uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Aplicaciones Web
Grado en Ciencia e Ingeniería de Datos
OpenCourseWare 2023

Examen final, 2ª parte
Duración: 110 min.

Nota: para facilitar contestar a este examen, en las últimas páginas encontrarás algunos fragmentos de código de referencia.

Problema (5 puntos)

En el contexto de la aplicación web del zoo que has desarrollado para el proyecto, necesitas programar una nueva funcionalidad relacionada con la definición de diferentes zonas en el zoo. Para cada tipo de animal (por ejemplo, elefantes) puede declararse que se encuentra en una zona concreta, y cada zona puede contener varios tipos de animales. Puede ocurrir que para algunos animales no se haya declarado en qué zona se encuentran.

Apartado 1 (1 punto)

Actualiza el modelo de datos de *SQLAlchemy* para permitir esta funcionalidad:

- Programa la clase de modelo que representa zonas con un identificador numérico, un nombre de zona (hasta 64 caracteres) y una descripción textual (hasta 1024 caracteres).
- Actualiza la clase de modelo para animales con el atributo o atributos necesarios (no es necesario programar el resto de la clase, solo el nuevo atributo o atributos).

Tu modelo debe declarar la relación de forma que los objetos de tipo zona tengan un atributo que contenga la lista de animales que se encuentran en esa zona

Soluciones

```
class Zone(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), nullable=False)
    description = db.Column(db.String(1024), nullable=False)
    animals = db.relationship(
        "Animal",
        backref="zone",
        lazy=True
    )

class Animal(db.Model):
    (...)
    zone_id = db.Column(db.Integer, db.ForeignKey("zone.id"), nullable=False)
```

Apartado 2 (1 punto)

La aplicación necesita una vista que muestre los animales que se encuentran en una zona dada. La ruta para acceder a esta vista será la concatenación de `/zone/` y el identificador de la zona, por ejemplo `/zone/5`.

Programa la función de controlador que responde a esta ruta. La plantilla a cargar para esta vista, que programarás en el siguiente apartado del problema, se llamará `zone.html`.

Soluciones

```
@bp.route("/zone/<int:zone_id>")
@flask_login.login_required
def post(zone_id):
    zone = model.Zone.query.filter_by(id=zone_id).first()
    if not zone:
        abort(404, f"Zone id {zone_id} doesn't exist.")
    return render_template("main/post.html", zone=zone)
```

Apartado 3 (1 punto)

Escribe la parte de la plantilla Jinja `zone.html` que muestra el nombre de la zona, su descripción y la lista (lista con puntos de HTML) de animales que se encuentran en ella. Para cada animal debes mostrar su nombre como un hipervínculo a su vista de animal. Supón que la vista de animal es mostrada por una función con la siguiente declaración en el blueprint `main`:

```
@bp.route("/animal/<int:animal_id>")
def view_animal(animal_id): ...
```

Soluciones

```
<div>Zona: {{ zone.name }}</div>
<div>{{ zone.description }}</div>
<div>Animales en esta zona:<div>
<ul>
{% for animal in zone.animals %}
    <li>
        <a href="{{ url_for('main.view_animal', animal_id=animal.id) }}">
            {{ animal.name }}
        </a>
    </li>
{% endif %}
</ul>
```

Apartado 4 (1 punto)

Escribe una función de controlador para la ruta `/move`. Recibe una petición POST con dos parámetros que provienen de un formulario HTML: el identificador de la zona y el identificador del animal. La función debe registrar que el animal especificado se encuentra ahora en la zona especificada.

Finalmente, la función debe enviar una respuesta de redirección indicando al navegador que cargue, para dicha zona, la vista de la zona que has programado en los apartados 2 y 3.

La función de controlador debe comprobar que existe un usuario autenticado y que ese usuario tiene el rol de manager, y devolver un error 403 (*forbidden*) en caso contrario.

También debe comprobar que el animal y la zona existen en la base de datos, y devolver un error 404 (*not found*) en caso contrario.

Soluciones

```
@bp.route("/move", methods=["POST"])
@flask_login.login_required
def move():
    if not current_user.is_manager():
        abor(403, "This action needs manager privileges.")
    zone_id = request.form.get("zone_id")
    animal_id = request.form.get("animal_id")
    zone = model.Zone.query.filter_by(id=int(zone_id)).first_or_404()
    animal = model.Animal.query.filter_by(id=int(animal_id)).first_or_404()
    animal.zone = zone
    db.session.commit()
    return redirect(url_for("main.zone", zone_id=zone.id))
```

Apartado 5 (1 punto)

El formulario que llama al controlador del apartado 4 se colocará en la plantilla Jinja de la vista de animal. Programa ese formulario teniendo en cuenta que:

- El formulario solo se mostrará en el caso de que haya un usuario autenticado y que ese usuario tenga el rol de administrador. Tu código debe comprobar eso (el usuario está disponible en la variable llamada `current_user`, que contiene un atributo `is_authenticated`).
- El identificador de animal debe especificarse en un campo oculto del formulario.
- El usuario seleccionará la zona con un menú desplegable (un elemento `select`) que muestre los nombres de todas las zonas.
- El botón que envía el formulario mostrará el texto "mover".
- La plantilla recibe una variable llamada `animal` con el objeto animal y una variable llamada `zones` con la lista de todos los objetos zona.

Nota: Solo necesitas programar la parte de la vista de animal que muestra el formulario.

Soluciones

```
{% if current_user.is_authenticated() and current_user.is_manager() %}
<form action="{{ url_for('move') }}" method="post">
    <input type="hidden" name="animal_id" value="{{ animal.id }}">
    <legend>
        Selecciona la zona:
        <select name="zone_id">
        {% for zone in zones %}
            <option value="{{ zone.id }}">{{ zone.name }}</option>
        {% endif %}
        </select>
    </legend>
    <input type="submit" value="Move">
</form>
{% endif %}
```

Código de referencia

Fragmento 1

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(128), unique=True, nullable=False)
    name = db.Column(db.String(64), nullable=False)
    password = db.Column(db.String(100), nullable=False)
    role = db.Column(db.Enum(UserRole), nullable=False)
    reservations = db.relationship(
        "Reservation",
        backref="user",
        lazy=True
    )

class Movie(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(512), nullable=False)
    director = db.Column(db.String(512), nullable=False)
    starring = db.Column(db.String(512), nullable=False)
    projections = db.relationship(
        "Projection",
        backref="movie",
        lazy=True
    )

class Screen(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(32), nullable=False)
    num_seats = db.Column(db.Integer, nullable=False)
    projections = db.relationship(
        "Projection",
        backref="screen",
        lazy=True
    )

class Projection(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    movie_id = db.Column(db.Integer, db.ForeignKey("movie.id"), nullable=False)
    screen_id = db.Column(db.Integer, db.ForeignKey("screen.id"), nullable=False)
    date = db.Column(db.DateTime, nullable=False)
    reservations = db.relationship(
        "Reservation",
        backref="projection",
        lazy=True
    )
```

```
class Reservation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    projection_id = db.Column(db.Integer, db.ForeignKey("projection.id"), nullable=False)
    num_seats = db.Column(db.Integer, nullable=False)
    confirmation_date = db.Column(db.DateTime, nullable=False)
```

Fragmento 2

```
<div class="message">
    <div class="text">{{ post.text}}</div>
    <div class="metadata">
        <span class="author"><a href="{{ url_for('main.user', user_id=post.user.id) }}">
            {{ post.user.name }}</a></span>
        <span class="date">{{ post.timestamp }}</span>
    </div>
    {% if post.response_to is none %}
        <div>
            <a href="{{ url_for('main.post', message_id=post.id) }}">Go to the post</a>
        </div>
    {% endif %}
</div>
```

Fragmento 3

```
@bp.route("/post/<int:message_id>")
@flask_login.login_required
def post(message_id):
    message = model.Message.query.filter_by(id=message_id).first()
    if not message:
        abort(404, "Post id {} doesn't exist.".format(message_id))
    if message.response_to is not None:
        abort(403, "No view for response messages is available")
    responses = (
        model.Message.query.filter_by(response_to=message)
        .order_by(model.Message.timestamp.desc())
        .all()
    )
    return render_template("main/post.html", post=message, responses=responses)
```

Fragmento 4

```
@bp.route("/new_post", methods=["POST"])
@flask_login.login_required
def new_post():
    text = request.form.get("text")
    response_to_str = request.form.get("response_to")
    if response_to_str is not None:
        response_to = model.Message.query.filter_by(id=int(response_to_str))\
            .first_or_404()
    else:
        response_to = None
    message = model.Message(
        user=flask_login.current_user,
        text=text,
        timestamp=datetime.datetime.now(dateutil.tz.tzlocal()),
        response_to=response_to,
    )
    db.session.add(message)
    db.session.commit()
    if response_to is not None:
        message_to_show = response_to.id
    else:
        message_to_show = message.id
    return redirect(url_for("main.post", message_id=message_to_show))
```