Proyecto: aplicación Web para un zoo

Introducción

En este proyecto desarrollarás la aplicación Web de un zoo.

El proyecto se divide en funcionalidad obligatoria, que todas las entregas deben implementar, y funcionalidad adicional. Para esta última, los estudiantes pueden elegir libremente otra funcionalidad para integrarla en su implementación.

La funcionalidad obligatoria incluye lo siguiente:

- Los clientes podrán consultar la lista de animales del zoo y leer información sobre ellos.
- Los clientes podrán ver las actividades especiales que organiza el zoo, como visitas guiadas, talleres, interacciones con animales, etc., y reservar plaza en ellas.
- Los gestores podrán introducir nuevas actividades y registrar nuevas fechas para actividades existentes.

Funcionalidad obligatoria

Todas las entregas deben implementar la siguiente funcionalidad:

- **Gestión de cuentas de usuario:** hay dos tipos de usuarios, clientes y gestores. Los usuarios podrán crear cuentas, iniciar sesión en ellas y cerrar sesión.
- **Catálogo de animales:** los usuarios podrán consultar la lista de animales del zoo. Para cada animal se mostrará información adicional como, por ejemplo,

nombre científico y clasificación, distribución geográfica, dieta, una breve descripción textual, etc.
- **Actividades:** los gestores podrán introducir nuevas actividades en la aplicación (visitas guiadas, talleres, interacciones con animales, etc.). Para las actividades existentes podrán registrar nuevas fechas. Además, podrán marcar o desmarcar las actividades que se mostrarán en la página principal de la aplicación (solo se mostrarán en la página principal las actividades marcadas con tal fin).
- **Reserva de actividades:** los clientes podrán reservar una o varias plazas para una fecha concreta de una actividad. La aplicación debe comprobar que hay plazas libres antes de confirmar la reserva. Los clientes también podrán consultar la lista de actividades pasadas y futuras en las que se han inscrito.

Las características no enumeradas explícitamente no son obligatorias y se considerarán funcionalidad adicional.

Modelo de datos

El proyecto debe soportar el siguiente modelo de datos para la funcionalidad obligatoria. Puedes cambiar el modelo de datos si tienes razones justificadas para ello. Por ejemplo, puedes enriquecer este modelo de datos debido a las necesidades de cualquier característica adicional que implementes (añadir nuevas entidades, añadir nuevas propiedades para algunas entidades, etc.), o si crees que un esquema diferente es mejor en el contexto de tu diseño del proyecto.

Las principales entidades a almacenar en la base de datos son:

- **Usuario:** los usuarios se identifican por un email y se autentican con una contraseña, que debe almacenarse salada y cifrada en la base de datos. La mayoría de usuarios tendrán el rol de *cliente*, pero algunos usuarios tendrán el rol de *gestor*, que les permite utilizar las funcionalidades de gestión.
- **Animal:** los animales tienen un nombre y otra información asociada.
- **Actividad:** las actividades tienen un título, una descripción textual y cualquier otra información que se desee como, por ejemplo, un rango de edades recomendado. Pueden estar marcadas o desmarcadas para que se muestren en la página principal de la aplicación.
- **Actividad programada:** una misma actividad puede programarse para muchas fechas. Para cada actividad programada, hay que almacenar su fecha y hora, el número de plazas ofertadas y su precio.
- **Reserva de actividad:** cada reserva está definida por el usuario que la realizó, una actividad programada, un número de plazas y la fecha y hora en que se confirmó la reserva.

Vistas

La aplicación debe proporcionar al menos las siguientes vistas:

- **Vista principal** (abierta para usuarios no autenticados): se muestra información promocional sobre el zoo. Se muestran las actividades marcadas para aparecer en la página principal. En algún lugar de esta vista se muestra un enlace al catálogo de animales.
- **Vista de catálogo de animales** (abierta para usuarios no autenticados): se muestra el catálogo de animales. Cada animal enlaza a su *vista de animal* descrita más abajo.
- **Vista de animal** (abierta para usuarios no autenticados): se muestra información detallada sobre un animal.
- **Vista de actividad** (abierta para usuarios no autenticados): se muestra información detallada sobre una actividad y sus fechas para los próximos días. Se debe indicar para qué fechas hay plazas disponibles y para cuáles no. Los clientes autenticados deben poder iniciar el proceso de reserva para las fechas que se muestren en esta vista y que tengan plazas disponibles. Los gestores autenticados deben poder programar la actividad para nuevas fechas y horas, y marcar o desmarcar la actividad para que se muestre en la página principal.
- **Vista de cliente** (autenticación requerida): los clientes pueden ver en esta vista sus propias reservas futuras y pasadas.

Toma la lista de vistas anterior como una sugerencia. Tienes libertad para diseñar tu aplicación con vistas diferentes siempre que proporciones la misma funcionalidad. También puedes cambiar estas vistas para acomodar características adicionales.

Aspectos que no están incluidas en la funcionalidad obligatoria

Como se ha dicho antes, las características no enumeradas explícitamente no son obligatorias y se considerarán funcionalidad adicional. En particular, desde el punto de vista de la funcionalidad obligatoria:

- Puedes insertar animales y sus datos manualmente en la base de datos mediante un programa Python que simplemente cree unos pocos animales, con unos pocos datos cada uno, y los inserte en la base de datos.
- La forma más sencilla de mostrar una foto por cada animal es guardar todas las fotos en el directorio donde se almacenan los ficheros estáticos de tu aplicación, con un nombre de fichero que dependa del identificador del animal. Por ejemplo, si el identificador del oso panda es 42, puedes almacenar su foto

con un nombre de fichero como **`animal/42.jpg`** dentro del directorio de ficheros estáticos de tu aplicación. De esta forma, es fácil en tus plantillas construir la ruta de la foto a partir del identificador del animal.
- Puedes asignar el rol de gestor a usuarios concretos manualmente en la base de datos, mediante un programa Python que simplemente busque al usuario y le asigne el rol de gestor. De esta forma, puedes crear una cuenta de cliente normal con la funcionalidad normal de tu aplicación y después actualizarla manualmente al rol de gestor.

Funcionalidad adicional

Los estudiantes pueden integrar cualquier funcionalidad adicional en su proyecto. Cualquier característica más allá de las obligatorias se considerará funcionalidad adicional.

Criterios de evaluación

La evaluación tendrá en cuenta la correcta implementación de la funcionalidad obligatoria y la funcionalidad adicional, así como algunos aspectos de calidad de su implementación:

- **Funcionalidad obligatoria (7,5 puntos):** la máxima puntuación en este bloque se otorgará a las entregas con una implementación de toda la funcionalidad obligatoria que funcione correctamente y cumpla todos los requisitos de calidad que se enumeran a continuación.
- **Funcionalidad adicional (2,5 puntos):** los estudiantes deben implementar funcionalidad adicional, también funcionando correctamente y cumpliendo todos los requisitos de calidad. Las características *interesantes*, en el sentido de que su implementación vaya más allá de lo que hemos visto en clase, obtendrán puntuaciones más altas. Un ejemplo de ello es el uso de JavaScript para hacer que la aplicación sea más interactiva en el lado del navegador y para comunicarse de forma asíncrona con el servidor. Las características *aburridas*, en el sentido de que son bastante repetitivas con respecto a otras características que ya has implementado, podrían obtener pocos puntos o ningún punto en absoluto. Una interfaz de usuario brillante, en el sentido de que su apariencia es notable, al nivel de los estándares actuales para aplicaciones comerciales, también se considerará una característica adicional. Puedes consultar con tus profesores a cuántos puntos aspiras con las características adicionales que tengas en mente.

Los **requisitos de calidad** que se tendrán en cuenta son:

- Que cada característica implementada funcione correctamente, es decir, se comporte comforme a sus requisitos.
- Que el código fuente esté correctamente organizado y sea fácil de leer y entender. Debería seguir los principios explicados en las prácticas 3 a 6 en cuanto a organización del código. No se requieren comentarios extensos en el código.
- Que el uso de HTML y CSS sea correcto.
- Que las vistas de la aplicación (su interfaz de usuario) sean claras y fáciles de usar. Que la apariencia visual sea *razonablemente* buena.
- Que se manejen correctamente potenciales situaciones de error, dando a los usuarios una respuesta adecuada cuando sea necesario.

Pistas de implementación

Esta sección da algunas pistas sobre cómo implementar ciertas características obligatorias. Los profesores pueden añadir nuevas pistas a esta sección en el futuro, cuando consideren que es apropiada ayuda adicional con algunas características.

Cuentas de usuario y autenticación

Puedes reutilizar el código que necesites de los laboratorios 3 a 6.

Inserciones manuales de datos en base de datos

Para algunas entidades, como animales y sus datos, está bien insertarlos manualmente en la base de datos, ya que no es obligatorio para tu aplicación implementar el código necesario para crearlos y editarlos.

Modelo de datos: relaciones entre entidades

Recuerda añadir una columna con un identificador (número entero) a cada entidad, que será la clave primaria de esa tabla. Revisa tu código de laboratorios anteriores para ver ejemplos de claves primarias.

Modela las relaciones *muchos-a-uno* de la misma forma que en laboratorios anteriores. Por ejemplo, hay una relación *muchos-a-uno* entre actividades y

actividades programadas, en el sentido de que una actividad programada está asociada a una única actividad y una misma actividad puede estar programada muchas veces. Esta relación es similar a la que hay entre mensajes y usuarios en los laboratorios, y por tanto debes usar anotaciones similares para modelarlas.

Otros ejemplos de relaciones *muchos-a-uno* son las que hay entre usuarios y reservas de actividades y entre reservas de actividades y actividades programadas.

Flask

Modela las relaciones *muchos-a-uno* definiendo una clave ajena (**`db.ForeignKey`**) en el lado *muchos* y una relación (**`relationship`**) en el lado *uno*. El siguiente ejemplo modelaría la relación entre reservas y usuarios:

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    (...)
    reservations = db.relationship(
        "Reservation",
        backref="user",
        lazy=True
    (...)
)

class Reservation(db.Model):
    (...)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    (...)
```

La declaración **`relationship`** creará automáticamente un atributo **`user`** en la clase **`Reservation`** y un atributo **`reservations`** en la clase **`User`**. Dada una reserva llamada **`reservation`**, puedes leer o escribir su usuario a través de **`reservation.user`**. Dado un usuario llamado **`user`**, puedes acceder a su lista de reservas con **`user.reservations`**.

Modelo de datos: tipos de datos enumerados

Un usuario de la aplicación tendrá uno de los siguientes dos roles: *cliente* o *gestor*. En lugar de modelarlos como cadenas, es mejor usar un tipo de datos enumerado.

Flask

Las columnas con tipos de datos enumerado se declaran así:

```
class UserRole(enum.Enum):
    customer = 1
    manager = 2

class User(db.Model):
    (...)
    role = db.Column(db.Enum(UserRole), nullable=False)
    (...)
```

La sintaxis para acceder a estas columnas en tus objetos de modelo es la siguiente:

```
user = model.User(
    name="Mary",
    role=model.UserRole.manager,
    (...)
)

if user.role == model.UserRole.manager:
    (...)
```

Desde las plantillas Jinja, el acceso es menos intuitivo:

```
{% if user.role == user.role.__class__.manager %}
```

Actualizaciones de tu modelo de datos

Es normal, durante el desarrollo de tu proyecto, y especialmente mientras defines tu modelo de datos, que necesites hacer algunos cambios a un modelo que ya has creado en tu base de datos.

Si en algún momento cambias tus clases de modelo, la forma más sencilla de propagar los cambios a la base de datos ya existente es eliminar todas las tablas y crearlas de nuevo (pero ten en cuenta que perderás todos los datos que hayas almacenado allí).

Flask

En Flask, puedes ejecutar los siguientes comandos en un terminal de Python:

```
# Change 'zoo' to your package's name below
from zoo import db, create_app
app = create_app()
db.drop_all(app=app)
db.create_all(app=app)
```

Contar sitios reservados

Necesitarás ser capaz de contar el número total de sitios que se han reservado para una actividad programada. Por ejemplo, no puedes aceptar reservas si no hay suficientes sitios libres.

Flask

Puedes usar la función **`func.sum`** para ejecutar una consulta que sume valores:

```
# Example with the projection with id 10:
scheduled_activity = (model.ScheduledActivity.query
                        .filter(model.ScheduledActivity.id == 10)
                        .one()
)
sum_result = db.session.query(
    db.func.sum(model.Reservation.num_places).label('reserved')
).filter(
    model.Reservation.scheduled_activity == scheduled_activity
).one()
num_reserved_places = sum_result.reserved
```

Dónde obtener documentación

Flask

Consulta la documentación oficial de Flask si necesitas ayuda con el *framework* Flask. También puedes leer la documentación oficial de Jinja para obtener ayuda sobre el uso de plantillas en Flask.

Usarás el ORM SQLAlchemy para acceder a tu base de datos. Sin embargo, en lugar de usar SQLAlchemy directamente, usarás la extensión **`Flask-SQLAlchemy`**, que hace que SQLAlchemy sea un poco más fácil de usar en Flask. Sin embargo, cambia algunas cosas con respecto a SQLAlchemy puro. Lee primero la documentación de **`Flask-SQLAlchemy`** y luego el tutorial ORM de SQLAlchemy. Para saber cómo declarar relaciones entre tablas (*uno a uno*, *uno a muchos*, *muchos a muchos*, etc.) puedes consultar la guía de configuración de relaciones.

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática