Proyecto Control De Inventarios

# Control de inventarios

Actualmente soy la encargada de llevar la gestión de inventario de componentes y libros en un laboratorio.

Utilizo un sistema online gratuito, debido a esto cuenta con muchos campos innecesarios y otros faltantes por lo que una parte la trabaja en Excel.

Por supuesto que hay sistemas de inventarios online a la medida y más robustos, pero tienen un costo mientras que algunos genéricos tienen liberado cierta parte de sus funciones completamente gratis.

El objetivo es realizar un sistema de inventario a la medida de lo que se necesite en el laboratorio, que sea fácil de usar, que facilite mi trabajo ahorrándome tiempo.

Después de revisar varios sistemas de inventario mi principal competidor es Quartzy ya que este sistema es el que se ha estado utilizando desde hace 3 años por tener las características más cercanas a lo que necesito.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Competitor | URL of Website or App Store Location | Purpose of Site | Year Founded | Funding Rounds | Social Networks | Content Types |
| Quartzy | [www.quartzy.com](http://www.quartzy.com/) | Proveedor de productos científicos de software de laboratorio | 2011 | 25 millones | Facebook, Twitter, LinkedIn, Instagram | Servicios de gestión de laboratorio Servicios de  adquisiciones |

2. Personas

3.

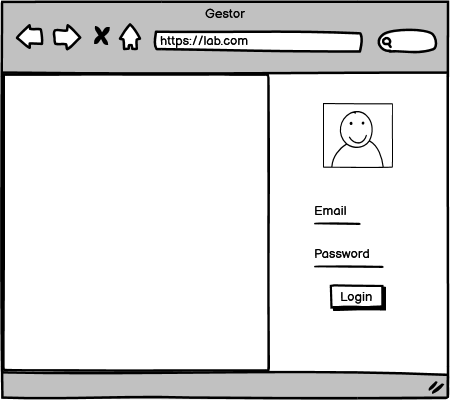


Ilustración 1 Login

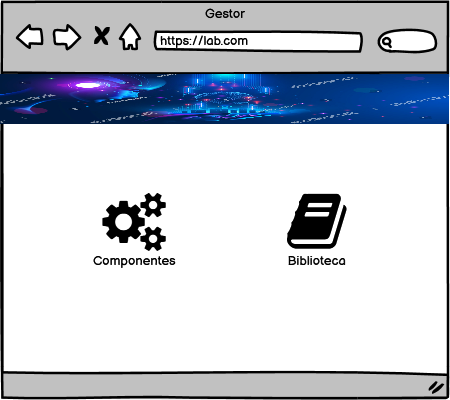


Ilustración 2 página principal.

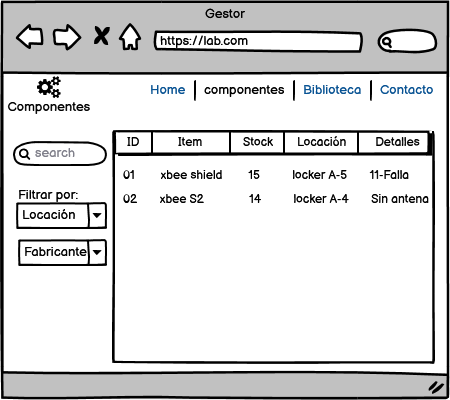


Ilustración 3 Componentes.

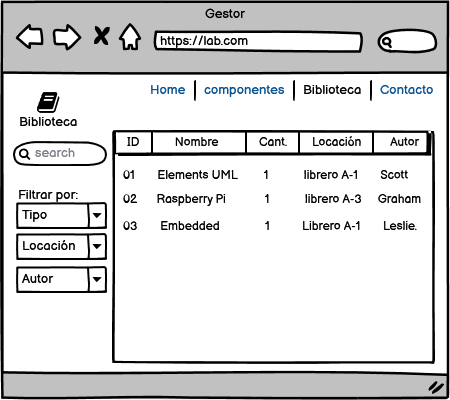


Ilustración 4 Biblioteca

4. Mejoras

El login es Bueno tanto el de mi competidor como el mío. Solo hace falta elegir colores e imágenes que combinen.

Página Principal

Tiene las opciones necesarias para funcionar, pero falta un poco más de diseño.

Componentes y biblioteca

Estas páginas tienen las opciones suficientes para ser entendido por el usuario.

En resumen, si realizare cambios y en especial en el diseño para que sea más amigable con el usuario y que sea muy fácil para el administrador de la página realizar los movimientos necesarios.

114.

Requerimientos Funcionales

1. El Sistema deberá autentificar con un usuario
2. El sistema deberá autentificar con una contraseña
3. El sistema deberá permitir loguearse.
4. El sistema deberá contar con una página principal
5. El usuario podrá observar sus opciones en la página principal
6. El sistema deberá de contar con una opción <Componentes>
7. El sistema deberá de filtrar por cualquiera de las categorías
8. El sistema podrá filtrar por categorías predeterminadas
9. El sistema podrá agregar un nuevo ítem
10. El sistema podrá modificar un ítem
11. El sistema podrá eliminar un ítem
12. El sistema deberá agregar cada ítem con un ID único.
13. El sistema deberá de tener una opción <Biblioteca>
14. El sistema deberá agregar cada libro con un ID único
15. El usuario podrá agregar un nuevo libro.
16. El usuario podrá modificar los datos de un libro
17. El usuario podrá eliminar un libro
18. El usuario podrá realizar búsquedas por cualquier categoría.
19. El usuario podrá filtrar libros por categorías ya preestablecidas
20. El usuario podrá visualizar los datos ingresados de los libros
21. El sistema deberá contar con la opción de regresar a la página principal
22. El sistema deberá tener la opción de salir de opción

Metodología Ágil

Scrum

|  |  |  |
| --- | --- | --- |
| Fase | Actividades | Tareas |
| Reunión de planificación de Sprint | Definir requerimientos  Definir funcionalidad | Diseño de login  Funcionalidad en login. |
| Scrum diario | Agregar product backlogs | Evaluar el progreso |
| Desarrollo durante el Sprint | Desarrollo del sprint | Observar que los cambios no afecten el sprint.  Aumentar o disminuir la complejidad del sprint para garantizar la adptacion de los avances. |
| Revisión | Revisión del trabajo realizado. | Definición de los próximos product backlog. |
| Retrospectiva | Inspeccion y creación de plan de mejoras. | Identificar lo que salio bien y potenciar las mejoras.  Crear plan para implementar las mejoras. |

Esquema Scrum

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteración | Fase | Actividades | Tareas | Tiempo |
| 1 | Planeación | Product backlog  Sprint planning | Crear interfaz  Crear funcionamiento login | 1 semana |
| Desarrollo | Reuniones para avances y problemas | Desarrollo del producto |
| Ejecución | Incremento del producto | Desarrollo más desarrollos generados en el sprint  Producto funcional |
| Revisión | Verificación del avance | Revisión de lo que falta.  Pasos siguientes |
| Retrospectiva | Revisión del desempeño | Verifica el proyecto y menciona mejoras. |
| 2 | Planeación | Product backlog  Sprint planning | Crear interfaz y funcionamiento modulo componentes | 1 semana |
| Desarrollo | Reuniones para avances y problemas | Desarrollo del producto |
| Ejecución | Incremento del producto | Desarrollo más desarrollos generados en el sprint  Producto funcional |
| Revisión | Verificación del avance | Revisión de lo que falta.  Pasos siguientes |
| Retrospectiva | Revisión del desempeño | Verifica el proyecto y menciona mejoras. |
| 3 | Planeación | Product backlog  Sprint planning | Trabajar en mejoras.  Crear interfaz modulo biblioteca. | 1 semana |
| Desarrollo | Reuniones para avances y problemas | Desarrollo del producto |
| Ejecución | Incremento del producto | Desarrollo más desarrollos generados en el sprint  Producto funcional |
| Revisión | Verificación del avance | Revisión de lo que falta.  Pasos siguientes |
| Retrospectiva | Revisión del desempeño | Verifica el proyecto y menciona mejoras. |
| 4 | Planeación | Product backlog  Sprint planning | Crear funcionamiento  modulo biblioteca | 1 semana |
| Desarrollo | Reuniones para avances y problemas | Desarrollo del producto |
| Ejecución | Incremento del producto | Desarrollo más desarrollos generados en el sprint  Producto funcional |
| Revisión | Verificación del avance | Revisión de lo que falta.  Pasos siguientes |
| Retrospectiva | Revisión del desempeño | Verifica el proyecto y menciona mejoras. |

# **Singleton**

El patrón Singleton tiene como objetivo asegurar que una clase sólo posee una instancia y proporcionar un método de clase único que devuelva esta instancia.

En ciertos casos es útil gestionar clases que posean una única instancia. En el marco de los patrones de construcción, podemos citar el caso de una fábrica de productos (patrón Abstract Factory) del que sólo es necesario crear una instancia.

## Estructura de Singleton

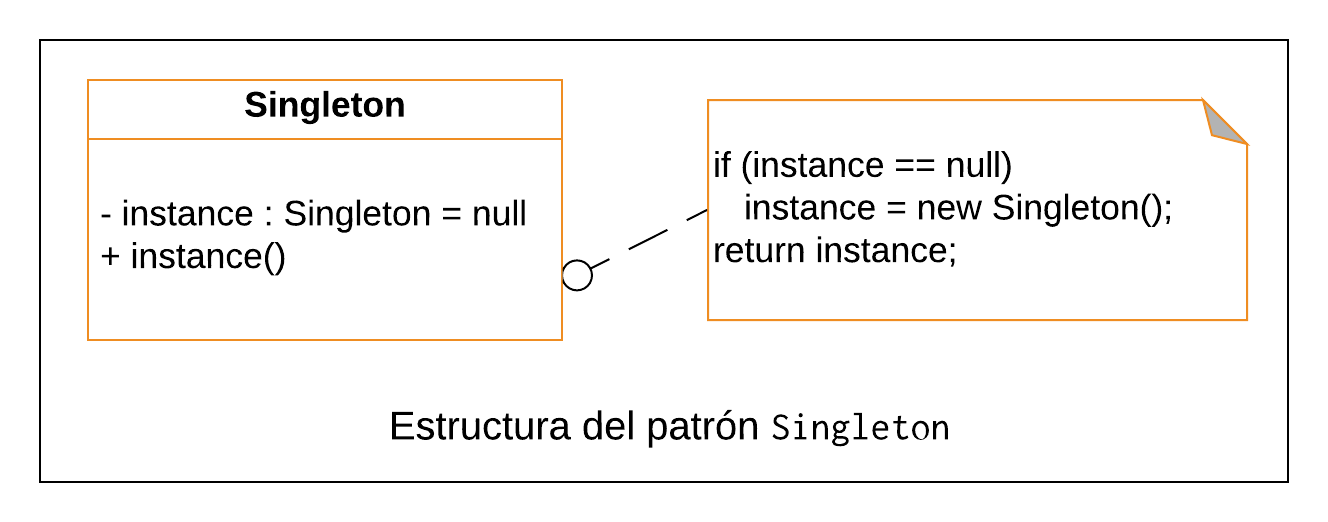


Figure 6. Estructura Patrón Singleton.

## Participante

El único participante es la clase Singleton que ofrece acceso a la instancia única mediante el método de clase *Instance*.

Por otro lado, la clase Singleton posee un mecanismo que asegura que sólo puede existir una única instancia. Este mecanismo bloquea la creación de otras instancias.

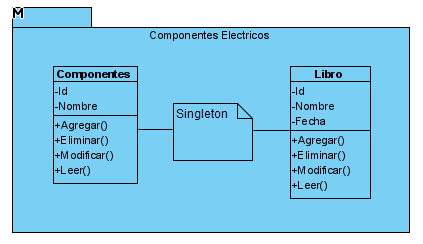
## Colaboración

Cada cliente de la clase Singleton accede a la instancia única mediante el método de clase Instance. No puede crear nuevas instancias utilizando el operador habitual de instanciación (operador *new*) que está bloqueado.

## Dominio de uso

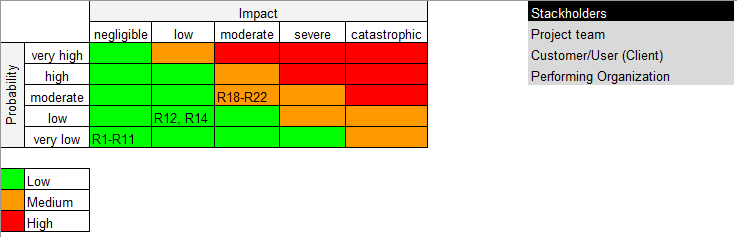
* Sólo debe existir una única instancia de una clase.
* Esta instancia sólo debe estar accesible mediante un método de clase.

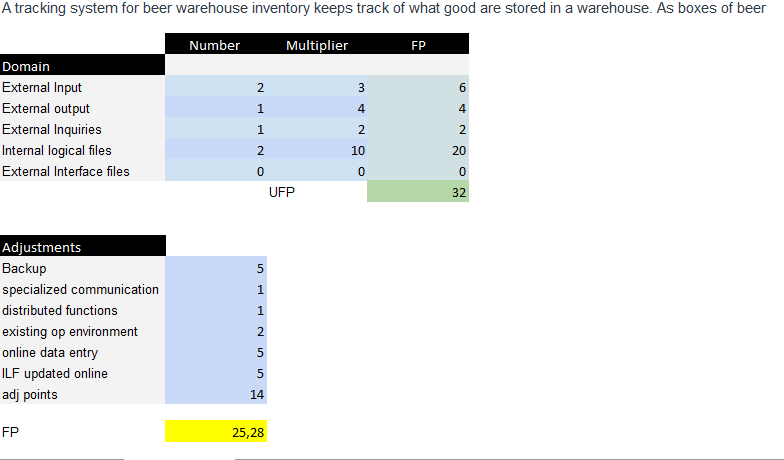
El uso del patrón Singleton ofrece a su vez la posibilidad de utilizar únicamente variables globales.

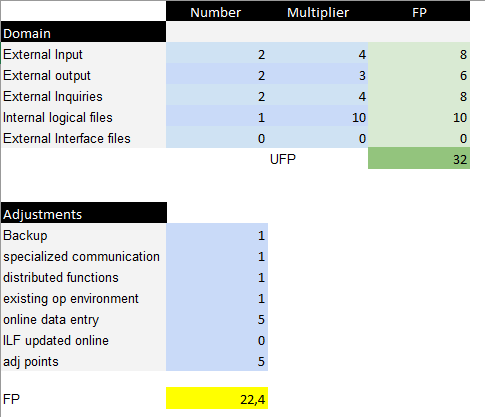


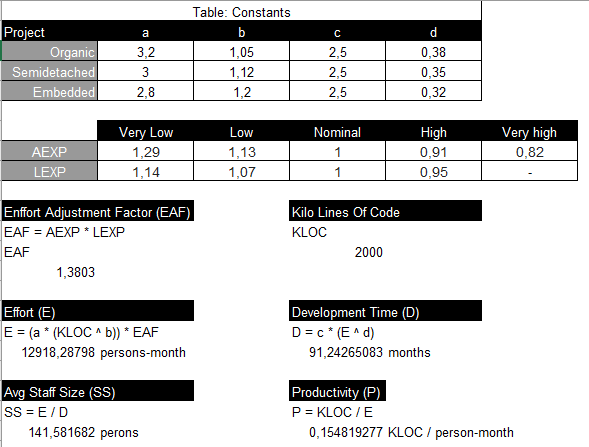
Identificación de riesgos durante los Sprint

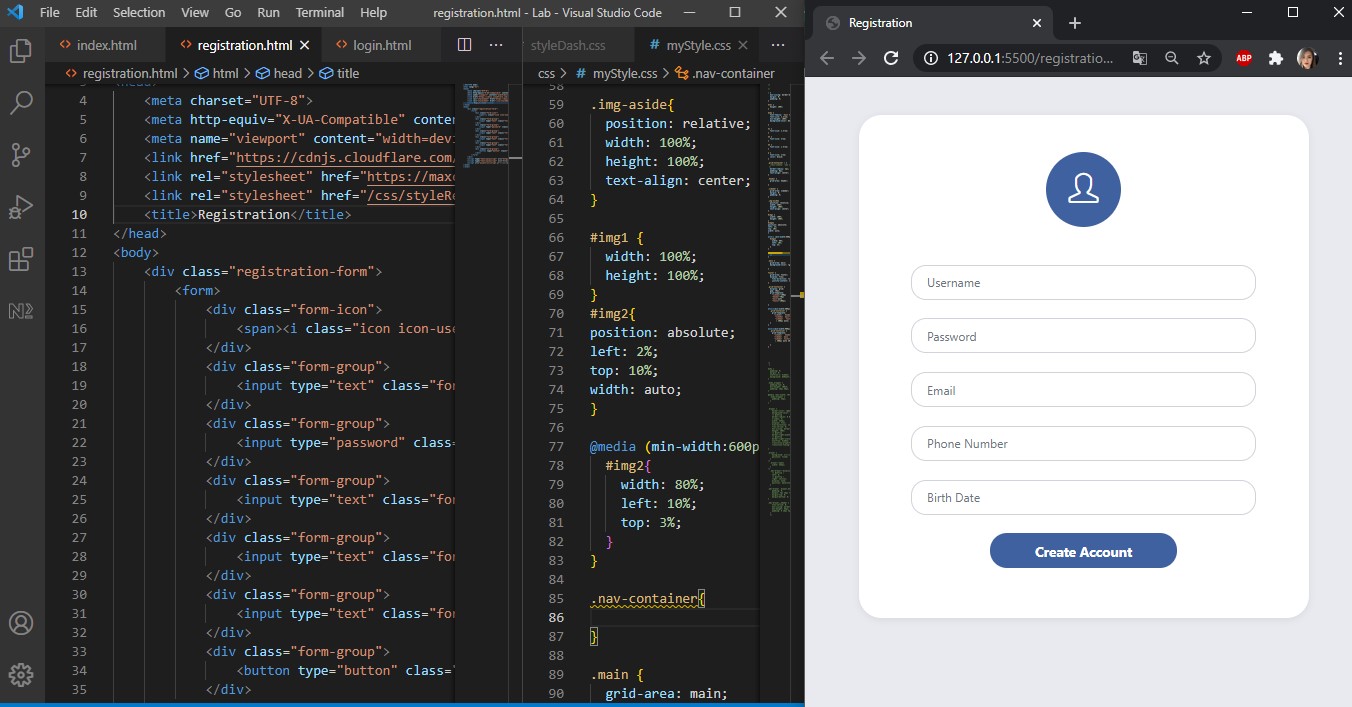
|  |  |
| --- | --- |
| Riesgo | Como mitigarlo |
| Sprint no completado. | Redistribución de tareas y modificación de tiempos. |
| Falta de detalles extras de diseño de interfaz. | Mejor descripción de las tareas. |
| Funcionalidad incompleta. | Redistribución de tiempo durante el sprint. |

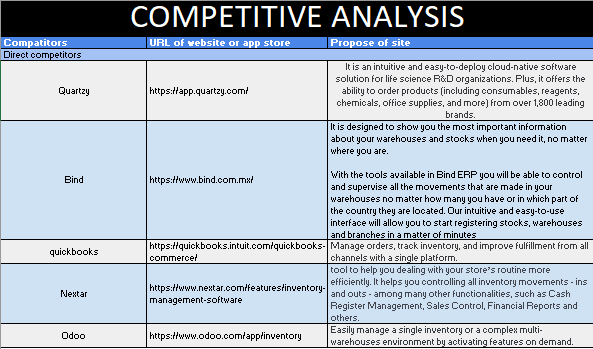


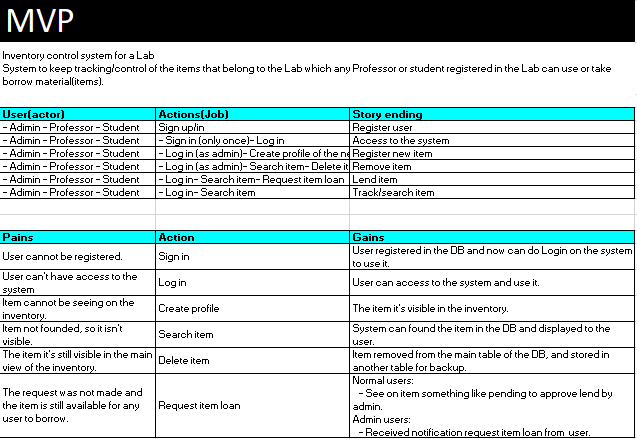












**Heuristics Evaluation of Control de Inventario.**

1. Visibility of system status

Users have visible action buttons and inventory and library data.

1. Match between the system and the real world

Users use appropriate language, information is placed in a logical order.

1. User control and freedom

The exit indicators are placed in a visible way so that there are no doubts on the part of the User.

1. Consistency and standards

The system design is simple and consistent for the user.

1. Error prevention

It has a simple design to follow so as not to make mistakes.

1. Recognition rather than recall

The system has CRUD options visible to the user.

1. Aesthetic and minimalist design

It has an aesthetic and minimalist design to facilitate its use.

1. Help users recognize, diagnose, and recover from errors.

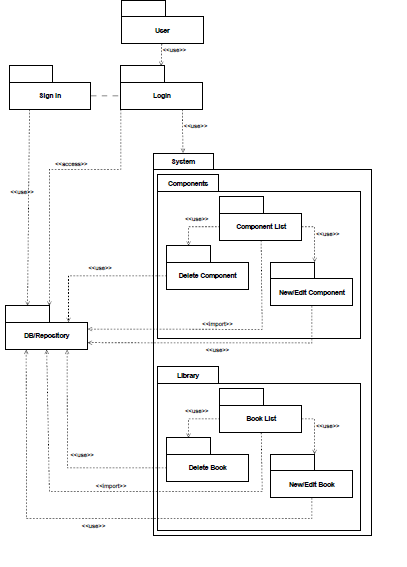
You will have messages suggesting the use of the resource.

1. Help and documentation

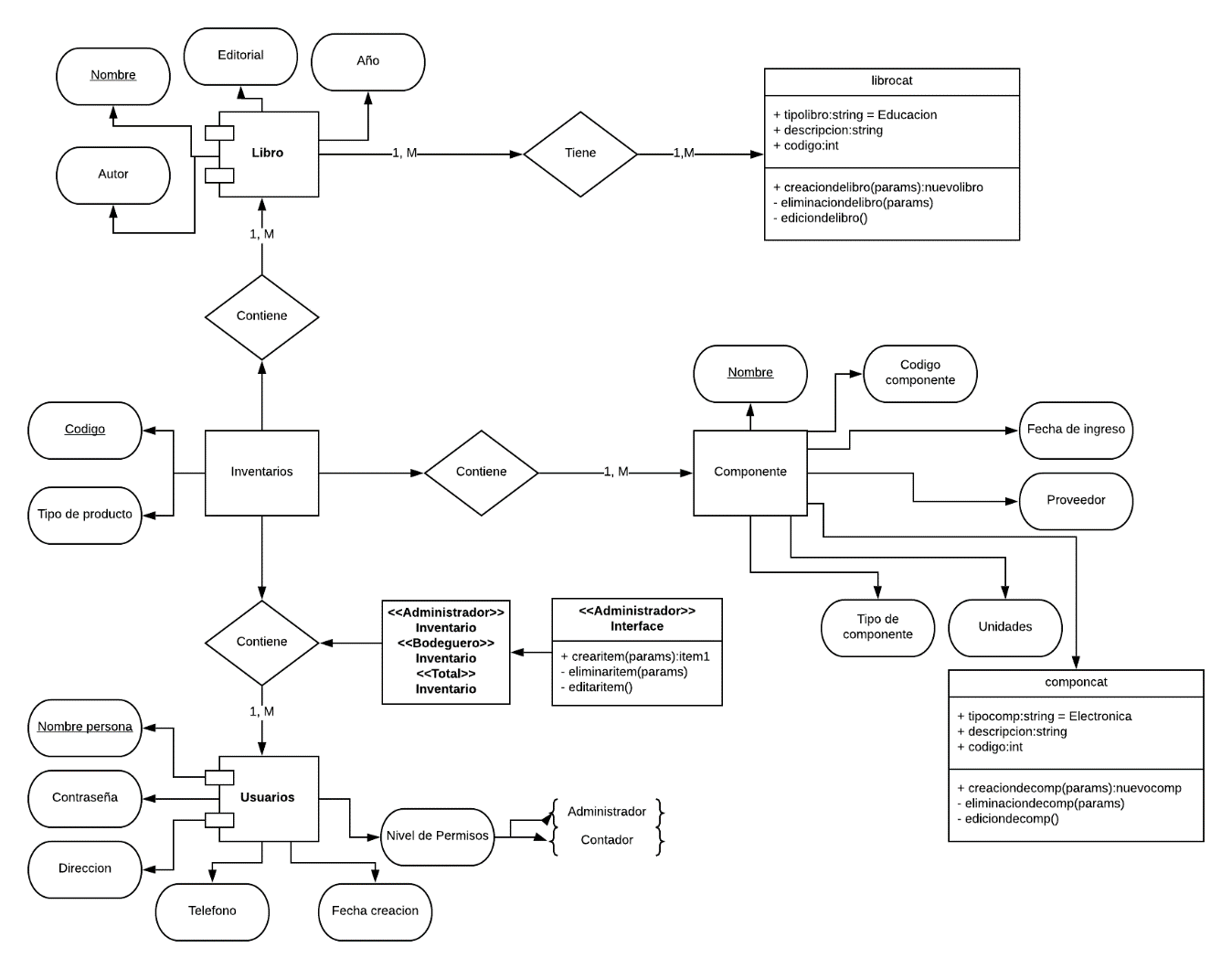
It will have an instructions section for its best use.

A layout was created showing the elements of the query, in this case the component button and the library button. A side menu that can be consulted by the user each time it is required and a header with the search element and exit of the session, a main one that contains the information of the components and books, as well as a footer where it is detailed the website.

**Inventory system package diagram**



|  |  |  |
| --- | --- | --- |
| **No.** | **Requerimiento** | **Módulo** |
| 1 | El Sistema deberá autentificar a un usuario | Login |
| 2 | El sistema deberá autentificar con una contraseña | Login |
| 3 | El sistema deberá permitir loguearse. | Login |
| 4 | El sistema deberá contar con una página principal | Main |
| 5 | El usuario podrá observar sus opciones en la página principal | Main |
| 6 | El sistema deberá de contar con una opción <Componentes> | Main |
| 7 | El sistema deberá de filtrar por cualquiera de las categorías | Componentes |
| 8 | El sistema podrá filtrar por categorías predeterminadas | Componentes |
| 9 | El sistema podrá agregar un nuevo ítem | Componentes->Agregar |
| 10 | El sistema podrá modificar un ítem | Componentes->Editar |
| 11 | El sistema podrá eliminar un ítem | Componentes->Eliminar |
| 12 | El sistema deberá agregar cada ítem con un ID único. | Componentes->Agregar |
| 13 | El sistema deberá de tener una opción <Biblioteca> | Main |
| 14 | El sistema deberá agregar cada libro con un ID único | Biblioteca->Agregar |
| 15 | El usuario podrá agregar un nuevo libro. | Biblioteca->Agregar |
| 16 | El usuario podrá modificar los datos de un libro | Biblioteca->Editar |
| 17 | El usuario podrá eliminar un libro | Biblioteca->Eliminar |
| 18 | El usuario podrá realizar búsquedas por cualquier categoría. | Biblioteca |
| 19 | El usuario podrá filtrar libros por categorías ya preestablecidas | Biblioteca |
| 20 | El usuario podrá visualizar los datos ingresados de los libro | Biblioteca->Detalles |
| 21 | El sistema deberá contar con la opción de regresar a la página principal | Main |
| 22 | El sistema deberá tener la opción de salir de opción | Main |



# **Design patterns**

# **Factory Method**

El objetivo del patrón Factory Method es proveer un método abstracto de creación de un objeto delegando en las subclases concretas su creación efectiva.

## Estructura Factory Method

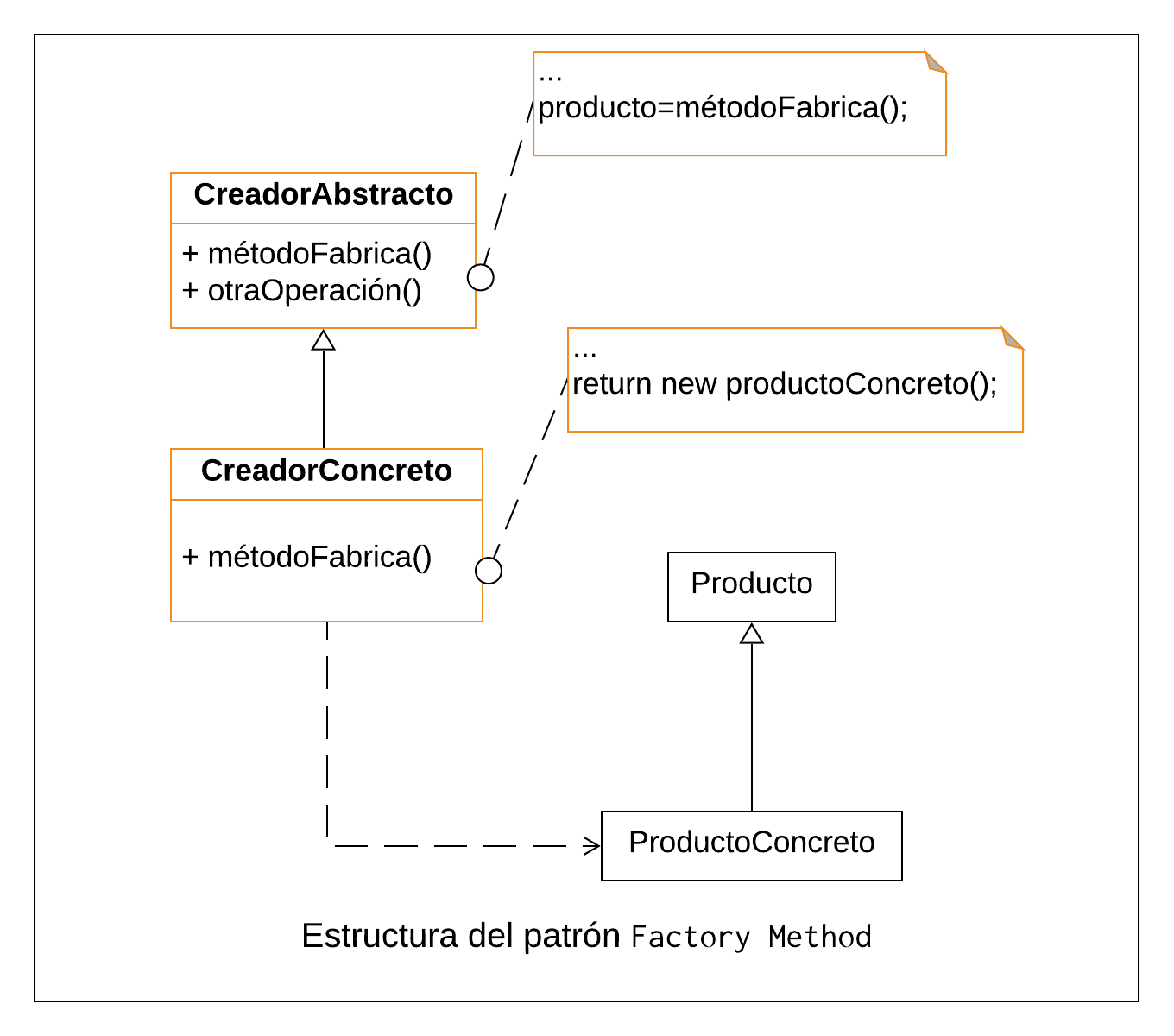


Figure 1. Estructura del Patrón Factory Method.

## Participantes

Los participantes del patrón son los siguientes:

* *CreadorAbstracto* (Cliente) es una clase abstracta que implementa la firma del método de fabricación y los métodos que invocan al método de fabricación.
* *CreadorConcreto* (ClienteContado, ClienteCrédito) es una clase concreta que implementa el método de fabricación. Pueden existir varios creadores concretos.
* *Producto* (Pedido) es una clase abstracta que describe las propiedades comunes de los productos.
* *ProductoConcreto* (PedidoContado, PedidoCrédito) es una clase concreta que describe completamente un producto.

## Colaboraciones

Los métodos concretos de la clase *CreadorAbstracto* se basan en la implementación del método de fabricación en las subclases. Esta implementación crea una instancia de la subclase adecuada de Producto.

## Dominios de uso

El patrón se utiliza en los casos siguientes:

* Una clase que sólo conoce los objetos con los que tiene relaciones
* Una clase quiere transmitir a sus subclases las elecciones de instanciación aprovechando un mecanismo de polimorfismo.

# **Singleton**

El patrón Singleton tiene como objetivo asegurar que una clase sólo posee una instancia y proporcionar un método de clase único que devuelva esta instancia.

En ciertos casos es útil gestionar clases que posean una única instancia. En el marco de los patrones de construcción, podemos citar el caso de una fábrica de productos (patrón Abstract Factory) del que sólo es necesario crear una instancia.

## Estructura de Singleton

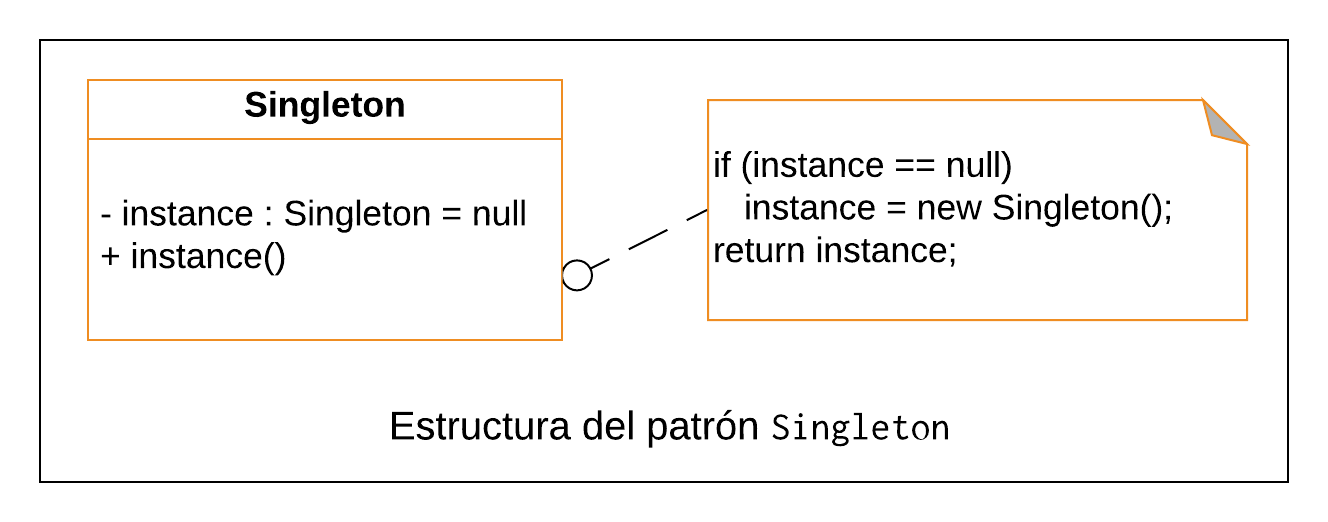


Figure 6. Estructura Patrón Singleton.

## Participante

El único participante es la clase Singleton que ofrece acceso a la instancia única mediante el método de clase *Instance*.

Por otro lado, la clase Singleton posee un mecanismo que asegura que sólo puede existir una única instancia. Este mecanismo bloquea la creación de otras instancias.

## Colaboración

Cada cliente de la clase Singleton accede a la instancia única mediante el método de clase Instance. No puede crear nuevas instancias utilizando el operador habitual de instanciación (operador *new*) que está bloqueado.

## Dominio de uso

* Sólo debe existir una única instancia de una clase.
* Esta instancia sólo debe estar accesible mediante un método de clase.

El uso del patrón Singleton ofrece a su vez la posibilidad de utilizar únicamente variables globales.

# **Observer**

El patrón Observer tiene como objetivo construir una dependencia entre un sujeto y los observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado.

## Diagrama de Patrón Observer

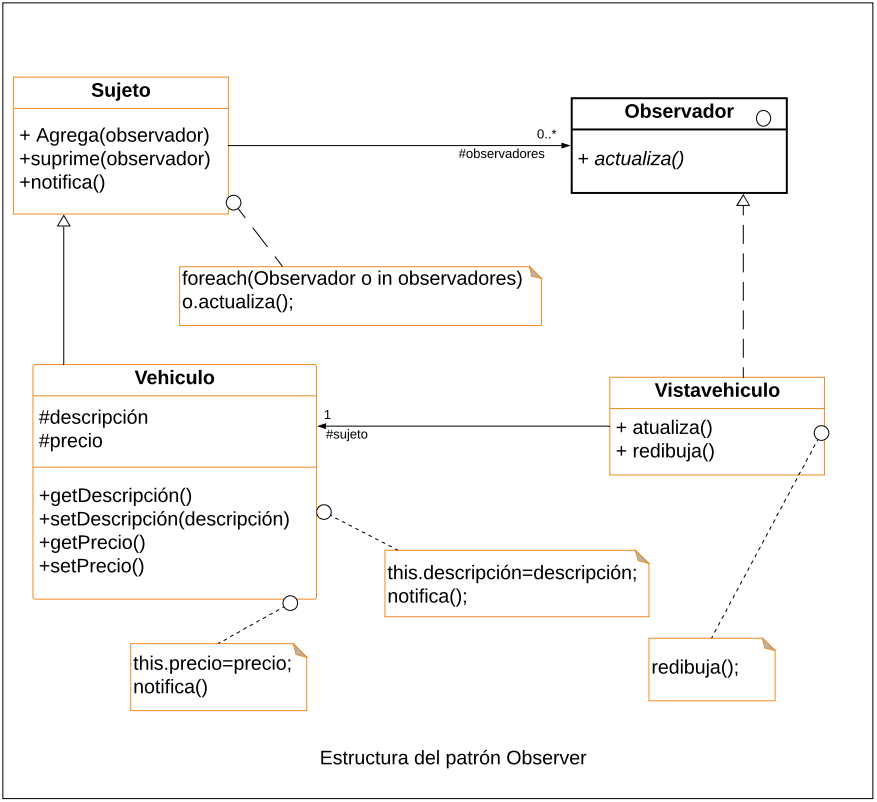


Figure 1. Patrón Observer aplicado a la visualización de vehículos.

## Clases del diagrama

El diagrama contiene las cuatro clases siguientes:

* Sujeto es la clase abstracta que incluye todo objeto que notifica a los demás objetos de las modificaciones en su estado interno.
* Vehículo es la subclase concreta de Sujeto que describe a los vehículos. Gestiona dos atributos: descripción y precio.
* Observador es la interfaz de todo objeto que necesite recibir las notificaciones de cambio de estado provenientes de los objetos a los que se ha inscrito previamente.
* VistaVehículo es la subclase concreta correspondiente a la implementación de Observador cuyas instancias muestran la información de un vehículo.

## Dominios de aplicación

* Una modificación en el estado de un objeto genera modificaciones en otros objetos que se determinan dinámicamente.
* Un objeto quiere avisar a otros objetos sin tener que conocer su tipo, es decir sin estar fuertemente acoplado a ellos
* No se desea fusionar dos objetos en uno solo.

# Referencias

Debrauwer, L. (2012). *Patrones de diseño para c#: los 23 modelos de diseño.descripcion y soluciones ilustradas en uml 2 y c#.* ENI.