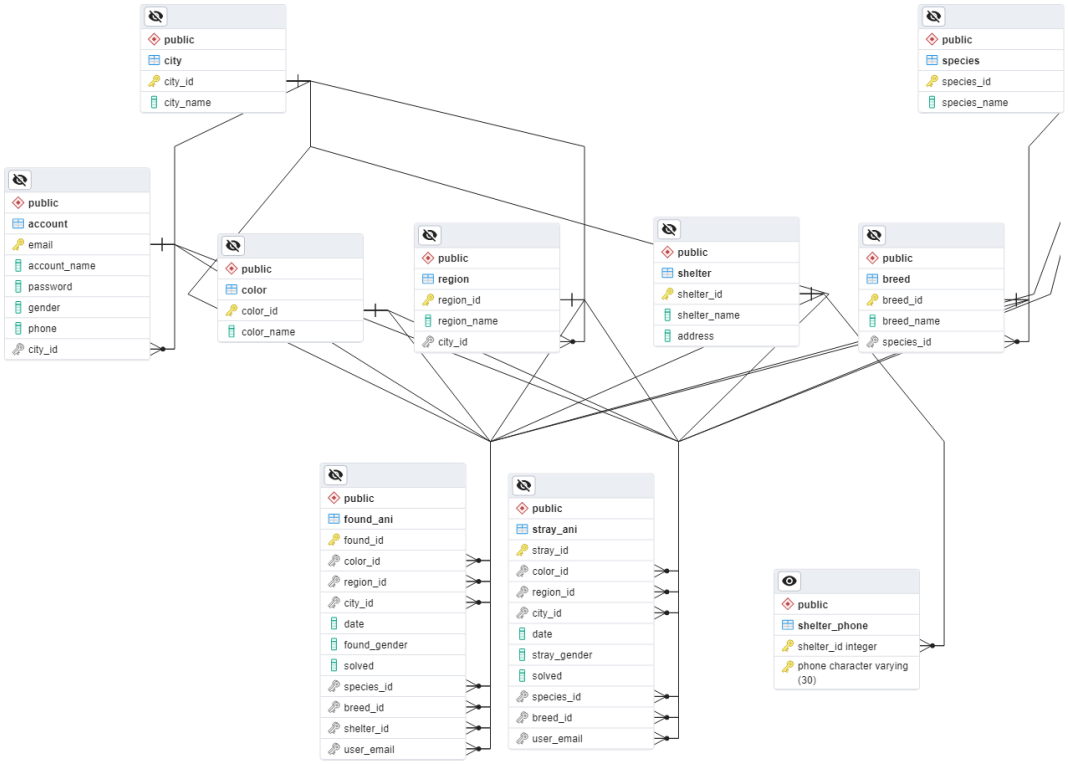
Final Project Report

Group 6
Member: 111550067 李芫、111550139 郭芷安、111550108 吳佳諭、
111550002 林宜頡、11155075 顏名柔
Demo: https://youtu.be/2N-fpiiBg14?si=1wRoievq2xPjsWjc

1. An introduction of your application, including why you want to develop the application and the main functions of your application.
   在現代生活中，許多人都會選擇養毛小孩來陪伴自己，但這也導致流浪動物的數量上升，其中以流浪貓狗最為常見。於是我們設計了一個流浪動物申報的網站，這個網站有以下三種功能：
   (1) 我要申報：在這個頁面，可以選擇要申報動物走失或動物尋獲。若寵物走失，可至此發布協尋。若尋獲動物，也可以在此登記尋獲寵物的基本資訊。
   (2) 我要幫忙：可以用寵物特徵查詢擁有相關特徵的所有網站記錄。
   (3) 帳號登入：登入帳號後，可以執行登記走失或尋獲的紀錄，並察看曾經刊登的走失/尋獲紀錄。
   (4) 其餘資訊：全台31間公立收容所的地址與電話。

2. Database design - describe the schema of all your tables in the database, including keys and index, if applicable (why you need the keys or why you think adding an index is or is not helpful).
   - Schema



   - index:
     因為台灣的收容所數目以及常見的流浪貓狗的品種並不是非常多，搜索時不用花費大量的時間，故我們尚沒有使用index來優化此部分的速度。

3. Database design - describe the normal form of all your tables. If the tables are not in BCNF, please include the reason (performance trade-off, etc.).
    - city
      city(city_id, city_name)
      Normal Form: BCNF
      Functional Dependency:
        (1) {city_id } -> { city_name}
        (2) {city_name } -> { city_id}
      Test: {city_id} and {city_name} are superkey.

    - region
      region(region_id, region_name, city_id)
      Normal Form: BCNF
      Functional Dependency: {region_id } -> { region_name, city_id}
      Test: There is only one primary key and no transitive dependency.

    - species
      species(species_id, species_name)
      Normal Form: BCNF
      Functional Dependency: {species_id } -> { species_name}
      Test: Both {species_id} and {species_name} are superkey.

    - breed
      breed(breed_id, breed_name, species_id)
      Normal Form: BCNF
      Functional Dependency: {breed_id } -> { breed_name, species_id}
      Test: There is only one primary key and no transitive dependency.

    - color
      color(color_id, color_name)
      Normal Form: BCNF
      Functional Dependency: {color_id } -> { color_name}
      Test: Both {color_id} and {color_name} are superkey.

    - shelter
      shelter(shelter_id, shelter_name, address)
      Normal Form: BCNF
      Functional Dependency: {shelter_id } -> { shelter_name, address}
      Test: There is only one primary key and no transitive dependency.

    - shelter_phone
      shelter_phone(shelter_id, phone)
      Normal Form: BCNF
      Functional Dependency: {shelter_id, phone } -> {shelter_id, phone}
      Test: All of the columns are primary keys.

- account
  account(email, account_name, password, gender, phone, city_id)
  Normal Form:
  Functional Dependency: BCNF
  (1) {email } -> { account_name, password, gender, phone, city_id}
  Test: There is only one primary key and no transitive dependency.

- found_ani
  found_ani(found_id, date, found_gender, solved, user_email, color_id, region_id, city_id, species_id, breed_id, shelter_id)
  Normal Form: 2NF
  Functional Dependency:
  (1) {found_id } -> { date, found_gender, solved, user_email, color_id, region_id, city_id, species_id, breed_id, shelter_id}
  (2) {region_id} -> {city_id}
  Test: There is no partial dependency.
  Reason: 我們在application中設計了一個功能：選擇城市/地區，可以將該處所有尋獲的動物列出來。為了方便實現這部分的功能，我們將region和city同時保留在found_ani裡。

- stray_ani
  stray_ani(stray_id, date, stray_gender, solved, user_email, color_id, region_id, city_id, species_id, breed_id)
  Normal Form: 2NF
  Functional Dependency:
  (2) {stray_id } -> { date, stray_gender, solved, user_email, color_id, region_id, city_id, species_id, breed_id}
  (3) {region_id} -> {city_id}
  Test: There is no partial dependency.
  Reason: 同found_ani中所述，是基於application層面的考量。

4. From the data sources to the database - describe the data source and the original format.

Table	Data Source	Original Format
city	維基百科 (https://reurl.cc/97GnMO)	網頁資料(html)
region	https://reurl.cc/545R9y	.csv
breed	貓：https://reurl.cc/545R6M 狗：https://reurl.cc/mroYgj 禽：https://reurl.cc/VNDWrA	貓&鳥：網頁資料(html) 狗：jpg
shelter	https://reurl.cc/rrDYzO	網頁資料(html)

species：由於我們的目的是著重於刊登流浪貓狗的部分，所以只將物種分為常見貓、狗和鳥，其餘歸類至其他。

color：紀錄生活中常見的流浪動物顏色，自行整理成csv檔。
found_ani/ stray_ani / account：展示用資料，使用ChatGPT隨機生成。

5. From the data sources to the database - describe the methods of importing the original data to your database and strategies for updating the data, if you have one.
除了原本就是csv files的資料以外，我們都是利用excel，將網路上搜尋到的資料彙整，提取較常見的部分(例：常見的流浪貓狗品種)，手動輸入到excel，儲存成csv files。最後使用pgAdmin，將csv files匯入database。

在city、region、shelter、shelter_phone、species、breed和color中，我們使用固定的data，沒有更新的功能。而在found_ani、stray_ani和account，每當有新的使用者註冊或尋獲/走失的申報時，會觸發資料的更新，將其新增的帳號及尋獲/走失加入資料庫中。

6. Application with database - explain why your application needs a database.
因為流浪動物的移動範圍較廣，可能在某地區走失的動物，會被送到其他地區的收容所，以致發生重複上傳資料或者資料並未同時編輯/刪除的情況。因此，我們使用資料庫來整合這些更動頻率很高的資料，來保證各地收容所資料的一致性以及避免重複。另外，這個網站也可以協助遺失寵物的人可以有系統的查詢是否有人尋獲他遺失的寵物，或尋獲遺失寵物的人也可以用網站找到疑似的主人並連繫。

7. Application with database - includes the queries that are performed by your application, how your application performed these queries (connections between application and database), and what the cooperating functions for your application.
   - 使用SQLALchemy 連線後端資料庫並向資料庫發送query。先建立SQLALchemy 的table 模型，之後就能以此模型作為媒介對資料庫進行操作。
     - 以speicies 的table 為例

```
class speciesTable(db.Model):
    __tablename__ = 'species'
    species_id = db.Column(db.Integer, primary_key=True)
    species_name = db.Column(db.String(30))
```

   - 當使用者使用查詢功能時，先使用request.form.get()取得使用者在頁面所選的選項，再使用filter_by()設定query的條件，使用query將資料取出後，把取得的資料return回頁面。
     - 以page3查詢存在資料庫中的案件為例，使用request.form.get()取得使用者在頁面所選的選項

```
if request.method == 'POST':
    select_species = request.form.get('whatspecies')
    select_breed = request.form.get('whatbreed')
    select_color = request.form.get('whatcolor')
    select_gender = request.form.get('whatgender')
    select_city = request.form.get('whatcity')
    select_area = request.form.get('whatarea')
    select_date = request.form.get('whatdate')
```

- 將資料query出來

```
    found_animals = (
db.session.query(
    found_ani.found_id,
    speciesTable.species_name,
    breedTable.breed_name,
    colorTable.color_name,
    found_ani.found_gender,
    cityTable.city_name,
    regionTable.region_name,
    found_ani.date,
    found_ani.solved
)
.join(speciesTable, found_ani.species_id == speciesTable.species_id)
.join(breedTable, found_ani.breed_id == breedTable.breed_id)
.join(colorTable, found_ani.color_id == colorTable.color_id)
.join(cityTable, found_ani.city_id == cityTable.city_id)
.join(regionTable, found_ani.region_id == regionTable.region_id)
.filter(
    and_(
        (found_ani.species_id == select_species if select_species is not None else True),
        (found_ani.breed_id == select_breed if select_breed is not None else True),
        (found_ani.color_id == select_color if select_color is not None else True),
        (found_ani.found_gender == select_gender if select_gender is not None else True),
        (found_ani.city_id == select_city if select_city is not None else True),
        (found_ani.region_id == select_area if select_area is not None else True),
        (found_ani.date == select_date if (select_date is not None and select_date != "") else True)
    )
)
.order_by(found_ani.found_id) .all()
```

- 將query出來的資料return到介面

```
return render_template('page3.html', title=title, breed_names=breed_names, species_names=species_names,
                       color_names=color_names,city_names=city_names, region_names=region_names)
```

➢ 使用者要提交資料時先使用request.form.get()取得使用者填寫的資料後，使用add將資料回傳給資料庫。

- 以註冊功能為例，先使用request.form.get()取得使用者填寫的資料

```
if request.method == 'POST':
    name = request.form.get('name')
    email = request.form.get('email')
    password = request.form.get('password')
    phone = request.form.get('phone')
    gender = request.form.get('select-gender')
    city_id = int(request.form.get('select-city'))
```

➢ 使用者可以點選更新狀態，用下拉式表單選取自己曾經的提交資料中，狀態為未解決的案件，讓案件狀態可以更改為已解決。

- 以update為例，為了讓所有使用者都可以知道該案件是否已被解決，我們讓使用者擁有可以自己更新案件是否已解決的功能。做法是定義一個實體 st 獲取 id 與前端表單回傳 id 相同的資料，更改實體的值並執行，達到更改資料庫內容的目的

```
stray_alias = aliased(stray_ani)
species_alias = aliased(speciesTable)
color_alias = aliased(colorTable)
region_alias = aliased(regionTable)
city_alias = aliased(cityTable)
breed_alias = aliased(breedTable)
stray = (
    db.session.query(stray_alias.stray_id, stray_alias.solved, stray_alias.stray_gender, stray_alias.date, species_alias.species_
.join(species_alias, stray_alias.species_id == species_alias.species_id)
.join(color_alias, stray_alias.color_id == color_alias.color_id)
.join(region_alias, stray_alias.region_id == region_alias.region_id)
.join(city_alias, stray_alias.city_id == city_alias.city_id)
.join(breed_alias, stray_alias.breed_id == breed_alias.breed_id)
)
stray = stray.filter(stray_alias.user_email == current_user.email, (stray_alias.solved == 'no' or stray_alias.solved == '否'))
request_stray = stray.all()
if request.method == 'POST':
    select_ani = request.form.get('done')
    # if(select_ani):
    st = stray_ani.query.filter_by(stray_id = select_ani).first()
    if st:
        st.solved = "yes"
        db.session.flush()
        db.session.commit()
    return redirect(url_for('update_success'))
return render_template('change_condition.html', request_stray = request_stray,  current_user = current_user, title = title)
```

8. All the other details of your application that you want us to know.

- 連線問題
  不是所有人都能夠連上我們的網站。
  我們透過AWS建立資料庫。一開始為了方便作業，在security group中添加了0.0.0.0的IP位址讓所有人都可以連進資料庫，但也導致我們的資料被盜。因此在新建的資料庫，我們的security group只有添加組員們的IP位址，讓資料更安全。

- 登入系統
  我們使用flask內建的flask_login建立登入系統。
  先初始化此網站的LoginManager。

```
# Flask-Login setup
login_manager = LoginManager(app)
login_manager.login_view = 'login'
```

接著再設定使用者的資訊，並將使用者的id設為其email。

```
# User class for Flask-Login
class User(UserMixin, db.Model):
    __tablename__ = 'account'
    account_name = db.Column(db.String(60))
    email = db.Column(db.String(30), primary_key=True)
    password = db.Column(db.String(20))
    phone = db.Column(db.String(20))
    gender = db.Column(db.String(3))
    city_id = db.Column(db.Integer)
    def get_id(self):
        return str(self.email)
```

設置使用者登入時，返回User物件，且id為剛才定義的email。

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(user_id)
```

完成後，便能進行登入系統的操作。此處驗證email與password是否存於資料庫中的"account" table，若有的話，則login_user；否則則登入失敗。

```
user = User.query.filter_by(email=email, password=password)
if user:
    login_user(user)
    flash('登入成功！', 'success')
    return redirect(url_for('index'))
else:
    flash('登入失敗，請檢查您的帳號和密碼。', 'error')
```

網頁中亦添加了需登入方能訪問的功能，像是下圖的”dashboard”，功能為顯示該使用者曾經申報過的紀錄。便於其管理或查看資料。

```
@app.route('/dashboard')
@login_required
def dashboard():
    # stray animals
    stray_alias = aliased(stray_ani)
    species_alias = aliased(speciesTable)
    color_alias = aliased(colorTable)
    region_alias = aliased(regionTable)
    city_alias = aliased(cityTable)
    breed_alias = aliased(breedTable)
```

以及登出功能

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('登出成功！', 'success')
    return redirect(url_for('index'))
```

- 下拉式選單
  因為選項繁多，且資料庫日後也可能持續更新，所以我們下拉式選單的選項並不是事先寫好再HTML檔中的，而是從資料庫query來的。
  方法大致上是，使用for 迴圈不斷抓取後端傳來的資料來新增為選項，直到資料都抓完為止。
  以page2 的species 選單為例

```
<div class="sel sel--black-panther">
    <select name="select-spieces" id="select-spieces" onchange="updateBreeds()">
        <option value="" disabled>物種</option>
        {% for species in species_names %}
            <option value="{{ species.species_id }}">{{ species.species_name }}</option>
        {% endfor %}
    </select>
</div>
```

- dashboard
  讓使用者在登入後可以查看自己過往繳交的資料
  將資料query出來後
  指定輸出mail = user_mail的資料

```
found = found.filter(found_alias.user_email == current_user.email)
request_found = found.all()
title = "過往提交紀錄"

return render_template('page7.html', request_stray = request_stray, request_found = request_found, current_user = current_user, title = title)
```

因為資料會時常更改，所以我們一樣將後端從資料庫提取到的資料傳送到前端，用for迴圈印出所有資料

```
<tbody>
    {% for stray in request_stray %}
        <tr>
            <td>走失</td>
            <td>{{ stray.species_name }}</td>
            <td>{{ stray.breed_name }}</td>
            <td>{{ stray.color_name }}</td>
            <td>{{ stray.region_name }}</td>
            <td>{{ stray.city_name }}</td>
            <td>{{ stray.date }}</td>
            <td>{{ stray.stray_gender }}</td>
            <td>{{ stray.solved }}</td>
        </tr>
    {% endfor %}
    {% for found in request_found %}
        <tr>
            <td>尋獲</td>
            <td>{{ found.species_name }}</td>
            <td>{{ found.breed_name }}</td>
            <td>{{ found.color_name }}</td>
            <td>{{ found.region_name }}</td>
            <td>{{ found.city_name }}</td>
            <td>{{ found.date }}</td>
            <td>{{ found.found_gender }}</td>
            <td>{{ found.solved }}</td>
        </tr>
    {% endfor %}
</tbody>
```