GRADO EN INGENIERIA INFORMATICA

INGENIERIA WEB

PROYECTO WEB

BADUCO

Página web dedicada a estimular la interacción y la generación y el intercambio de información entre usuarios

Autores:

Emilio López Piña
Manuel Gallego Cerrillo
Juan Carlos Horcas Cerrillo
Miguel Ángel Víctor Segado

Índice de Contenido

Índice de Tablas

Índice de Ilustraciones

1. INTRODUCCIÓN

1.1 ¿Qué podemos hacer con esta aplicación?

A veces nos gustaría hablar con personas que tenemos a nuestro alrededor, pero no lo hacemos. Esto puedo deberse a varios motivos: bien porque no nos fiamos de hablar con personas desconocidas, porque pensemos que el otro no tiene interés en hablar con nosotros, o simplemente por vergüenza.

Por ello, hemos creado una red social que ayude a superar estas barreras y permita interactuar con personas que tengamos cerca, a nuestro alrededor.

Cuando estemos interesados en hablar con una persona que tenemos cerca, podemos visitar su perfil para decidir si queremos interactuar con ella o no. Si decidimos interactuar con ella pero no queremos hacerlo en persona, podemos enviarle un comentario e iniciar una conversación.

2. CICLO DE VIDA DEL PROYECTO

Para construir el Proyecto Baduco, nuestro equipo se ha centrado principalmente en las siguientes etapas del ciclo de vida del software: Análisis, Diseño, Implementación, Pruebas y Documentación. El uso del ciclo de vida del software a la hora de diseñar un proyecto web es tan útil como fiable puesto que en él, se describen los diferentes pasos que se deben seguir para el desarrollo de un software, partiendo desde una necesidad hasta llegar a la puesta en marcha de una solución y su apropiado mantenimiento. El ciclo de vida para un software comienza cuando se tiene la necesidad de resolver un problema, y termina cuando el programa que se desarrolló para cumplir con los requerimientos, deja de ser utilizado.

Analizamos brevemente las fases del software, a lo largo de la documentación se explicaran con más detalle.

-Análisis: En esta etapa se debe entender y comprender de forma detallada cual es la problemática a resolver, verificando el entorno en el cual se encuentra dicho problema, de tal manera que se obtenga la información necesaria y suficiente para afrontar su respectiva solución. Esta etapa es conocida como la del QUÉ se va a solucionar.

- Diseño: Una vez que se tiene la suficiente información del problema a solucionar, es importante determinar la estrategia que se va a utilizar para resolver el problema. Esta etapa es conocida bajo el CÓMO se va a solucionar.

- Implementación: partiendo del análisis y diseño de la solución, en esta etapa se procede a desarrollar el correspondiente programa que solucione el problema mediante el uso de una herramienta computacional determinada.

3. ANÁLISIS

En la primera etapa de la vida de nuestro proyecto, nos centramos en el estudio y compresión del problema que se quería solucionar. Para simplificar esta compresión se analiza el enunciado del problema y se obtienen los requisitos, tanto principales como secundarios. El análisis de requisitos permite especificar la función y el rendimiento de la aplicación, indica la interacción de la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software.

Los requisitos principales que hemos obtenidos son los siguientes: La página ofrecerá diferentes servicios a sus usuarios, crear un perfil con información personal, permitir crear comentarios sobre diversos temas, visitar perfiles de otras personas e interactuar con ellos. En las siguientes etapas de vida del proyecto se mostrara mas detalle sobre estos requisitos principales y secundarios.

3.1 Modelado de requisitos

Este tipo de modelado nos ha permitido identificar los usuarios (perfiles) así como las necesidades de información, navegación, adaptación, presentación y seguridad. Para comprender mejor su funcionamiento los diagramas de caso de uso son los idóneos.

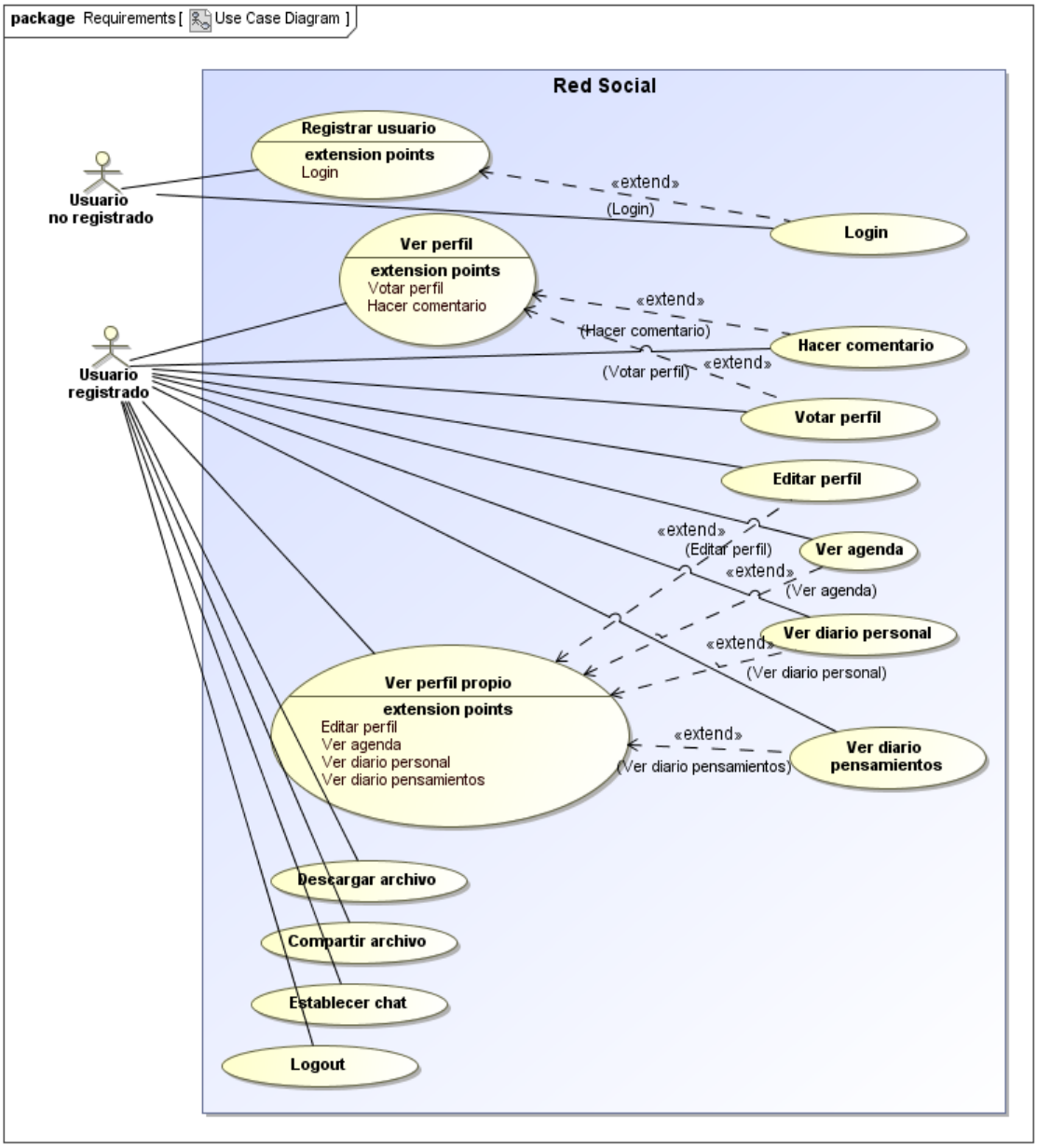

Ilustración 1. Diagrama de Casos de Uso

A partir del diagrama de casos de uso generado, podemos obtener mucha información. En la aplicación existirán dos tipos de usuario, registrados y no registrados, y estos podrán realizar diversas actividades dentro de la página web. Como se puede observar un usuario registrado puede acceder a todas las funcionalidades de la

aplicación web, mientras que el no registrado se encuentra más limitado debido a las exigencias propuestas por el enunciado.

Las actividades que puede realizar un usuario no registrado en la aplicación son las siguientes:

- Registrar Usuario
- Login
- Ver Perfiles Públicos

Las actividades que puede realizar un usuario registrado son las siguientes:

- Ver Perfiles (Públicos y Privados)
- Ver Perfil Propio
- Editar Perfil Propio
- Editar Estado Actual
- Ver Comentarios Realizados
- Ver Comentarios
- Ver Agenda:
- Ver Diario Personal
- Descargar Archivo
- Logout
- Establecer Chat
- Compartir Archivo
- Ver Diario de Pensamientos
- Ver Comentarios
- Buscar Perfil
- Hacer Comentario
- Votar

4. DISEÑO

Una vez entendida la problemática a resolver a través de los requisitos, hemos seleccionado una estrategia para la resolución del problema. La herramienta que hemos usado para modelar nuestra aplicación web ha sido UWE que es una herramienta para modelar aplicaciones web, utilizada en la ingeniería web, prestando especial atención en sistematización y personalización (sistemas adaptativos) además de que cubre todo el ciclo de vida del software.

Se han hecho uso de modelos los cuales han ayudado a simplificar el problema y dotar de una estructura firme y simple al proyecto.

El modelo que propone UWE está compuesto por 6 etapas o sub-modelos:

1. Modelo de Casos de Uso: modelo para capturar los requisitos del sistema.

2. Modelo de Contenido: es un modelo conceptual para el desarrollo del contenido.

3. Modelo de Usuario: es modelo de navegación, en el cual se incluyen modelos estáticos y modelos dinámicos.

4. Modelo de estructura: en el cual se encuentra la presentación del sistema y el modelo de flujo.

5. Modelo Abstracto: incluye el modelo a de interfaz de usuario y el modelo de ciclo de vida del objeto.

6. Modelo de Adaptación.

4.1 Modelado del Contenido

El contenido es la parte más importante de cualquier página web. Por ello debemos analizar y comprender que necesita nuestra aplicación web.

4.1.1 ¿Qué queremos incluir en la aplicación?

En primer lugar, el usuario debe registrarse. Para ello deberá rellenar los siguientes campos:

- Nombre y apellidos
- Correo electrónico
- Contraseña
- Fecha de nacimiento
- Hombre o mujer
- Aceptación de las condiciones de uso de la aplicación. Deben aceptar que son mayores de 18 años.
- Dirección particular (guardar también la geolocalización)
- Una vez registrado, la aplicación llevará al usuario a rellenar el perfil obligatoriamente.

4.1.2 Perfil

- ¿Qué es el perfil? Es la información sobre nosotros que queremos que las demás personas vean al hacer "click" sobre nuestro icono.
- ¿Qué información incluye el perfil? De la información que puede incluir el perfil, alguna deberá ser obligatoria y otra opcional (aquella información que se considere lesiva para el derecho a la privacidad individual):

    a) Nombre o pseudónimo
    b) Edad
    c) Hombre o mujer
    d) Orientación sexual
    e) Profesión
    f) Estudios
    g) Relaciones: con pareja, sin pareja, abierto a conocer nuevas personas, etc. (esto hay que concretarlo)
    h) Creencias religiosas: ateo, agnóstico, cristiano, musulmán, judío …

i) Intereses o aficiones
j) Otra información de interés a completar por el usuario

- Daremos la opción de crear dos perfiles diferentes, uno supuestamente verdadero y otro supuestamente falso. Los sujetos podrán activar el perfil que deseen en cada momento.

- El perfil incluirá un apartado en el que quede reflejada la media de las puntuaciones en cada una de las características a votar.

4.1.3 Comentarios

En tiempo real (deseos, intereses, necesidades) disponibles solamente para aquellas personas que tengamos cerca.

- Públicos: "Me gustaría hablar de política con alguna persona". Los comentarios públicos dejarán de ser visibles cuando lo deseemos. Cuando una persona haya hecho un comentario público, saldrá de su icono un bocadillo de color azul que podrán ver todas las personas a su alrededor. Los comentarios públicos solo podremos verlos cuando estemos cerca del emisor. Cuando nos alejemos el comentario no se quedará guardado en nuestro dispositivo.

- Privados: "Me pareces muy atractiva. ¿Estarías dispuesta a conocerme?" El emisor del mensaje debe recibir un indicativo de que el destinatario lo ha leído. Cuando una persona haya hecho un comentario privado, saldrá de su icono un bocadillo de color marrón que solo podremos ver nosotros. Estos comentarios sí se quedarán guardados en nuestro dispositivo, aunque ya no estemos cerca del emisor.

- Los comentarios públicos y privados podrán hacerse de forma anónima cuando se desee.

- Podremos incluir un botón de "frases comunes", en el que al clicar se desplieguen varias frases (por concretar). Además, daremos la opción de que puedan rellenar sus propias frases comunes.

- Se desea almacenar un histórico para estas actividades.

4.1.4 Diario de pensamientos

El usuario podrá registrar, si lo desea, aquellos pensamientos que tenga respecto a cualquier ámbito de su vida. Lo que escriba en este apartado quedará registrado y podrá ser consultado por el usuario en cualquier momento. En este apartado también debe ser almacenada la información de la localización, hora y fecha en el momento que se genera la reflexión.

4.1.5 Diario personal

Además del diario sobre ideas y pensamientos la aplicación permitirá disponer de un diario personal usando algún tipo de plantilla estándar.

4.1.6 Agenda

La aplicación dispondrá de una utilidad que permita la planificación de las actividades personales.

4.1.7 Votaciones

La opción de votar y ser votado será libre. Para poder votar, debe permitirse ser votado.

Se podrá votar, a los usuarios que así lo permitan, en las siguientes características:
a) Grado de atracción
b) Simpatía
c) Otras.

4.1.8 Intercambio de Ficheros

La aplicación dispondrá de una zona para el intercambio de ficheros entre los usuarios, se quiere tener un histórico para esta actividad.

Mediante un diagrama de clases recogemos toda esta información:

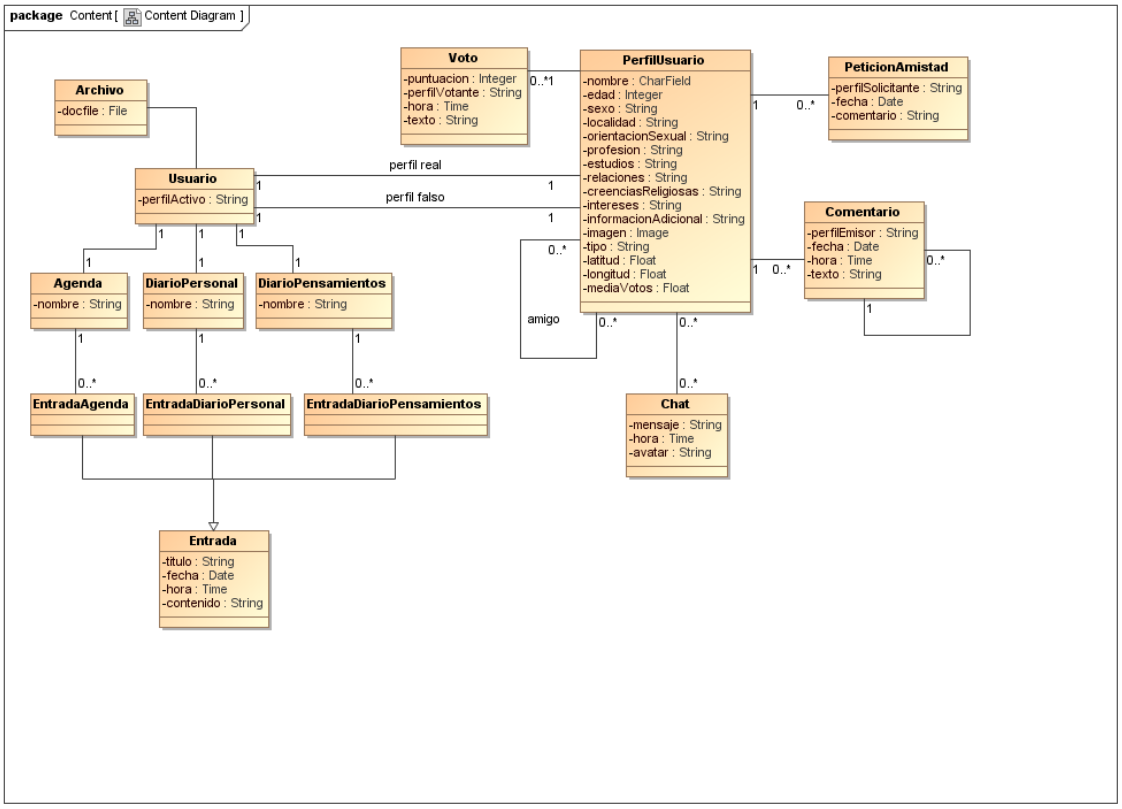

Ilustración 2. Diagrama de Clases

4.2 Modelado de la Navegación

El modelo de navegación sirve para modelar los caminos disponibles para cada usuario, incluye las clases de navegación (nodos navegables) y enlaces de navegación (enlaces directos entre las clases). Se basa en el modelo de contenido. A continuación se muestra el modelo de navegación del proyecto completo:

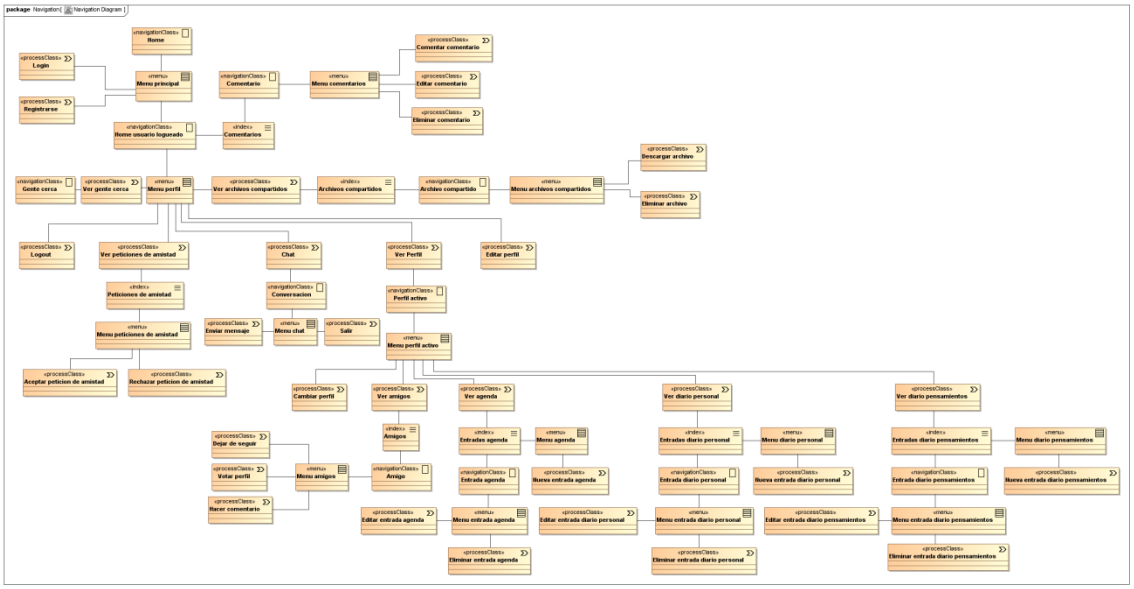

Ilustración 3. Modelo de Navegación

4.3 Modelo de presentación

En este modelo se presentan las clases de navegación y procesos que pertenecen a cada página web. A continuación se muestran todos los diagramas necesarios:

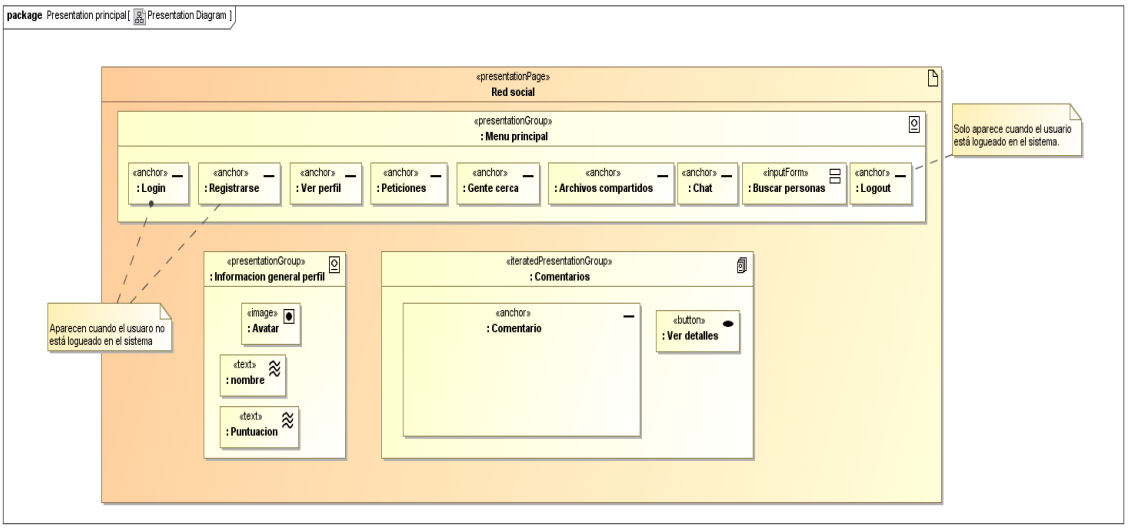

Ilustración 4. Diagrama presentación principal

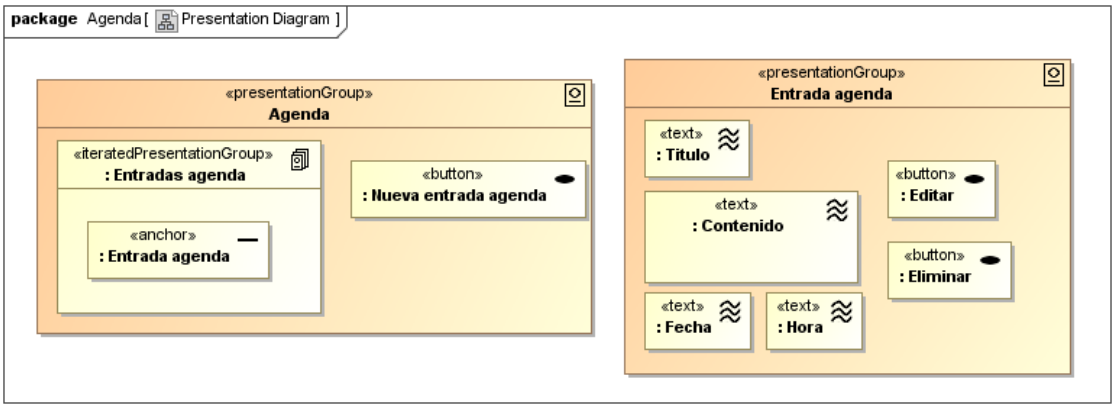

Ilustración 5. Diagrama presentación Agenda

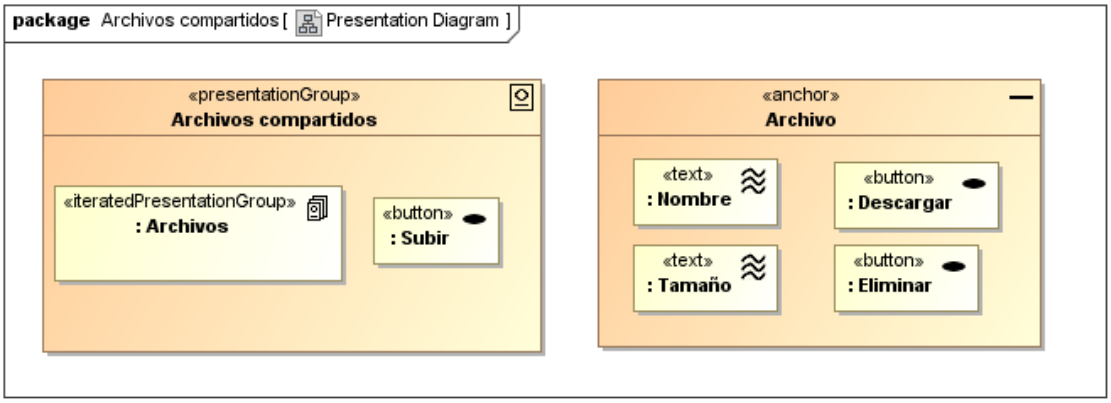

Ilustración 6. Diagrama presentación Archivos compartidos

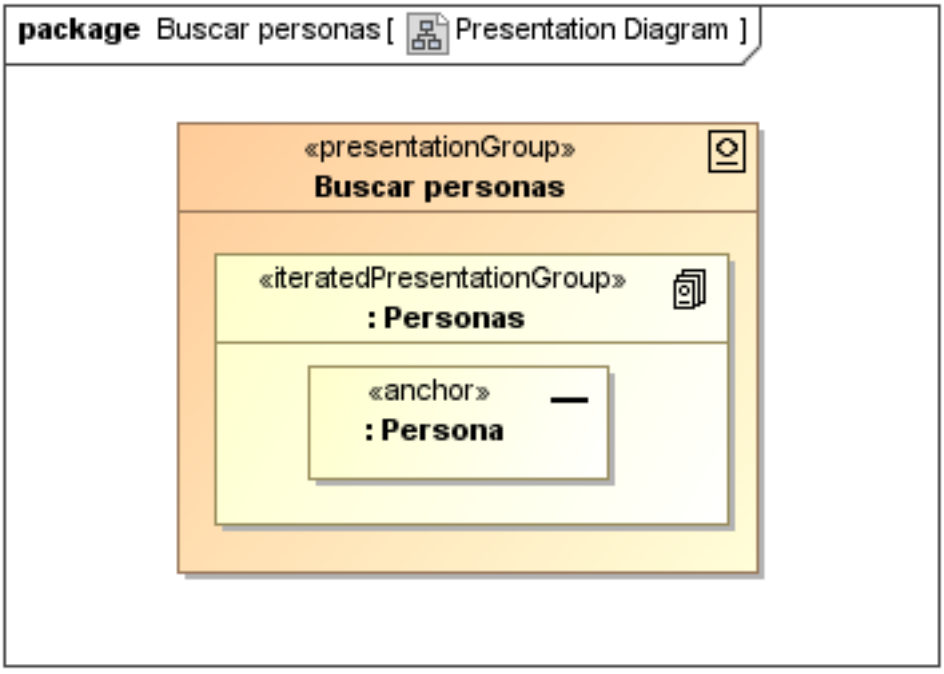

Ilustración 7. Diagrama presentación Buscar personas

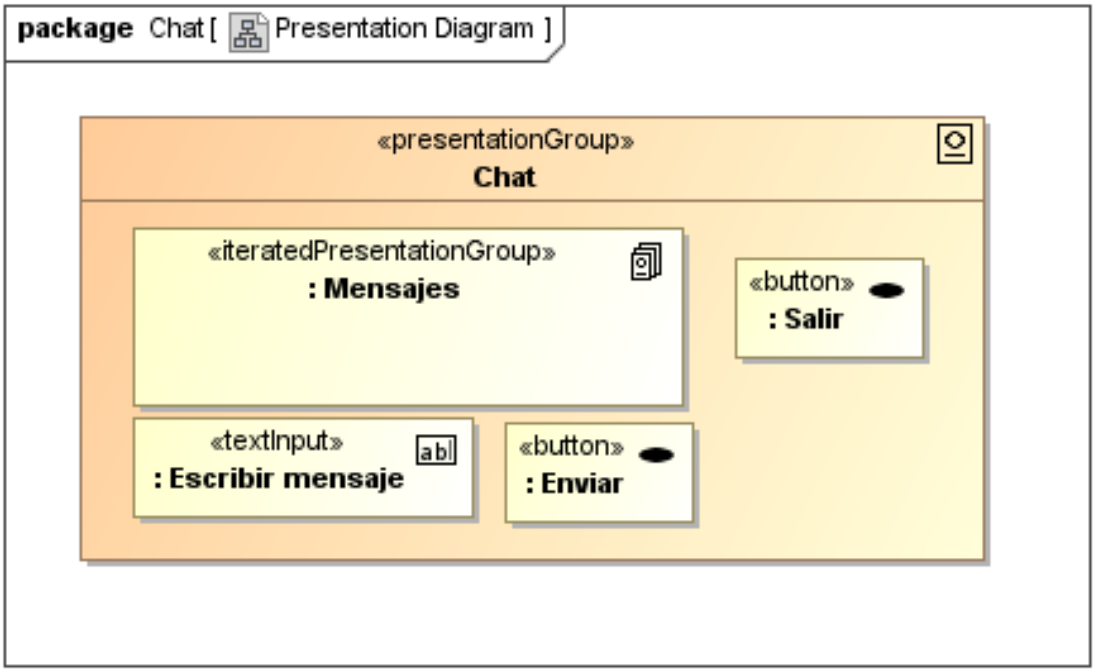

Ilustración 8. Diagrama de presentación Chat

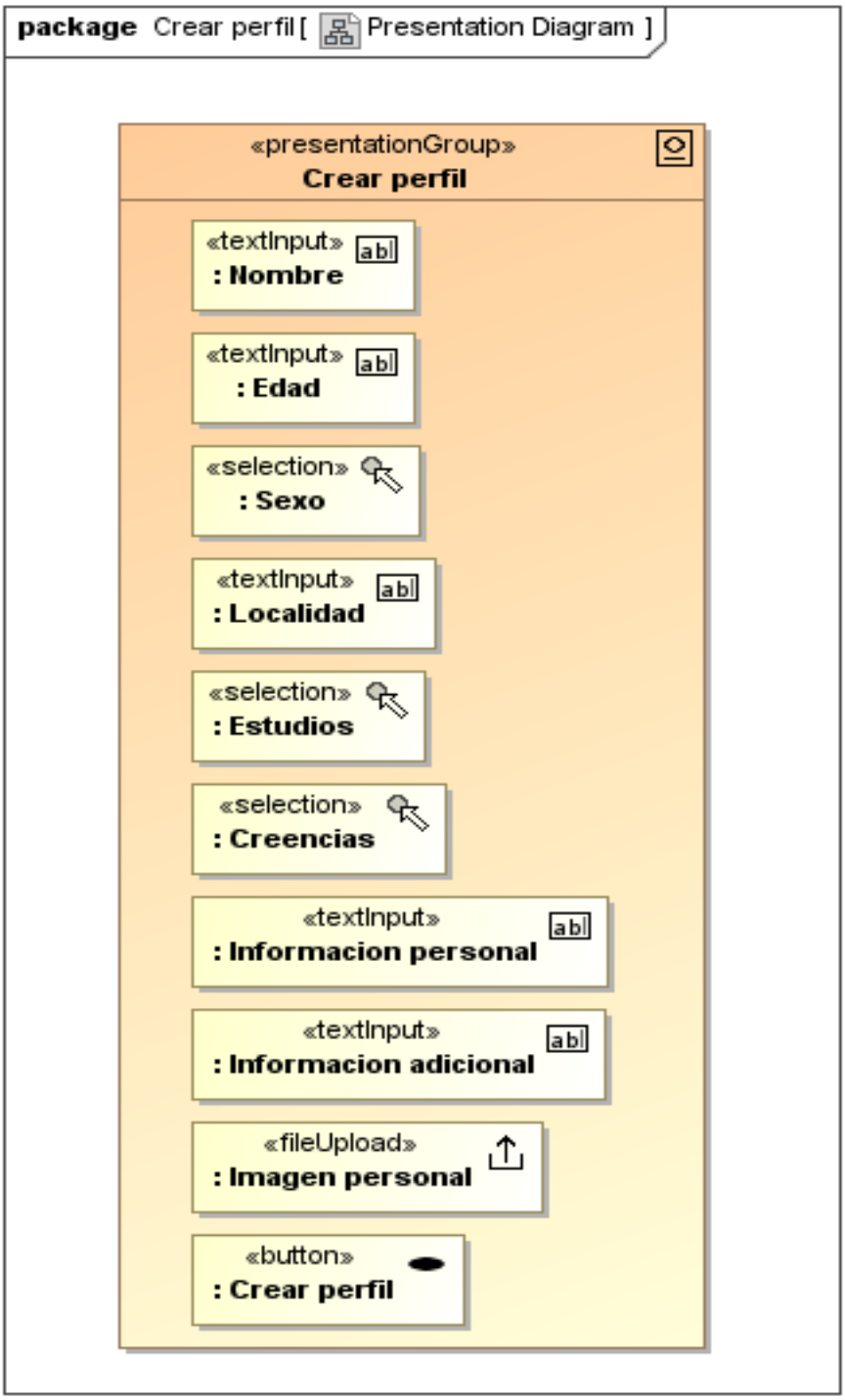

Ilustración 9. Diagrama presentación Crear perfil

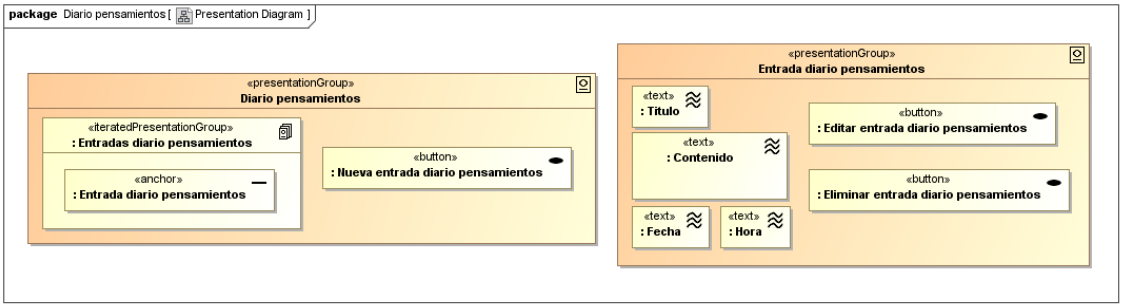

Ilustración 10. Diagrama presentación Diario personal

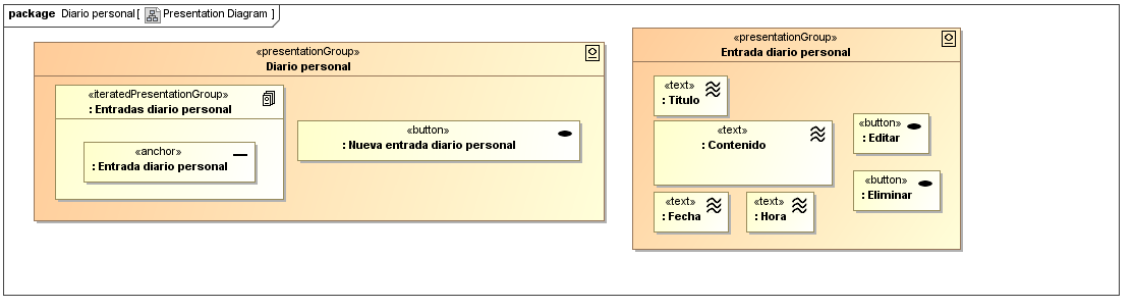

Ilustración 11. Diagrama presentación Diario personal

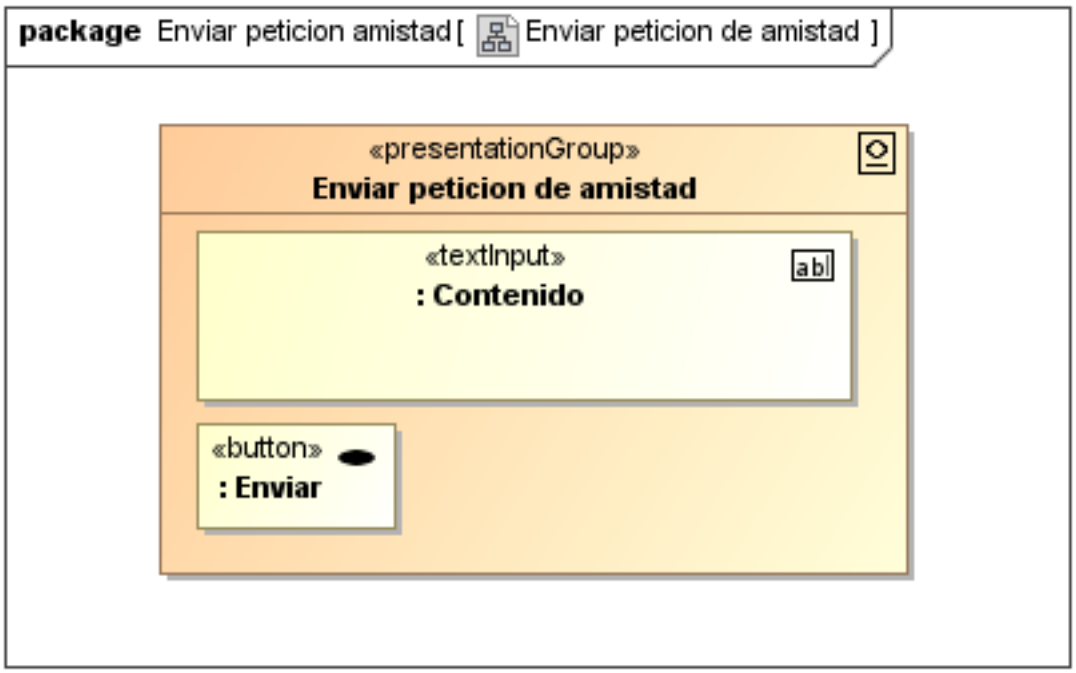

Ilustración 12. Diagrama presentación Enviar petición amistad

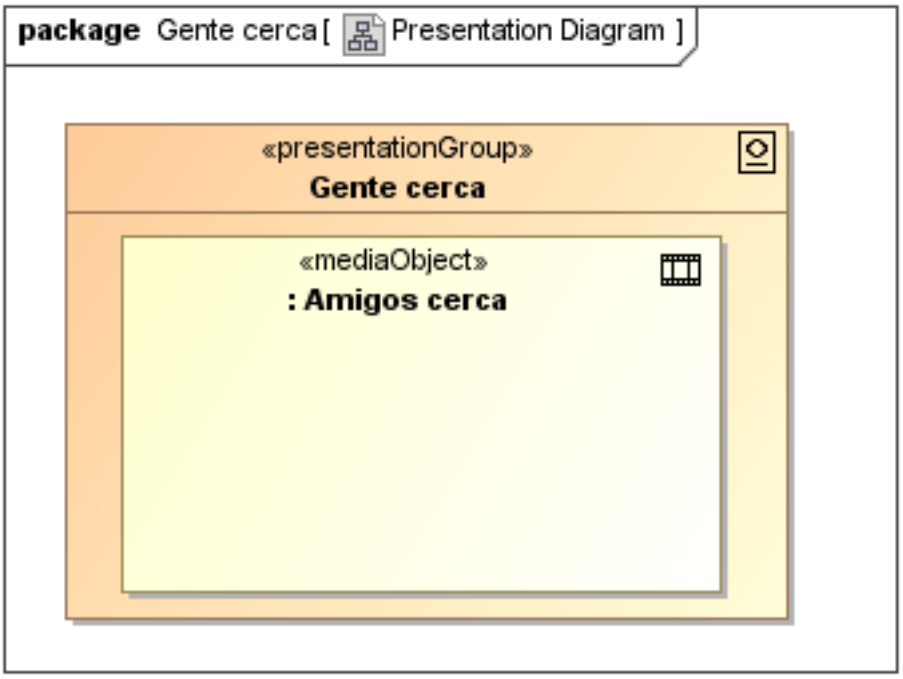

Ilustración 13. Diagrama presentación Gente cerca

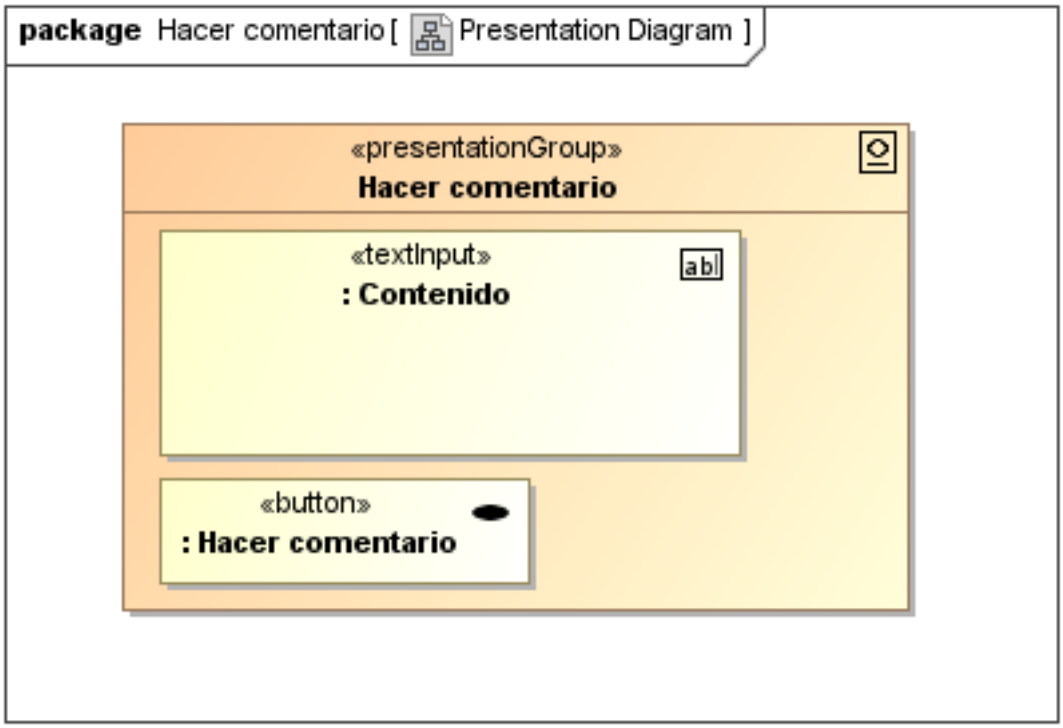

Ilustración 14. Diagrama presentación Hacer comentario.

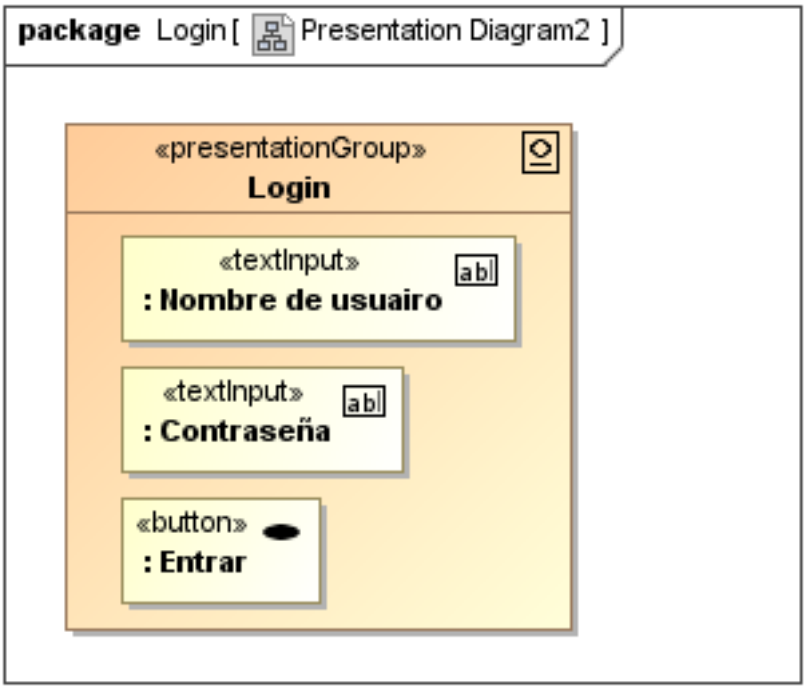

Ilustración 15. Diagrama presentación Login

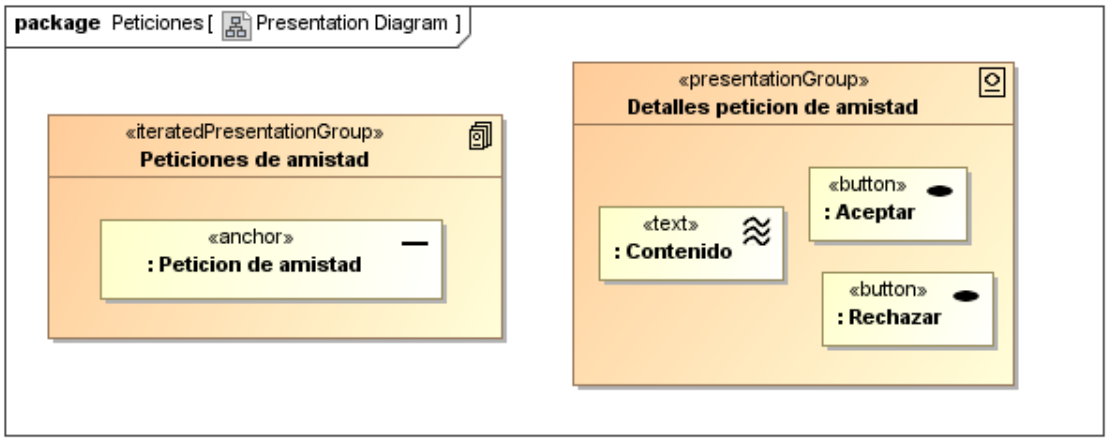

Ilustración 16. Diagrama presentación Peticiones

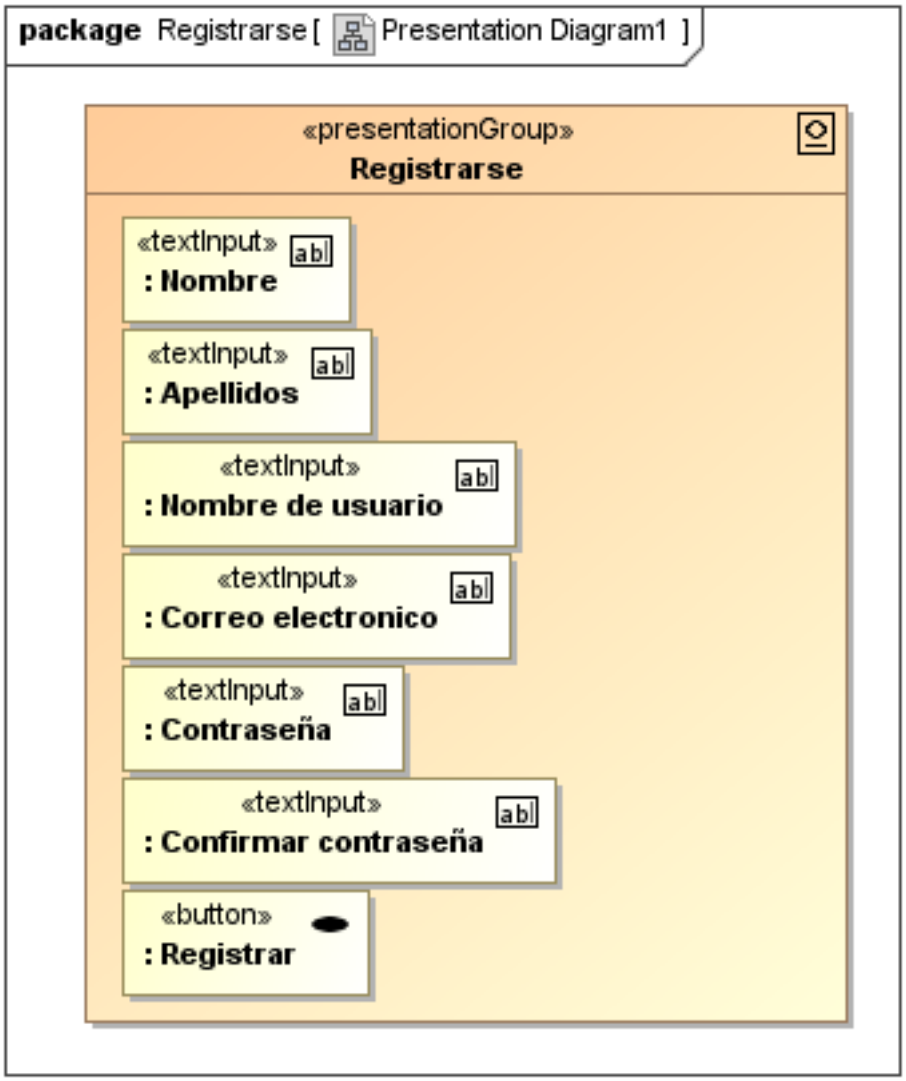

Ilustración 17. Diagrama presentación Registrarse

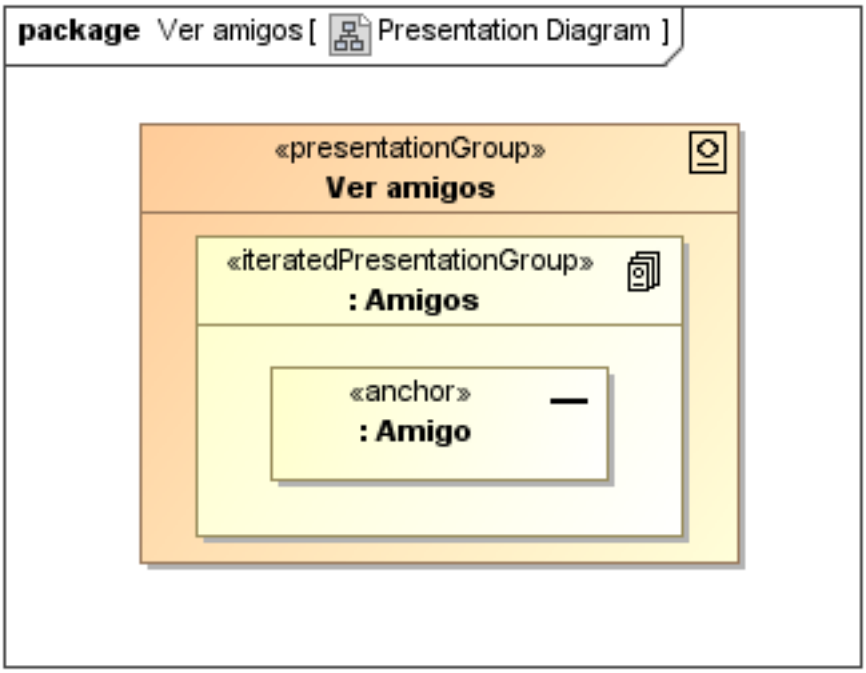

Ilustración 18. Diagrama presentación Ver amigos

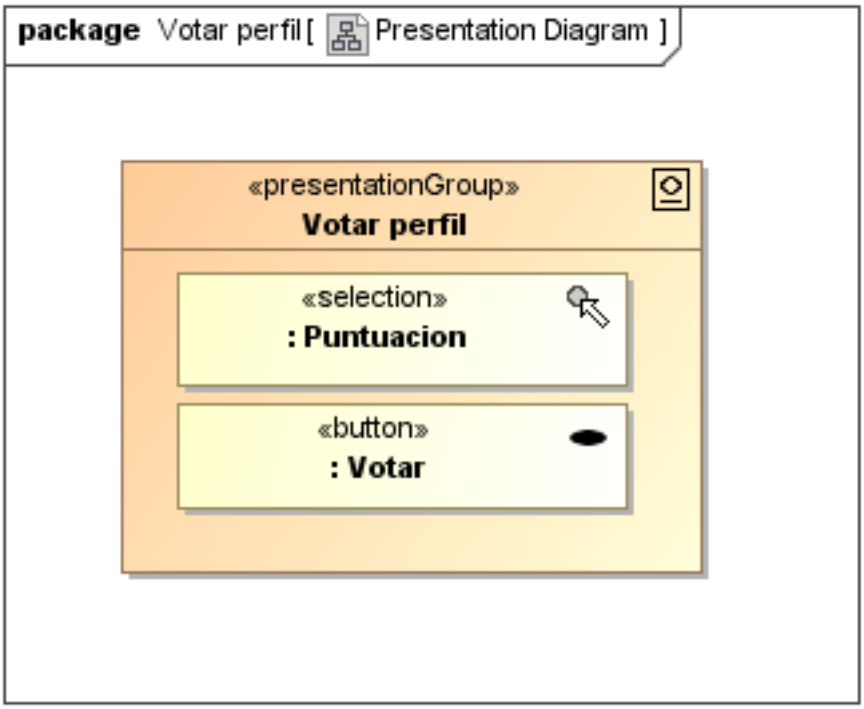

Ilustración 19. Diagrama presentación Votar perfil

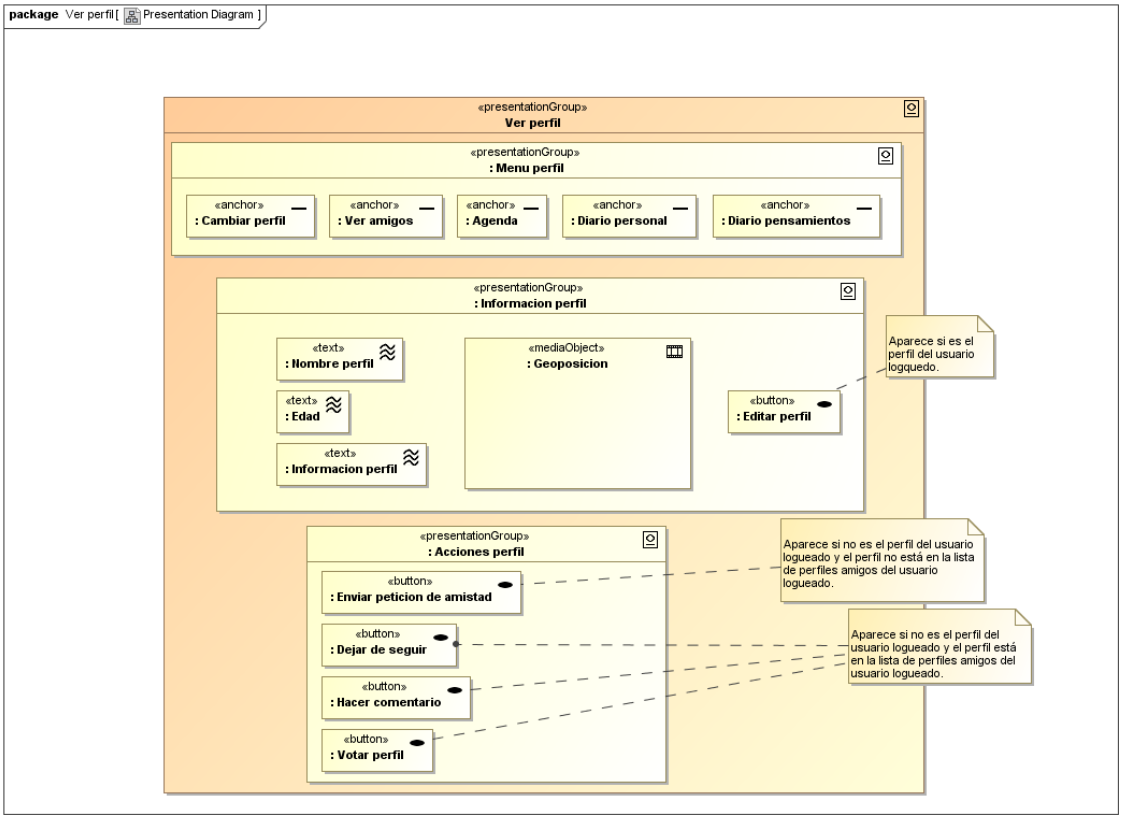

Ilustración 20.Diagrama presentación Ver perfil

4.4 Modelo de Proceso

Este modelo específica las acciones que realiza cada clase de proceso, en este modelo se incluye:

- Modelo de Estructura de Procesos: que define las relaciones entre las diferentes clases proceso.

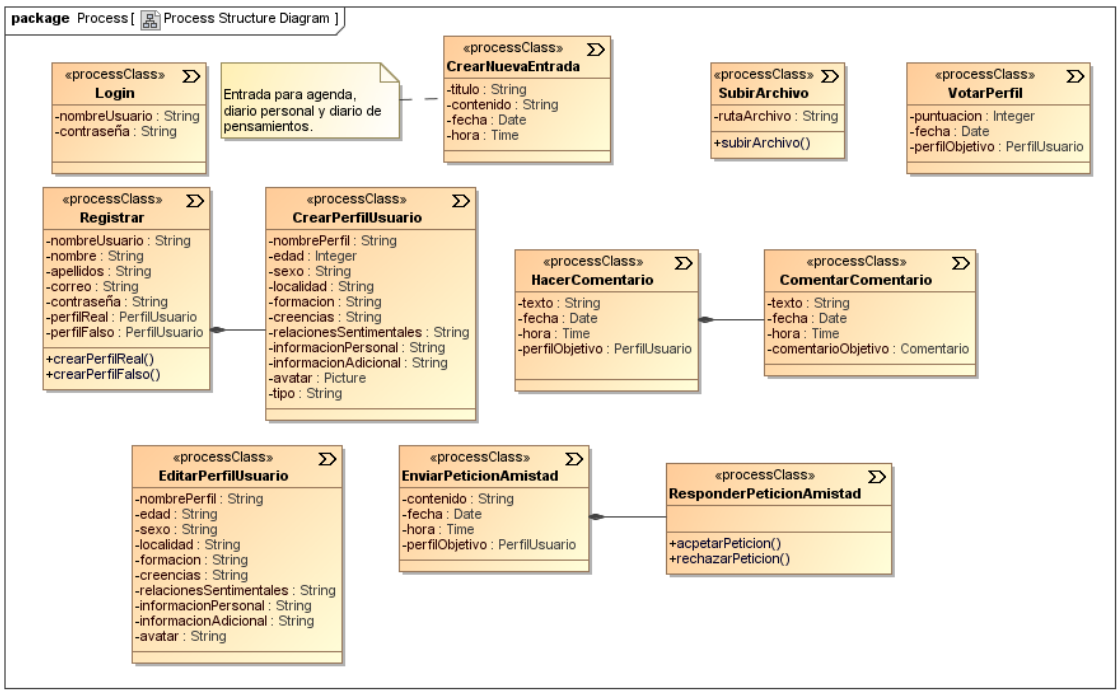

Ilustración 21. Diagrama de Procesos

5. IMPLEMENTACIÓN

Baduco se ha implementado en Django, es un framework de desarrollo web de código abierto, escrito en Python. Está fuertemente inspirado en la filosofía de desarrollo Modelo Vista Controlador, el cual tiene como ventaja que es un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones. Además Django permite el reparto de tareas dentro del equipo de trabajo.

Se podría clasificar a Django como parte de la tercera generación del desarrollo web:

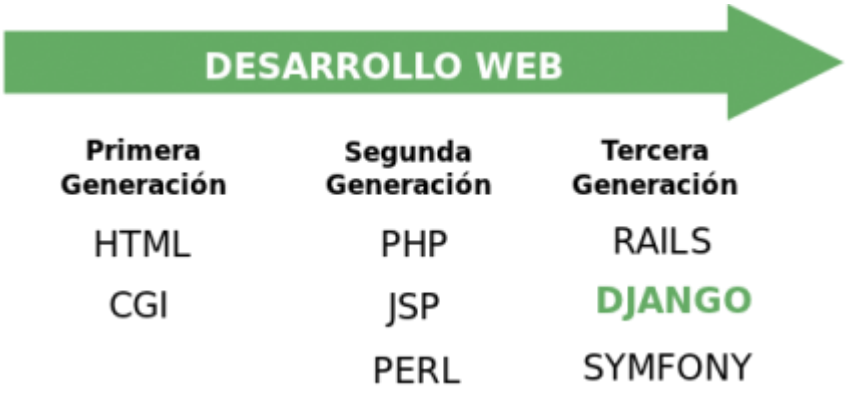

Ilustración 22. Generaciones del desarrollo web

Django es realmente un framework MTV (una modificación de MVC) esto se debe a que los desarrolladores no tuvieron la intención de seguir algún patrón de desarrollo, sino hacer el framework lo más funcional posible.

Para empezar a entender MTV debemos fijarnos en la analogía con MVC.

- El *modelo* en Django sigue siendo **modelo**
- La *vista* en Django se llama **Plantilla (Template)**
- El *controlador* en Django se llama **Vista**

Una imagen nos hará entender mejor esta relación:

Ilustración 23. Funcionamiento del MTV de Django

A lo largo de esta sección veremos que hace cada uno de ellos con más detalle.

5.1 Modelo

El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

A partir del diagrama de clases generado previamente se han creado las diferentes clases del proyecto.

5.1.1 Clase Usuario

```
class Usuario (User):
      perfilReal = models.OneToOneField ('PerfilUsuario', default
= None, related_name='usuario_PerfilReal')
      perfilFalso = models.OneToOneField ('PerfilUsuario', default
= None, related_name='usuario_PerfilFalso')
      perfilActivo = models.CharField (max_length=20, default =
'Real', choices = ABREVIATURAS6)
      def __unicode__(self):
            return self.username
```

Tabla 1. Clase Usuario

Se crea la clase "Usuario" que pertenece al tipo "User" el cual tiene diferentes atributos, algunos de ellos son: username, first_name, last_name, email, password, last_login. Ademas de estos atributos predefinidos, se han definido los siguientes:

- perfilReal: Este atributo sirve para mostrar el perfil que el usuario ha introducido en su registro como información real.
- perfilFalso: Este atributo sirve para mostrar el perfil que el usuario ha introducido en su registro como información falsa.
- perfilActivo: Este atributo mediante la selección de "Real" o "Falso" activa el perfil asociado a cada uno.

5.1.2 Clase PerfilUsuario

```
class PerfilUsuario (models.Model):
      nombre = models.CharField (max_length=30, unique=True, blank
= True, null = True)
      edad = models.IntegerField (range(18,200), blank = True, null
= True)
      sexo = models.CharField (max_length=50, blank = True, null =
True, choices=ABREVIATURAS1)
      localidad = models.CharField (max_length= 50, blank = True,
null = True)
      orientacionSexual = models.CharField (max_length=20, blank =
True, null = True, choices=ABREVIATURAS2)
      profesion = models.CharField (max_length=50, blank = True,
null = True)
      estudios = models.CharField (max_length=50, blank = True,
null = True, choices=ABREVIATURAS3)
      relaciones = models.CharField (max_length=50, blank = True,
null = True, choices=ABREVIATURAS4)
      creenciasReligiosas = models.CharField (max_length=50, blank
= True, null = True, choices=ABREVIATURAS5)
      intereses = models.CharField(max_length=100, blank = True,
null = True)
      informacionAdicional = models.TextField (blank = True, null =
True, help_text="Informacion adicional sobre el usuario")
      amigos = models.ManyToManyField ('PerfilUsuario',
default='None', related_name = "PerfilAmigo_PerfilUsuario")
      imagen = models.ImageField (upload_to = 'usuario',
verbose_name = 'ImagenPerfil')
      # Indica si es el perfil real o el falso
      tipo = models.CharField (max_length = 30)
      latitud=models.FloatField(null=True,blank=True,default=0)
      longitud=models.FloatField(null=True,blank=True,default=0)
      mediaVotos=models.FloatField(null=True,blank=True,default=0)
      def __unicode__ (self):
            return self.nombre
```

Tabla 2. Clase PerfilUsuario

La clase PerfilUsuario contiene la información referente al usuario. Los atributos y los posibles valores de estos son:

- nombre
- edad
- sexo
- localidad
- orientacionSexual
- profesion
- estudios
- relaciones

- creenciasReligiosas
- intereses
- informacionAdicional
- imagen
- tipo
- latitud
- longitud
- mediaVotos

Los atributos "latitud" y "longitud" hacen referencia a la ubicación en tiempo real del usuario dentro del mapa. El atributo "mediaVotos" mantiene el valor de la puntuación obtenida al realizar la media sobre los votos que el usuario recibe en su perfil.

5.1.3 Clase PeticionAmistad

```
class PeticionAmistad (models.Model):
      perfil_solicitante = models.CharField (max_length = 30,
blank = True, null = True)
      fecha = models.DateField (default = None, blank = True, null
= True)
      comentario = models.TextField (blank = True, null = True)
      perfil_objetivo = models.ForeignKey ('PerfilUsuario',
default = 'None', blank = True, null = True, related_name =
"PeticionAmistad_PerfilUsuairo")
      def __unicode__ (self):
            return self.usuario_solicitante
```

Tabla 3. Clase PeticionAmistad

Si el usuario cuenta con un perfil privado, esta clase solicitara la información de los usuarios que quieren visitar su perfil.

Cuenta con los atributos:

- perfil_solicitante
- fecha
- comentario
- perfil_objetivo

El atributo "perfil_objetivo" contiene toda la información sobre el usuario que envía la petición. Los demás atributos indican el nombre del solicitante, el día y hora en que realizo la petición además de un comentario.

5.1.4. Clase Comentario

```
class Comentario (models.Model):
      perfilReceptor = models.ForeignKey ('PerfilUsuario',
related_name = 'Comentario_PerfilUsuario')
      perfilEmisor = models.CharField (max_length = 50, blank =
True, null = True)
      fecha = models.DateField (default = None, blank = True, null
= True)
      hora = models.TimeField (default = None, blank = True, null =
True)
      texto = models.TextField (default = None, blank = True, null
= True)
      comentario = models.ManyToManyField ('Comentario', default =
'None', related_name = "Comentario_Comentario")
      def __unicode__ (self):
            return self.perfilEmisor
```

Tabla 4. Clase Comentario

La clase comentario sirve para que el usuario realice comentarios sobre otros perfiles.

Cuenta con los atributos:

- perfilReceptor
- perfilEmisor
- fecha
- hora
- texto
- comentario

5.1.5. Clase Voto

```
class Voto (models.Model):
      perfilVotado = models.ForeignKey ('PerfilUsuario',
related_name = 'Voto_PerfilUsuario')
      puntuacion = models.IntegerField (choices = RANGO_VOTOS,
blank = True, null = True)
      perfilVotante = models.CharField (max_length = 50, blank =
True, null = True)
      hora = models.TimeField (default = None, blank = True, null =
True)
      texto = models.TextField (default = None, blank = True, null
= True)
      def __unicode__ (self):
                  return self.perfilVotante
```

Tabla 5. Clase Voto

La clase voto sirve para que el usuario pueda valorar los perfiles de otros usuarios.

Cuenta con los atributos:

- perfilVotado
- puntuación
- perfilVotante
- hora
- texto

5.1.6. Clase Agenda

```
class Agenda (models.Model):
      nombre = models.CharField (max_length = 30)
      usuario = models.OneToOneField (Usuario, related_name =
"Agenda_Usuario")

      def __unicode__ (self):
            return self.nombre
```

Tabla 6 Clase Agenda

La clase agenda permite la planificación de las actividades personales del usuario.

Cuenta con los atributos:

- nombre
- usuario

5.1.7. Clase DiarioPersonal

```
class DiarioPersonal (models.Model):
      nombre = models.CharField (max_length = 30)
      usuario = models.OneToOneField (Usuario, related_name =
"DiarioPersonal_Usuario")
      def __unicode__ (self):
            return self.usuario
```

Tabla 7. Clase DiarioPersonal

La clase DiarioPersonal permite disponer de un diario personal del usuario.

Cuenta con los atributos:

- nombre
- usuario

5.1.8. Clase DiarioPensamientos

```
class DiarioPensamientos (models.Model):
      nombre = models.CharField (max_length = 30)
      usuario = models.OneToOneField (Usuario, related_name =
"DiarioPensamientos_Usuario")
      def __unicode__ (self):
            return self.perfilUsuario
```

Tabla 8. Clase DiarioPensamientos

La clase DiarioPensamientos permite registrar los pensamientos que tenga respecto a cualquier ámbito de su vida.

Cuenta con los atributos:

- nombre
- usuario

5.1.9. Clase de Entrada

```
class Entrada (models.Model):
      titulo = models.CharField (max_length = 50, default = None,
blank = True, null = True)
      fecha = models.DateField (default = None, blank = True, null
= True)
      hora = models.TimeField (default = None, blank = True, null =
True)
      contenido = models.TextField (default = None, blank = True,
null = True)
```

Tabla 9. Clase Entrada

La clase entrada contiene todos los atributos para crear una nueva publicación, a continuación se muestran las clases que heredan de ella y pueden crear las diferentes publicaciones según el tipo agenda, diario pensamientos y personal.

Cuenta con los atributos:

- titulo
- fecha
- hora
- contenido

5.1.10. Clase de EntradaAgenda

```
class EntradaAgenda (Entrada):
      agenda = models.ForeignKey ('Agenda', related_name =
"EntradaAgenda_Agenda")
      def __unicode__ (self):
            return self.titulo
```

Tabla 10. Clase EntradaAgenda

La clase EntradaAgenda hereda de la clase Agenda y permite una publicación en la Agenda.

Cuenta con el atributo:

- agenda

5.1.11. Clase de EntradaDiarioPersonal

```
class EntradaDiarioPersonal (Entrada):
      diarioPersonal = models.ForeignKey ('DiarioPersonal',
related_name = "EntradaDiarioPersonal_DiarioPersonal")
      def __unicode__ (self):
            return self.titulo
```

Tabla 11. Clase de EntradaDiarioPersonal

La clase de EntradaDiarioPersonal hereda de la clase Agenda y permite una publicación en el Diario personal.

Cuenta con el atributo:

- diarioPersonal

5.1.12. Clase de EntradaDiarioPensamientos

```
class EntradaDiarioPensamientos (Entrada):
      diarioPensamientos = models.ForeignKey ('DiarioPensamientos',
related_name = "EntradaDiarioPensamientos_DiarioPensamientos")
      def __unicode__ (self):
            return self.titulo
```

Tabla 12. Clase EntradaDiarioPensamientos

La clase de EntradaDiarioPensamientos hereda de la clase Agenda y permite una publicación en el Diario de pensamientos.

Cuenta con el atributo:

- diarioPensamientos

5.2 Vistas

Una función de vista o una vista, como es conocida generalmente, es una función en Python que hace una solicitud Web y devuelve una respuesta Web, esta respuesta puede ser el contenido de una página, un error 404, una imagen, un documento XML, entre muchas cosas más.

La vista contiene toda la lógica necesaria para devolver una respuesta, todas estas respuestas se encuentran en un único archivo y este archivo se llama: views.py, que se encuentra dentro de cada aplicación de Django.

A continuacion se muestra una vista de la aplicación:

```
# Editar perfil
def editarPerfil (request, perfil_id):
      perfil = PerfilUsuario.objects.get (pk = perfil_id)
      if request.method == 'POST':
            form = PerfilForm (request.POST)
            if form.is_valid ():
                  form.save ()
                  return redirect ('/usuario/')
      else:
            form = PerfilForm (instance = perfil)
      context = {'form': form}
      return render (request, 'usuario/editar_perfil.html',
context)
```

Tabla 13. Vista Ver Perfil

5.3 Plantillas

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

Las plantillas son muy importantes, permiten acomodar el resultado que devuelve la vista. Django tiene un estupendo motor de plantillas que permite separar eficientemente la presentación, de la lógica de programación.

A continuación se muestra la plantilla correspondiente a la vista Ver perfil:

```
{% extends 'index.html' %}
{% block TitleWeb %} Editar perfil {% endblock %}
{% block TitleContent %} Editar perfil {{perfil.tipo}} {% endblock %}
{% block Content %}
        <form role = "form" id = "form" method = "post" enctype='multipart/form-data' action = ""> {% csrf_token %}
                <div class = "form-group">
                        <table class = 'formTable'> {{form}} </table>
                        <p><input type = 'submit' value = 'Editar'></p>
        </form>
{% endblock %}
```

Tabla 14. Plantilla Editar Perfil

5.4 Configuración de rutas

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf está construido con expresiones regulares en Python. Este URLConf permite que las rutas que maneje Django seán agradables y entendibles para el usuario.

Si consideramos al URLConf en el esquema anterior tendríamos este resultado más completo.

Ilustración 24. Funcionamiento del MTV de Django y su URLConf

Url correspondiente a Editar Perfil:

```
# Editar perfil
url (r'editar-perfil-(?P<perfil_id>\d+)$', views.editarPerfil,
name = 'editarPerfil'),
```

Tabla 15. Url Editar Perfil

Durante la muestra del código solo se ha hecho una breve reseña a este, para servir de ejemplo en la compresión de cada una de las partes de Django. Se puede consultar el código de la aplicación en los archivos adjuntos.

Bibliografía

[1] UML- Based Web Engineering [Universidad Carlos III de Madrid]