

Gestión de un banco de sangre utilizando bases de datos y glassfish/payara

Miembros:

Enrique de la Moral Falagán

DIEGO VÁZQUEZ GONZÁLEZ

Grado en ingeniería informática

Ingeniería de Software II

Resumen breve del informe

Se desarrollará de forma breve la funcionalidad a realizar para la gestión de un banco de sangre.

Índice

1.Funcionalidad

2.Diseño de las tablas de la base de datos

3.Tecnologías a usar

4.Ramas git

5.Pruebas caja blanca-caja negra

6.Sprints

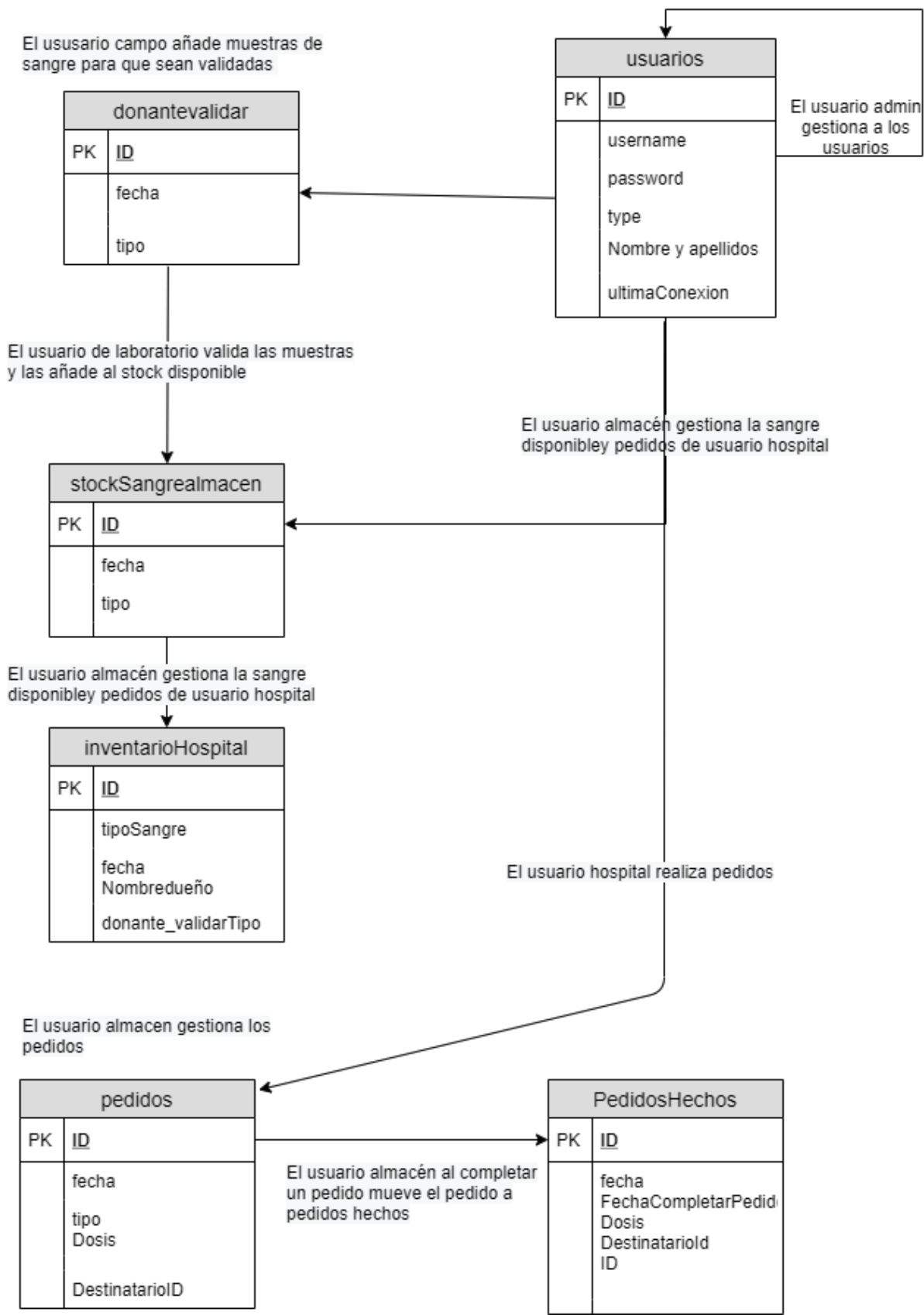
1.Funcionalidad

En rojo significa no implementado.

Se quiere gestionar el inventario / recolección y envíos de donaciones de sangre para una entidad , para ello se implementarán las siguientes funcionalidades:

- **El programa tendrá una función de ayuda al usuario**
- El programa tendrá una base de datos que contendrá la información que se gestionará
- El programa podrá hacer copias de seguridad
- El programa implementará requisitos que se comprobarán de manera automática
- El programa implementa filtros para su funcionamiento
- Habrá diferentes usuarios cada uno con restricciones de acceso a la funcionalidad(google:login java oracle sql)
- Opcionalmente se podrán obtener estadísticas del inventario y demanda(contar elementos en una tabla sql)
- **Opcionalmente se podrán generar alertas basadas en las estadísticas anteriores, por ejemplo , si el stock de un tipo de sangre baja mucho se manda una alerta por email.**
- El programa funcionará de manera remota en vez de manera local (usando glassfish/payara)
- **Se harán logs que reflejen la actividad de cada usuario.**
- 5 tipos de usuarios, que según el mismo se podrán realizar determinadas acciones
 - Usuario administrador que se encarga de hacer la backup de la base de datos , registrar o borrar usuarios de distinto tipo y ver la ultima vez que se conectaron.
 - Usuario campo que se encarga de recoger muestras de sangre una vez rellenado un cuestionario
 - Usuario laboratorio que valida o no las muestras de sangre del usuario campo
 - Usuario almacén que se encarga de ver el inventario total de sangre dependiendo de sus tipos y que gestiona el envío a hospitales
 - Usuario hospital que solicita pedidos sí sus reservas bajan mucho

2.Diseño de las tablas de la base de datos



3. Tecnologías a usar

En principio se usaría como IDE Netbeans , el lenguaje de programación es java , la aplicación que almacene / gestione la base de datos será mySql y se usará el servidor de aplicaciones glassfish o payara junto al driver jdbc que conectará java con la base de datos. También se mapeara la base de datos para usar hibernate para hacer las querys de inserción y de consulta.



















4. Ramas git

Rama master, pruebas caja blanca y otras mejoras es un merge de esta rama:

Pruebas caja blanca y otras mejoras edelafo1 • 44m
Implementado backup base de datos para admin edelafo1 • 18h
Practica acabada solo quedaria pulir edelafo1 • 18h
Ahora el sql tiene dueño para la sangre de cada hospital edelafo1 • 19h
Usuario almacen acabado faltan retoques edelafo1 • 21h
Mapeadas las clases de alguna manera edelafo1 • 1d
Usuario laboratorio acabado edelafo1 • 1d
Borrar sangre laboratorio edelafo1 • 1d
funcionalidad campo hecha edelafo1 • 1d
Ya esta la ultima fecha de login edelafo1 • 1d
Hecha la tabla de usuarios falta arreglar mensajes edelafo1 • 2d
Borrar hecho edelafo1 • 2d
Formulario registro acabado edelafo1 • 2d
no se como pero ya esta el hibernate edelafo1 • 3d
Cambios2 edelafo1 • 3d
Revert "Cmbios en general de todo el proyecto" edelafo1 • 3d
Borrar lo anterior edelafo1 • 3d
Cmbios en general de todo el proyecto edelafo1 • 4d

Rama-cambios:

La rama se dividió en Implementado backup base de datos para admin.

Prueba  edelaf01 • 2m
Pruebas caja blanca y otras mejoras  edelaf01 • 46m
Implementado backup base de datos para admin  edelaf01 • 18h
Practica acabada solo quedaria pulir  edelaf01 • 18h
Ahora el sql tiene dueño para la sangre de cada hospital  edelaf01 • 19h
Usuario almacen acabado faltan retoques  edelaf01 • 21h
Mapeadas las clases de alguna manera  edelaf01 • 1d
Usuario laboratorio acabado  edelaf01 • 1d
Borrar sangre laboratorio  edelaf01 • 1d
funcionalidad campo hecha  edelaf01 • 1d
Ya esta la ultima fecha de login  edelaf01 • 1d
Hecha la tabla de usuarios falta arreglar mensajes  edelaf01 • 2d
Borrar hecho  edelaf01 • 2d
Formulario registro acabado  edelaf01 • 2d
no se como pero ya esta el hibernate  edelaf01 • 3d
Cambios2  edelaf01 • 3d
Revert "Cmabios en general de todo el proyecto"  edelaf01 • 3d
Borrar lo anterior  edelaf01 • 3d

5.Pruebas caja blanca/ caja negra.

Caja blanca: Beans.Pedidos.java

```
Start Page X Almacen.xhtml X admin.xhtml X Login.java X AlmacenImpl.java X Pedidos.java X
Source History

66
67 //Para enviar necesitamos recibir un id
68 public void enviarPedido() {
69     ule.edi.model.Pedidos p = new ule.edi.model.Pedidos();
70     p.setId(id);
71     //Con esto tenemos el id del pedido que queremos hacer
72     AlmacenImpl adao = new AlmacenImpl();
73     System.out.println("Prueba caja blanca Pedidos:\nVoy a buscar el pedido con el id: " + id);
74     List<ule.edi.model.Pedidos> lista = adao.getPedido(p);
75
76     if (lista.isEmpty()) {
77         System.out.println("Prueba caja blanca Pedidos resultado: Lista esta vacia , no se ha encontrado el pedido con el id: " + id);
78         FacesContext.getCurrentInstance().addMessage(
79             null,
80             new FacesMessage(FacesMessage.SEVERITY_WARN,
81                 "Pedido con el id dado no se ha encontrado , error", ""
82             ));
83     } else {
84         System.out.println("Prueba caja blanca Pedidos resultado: Se ha encontrado el pedido con el id: " + id);
85         FacesContext.getCurrentInstance().addMessage(
86             null,
87             new FacesMessage(FacesMessage.SEVERITY_WARN,
88                 "Pedido encontrado", ""
89             ));
90
91         String enviarSangreTipo = lista.get(0).getTipo();
92
93         int cantidadAenviar = lista.get(0).getDosis();
94         System.out.println("Voy a enviar sangre de tipo " + enviarSangreTipo
95             + " una cantidad total de " + cantidadAenviar + " dosis por completar el pedido");
96         List<Stocksangrealmacen> listaStockActual = adao.generarTablaAlmacen();
97         //tengo todo el stock disponible en listastockactual
98         if (listaStockActual.isEmpty()) {
99
100             if (listaStockActual.isEmpty()) {
101                 System.out.println("Prueba caja blanca Pedidos resultado: El stock actual del almacen no dispone del tipo o la cantidad de sangre necesarias , StockActual para el tipo");
102                 FacesContext.getCurrentInstance().addMessage(
103                     null,
104                     new FacesMessage(FacesMessage.SEVERITY_WARN,
105                         "Error no hay stock", ""
106                     ));
107             } else {
108                 System.out.println("Hay stock disponible para el tipo de sangre: " + enviarSangreTipo + " voy a intentar enviar los maximos posibles de dosis");
109                 int tmp = cantidadAenviar, id2 = 0;
110                 String nom = "";
111                 for (int i = 0; i < listaStockActual.size(); i++) {
112                     if ((listaStockActual.get(i).getTipo().equals(enviarSangreTipo) && cantidadAenviar > 0) {
113
114                         cantidadAenviar--;
115                         System.out.println("He enviado una dosis , la cantidad ha enviar actual es de : " + cantidadAenviar);
116                         //enviar esta sangre a stockhospital
117                         Stocksangrealmacen enviaryborrar = listaStockActual.get(i);
118                         Inventariohospital recibir = new Inventariohospital();
119                         recibir.setTipoSangre(enviaryborrar.getTipo());
120                         recibir.setFecha(enviaryborrar.getFecha());
121                         if (i == 0) {
122
123                             id2 = lista.get(0).getDestinatarioid();
124                             System.out.println("Encuento el id al que le voy a enviar el pedido que es id: " + id2 + " . Ahora busco el nombre para ese id de usuario destinatario");
125                             nom = adao.getNombreDestinatario(id2);
126                             System.out.println("El nombre del usuario que va a recibir es: " + nom);
127                         }
128                         //OBTENGO EL USERNAME PARA EL ID
129
130                         recibir.setNombreDuenyo(nom);
131                         //recibir.setNombreDuenyo();
132                         System.out.println("Envio sangre al usuario " + nom);
133                         adao.enviarSangre(recibir);
134
135                         adao.borrarSangre(enviaryborrar);
136                         System.out.println("Borro una dosis del stock almacen ");
137                         //uno menos en el pedido
138                         adao.actualizarPedido(lista.get(0));
139                         System.out.println("Actualizo el pedido, ahora quedan " + lista.get(0).getDosis());
140                     }
141                 }
142             }
143         }
144         if (cantidadAenviar == 0) {

```

```

if (cantidadAenviar == 0) {
    System.out.println("Pedido con el id " + id2 + " completado, procedo a borrar el pedido y moverlo a pedidos completados");
    ule.edi.model.Pedidos p2 = new ule.edi.model.Pedidos();
    p2 = lista.get(0);
    Pedidoshechos p3 = new Pedidoshechos();
    p3.setDestinatarioid(p2.getDestinatarioid());
    p3.setDosis(p2.getDosis());
    p3.setFecha(p2.getFecha());
    p3.setTipo(p2.getTipo());
    Date date = new Date(System.currentTimeMillis());
    p3.setFechaCompletarpedido(date);
    //lo envio
    adao.enviarPedidoCompletado(p3);
    //lo borro
    adao.borrarPedidoCompletado(p2);
    FacesContext.getCurrentInstance().addMessage(
        null,
        new FacesMessage(FacesMessage.SEVERITY_WARN,
            "Se ha completado un pedido,basado", ""
        ));
}
if (tmp == cantidadAenviar) {
    System.out.println("Pedido con el id " + id2 + " no puede ser completado por falta de stock error");
    FacesContext.getCurrentInstance().addMessage(
        null,
        new FacesMessage(FacesMessage.SEVERITY_WARN,
            "Error no hay stock para completar el pedido", ""
        ));
}
}

```

Caja negra:

Beans Login.java

```

public String validateUsernamePassword() {
    String metodo = "login";
    boolean valid = ldao.validate(user, pwd, type, metodo);
    if (valid) {
        //ultimo login

        HttpSession session = SessionUtils.getSession();
        session.setAttribute("username", user);
        type2 = type;
        user2 = user;
        return type;

    } else {
        FacesContext.getCurrentInstance().addMessage(
            null,
            new FacesMessage(FacesMessage.SEVERITY_WARN,
                "Incorrect Username and Password",
                "Please enter correct username and Password"));
        return "login";
    }
}
}

```

6.Sprint Scrum

-1er Sprint:1 mayo

Objetivo: Implementar la conexión a la base de datos usando hibernate , jsf y Glassfish.

-División del trabajo:

Implementar java con hibernate para conectarse a mysql: Enrique (Finalizado a las dos semanas de comenzar el sprint)

Implementar el servidor glassfish para que tenga conexión a la base de datos: Diego (Finalizado el tercer día del sprint)

Crear base de datos: Diego (Finalizado el tercer día del sprint)

Tiempo estimado 1 semana.

Se completó en 2 semanas.

Problemas:

Surgieron unos problemas con el hibernate y el netbeans lo que retrasó considerablemente el sprint pero se acabó arreglando.

Review Sprint: Se ha trabajado bien salvo por el problema del hibernate-netbeans que sustrajo tiempo de desarrollo.

-2º Sprint:13 mayo

Objetivo: Implementar la interfaz xhtml, mvc , y la funcionalidad de la aplicación.

-División del trabajo:

- Modificación de la base de datos: Diego(Se iba haciendo según se necesitaba y se cambiaba si se nos ocurría una mejor idea , terminó el 31 mayo)

- Implementar los controladores o beans para hacer de puente entre los modelos y la interfaz: Diego / Enrique(Se fue implementando gradualmente hasta el 31 de mayo donde se implementaron los últimos controladores que faltaban)

- Implementar la clase modelo para comunicarse con la base de datos usando hibernate:Enrique(Se fue implementando gradualmente hasta el 31 de mayo donde se implementaron los últimos controladores que faltaban)

- Creación de las interfaces xhtml usando JSF:Enrique/Diego(Se fue implementando gradualmente hasta el 31 de mayo)

- Creación pruebas caja blanca/ caja negra:Enrique(Implementado el 31 de mayo)

- Modificación de la memoria final:Enrique(31 de mayo finalizó)

Tiempo estimado 2 semanas .

Se completó en 2 semanas y 3 días.

Problemas:

Debido a que Diego está de erasmus hubo ciertos problemas con el horario que impidió que el desarrollo de la aplicación fuera más eficiente o rápido, esto sumado con otros factores como exámenes le quitaron tiempo de desarrollo a este sprint y otra vez problemas con

netbeans y hibernate. También surgieron problemas con el JSF y la interfaz pero no afecta a la funcionalidad de la aplicación , solamente muestra un mensaje de error que no afecta al funcionamiento.

Review Sprint: Se trabajó con el tiempo justo pero se acabó terminando el proyecto . Se hicieron algunos cambios como por ejemplo que la aplicación paso de ser planeada para sangre/glóbulos/plaquetas a solamente sangre, pero se añadió un historial de pedidos completados.