# TFC Ingeniería del Software

## Aplicación SaaS para la gestión de pequeñas empresas

### Caso de aplicación: "*Centro de belleza*"

Autor: José Luis Barrera Trancoso

Consultor: Oriol Martí Girona

# Índice

1. Introducción
2. Planificación del proyecto
3. Análisis funcional
4. Diseño técnico
5. Implementación
6. Conclusiones
7. Demo

# Introducción

## Objetivos

- Realizar el análisis y diseño de una solución tecnológica que cubra las necesidades de gestión de una pequeña empresa
  - se plantea como escenario concreto un centro de belleza
- Diseñar un sistema con las siguientes características:
  - Desplegable en un entorno ***cloud computing***
  - Con soporte a múltiples clientes de forma concurrente (***multitenancy***)
  - Diseñado para permitir modelos de explotación del software basado en servicios: proporcionarse como **SaaS (*Software as a Service*)**
  - Capacidad de adaptarse a cualquier tipo de negocio
  - Disponer de una arquitectura **modular y extensible**: añadir nuevas funcionalidades o extender las existente en base a módulos
- Realizar una implementación para la validación del análisis y el diseño

# Planificación

El proyecto se ha divido en 4 fases:

- **Fase I: Plan de trabajo (8 días)**
  - Decisión de la temática del proyecto
  - Descripción del proyecto
  - Identificación de las tareas
  - Realización de la planificación del proyecto
  - Redacción del plan de trabajo
- **Fase II: Análisis y diseño (25 días)**
  - Identificación de requisitos
  - Definición del alcance del proyecto
  - Descripción de los casos de uso
  - Prototipos de interfaz gráfica
  - Modelización de las clases de objetos y relaciones
  - Diseño de la arquitectura tecnológica
  - Identificación de tecnologías
  - Documentación del modelo de la base de datos
- **Fase III: Implementación (25 días)**
  - Instalación y configuración del entorno de desarrollo
  - Implantación y despliegue inicial
  - Implementación del modelo de clases
  - Implementación de la lógica de negocio
  - Desarrollo interfaz de usuario
- **Fase IV: Publicación de resultados (15 días)**
  - Composición y revisión de la memoria final del proyecto
  - Preparación de la presentación
  - Pruebas y corrección de errores en la aplicación

# Análisis funcional

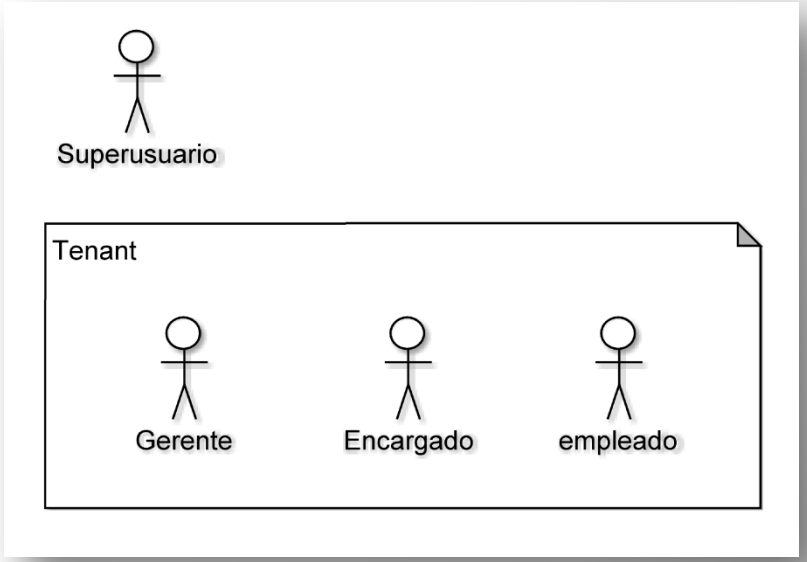
- El análisis se ha centrado en una **aplicación web de gestión para centros de belleza**, orientada a gestionar el trabajo diario que se realiza en el centro, teniendo en cuenta que:
  - sea lo suficientemente flexible para ser extendida en funcionalidades y a otro tipo de negocios
  - pueda ser utilizada por varios clientes al mismo tiempo, utilizando la misma instalación de la aplicación y la misma base de datos

- Requisitos
  - **Funcionales**. Engloba las funcionalidades básicas para la gestión diaria de un centro de belleza, en base a las entrevistas realizadas con la propietaria de uno. Estas funcionalidades abarcan la gestión de empleados, clientes, citas, servicios, productos, bonos y ventas.
  - **Técnicos**. Engloban las características técnicas necesarias para proporcionar un sistema modular y extensible, y capaz de ofrecerse en modo SaaS.

# Análisis funcional

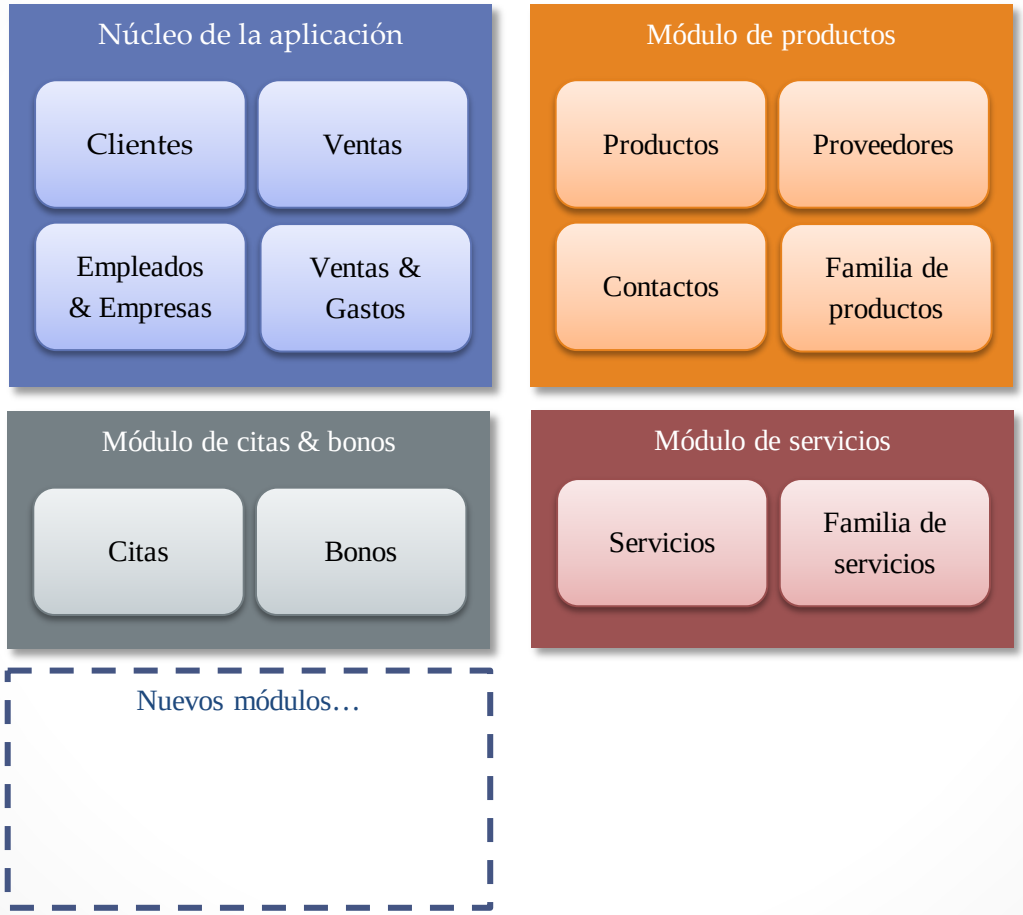
## Actores

- **Superusuario**
  - Es el dueño de la infraestructura y dispone de todos los permisos sobre los contenidos del SaaS, incluyendo la de todos los tenants
- **Tenant**
  - Es un cliente de la aplicación de gestión. No es un rol como tal, sino que un tenant hace referencia a los datos de un cliente, es decir, de una empresa o centro de belleza. **Todos los empleados de una misma empresa trabajan sobre el mismo tenant**
  - **Gerente**
    - Dispone de todos los privilegios sobre los datos de su negocio en la aplicación.
  - **Encargado**
    - Es la persona en la empresa que dispone de privilegios para gestionar todas las entidades, excepto los datos de la empresa y de los empleados
  - **Empleado**
    - Sólo tiene permisos para gestionar determinadas entidades



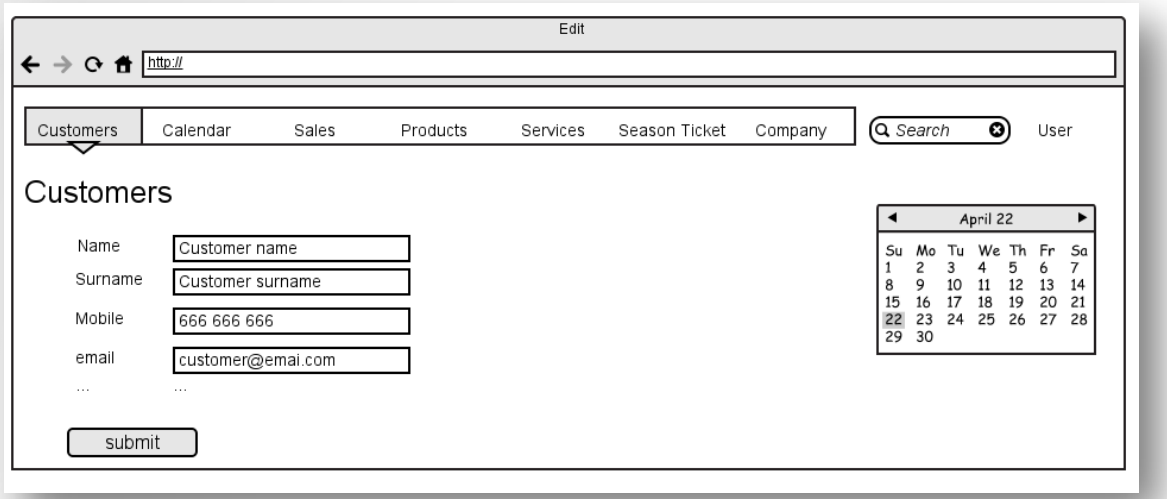
# Análisis funcional

## Diagrama funcional

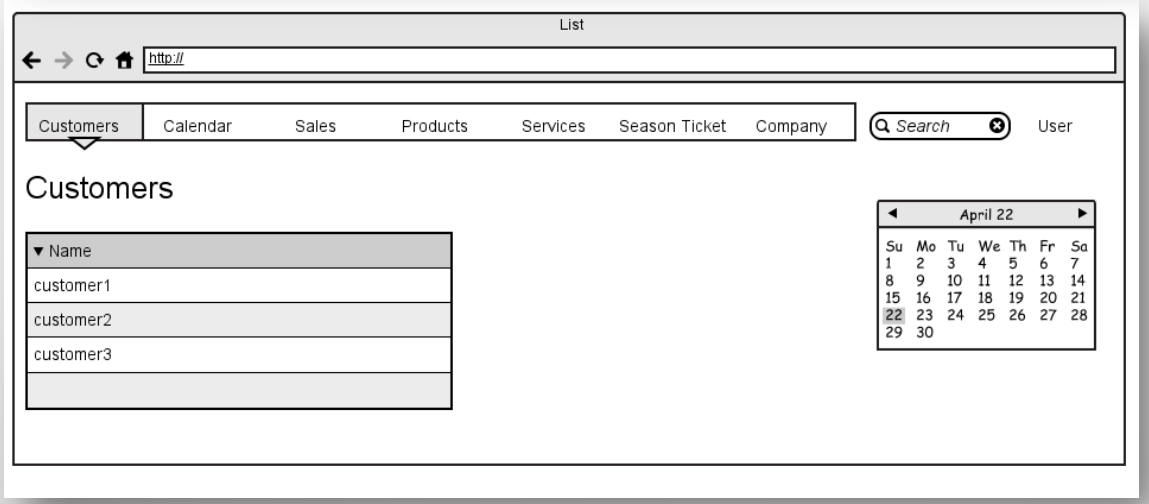


# Análisis funcional

## Prototipo interfaces de usuario




Edición de un cliente

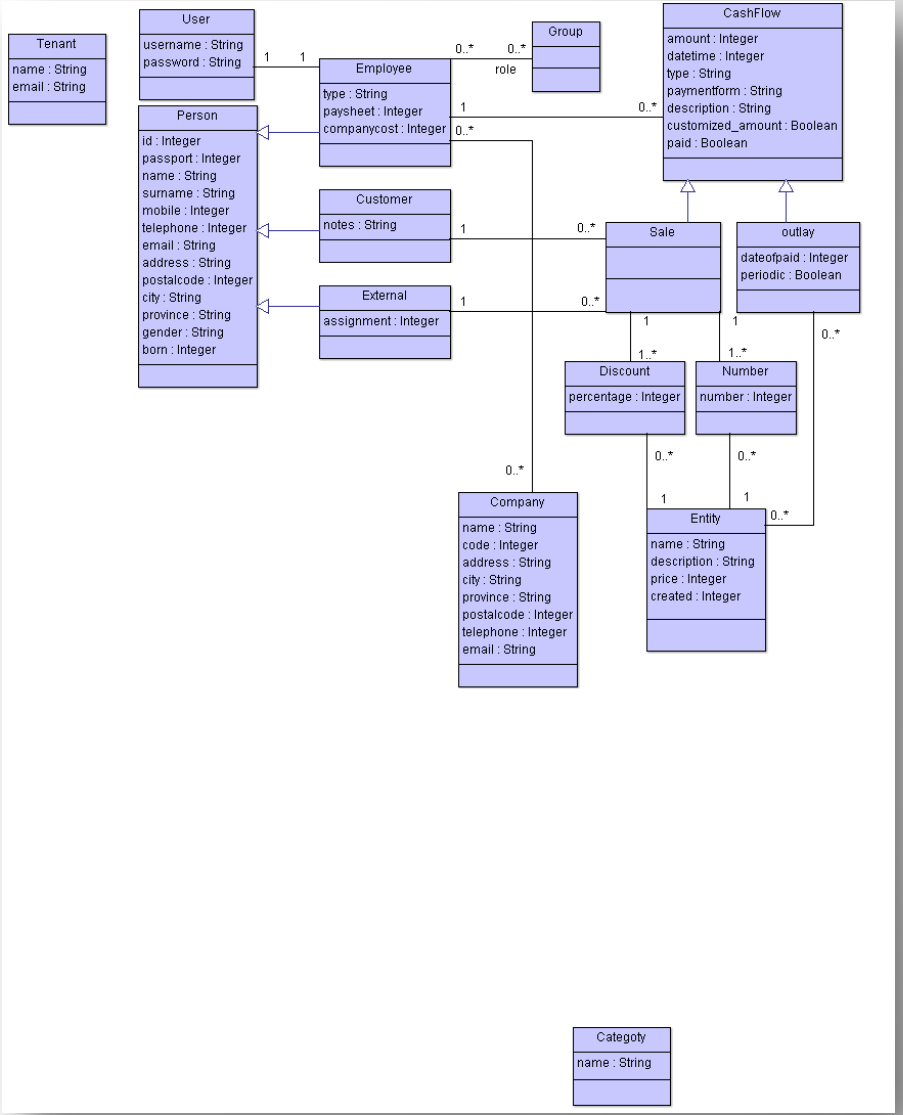



Listado de clientes

# Diseño técnico

Diseño modular:
**(Núcleo)**

- Tenant
- User
- Person
  - Employee
  - Customer
  - External
- Group
- Company
- CashFlow
  - Sale
  - Outlay
- Discount
- Number
- Entity

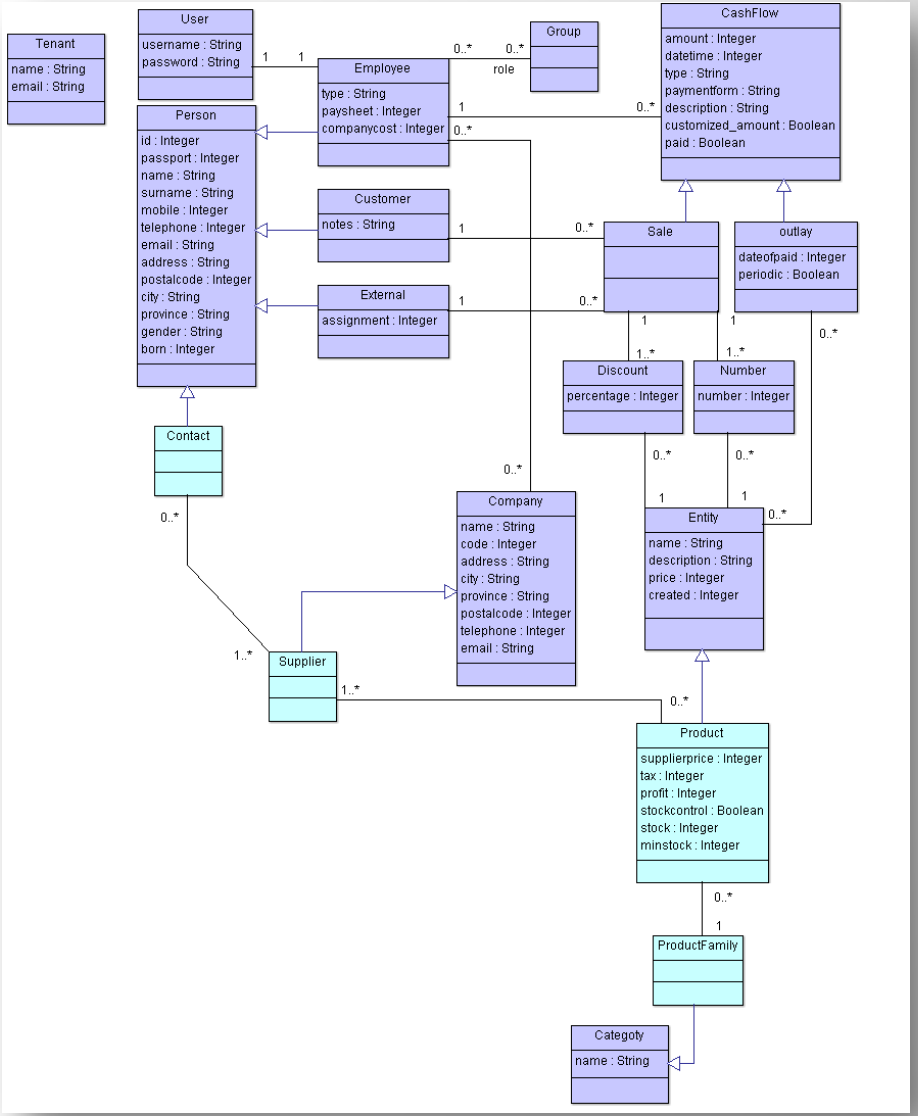


# Diseño técnico

Diseño modular:

**(Núcelo + Productos)**

- Product
- Supplier
- Contact
- ProductFamily

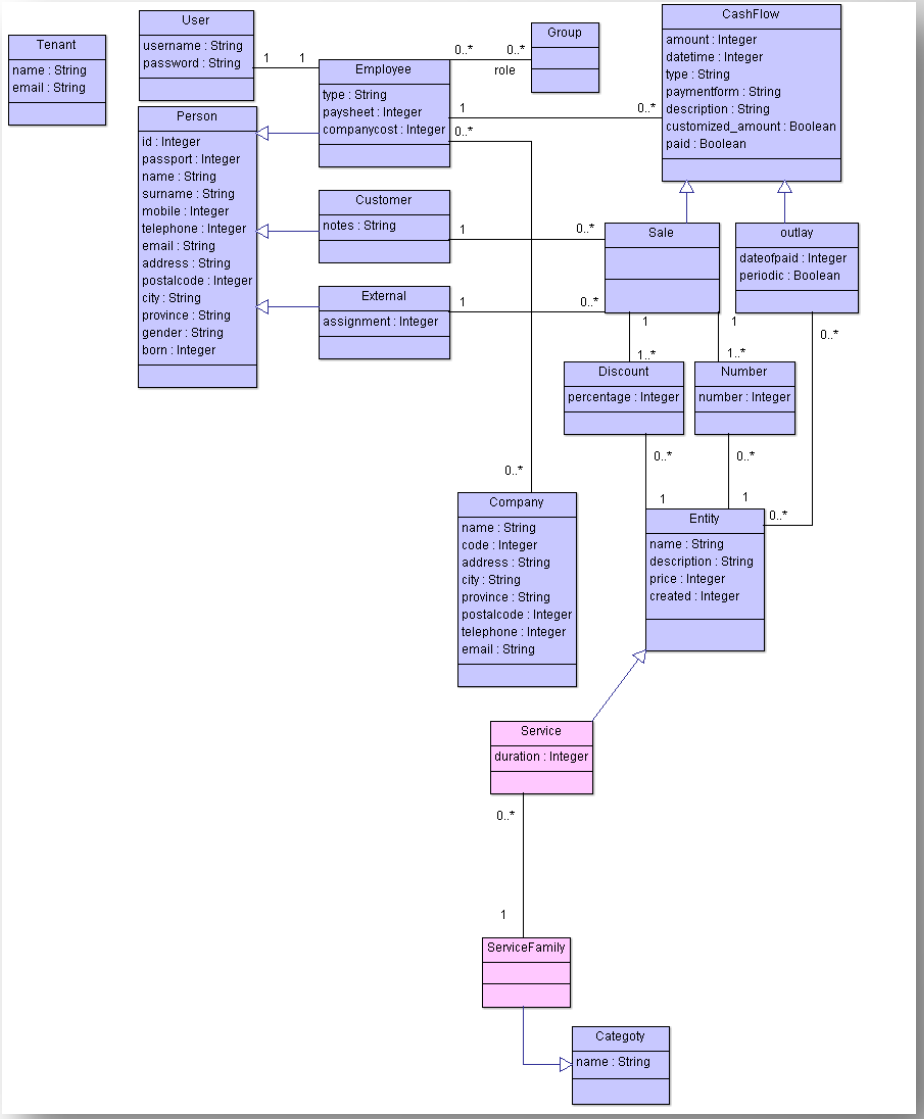


# Diseño técnico

Diseño modular:

**(Núcleo + Servicios)**

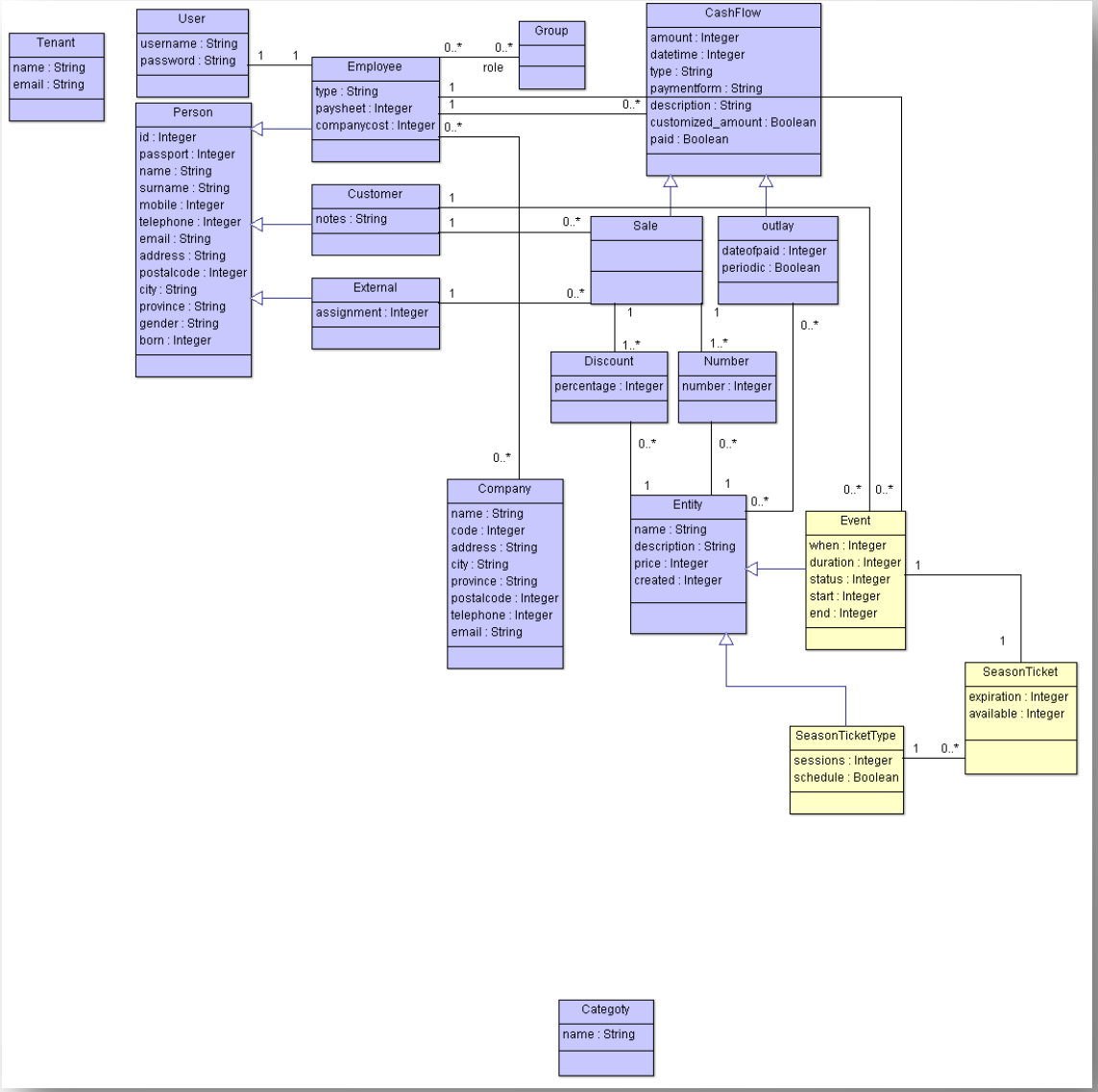
- Service
- ServiceFamily



# Diseño técnico

Diseño modular:
**(Núcleo + Citas)**

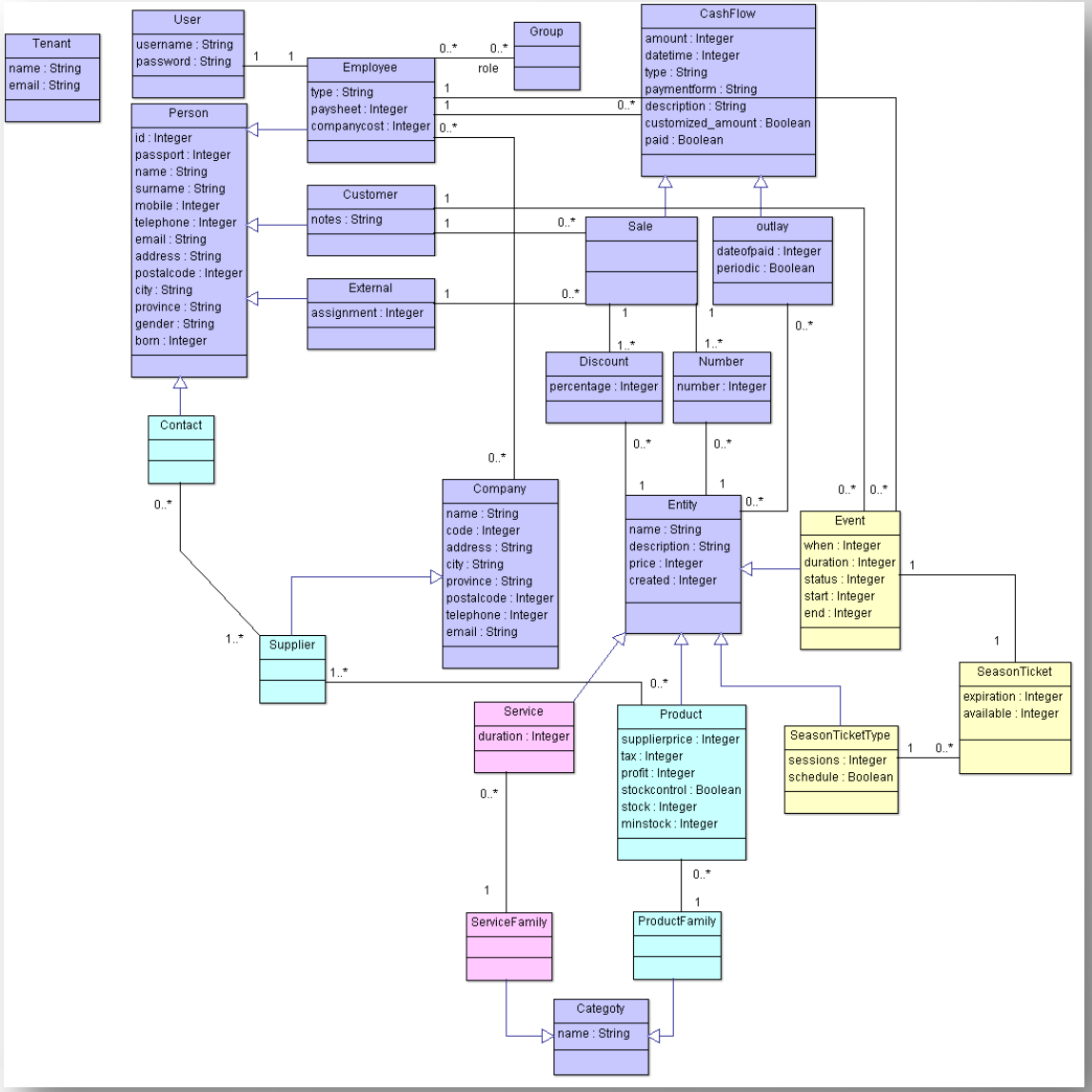
- Event
- SeasonTicket
- SeasonTicketType



# Diseño técnico

Diseño modular:

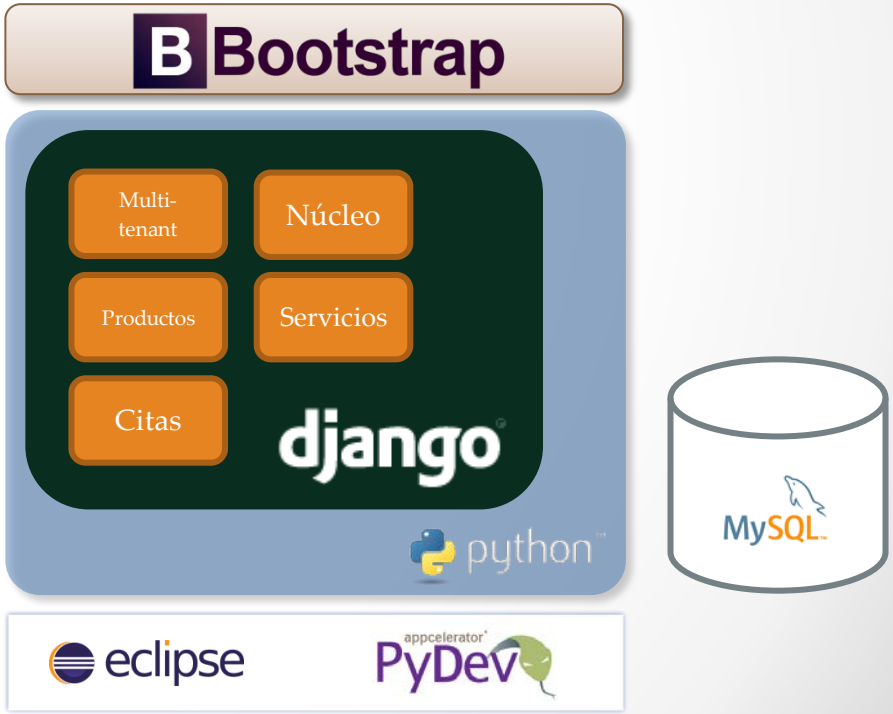
**Completo**



# Diseño técnico

Arquitectura

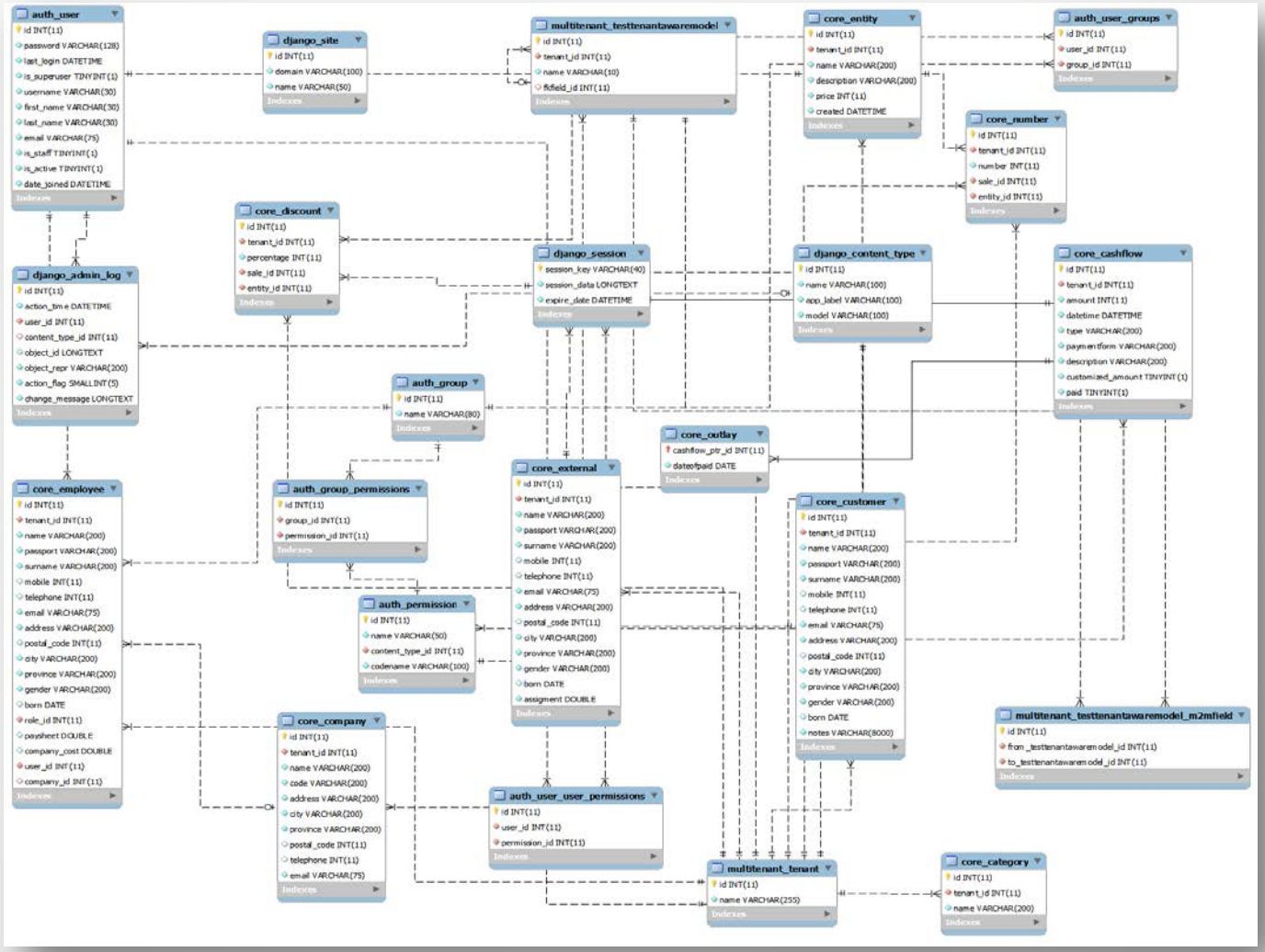
- **Django**. Framework *open source*, que sigue el paradigma MTV (*Model, Template and View*).
- **Python**. Lenguaje de programación interpretado
- **MySQL**. Base de datos relacional
- **Bootstrap**. Framework *front-end* para aplicaciones web
- **Eclipse+PyDev**. Entorno de desarrollo integrado




Arquitectura técnica

# Diseño técnico

## Modelo de base de datos



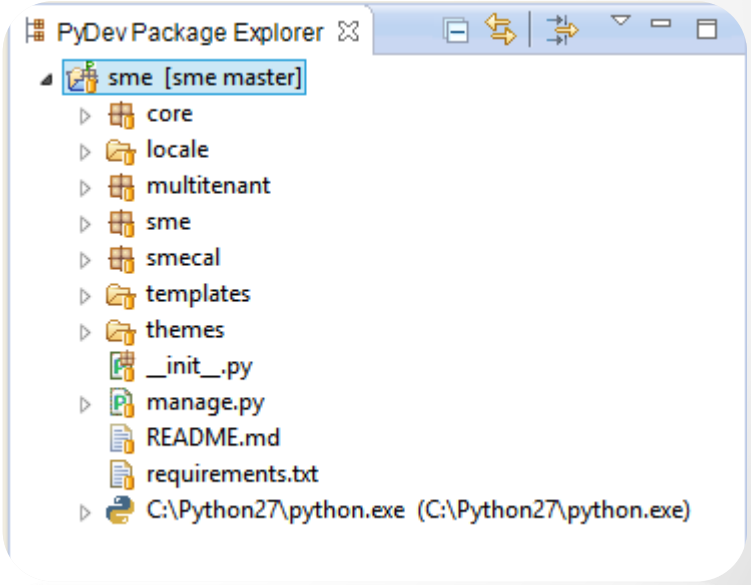
# Implementación

- Entorno de desarrollo
  - IDE: Eclipse + PyDev + Egit
  - Base de datos y servidor web: XAMPP versión 3.2.1 [Mayo de 2013]
    - MySQL 5.5.34 + phpmyadmin
    - Apache/2.4.7 (Win32) OpenSSL/0.9.8 y PHP/5.4.22
  - Intérprete de Python: 2.7.6
  - Librerías adicionales
    - Módulo de python: MySQLdb versión 1.2.4
    - Gnu gettext-utils
  - Gestión del código fuente: Git

# Implementación

## Estructura del código fuente

- **Core**. Es la aplicación donde se encuentra el código del núcleo del sistema.
- **Locale**. Es el lugar donde se encuentran los ficheros de traducción de la aplicación para la internacionalización de la misma
- **Multitenant**. Es la aplicación que gestiona el *multitenancy*
- **sme**. Es el proyecto en sí, que engloba a todas las aplicaciones y donde se encuentran los ficheros de configuración, administración y construcción de url's entre otros.
- **smecal**. Es la aplicación que implementa el módulo de citas.
- **Templates**. Es el lugar donde se almacenan las plantillas web de la aplicación.
- **Themes**. Es el lugar donde se almacenan los temas gráficos de la aplicación, incluidos los css, javascript e imágenes.
- **Manage.py** se trata del script para la gestión de django.
- En **requirements.txt** se encuentran los requisitos del sistema en cuanto a librerías externas, estos son: Django==1.6 y MySQL-python==1.2.3



# Implementación

## Configuración de la aplicación: **settings.py**

- Base de datos

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'test',
        'USER': os.environ.get('MYSQL_USER'),
        'PASSWORD': os.environ.get('MYSQL_PASSWORD'),
        'HOST': '127.0.0.1',
    }
}
```

- Idiomas

```
LANGUAGES = (
  ('es', _('Spanish')),
  ('en', _('English')),
)
```

- Aplicaciones instaladas (módulos)

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.humanize',
    'multitenant',
    'core',
    'smecal',
    'django.contrib.admin',
    'django.contrib.admindocs',
)
```

Módulo de usuarios

Otros módulos django

Módulo de multitenant

Núcleo de la aplicación

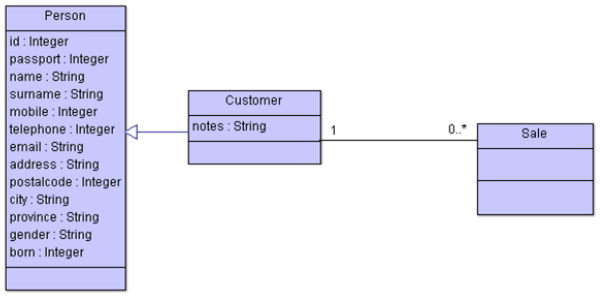
Módulo de citas

Módulo de administración

# Implementación

- Implementación del modelo

Extracto del modelo




```python
class Person(TenantModel):
    name = models.CharField(max_length=200, unique = True)
    passport = models.CharField(max_length=200, null=True, blank=True)
    surname = models.CharField(max_length=200, null=True, blank=True)
    mobile = models.IntegerField(blank=True, null=True)
    telephone = models.IntegerField(blank=True, null=True)
    email = models.EmailField(blank=True)
    address = models.CharField(max_length=200, null=True, blank=True)
    postal_code = models.IntegerField(blank=True, null=True)
    city = models.CharField(max_length=200, null=True, blank=True)
    province = models.CharField(max_length=200, null=True, blank=True)
    gender = models.CharField(max_length=200, null=True, blank=True, choices=GENDER)
    born = models.DateField(blank=True, null=True)

    class Meta:
        abstract = True
```

Hereda de

```python
class Customer(Person):
    notes = models.CharField(max_length=8000, null=True, blank=True)

    def __unicode__(self):
        return '%s %s' % (self.name, self.surname)
```

Asociación Cliente/venta

```python
class Sale(CashFlow):
    customer = models.ForeignKey(Customer)
    external = models.ForeignKey(External, blank=True, null=True)
    entity = models.ManyToManyField(Entity,null=True,blank=True)

    def __unicode__(self):
        return '%s' % (self.amount)
```

# Implementación

- Implementación de las vistas

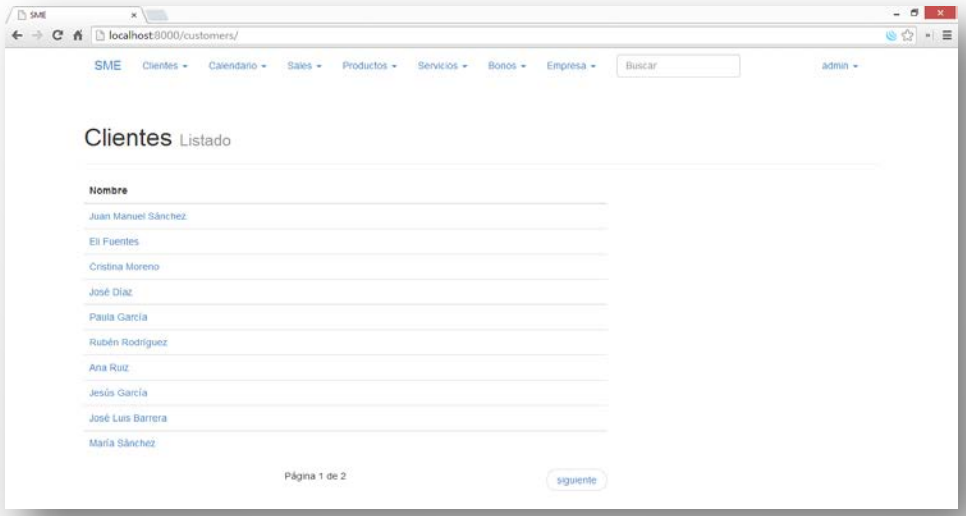
```python
class CustomerList(ListView):
    paginate_by = 10

    def get_queryset(self):
        return Customer.tenant_objects.all().order_by('id').reverse()
```

Vista

```
{% extends "core/person_list.html" %}
{% load i18n %}
{% block section %}
<div class="page-header">
   <h1>{% trans "Customers" %} <small>{% trans "List" %}</small></h1>
</div>
{% endblock %}
```

Tempate



# Implementación

- Implementación del multiidioma
  - En Templates: etiqueta “trans”

```
<h1>{% trans "Customers" %} <small>{% trans "List" %}</small></h1>
```

  - En código python: función “_”, abreviatura de ugettext_lazy

```
GENDER = (
        ('M', _("Male")),
        ('F', _("Female"))
    )
```

  - La utilidad de traducciones busca automáticamente en el código y genera el fichero de traducciones .po, únicamente tendremos que completar la cadena “msgstr” para cada idioma

```
#: .\core\models.py:73
msgid "Enter Description"
msgstr "Introduce descripción"
```

  - La aplicación detecta el idioma del usuario de forma automática y, si está soportado, muestra la aplicación en ese idioma. En caso contrario se muestra en el idioma por defecto: inglés.
    - Actualmente sólo están soportados EN y ES

# Implementación

- Implementación del multitenancy 1/2
  - Cada usuario del sistema (Employee) dispone de un **número de identificación de tenant (tenant_id)**. Ejemplo tabla users:

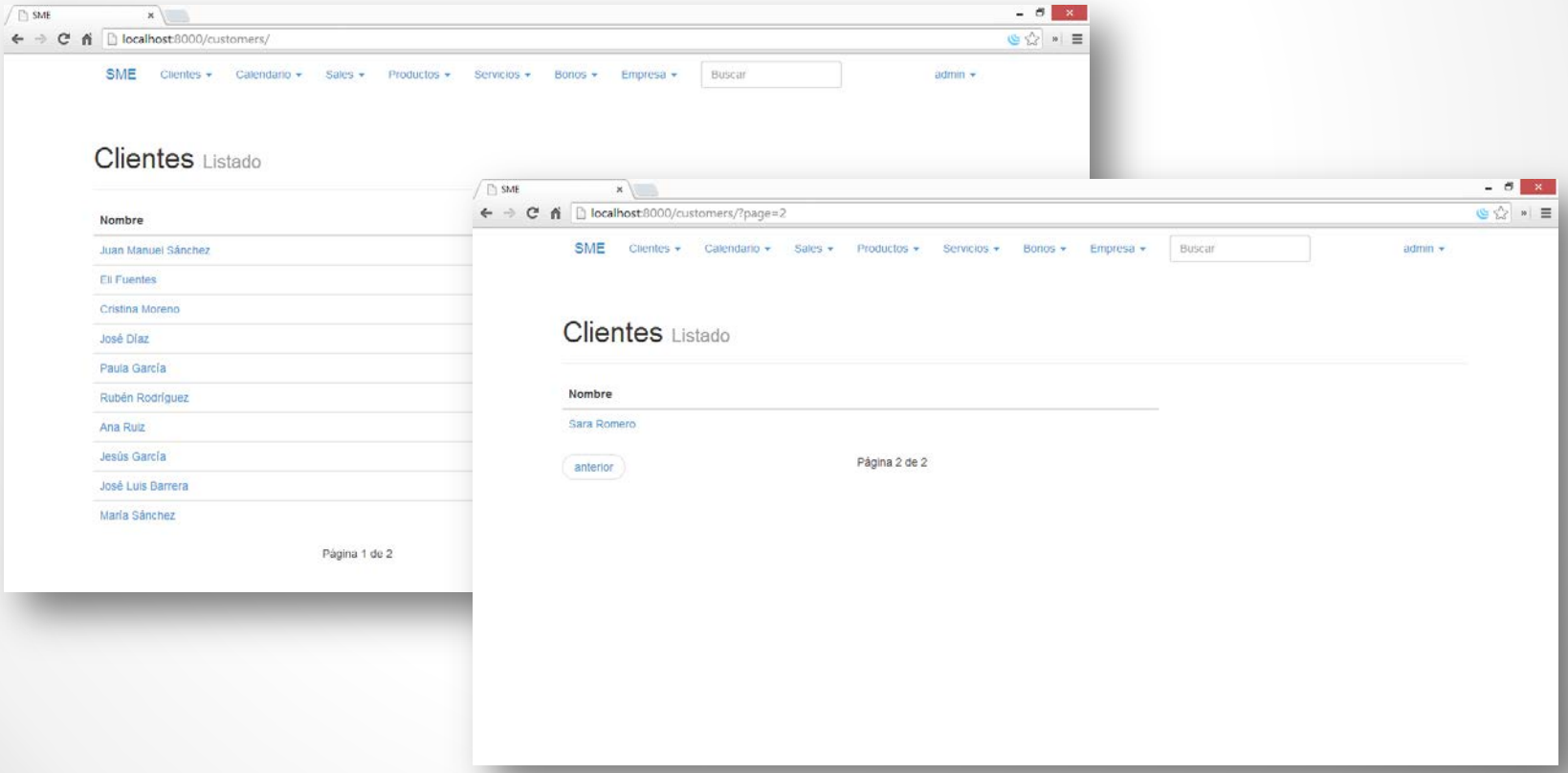
| | id | tenant_id | name |
|---|---|---|---|
| Editar Copiar Borrar | 1 | 2 | admin |
| Editar Copiar Borrar | 9 | 8 | employee1 |

  - Todas las entidades modeladas en el sistema están gestionadas por una meta-entidad que controla la propiedad de los contenidos mediante el identificador **tenant_id**. Ejemplo tabla Customer:

| | id | tenant_id | name | passport | surname |
|---|---|---|---|---|---|
| Editar Copiar Borrar | 1 | 2 | Sara | | Romero |
| Editar Copiar Borrar | 2 | 2 | María | NULL | Sánchez |
| Editar Copiar Borrar | 3 | 8 | Luis | NULL | Romero |
| Editar Copiar Borrar | 4 | 2 | José Luis | NULL | Barrera |
| Editar Copiar Borrar | 5 | 2 | Jesús | NULL | García |
| Editar Copiar Borrar | 6 | 2 | Ana | NULL | Ruiz |
| Editar Copiar Borrar | 7 | 2 | Rubén | NULL | Rodríguez |
| Editar Copiar Borrar | 8 | 2 | Paula | NULL | García |
| Editar Copiar Borrar | 9 | 2 | José | NULL | Díaz |
| Editar Copiar Borrar | 10 | 2 | Cristina | NULL | Moreno |
| Editar Copiar Borrar | 11 | 2 | Eli | NULL | Fuentes |
| Editar Copiar Borrar | 12 | 2 | Juan Manuel | NULL | Sánchez |
| Editar Copiar Borrar | 13 | 8 | Pablo | NULL | Vergara |
| Editar Copiar Borrar | 14 | 8 | Marta | NULL | Salado |

# Implementación

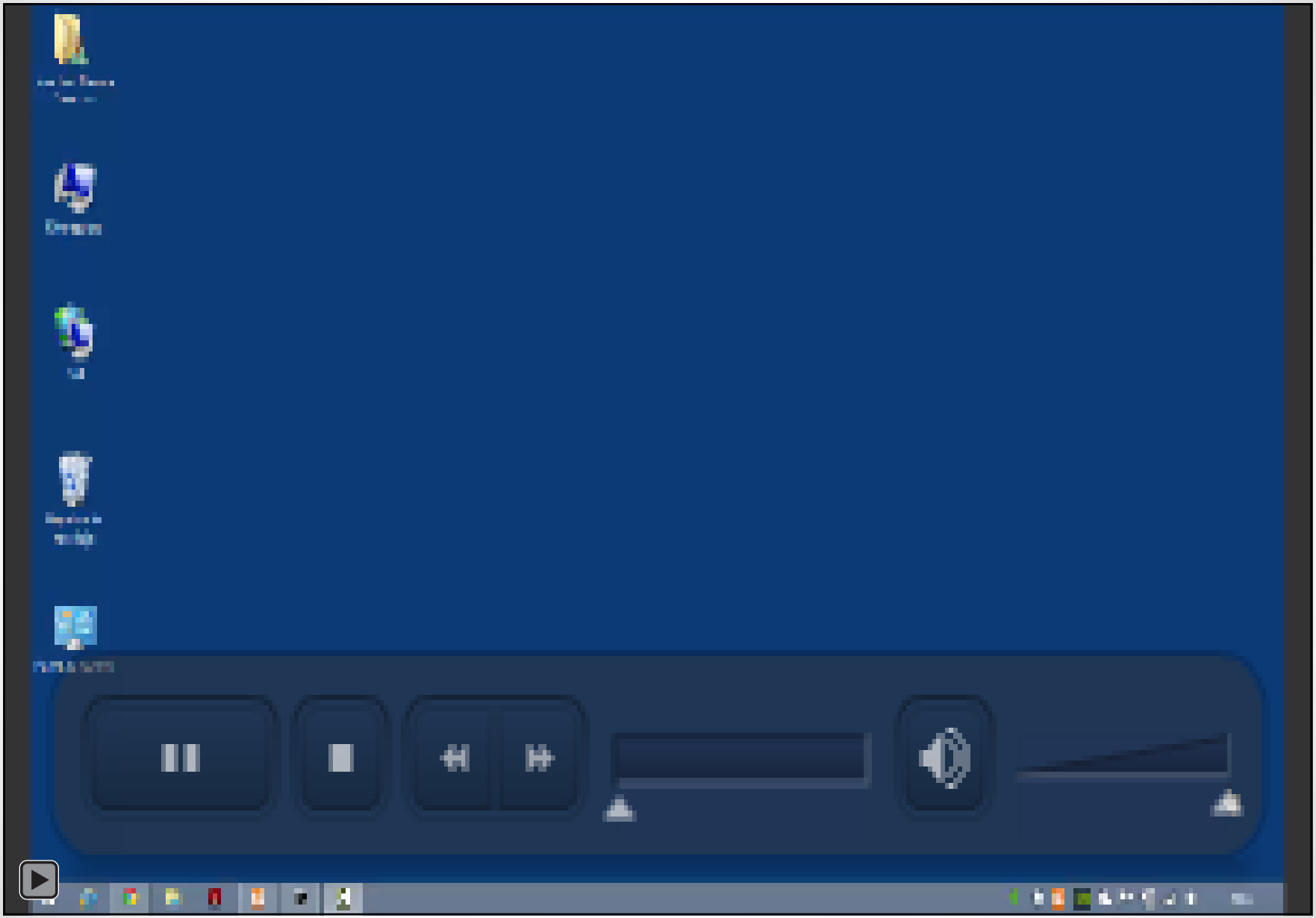
- Implementación del multitenancy 2/2
  - Este identificador es utilizado posteriormente para que cada usuario trabaje únicamente con aquellos contenidos que le pertenecen.
  - Como podemos ver, en la vista del listado de clientes (tabla Customer) para el usuario admin, sólo aparecen los clientes con tenant_id = 8



# Conclusiones

- Desde el punto de vista personal
  - Reto tecnológico: utilización de tecnologías relativamente nuevas para mí.
  - El TFC me ha permitido:
    - Poner en práctica los conocimiento adquiridos de ingeniería del software
    - Diseñar desde cero un sistema modular y extensible
    - Aprender nuevas tecnologías como ha sido django
    - Utilizar nuevos sistemas de gestión del código fuente actualmente en auge como git
    - Utilizar las últimas tecnologías en frameworks de UI como bootstrap
    - Profundizar en nuevas arquitecturas de aplicaciones web orientadas a modelos de explotación del software como servicio
    - Descubrir el paradigma de arquitecturas MTV no utilizadas anteriormente
- Desde el punto de vista del proyecto
  - Se ha conseguido diseñar un sistema flexible y modular
  - El sistema sienta las bases para ser extendido en cuanto a funcionalidades
  - Favorece la reutilización del código
  - El diseño multitenancy que se ha implantado permite ofrecer el software como servicio *out of the box*, sin añadir complejidad extra para el desarrollador
  - La intención es seguir evolucionando el software
  - El código fuente ha sido publicado en github, disponible para que cualquier persona interesada: https://github.com/jlbarrera/sme

# Demo



# Gracias