TFG JAVA EE: Loulou Services

Entrega Final

Francisco Javier Guerrero de Pablo

16/06/2014

Índice

1. Funciones básica del proyecto

Resumen conceptual

Como Trabajo de Fin de Grado (TFG) se va a realizar el análisis, diseño, implementación, prueba y documentación de una aplicación Web para gestionar la búsqueda y selección de personal doméstico y administrar los servicios de limpieza contratados por los clientes de MasVidaRed.[1]

Loulou Services crea el área denominada Loulou Agency con la misión de satisfacer la externalización del servicio ofrecido por MasVidaRed de búsqueda y selección de personal doméstico. Se trata de una cartera de clientes cerrada y de gran tamaño.

El otro servicio ofrecido por Loulou Services se denomina Loulou Cleanner y consiste en la tramitación de los servicios de limpieza solicitados por los clientes.

Este proyecto reúne los estudios y competencias adquiridas a lo largo del Grado en Ingeniería Informática y su especialización en el área de Ingeniería del software. Serán necesarios unos conocimientos técnicos en análisis y diseño, así como en bases de datos y programación.

El área del TFG es **JavaEE (Java Enterprise Edition)** por lo que ha sido necesario un estudio de su tecnología, así como sus patrones de diseño y documentación.

Debido a la gran cantidad de información, a la complejidad del negocio y al número de clientes, se va a desarrollar el proyecto en la tecnología Java EE.

Justificación

MasVidaRed es una empresa dedicada a proveer servicios que permitan conciliar en un mayor grado de satisfacción la vida laboral y familiar de sus clientes, los cuales son empleados de grandes empresas y/o socios de alto nivel, como entidades pertenecientes al sector financiero.

Debido a la situación actual, en un momento dado, MasVidaRed considera la opción de una posible externalización de algunos de los servicios que ofrece con el fin de poder controlar los gastos de la compañía, así como focalizarse en otros servicios que proporcionan un mayor valor a la organización.

[1] http://www.masvidared.com/ - http://www.masvidared.com/servicios.htm#domestico

Loulou Services nace con la idea de cubrir esta externalización del servicio, y además, ofrecer un servicio de limpieza doméstica a los clientes, debido a que en el mercado actual no existen grandes compañías dedicadas a ello.

El servicio de búsqueda y selección de personal doméstico, al tratarse de una externalización, sólo ofrecerá a los clientes el perfil de los empleados candidatos y diferentes formas de contactar con ellos (teléfono, email), sin ser misión de la compañía la negociación del salario y su contratación, quedando a cargo del contratante.

La aplicación Web que se desarrollará durante el proyecto debe cubrir las necesidades siguientes:

- Debe permitir a los clientes darse de alta en la compañía, así como modificar sus datos personales y consultarlos.

- Debe permitir dar de alta y/o baja a los empleados, así como poder modificar y consultar sus datos en cualquier momento.

- Debe permitir poder consultar la información relativa a un servicio, pudiendo acceder a los datos del cliente y el empleado que lo realiza. También se podrán consultar los históricos según servicio, empleado y cliente.

- Debe permitir añadir, consultar y modificar los servicios de limpieza ofrecidos a los clientes

2. Casos de uso de la aplicación

Así pues, tras una reunión con los responsables de la compañía Loulou Services, se han definido con bastante precisión los requisitos funcionales que, en una primera versión, debe ofrecer la aplicación de comercio electrónico para servicios de atención personal.

Estos casos de uso serán separados en tres componentes principalmente: Cliente, Servicio y Catálogo.

Para cada una de las funcionalidades se especifican los usuarios (Actores) que harán uso de ellas: Cliente(C), Empleado (E) y Administrador (A).

Componente Cliente

- CU_01: Registrarse en el sistema: (C,A)
- CU_02: Añadir dirección (C,A)
- CU_04: Consultar datos personales: (C,E,A)

- CU_05: Modificar datos personales: (C,A)
- CU_06: Cambiar contraseña (C,E,A)
- CU_20: Listar todos los clientes (A)
- CU_21: Buscar clientes por Id (A)
- CU_24: Direcciones por cliente (C,A)
- CU_25: Ver dirección (C,A)
- CU_26: Modificar dirección (C,A)

Componente Servicio

- CU_07: Solicitar servicio de limpieza (C)
- CU_08: Listar tipos de servicio de limpieza (C,A)
- CU_09: Cancelar servicio de limpieza : (C,A)
- CU_10: Mostrar servicio de limpieza: (C,A)
- CU_11: Ver Servicio de limpieza por Id: (C,A)
- CU_12: Listar servicios de limpieza por Cliente : (C,A)
- CU_13: Listar todos los servicios de limpieza(A)
- CU_23: Modificar servicio de limpieza(C, A)

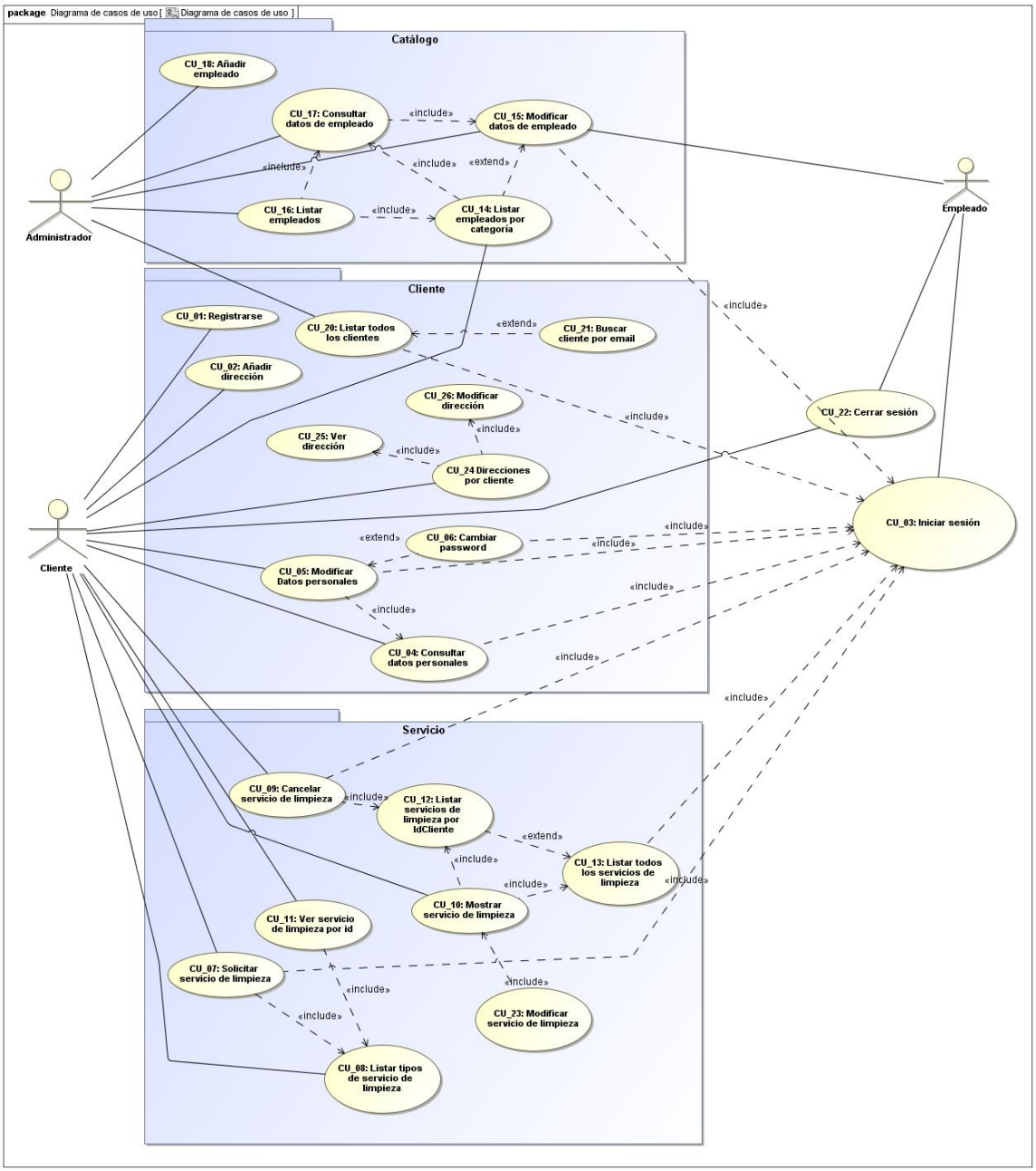
Componente Catálogo

- CU_14: Buscar empleado por categoría (C,A)
- CU_15: Buscar empleado(A)
- CU_16: Buscar empleado por Id (A)
- CU_17: Consultar estadísticas de empleado (A)
- CU_18: Añadir empleado (A)

Casos de uso comunes

- CU_03: Iniciar sesión en el sistema (C,E,A)
- CU_22: Cerrar sesión (C,E,A)

A continuación, se muestra el diagrama de casos de uso descrito anteriormente donde se pueden apreciar los 3 componentes principales en los que se ha dividido la aplicación, así como los 3 roles de usuario que harán uso de ella.



A pesar de que no está especificado de manera explícita en el diagrama anterior, el actor "Administrador" puede utilizar todos los casos de uso descritos. Se ha decidido únicamente enlazar los más relevantes de su rol para no tener un diagrama de difícil lectura, debido a la gran cantidad de casos de uso que contempla el modelo.

2.1.Especificación detallada de casos de uso

Componente Cliente

Caso de Uso		▪ **CU_01: Registrarse en el sistema: (C) Register**
Actor Principal		Usuario --> (Cliente)
PreCondición		No estar registrado ni logueado
PostCondición		Estar registrado y logueado
Casos de uso relacionados		▪ CU_21: Buscar clientes por email
Escenario principal	**1**	El usuario introduce su email , una contraseña y la repite.
	2	El usuario hace clic en el botón registrarse y se loguea automáticamente.
Flujo alternativo	**2.1**	El email introducido ya existe. Se le comunica con un mensaje emergente.
	2.2	Las contraseña no coinciden

Caso de Uso		▪ **CU_02:Añadir dirección: (C) : addServiceAddress**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Dirección registrada o cancelado por el usuario.
Casos de uso relacionados		Ninguno
Escenario principal	**1**	El usuario hace clic enmis direcciones.
	2	Selecciona añadir dirección e introduce los datos.
	3	Selecciona aceptar y se registra la dirección
Flujo alternativo	**2.1**	Ya existe esa dirección registrada para ese usuario
	2.2	Ha seleccionado Cancelar. No se hace nada

Caso de Uso		▪ **CU_04:Consultar datos personales: (C) : personalData**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_21: Buscar clientes por email
Escenario principal	**1**	El usuario hace clic en el boton Mi cuenta
	2	El sistema muestra la información personal del usuario actual.

Caso de Uso		▪ **CU_05:Modificar datos personales: (C) : modifyPersonalData**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_21: Buscar clientes por email
Escenario principal	**1**	El usuario hace clic en el boton Mi cuenta
	2	El sistema muestra la información personal del usuario actual.
	3	El usuario valida los cambios haciendo clic en guardar.
Flujo alternativo	**3.1**	Las contraseñas no coinciden. Se le comunica con un mensaje emergente.
	3.2	Ha seleccionado Cancelar. No se hace nada.

Caso de Uso		▪ **CU_06:Cambiar Password: (C) : changePassword**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Se ha modificado la contraseña
Casos de uso relacionados		▪ CU_21: Buscar clientes por email
Escenario principal	**1**	El usuario hace clic en modificar password
	2	El usuario introduce la contraseña y la repite.
	3	El usuario valida los cambios haciendo clic en guardar.
Flujo alternativo	**3.1**	Las contraseñas no coinciden. Se le comunica con un mensaje emergente.
	3.2	Ha seleccionado Cancelar. No se hace nada.

Caso de Uso		▪ **CU_20:Listar todos los clientes: (A) : allClients**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_21: Buscar clientes por email
Escenario principal	**1**	Se hace clic en listar todos los clientes
	2	El sistema muestra la lista de todos los clientes registrados.

Caso de Uso		▪ **CU_21:Buscar clientes por email: (A) : showClient**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Listar todos los clientes
Escenario principal	**1**	Se selecciona un usuario de una lista
Flujo alternativo	**1.1**	El email introducido no pertenece a ningún cliente del sistema.
	1.2	Ha seleccionado Cancelar. No se hace nada.

Caso de Uso		▪ **CU_24: Direcciones por cliente: (A) (C): serviceAddressByClient**
Actor Principal		Administrador, Cliente
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Ninguno
Escenario principal	**1**	Lista las direcciones del cliente registrado, o seleccionado.
Flujo alternativo	**1.1**	El usuario no tiene direcciones añadidas. Se le da la opción de añadir una.
	1.2	Ha seleccionado Cancelar. No se hace nada.

Caso de Uso		▪ **CU_25: Ver dirección (A) (C): showServiceAddress**
Actor Principal		Administrador, Cliente
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Listar direcciones por cliente.
Escenario principal	**1**	Se muestra los datos de la dirección seleccionada.
Flujo alternativo	**1.1**	El usuario no tiene direcciones añadidas. Se le da la opción de añadir una.
	1.2	Ha seleccionado Cancelar. No se hace nada.

Caso de Uso		▪ **CU_26: Modificar dirección: (A) (C): modifyServiceAddress**
Actor Principal		Administrador, Cliente
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Ninguno
Escenario principal	**1**	Se muestra los datos de la dirección seleccionada y permite editarlos.
Flujo alternativo	**1.1**	El usuario no tiene direcciones añadidas. Se le da la opción de añadir una.
	1.2	Ya existe una dirección registrada con los nuevos datos introducidos.
	1.3	Ha seleccionado Cancelar. No se hace nada.

Componente Servicio

Caso de Uso		▪ **CU_07: Solicitar servicio de limpieza: (C) : addService**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_13: Listar todos los servicios de limpieza
Escenario principal	**1**	El usuario selecciona un servicio de limpieza
	2	El usuario selecciona contratar servicio de limpieza
Flujo alternativo	**2.1**	El usuario selecciona periodicidad para el servicio que desea contratar

Caso de Uso		▪ **CU_08: Listar tipos de servicio de limpieza: (C) : serviceByCategory**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_13: Listar todos los servicios de limpieza
Escenario principal	**1**	El usuario selecciona un servicio de limpieza
	2	El usuario selecciona visulizar el servicio o contratarlo.
Flujo alternativo	**2.1**	El usuario selecciona periodicidad para el servicio que desea contratar

Caso de Uso		▪ **CU_09: Cancelar servicio de limpieza: (c) : cancelService**
Actor Principal		Cliente
PreCondición		Estar registrado , logueado y con un servicio de limpieza activo.
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_12: Buscar servicio de limpieza por Cliente: (C)
Escenario principal	**1**	El usuario selecciona un servicio de limpieza para ver los detalles
	2	El usuario selecciona cancelar el servicio de limpieza.
Flujo alternativo	**2.1**	El servicio ya se está ejecutando o se ha finalizado

Caso de Uso		▪ **CU_10: Mostrar servicio de limpieza: (C) : showService**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado.
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_13: Listar todos los servicios de limpieza
Escenario principal	**1**	El usuario selecciona un servicio de limpieza para ver los detalles
	2	El sistema muestra los detalles del servicio de limpieza seleccionado

Caso de Uso		▪ **CU_11: Ver servicio de limpieza por Id: (C) : showServicePlanning**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado.
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_13: Listar todos los servicios de limpieza
Escenario principal	**1**	El usuario introduce un id de servicio de limpieza y selecciona buscar
	2	El sistema muestra los detalles del servicio de limpieza
Flujo alternativo	**2.1**	No existe ningún servicio de limpieza con ese Id

Caso de Uso		▪ **CU_12: Listar servicios de limpieza por Cliente: (C) : servicesByclient**
Actor Principal		Cliente
PreCondición		Estar registrado y logueado.
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_13: Listar todos los servicios de limpieza
Escenario principal	**1**	El usuario selecciona mis servicios solicitados del apartado mi cuenta
	2	El sistema muestra todos los servicios asociados a ese cliente
Flujo alternativo	**2.1**	No existe ningún servicio de limpieza para ese cliente

Caso de Uso		▪ **CU_13: Listar todos los servicios de limpieza: (A) : allServices**
Actor Principal		Administrador
PreCondición		Estar registrado y logueado.
PostCondición		Ninguna
Casos de uso relacionados		Ninguno
Escenario principal	**1**	El adminsitrador selecciona listar todos los servicios
	2	El sistema muestra todos los servicios de limpieza registrados

Caso de Uso		▪ **CU_23: Modificar servicio de limpieza: (A) : modifyService**
Actor Principal		Administrador
PreCondición		Estar registrado y logueado.
PostCondición		Ninguna
Casos de uso relacionados		Listar servicios por cliente
Escenario principal	**1**	El usuario cancela el servicio.
	2	El administrador modifica los datos del servicio.

Componente Catálogo

Caso de Uso		▪ **CU_14: Listar empleados por categoría: (A) : EmployeeByCategory**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Ninguno
Escenario principal	**1**	El administrador selecciona una categoría y hace clic en aceptar
	2	El sistema muestra los empleados de esa categoría
Flujo alternativo	**2.1**	El email introducido ya existe. Se le comunica con un mensaje emergente.

Caso de Uso		▪ **CU_15: Modificar datos de empleado: (A,E) : modifyEmployeeData**
Actor Principal		Administrador, Empleado
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_14: Listar empleados por categoría: (A)
		▪ CU_16: Buscar empleado por email: (A,E)
		▪ CU_17: Consultar datos de empleado: (A)
Escenario principal - Administrador	**1**	El administrador accede a la ficha técnica de un empleado
	2	Modifica los datos oportunos y selecciona la opción guardar cambios
Escenario principal - Empleado	**A**	El empleado selecciona modificar sus datos personales
	B	Tras modificar los datos, selecciona guardar cambios

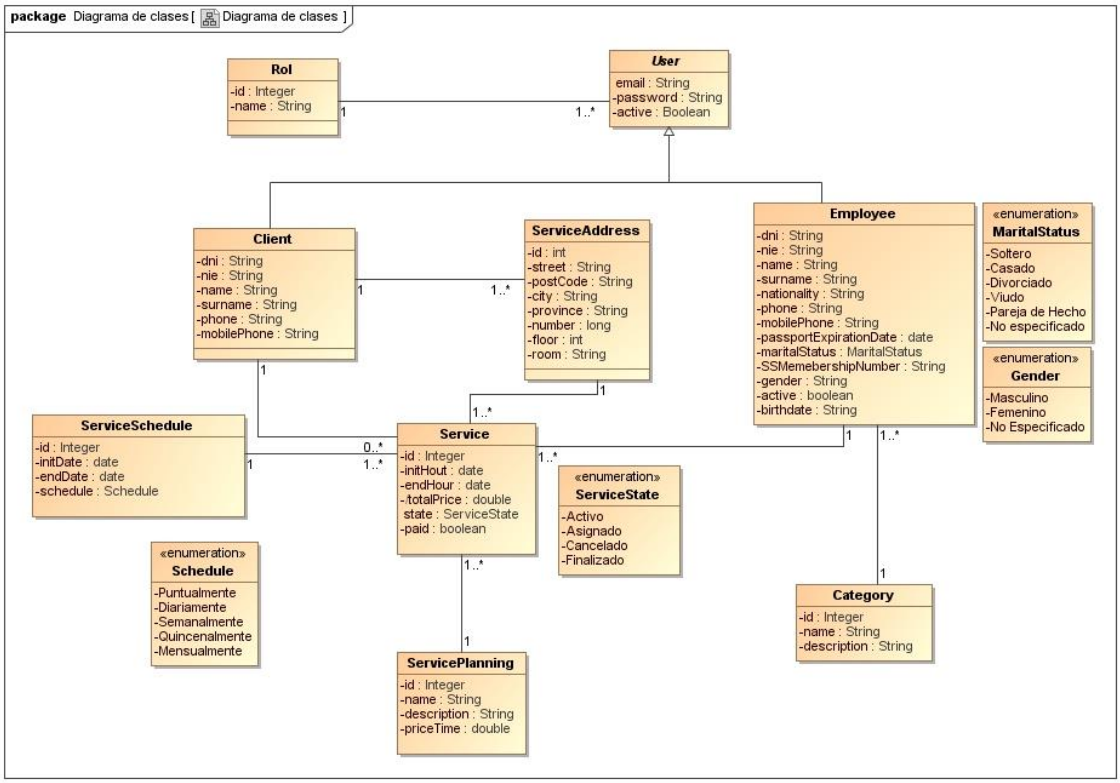
Caso de Uso		▪ **CU_16: Buscar empleado por email: (A) : employeeByEmail**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		Ninguno
Escenario principal	**1**	El administrador introduce un email y le da a buscar
	2	El sistema muestra el empleado
Flujo Alternativo	**2.1**	El email introducido no existe

Caso de Uso		▪ **CU_17: Consultar datos de empleado (A) : employeeData**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Ninguna
Casos de uso relacionados		▪ CU_15: Modificar datos de empleado: (A,E)
Escenario principal	**1**	Se selecciona la opción ver detalles de un empleado
	2	El sistema muestra los datos del empleado seleccionado o del logueado

Caso de Uso		▪ **CU_18: Añadir empleado (A) : addEmployee**
Actor Principal		Administrador
PreCondición		Estar logueado
PostCondición		Se ha añadido un nuevo empleado al sistema.
Casos de uso relacionados		▪ CU_16: Buscar empleado por email: (A)
Escenario principal	**1**	El administrador selecciona añadir empleado
	2	El administrador rellena todos los datos y selecciona guardar cambios
Flujo Alternativo	**2.1**	El email introducido ya existe.

3. Diagrama estático

A continuación se muestra el diseño estático de clases del sistema.



4. Arquitectura

El proyecto se va a realizar basándose en la tecnología Java EE y será una aplicación web basada en el modelo cliente/servidor. La parte cliente se basa de una única capa, que estaría formada por cualquier navegador Web estándar. La parte Web, sin embargo, estará formada por tres capas:

- Una capa encargada de la **presentación**: permitirá la interacción de los usuarios con el servicio.
- Otra capa será la encargada de la **lógica de negocio**: implementa las funcionalidades básicas de la aplicación.

- Y otra capa de **integración o datos**: permiten interactuar con las fuentes de datos que almacenan la información persistente (principalmente, BBDD relacional).

Nos interesa desacoplar la interfaz gráfica de nuestro sistema del resto del sistema, para que el mantenimiento no resulte muy costoso, al ser una parte muy propensa a cambios. Se utilizará pues, el patrón MVC (Model-view-controller).

4.1. Componentes de la capa de presentación

Habrá un único controlador para todas las operaciones de cada componente (patrón FrontController). Se utilizará un esquema basado en el patrón Command, donde el controlador será el responsable de coordinar todo el proceso, pero la implementación está desacoplada. Cada acción que se pueda realizar en la interfaz de usuario se mapeará con una vista y el controlador ejecutará esas acciones.

Así pues, utilizaremos el framework Java Server Faces (JSF) 2.2 que viene con la especificación de Java EE. El controlador lo hará el servlet 'Faces Servlet' que ya tenemos disponible si utilizamos JSF.

Las acciones que corresponden a los Commands estarán definidas con ManagedBeans. Las vistas las implementaremos con Facelets.

Especificación de Managed Bean

A continuación se muestra una breve explicación de las principales funcionalidades ofrecidas por los Managed Bean por componentes.

Componente Catálogo

- **ListEmployeesMBean**: lista los empleados del sistema según categoría, disponibilidad y tipo de empleado (de limpieza o de hogar).
    - Se relaciona con las vistas 'employeeListView' y 'showUserView'
- **ModifyEmployeeMbean:** permite modificar los datos de un empleado.
    - Se relaciona con las vistas 'EmployeeListView.xhtml' y 'showUserView.xhtml'.
- **RegisterEmployeeMBean:** permite al administrador registrar un empleado en el sistema.
    - Se relaciona con las vistas 'registerEmployeeView.xhtml', 'employeeListView.xhtml' y 'showUserView.xhtml'.
- **ShowEmployeeMBean:** obtiene los datos de un cliente para ser visualizados.
    - Se relaciona con las vistas 'employeeListView.xhtml' y 'showUserView.xhtml'.

Componente Servicio

- **ServiceMBean**: permite añadir un servicio al sistema asociándolo a un tipo de servicio, dirección, cliente y plan.

- Se relaciona con las vistas 'listServicesView.xhtml' 'service.xhtml' y 'showService.xhtml'

- **ServicePlanningMBean**: Permite visualizar, añadir y modificar los plannings ofrecidos.

  - Se relaciona con las vistas 'servicePlanningListView.xhtml', 'showServicePlanning.xhtml' y 'modifyServicePlanningView.xhtml'.

Componente cliente

- **ChangePasswordMBean**: permite la modificación de la contraseña de un usuario. Si el usuario logueado es cliente o empleado, sólo pueden modificar su contraseña.

  Si el usuario es administrador puede cambiar la contraseña de todos los usuarios.

  - Se relaciona con las vistas 'showUserView.xhtml' 'changePasswordView .xhtml' . Si el usuario es administrador no debe introducir la contraseña antigua.

- **ListClientsMBean:** obtiene una lista de todos los clientes del sistema.
  - Se relaciona con la vista 'clientListView.xhtml' y 'showUserView.xhtml'.

- **ListUsersMBean:** obtiene todos los usuarios registrados en el sistema (Clientes, Empleados y Administradores).

  - Se relaciona con las vistas 'userListView.xhtml' y 'showUserView.xhtml'.

- **LoginMbean:** permite que un usuario registrado inicie sesión en el sistema.

  - Se relaciona con las vista 'HomeView.xhtml' aunque sus datos son utilizados para "renderizar" los elementos de las vistas según el rol del usuario registrado.

- **LogoutMBean:** permite a un usuario que tiene la sesión iniciada cerrarla y salir del sistema.

  - Se relaciona con la vista 'HomeView.xhtml'.

- **ModifyClientMBean:** permite modificar los datos de un cliente.

  - Se relaciona con las vistas 'clientListView.xhtml'

- **RegisterMBean:** permite a un usuario darse de alta en el sistema como cliente.

- Se relaciona con las vistas 'homeView.xhtml' , 'registerView.xhtml' y 'showUserView.xhtml'

- **ServiceAddressMBean:** permite añadir, visualizar y editar direcciones de un cliente.

  - Se relaciona con las vistas 'addServiceAddressView.xhtml', serviceAddressListView.xhtml' y 'modifyServiceAddressView.xhtml'.

- **ShowClientMBean:** obtiene los datos de un cliente.

  - Se relaciona con las vistas 'clientListView.xhtml' y 'showUserView.xhtml'.

- **ShowUserMBean:** obtiene los datos de un usuario**.**

  - Se relaciona con las vistas 'userListView.xhtml' y 'showUserView.xhtml'

4.2.Componentes de la capa de negocio

Implementaremos la capa de negocio siguiendo un patrón Fachada (Facade), reduciendo de esta manera el acoplamiento y ofreciendo un único punto de entrada al subsistema consiguiendo una mayor seguridad y abstracción de las clases de la capa de negocio.

Nos interesa dar acceso tanto local como remoto a los componentes de negocio. Se decide que sea el contenedor Java EE quien gestione las transacciones ya que las operaciones a realizar son síncronas y no tienen estado.

Por tanto, la mejor opción para implementar el componente de negocio es un EJB (Enterprise JavaBeans) de sesión sin estado para que el contenedor Java EE nos resuelva toda la complejidad inherente a las comunicaciones remotas, comportamiento transaccional y la gestión de la seguridad.

Especificación de Enterprise Java Beans

A continuación se muestra una breve explicación de las principales funcionalidades ofrecidas por los Enterprise Java Beans por componentes.

Componente Catálogo

Este componente está compuesto de una clase llamada '**CatalogFacadeBean**' que implementa los métodos definidos en las dos interfaces: '**CatalogFacade**' y '**CatalogFacadeRemote' .**

Las signaturas de los métodos desarrollados son:

```
public EmployeeJPA showEmployee(String email);
```

```
public Collection<EmployeeJPA> allEmployees();

public Collection<EmployeeJPA> allHomeEmployees();
public Collection<EmployeeJPA> allHomeAvailableEmployees();
public Collection<EmployeeJPA> allHomeBusyEmployees();

public Collection<EmployeeJPA> allCleannerEmployees();
public Collection<EmployeeJPA> allCleannerAvailableEmployees();
public Collection<EmployeeJPA> allCleannerBusyEmployees();

public EmployeeJPA modifyEmployeeData(EmployeeJPA employee);
public boolean addEmployee (EmployeeJPA employee);
public CategoryJPA getEmployeeCategory(long categoryType);
public RolJPA getEmployeeRol(long id);
```

Componente Servicio

Este componente está compuesto de una clase llamada '**ServiceFacadeBean**' que implementa los métodos definidos en las dos interfaces: '**ServiceFacade**' y '**ServiceFacadeRemote' .**

Las signaturas de los métodos desarrollados son:

```
public boolean addServicePlanning(ServicePlanningJPA planning);
public Collection<ServicePlanningJPA> allServicePlannings();
public ServicePlanningJPA modifyServicePlanningData(ServicePlanningJPA
dataPlanning);
public ServicePlanningJPA showServicePlanning(long id);
public ServicePlanningJPA showServicePlanning(String name);

public ServiceJPA showService(long serviceId);
public ServiceJPA addService(ServiceJPA service);
public ServiceScheduleJPA getServiceSchedule(Schedule serviceSchedule);
public ServiceScheduleJPA addSchedule(ServiceScheduleJPA serviceSchedule);
public ServiceScheduleJPA showSchedule(Schedule schedule);
public Collection<ServiceJPA> allServices();
public ServiceJPA modifyServiceData(ServiceJPA dataService);
```

Componente Cliente

Este componente está compuesto de una clase llamada '**ClientFacadeBean**' que implementa los métodos definidos en las dos interfaces: '**ClientFacade**' y '**ClientFacadeRemote' .**

Las signaturas de los métodos desarrollados son:

```
public UserJPA login(String email, String pwd);
public void logout();

public Collection<UserJPA> allUsers();
public UserJPA changePassword(UserJPA user, String newPass, String oldPass);


public UserJPA showUser(String email);
```

```
public ClientJPA showClient(String email);

public ClientJPA addClient(ClientJPA client);
public RolJPA getClientRol(long id);
public Collection<ClientJPA> allClients();
public ClientJPA modifyPersonalData(ClientJPA client);

public ServiceAddressJPA addServiceAddress(ServiceAddressJPA address);
public ServiceAddressJPA modifyServiceAddress(ServiceAddressJPA address);
public ServiceAddressJPA showServiceAddress(Long id);
```

4.3.Componentes de la capa de persistencia

Los datos de la aplicación se almacenarán en una base de datos relacional, y el acceso a la capa de integración desde la capa de negocio será local, por lo que se ha decidido implementar los componentes con JPA 2.0 (Java Persistence API).

Especificación de entidades JPA

A continuación se muestra una breve explicación de las entidades JPA manejadas por la aplicación.

- **RolJPA:** Representa la información relativa a los distintos roles que puede haber en la organización. En este caso se han definido 3: 'Administrator', 'Client' y 'Employee'.

- **UserJPA:** Representa el email del usuario, su contraseña e indica si está activo o no el usuario en el sistema.

- **ClientJPA:** Representa toda la información relativa a un cliente y hereda de la clase 'UserJPA', por lo que tiene sus atributos.

- **EmployeeJPA:** Representa toda la información relativa a un empleado y hereda de la clase 'UserJPA', por lo que tiene sus atributos. Un empleado puede ser de dos categorías diferentes('CategoryJPA') siendo del hogar ('Agency') o de limpieza ('Cleanner').

- **ServiceJPA:** Representa toda la información relativa a un servicio. Cliente asociado, plan solicitado, dirección seleccionada, fecha de inicio y fin, estado del servicio, etc.

  Esta entidad representa la piedra angular del sistema, siendo la encargada de encajar casi todos los datos del sistema.

- **ServiceAddressJPA:** Representa la información relativa a la dirección de un cliente. Por el momento, sol o se contempla trabajar en la comunidad de Madrid, motivo por el cual no se permite editar la provincia.

- **ServiceScheduleJPA:** Representa la programación de un servicio, que puede ser: 'Puntual', 'Diariamente', 'Semanalmente', 'Quincenalmente' y 'Mensualmente'.

- **ServicePlanningJPA:** Representa la información relativa a los servicios ofrecidos con su nombre y descripción.

- **CategoryJPA:** Representa la categoría del empleado, habiendo actualmente dos: hogar ('Agency') o de limpieza ('Cleanner').

5. Software a utilizar

A continuación, se mencionan las principales aplicaciones software a utilizar para el desarrollo.

Diseño

- **MagicDraw UML Version 17.0**: Esta aplicación se utilizará para diseñar e implementar los diagramas necesarios para el proyecto.
  http://www.nomagic.com/products/magicdraw.html
  http://www.nomagic.com/support/documentation.html

- **Eclipse** Java EE IDE for Web Developers Version **Kepler** 4.3 Release
  http://www.eclipse.org/downloads/
  http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/kepler/R/eclipse-jee-kepler-R-win32.zip&mirror_id=514

Arquitectura

- **Servidor de aplicaciones: JBoss 7.1.1**
  http://www.jboss.org/jbossas/downloads
  http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.zip

- **SGBD: PostgreSQL 9.3**
  http://www.enterprisedb.com/products-services-training/pgdownload#windows .
  http://www.postgresql.org/docs

6. Configuración inicial

Actualización de JBoss para permitir JSF 2.2

Para poder utilizar "Java Server Faces 2.2" se ha tenido que actualizar el servidor JBoss y realizar configuraciones oportunas en el proyecto de Eclipse. Se pueden consultar los pasos realizados para la actualización de JBoss en el siguiente enlace:

https://community.jboss.org/thread/203257

Se ha tenido que modificar el fichero 'faces-config.xml' para ajustarse a la actualización:

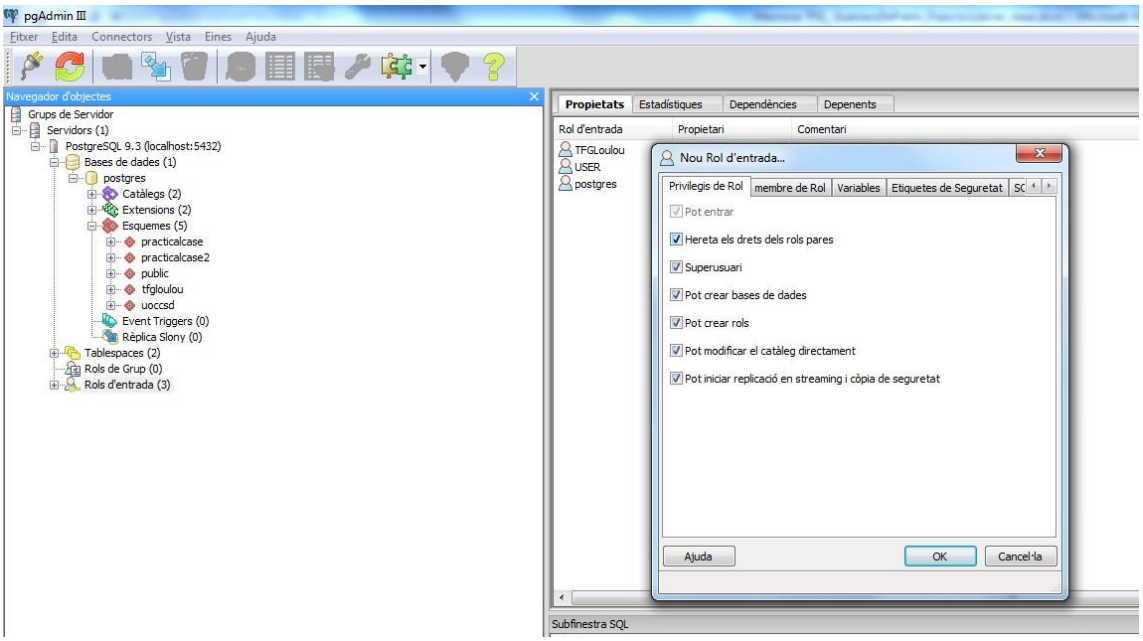
```
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee;
	http://java.sun.com/xml/ns/javaee/web-facesconfig_2_2.xsd" version="2.2">
	<lifecycle>
		<phase-listener>managedbean.MultiPageMessagesSupport</phase-listener>
	</lifecycle>
</faces-config>
```

Además, se ha de modificar el fichero '\standalone\configuration\standalone.xml' del servidor creando un nuevo datasource asociado al proyecto donde se define el usuario y la contraseña que accederán al esquema de base de datos.

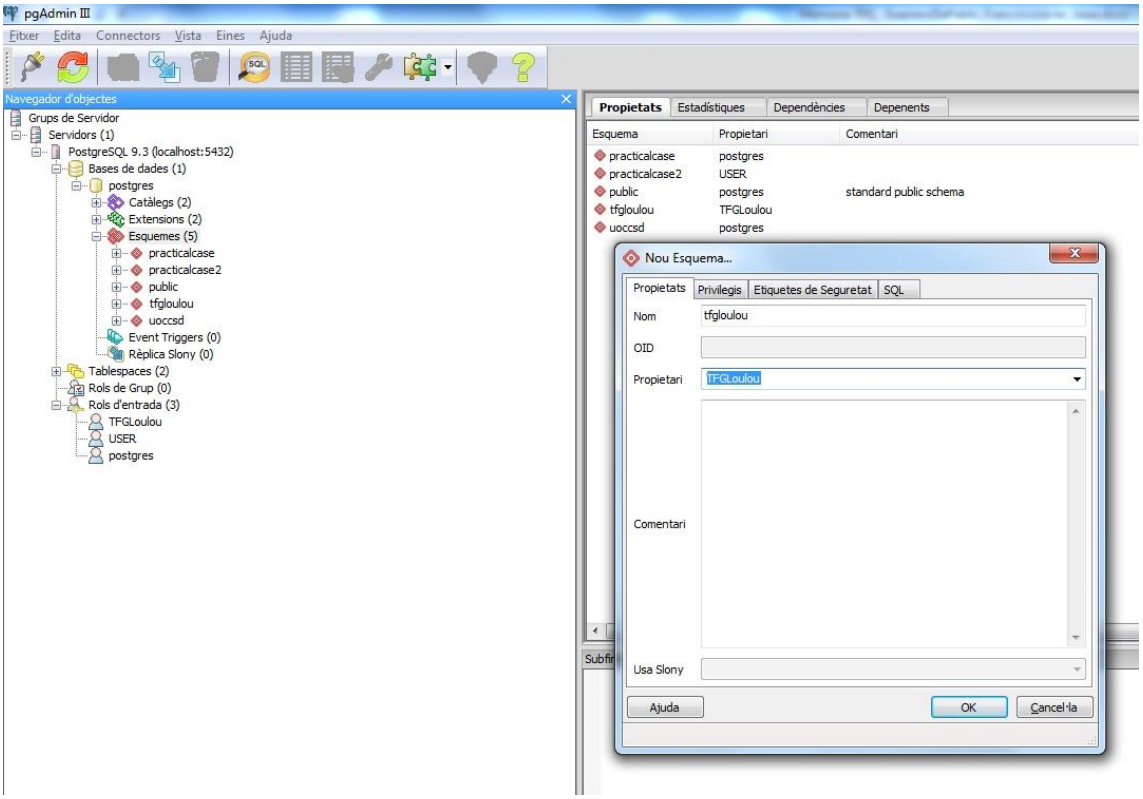
```
<datasource jta="false" jndi-name="java:jboss/TFGLoulouDS" pool-name="TFGLoulouDS"
enabled="true" use-java-context="true" use-ccm="false">
	<connection-url>jdbc:postgresql://localhost:5432/postgres</connection-url>
	<driver-class>org.postgresql.Driver</driver-class>
	<driver>postgresql</driver>
	 <security>
		<user-name>TFGLoulou</user-name>
		<password>Loulou</password>
	 </security>
</datasource>
```

Configuración de PostgreSQL

Se ha de crear un nuevo rol de entrada en Base de datos con nombre 'TFGLoulou' y contraseña 'Loulou'. Este usuario debe tener todos los privilegios de Rol habilitados, tal y como se puede ver en la siguiente imagen:



Se ha de crear un nuevo schema denominado 'tfgloulou' tal y como se ha definido en el apartado anterior.



Posteriormente, se deben crear las tablas necesarias para la base de datos y sus registros iniciales de configuración.

Creación de tablas

```
-- Table: tfgloulou.category

-- DROP TABLE tfgloulou.category;

CREATE TABLE tfgloulou.category
(
  id bigint NOT NULL,
  description character varying(255),
  name character varying(255),
  CONSTRAINT category_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.category
  OWNER TO "TFGLoulou";


-- Table: tfgloulou.employee

-- DROP TABLE tfgloulou.employee;

CREATE TABLE tfgloulou.employee
(
  ssmembershipnumber character varying(255),
  available boolean NOT NULL,
  birthdate character varying(255),
  dni character varying(255),
  gender integer,
  maritalstatus integer,
  mobile character varying(255),
  name character varying(255),
  nationality character varying(255),
  nie character varying(255),
  passportexpirationdate character varying(255),
  surname character varying(255),
  telephone character varying(255),
```

```
  user_id character varying(255) NOT NULL,
  category bigint,
  CONSTRAINT employee_pkey PRIMARY KEY (user_id),
  CONSTRAINT fk3031394789882148 FOREIGN KEY (category)
      REFERENCES tfgloulou.category (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk30313947b0e5c2ec FOREIGN KEY (user_id)
      REFERENCES tfgloulou."user" (email) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.employee
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.client
-- DROP TABLE tfgloulou.client;
CREATE TABLE tfgloulou.client
(
  dni character varying(255),
  mobile character varying(255),
  name character varying(255),
  nie character varying(255),
  surname character varying(255),
  telephone character varying(255),
  user_id character varying(255) NOT NULL,
  CONSTRAINT client_pkey PRIMARY KEY (user_id),
  CONSTRAINT fke89718a4b0e5c2ec FOREIGN KEY (user_id)
      REFERENCES tfgloulou."user" (email) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

```
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.client
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.rol
-- DROP TABLE tfgloulou.rol;
CREATE TABLE tfgloulou.rol
(
  id bigserial NOT NULL,
  rol integer,
  CONSTRAINT rol_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.rol
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.service
-- DROP TABLE tfgloulou.service;
CREATE TABLE tfgloulou.service
(
  id bigserial NOT NULL,
  paid boolean NOT NULL,
  state integer,
  totalprice double precision NOT NULL,
  serviceaddress bigint,
  client character varying(255),
  employee character varying(255),
  planning bigint,
```

```
  schedule bigint,
  CONSTRAINT service_pkey PRIMARY KEY (id),
  CONSTRAINT fk6d443cfc24d9c168 FOREIGN KEY (client)
      REFERENCES tfgloulou.client (user_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk6d443cfc44c24008 FOREIGN KEY (serviceaddress)
      REFERENCES tfgloulou.serviceaddress (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk6d443cfc89aac779 FOREIGN KEY (schedule)
      REFERENCES tfgloulou.serviceschedule (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk6d443cfc904449b9 FOREIGN KEY (planning)
      REFERENCES tfgloulou.serviceplanning (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk6d443cfca87b8348 FOREIGN KEY (employee)
      REFERENCES tfgloulou.employee (user_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.service
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.serviceaddress
-- DROP TABLE tfgloulou.serviceaddress;
CREATE TABLE tfgloulou.serviceaddress
(
  id bigserial NOT NULL,
  city character varying(255),
  "number" bigint NOT NULL,
  postcode character varying(255),
```

```
  province character varying(255),
  street character varying(255),
  clientaddress character varying(255),
  CONSTRAINT serviceaddress_pkey PRIMARY KEY (id),
  CONSTRAINT fk538a358e732ab66 FOREIGN KEY (clientaddress)
      REFERENCES tfgloulou.client (user_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.serviceaddress
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.serviceplanning
-- DROP TABLE tfgloulou.serviceplanning;
CREATE TABLE tfgloulou.serviceplanning
(
  id bigserial NOT NULL,
  description character varying(255),
  name character varying(255),
  pricetime double precision NOT NULL,
  CONSTRAINT serviceplanning_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.serviceplanning
  OWNER TO "TFGLoulou";

-- Table: tfgloulou.serviceschedule
-- DROP TABLE tfgloulou.serviceschedule;
```

```
CREATE TABLE tfgloulou.serviceschedule
(
  id bigserial NOT NULL,
  enddate timestamp without time zone,
  initdate timestamp without time zone,
  schedule integer,
  CONSTRAINT serviceschedule_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou.serviceschedule
  OWNER TO "TFGLoulou";

-- Table: tfgloulou."user"
-- DROP TABLE tfgloulou."user";
CREATE TABLE tfgloulou."user"
(
  email character varying(255) NOT NULL,
  active boolean NOT NULL,
  password character varying(255),
  rol bigint,
  CONSTRAINT user_pkey PRIMARY KEY (email),
  CONSTRAINT fk860072e4744792ee FOREIGN KEY (rol)
      REFERENCES tfgloulou.rol (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tfgloulou."user"
  OWNER TO "TFGLoulou";
```

Inserción de registros

```
-- Inserts: Table: tfgloulou.rol
-- 0 : administrator, 1 : client, 2 : employee
insert into tfgloulou.rol(id,rol) values (0,0);
insert into tfgloulou.rol(id,rol) values (1,1);
insert into tfgloulou.rol(id,rol) values (2,2);

-- Inserts: Table: tfgloulou.user
-- Creación del usuario por defecto
insert into tfgloulou.user(email,password,active,rol) values
('admin@loulou.com','admin',true,0);


-- Inserts: Table: tfgloulou.category
-- Creación de los dos servicios ofrecidos: cleanner y agency
insert into tfgloulou.category (id, description, name) values (1,'Servicios
domésticos por horas (Planchar, Cocinar, Limpiar)', 'LIMPIEZA');
insert into tfgloulou.category (id, description, name) values (2,'Empleadas
del hogar internas', 'EMPLEADA DEL HOGAR');

-- Inserts: Table: tfgloulou.serviceplanning
-- Creación de los servicios domésticos ofrecidos
insert into tfgloulou.serviceplanning (id,name,description,priceTime) values
(1,'Limpieza de mantenimiento','Para facilitar el día de su trabajadora fija
 o para usted, es necesario una buena limpieza a fondo de acuerdo con las
necesidades de cada uno. Así su casa estará siempre limpia.',15);


insert into tfgloulou.serviceplanning (id,name,description,priceTime) values
(2,'Limpieza general','Realizamos los trabajos de limpieza con todos los
productos
y materiales necesarios, además de los equipamientos de seguridad, o
simplemente ponemos a su disposición nuestro equipo de profesionales. ',15);

insert into tfgloulou.serviceplanning (id,name,description,priceTime) values
(3,'Limpieza de fin de obras o eventos','Limpieza realizada por un equipo
cualificado bajo supervisión y orientación de un profesional técnico, para la
aplicación correcta de los procesos adecuados a cada tipo de
superficie.',18);

insert into tfgloulou.serviceplanning (id,name,description,priceTime) values
(4,'Limpieza de siniestro','Servicio contratado al finalizar la obra, con
el objetivo de limpiar de manera adecuada y profesional los diversos tipos y
grados de suciedad',25);

insert into tfgloulou.serviceplanning (id,name,description,priceTime) values
(5,'Servicios Complementarios','Ofrecemos una gran variedad de servicios
complementarios: Limpieza de cristales cara interna y externa. Limpieza de
moquetas, alfombras, sillones o tresillos. Plancha a domicilio',0);
```

Configuración del proyecto

Fichero ‘persistence.xml’

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
              http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
       version="1.0">
   <persistence-unit name="PracticalCase">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:jboss/postgresDS</jta-data-source>
     <properties>
         <property name="hibernate.dialect"
                  value="org.hibernate.dialect.PostgreSQLDialect"/>
         <property name="hibernate.hbm2ddl.auto" value="update"/>
      </properties>
   </persistence-unit>
</persistence>
```

Fichero 'application.xml'

```
<application>
  <display-name>eAgenda</display-name>
  <module>
    <web>
      <web-uri>eAgenda.war</web-uri>
      <context-root>/eAgenda</context-root>
    </web>
  </module>
  <module>
    <ejb>eAgenda.jar</ejb>
  </module>
</application>
```

Fichero 'build.xml'

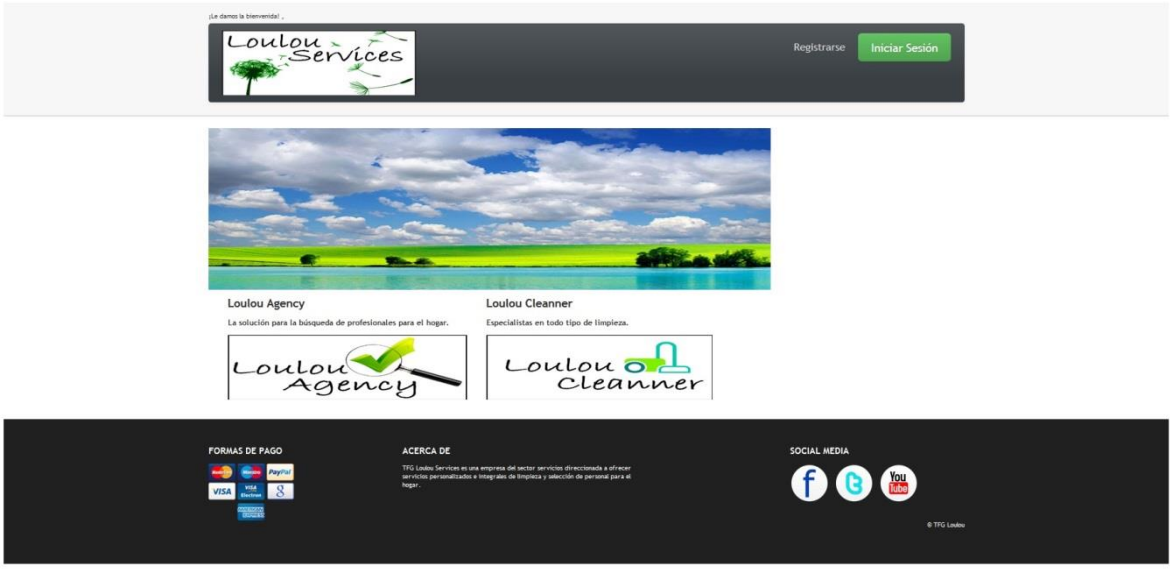
```
<project name="Practical Case Study Web service" default="all" basedir=".">

  <description>
       This is a file that compiles and distributes the Case Study, which
appears
       sample tutorials UOC </description>

  <!-- definition of global property -->
  <property environment="env"/>
  <property name="jboss.home" value="${env.JBOSS_HOME}"/>
  <property name="source" value="."/>
  <property name="sourcesrc" value="${source}/src"/>
  <property name="build" value="${source}/build"/>
  <property name="buildjar" value="${build}/jar"/>
  <property name="buildwar" value="${build}/war"/>
  <property name="dist" value="${source}/dist"/>
  <property name="jboss-config" value="default"/>
  <property name="deploy" value="${jboss.home}\standalone\deployments"/>
  <property name="jboss.module.dir" value="${jboss.home}/modules" />

  <path id="jboss.classpath">
       <fileset dir="${jboss.module.dir}">
```

```
            <include name="**/*.jar"/>
        </fileset>
    </path>


    <target name="all" depends="clean, init, compileEjb, compileWar, jarEjb,
deployClient,
        ear, deployear"/>

    <target name="init"
            description = "inicialitzacions is relevant: the structure
created
            copy files and directories there. xml ">    <!-- Crea el time-
stamp -->
        <tstamp/>
        <!-- It creates the directory structure -->
        <mkdir dir="${buildjar}"/>
        <mkdir dir="${buildwar}"/>
        <mkdir dir="${buildjar}/META-INF"/>
        <mkdir dir="${buildwar}/WEB-INF"/>
          <mkdir dir="${buildwar}/WEB-INF/classes"/>
          <mkdir dir="${dist}"/>
     </target>

    <!--Compiling the EJB classes and makes the build directory -->
    <target name="compileEjb" depends="init">
        <copy file="${sourcesrc}/META-INF/persistence.xml"
todir="${buildjar}/META-INF"/>
        <copy file="${sourcesrc}/log4j.properties" todir="${buildjar}"/>
     <javac srcdir="${sourcesrc}" destdir="${buildjar}"
            includes="/ejb/*.java, /jpa/*.java"
          classpathref="jboss.classpath"
        includeantruntime="true"
      />
    </target>

    <!-- Compile the client application, creating the structure buildwar -->
    <target name="compileWar" depends="compileEjb">
        <copy todir="${buildwar}">
            <fileset dir="${source}/docroot"/>
        </copy>
        <javac srcdir="${sourcesrc}" destdir="${buildwar}/WEB-INF/classes"
includes="/managedbean/*.java"
          classpathref="jboss.classpath"
        includeantruntime="true"
        />
    </target>

    <!-- Update the EJB jar file and create if not exist -->
    <target name="jarEjb" depends="compileEjb">
      <jar jarfile="${dist}/eAgenda.jar"
        basedir="${buildjar}"
        update="yes">
      </jar>

    </target>

    <!-- Update the WAR file and create if not exist -->
    <target name="deployClient" depends="compileWar">
```

```
    <jar jarfile="${dist}/eAgenda.war"
      basedir="${buildwar}"
      excludes="/WEB-INF/classes/ejb/*.*, /WEB-INF/classes/jpa/*.*"
      update="yes">
    </jar>
  </target>

  <!-- Update the application ear file and created if not exist -->
  <target name="ear" depends="clean, jarEjb, deployClient">
    <copy file="${sourcesrc}/META-INF/application.xml" todir="${dist}/META-
INF"/>
      <jar jarfile="${dist}/eAgenda.ear"
         basedir="${dist}" update="yes">
      </jar>
  </target>

  <!-- Deploy the ear. Copy the ear of the JBoss deployment directory -->
  <target name="deployear" depends="ear">
    <copy file="${dist}/eAgenda.ear" todir="${deploy}"/>
  </target>

  <!-- Clean the build directory -->
  <target name="clean">
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>

</project>
```

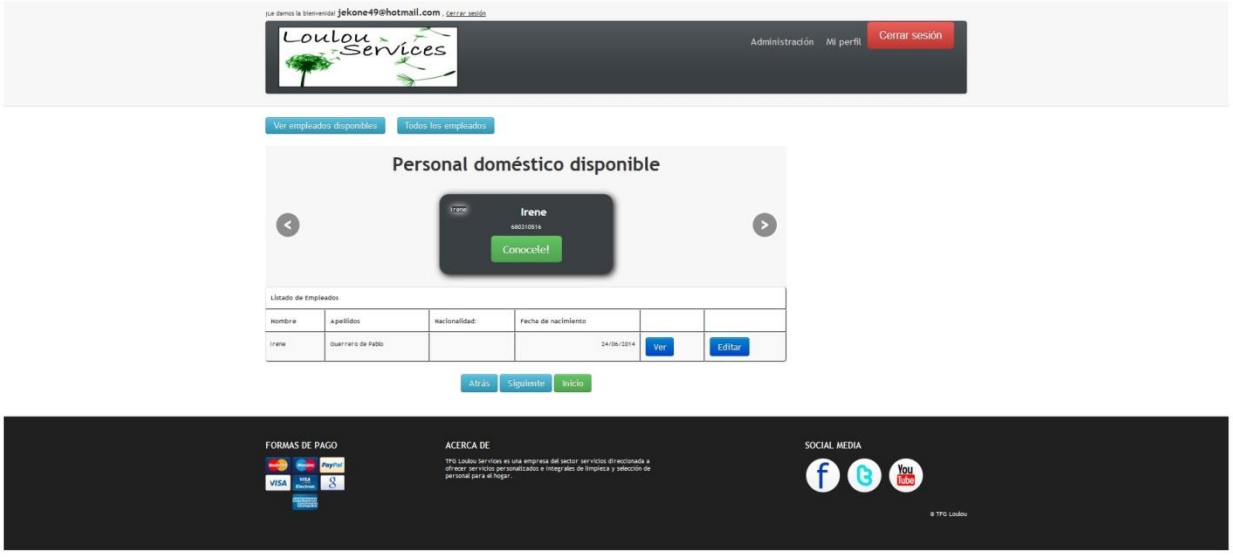
7. Principales vistas de la aplicación

homeView.xhtml

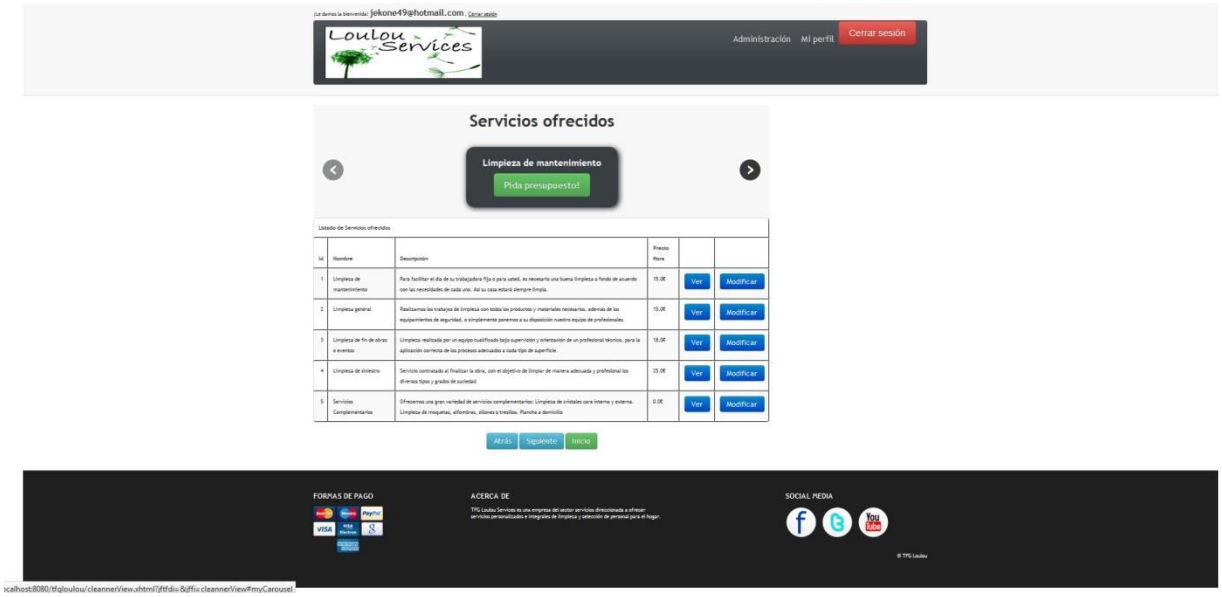


adminView.xhtml

agencyView.xhtml



cleannerView.xhtml



serviceView.xhtml