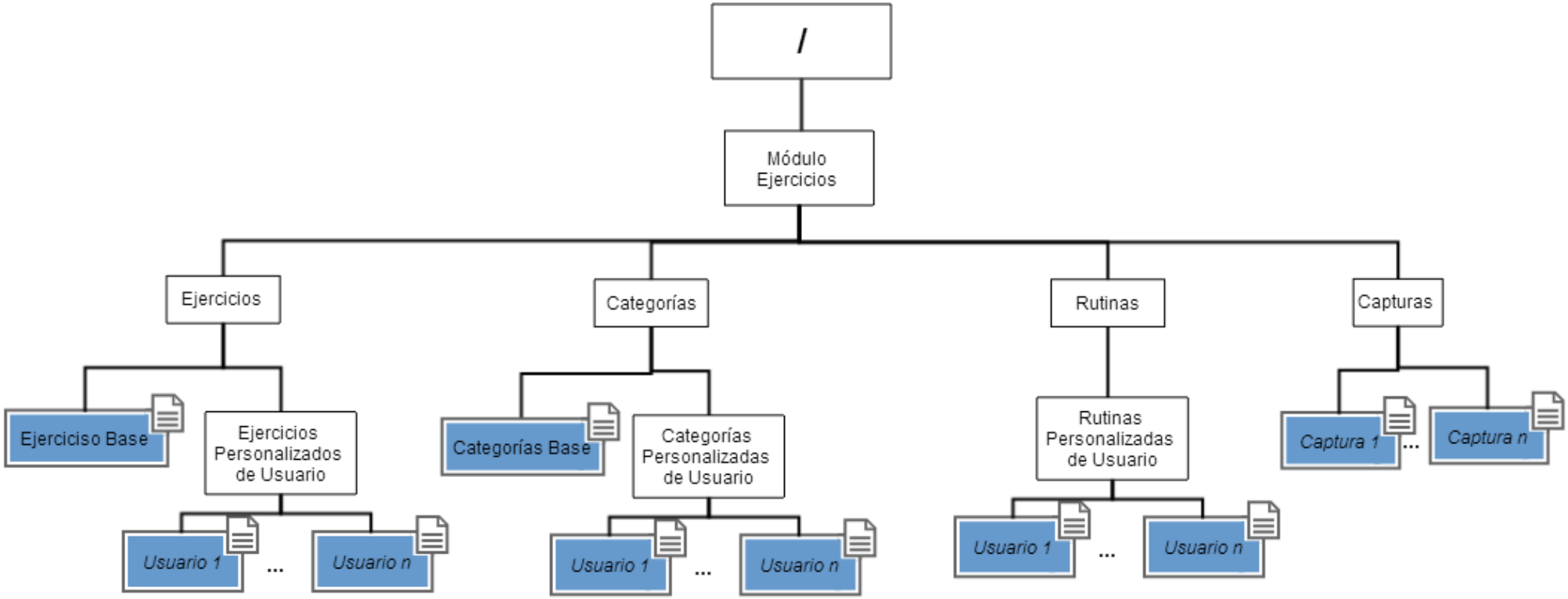


Ilustración 9: Estructura de la base de datos, Módulo Ejercicios

3.3.1.4 Módulo Dietista

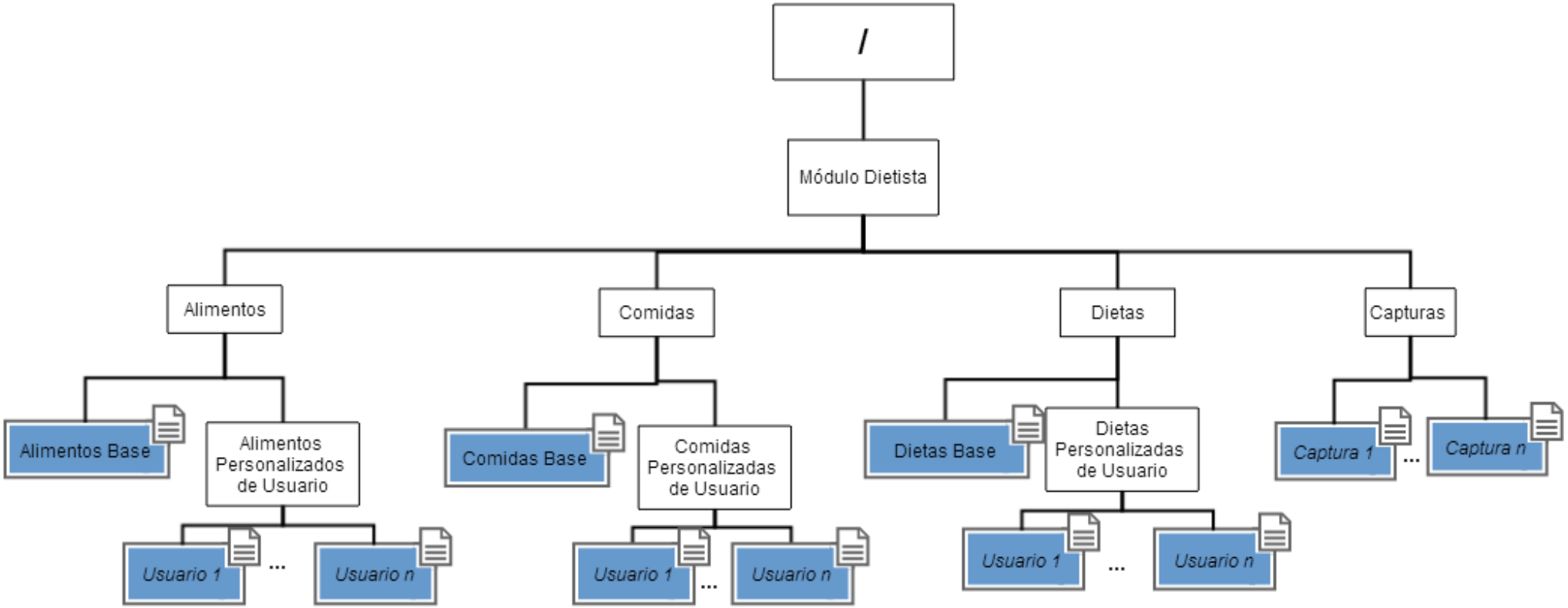


Ilustración 10: Estructura de la base de datos, Módulo Dietista

Una descripción más detallada de la estructura interna de los ficheros que forman la base de datos y su funcionamiento aparece en el punto 4 de esta memoria—*Implementación.*

3.4 Diseño de la Interfaz

El diseño de la interfaz se realizó primeramente a lápiz a mano alzada. Una vez realizado este diseño previo a mano se usó la herramienta web de diseño de diagramas *Creately*[2] para llevar a cabo el diseño que después se utilizaría para diseñar la implementación final de la interfaz gráfica de la aplicación.

A la hora de diseñar la interfaz se dio máxima prioridad a facilitar la interacción entre el usuario y la aplicación. Se buscó que la curva de aprendizaje de uso fuera lo menos *empinada* posible y que el uso de la aplicación fuera lo más intuitivo posible, incluso para usuarios que no estén habituados al uso del ordenador.

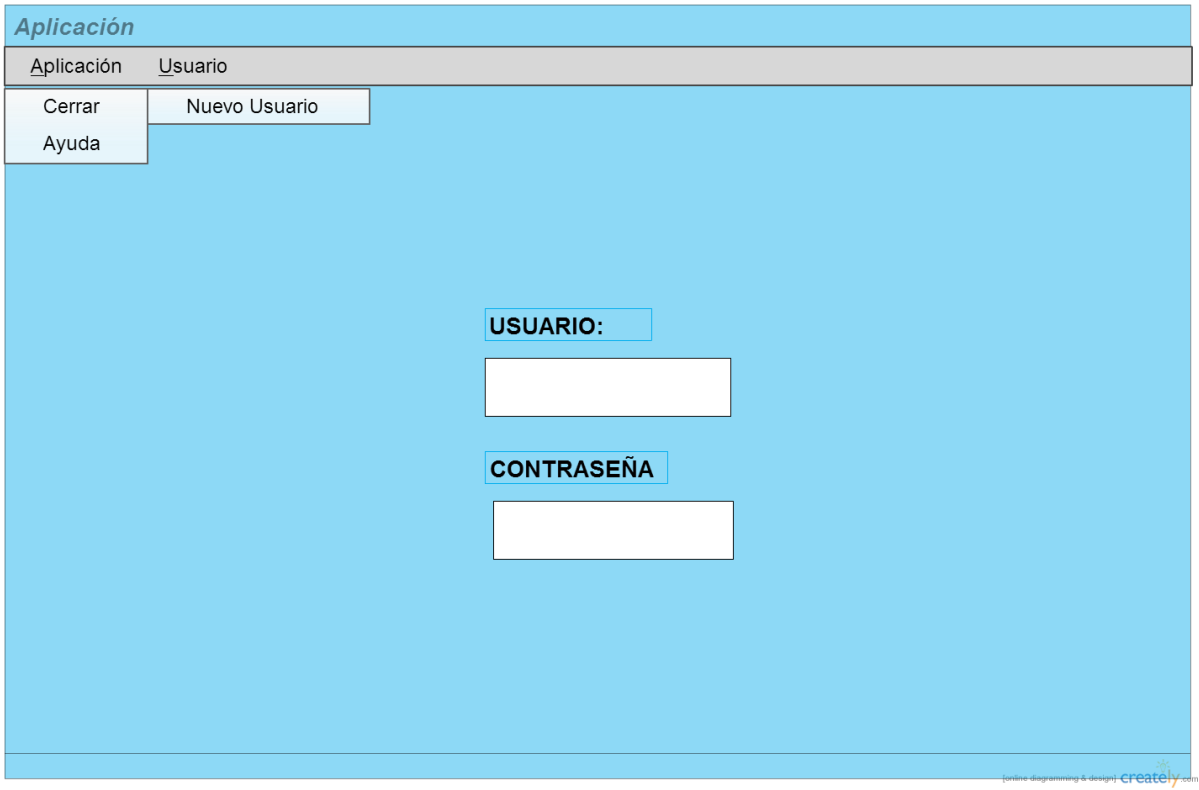


Ilustración 11: Diseño de la ventana principal del Módulo General, la ventana que aparece al ejecutar la aplicación.

Otro factor al que se le dio gran importancia fue intentar que la información fuera mostrada por pantalla de forma clara y sencilla, evitando recargar de información las ventanas y pestañas. Esto, junto con una paleta de colores *suave*, genera un ambiente en el que el usuario se siente relajado y cómodo al usar la aplicación.

En la imagen de la página siguiente se muestra un diseño de la ventana principal del módulo *Ejercicios* con varias pestañas abiertas.

[2] www.creately.com/app

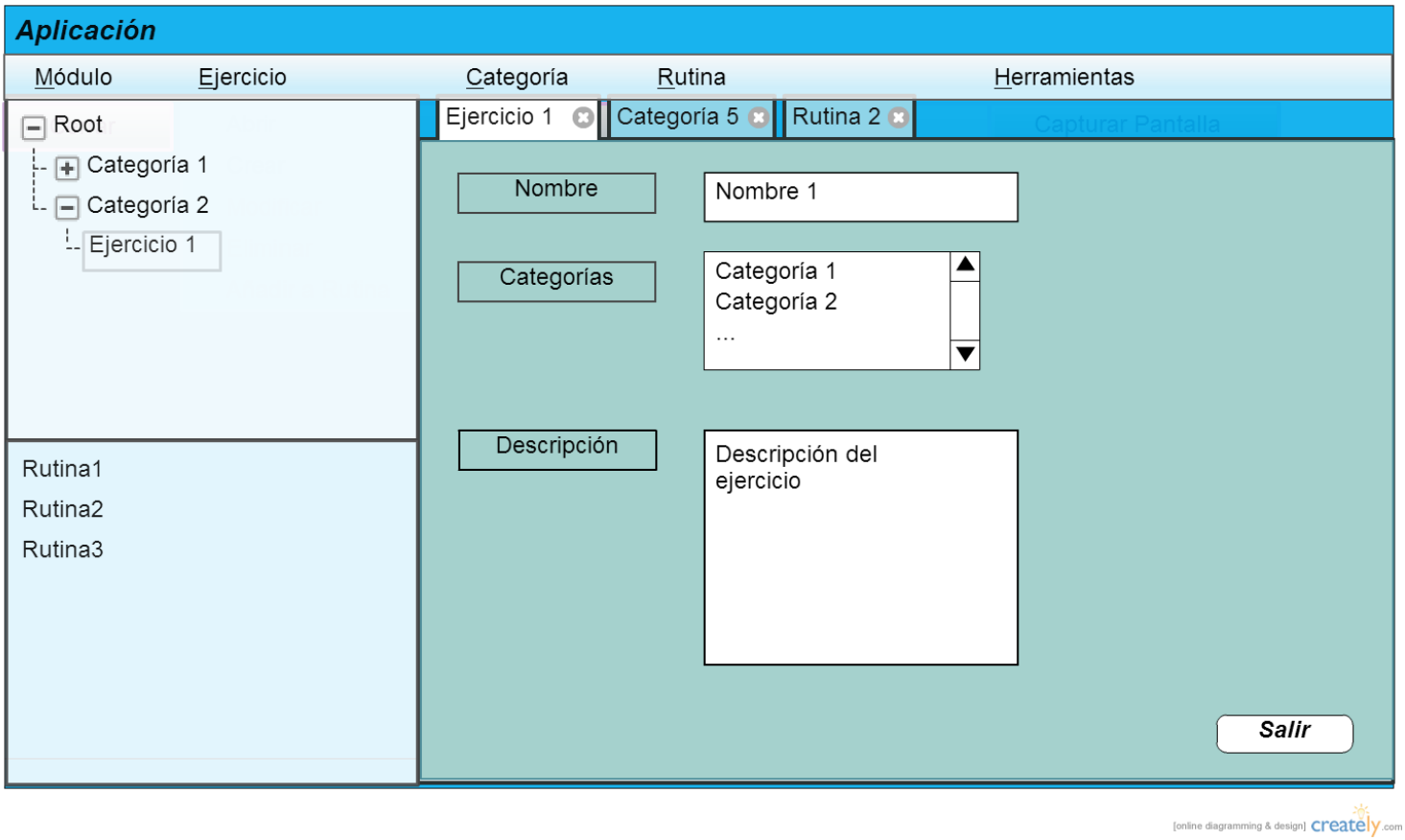


Ilustración 12: Diseño de la ventana principal del Módulo Ejercicios.

Intentando facilitar el uso de la aplicación se diseñó la interfaz de tal manera que los pasos para realizar las diferentes funcionalidades que ofrece la aplicación sean en la medida de lo posible secuenciales. Esto consigue que el usuario no se *pierda* a mitad de acción. Por ejemplo, un usuario que quiera añadir un nuevo evento, al seleccionar dicha opción, irá siguiendo un conjunto de ventanas emergentes que irán apareciendo una seguida de la otra hasta acabar con el evento añadido. El usuario irá tomando una decisión cada vez. De esta manera el usuario no podrá realizar ninguna otra acción mientras la que está haciendo en este momento no esté terminada. Así se evita, además, generar situaciones que pueden crear conflictos internos en la aplicación. Esta forma de proceder se repite en la mayoría de las funcionalidades de la aplicación. Aun y así existen algunas excepciones: un usuario puede abrir diferentes pestañas en un mismo módulo, para por ejemplo ver diferentes ejercicios a la vez, pero esto no implica realizar dos acciones a la vez, ya que mientras se ve el contenido de una pestaña no se puede interactuar con las otras.

La ilustración de la siguiente página representa los pasos a seguir por el usuario para añadir un evento.

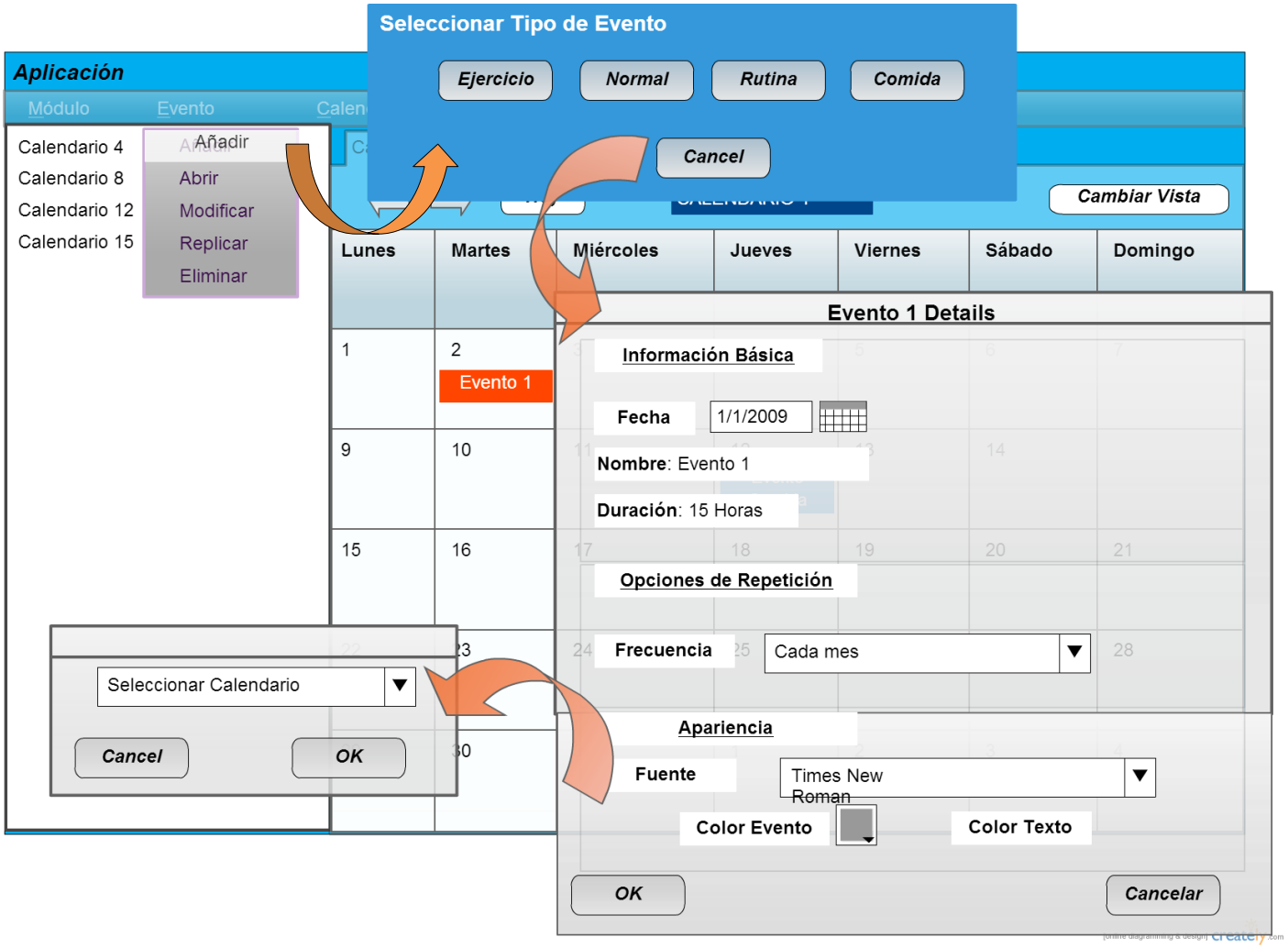


Ilustración 13: Diseño de los pasos a seguir por el usuario de la aplicación para añadir un nuevo evento.

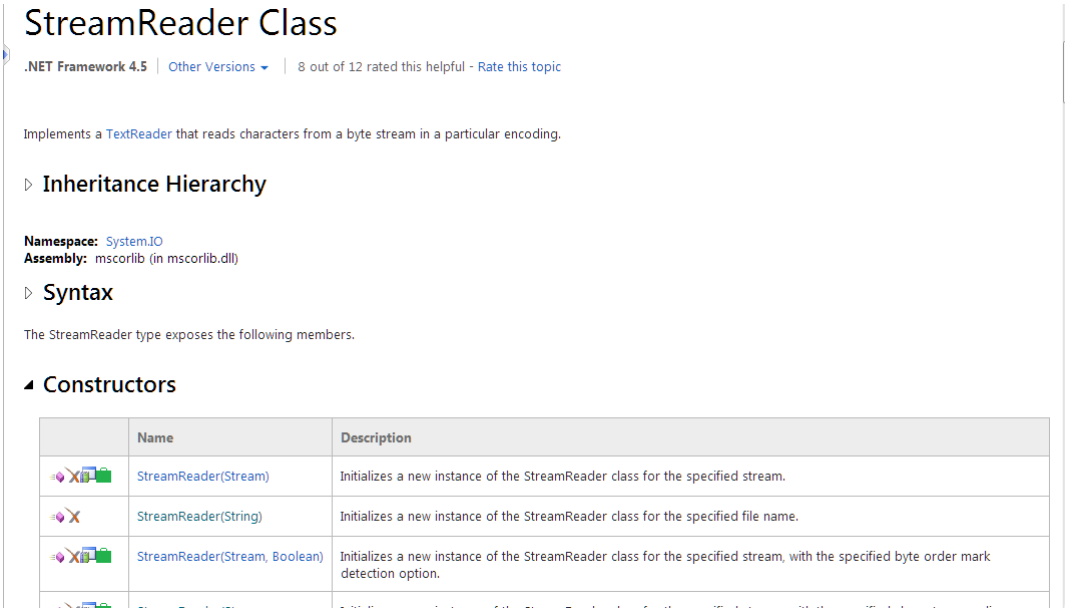
En el anexo *INTERFAZ (DISEÑO)* se han adjuntado otras ilustraciones del diseño de la interfaz gráfica de la aplicación —de los distintos módulos. No se ha incluido ninguna ilustración del módulo dietista porque su diseño es idéntico al del módulo *Ejercicios*.

Una vez finalizado el diseño pasamos a la siguiente fase de desarrollo de la aplicación: la *Implementación*.

4. Implementación y pruebas

4.1 Lenguaje utilizado y herramienta de desarrollo

El lenguaje de programación escogido ha sido C# ya que al ser un lenguaje de programación orientado a objetos facilita la conversión del diseño *en papel* a la estructura de clases que ofrece dicho lenguaje. Otro factor determinante es que Microsoft dispone de una librería de clases[3] (similar a la API de Java) que permite consultar con facilidad las características de las clases existentes, su funcionamiento, utilidad, etc. Además es un lenguaje derivado directamente de *C/C++*, un lenguaje muy utilizado durante toda la carrera y por lo tanto permite un trabajo más eficiente.


StreamReader Class

.NET Framework 4.5 | Other Versions | 8 out of 12 rated this helpful - Rate this topic

Implements a TextReader that reads characters from a byte stream in a particular encoding.

Inheritance Hierarchy

Namespace: System.IO
Assembly: mscorlib (in mscorlib.dll)

Syntax

The StreamReader type exposes the following members.

Constructors

	Name	Description
	StreamReader(Stream)	Initializes a new instance of the StreamReader class for the specified stream.
	StreamReader(String)	Initializes a new instance of the StreamReader class for the specified file name.
	StreamReader(Stream, Boolean)	Initializes a new instance of the StreamReader class for the specified stream, with the specified byte order mark detection option.

Ilustración 14: Descripción de la clase *StreamReader* en la librería de clases de Microsoft.

En el punto **1.5** de esta misma memoria—

[3] http://msdn.microsoft.com/en-us/library

Herramientas utilizadas— ya se menciona que la herramienta utilizada para desarrollar la aplicación ha sido *Microsoft Visual Studio 2010*. Como se comenta en dicho punto, se eligió esta herramienta porque agrupaba todas las necesidades que a nivel de desarrollo requería nuestra aplicación. En una misma *Solución* —como *Visual Studio* lo llama— están contenidos todos los elementos que implementan la aplicación, desde la implementación del sistema de clases, los juegos de pruebas de este, la implementación de la interfaz e incluso los diagramas UML que sean necesarios. A cada uno de estos *Visual Studio* los denomina *proyectos*.

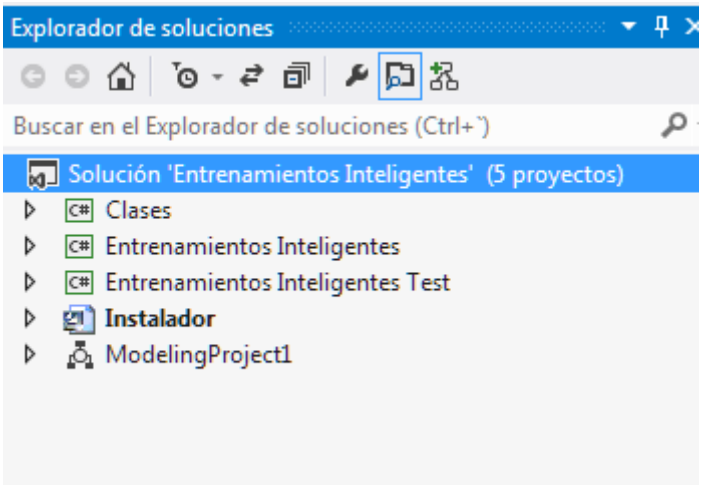


Ilustración 15: Explorador de Soluciones de Visual Studio 2010

El único punto negativo, si es que se le puede considerar de tal manera, es que esta herramienta no permite el uso de ingeniería inversa en algunos casos —como por ejemplo crear la representación en UML de una clase o conjunto de clases una vez estas están terminadas.

4.2 Implementación de la aplicación.

4.2.1 Sistema de clases

La implementación del sistema de clases que forman el núcleo de la aplicación es una traducción directa de lo representado en el diagrama de clases del apartado *Diseño* de la memoria. A partir de la descripción de los atributos y métodos de cada clase —definidos en dicho apartado— estos se fueron implementando y testeando clase a clase.

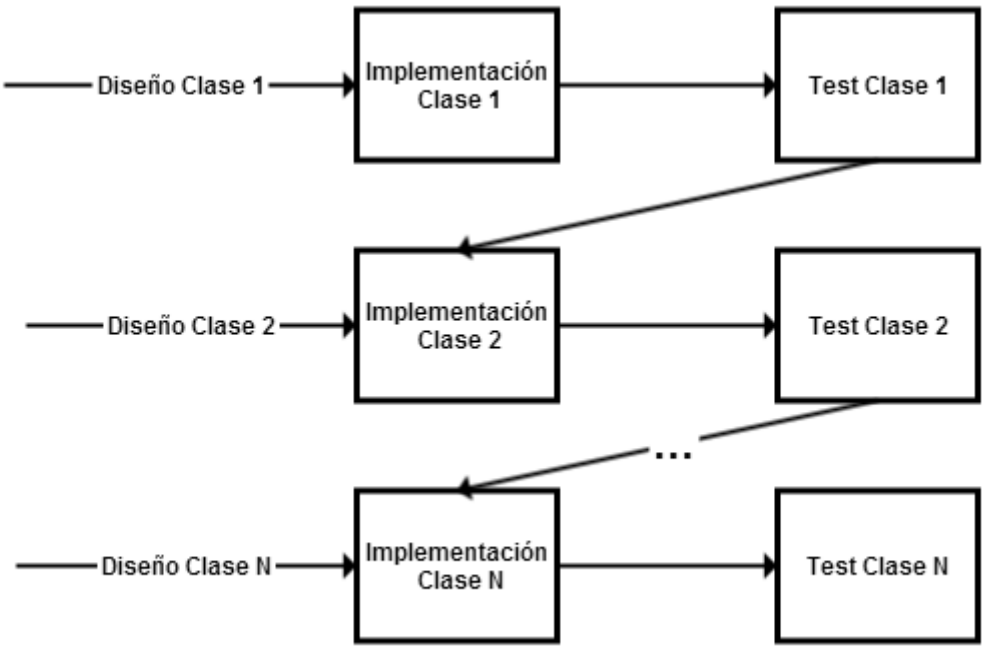


Ilustración 16: Diagrama del procedimiento de implementación de las clases

Este método de proceder no implica que a cada clase le corresponda una clase de prueba. Las clases de test se dividen por módulos y las clases que pertenecen a ese módulo se testean en diferentes métodos de dicha clase. Por ejemplo: El correcto funcionamiento de las clases del Módulo *Ejercicios* —*Ejercicio_tipo, Ejercicio_Implementación, Comida* y *Dieta*— se comprueba en la clase de pruebas —*TestClass*— *ModuloEjerciciosTest*. Esta clase incluye los métodos *CrearEjercicio(), AdministrarMúsculosTrabajados(), AdministrarRutina()*... entre otros donde se comprueba el funcionamiento de las clases anteriormente mencionadas.

```
namespace Modulo_Dietista
{
    using Modulo_Calendario;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    [Serializable]
    public class Comida
    {
        public Comida()
        {
            this.Nombre = "No especificado";
            this.Descripcion = "No especificada";
            this.ListaAlimentos = new List<Alimento>();

        }
        public Comida(string nombre, string descripcion, List<Alimento> listaAlimentos)
        {
            this.Nombre = nombre;
            this.Descripcion = descripcion;
            this.ListaAlimentos = listaAlimentos;
            this.CaloriasTotales = this.ListaAlimentos.Sum(item => item.AporteEnergetico);
        }
        public virtual string Nombre
        {
            get;
            set;
        }

        public virtual string Descripcion
        {
            get;
```

100 %

Entrenamientos_Inteligentes_Test.ModuloDietistaTest — CreacionDeAlimento()

```
        public void CrearComidaVacia()
        {

            Comida PatatasJudias = new Comida();

            Assert.AreEqual(PatatasJudias.Nombre, "No especificado");
            Assert.AreEqual(PatatasJudias.Descripcion, "No especificada");
            Assert.AreEqual(PatatasJudias.CaloriasTotales, 0);
            Assert.AreEqual(PatatasJudias.ListaAlimentos.Count, 0); //lisa vacía

        }

        [TestMethod]
        public void CrearComida()
        {

            string Nombre = "Patatas Cocidas y Judía Verde";
            string Descripcion = "Muchos hidratos, saludable";
            List<Alimento> ListaAlimentos = new List<Alimento>();

            Alimento Alimento1 = new Alimento("Alimento 1", "No especificado", "No especificado",
            Alimento Alimento2 = new Alimento("Alimento 2", "No especificado", "No especificado",

            ListaAlimentos.Add(Alimento1);
            ListaAlimentos.Add(Alimento2);

            Comida PatatasJudias = new Comida(Nombre, Descripcion, ListaAlimentos);

            Assert.AreEqual(PatatasJudias.Nombre, "Patatas Cocidas y Judía Verde");
            Assert.AreEqual(PatatasJudias.Descripcion, "Muchos hidratos, saludable");
            Assert.AreEqual(PatatasJudias.CaloriasTotales, 750);
```

Ilustración 17: Arriba, captura de la implementación de los constructores de la clase *Comida*; Abajo, captura de la implementación de los métodos de la clase de pruebas que comprueban el correcto funcionamiento de dichos constructores.

4.2.2 Implementación de la base de datos.

En el apartado de *Diseño* ya se explicó el diseño de la base de datos —basada en ficheros— y los motivos para haber elegido dicho diseño y estructura así que en este apartado nos centraremos únicamente en los aspectos que hacen referencia a su implementación en la aplicación.

Estructura de los ficheros

La estructura principal de los ficheros de la base de datos es compartida por todos ellos: cada línea del fichero representa un elemento —ya sea un usuario, una rutina, un ejercicio, una dieta, etc. Cada línea, a su vez, está dividida en campos por diferentes tipos de separador según si los campos contienen más campos dentro de sí o no. Esta explicación se ve más clara con un ejemplo:

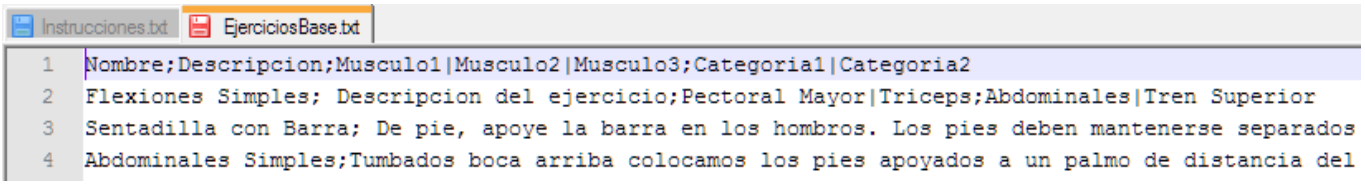


Ilustración 18: Captura del fichero de la base de datos EjerciciosBase.

En la línea número **1** encontramos la leyenda, que marca el patrón que siguen todas las líneas. Este patrón lo utiliza la aplicación para saber cuándo hay un nuevo campo y qué es cada uno de ellos. En este caso observamos que existen dos separadores: los caracteres **;** y **|**. El separador **;** es un separador de primer nivel, mientras que el separador **|** lo es de segundo nivel. Siguiendo esta lógica, en la línea número **2** encontramos la información referente al siguiente ejercicio:

- **Nombre:** Flexiones Simples.
- **Descripción:** Descripción del ejercicio.
- **Músculos trabajados:** Pectoral Mayor, Tríceps.
- **Categorías:** Abdominales, Tren Superior.

Y lo mismo sucede con el resto de líneas que forman el fichero.

Existen 6 niveles de separadores. Este es el menor número posible para evitar problemas a la hora de separar los campos correctamente. Los separadores son:

1r Nivel	;
2º Nivel	|
3r Nivel	#
4º Nivel Módulo Calendario Resto de Módulos	} *Para evitar conflictos con el formato de fecha dd/mm/aa* /
5º Nivel	{
6º Nivel	[

Tabla 1: Separadores de campo

Se han seleccionado estos separadores pensando en caracteres que no fueran a ser utilizados, caracteres cuya omisión no dificultaran el uso de la aplicación. Para evitar problemas, se avisa al usuario de que no use ninguno de los caracteres de la tabla bajo ningún concepto.

Esta estructura es común para todos los ficheros de la base de datos a excepción del módulo *Calendario*. A diferencia del resto de ficheros, donde cada línea representa un elemento, en el caso de los calendarios cada uno de ellos está representado por un fichero, con un par de peculiaridades. La primera peculiaridad es la necesidad de un separador extra para evitar problemas a la hora de guardar fechas —usar el separador / provocaría fallos a la hora de guardar la información. La segunda es en la estructura en sí. Las dos primeras líneas de cada fichero calendario están reservadas para la palabra clave *FechaCalendario* y la fecha en sí. La tercera línea marca la leyenda de cómo se representan los eventos que son contenidos en el calendario.

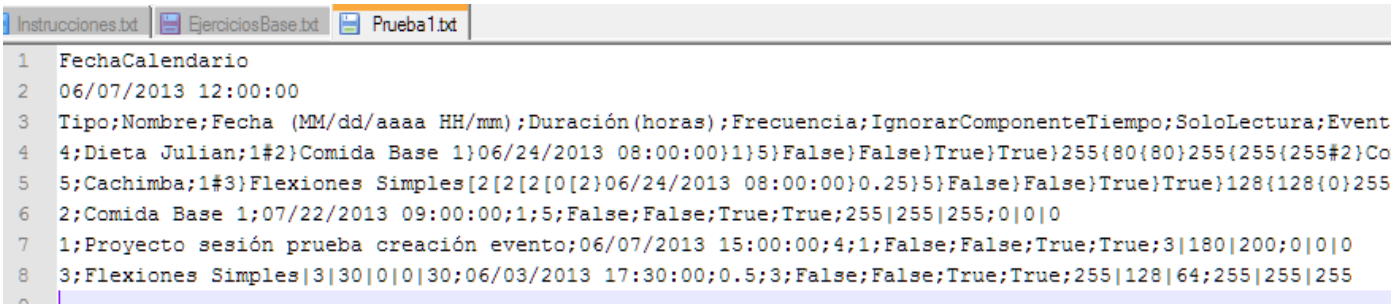


Ilustración 19: Captura de un fichero que codifica un calendario.

De acuerdo con la descripción anterior, viendo la *Ilustración 19* extraemos la información siguiente:

- **Nombre del calendario:** Prueba1 —El nombre del archivo.
- **Fecha del calendario:** 07/06/2013, 12:00:00 —fecha en formato 24h— Línea **2**.
- **Numero de eventos:** 5 —El número de líneas del archivo quitando las 3 primeras.

Y siguiendo la leyenda que marca la línea **3** podemos extraer la información correspondiente a cada evento del calendario, en este caso cinco eventos —uno de cada tipo posible.

La codificación de todos los ficheros es la misma, UTF-8. Esto es importante tenerlo en cuenta siempre que se quiera editar la base de datos directamente sobre los ficheros ya que por defecto —habitualmente— los ficheros de texto son creados bajo la codificación *ANSI*. Ignorar esta circunstancia provocaría fallos en el funcionamiento ya que al leer el archivo, la aplicación recibe la orden específica de que los ficheros están codificados según la representación UTF-8.

Interacción con la aplicación

La interacción de la base de datos con la aplicación es relativamente sencilla: cada vez que se carga un módulo —sea cual sea— se carga de los ficheros de la base de datos toda la información correspondiente a dicho módulo a unas estructuras de listas internas. A partir de este momento se utilizarán siempre dichas listas internas hasta que se cierre el módulo —o se elija la opción de guardar cambios si el módulo lo permite— que se guardarán todos los cambios en los archivos de nuevo. Con esto lo que conseguimos es no tener que acceder constantemente a la base de datos. Además, como el volumen de información a tratar no es muy elevado incluso leyendo unos centenares de líneas de cada fichero, el rendimiento de la aplicación no se ve afectado.

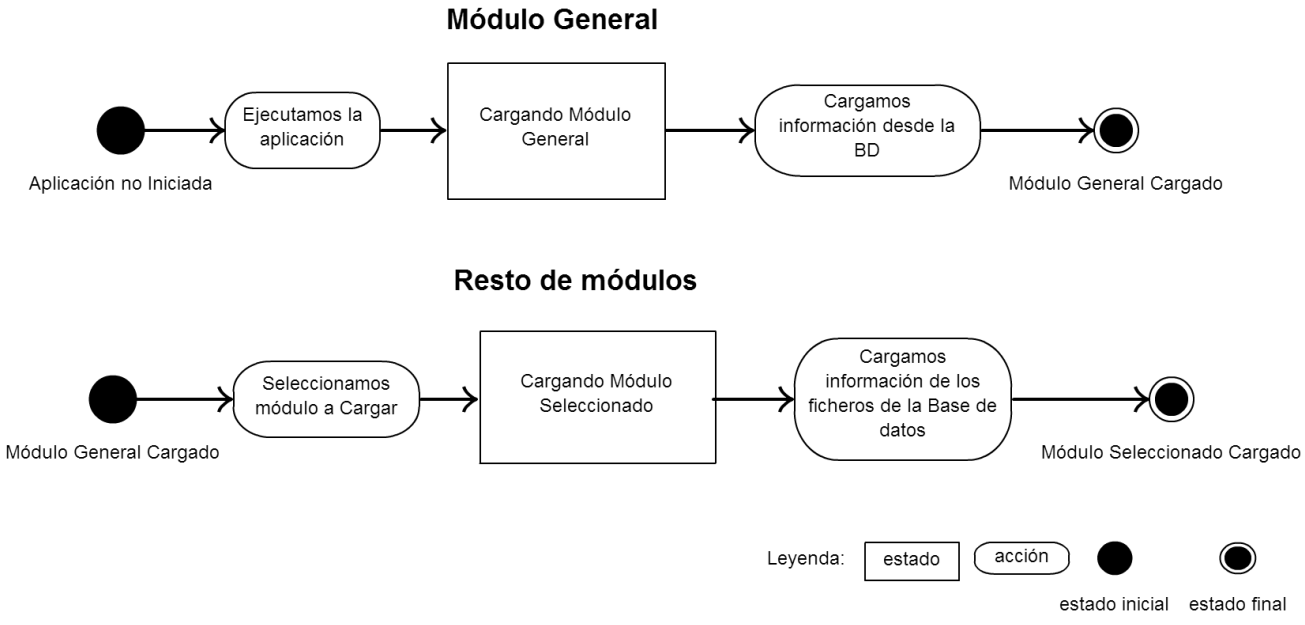


Ilustración 20: Diagrama de estados de la carga de información de la base de datos.

A continuación se mostrará el proceso descrito en el diagrama mediante capturas de pantalla del código fuente. Cada captura incluye una descripción del código para ir siguiendo el proceso de carga. Este proceso es análogo en todos los módulos, es por esto que únicamente se muestran capturas del módulo ejercicios.

```
65  //Cargar el nombre y la descripción de las categorías de un archivo.
66
67  CargarBaseDatosCategoriasBase();
68  CargarBaseDatosCategoriasPersonalizadas();
```

Ilustración 21: Las funciones *CargarBaseDatosCategoriasBase()* y *CargarBaseDatosCategoriasPersonalizadas()* leen de los ficheros *CategoriasBase.txt* y *<nombreusuario>.txt* la información referente a las categorías base y las específicas de cada usuario respectivamente.

```
70  //A continuación cargaremos los ejercicios base, por defecto de la aplicación.
71  System.IO.StreamReader ArchivoEjerciciosBase = new System.IO.StreamReader("Base de datos/Módulo Ejercicios/Ejercicios/EjerciciosBase.txt");
72  //Leemos la primera línea que es la de "información" de los campos, se desecha: Nombre;Descripcion;Musculo1|Musculo2|Musculo3;Categoria1|Cat
73  ArchivoEjerciciosBase.ReadLine();
74
75  while (!ArchivoEjerciciosBase.EndOfStream)
76  {
77      linea = ArchivoEjerciciosBase.ReadLine();
78      campos = linea.Split(';');
79
80      //List<String> ListaMusculos = new List<String>(campos[2].Split('|'));
81      List<String> ListaNombreCategorias = new List<String>(campos[3].Split('|'));
82      List<Categoria> ListaCategoriasEjercicio = new List<Categoria>(CrearListaCategorias(ListaNombreCategorias));
83
84      //Ejercicio_Tipo Ejercicio = new Ejercicio_Tipo(campos[0], campos[1], new List<String>(campos[2].Split('|')), ListaCategoriasEjercicio);
85      Ejercicio_Tipo Ejercicio = new Ejercicio_Tipo();
86      Ejercicio.Nombre = campos[0];
87      Ejercicio.Descripcion = campos[1];
88      Ejercicio.MusculosTrabajados = new List<String>(campos[2].Split('|'));
89
90      foreach (Categoria cat in ListaCategoriasEjercicio)
91      {
92          if (!ListaCategoriasBase.Any(categoria => categoria.Nombre == cat.Nombre))
93          {
94              cat.AñadirEjercicio(Ejercicio);
95              ListaCategoriasBase.Add(cat);
96              Ejercicio.AñadirCategoria(ListaCategoriasBase.Find(item => item.Nombre == cat.Nombre));
97          }
98          else
99          {
100             Ejercicio.ListaCategorias.Add(ListaCategoriasBase.Find(item => item.Nombre == cat.Nombre));
101             ListaCategoriasBase.Find(item => item.Nombre == cat.Nombre).AñadirEjercicio(Ejercicio);
102         }
103
104     }
105
106     ListaEjerciciosBase.Add(Ejercicio);
```

Ilustración 22: Lectura de los ejercicios base del fichero de la base de datos.

La Ilustración 22 describe el proceso de lectura de los *Ejercicios* Base del archivo *EjerciciosBase.txt* de la base de datos:

Se leen línea a línea cada uno de los ejercicios predefinidos. La primera línea se desecha ya que es la leyenda que describe la información de los campos del fichero. Mientras haya

líneas —ejercicios— que leer se divide la línea en campos —línea **78**— y se guardan los campos del ejercicio en cuestión en la instancia de la clase *Ejercicio_Tipo* que hemos creado—líneas **86**, **87**, **88**. En caso de que el campo contenga más campos dentro —como por ejemplo la lista de categorías— es dividido de nuevo —línea **81**— y se recorre la lista previamente creada de categorías —— buscando las categorías que aparecen en la lista de categorías del ejercicio que estamos leyendo —*ListaCategoriasEjercicio*. Cuando encontramos una coincidencia añadimos la categoría a la lista de categorías de la instancia que hemos creado anteriormente —línea **85**. Una vez finalizado este proceso se cierra el descriptor de fichero —línea **109**— y se repite el mismo proceso con el fichero de ejercicios personalizados del usuario en cuestión.

```
111 //ListaCategoriasBase.Add(new Categoria("Sin Categoría","Ejercicios que no pertenecen a ninguna categoría",new List<Ejercicio_Tipo>()));   La ca
112
113 //Ahora cargamos los ejercicios personalizados del usuario en cuestión
114 System.IO.StreamReader ArchivoEjerciciosPersonalizados = new System.IO.StreamReader("Base de datos/Módulo Ejercicios/Ejercicios/Ejercicios Tipo
115 //la primera linea corresponde de nuevo a la leyenda
116 ArchivoEjerciciosPersonalizados.ReadLine();
117
118 while (!ArchivoEjerciciosPersonalizados.EndOfStream)
119 {
120     linea = ArchivoEjerciciosPersonalizados.ReadLine();
121     campos = linea.Split(';');
122
123     List<String> ListaMusculos = new List<String>(campos[2].Split('|'));
124     List<String> ListaNombreCategorias = new List<String>(campos[3].Split('|'));
125     List<Categoria> ListaCategoriasEjercicio = new List<Categoria>(CrearListaCategorias(ListaNombreCategorias));
126
127     //Ejercicio_Tipo Ejercicio = new Ejercicio_Tipo(campos[0], campos[1], new List<String>(campos[2].Split('|')), ListaCategoriasEjercicio);
128
129     Ejercicio_Tipo Ejercicio = new Ejercicio_Tipo();
130     Ejercicio.Nombre = campos[0];
131     Ejercicio.Descripcion = campos[1];
132     Ejercicio.MusculosTrabajados = new List<String>(campos[2].Split('|'));
133
134     foreach (Categoria cat in ListaCategoriasEjercicio)
135     {
136         if (!ListaCategoriasBase.Any(categoria => categoria.Nombre == cat.Nombre))
137         {
138             if (!ListaCategoriasPersonalizada.Any(categoria => categoria.Nombre == cat.Nombre))
139             {
140                 cat.AñadirEjercicio(Ejercicio);
141                 ListaCategoriasPersonalizada.Add(cat);
142                 Ejercicio.AñadirCategoria(ListaCategoriasPersonalizada.Find(item => item.Nombre == cat.Nombre));
143             }
144             else
145             {
146                 Ejercicio.ListaCategorias.Add(ListaCategoriasPersonalizada.Find(item => item.Nombre == cat.Nombre));
147                 ListaCategoriasPersonalizada.Find(item => item.Nombre == cat.Nombre).AñadirEjercicio(Ejercicio);
148             }
149         }
150         else
151         {
152             Ejercicio.ListaCategorias.Add(ListaCategoriasBase.Find(item => item.Nombre == cat.Nombre));
153             ListaCategoriasBase.Find(item => item.Nombre == cat.Nombre).AñadirEjercicio(Ejercicio);
154         }
155
156     }
157
158     this.ListaEjerciciosPersonalizada.Add(Ejercicio);
159
160 }
161 ArchivoEjerciciosPersonalizados.Close();
```

Ilustración 23: Proceso para leer los ejercicios personalizados de usuario del fichero de la base de datos.

La Ilustración 23 describe el mismo proceso que la ilustración 21 pero para los Ejercicios personalizados del usuario en cuestión.

Una vez hecho esto se genera el árbol correspondiente a las categorías y ejercicios.

```
163  //Entonces generamos el árbol de categorías y ejercicios
164  GenerarArbolCategoriasYEjercicios();
```

Ilustración 24: *GenerarArbolCategoriasYEjercicios()* se encarga de crear y rellenar el arbol que mostrará por pantalla las categorías y ejercicios cargados.

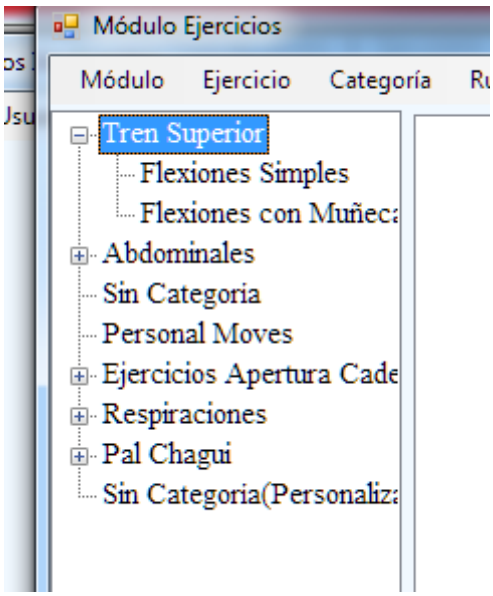


Ilustración 25: Captura de pantalla de la aplicación en ejecución donde se muestra el árbol con las categorías y ejercicios.

Una vez generado el árbol de categorías y ejercicios la aplicación carga de la base de datos las rutinas del usuario y se genera el árbol de rutinas correspondiente.

```
168  System.IO.StreamReader ArchivoRutinas = new System.IO.StreamReader("Base de datos/Módulo Ejercicios/Rutinas/Rutinas Personalizadas de Usuario/" + this.Usuar
169
170  String[] CamposPrincipales;
171
172  //La primera línea es la leyenda, se lee pero no se utiliza
173  //Separadores de campo:
174  //1r nivel:  ;      Separa los diferentes campos principales de la rutina
175  //2º nivel:  |      Cada día está separado por uno de estos
176  //3r nivel:  #      Separa los diferentes campos de cada día (día y ejercicios del día)
177  //4º nivel:  /      Separa los diferentes campos de la implementación del ejercicio (nombre, series, repeticones, etc)
178
179  //Leyenda: Nombre;Descripción;Día1#NombreEjercicio1/Series/Repeticiones/Peso/Duracion/ConsumoCalorico#NombreEjercicio2/Series/Repeticiones/Peso/Duracion/Co
180
```

Ilustración 26: Captura de los separadores de campo del fichero de la base de datos que contiene las rutinas del usuario.

En la Ilustración 26 se observa la leyenda de separadores del fichero que contiene las rutinas.

```
184 while (!ArchivoRutinas.EndOfStream)
185 {
186     linea = ArchivoRutinas.ReadLine();
187     CamposPrincipales = linea.Split(';');
188 
189     Rutina rutinaLeida = new Rutina();
190 
191     rutinaLeida.Nombre = CamposPrincipales[0];
192     rutinaLeida.Descripcion = CamposPrincipales[1];
193 
194     string[] CamposListaDias = CamposPrincipales[2].Split('|');
195 
196     foreach(string dia in CamposListaDias)
197     {
198         string[] CamposDia = dia.Split('#');
199 
200         for(int i = 1; i<CamposDia.Count(); i++) //el primer campo es el índice numérico del día, el resto, los ejercicios
201         {
202             string[] CamposEjercicio = CamposDia[i].Split('/');
203             Ejercicio_Tipo EjercicioTipo;
204             EjercicioTipo = ListaEjerciciosBase.Find(item => item.Nombre == CamposEjercicio[0]);
205             if (EjercicioTipo == null) EjercicioTipo = ListaEjerciciosPersonalizada.Find(item => item.Nombre == CamposEjercicio[0]);
206             //Si EjercicioTipo == null significa que el ejercicio no es de la lista base, sino de la lista personalziada de ejercicios
207 
208             Ejercicio_Implementacion ImplementacionConcreta = new Ejercicio_Implementacion();
209             ImplementacionConcreta.Nombre = EjercicioTipo.Nombre;
210             ImplementacionConcreta.Descripcion = EjercicioTipo.Descripcion;
211             ImplementacionConcreta.MusculosTrabajados = EjercicioTipo.MusculosTrabajados;
212             if (CamposEjercicio[1] != "") ImplementacionConcreta.Series = Convert.ToInt32(CamposEjercicio[1]);
213             else ImplementacionConcreta.Series = -1;
214             if (CamposEjercicio[2] != "") ImplementacionConcreta.Repeticiones = Convert.ToInt32(CamposEjercicio[2]);
215             else ImplementacionConcreta.Repeticiones = -1;
216             if (CamposEjercicio[3] != "") ImplementacionConcreta.Peso = Convert.ToSingle(CamposEjercicio[3], System.Globalization.Cultur
217             else ImplementacionConcreta.Peso = -1;
218             if (CamposEjercicio[4] != "") ImplementacionConcreta.duracion = Convert.ToSingle(CamposEjercicio[4], System.Globalization.Cu
219             else ImplementacionConcreta.duracion = -1;
220             if (CamposEjercicio[5] != "") ImplementacionConcreta.ConsumoCalorico = Convert.ToSingle(CamposEjercicio[5], System.Globaliza
221             else ImplementacionConcreta.ConsumoCalorico = -1;
222 
223             rutinaLeida.AñadirEjercicio(ImplementacionConcreta, Convert.ToInt32(CamposDia[0]), i);
224         }
225 
226     }
227 
228     rutinaLeida.Duracion = Convert.ToInt32(CamposPrincipales[3]);
229     rutinaLeida.CalcularConsumoCaloricoDiario();
230     rutinaLeida.CalcularConsumoCalorico();
231 
232     this.ListaRutinas.Add(rutinaLeida);
233 }
234 
235 ArchivoRutinas.Close();
236 
237 //una vez tenemos la lista de las rutinas, procedemos a generar el arbol de rutinas que se mostrará
238 GenerarArbolRutinas();
```

Ilustración 27: Captura del código correspondiente a cargar las rutinas del fichero.

La Ilustración 27 describe como la aplicación lee línea a línea el fichero de rutinas y guarda la información de cada rutina en la lista de rutinas interna. Como sucedía al leer los ejercicios cada línea se divide en campos y se rellena la instancia de la clase rutina —línea **189**, **191**, **192**. Cuando encontramos el campo correspondiente a la tabla de ejercicios se vuelve a dividir dicho campo en días —línea **194**—, cada día a su vez se vuelve a dividir para obtener los ejercicios que componen ese día —línea **198—** y cada ejercicio se divide para obtener los campos que componen el ejercicio —línea **202**— y rellenamos la información del ejercicio concreto en la instancia de la clase *Ejercicio_Implementación.* Este proceso se repite para todos los ejercicios y días de la tabla de ejercicios de la rutina que se está cargando —líneas **186** a **223.** Una vez leída y añadida la tabla de ejercicios se guardan el resto de campos de la rutina: *Duración,* línea **228**; se calcula el consumo calórico —líneas **229** y **230**— y se añade la rutina a la lista de rutinas — línea **232**. Una vez finalizado el proceso se cierra el descriptor de fichero y se genera el árbol de rutinas.

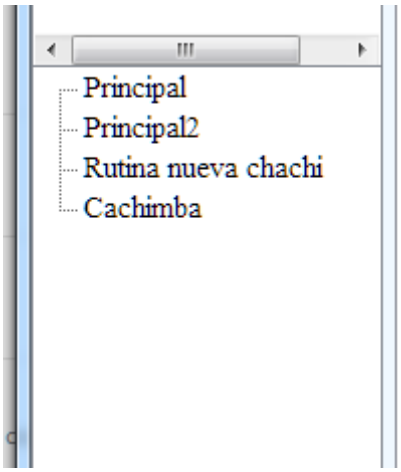


Ilustración 28: Captura de la lista de rutinas una vez el árbol de rutinas ha sido generado.

Esta forma de proceder se repite para todos los módulos, con las variaciones necesarias según las peculiaridades específicas de cada fichero.

Como ya se ha comentado al comienzo de este punto el otro momento en que la base de datos interactúa con la aplicación es a la hora de cerrar un módulo —habiendo seleccionado la opción cerrar de la barra de menús— o al seleccionar la opción *Guardar cambios* si es que el módulo lo permite. Existe una excepción con el módulo *General*. En este módulo los cambios no se guardan al salir —que sería al cerrar la aplicación— sino que cuando se crea un nuevo usuario —por ejemplo— todos los ficheros y carpetas de la base de datos que serán necesarios más adelante se crean en ese momento.

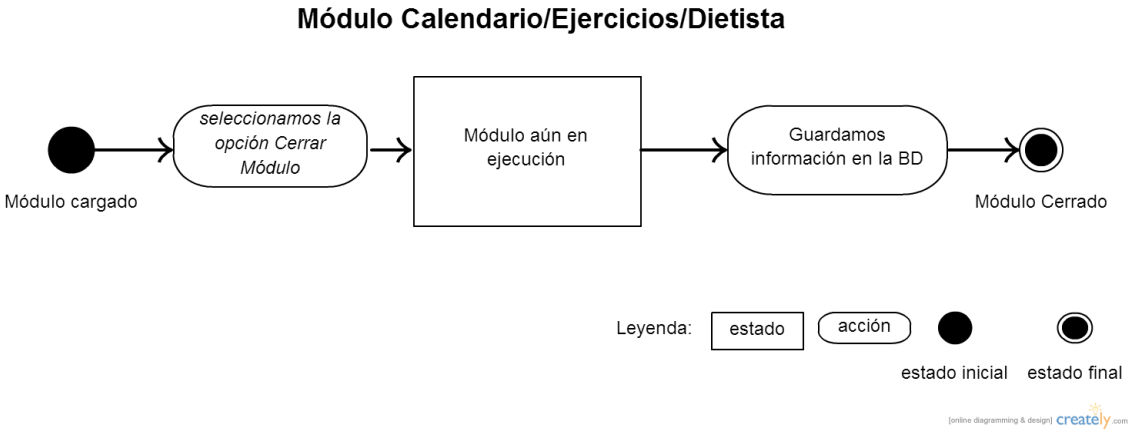


Ilustración 29: Diagrama de estados del proceso de guardar cambios en la BD.

A continuación se describe el proceso de volcado de la información de las listas internas a la base de datos del módulo *Calendario*. El módulo *Ejercicios* y *Dietista* sigue un funcionamiento análogo.

```
500 private void GuardarCambios()
501 {
502      //Abrimos el archivo con los nombres de los calendarios del usuario
503     System.IO.StreamWriter ArchivoNombresCalendario = new System.IO.StreamWriter("Base de
504     foreach (String nombre in this.ListaNombreCalendarios)
505     {
506         ArchivoNombresCalendario.WriteLine(nombre);
507     }
508
509     ArchivoNombresCalendario.Close();
510
```

Ilustración 30: Fragmento 1 de código del método *GuardarCambios()*. Este método es el encargado de guardar los cambios llevados a cabo en la base de datos.

Al seleccionar la opción de guardar los cambios en el módulo *Calendario* primeramente se guardan todos los nombres de los calendarios de dicho usuario en un fichero. Únicamente se guarda el nombre. Esto se describe en el código capturado en la Ilustración 30.

Hecho esto, para cada calendario de la lista se crea un fichero —se sobrescriben los que ya existieran— y se inserta la información línea a línea —Ilustración 31 e Ilustración 32.

```
511 //Guardamos cada calendario en un archivo
512 foreach( Calendar.NET.Calendar calendario in this.ListaCalendarios)
513 {
514     System.IO.StreamWriter ArchivoCalendario = new System.IO.StreamWriter("Base de datos/Módulo Calendario/"
515     ArchivoCalendario.WriteLine("FechaCalendario");
516     ArchivoCalendario.WriteLine(calendario.CalendarDate.ToString(System.Globalization.CultureInfo.InvariantC
517     ArchivoCalendario.WriteLine("Tipo;Nombre;Fecha (MM/dd/aaaa HH/mm);Duración(horas);Frecuencia;IgnorarComp
```

Ilustración 31: Fragmento 2 del código del método *GuardarCambios()*.

En la ilustración superior: la palabra clave *FechaCalendario,* línea **515**; la fecha propiamente dicha, línea **516**; y la leyenda de la estructura de las líneas evento, línea **517**. Entonces cada evento del calendario se guarda el fichero como una nueva línea de este —Ilustración 32, en la página siguiente.

```
518         foreach (IEvent evento in calendario.GetListofEvents())
519         {
520             int recurrencia = (int)evento.RecurringFrequency;
521             switch (evento.tipo)
522             {
523                 case 1:
524                     ArchivoCalendario.WriteLine(evento.tipo.ToString() + ";" + evento.EventText + ";" + ever
525                     break;
526                 case 2:
527                     ArchivoCalendario.WriteLine(evento.tipo.ToString() + ";" + evento.EventText + ";" + ever
528                     break;
529                 case 3:
530                     EventoEjercicio eventoEj = (EventoEjercicio) evento.Clone();
531                     ArchivoCalendario.WriteLine(evento.tipo.ToString() + ";" + evento.EventText + "|" + ever
532                     break;
533                 case 4:
534                     EventoDieta eventoDieta = (EventoDieta) evento.Clone();
535                     List<String> eventosDia = new List<string>();
536                     int i = 1;
537                     foreach (List<IEvent> dia in eventoDieta.ListaEventosComida)
538                     {
539                         List<String> eventosInternos = new List<String>();
540                         foreach (IEvent eventoComida in dia)
541                         {
542                             eventosInternos.Add(eventoComida.tipo.ToString() + "}" + eventoComida.EventText
543                         }
544                         String stringEventosInternos = i.ToString() + "#" + String.Join("#", eventosInternos
545                         eventosDia.Add(stringEventosInternos);
546                         i++;
547                     }
548                     String stringEventosDia = String.Join("|", eventosDia);
549                     ArchivoCalendario.WriteLine(eventoDieta.tipo.ToString() + ";" + eventoDieta.EventText +
550                     //Seguir
551                     break;
552                 case 5:
553                     EventoRutina eventoRutina = (EventoRutina) evento.Clone();
554                     List<String> eventosDiaRutina = new List<string>();
555                     int j = 1;
556                     foreach (List<IEvent> dia in eventoRutina.ListaEventosEjercicio)
557                     {
558                         List<String> eventosInternos = new List<String>();
559                         foreach (EventoEjercicio eventoEjercicio in dia)
560                         {
561                             string nombreYatrib = eventoEjercicio.EventText + "[" + eventoEjercicio.Series.T
562                             eventosInternos.Add(eventoEjercicio.tipo.ToString() + "}" + nombreYatrib + "}" +
563                         }
564                         String stringEventosInternos = j.ToString() + "#" + String.Join("#", eventosInternos
565                         eventosDiaRutina.Add(stringEventosInternos);
566                         j++;
567                     }
568                     String stringEventosDiaRutina = String.Join("|", eventosDiaRutina);
569                     ArchivoCalendario.WriteLine(eventoRutina.tipo.ToString() + ";" + eventoRutina.EventText
570                     break;
571             }
572         }
573         ArchivoCalendario.Close();
574     }
```

Ilustración 32: Fragmento 3 del código del método *GuardarCambios()*.

Según el tipo de evento que sea, la línea se *ensambla* de una u otra forma. En el caso de ser de tipo *4* —*Dieta,* líneas **533** y en adelante — o *5*—*Rutina,* líneas **552** y en adelante— hay que recorrer los eventos *internos* que forman al evento principal.

Una vez finalizado el proceso se cierra el descriptor de fichero.

4.2.3 Implementación de la interfaz

La implementación de la interfaz gráfica de la aplicación se ha realizado mediante formularios —*Windows Forms*. Estos formularios implementan cada una de las ventanas, pestañas y *PopUps* que forman la aplicación. La herramienta de desarrollo *Visual Studio* permite la creación de proyectos de este tipo y su interacción es intuitiva.

A la hora de implementar la interfaz gráfica se siguió con la filosofía descrita en el punto 3.4 —*Diseño de la Interfaz*, página 53— de facilitar en la medida de lo posible la interacción entre el usuario y la aplicación.

Qué es un formulario?

Los formularios son un tipo de representación de una interfaz gráfica de usuario. Una *form* contiene dentro de sí controles y componentes con los que el usuario interactuará una vez ejecutada la aplicación. Botones, *textBox*, tablas, *timers*… son ejemplos de este tipo de elementos.

Los formularios utilizan un sistema de eventos que se disparan bajo determinadas circunstancias —cuando el usuario clica en la pantalla, cuando escribe, cuando pasa un determinado tiempo…— y se encargan de llevar a cabo la funcionalidad que se ha asignado para cuando dicho evento se active. Un ejemplo de formulario, extraído de la aplicación, es el formulario descrito en las ilustraciones siguientes:

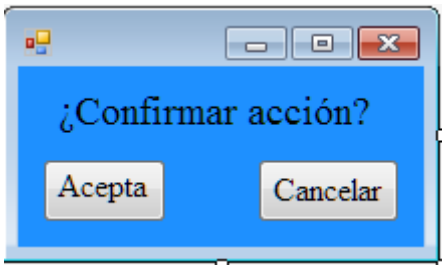


Ilustración 33: Formulario *ModuloGeneralMensajeConfirmacion*

```
private void BotonAceptar_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.OK;
}

private void BotonCancelar_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
}
```

Ilustración 34: Código que se ejecutará al dispararse los eventos enlazados al click de los botones *Aceptar* y *Cancelar.*

Dentro de los formularios, además de tratar la interacción del usuario con la aplicación también se controlan todos los cambios que estas interacciones suponen para la aplicación. Cuando un usuario crea un Ejercicio, lo elimina, lo modifica, etc. es dentro de los propios formularios que se implementa la lógica que controla que esto se lleve a cabo correctamente y que no haya incoherencias de datos.

Implementación de interfaz

El proceso de convertir la interfaz de una *idea en papel* a un elemento plenamente funcional de la aplicación pasa por un número de pasos.

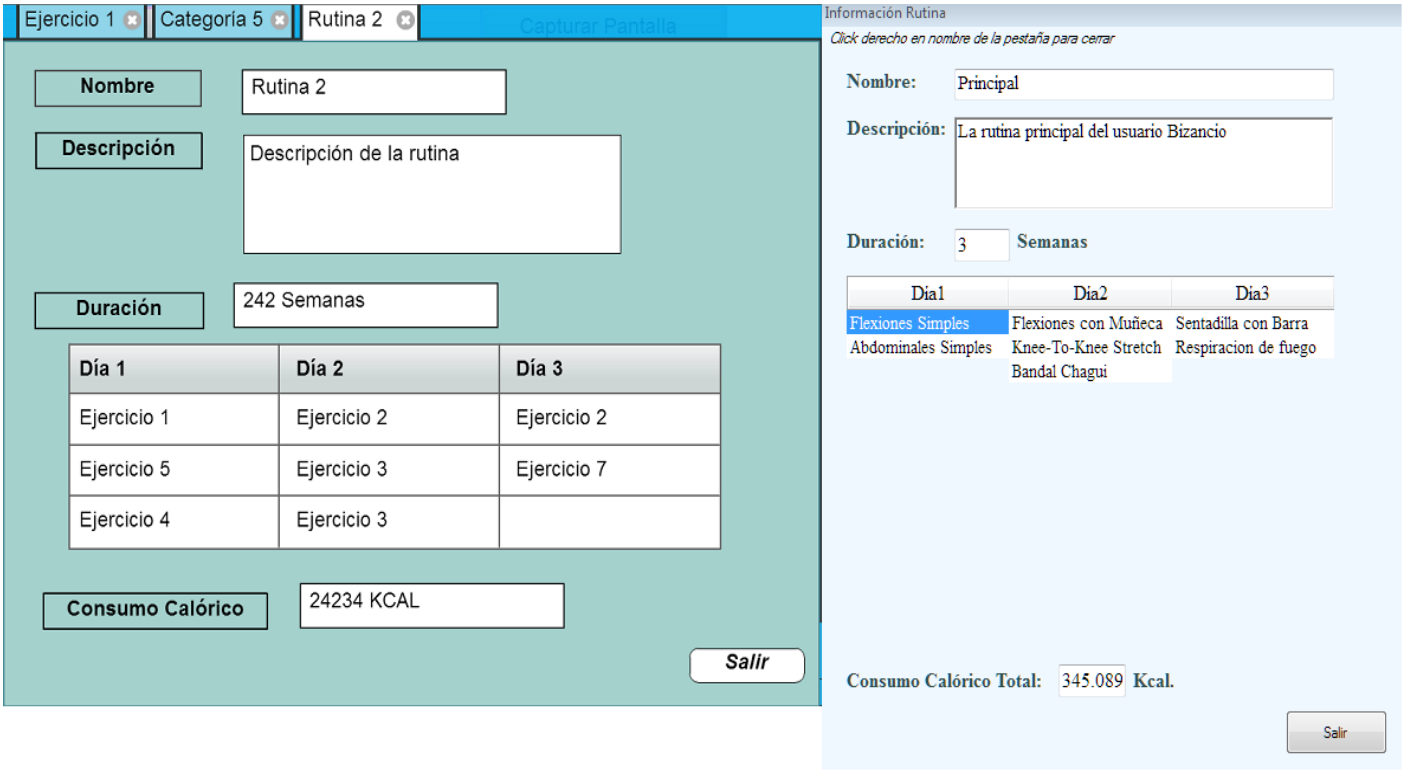
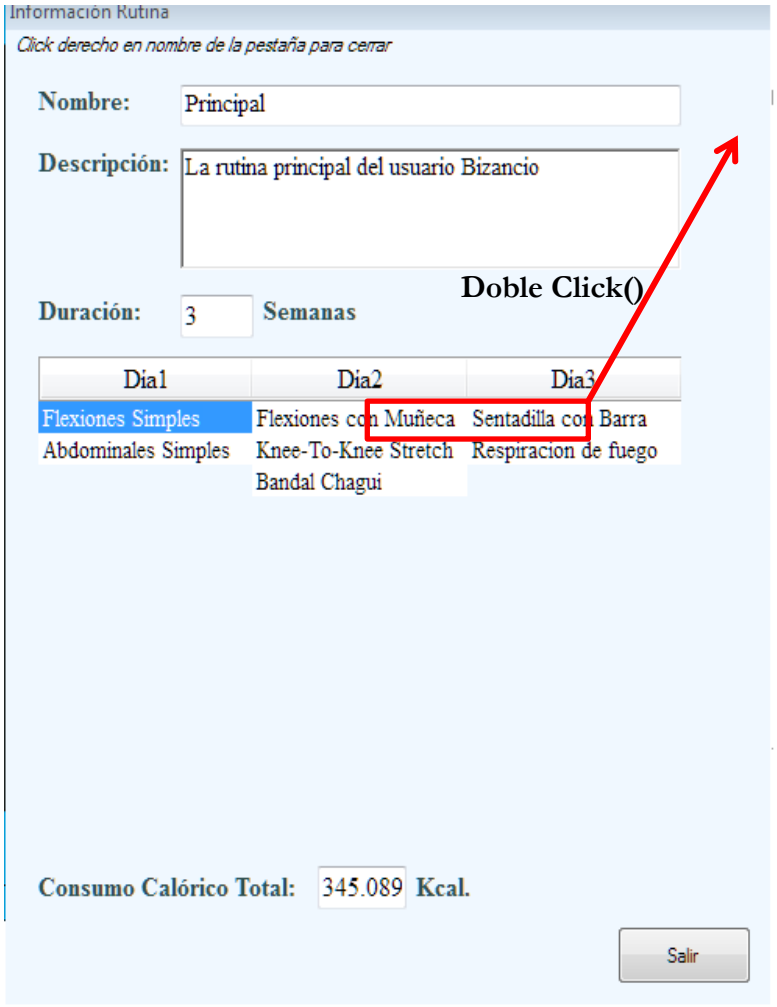


Ilustración 35: Formulario *Información de Rutina*, diseño (izquierda) e implementación final (derecha)

El primer paso es trasladar la apariencia del diseño al formulario. Esto se consigue añadiendo los elementos que aparecen en el diseño como controladores y componentes. Estos elementos son los botones, tablas, cajas de texto, etiquetas, etc. Finalizado este paso lo que tenemos es la *fachada* del formulario, la apariencia final de este.

El siguiente paso es configurar el comportamiento de estos controles y componentes. Hay que especificar para cada uno de ellos que tipo de reacción queremos según el tipo de interacción o *estímulo* que el usuario les dé. Esto lo conseguiremos mediante los mencionados eventos. Cada evento está ligado a un *Event Handler*, que define qué acción o acciones se llevarán a cabo cuando se produzca el evento.




```
private void TablaEjercicios_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    if (TablaEjercicios.Rows[e.RowIndex].Cells[e.ColumnIndex].Value != null)
    {
        Modulo_Ejercicios.Ejercicio_Tipo EjercicioAAbrir = this.parentForm.GetListaEjerciciosBase().Find(i
        formacionEjercicios EjercicioAbierto;
        ll)
        {
            EjercicioAAbrir = this.parentForm.GetListaEjerciciosPersonalizada().Find(item => item.Nombre.E
            EjercicioAbierto = new ModuloEjerciciosPestañaInformacionEjercicios(EjercicioAAbrir, this.pare
        }
        else EjercicioAbierto = new ModuloEjerciciosPestañaInformacionEjercicios(EjercicioAAbrir, this.par
        EjercicioAbierto.FormClosed += ((s, args) => this.parentForm.CerrarPestaña());
        EjercicioAbierto.WindowState = FormWindowState.Maximized;
        EjercicioAbierto.Dock = DockStyle.Fill;
        TabPage PestañaAbrirEjercicio = new TabPage();
        PestañaAbrirEjercicio.Text = EjercicioAAbrir.Nombre;
        EjercicioAbierto.TopLevel = false;
        EjercicioAbierto.Parent = PestañaAbrirEjercicio;
        this.parentForm.añadirPestañaBarraHerramientas(PestañaAbrirEjercicio);
        this.parentForm.GetBarraPestañas().SelectedTab = PestañaAbrirEjercicio;
        EjercicioAbierto.ControlBox = false;
        EjercicioAbierto.Show();
    }
}
```

namespace Modulo_Ejercicios

```
private void TablaEjercicios_CellMouseEnter(object sender, DataGridViewCellEventArgs e)
{
    if ((e.ColumnIndex >= 0) && (e.RowIndex >= 0))
    {
        if (TablaEjercicios[e.ColumnIndex, e.RowIndex].Value != null)
        {
            Modulo_Ejercicios.Ejercicio_Implementacion ejercicio = this.rutina.ListaEjercicios[e.Co
            string info = "Ejercicio:\t\t" + ejercicio.Nombre + "\nSeries:\t\t" + ejercicio.Series.
            TablaEjercicios.Rows[e.RowIndex].Cells[e.ColumnIndex].ToolTipText = info;
        }
    }
}
```

Ilustración 36: Implementación del formulario *InformacionRutina* (izquierda) y los métodos que se encargan de los eventos que se activan al hacer doble click en una celda y pasar por encima de la tabla de ejercicios.

En la Ilustración 36 se puede observar este segundo paso que hemos descrito en el párrafo anterior a ésta. A la izquierda tenemos el formulario y a la derecha el código que se ejecuta al dispararse el evento *CellDoubleClick* del formulario —hacer doble click sobre una celda de la tabla de ejercicios— y el código que se ejecuta al dispararse el evento *CellMouseEnter* —al entrar el cursor dentro de una celda de la tabla de ejercicios.

El tercer y último paso de este proceso de pasar del diseño a la implementación es el de comprobar el correcto funcionamiento de la implementación que se acaba de llevar a cabo. La manera de probar el funcionamiento no es otra que ejecutar la aplicación y seguir los pasos que el usuario seguiría para hacer uso de la funcionalidad que el formulario implementa. En el caso del formulario que estamos usando como ejemplo ejecutaríamos la aplicación, cargaríamos el módulo *Ejercicios* y abriríamos una rutina. Una vez hecho esto probaríamos que efectivamente el funcionamiento deseado y el funcionamiento real sean el mismo.

Este proceso descrito se repite para todos y cada uno de los formularios que forman la aplicación.

Pruebas e incidencias durante la implementación

Como ya se ha explicado en el último párrafo del apartado anterior las pruebas de la fase de implementación consistían en comprobar el correcto funcionamiento de cada una de las funcionalidades implementadas. En el anexo 7.5 —Incidencias durante la implementación, página 131— se encuentran documentadas las incidencias surgidas durante la fase de implementación.

4.3 Instalador y ejecutable

Para crear el instalador de la aplicación hemos de crear un nuevo proyecto en nuestra solución de *Visual Studio* del tipo *Instalación e Implementación*, concretamente un proyecto *InstallShield*.

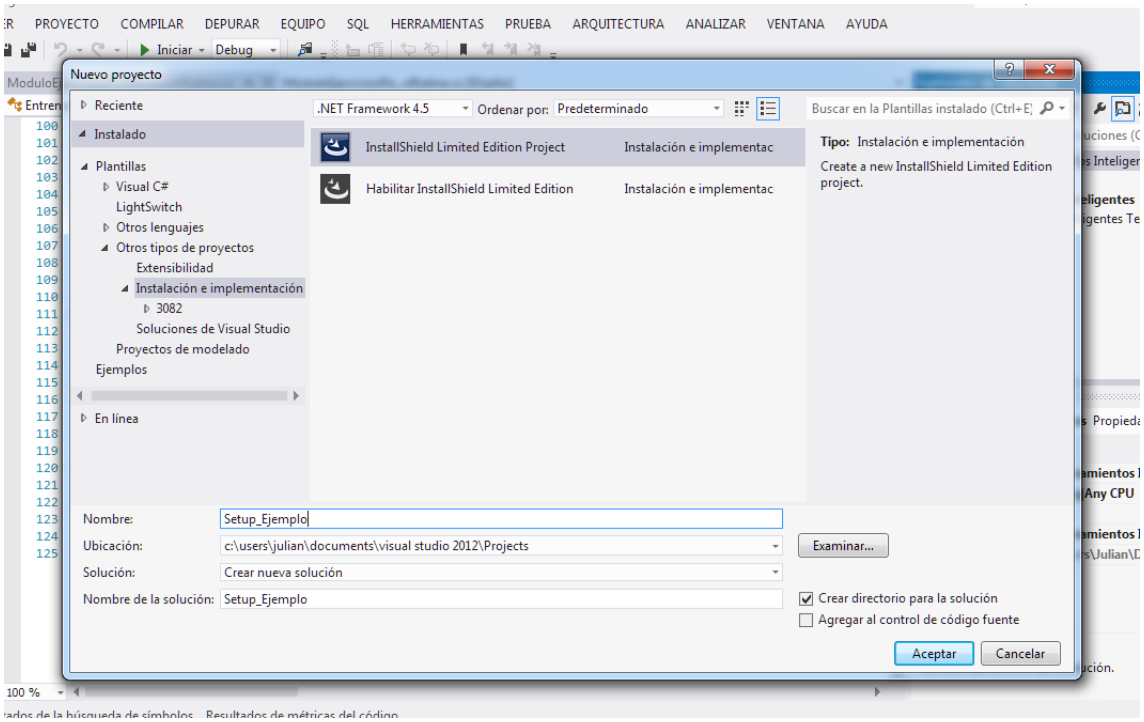


Ilustración 37: Captura de la ventana de creación de un nuevo proyecto de tipo *InstallShield*.

Una vez el proyecto está creado hay que seguir los pasos que el asistente nos indica para ir configurando los diferentes aspectos del instalador. El primer paso consiste en rellenar la información correspondiente al nombre de la compañía, nombre y versión de la aplicación, web de la compañía y el icono de la aplicación —Ilustración 38, página 76.

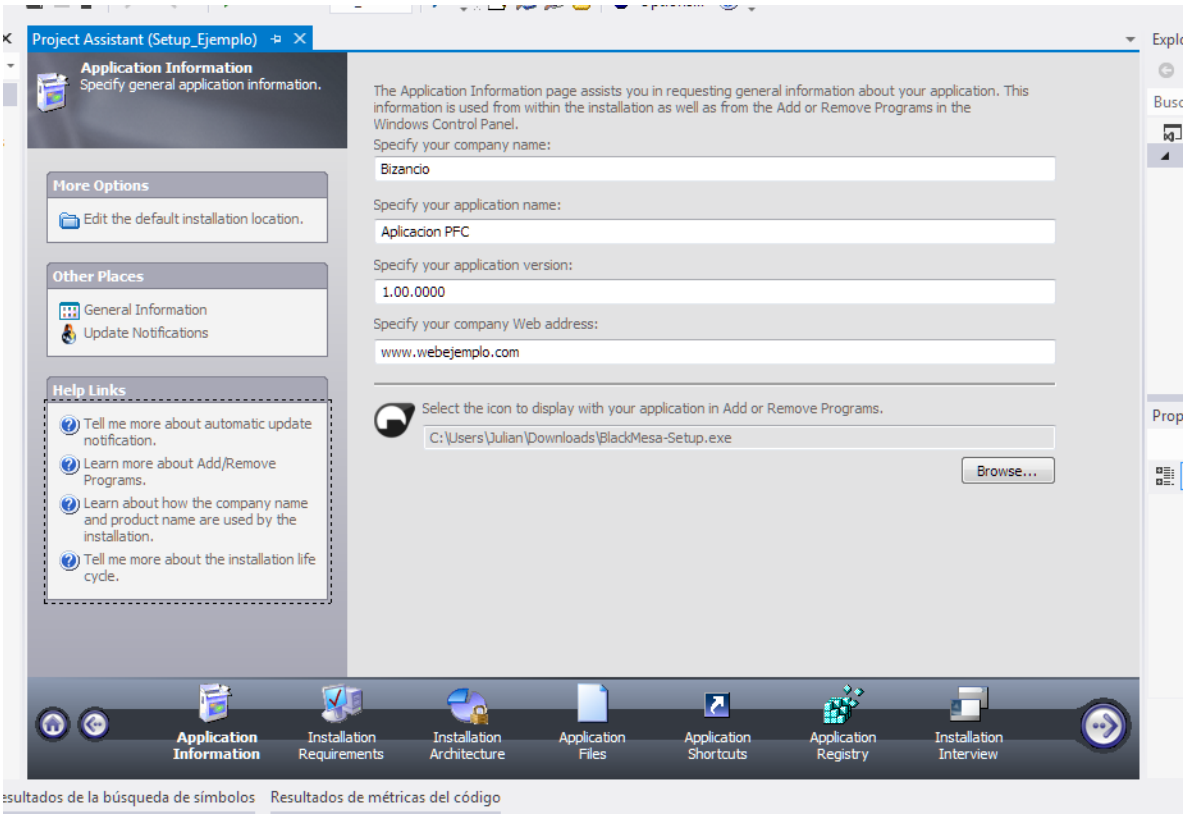


Ilustración 38: Asistente de creación del proyecto instalador, paso 1.

El siguiente paso del asistente consiste en elegir los requisitos de software adicional de la aplicación —Ilustración 39, página 76—, si es necesario un sistema operativo en concreto y si es necesario algún otro software. En nuestro caso es necesario tener *.NET 4.5* instalado.

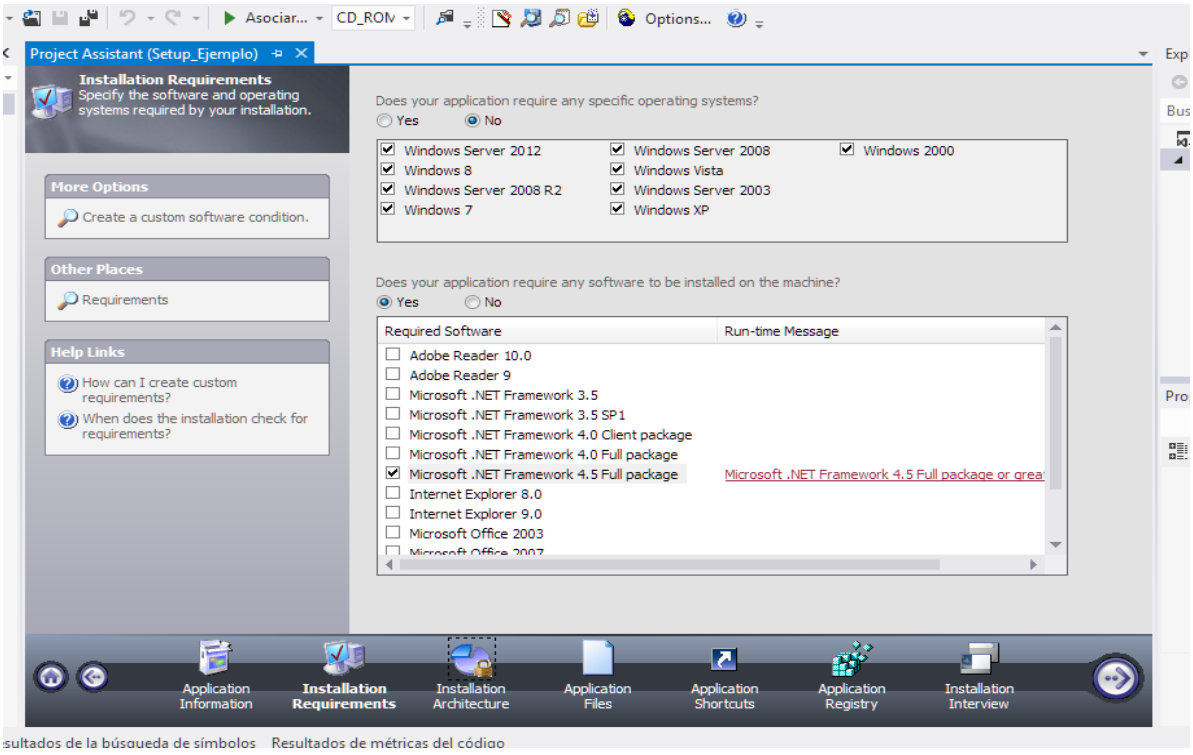


Ilustración 39: Asistente de creación del proyecto instalador, paso 2.

El siguiente paso no está disponible en la edición limitada de *InstallShield*. Este paso es el de personalizar la arquitectura de la instalación, especificando si se da opción al usuario de elegir qué componentes instalar o no, etc.

Saltamos entonces al siguiente paso, seleccionar los archivos necesarios para el funcionamiento de la aplicación, en nuestro caso las carpetas y ficheros que forman la base de datos.

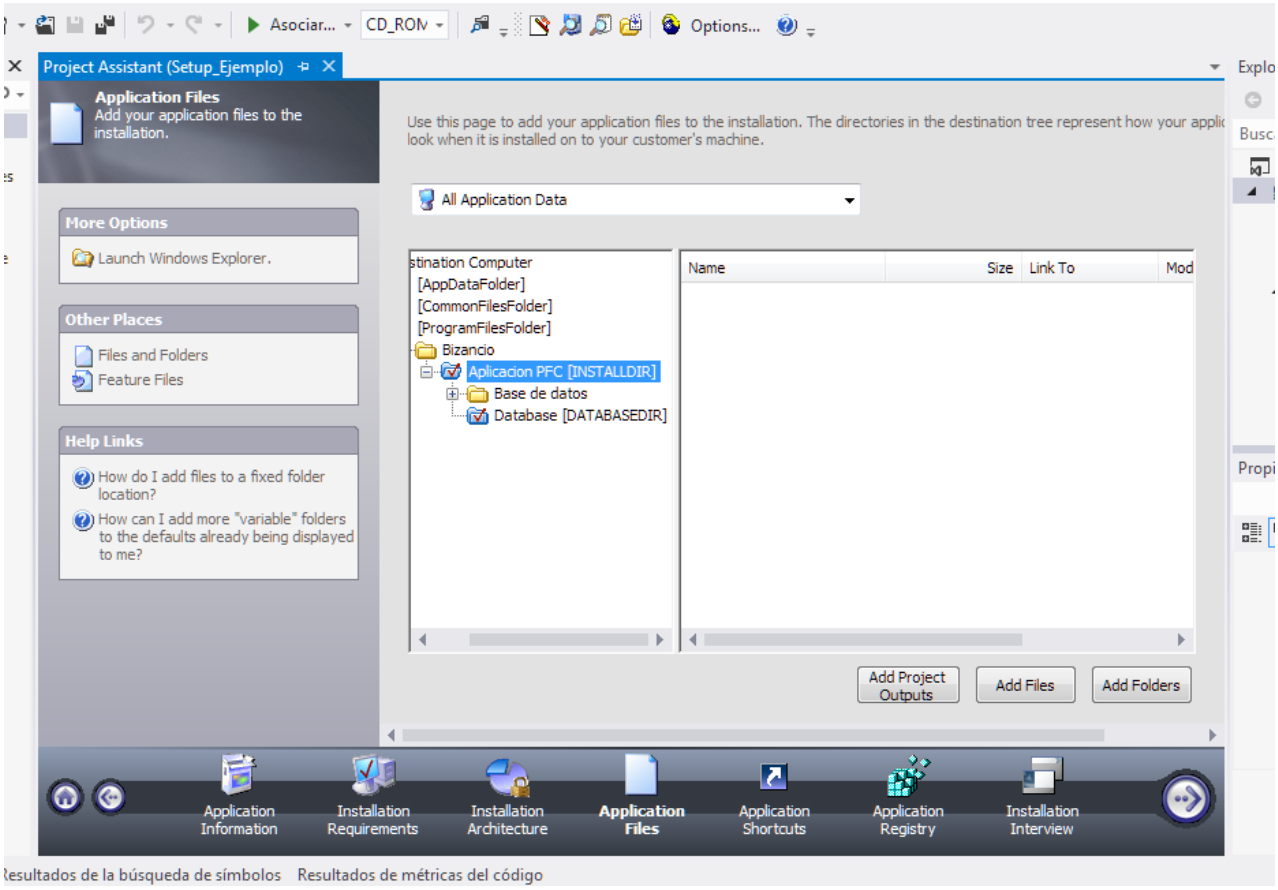


Ilustración 40: Asistente de configuración del instalador, paso 4

El quinto paso de la configuración del instalador consiste en seleccionar los accesos directos en el menú de inicio de programas de Windows. En nuestro caso no seleccionamos ninguno.

Tampoco seleccionaremos ningún archivo para añadir al registro de Windows, sexto paso de la configuración.

El último paso de la configuración del instalador es seleccionar si queremos que aparezca un mensaje de acuerdo de licencia, si queremos que el usuario tenga que introducir su nombre y compañía a la que pertenece, si permitimos al usuario cambiar dónde se instalará la aplicación y si damos la opción de ejecutar la aplicación directamente tras la instalación.

Una vez hemos completado el último paso del asistente ya podemos generar el proyecto. El instalador se encuentra ahora en la carpeta correspondiente al proyecto *InstallShield* que hemos creado:

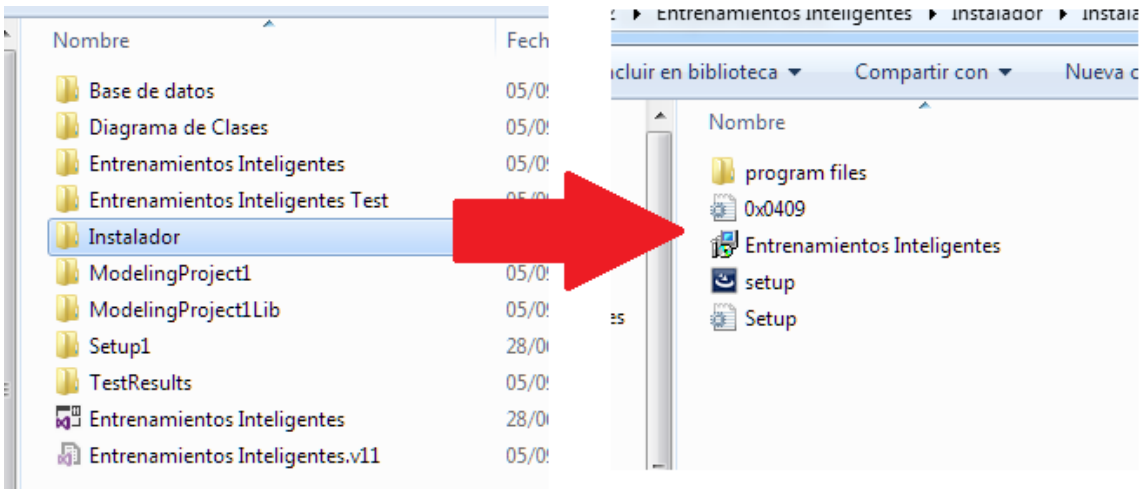


Ilustración 41: Captura de la carpeta donde se encuentra el proyecto del instalador (izquierda) y de los instalador y archivos necesarios para la instalación de la aplicación (derecha).

Si ejecutamos el archivo *setup.exe* se inicia el proceso de instalación de la aplicación. Seguimos los pasos de la instalación y una vez terminada encontraremos el ejecutable de la instalación en la carpeta que hayamos especificado.

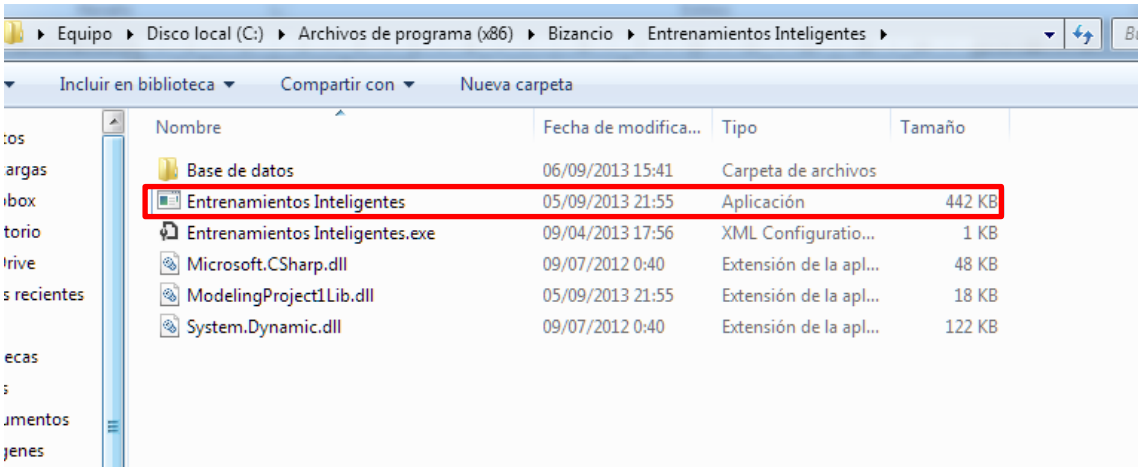


Ilustración 42: Captura de la carpeta donde se ha instalado la aplicación

5. Conclusiones y vías de continuación

Todo el proceso descrito en la memoria culmina con el ejecutable de la aplicación plenamente funcional.

Durante el proceso de desarrollo —desde la idea inicial hasta su estado final— el proyecto ha ido evolucionando y adaptándose a las necesidades y circunstancias que iban surgiendo. En consecuencia, el resultado final difiere en algunos aspectos de lo inicialmente planeado. El proyecto —que se pensó como un proyecto centrado mayoritariamente en la creación de una aplicación que aprovechara los beneficios ofrecidos por los algoritmos de inteligencia artificial y las redes neuronales— acabó convirtiéndose en un proyecto centrado en el ámbito de la Ingeniería del Software, concretamente en el desarrollo completo de una aplicación desde cero. Este cambio de rumbo, sin embargo, no afecta a la idea principal y más importante del proyecto: unir formación e interés personal en un proyecto cuyo resultado cubriera una necesidad real.

Al final de este proyecto lo que obtenemos —además de multitud de nuevos conocimientos producto del trabajo llevado a cabo— es una aplicación que puede ser usada para administrar programas de entrenamiento —en los que se incluye el apartado alimenticio— y que permite el seguimiento de estos mediante un sistema de calendarios y eventos que el usuario puede personalizar.

Vías de continuación

Como en cualquier proyecto existen siempre vías de continuación que amplíen el abanico de funcionalidades y opciones ofrecidas, así como revisiones que ayuden a mejorar el funcionamiento del mismo. Estas son algunas de estas vías posibles:

- **Implementar una base de datos SQL:** En el apartado de diseño ya se discutió acerca del uso de una base de datos SQL o no y, aunque se decidió optar por otro tipo de solución, una vía de continuación sería implementar la base de datos en formato SQL. Esto facilitaría la ampliación de la aplicación a otras plataformas —web, *Android*...
- **Módulo Inteligente:** Este módulo quedó excluido por causas relacionadas con la planificación temporal y sería una de las principales vías de continuación del proyecto.
- **Versión WEB/Móvil:** Teniendo en cuenta que hoy en día la conexión a internet es algo tan común como tener un ordenador, una versión web/móvil de la aplicación incrementaría el valor de esta y permitiría acceder a ella desde cualquier ordenador.

6. Bibliografía

1. *Walkthrough: Creating and Running Unit Tests for Managed Code*
   a. **Dirección:** http://msdn.microsoft.com/en-us/library/ms182532.aspx
   b. **Descripción:** Cómo crear y utilizar *Unit Tests* para un proyecto en C#.

2. *Calendar.NET*
   a. **Dirección:** http://www.codeproject.com/Articles/378900/Calendar-NET
   b. **Descripción:** Control utilizado como base para crear el control Calendario, el controlador principal del módulo calendario.

3. *Calories Burned During Exercise, Activities, Sports and Work*
   a. **Dirección:** http://www.nutristrategy.com/caloriesburned.htm
   b. **Descripción:** Tabla de actividades con el consumo calórico aproximado según el peso del deportista.

4. *Timetable Tutorial*
   a. **Dirección:** http://code.daypilot.org/65101/timetable-tutorial-asp-net-c-vb-net
   b. **Descripción:** Tutorial para crear una tabla horaria en Ajax.

5. *Printing the form*
   a. **Dirección:** http://msdn.microsoft.com/en-us/library/aa287529(v=vs.71).aspx
   b. **Descripción:** Ejemplo de cómo imprimir un formulario.

6. *Tabla de Calorías de los alimentos*
   a. **Dirección:** http://www.vitalimentos.es/cuantas-calorias/
   b. **Descripción:** Tabla de calorías de cada alimento agrupados por tipo.

7. *Estiramientos Piernas*
   a. **Dirección:** http://www.estiramientos.es/index.php?filt=piernas&size=gran
   b. **Descripción:** Ejemplos de diferentes estiramientos para el tren inferior.

8. *TabControl: How to capture Mouse RightClick on Tab*
   a. **Dirección:** http://social.msdn.microsoft.com/Forums/windows/en-US/e09d081d-a7f5-479d-bd29-44b6d163ebc8/tabcontrol-how-to-capture-mouse-rightclick-on-tab
   b. **Descripción:** Hilo del foro del MSDN de *Microsoft* donde se comenta cómo detectar que el botón derecho del ratón ha sido pulsado.

9. *How can I make my own event in C#?*
   a. **Dirección:** http://stackoverflow.com/questions/623451/how-can-i-make-my-own-event-in-c

b. **Descripción:** Página de la web www.stackoverflow.com en que se comenta la creación de un evento personalizado.

10. *DateTime Structure*

a. **Dirección:** http://msdn.microsoft.com/en-us/library/system.datetime.aspx

b. **Descripción:** Propiedades de la estructura *DateTime*.

11. *mongoDB*

a. **Dirección:** http://www.mongodb.org/

b. **Descripción:** Página principal de las base de datos orientadas a documentos.

12. *How to clone a control?*

a. **Dirección:** http://social.msdn.microsoft.com/Forums/vstudio/en-US/cf04f12e-eb88-4814-b413-b7cf72010231/how-to-clone-a-control

b. **Descripción:** Hilo del foro del MSDN de Microsoft donde se comenta cómo clonar un control —propiedades, event handlers, etc.

13. *It is possible to copy all the properties of a certain control? (C# window forms)*

a. **Dirección:** http://stackoverflow.com/questions/3473597/it-is-possible-to-copy-all-the-properties-of-a-certain-control-c-window-forms

b. **Descripción:** Página de la web www.stackoverflow.com donde se comenta si es posible copiar todas las propiedades de un control determinado.

14. *How to Clone/Serialize/Copy & Paste a Windows Forms Control*

a. **Dirección:** http://www.codeproject.com/Articles/12976/How-to-Clone-Serialize-Copy-Paste-a-Windows-Forms

b. **Descripción:** Enfoque de cómo clonar un Control con todas sus propiedades y event handlers.

15. *Control.InValidate vs. Control.Refresh()?*

a. **Dirección:** http://bytes.com/topic/c-sharp/answers/244445-control-invalidate-vs-control-refresh

b. **Descripción:** Página de la web *www.bytes.com* donde se comenta la diferencia entre los métodos *InValidate* y *Refresh()*.

16. *Enumerate and copy properties from one object to another object of same type.*

a. **Dirección:** http://stackoverflow.com/questions/4546381/enumerate-and-copy-properties-from-one-object-to-another-object-of-same-type

b. **Descripción:** Página de la web *StackOverflow* en que se comenta cómo copiar las propiedades de un objecto a otro del mismo tipo.

17. *How to instantly change label text during a method at runtime?*

a. **Dirección:** http://stackoverflow.com/questions/15265520/how-to-instantly-change-label-text-during-a-method-at-runtime

b. **Descripción:** Página de la web *StackOverflow* en que se comenta cómo modificar el texto de una etiqueta durante la ejecución de un programa.

18. *Document-oriented database*

a. **Dirección:** http://en.wikipedia.org/wiki/Document-oriented_database
b. **Descripción:** Artículo de la Wikipedia sobre las bases de datos orientadas a documentos.

19. *What are the advantages and disadvantages of SQL?*

a. **Dirección:** http://wiki.answers.com/Q/What_are_the_advantages_and_disadvantages_of_SQL#page3
b. **Descripción:** Ventajas y desventajas de las bases de datos SQL.

7. Anexos

7.1 Diagramas de Gantt

Informe previo:

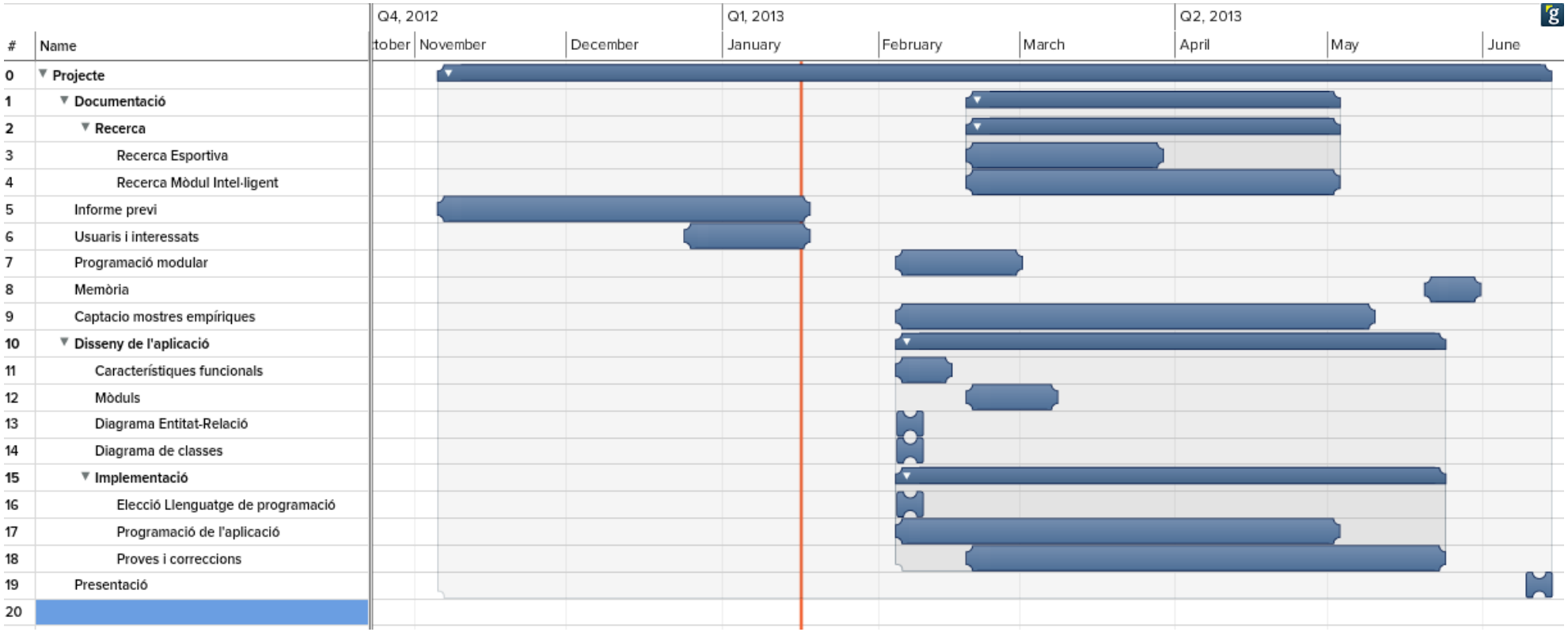


Ilustración 43: Como se puede observar prácticamente todas las tareas del proyecto se desarrollan en paralelo entre ellas (respetando la duración y restricciones de precedencia, etc.

Volver a la referencia

Diagrama de Gantt final, fase 1:

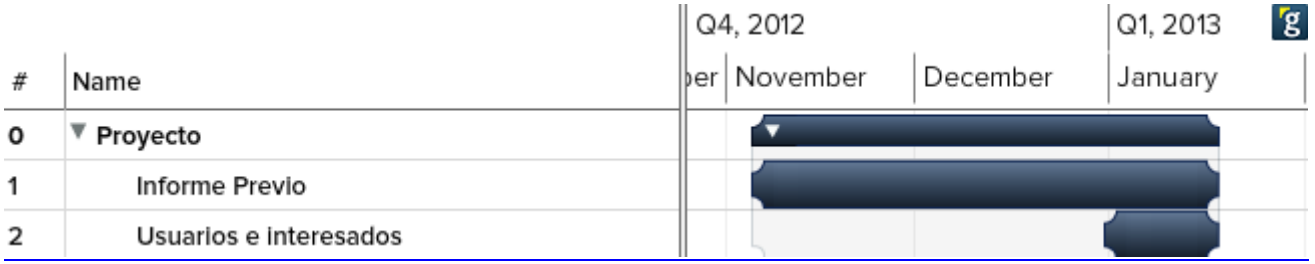


Ilustración 44: En este fragmento del diagrama de Gantt se muestran los primeros meses de desarrollo del proyecto.

Volver a la referencia

Diagrama de Gantt final, fase 2:

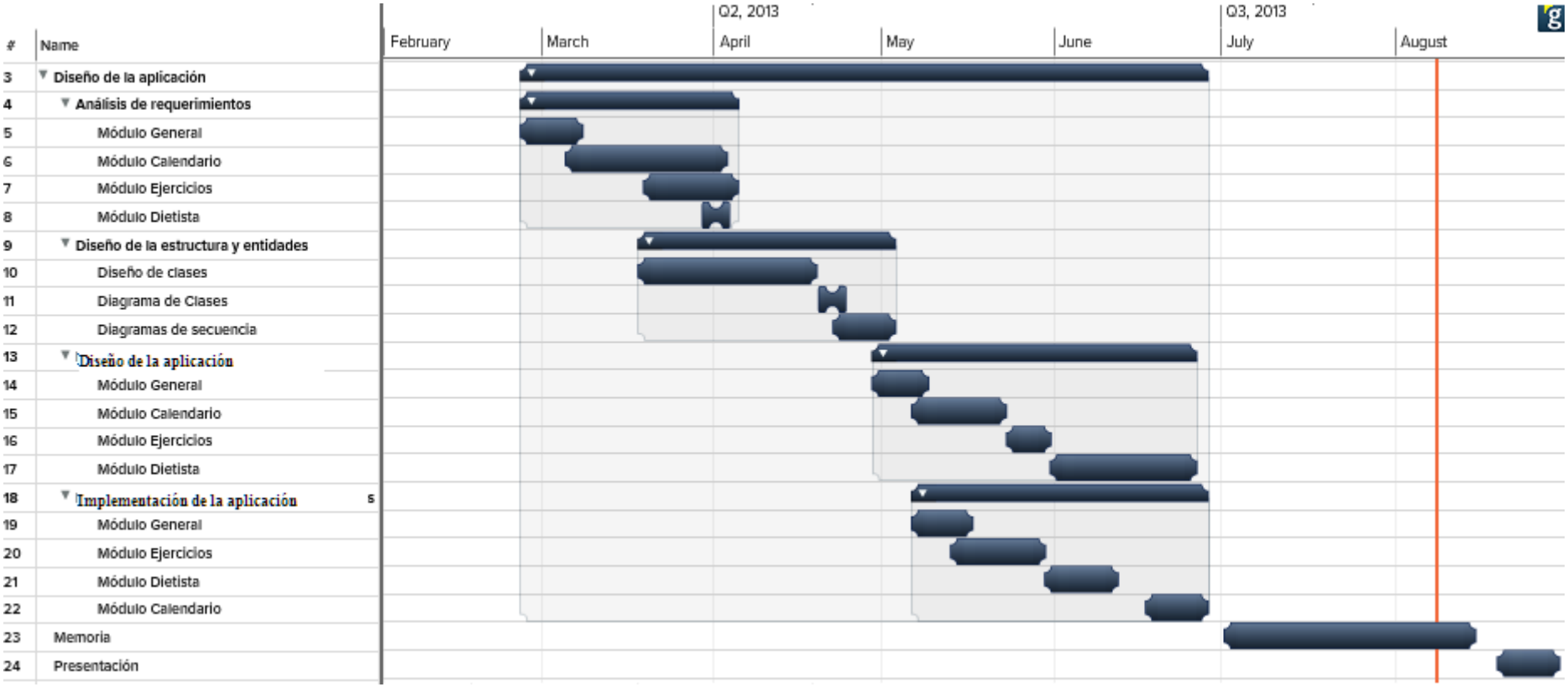


Ilustración 45: Este fragmento del diagrama muestra el desarrollo del proyecto de Febrero a Agosto de 2013.

Volver a la referencia

7.2 Diagrama Entidad/relación

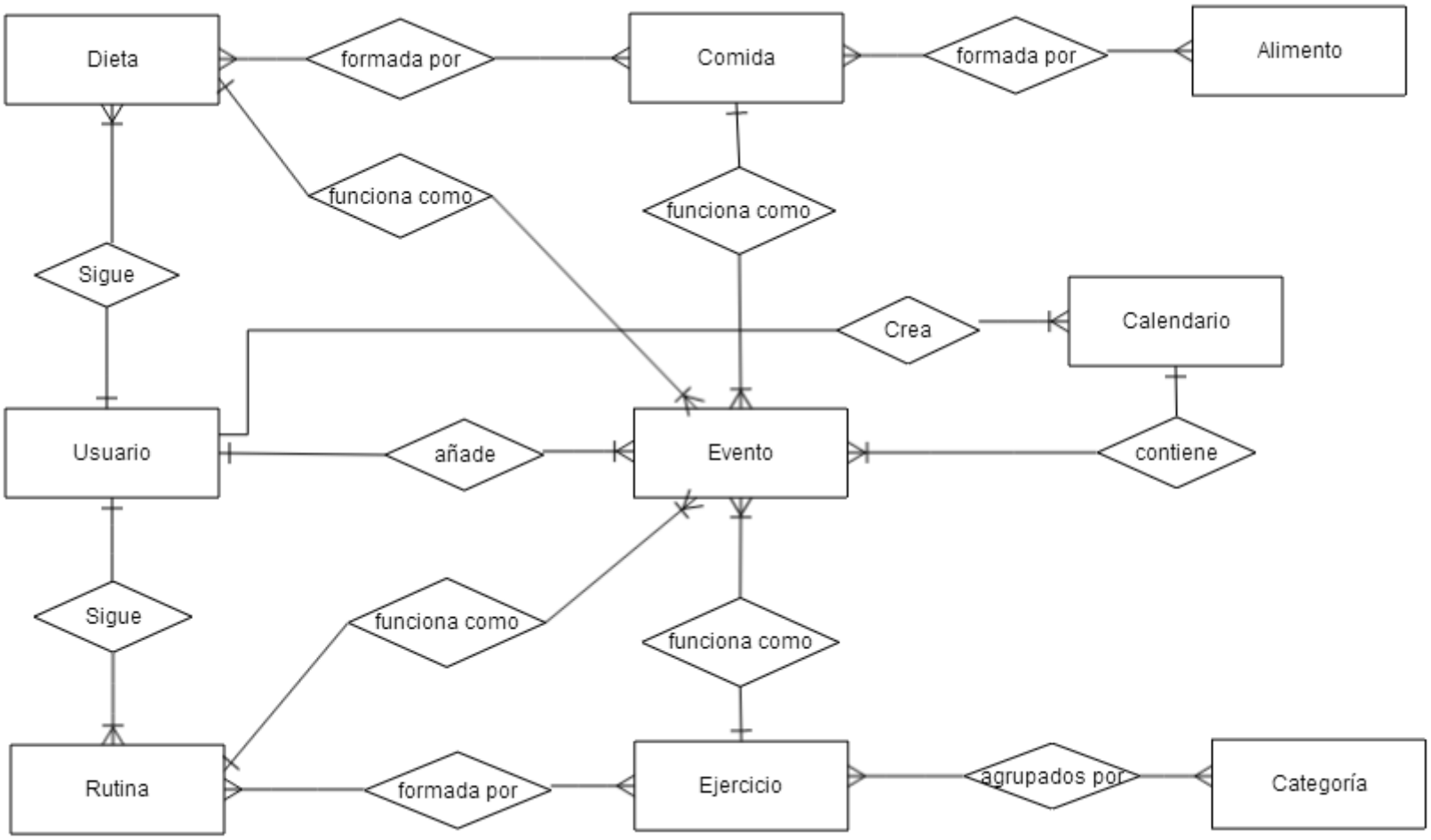


Ilustración 46: Diagrama Entidad/Relación de la aplicación

Volver a la Referencia

7.3 Diagramas de secuencia

7.3.1 Módulo General

Crear Usuario

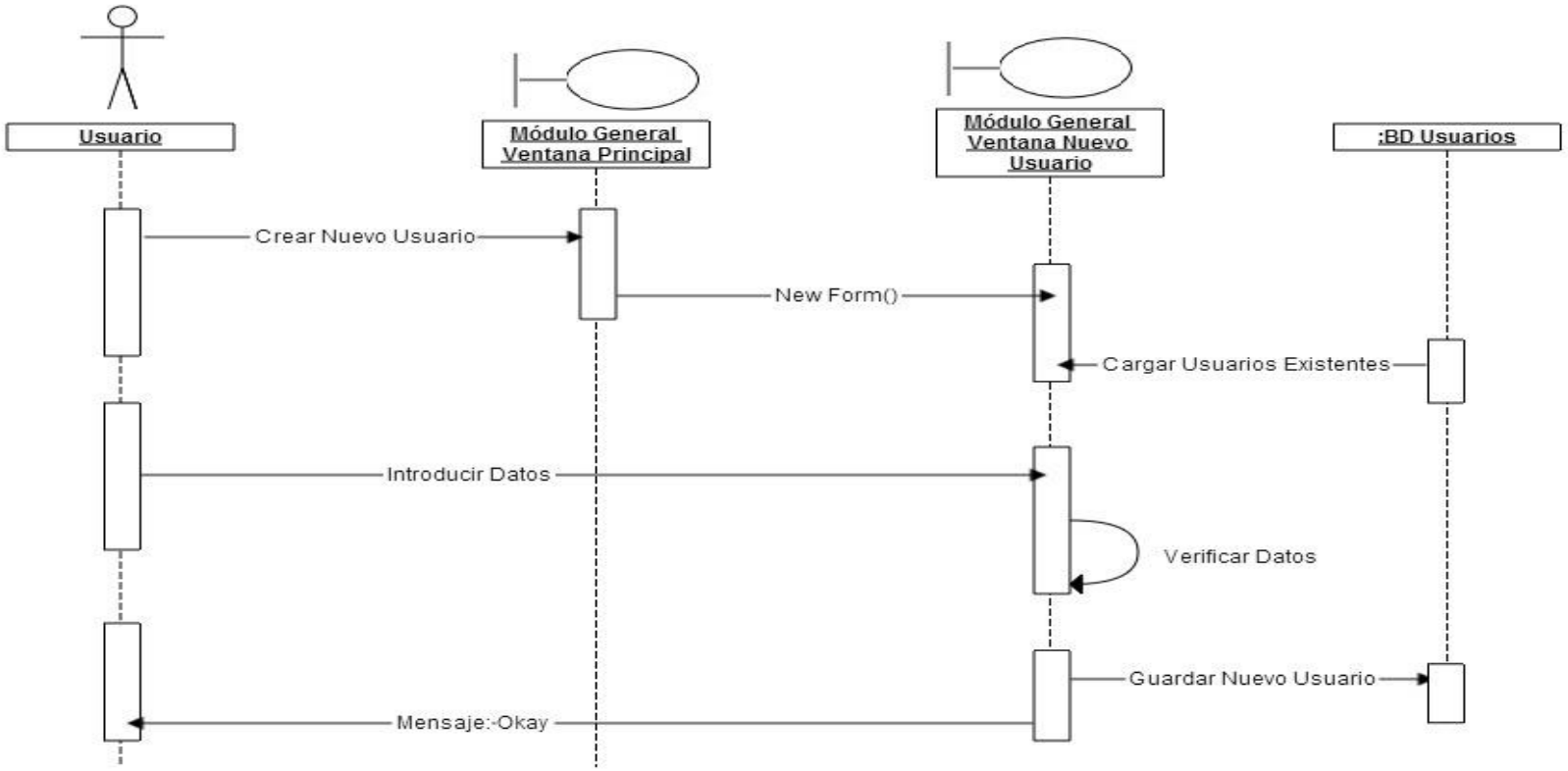


Ilustración 47: Diagrama de Secuencia de la creación de un nuevo usuario.

Loguear y desconectar Usuario

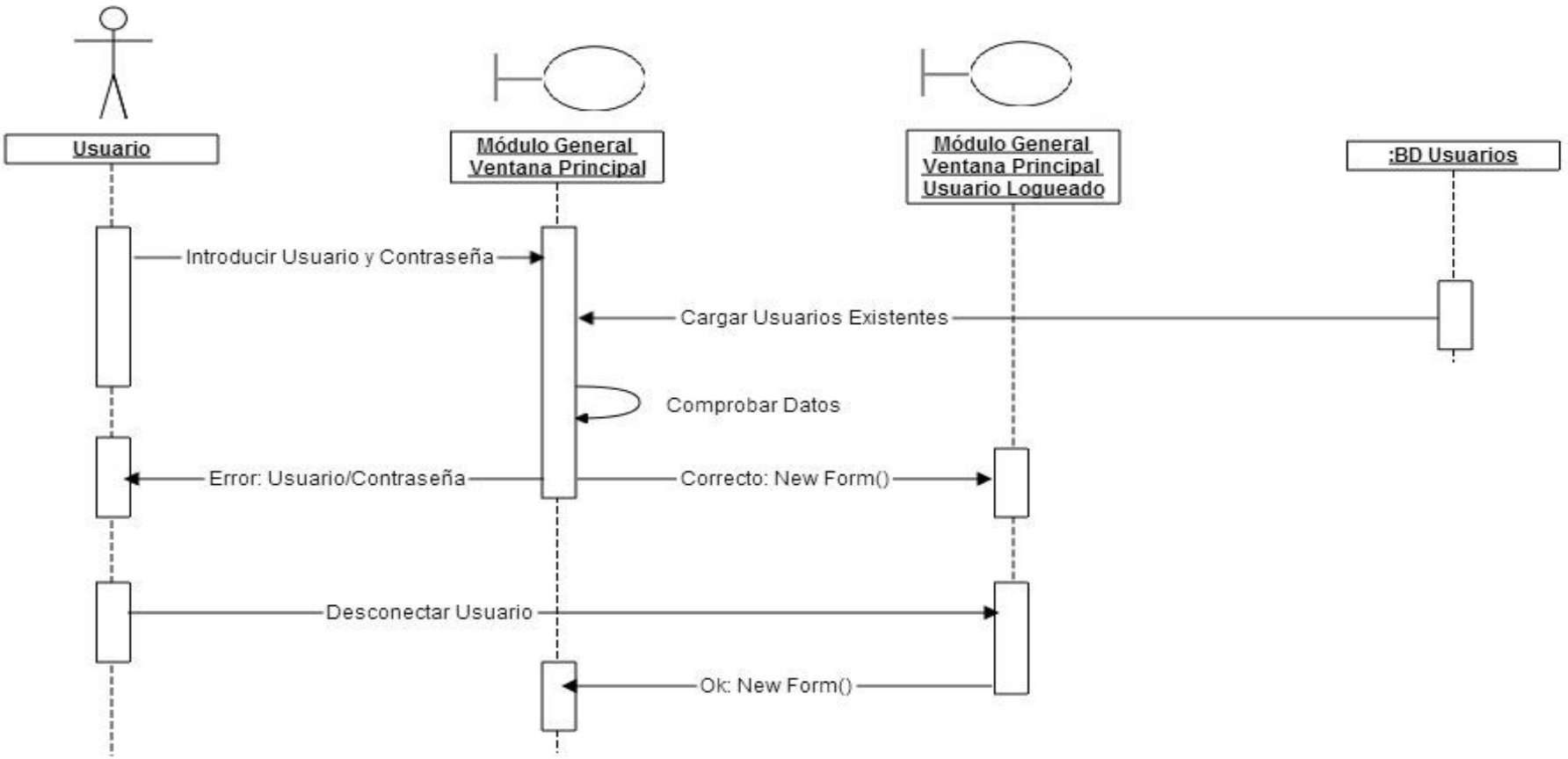


Ilustración 48: Diagrama de secuencia de conectarse y desconectarse como Usuario

Volver a la Referencia

Ver información de Usuario

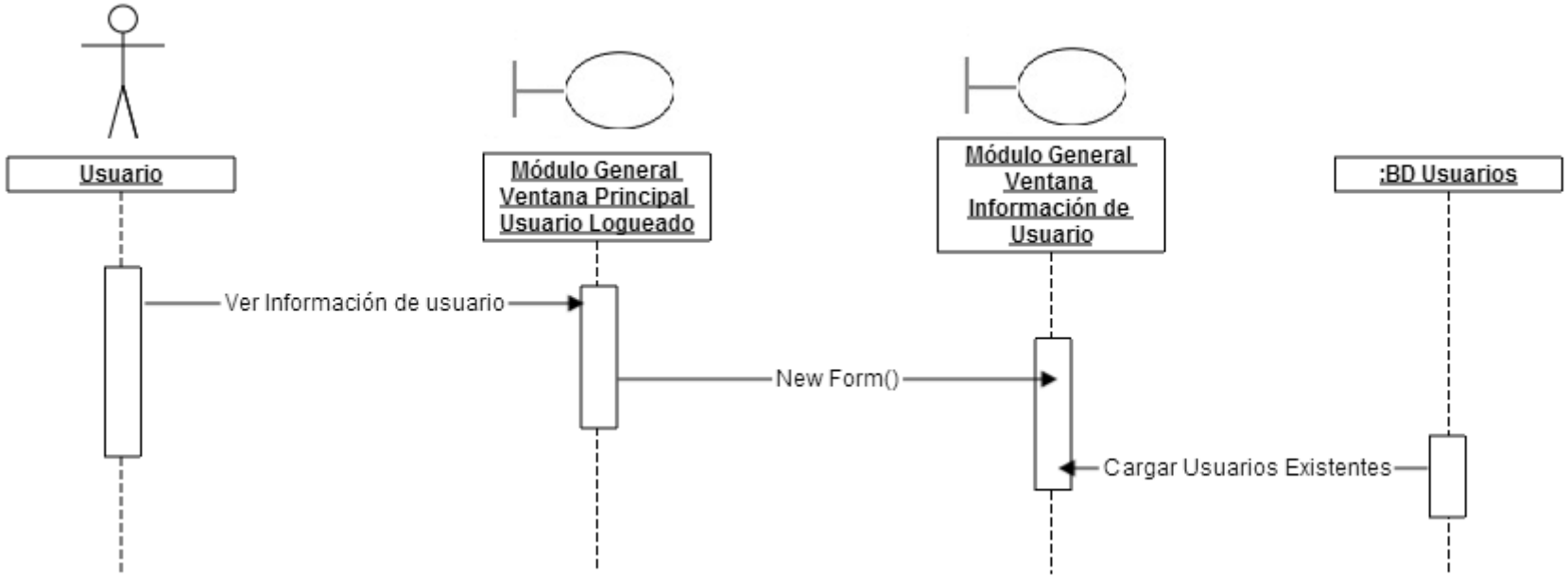


Ilustración 49: Diagrama de secuencia de visualizar la información personal de un Usuario

Volver a la Referencia

Cargar Módulo

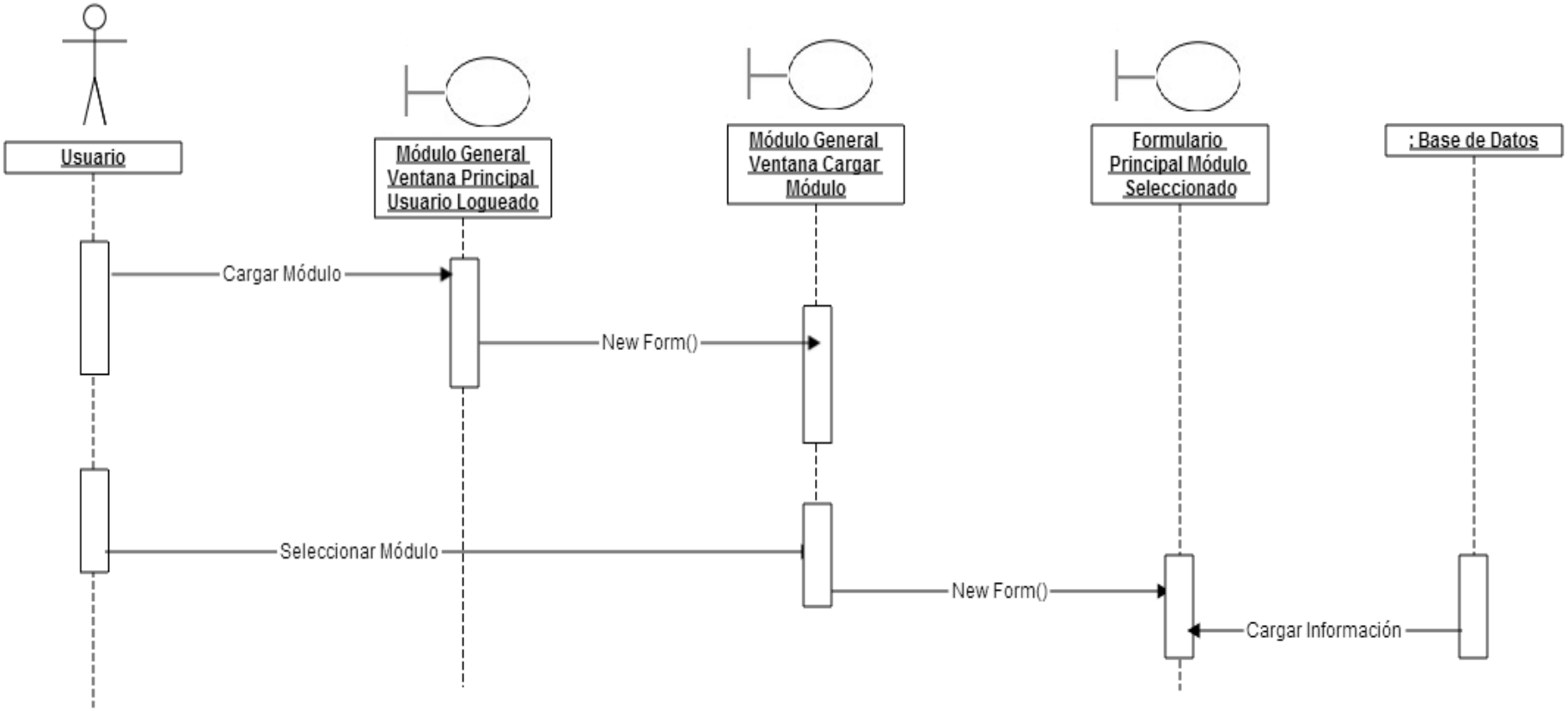


Ilustración 50: Diagrama de secuencia correspondiente a cargar un módulo seleccionado por el usuario

Volver a la referencia

Cerrar Aplicación

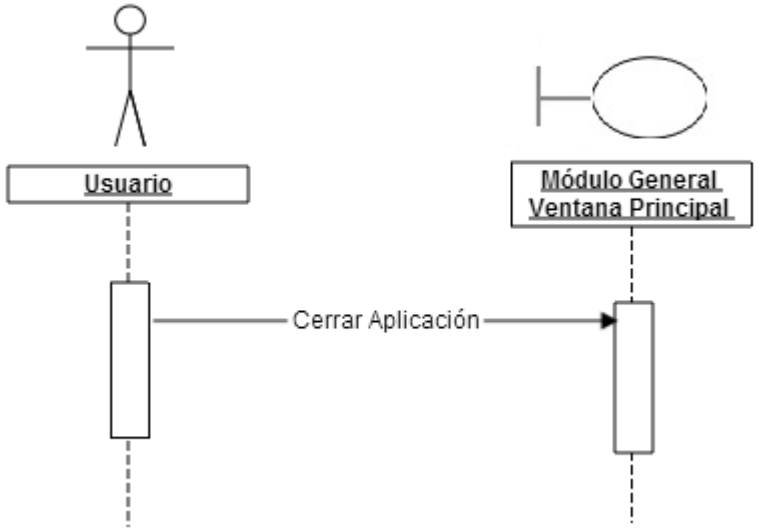


Ilustración 51: Diagrama de secuencia correspondiente a cerrar la aplicación

Volver a la referencia

7.3.2 Módulo Ejercicios

Crear Ejercicio

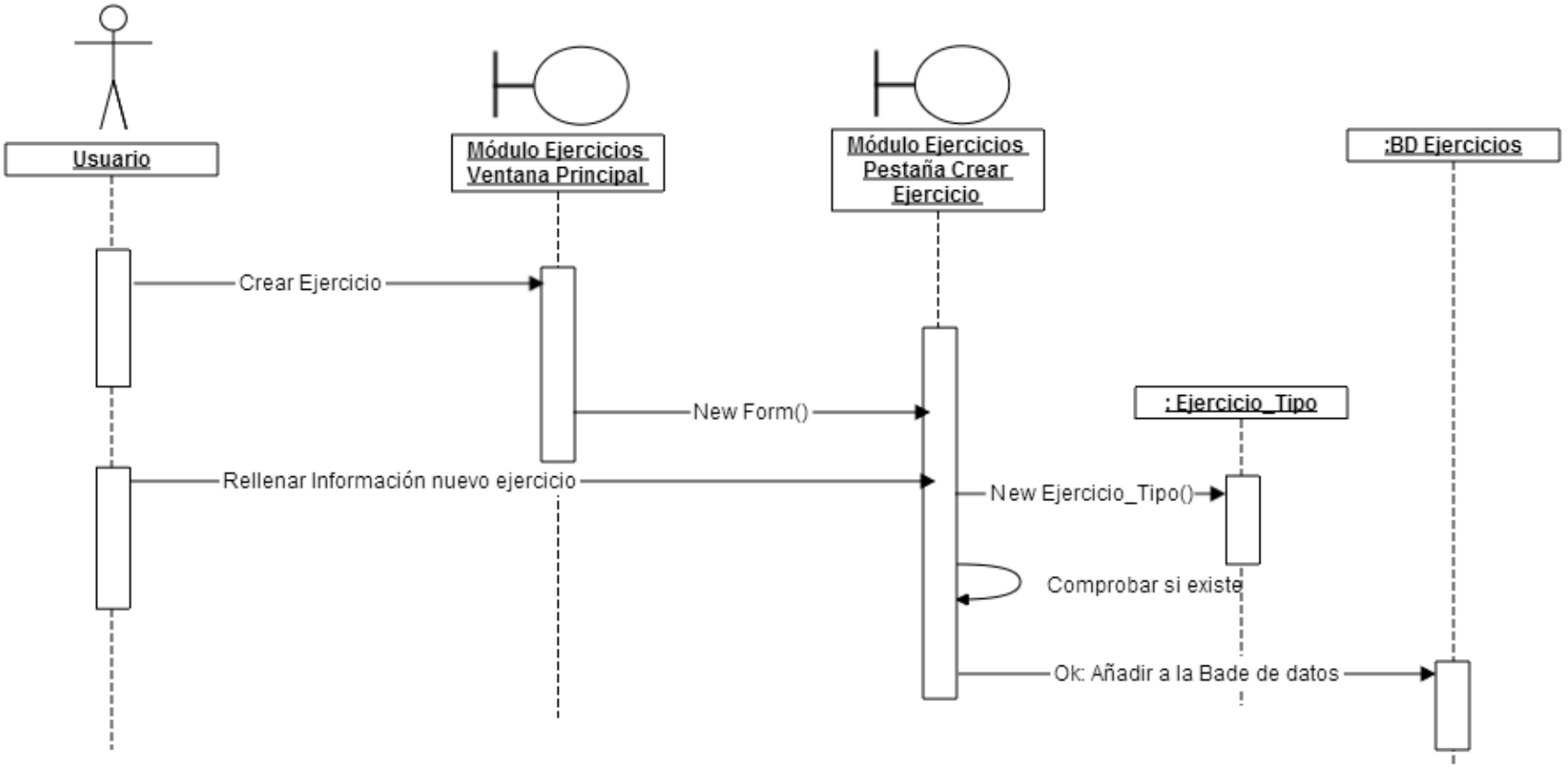


Ilustración 52: Diagrama de secuencia correspondiente a crear un nuevo ejercicio

Volver a la referencia

Abrir Ejercicio

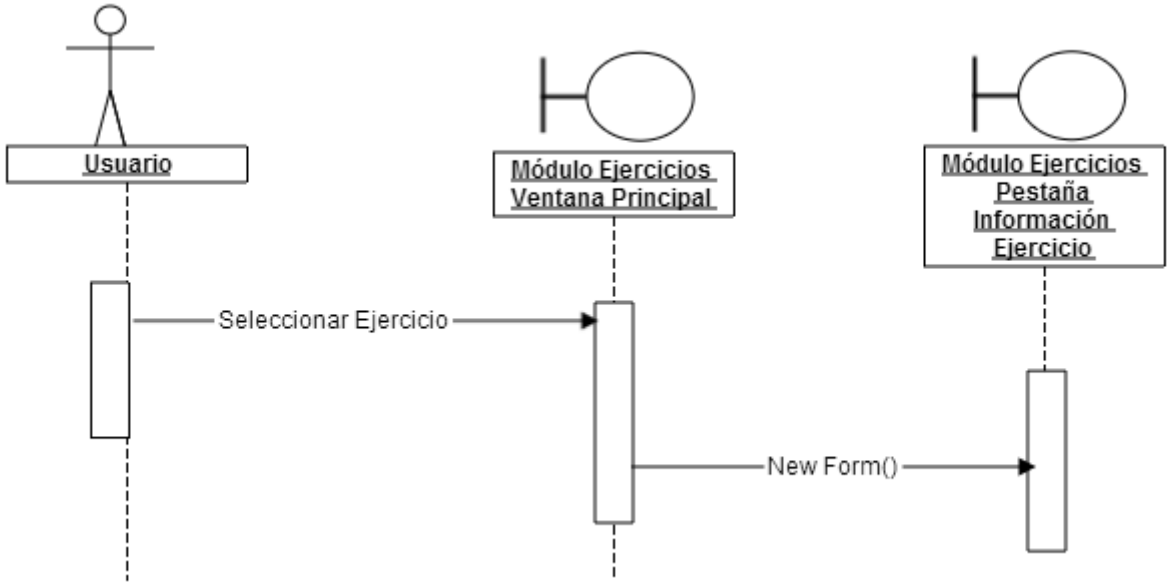


Ilustración 53: Diagrama de secuencia correspondiente a Abrir un ejercicio sin posibilidad de edición

Volver a la referencia

Modificar Ejercicio

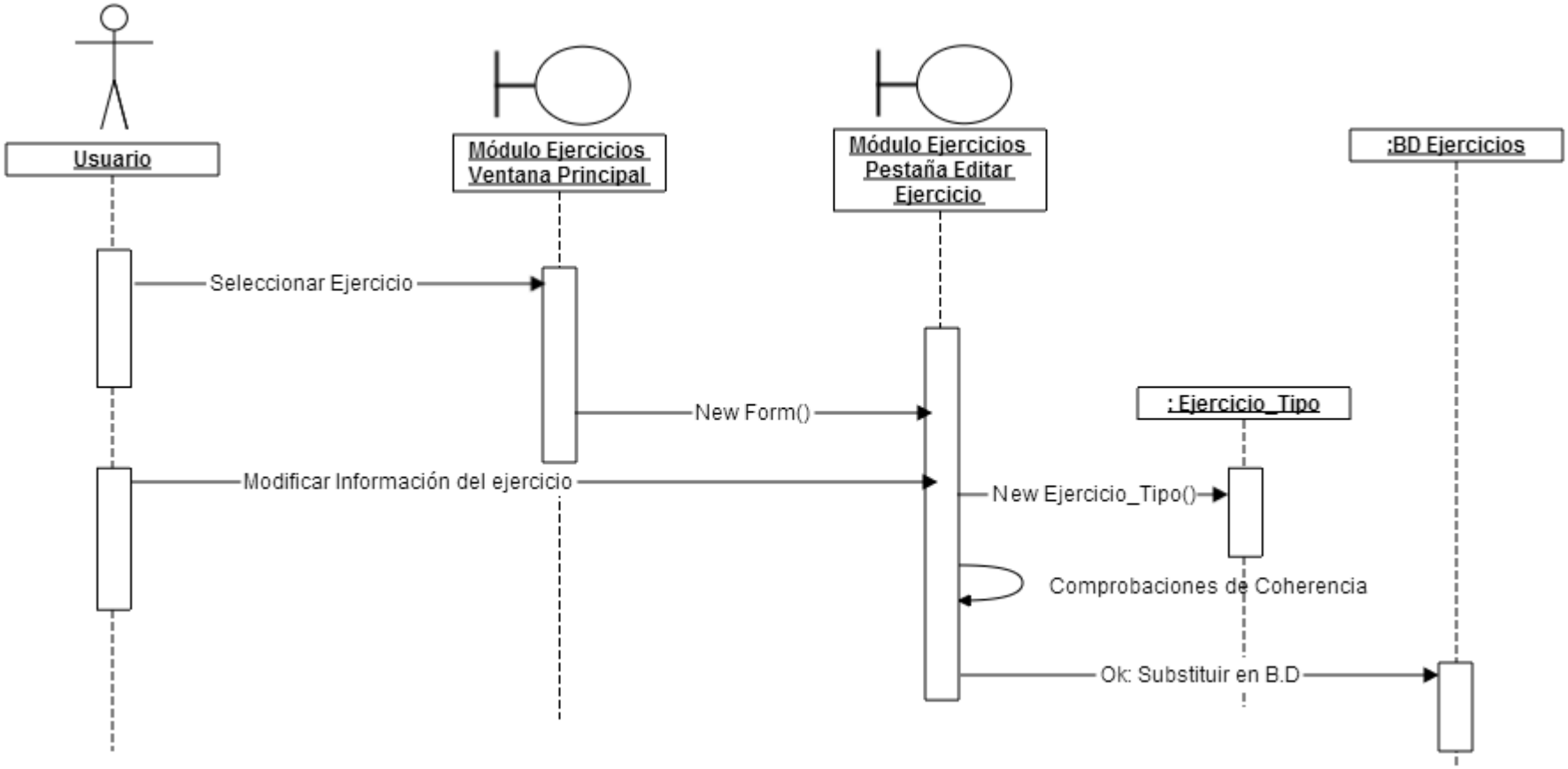


Ilustración 54: Diagrama de Secuencia correspondiente a modificar las opciones de un ejercicio existente

Volver a la referencia

Eliminar Ejercicio

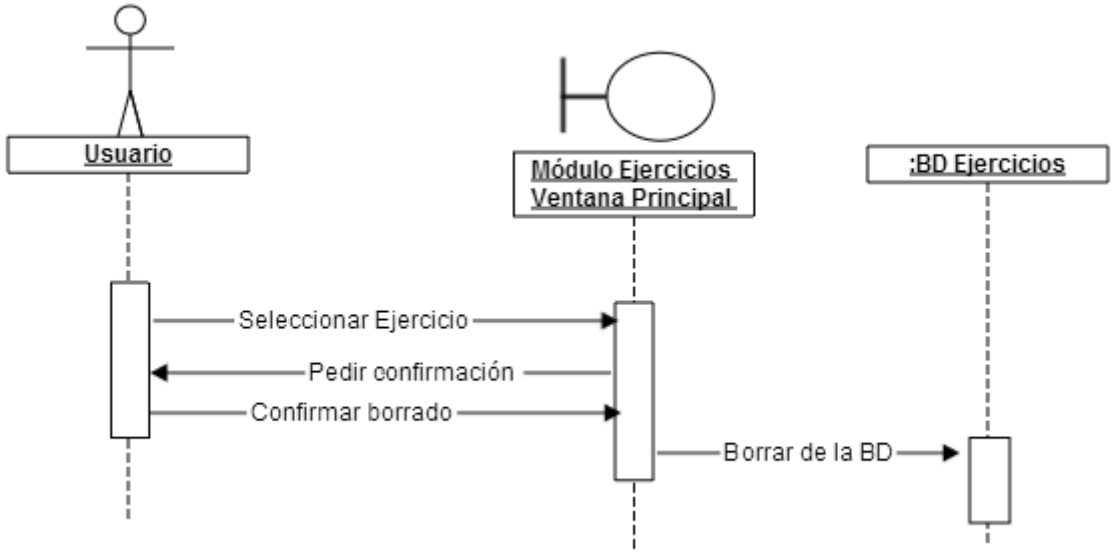


Ilustración 55: Diagrama de secuencia correspondiente a eliminar un ejercicio de la base de datos de ejercicios

Volver a la referencia

Añadir ejercicio a rutina

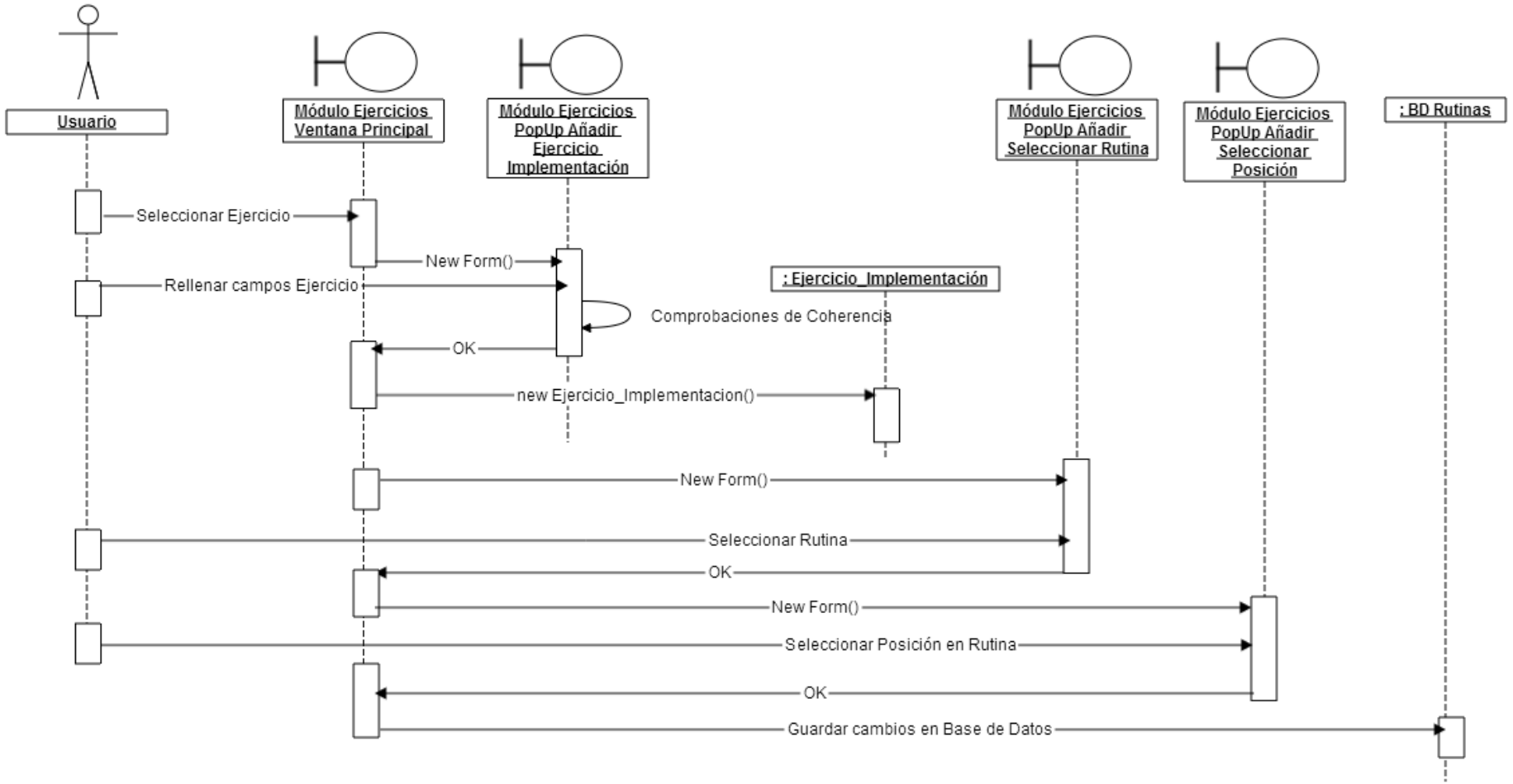


Ilustración 56: Diagrama de secuencia de las interacciones para añadir una ejecución de un ejercicio existente a una rutina Volver a la referencia

Crear Rutina

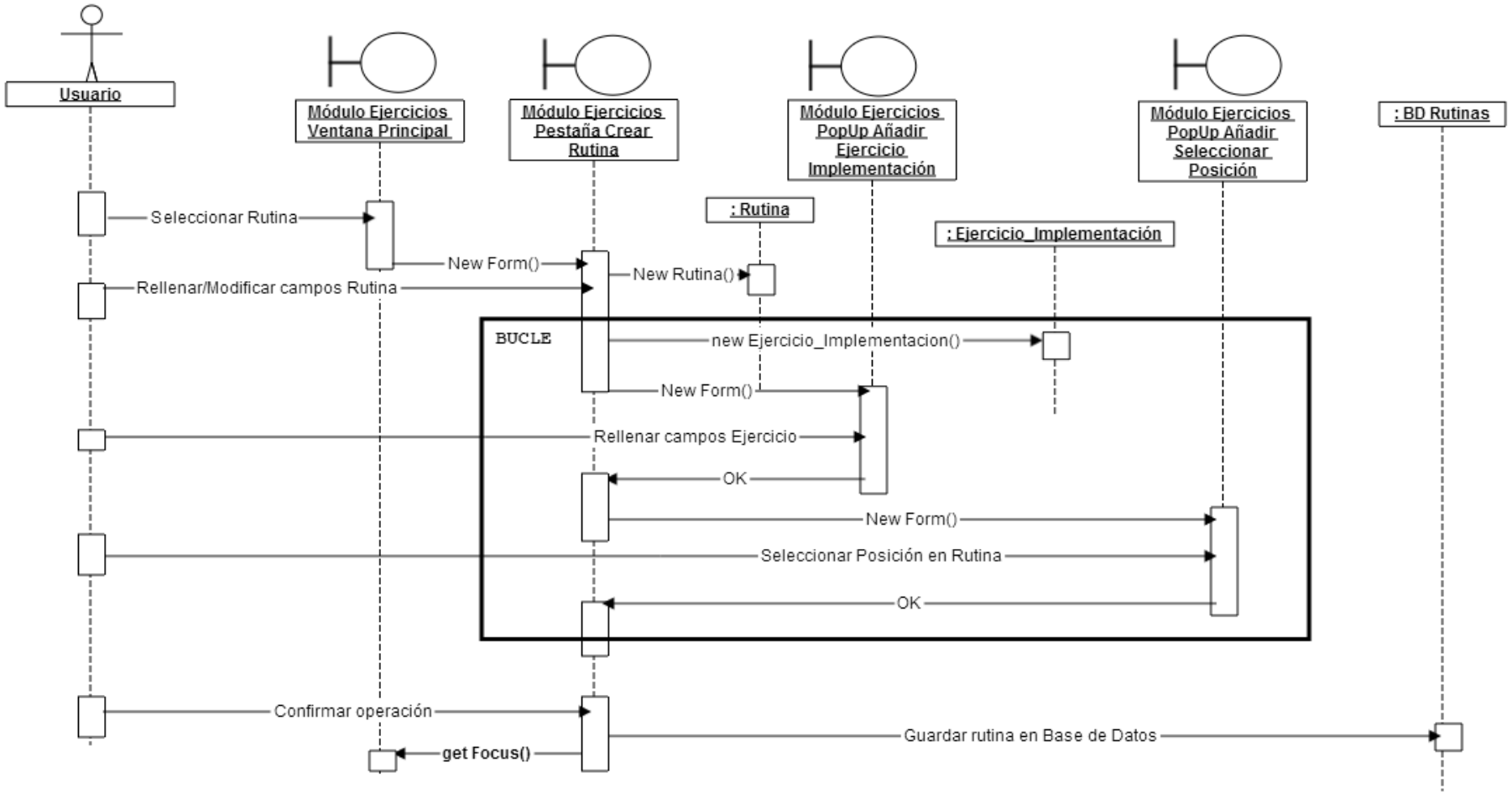


Ilustración 57: Diagrama de secuencia correspondiente a las interacciones entre elementos al crear una rutina nueva Volver a la referencia

Replicar Rutina

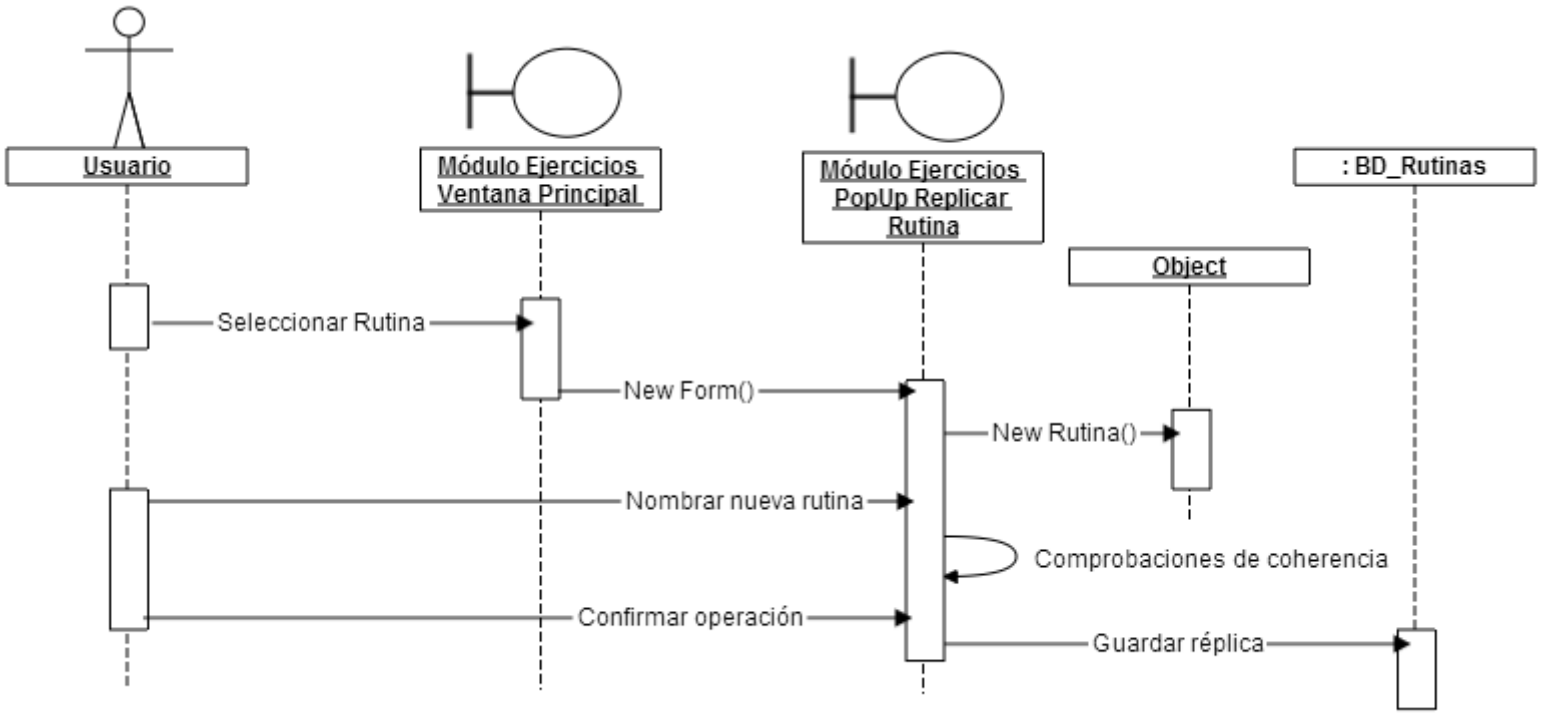


Ilustración 58: Diagrama de secuencia correspondiente a replicar una rutina existente

Volver a la referencia

Guardar Rutina como Imagen

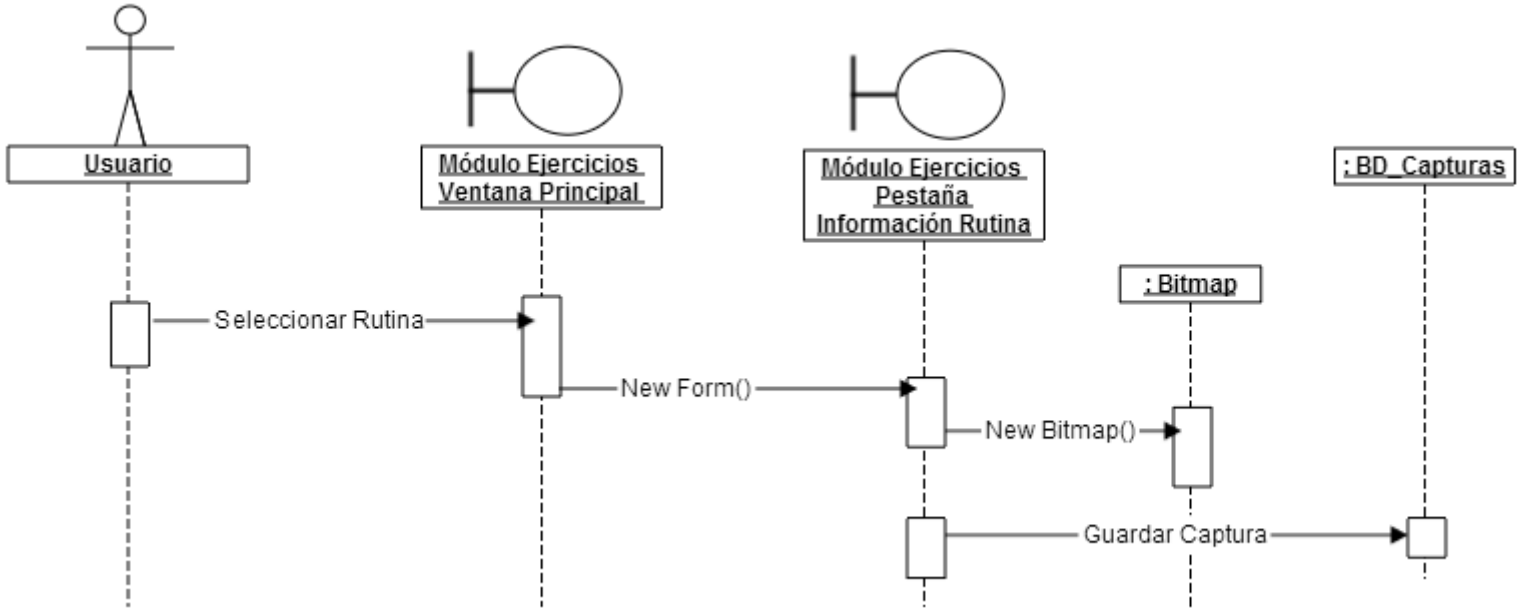


Ilustración 59: Diagrama de secuencia correspondiente a guardar una rutina como imagen

Volver a la referencia

Cerrar Módulo

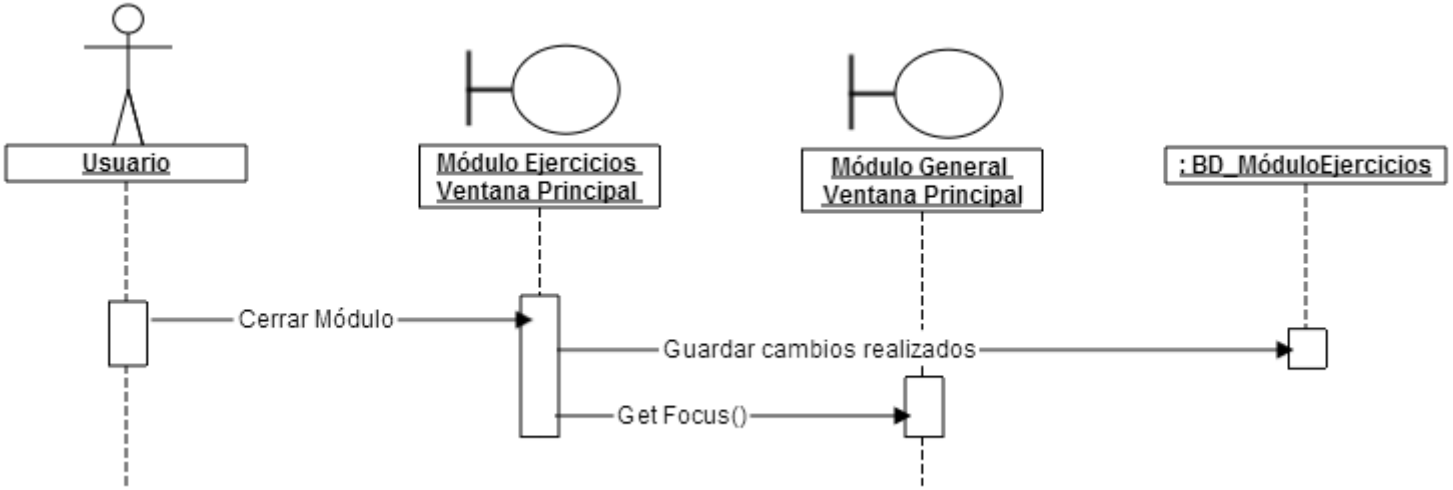


Ilustración 60: Diagrama de secuencia de cerrar el módulo abierto

Volver a la referencia

7.3.3 Módulo Dietista

Crear Alimento

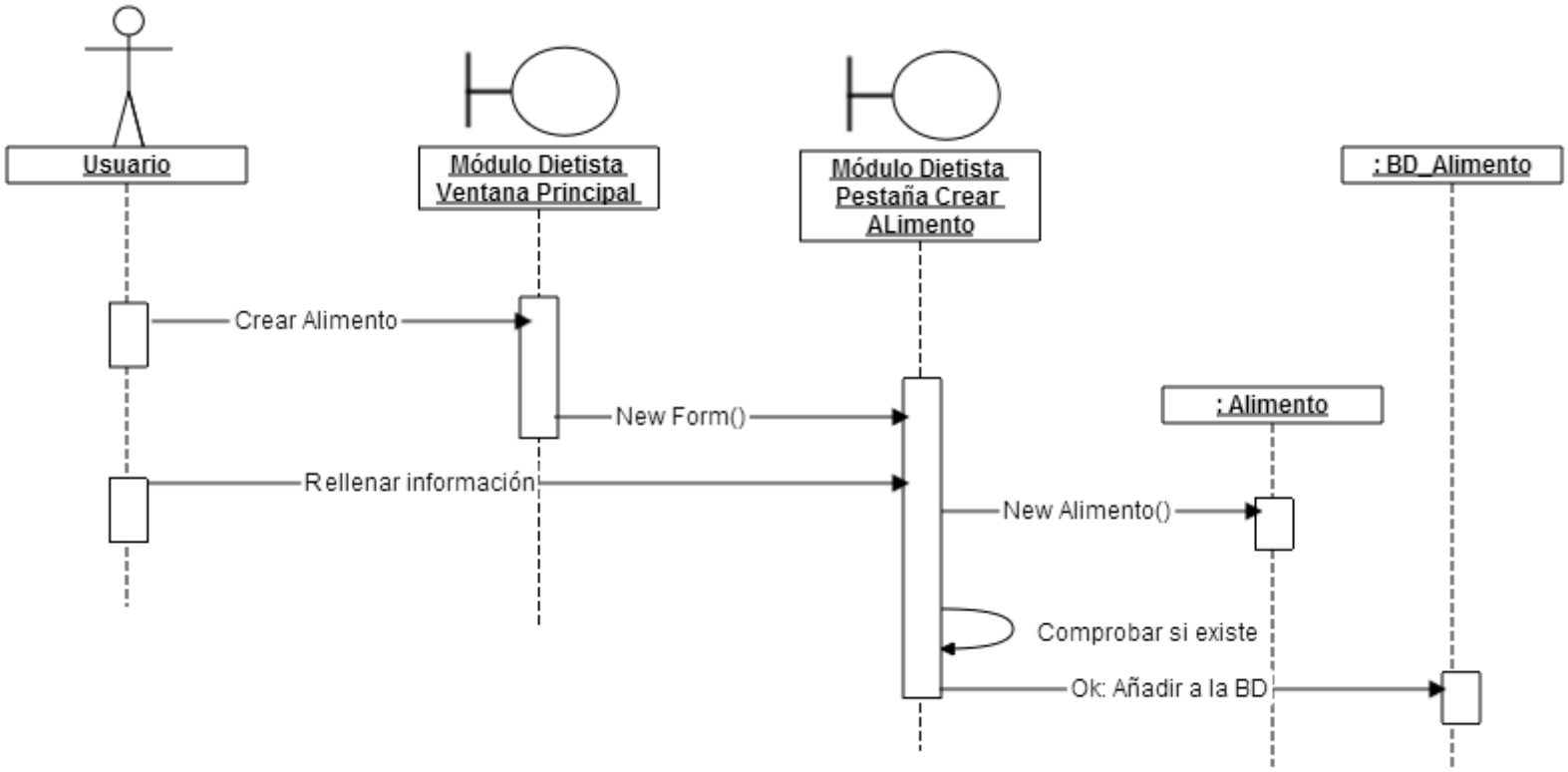


Ilustración 61: Diagrama de secuencia correspondiente a crear un nuevo alimento

Volver a la Referencia

Modificar Alimento

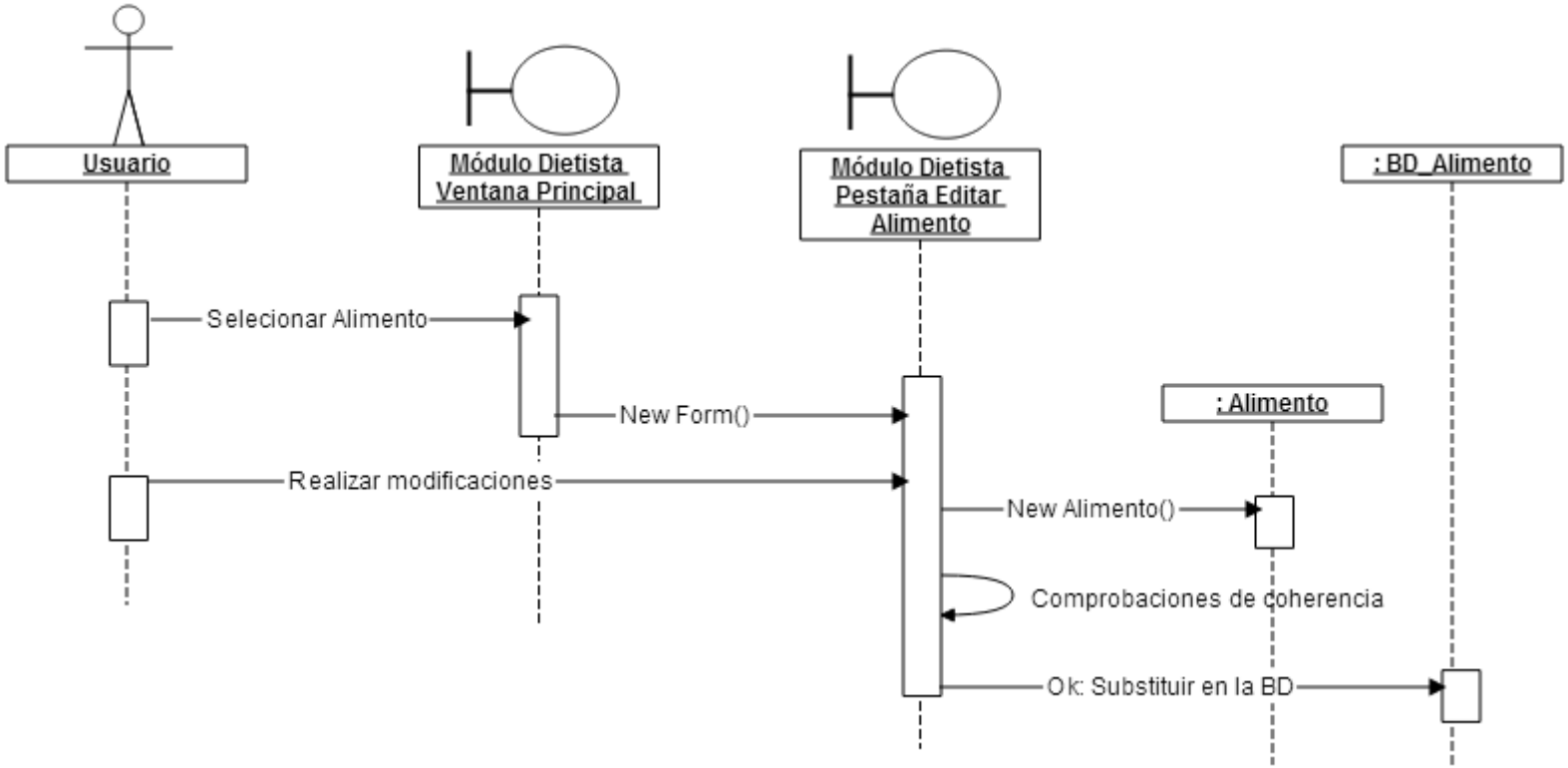


Ilustración 62: Diagrama de secuencia correspondiente a modificar un alimento existente

Volver a la Referencia

Eliminar Alimento

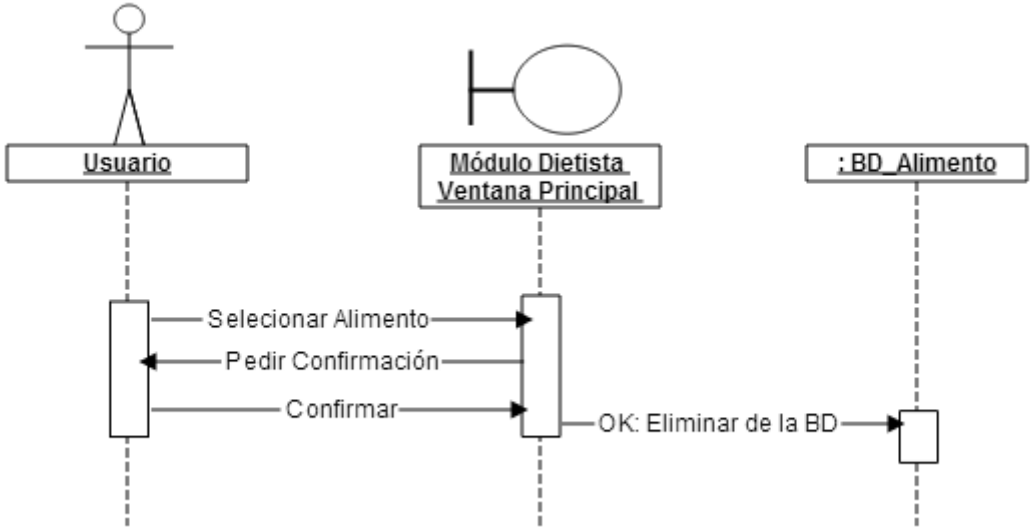


Ilustración 63: Diagrama de secuencia correspondiente a eliminar un alimento existente

Volver a la Referencia

Añadir alimento a comida

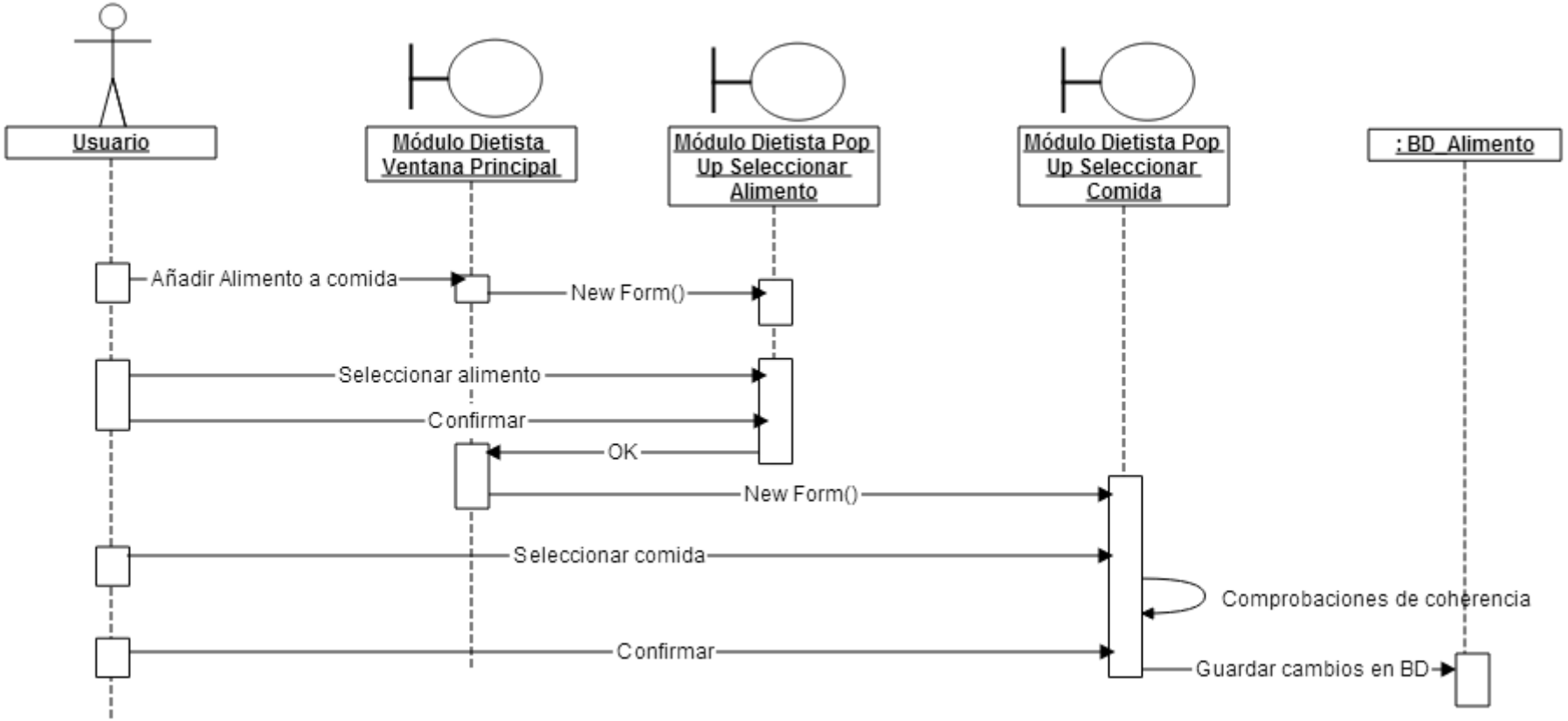


Ilustración 64: Diagrama de secuencia correspondiente a añadir un alimento a una comida existente

Volver a la Referencia

Crear/Modificar Dieta

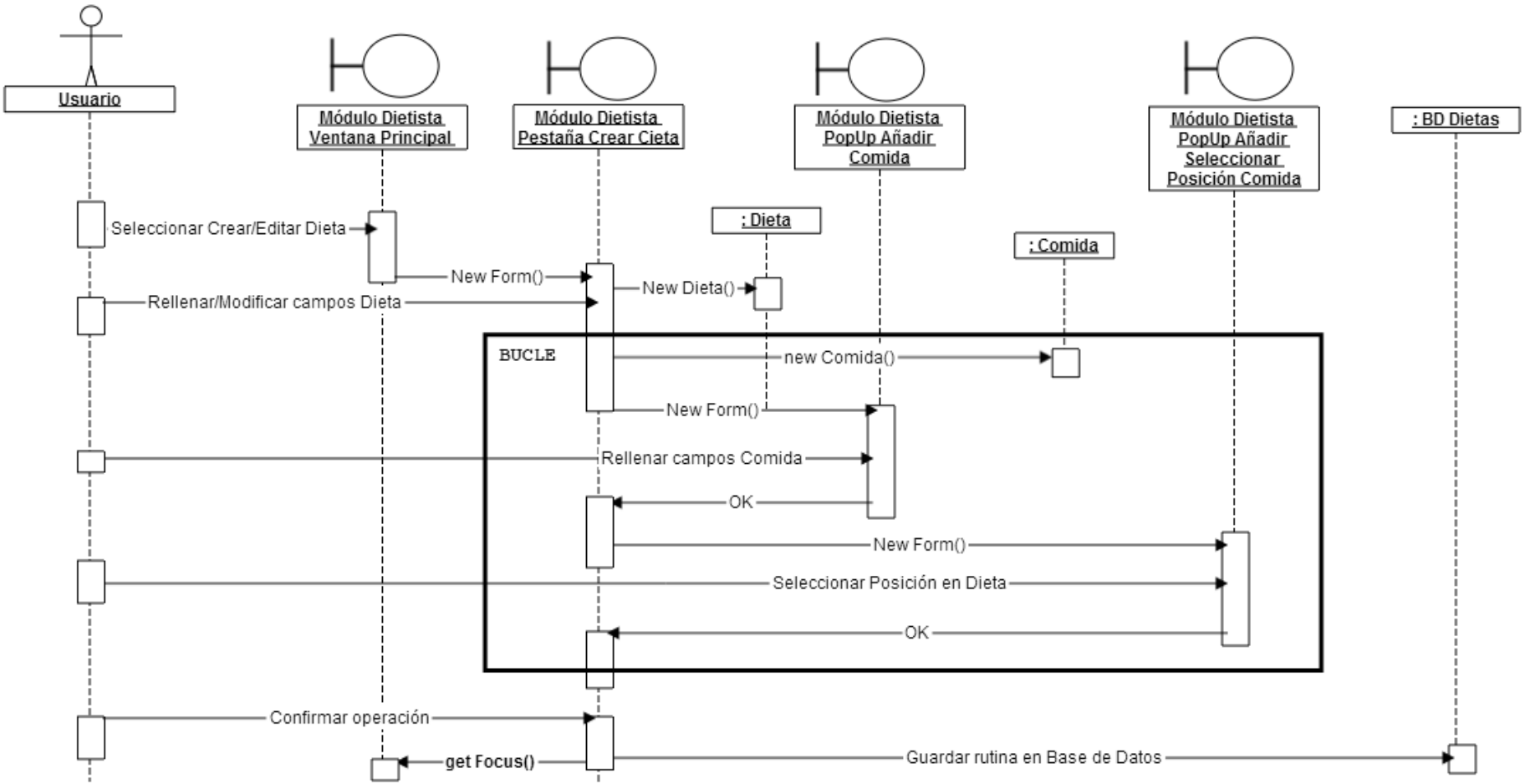
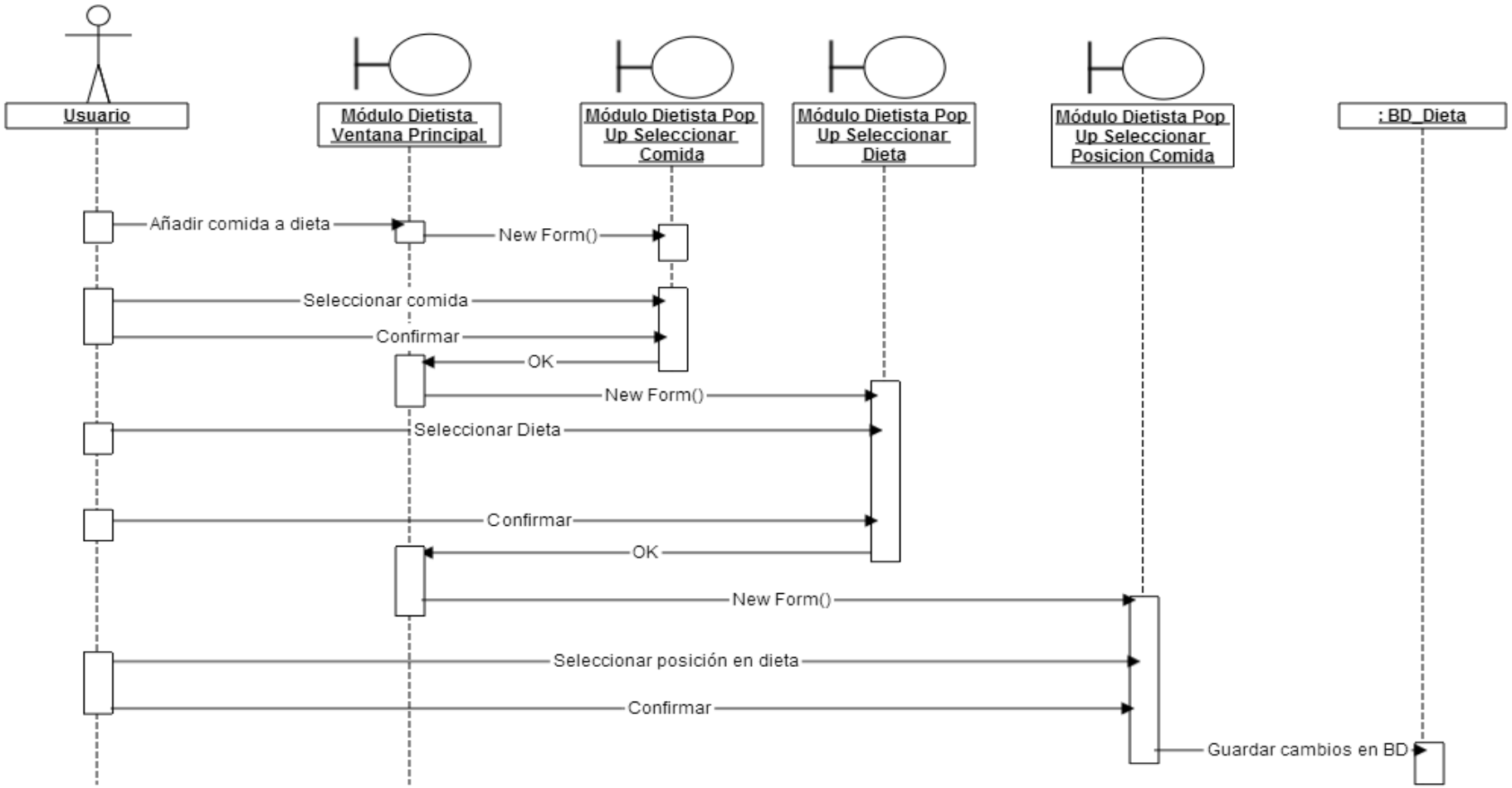


Ilustración 65: Diagrama de secuencia correspondiente a crear una dieta Volver a la Referencia

Añadir comida a dieta



Volver a la Referencia

Ilustración 66: Diagrama de secuencia correspondiente a añadir una comdia a una dieta existente

7.3.4 Módulo Calendario

Crear Calendario

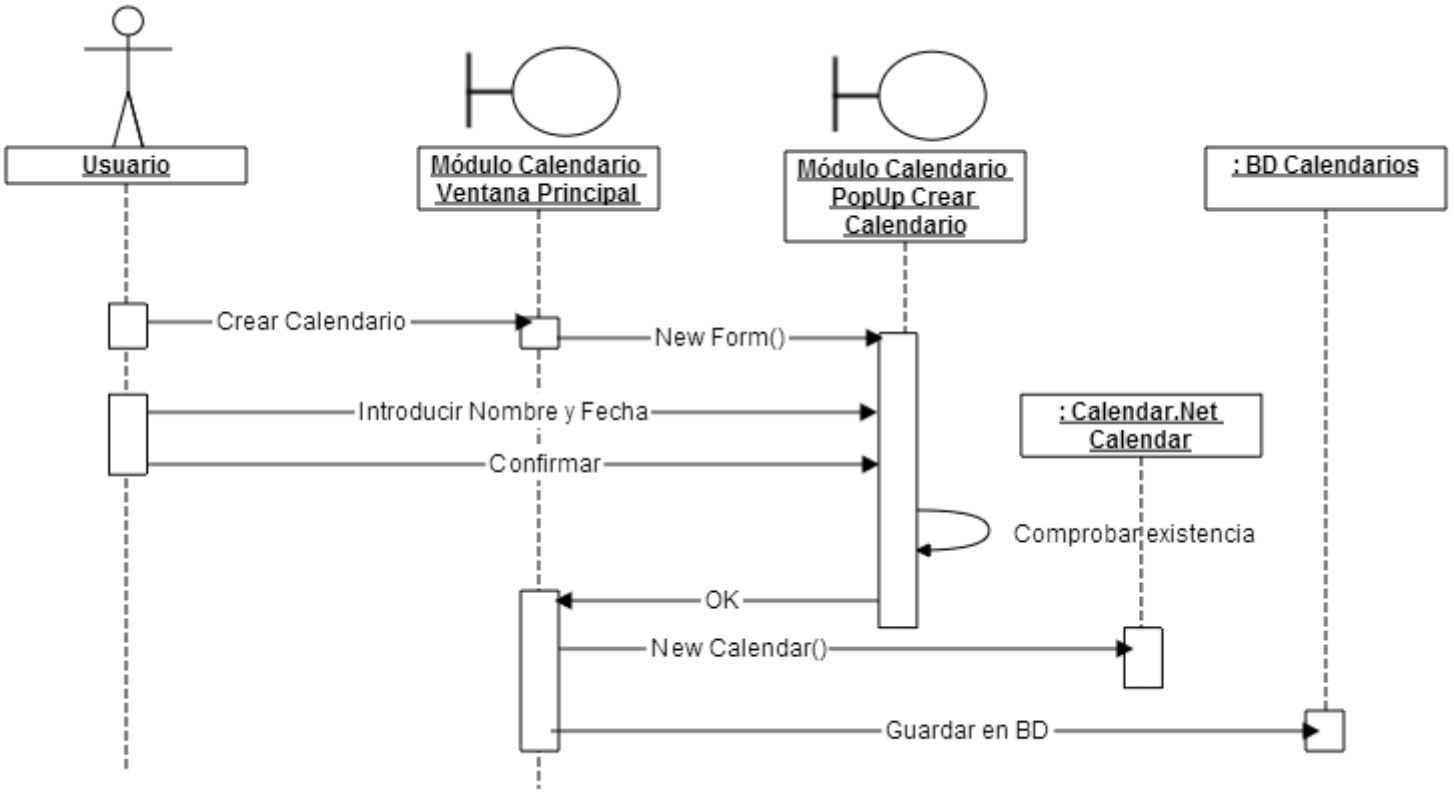
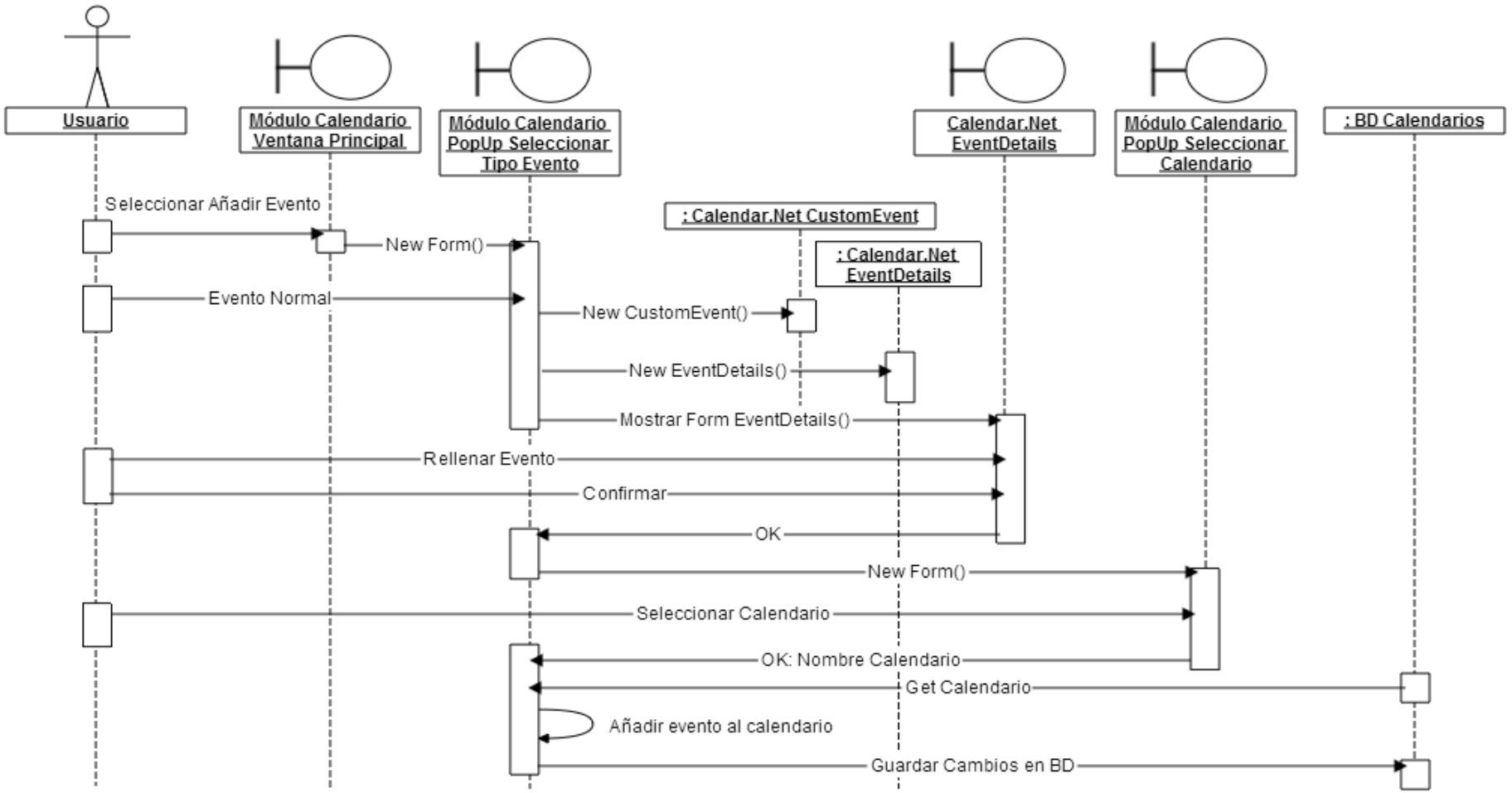


Ilustración 67: Diagrama de secuencia correspondiente a la creación de un nuevo calendario

Volver a la Referencia

Añadir evento Normal



Volver a la Referencia

Ilustración 68: Diagrama de secuencia correspondiente a añadir un evento del tipo normal

Añadir evento comida

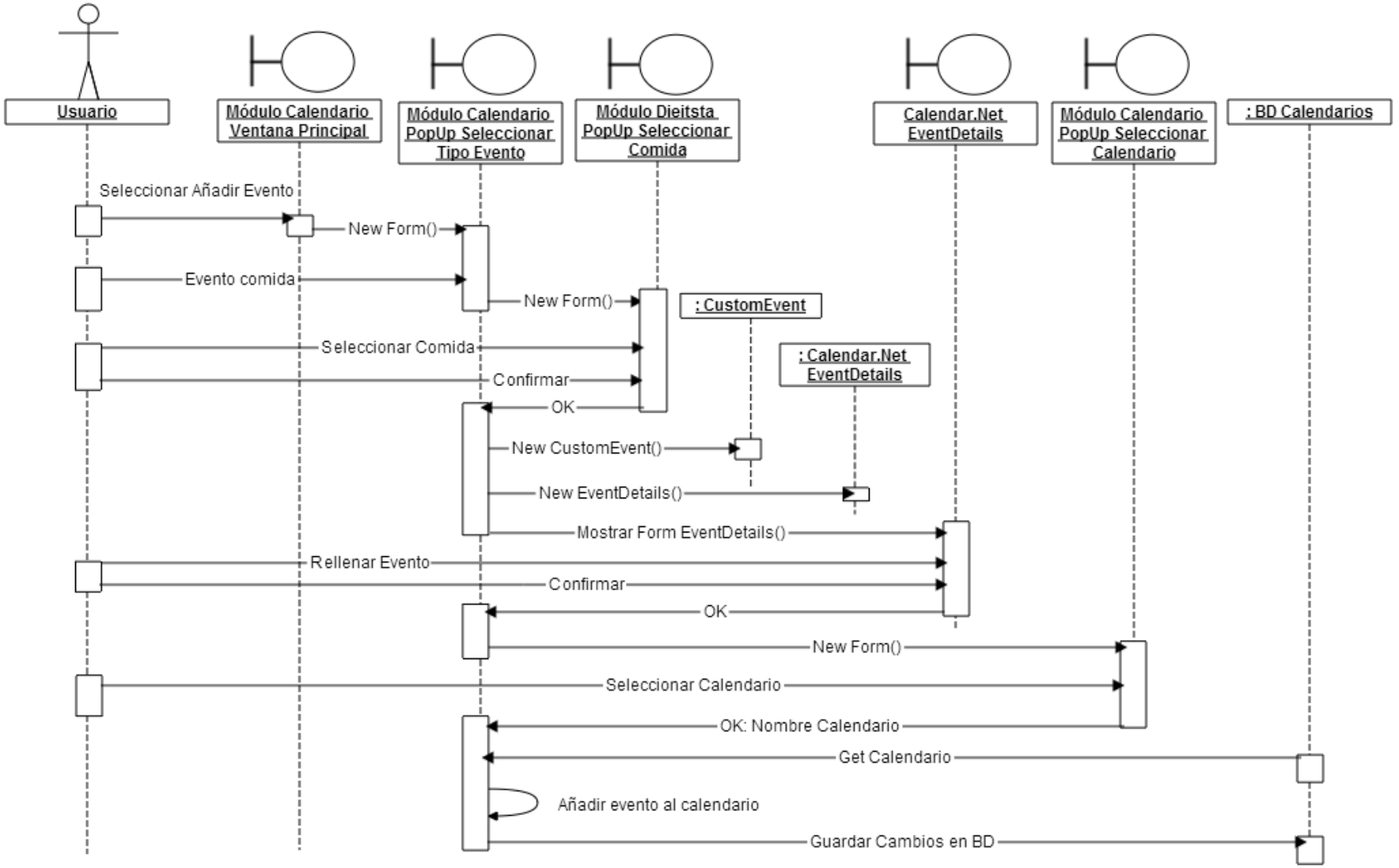


Ilustración 69: Diagrama de secuencia correspondiente a añadir un evento de tipo comida a un calendario existente

Volver a la Referencia

Añadir evento ejercicio

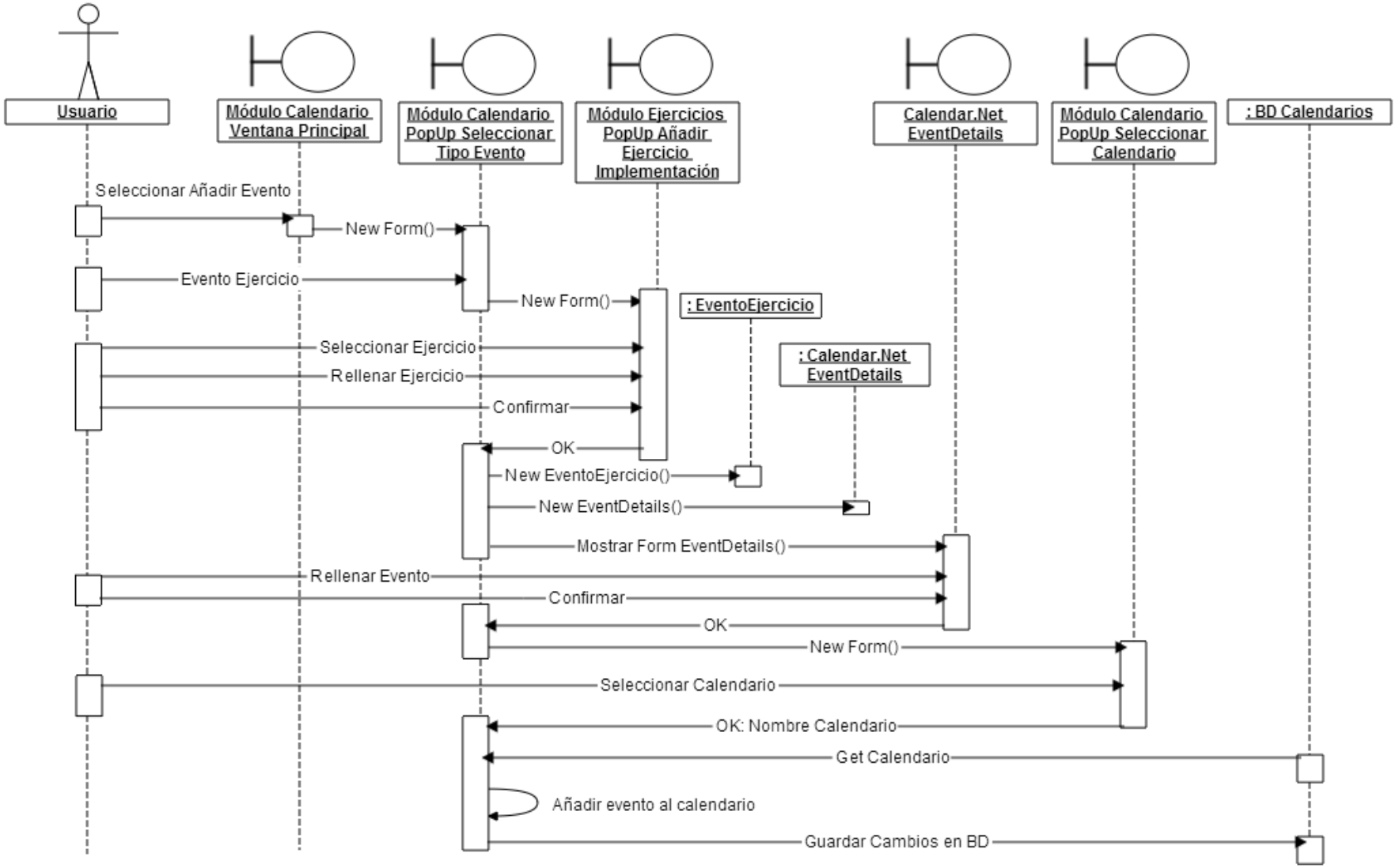


Ilustración 70: Diagrama de secuencia correspondiente a añadir un evento del tipo ejercicio a un calendario existente

Volver a la Referencia

Añadir evento dieta

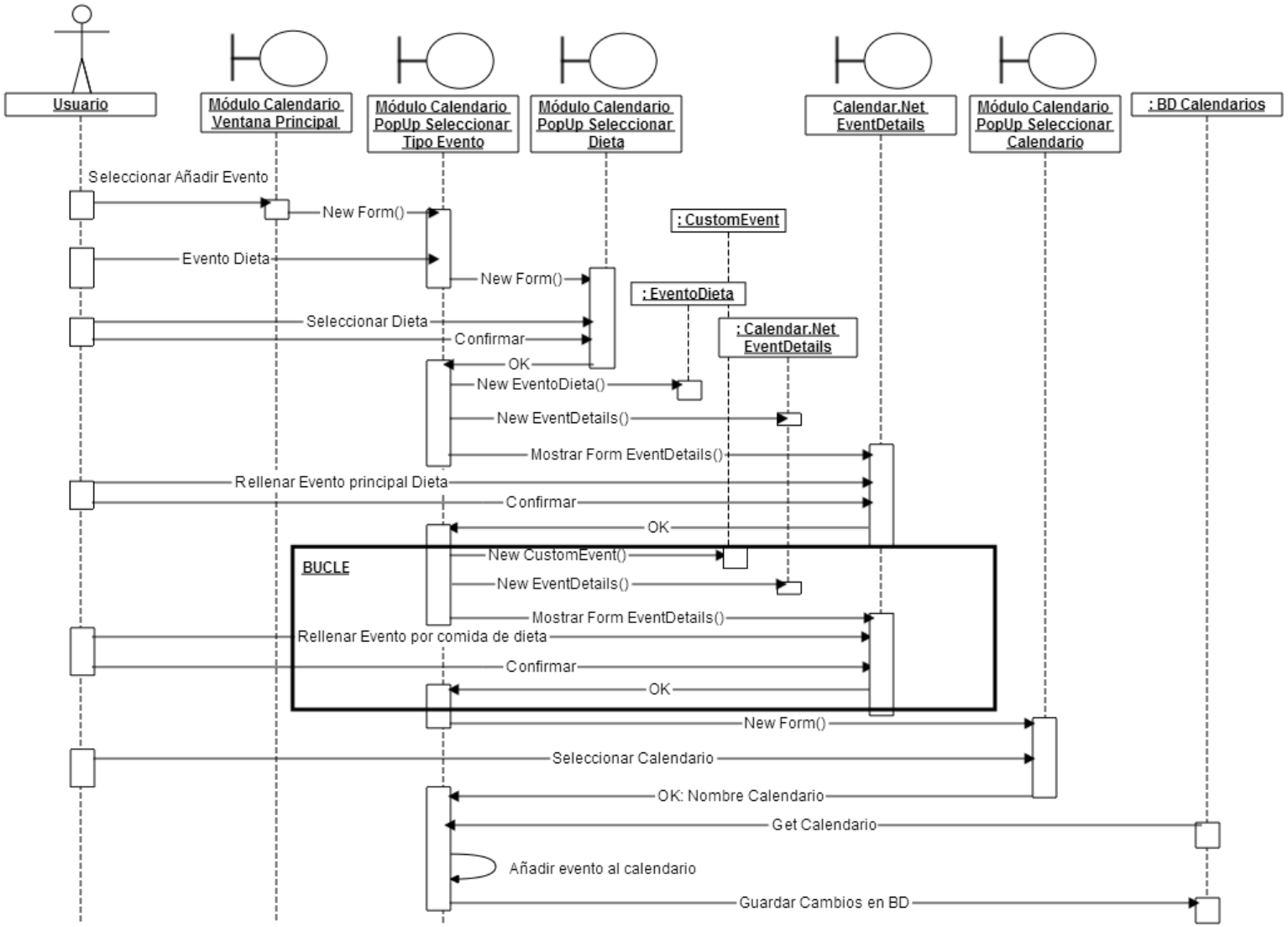


Ilustración 71: Diagrama de secuencia correspondiente a añadir un evento de tipo dieta a un calendario existente

Volver a la Referencia

Añadir evento rutina

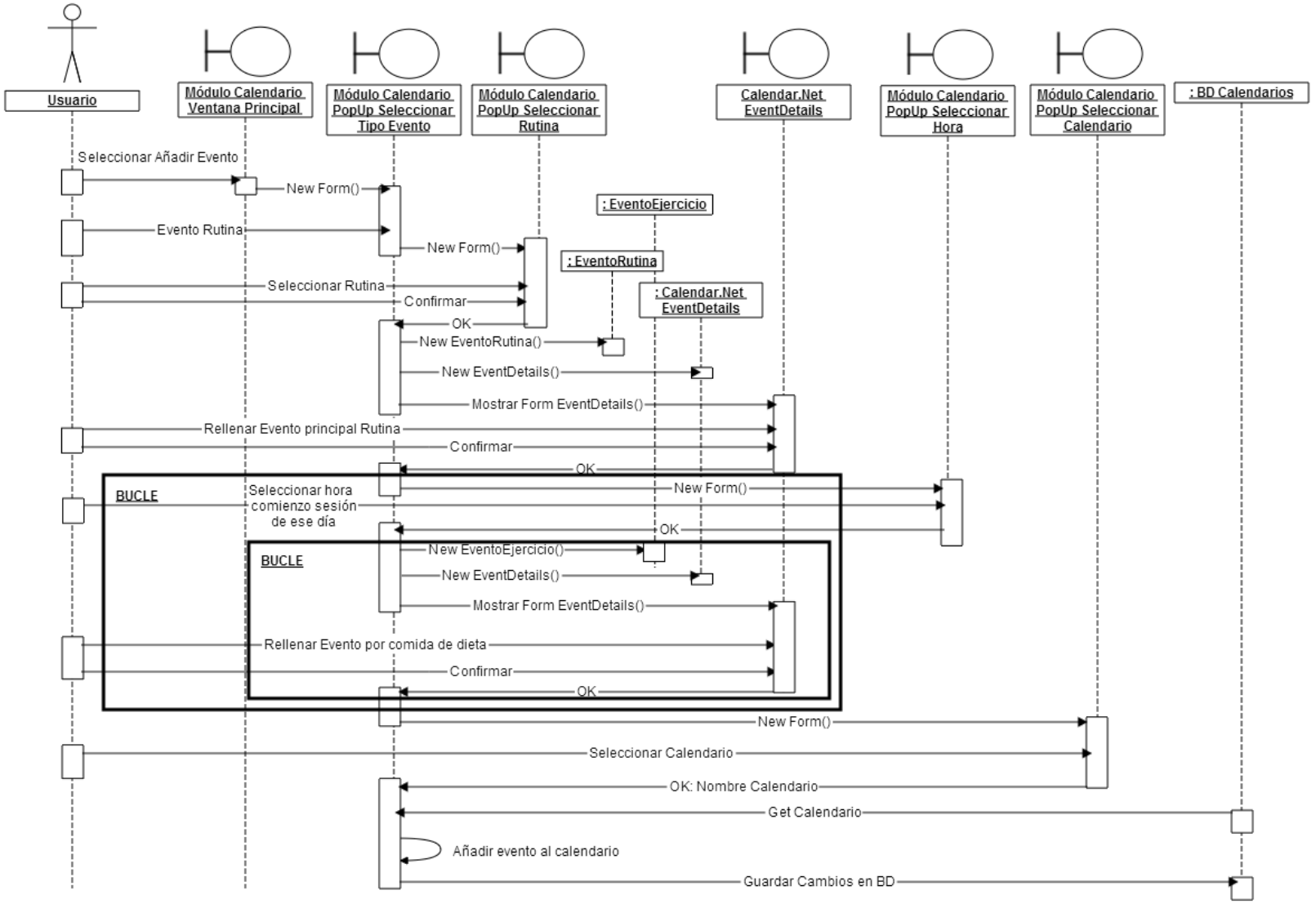


Ilustración 72: Diagrama de secuencia correspondiente a añadir un evento del tipo rutina a un calendario existente

Volver a la Referencia

Eliminar evento

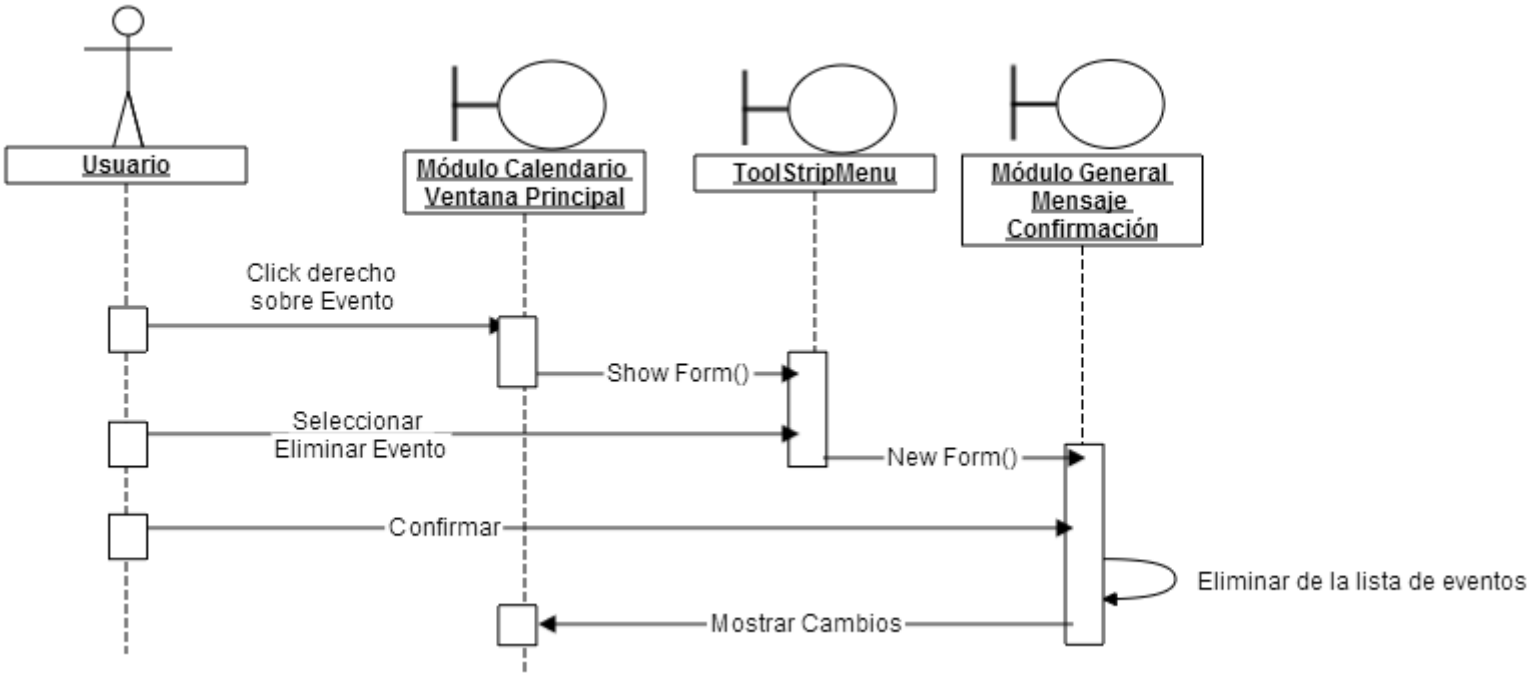


Ilustración 73: Diagrama de secuencia correspondiente a eliminar un evento existente

Volver a la Referencia

Replicar Evento

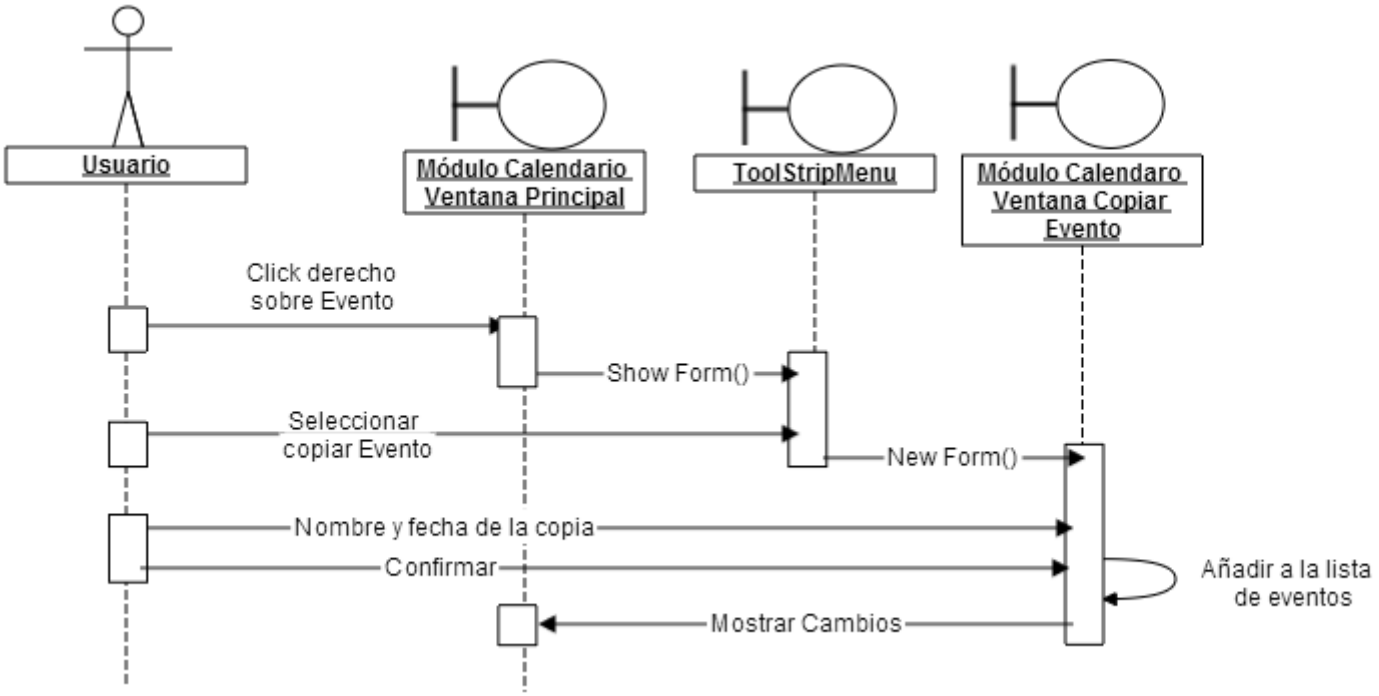


Ilustración 74: Diagrama de secuencia correspondiente a replicar un evento existente

Volver a la Referencia

Editar evento

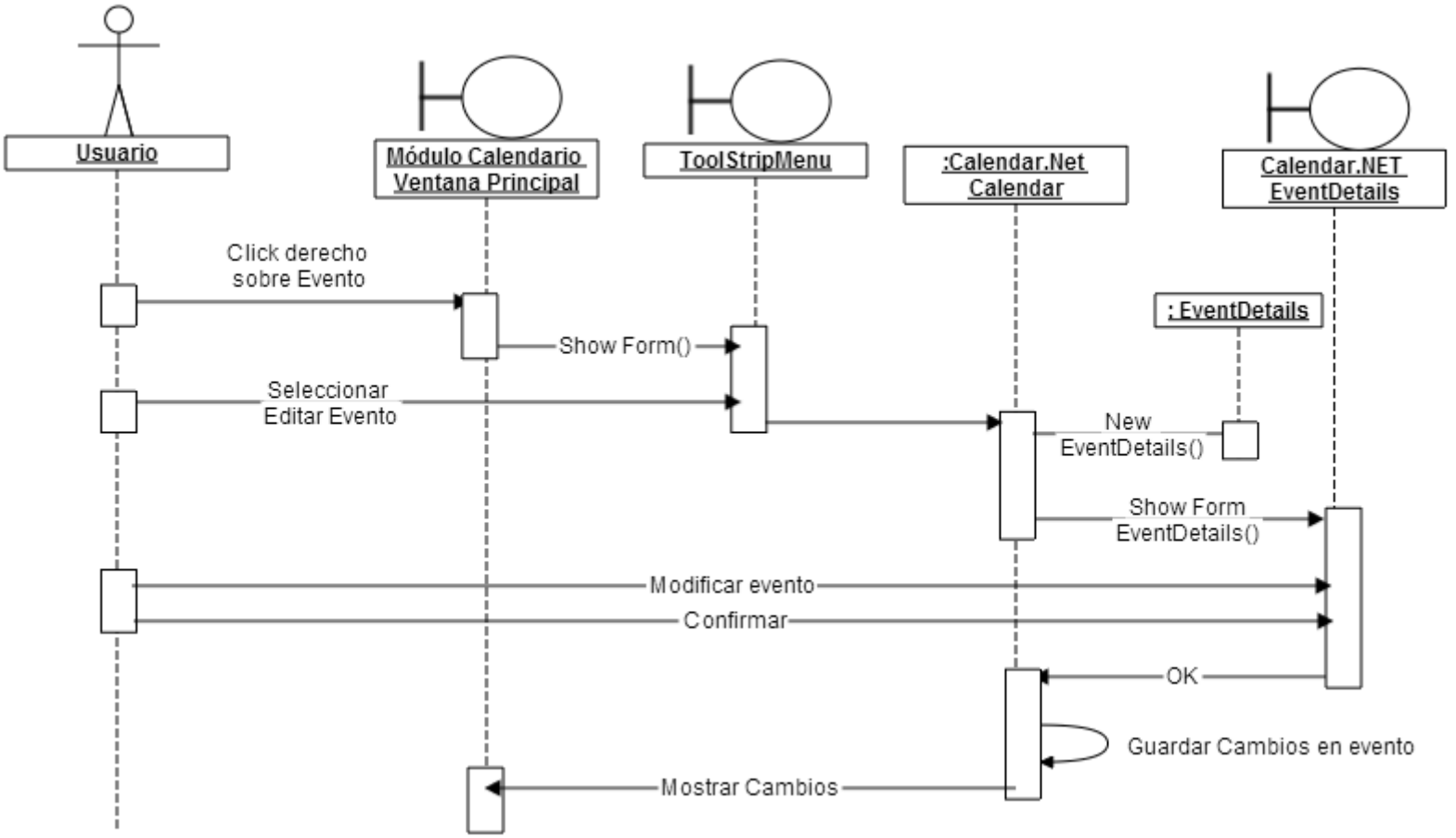


Ilustración 75: Diagrama de secuencia correspondiente a editar un evento existente

Volver a la Referencia

7.4 Interfaz (Diseño)

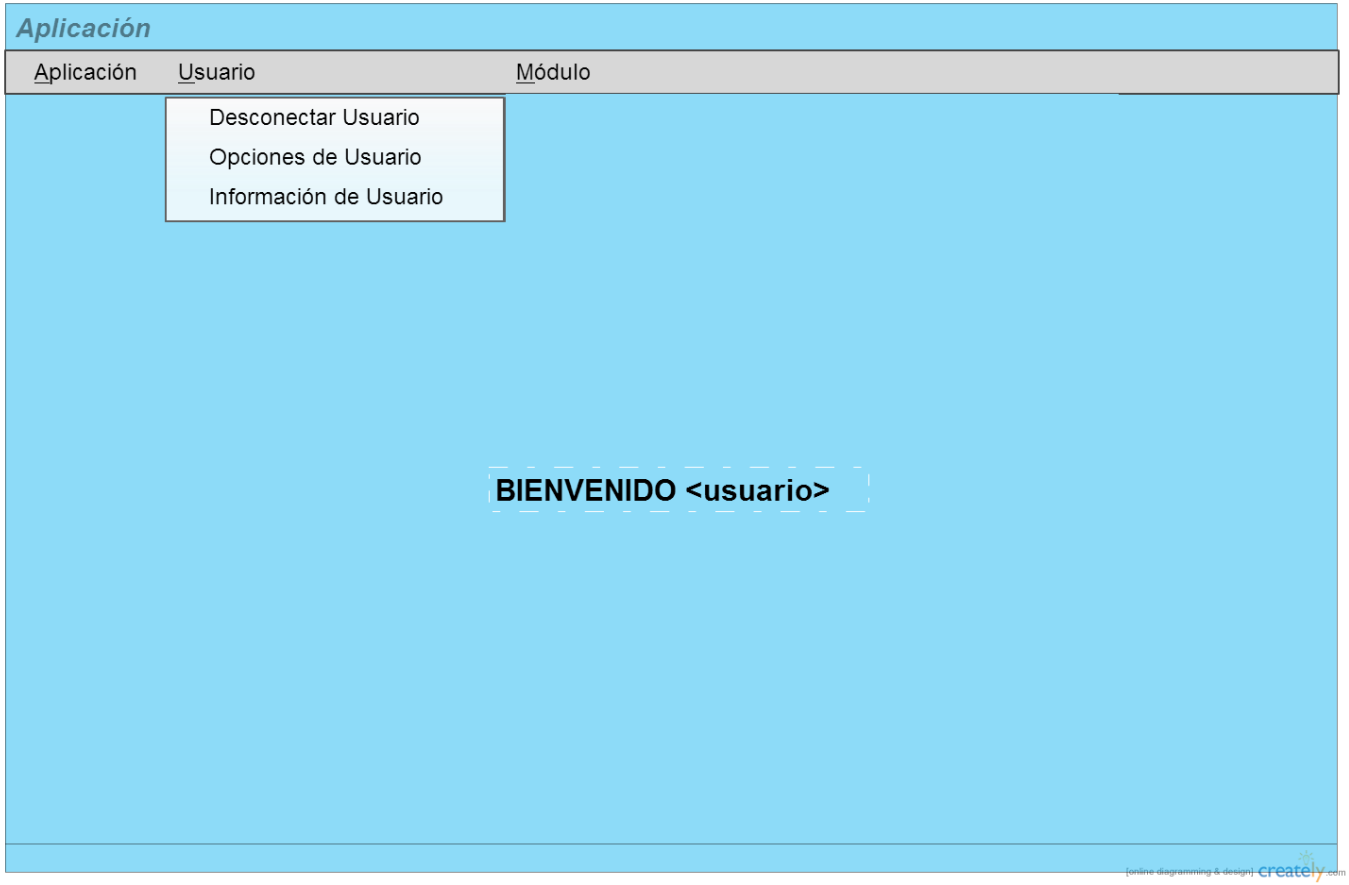


Ilustración 76: Módulo General ventana principal con usuario ya logueado

Volver a la Referencia

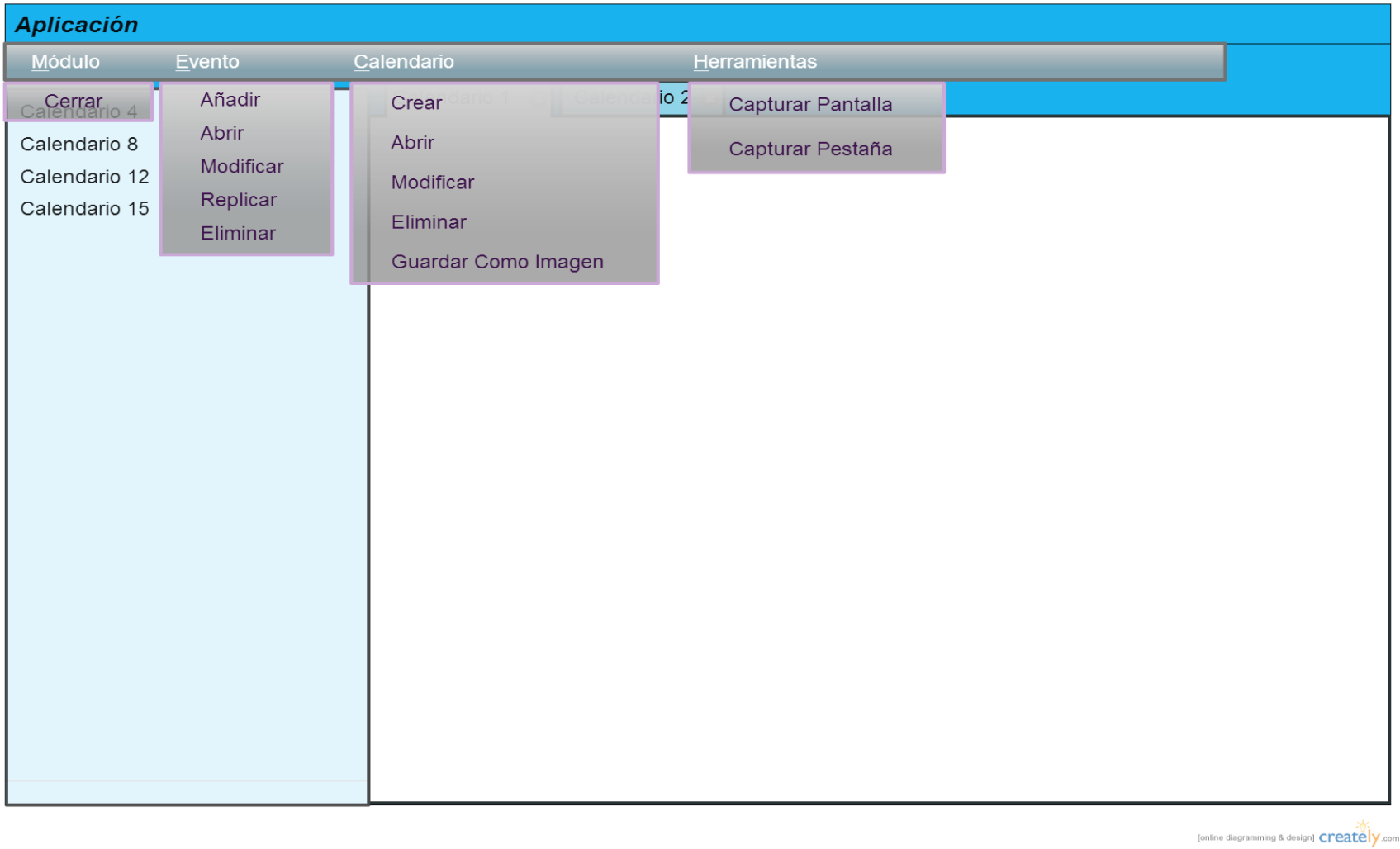


Ilustración 77: Diseño de la ventana principal del Módulo Calendario.

Volver a la Referencia

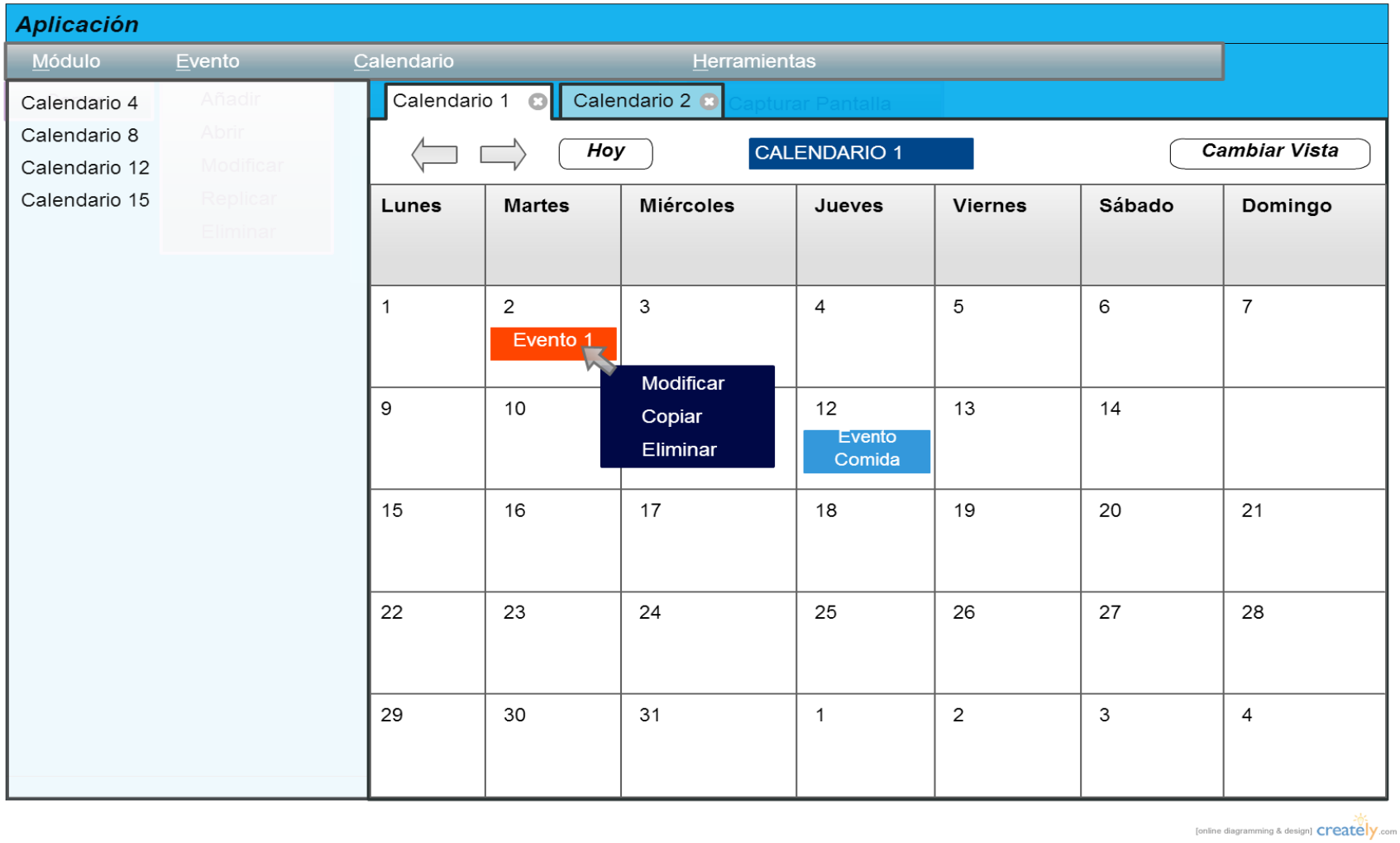


Ilustración 78: Diseño de la ventana principal del Módulo Calendario con un calendario abierto en modo *Vista mensual.*

Volver a la Referencia

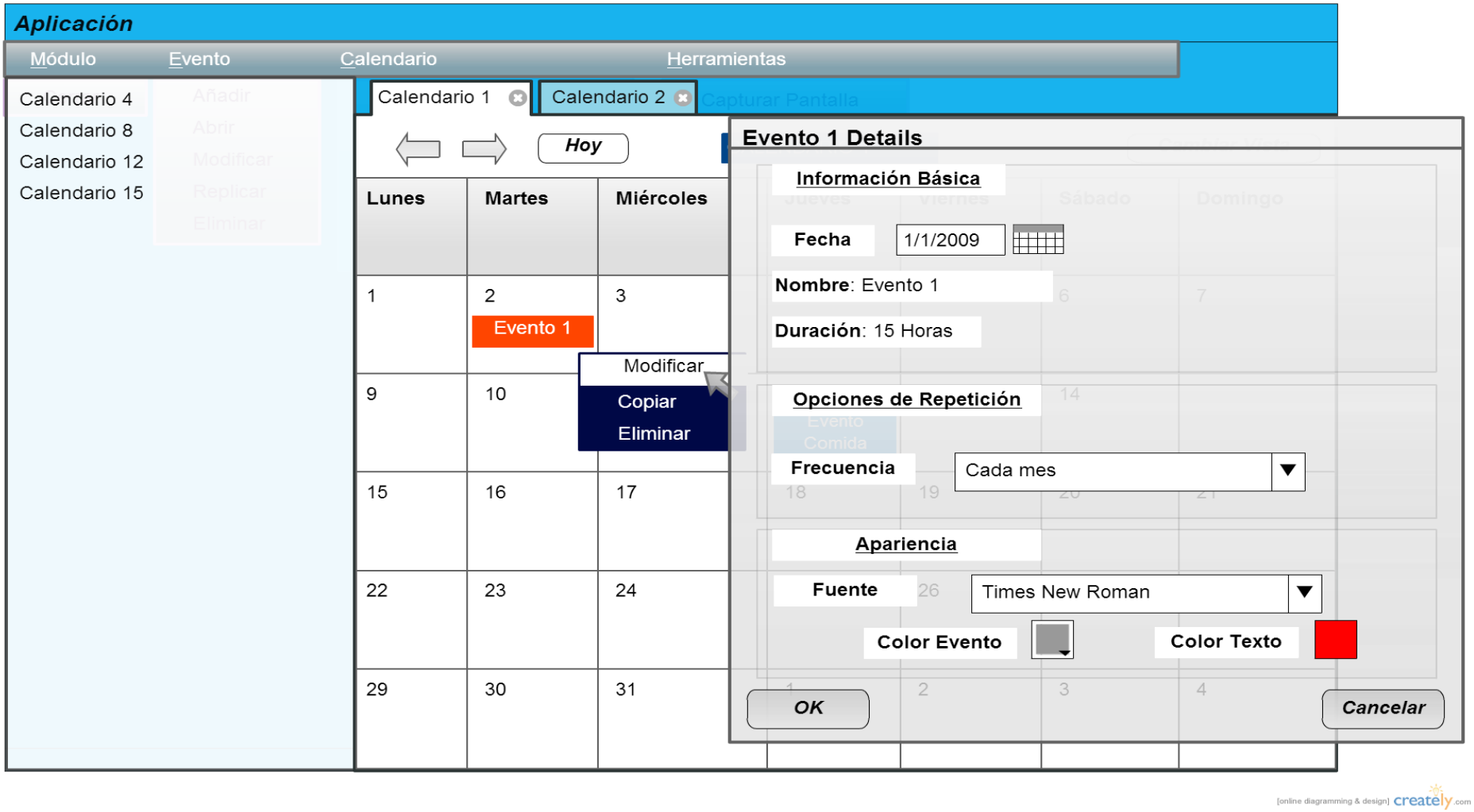


Ilustración 79: Diseño de la ventana principal del Módulo Calendario después de seleccionar la opción de *Ver información* de un evento

Volver a la Referencia

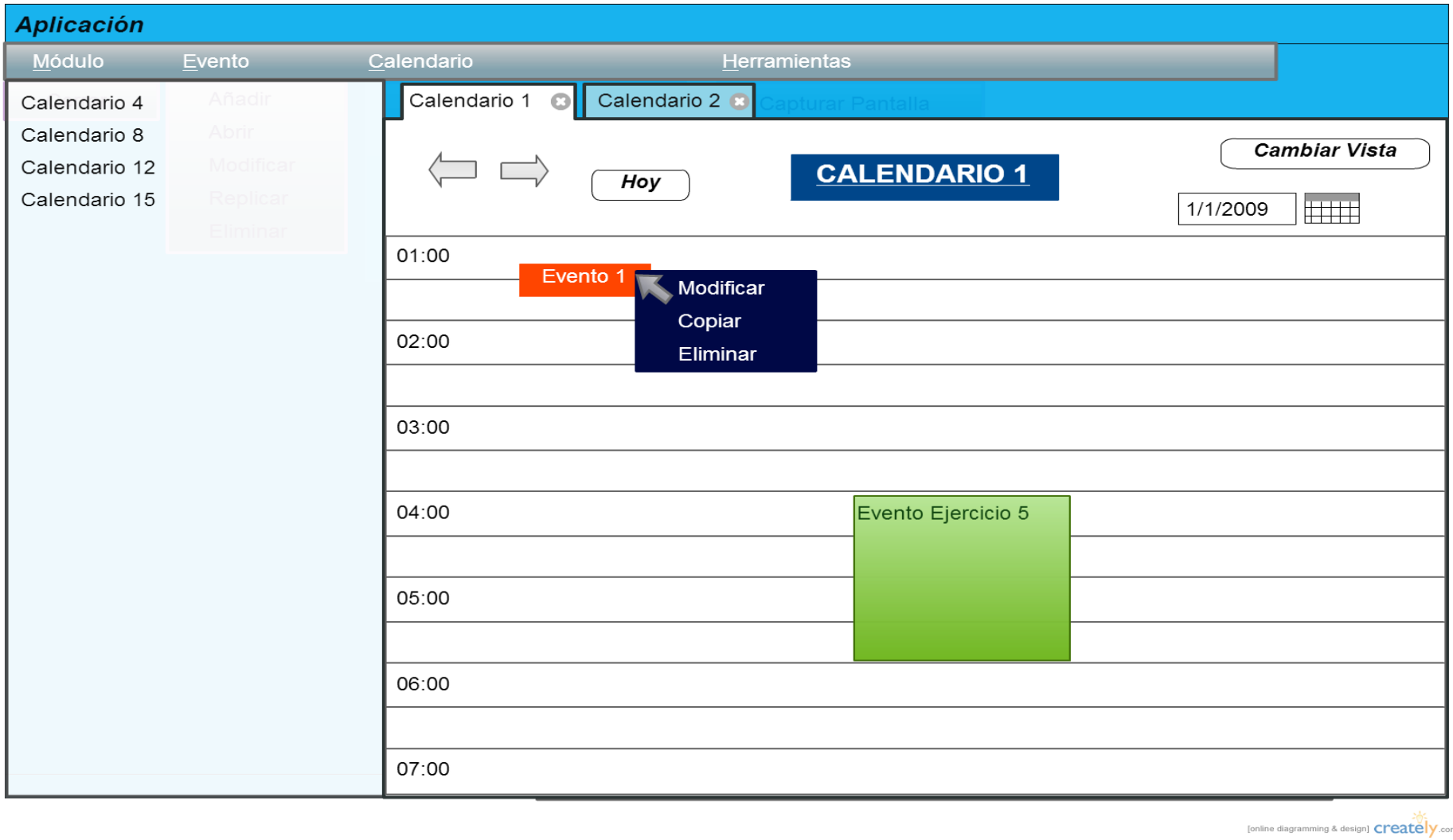


Ilustración 80: Diseño de la ventana principal del Módulo Calendario en modo *Vista Diaria*

Volver a la Referencia

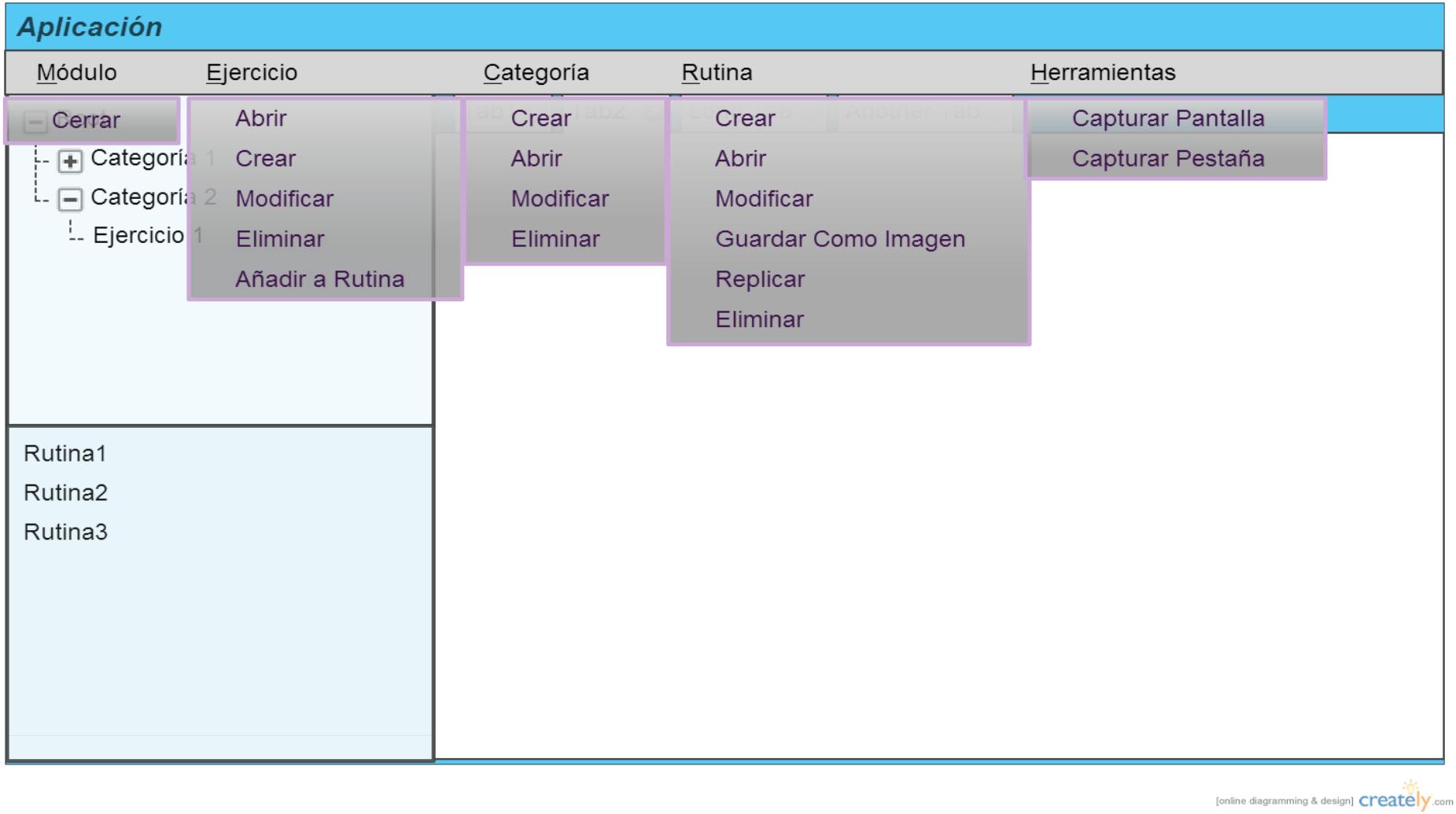


Ilustración 81: Diseño de la ventana principal del Módulo Ejercicios. A la izquierda se encuentran los árboles de Ejercicios y debajo la lista de Rutinas.

Volver a la Referencia

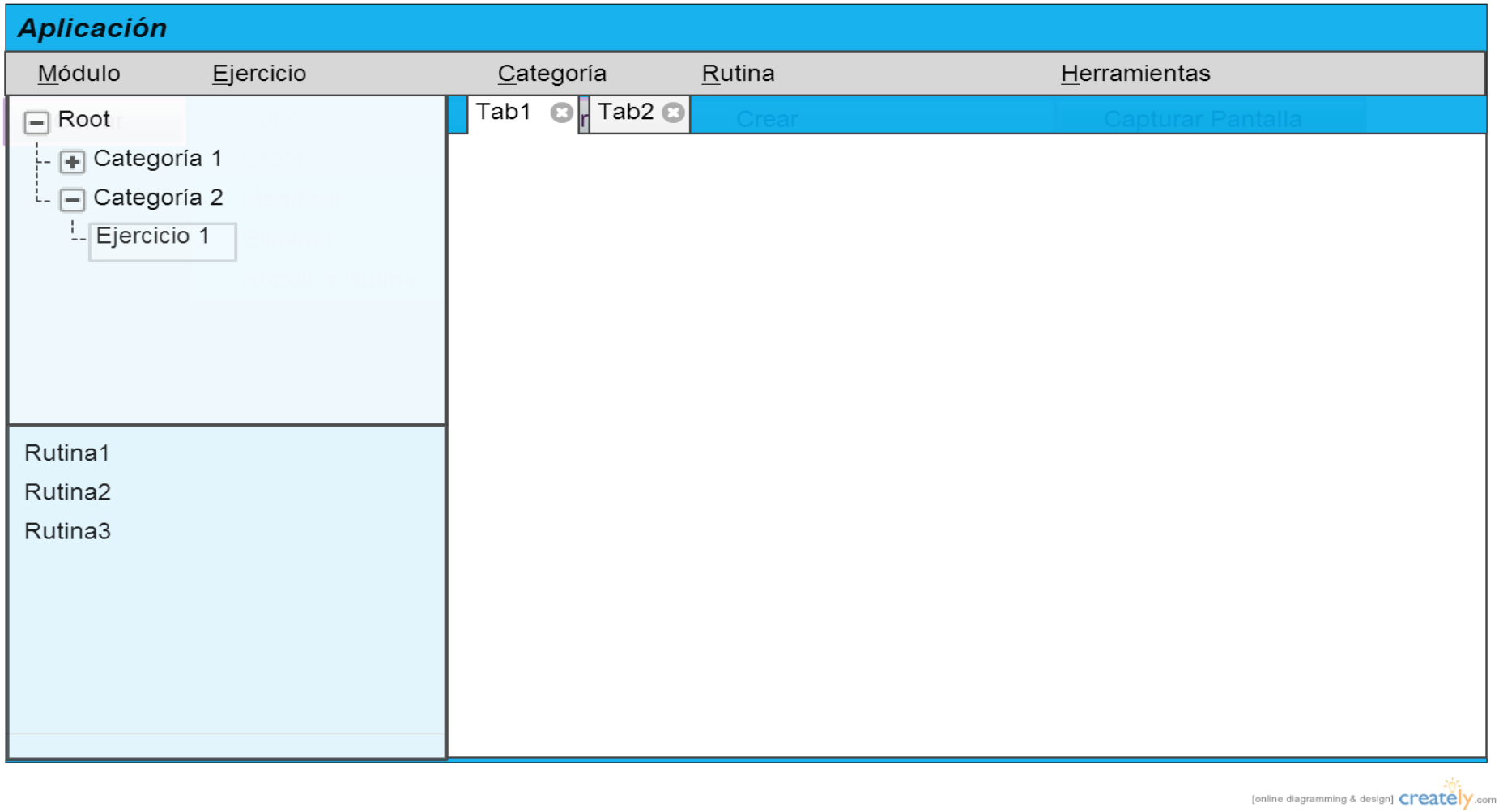


Ilustración 82: Diseño de la ventana principal del Módulo Ejercicios con la barra de pestañas.

Volver a la Referencia

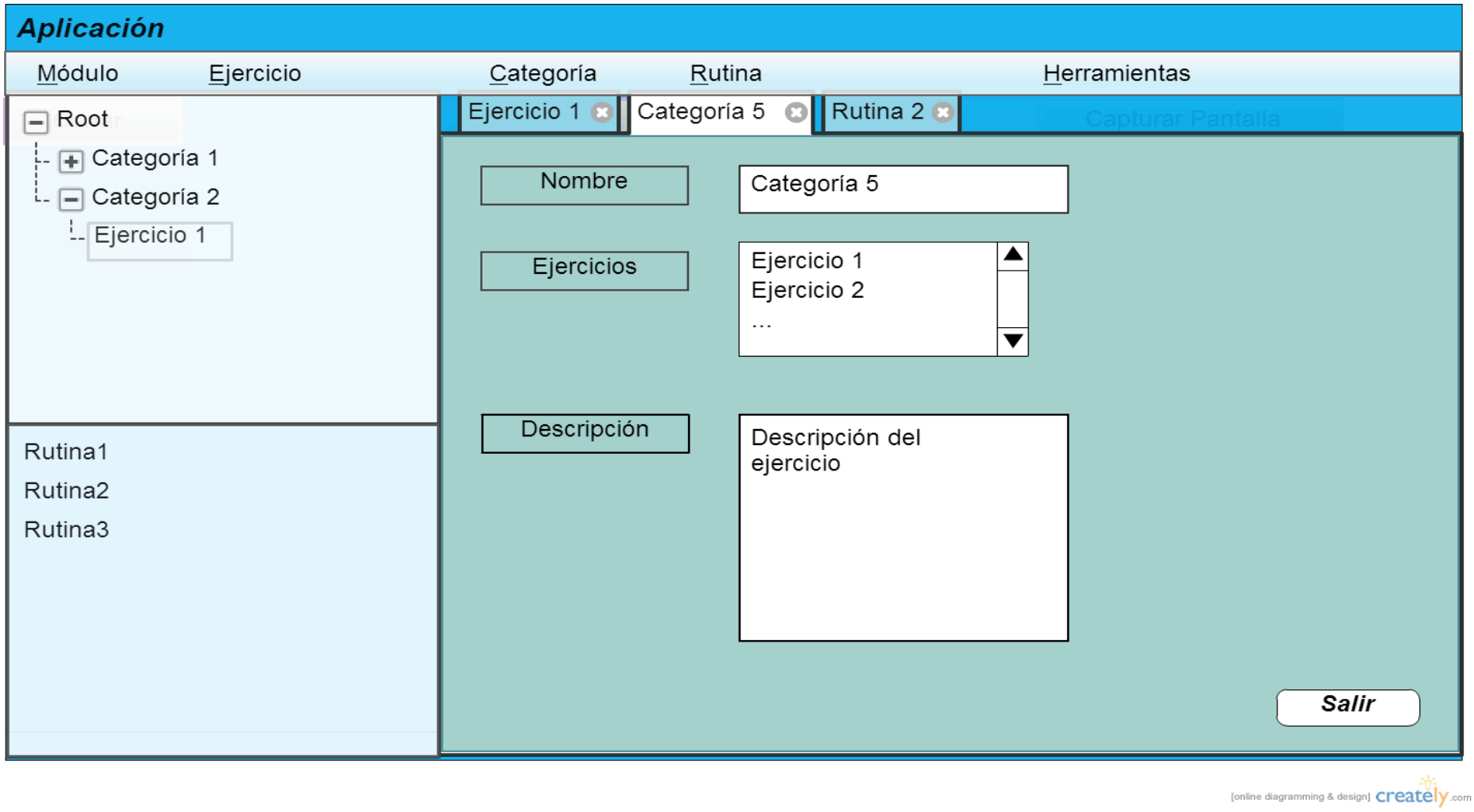


Ilustración 83: Diseño de la ventana principal del módulo *Ejercicios* con la pestaña de mostrar la información de *Categoría 6* seleccionada.

Volver a la Referencia

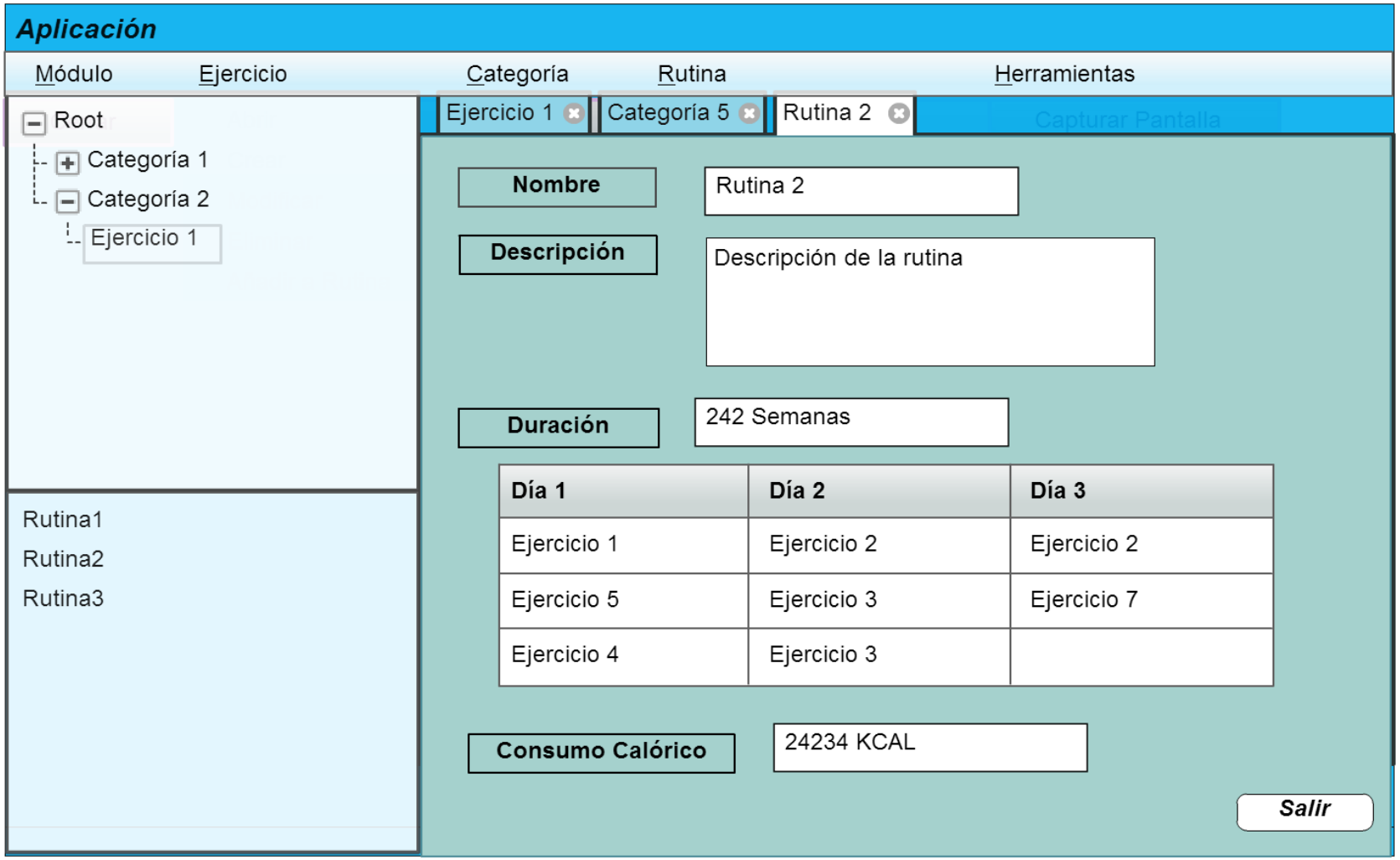


Ilustración 84: Diseño de la ventana principal del módulo Ejercicios con la pestaña de mostrar la información de *Rutina 1* seleccionada.

Volver a la Referencia

7.5 Incidencias durante la implementación

Módulo General

- Incidencia 1
  - **Formulario:** *ModuloGeneralVentanaPrincipal.cs*
  - **Descripción:** Path del archivo *usuarios.txt* de la base de datos incorrecto.

- Incidencia 2
  - **Formulario:** *ModuloGeneralMensajeError.cs*
  - **Descripción:** texto de la *textbox* incorrecto.

- Incidencia 3
  - **Formulario:** *ModuloGeneralVentanaNuevoUsuario.cs*
  - **Descripción:** Glich al clicar la pestaña *Información Biológica* tras clicar en botón *Siguiente* cuando se ha cometido un error al introducir el nombre o la contraseña.

- Incidencia 4
  - **Formulario:** *ModuloGeneralVentanaNuevoUsuario.cs*
  - **Descripción:** Revisar que al guardar un usuario en la base de datos, si no se añade el último campo (alergias), no se produzcan errores al leer dicho usuario de la base de datos.

- Incidencia 5
  - **Formulario:** *ModuloGeneralVentanaNuevoUsuario.cs*
  - **Descripción:** La codificación de los archivos ha de ser en UTF-8 para evitar problemas con caracteres especiales.

- Incidencia 6
  - **Formulario:** *ModuloGeneralVentanaNuevoUsuario.cs*
  - **Descripción:** Los usuarios se sobreescriben al agreagarlos a la base de datos.

- Incidencia 7
  - **Formulario:** *ModuloGeneralVentanaPrincipal.cs* y *ModuloGeneralVentanaPrincipalUsuarioLogueado.cs*
  - **Descripción:** No se puede reloguear con un usuario diferente (o el mismo) una vez logueado.

Módulo Ejercicios

- Incidencia 1
  - **Formulario:** *MóduloEjerciciosVentanaPrincipal.cs*
  - **Descripción:** Al hacer doble click en una categoría se abre como si fuera un ejercicio.

- Incidencia 2
  - **Formulario:** *MóduloEjerciciosVentanaPrincipal.cs*
  - **Descripción:** Si se abre un ejercicio y se intenta abrir de nuevo sin cambiar la selección, se produce un error.

- Incidencia 3
  - **Formulario:** *ModuloEjerciciosVentanaPrincipal.cs*
  - **Descripción:** A veces se produce un error al cerrar el módulo porque la aplicación intenta generar un árbol de ejercicios cuando ya no existen.

- Incidencia 4
  - **Formulario:** *ModuloEjerciciosVentanaPrincipal.cs*
  - **Descripción:** Al asignar el valor -1 a los campos que no aparecen en una rutina se producen micro errores en el cálculo de las calorías consumidas a nivel de rutina (ya que se asigna el valor -1 también al consumo calórico si no es introducido).

- Incidencia 5
  - **Formulario:** *ModuloEjerciciosPestañaEditarRutina.cs*
  - **Descripción:** Al borrar todos los ejercicios de una rutina el contador de calorías se queda a cero y no vuelve a subir al añadirse nuevos ejercicios.

- Incidencia 6
  - **Formulario:** *ModuloEjerciciosVentanaPrincipal.cs*
  - **Descripción:** Las capturas de pantalla no guardan el contenido de las *textbox* de tipo *RichTextBox*.

Módulo Dietista

No hay incidencias reportadas ya que el funcionamiento es prácticamente igual al del módulo *Ejercicios*.

Módulo Calendario

- Incidencia 1
  - **Formulario:** *ModuloCalendarioEventDetails.cs*
  - **Descripción:** Según el tipo de evento que se edite hay que cargar unos campos u otros en el formulario.

- Incidencia 2
  - **Formulario:** *ModuloCalendarioEventDetails.cs*
  - **Descripción:** Al cargar una dieta como evento hay que preguntar a qué hora se ingerirá cada comida —un evento por comida.
- Incidencia 3
  - **Formulario:** *Calendar.NET Calendar.cs*
  - **Descripción:** En la vista *Día a Día* no se cargan los eventos correctamente.

- Incidencia 4
  - **Formulario:** *Calendar.NET Calendar.cs*
  - **Descripción:** Hay que revisar la recurrencia de los eventos. El control deja de funcionar según el tipo de recurrencia elegido.

- Incidencia 5
  - **Formulario:** *Calendar.NET Calendar.cs*
  - **Descripción:** El control por defecto no permite elegir la hora a la que se realiza un evento, solo la fecha.

- Incidencia 6
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Descripción:** La relación entre la duración de un evento de tipo *EventoEjercicio* y la duración del ejercicio en sí ha de revisarse ya que si se establece una relación 1:1 —el evento dura lo que dure la ejecución del ejercicio— este no se ve reflejado en pantalla cuando la duración del ejercicio es muy corta.

- Incidencia 7
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Descripción:** Cuando dos eventos se superponen temporalmente hay que dividir el espacio de muestra para que ambos eventos sean visibles en la vista *Día a Día.*

- Incidencia 8
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Descripción:** Por qué algunos eventos se ven repetidos en la vista *Día a Día?*

- Incidencia 9
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Descripción:** Si un evento dura *hh:mm* —*h*, horas; *m*, minutos— si *m* es diferente de cero, este valor se ignora.

- Incidencia 10
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Decripción:** Revisar y rediseñar toda la función encargada de dibujar el calendario para solucionar los problemas con las funciones de recurrencia de los eventos.

- Incidencia 11
  - **Formulario:** *ModuloCalendarioVentanaPrincipal.cs*
  - **Descripción:** En el modo de vista *Mensual* del calendario, si un mismo día incluye más de cuatro eventos se dejan de visualizar el resto por falta de espacio.

Contraportada

Català

Aquesta memòria recull el procés seguit per al desenvolupament i implementació d'una aplicació Software. Aquest procés s'inicia amb la presentació del tema, estat de l'art i planificació temporal per continuar amb l'anàlisi de requeriments de l'aplicació —identificació dels interessats, entitats existents i les relacions d'aquestes...— el disseny de la mateixa —mètode de desenvolupament emprat, arquitectura...— la implementació de l'aplicació i les proves corresponents.

Finalment la memòria es completa amb les conclusions i vies de continuació així com la bibliografia i annexos complementaris.

Castellano

Esta memoria recoge el proceso seguido para el desarrollo e implementación de una aplicación Software. Este proceso se inicia con la presentación del tema, estado del arte y planificación temporal para continuar con el análisis de requerimientos de la apliación —identificación de los interesados, entidades existentes y sus relaciones…— el diseño de la misma —método de desarrollo, arquitectura…— la implementación de la aplicación y las pruebas correspondientes.

Finalmente la memoria se completa con las conclusiones y las vías de continuación así como la bibliografía y anexos complementarios.

English

This report joins the several stages that make up the process followed to develop and implement a Software application. This process starts with the presentation of the project's subject, the state of the art and its planning; to continue with the requirements analysis —stakeholder identification, existent entities and their relationships…—, the design of the application —development method used, architecture…—, the implementation and the application testing.

Finally the report is completed with the conclusions and future development paths, the bibliography and complementary annexes.