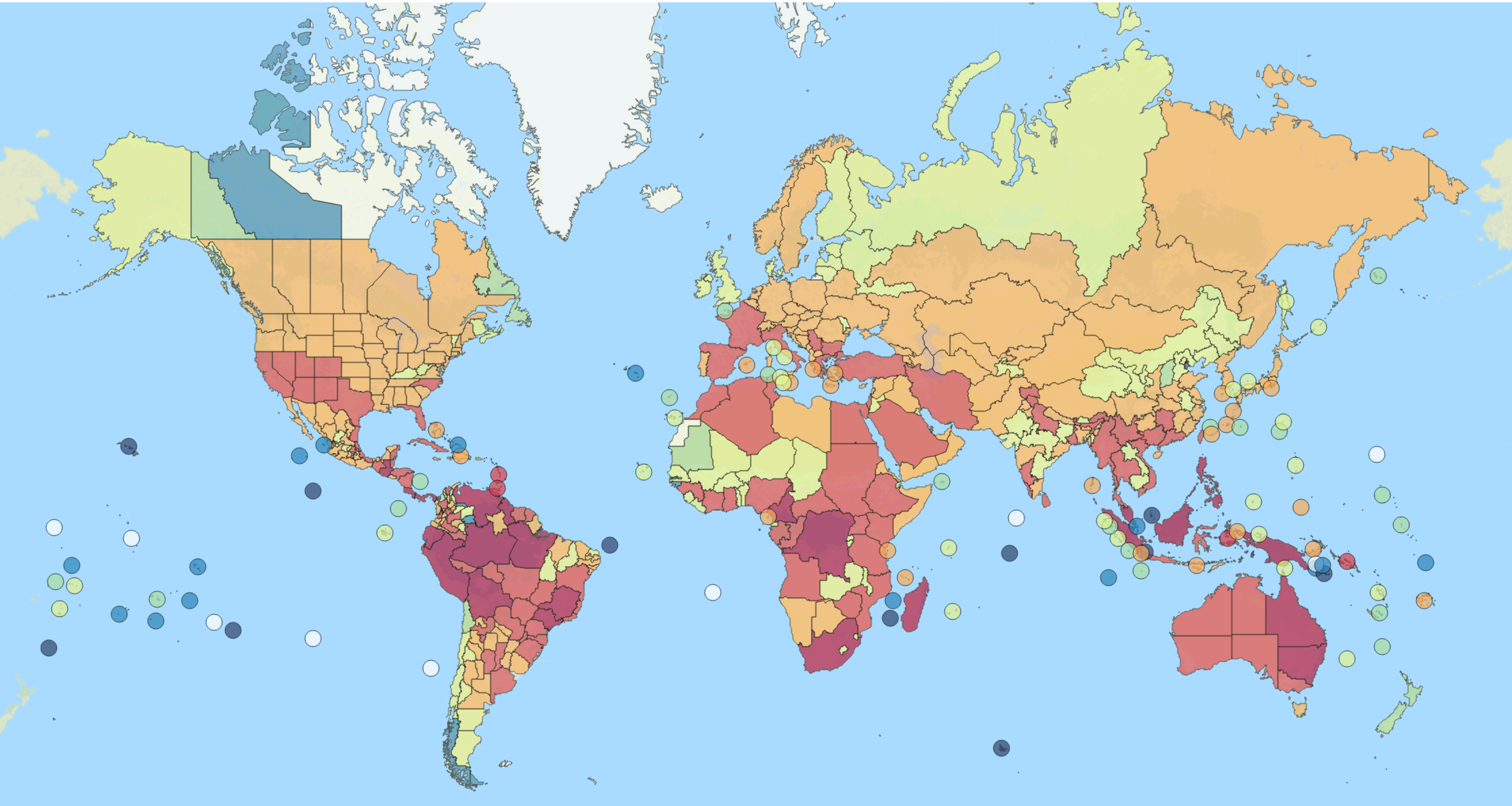
Visualizing the GABI database: antmaps.org

By **Julia Janicki**



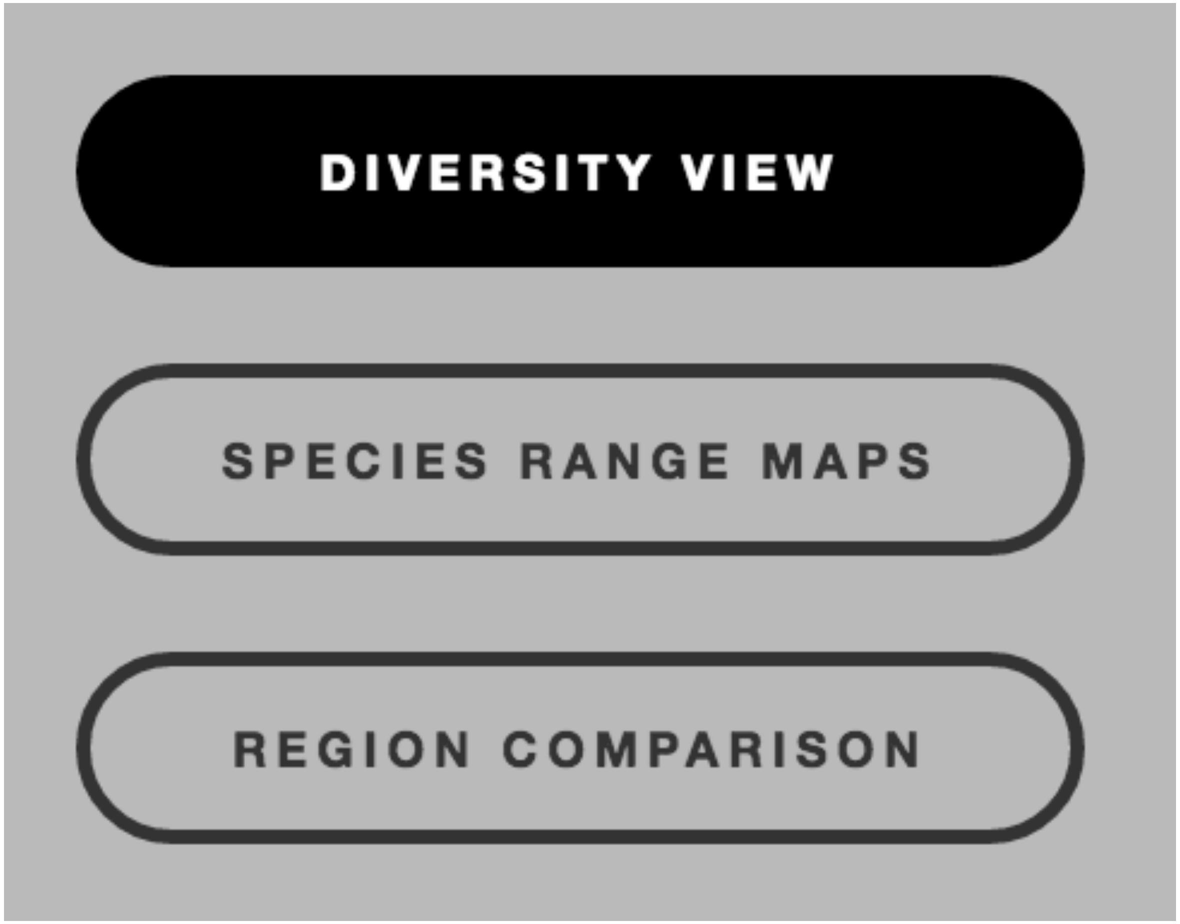
What is **antmaps**?

- Antmaps is a web application that is developed to visualize the biogeographic information available in the GABI database.

What is **GABI**

- Ants!
- The basic unit stored in the database is a **record**, or **species occurrences**
- Records are from museum collection data, online specimen databases and published literature
- There are over **1.7 million** records
- GABI is more of a **spatial dataset** and it doesn’t place as much emphasis on the temporal aspects of the data (i.e. it doesn’t include a date field) → so we decided to visualize it in the form of **maps**

How to structure antmaps in a useful way based on the GABI dataset



Structure of **GABI**

PostgreSQL

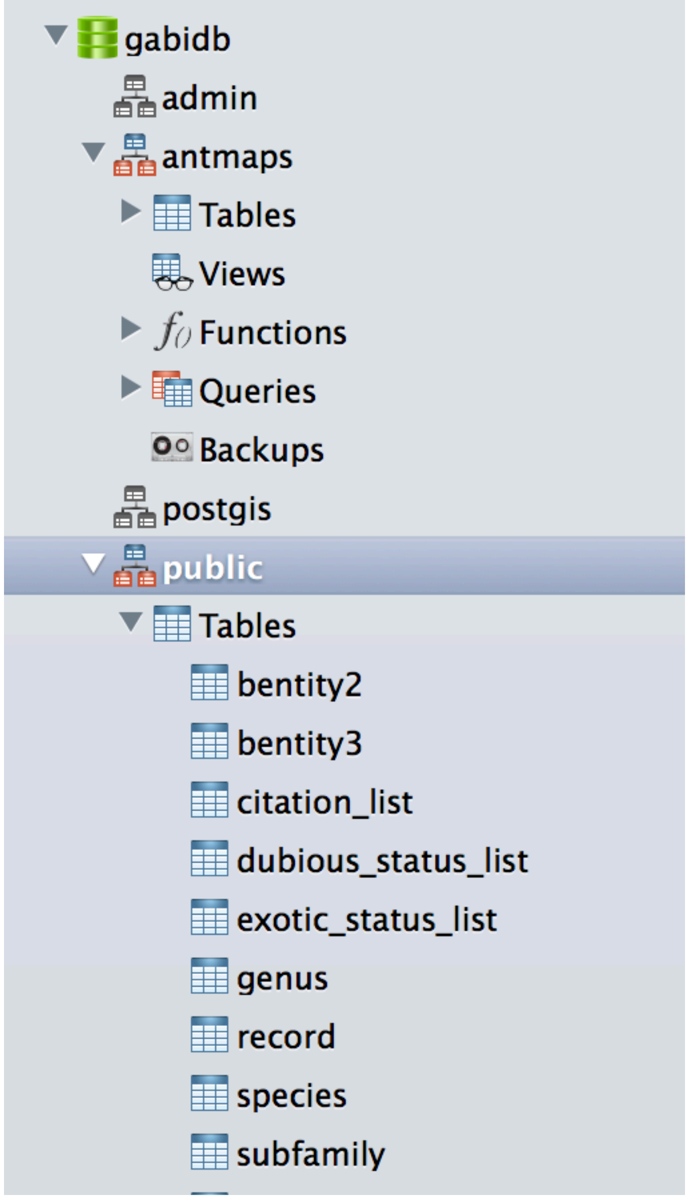
Database

↓

Schema

↓

Tables
Views
Materialized Views
Functions



- gabidb
  - admin
  - antmaps
    - Tables
    - Views
    - Functions
    - Queries
    - Backups
  - postgis
  - public
    - Tables
      - bentity2
      - bentity3
      - citation_list
      - dubious_status_list
      - exotic_status_list
      - genus
      - record
      - species
      - subfamily

Examples

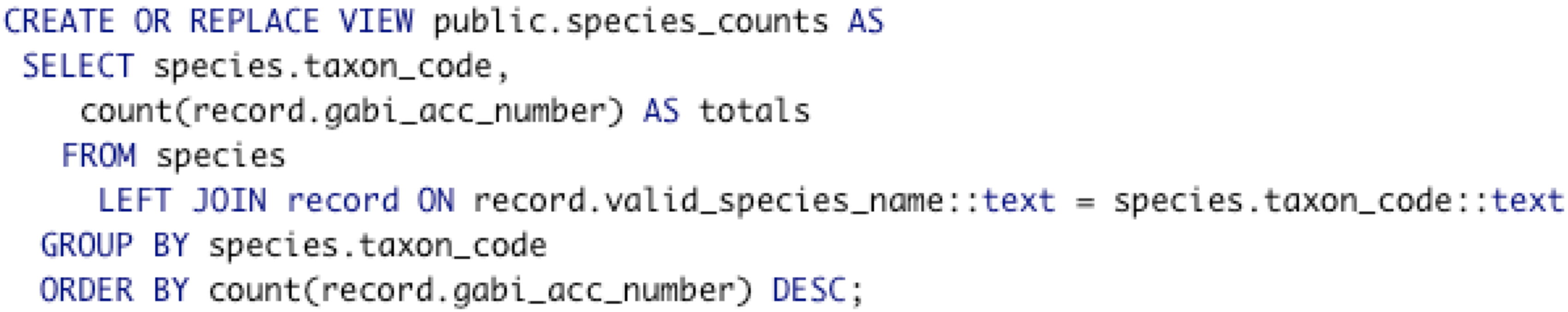
- Two main schemas in gabi db:
  - public
  - antmaps
- Major Tables
  - public.record
  - public.species
  - public.bentity2
- Views (example)
  - public.species_count
- Materialized Views (example)
  - antmaps.map_record

Materialized View

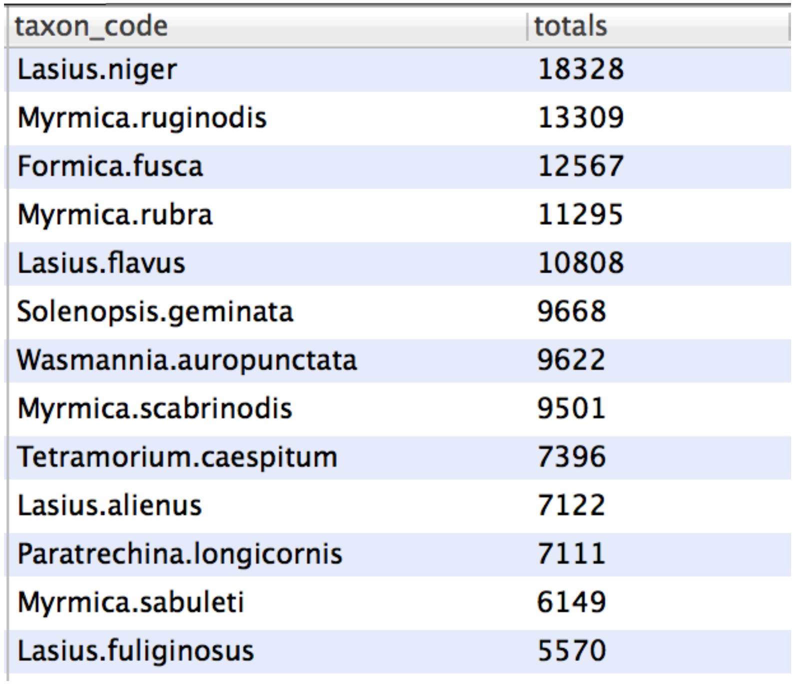
- Most of the GABI data that is being displayed on antmaps web application resides in the antmaps schema.
- We are using **materialized views** to allow for **faster performance**.
- Materialized views are database objects that contain the results of a query

Views

Example:



```
CREATE OR REPLACE VIEW public.species_counts AS
 SELECT species.taxon_code,
    count(record.gabi_acc_number) AS totals
   FROM species
     LEFT JOIN record ON record.valid_species_name::text = species.taxon_code::text
  GROUP BY species.taxon_code
  ORDER BY count(record.gabi_acc_number) DESC;
```



taxon_code	totals
Lasius.niger	18328
Myrmica.ruginodis	13309
Formica.fusca	12567
Myrmica.rubra	11295
Lasius.flavus	10808
Solenopsis.geminata	9668
Wasmannia.auropunctata	9622
Myrmica.scabrinodis	9501
Tetramorium.caespitum	7396
Lasius.alienus	7122
Paratrechina.longicornis	7111
Myrmica.sabuleti	6149
Lasius.fuliginosus	5570

antmaps.org overview

Uses a client–server architecture

- **Client-side (Frontend)**

  HTML
  CSS
  Javascript
  jQuery
  D3
  Leaflet

- **Server-side (Backend)**

  Django Framework
  PostgreSQL database

HTML

```
<!DOCTYPE HTML PUBLIC "-//
<html>
  <head>
    <meta name="TITLE" content
     <meta name="KEYWORDS" content
      <meta name="DESCRIPTION" con
       <link rel="stylesheet" type
        <script language="javasc
  </head>
  <body bgcolor="#ffff
                width="1
```

- **Client (Frontend)**

  HTML
  CSS
  Javascript
  jQuery
  D3
  Leaflet

- **Server (Backend)**

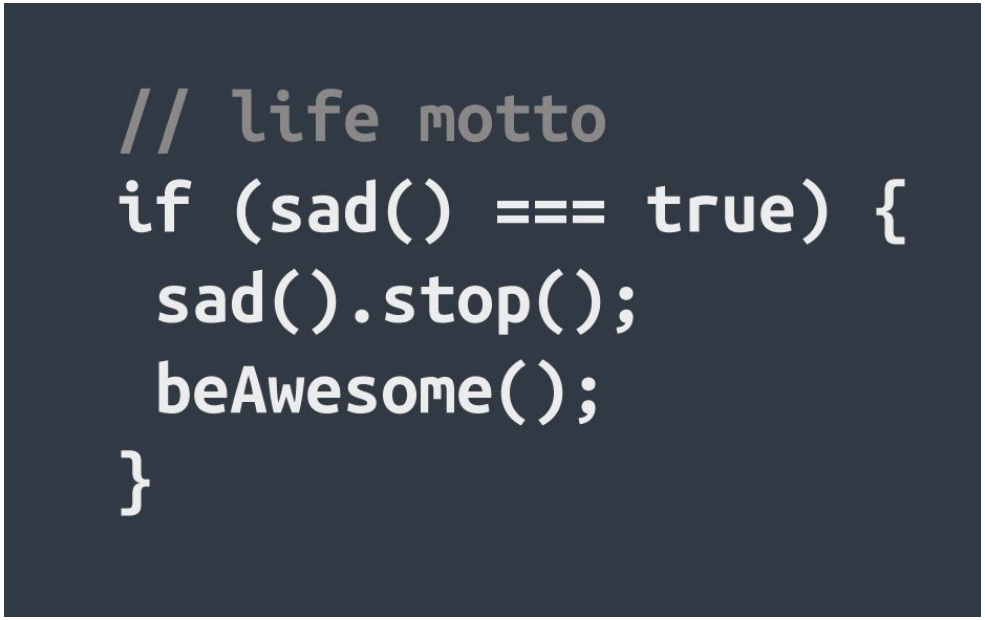
  Django Framework
  PostgreSQL database

CSS

```
body {
    font-family: Verdana, Helve
    font-size: 8.2pt;
    color: #3B3D3F;
}

element.style {
    position: absolu
    visibility:
    z-index:
    lef
```

- **Client (Frontend)**
  - HTML
  - CSS
  - Javascript
  - jQuery
  - D3
  - Leaflet
- **Server (Backend)**
  - Django Framework
  - PostgreSQL database

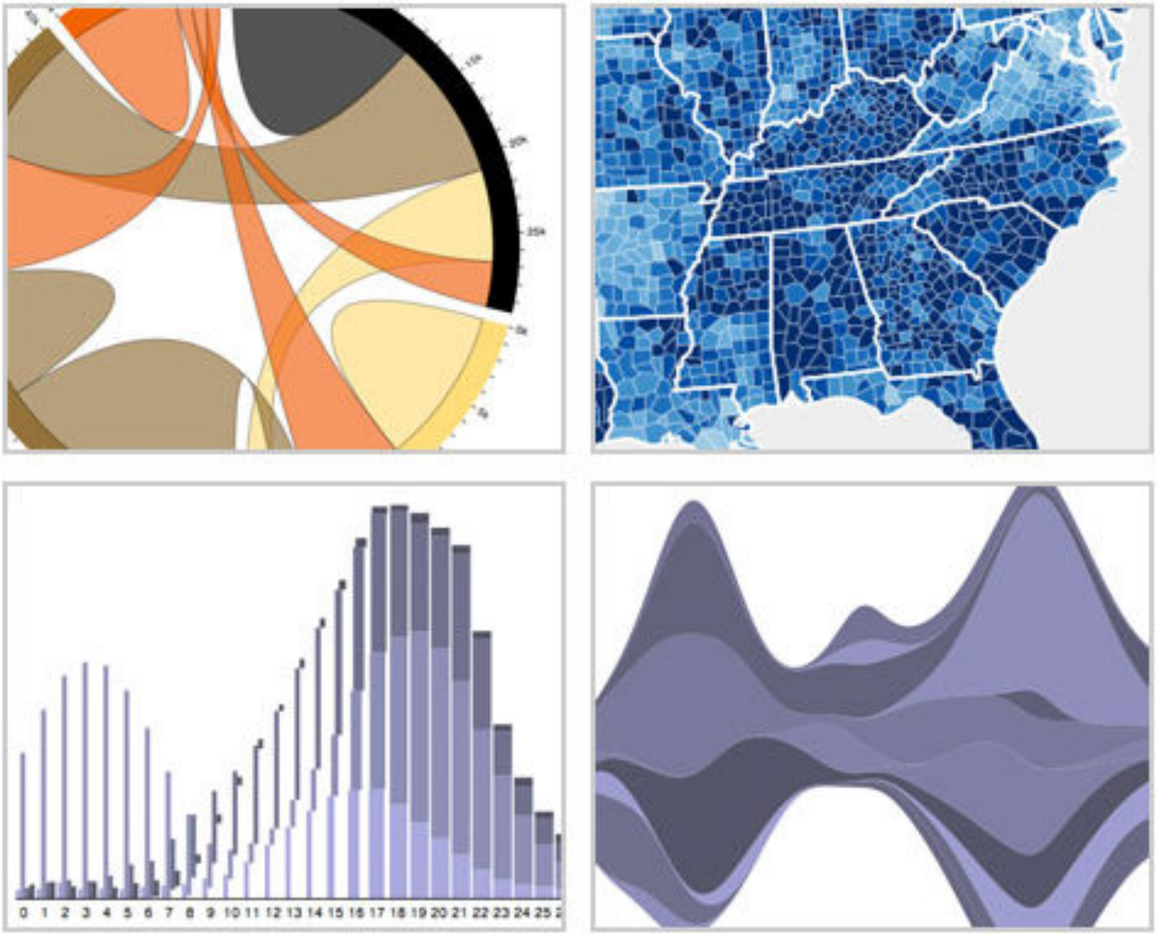
Javascript



```
// life motto
if (sad() === true) {
  sad().stop();
  beAwesome();
}
```

- **Client (Frontend)**
  - HTML
  - CSS
  - Javascript
  - jQuery
  - D3
  - Leaflet
- **Server (Backend)**
  - Django Framework
  - PostgreSQL database

JS Libraries



- **Client (Frontend)**

  HTML

  CSS

  Javascript

  jQuery

  D3

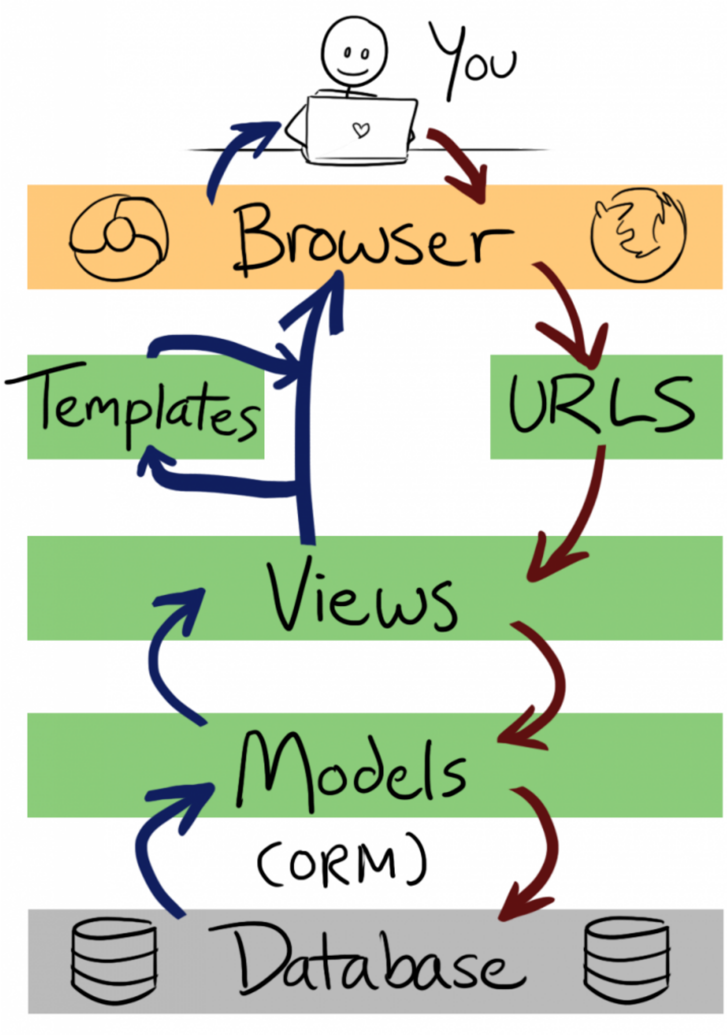
  Leaflet

- **Server (Backend)**

  Django Framework

  PostgreSQL database

Django

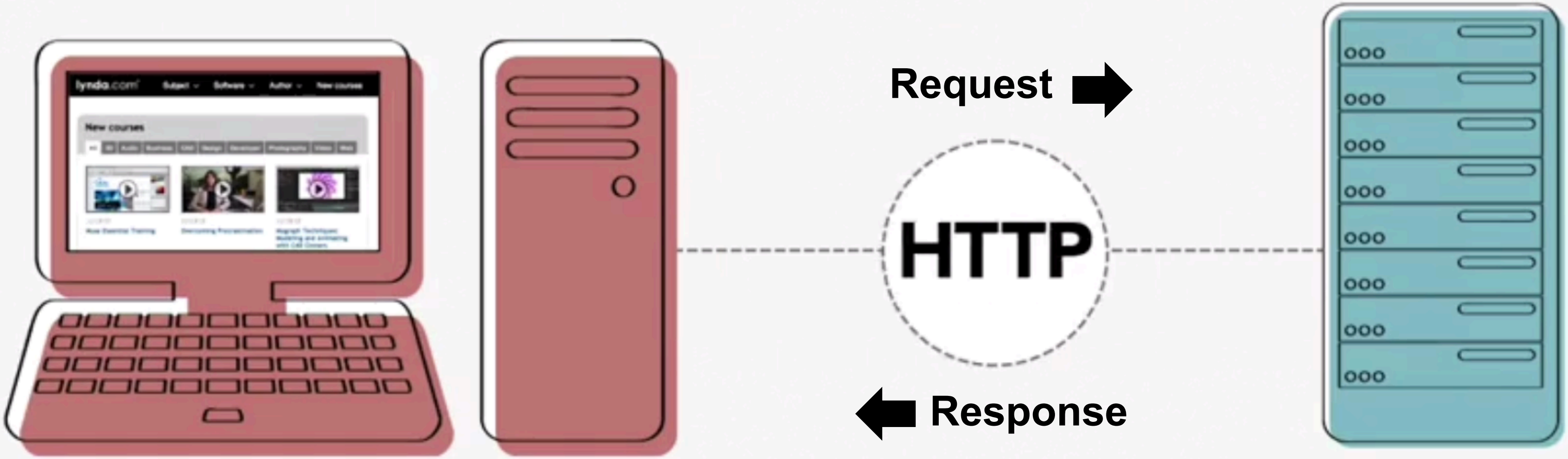


- **Client (Frontend)**
  - HTML
  - CSS
  - Javascript
  - jQuery
  - D3
  - Leaflet
- **Server (Backend)**
  - Django Framework
  - PostgreSQL database

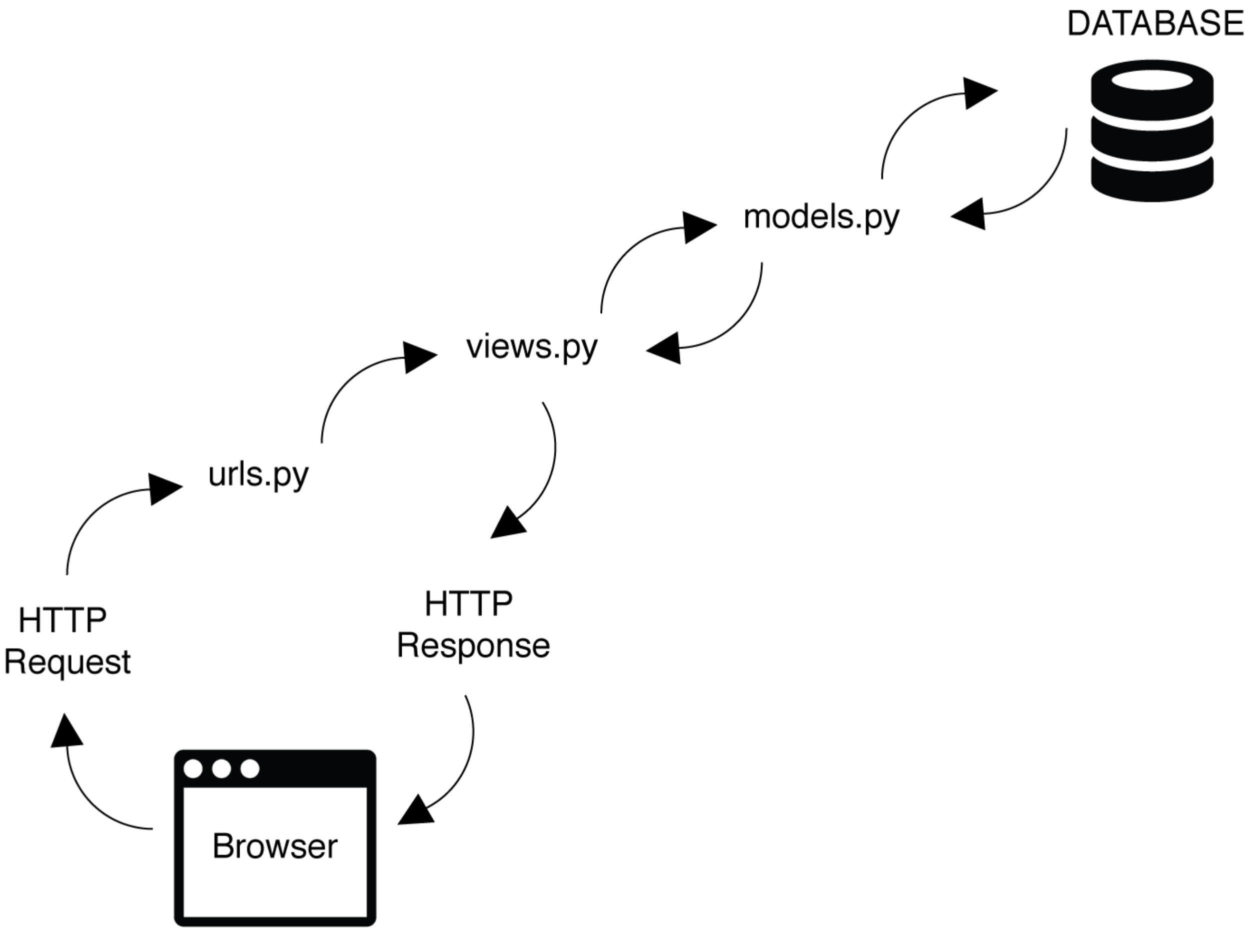
HTTP

- **Hyper Text Transfer Protocol**

  → Standard protocol to transfer resources on the web
- Functions in a Client-Server Request-Response method



HTTP cycle



DATABASE

models.py

views.py

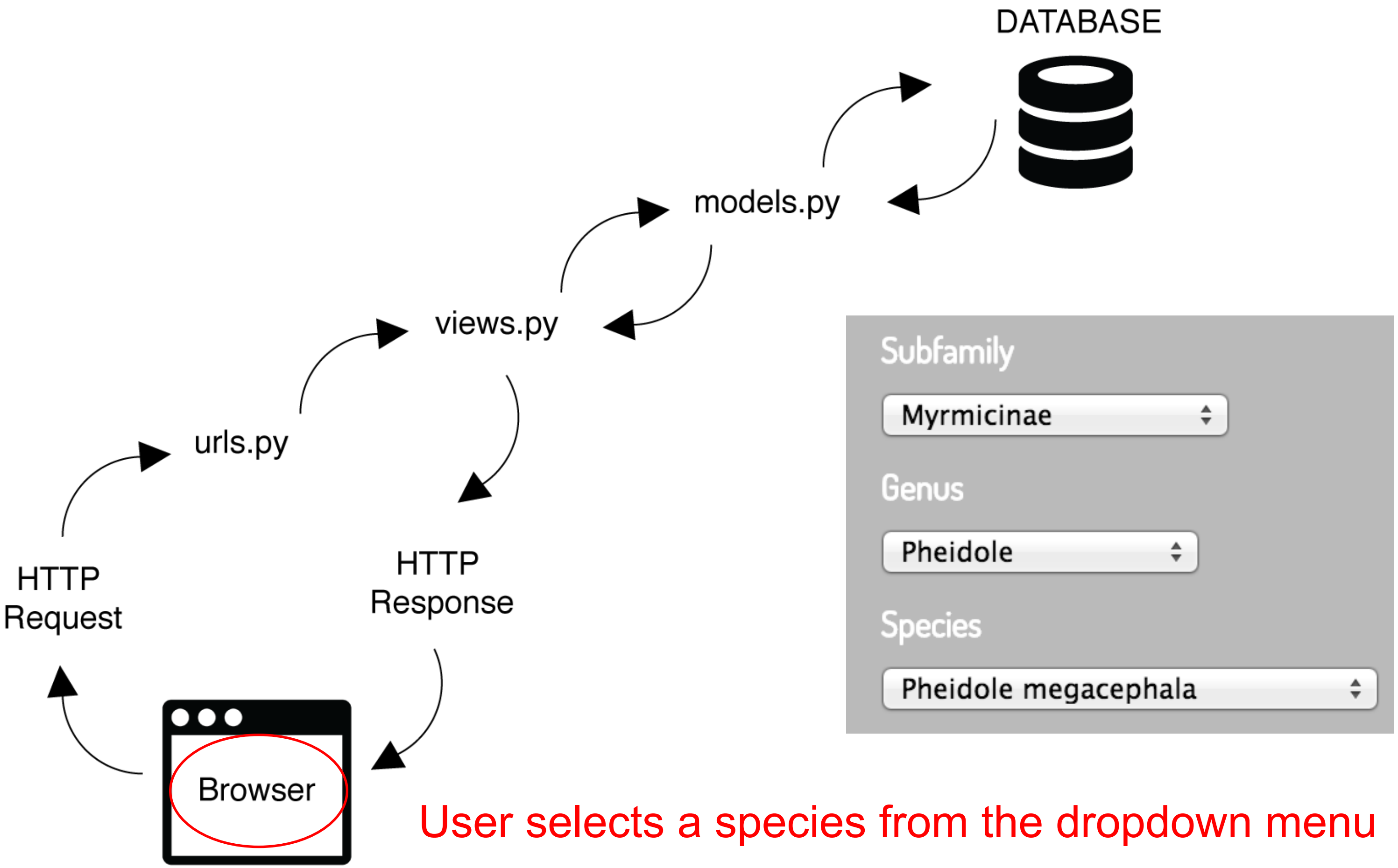
urls.py

HTTP
Request

HTTP
Response

Browser

HTTP cycle


DATABASE

models.py

views.py

urls.py

HTTP Request

HTTP Response

Browser

Subfamily

Myrmicinae

Genus

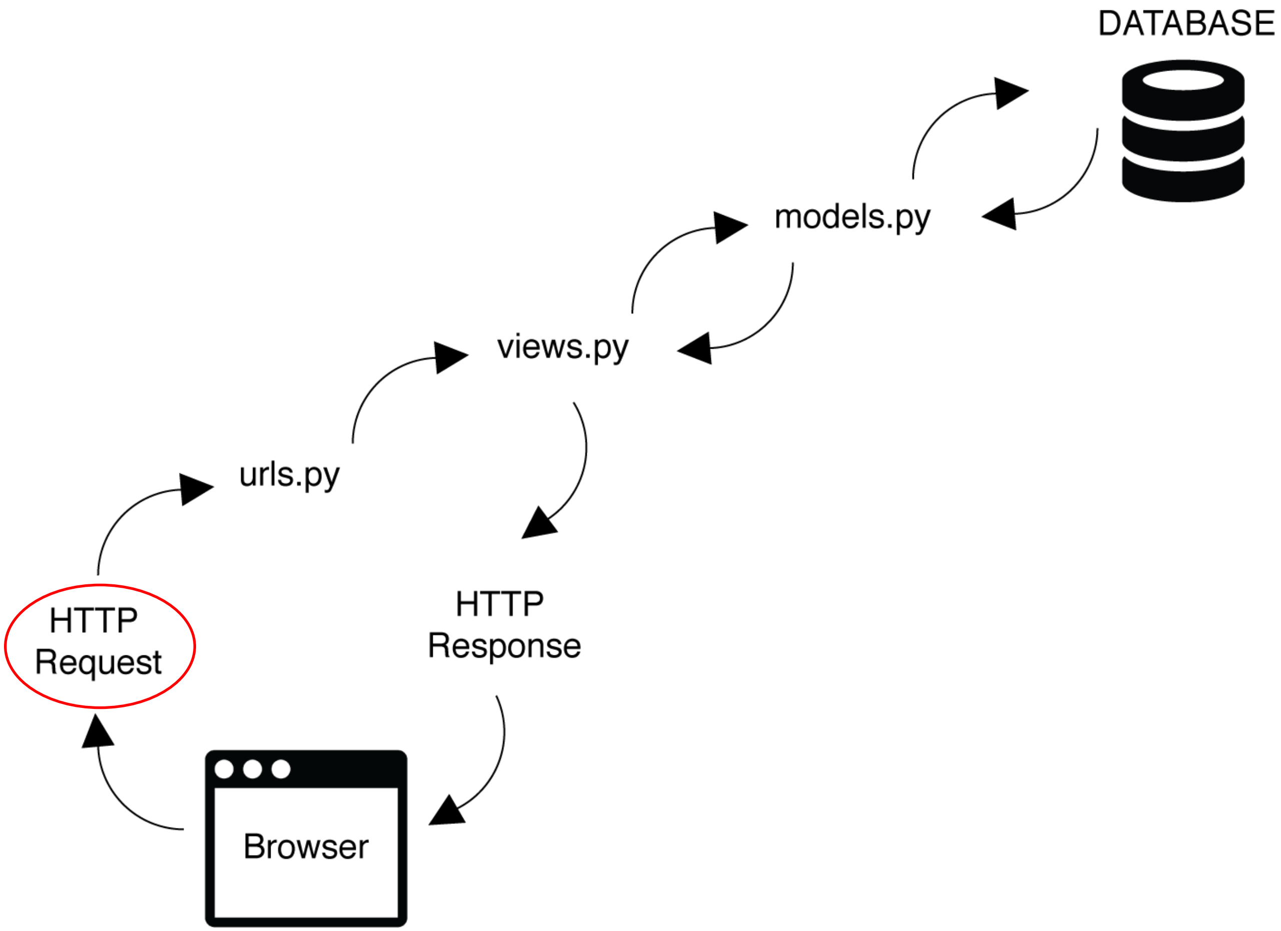
Pheidole

Species

Pheidole megacephala

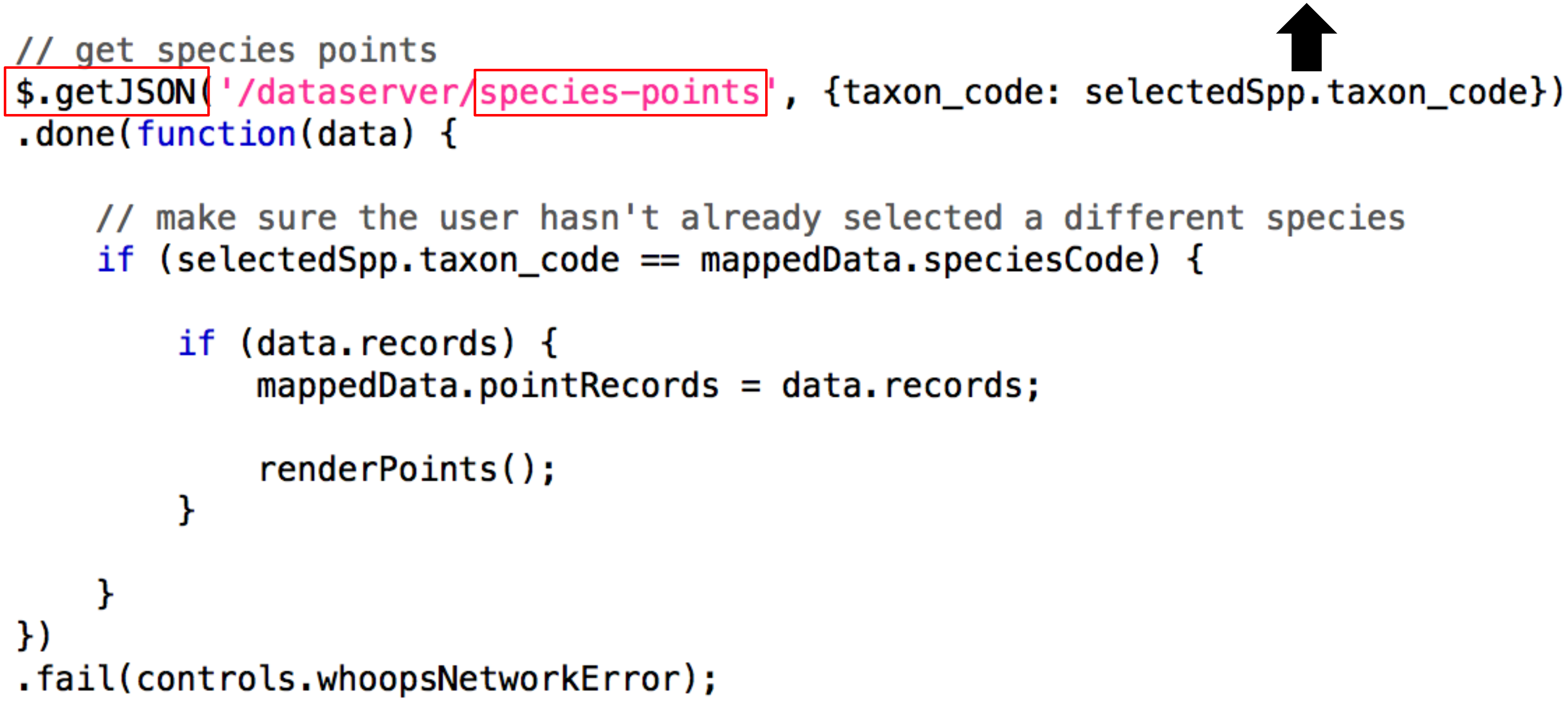
User selects a species from the dropdown menu

HTTP cycle


DATABASE
models.py
views.py
urls.py
HTTP Request
HTTP Response
Browser

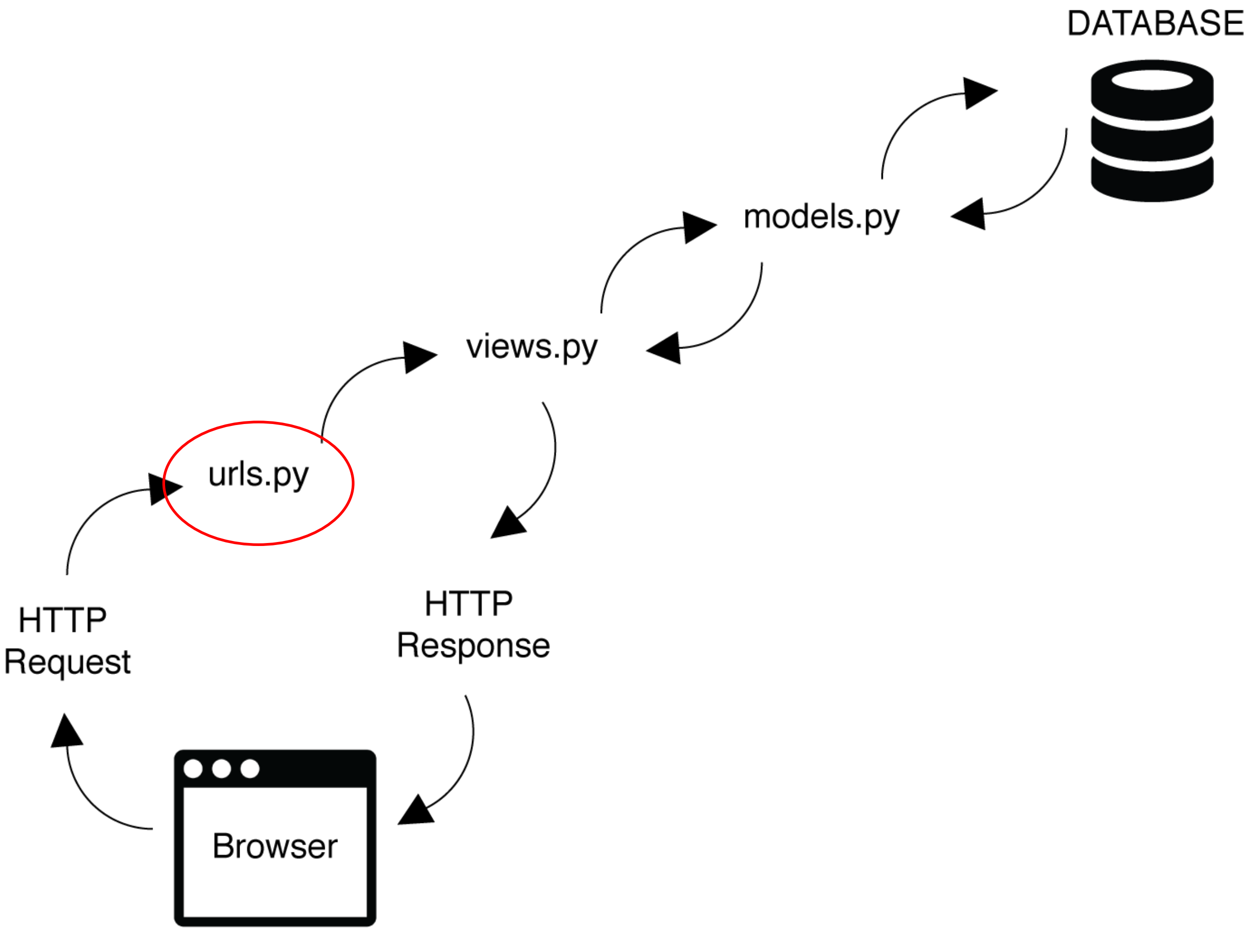
speciesMode.js

Pheidole.megacephala



```
// get species points
$.getJSON('/dataserver/species-points', {taxon_code: selectedSpp.taxon_code})
.done(function(data) {

    // make sure the user hasn't already selected a different species
    if (selectedSpp.taxon_code == mappedData.speciesCode) {

        if (data.records) {
            mappedData.pointRecords = data.records;

            renderPoints();
        }

    }
})
.fail(controls.whoopsNetworkError);
```

HTTP cycle


DATABASE

models.py

views.py

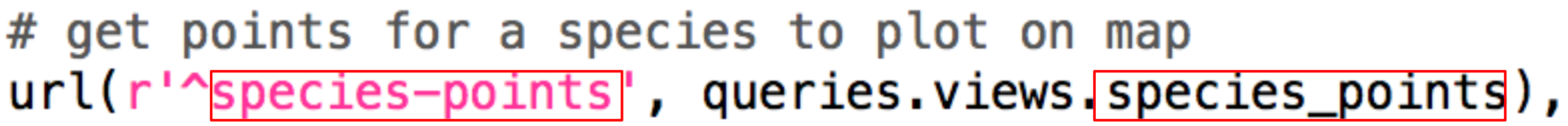
urls.py

HTTP
Request

HTTP
Response

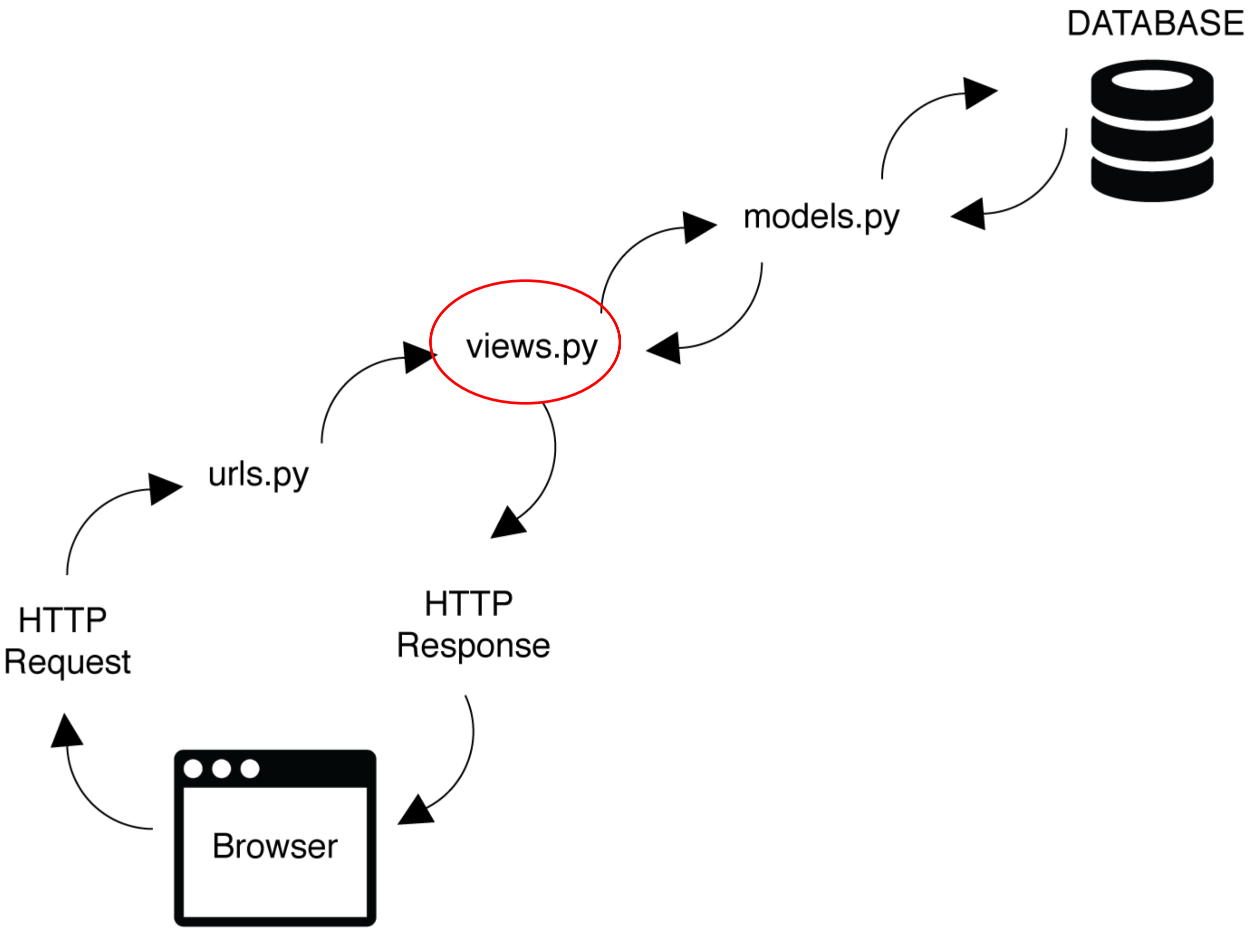
Browser

urls.py



```
# get points for a species to plot on map
url(r'^species-points', queries.views.species_points),
```

HTTP cycle


DATABASE

models.py

views.py

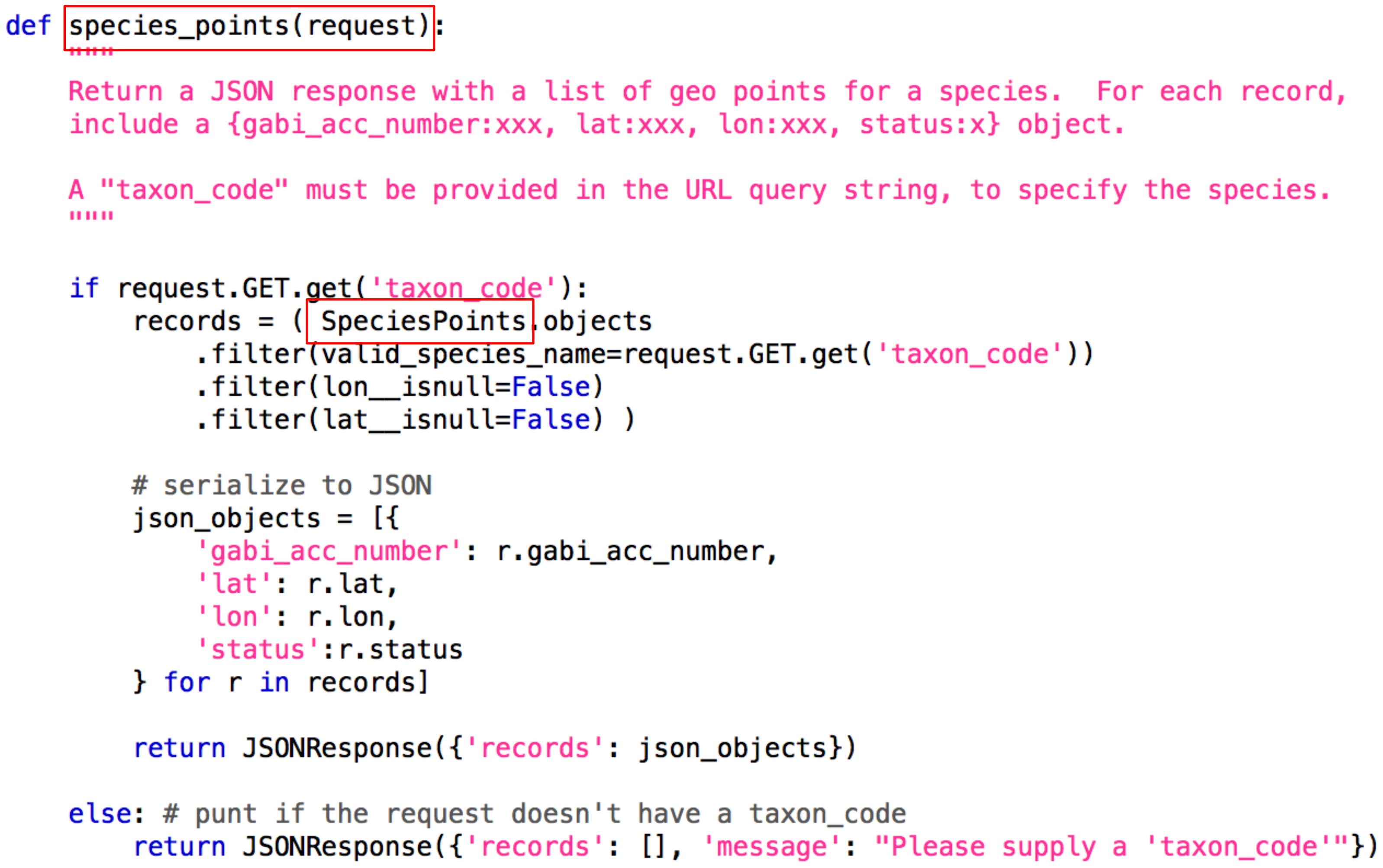
urls.py

HTTP
Request

HTTP
Response

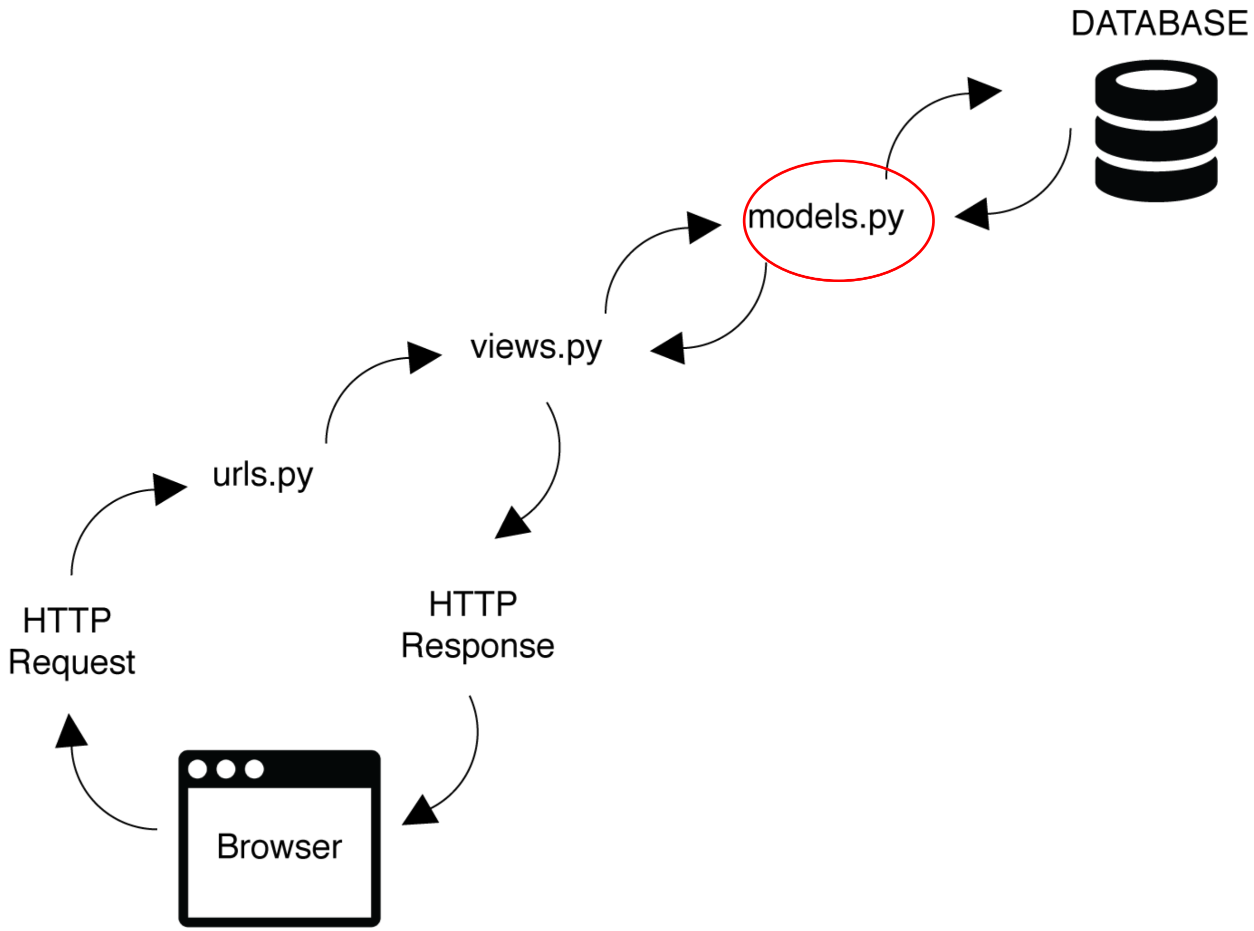
Browser

views.py



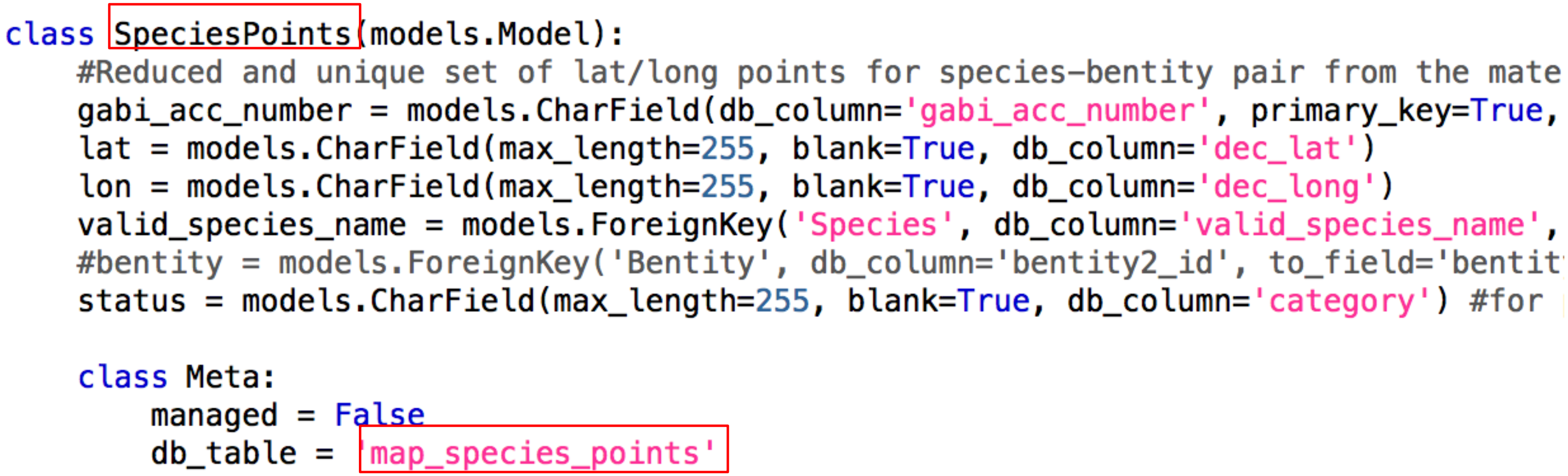
```
def species_points(request):
    """
    Return a JSON response with a list of geo points for a species.  For each record,
    include a {gabi_acc_number:xxx, lat:xxx, lon:xxx, status:x} object.

    A "taxon_code" must be provided in the URL query string, to specify the species.
    """

    if request.GET.get('taxon_code'):
        records = ( SpeciesPoints.objects
            .filter(valid_species_name=request.GET.get('taxon_code'))
            .filter(lon__isnull=False)
            .filter(lat__isnull=False) )

        # serialize to JSON
        json_objects = [{
            'gabi_acc_number': r.gabi_acc_number,
            'lat': r.lat,
            'lon': r.lon,
            'status':r.status
        } for r in records]

        return JSONResponse({'records': json_objects})

    else: # punt if the request doesn't have a taxon_code
        return JSONResponse({'records': [], 'message': "Please supply a 'taxon_code'"})
```

HTTP cycle



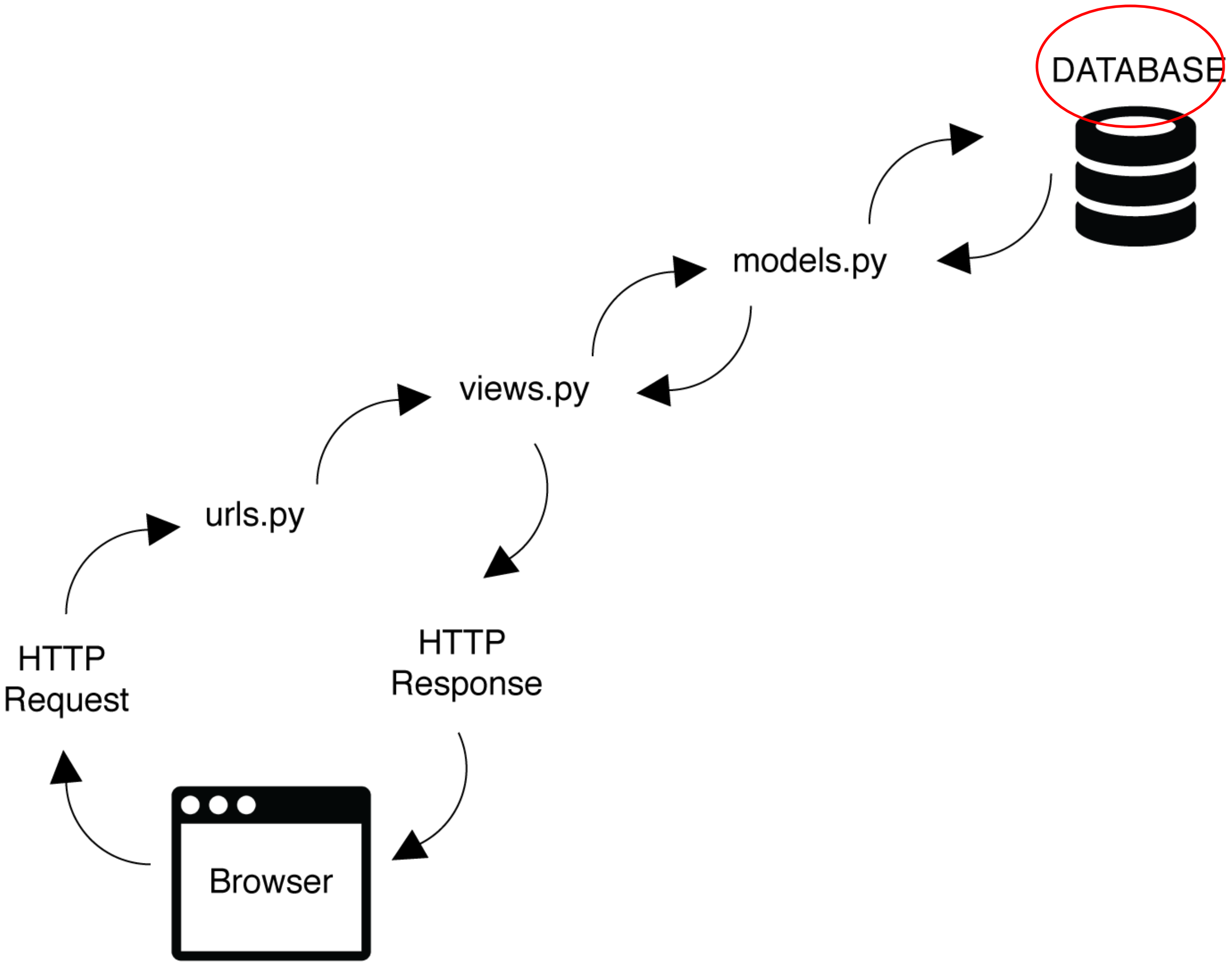
DATABASE
models.py
views.py
urls.py
HTTP Request
HTTP Response
Browser

models.py



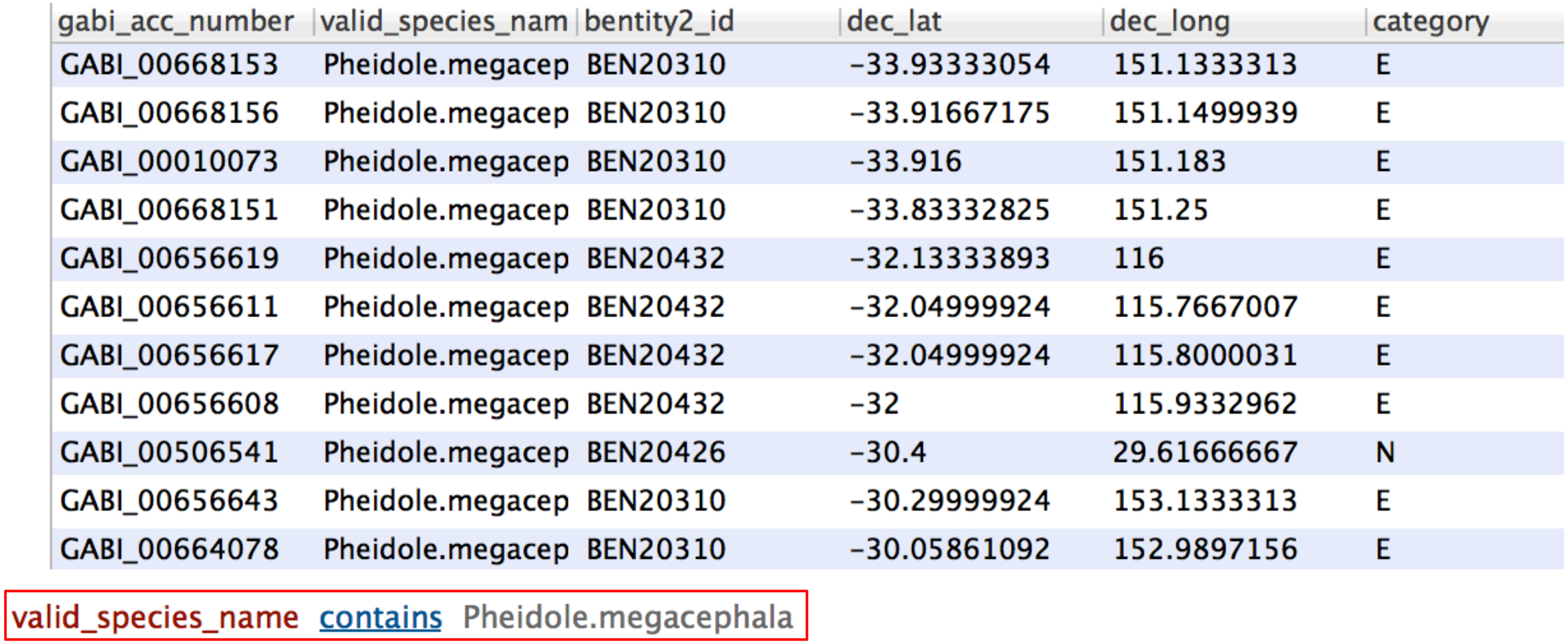
```
class SpeciesPoints(models.Model):
    #Reduced and unique set of lat/long points for species-bentity pair from the mate
    gabi_acc_number = models.CharField(db_column='gabi_acc_number', primary_key=True,
    lat = models.CharField(max_length=255, blank=True, db_column='dec_lat')
    lon = models.CharField(max_length=255, blank=True, db_column='dec_long')
    valid_species_name = models.ForeignKey('Species', db_column='valid_species_name',
    #bentity = models.ForeignKey('Bentity', db_column='bentity2_id', to_field='bentit
    status = models.CharField(max_length=255, blank=True, db_column='category') #for

    class Meta:
        managed = False
        db_table = 'map_species_points'
```

HTTP cycle


DATABASE
models.py
views.py
urls.py
HTTP
Request
HTTP
Response
Browser

map_species_points materialized view

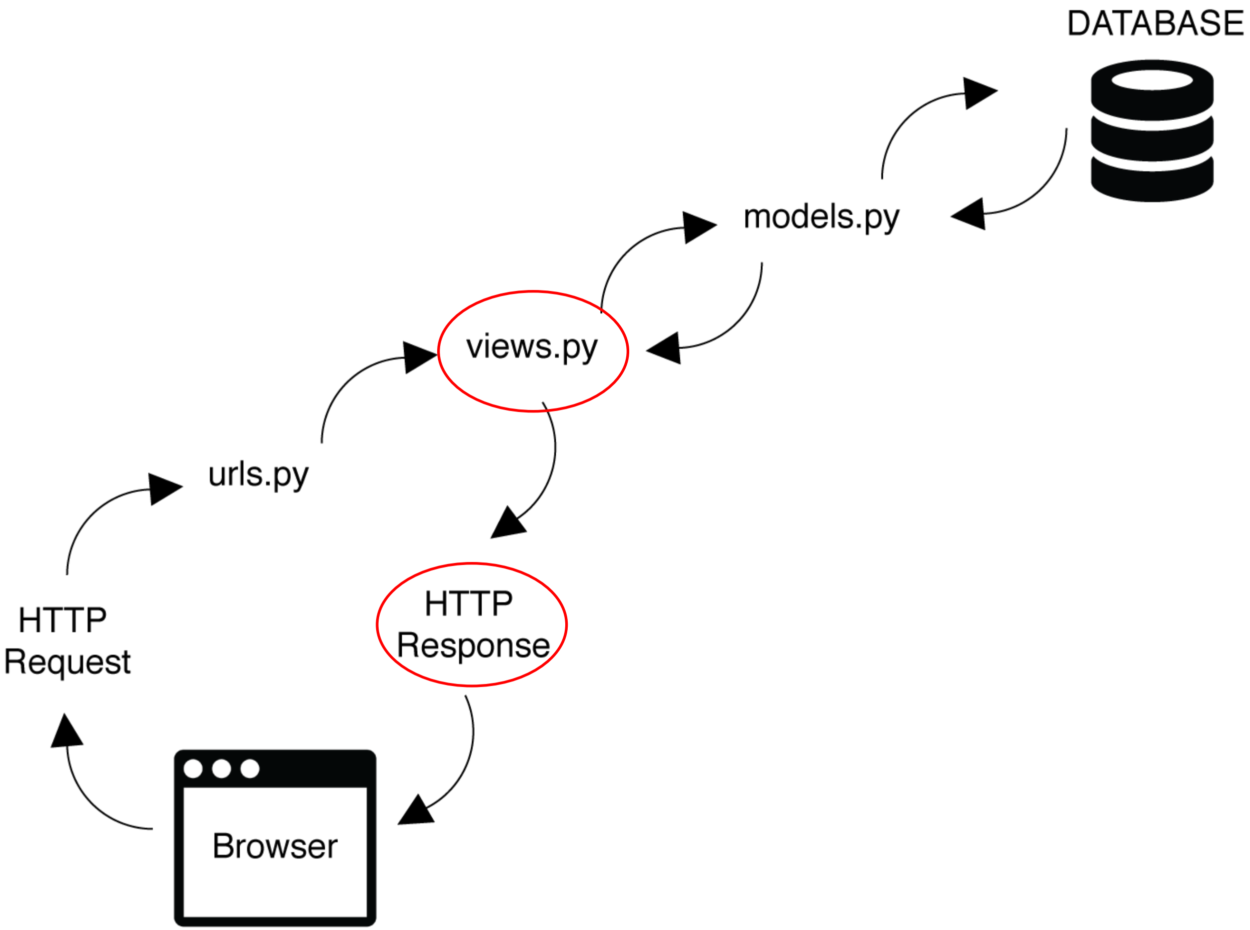
```
CREATE MATERIALIZED VIEW antmaps.map_species_points AS
SELECT aggregate_random_gabi_acc(map_record.valid_species_name, map_record.bentity2_id,
   map_record.dec_lat, map_record.dec_long) AS gabi_acc_number,
   map_record.valid_species_name,
   map_record.bentity2_id,
   map_record.dec_lat,
   map_record.dec_long,
   aggregate_antmaps_category(map_record.valid_species_name, map_record.bentity2_id) AS category
  FROM map_record
 WHERE map_record.dec_lat IS NOT NULL
 GROUP BY map_record.valid_species_name, map_record.dec_lat, map_record.dec_long, map_record.bentity2_id;
```



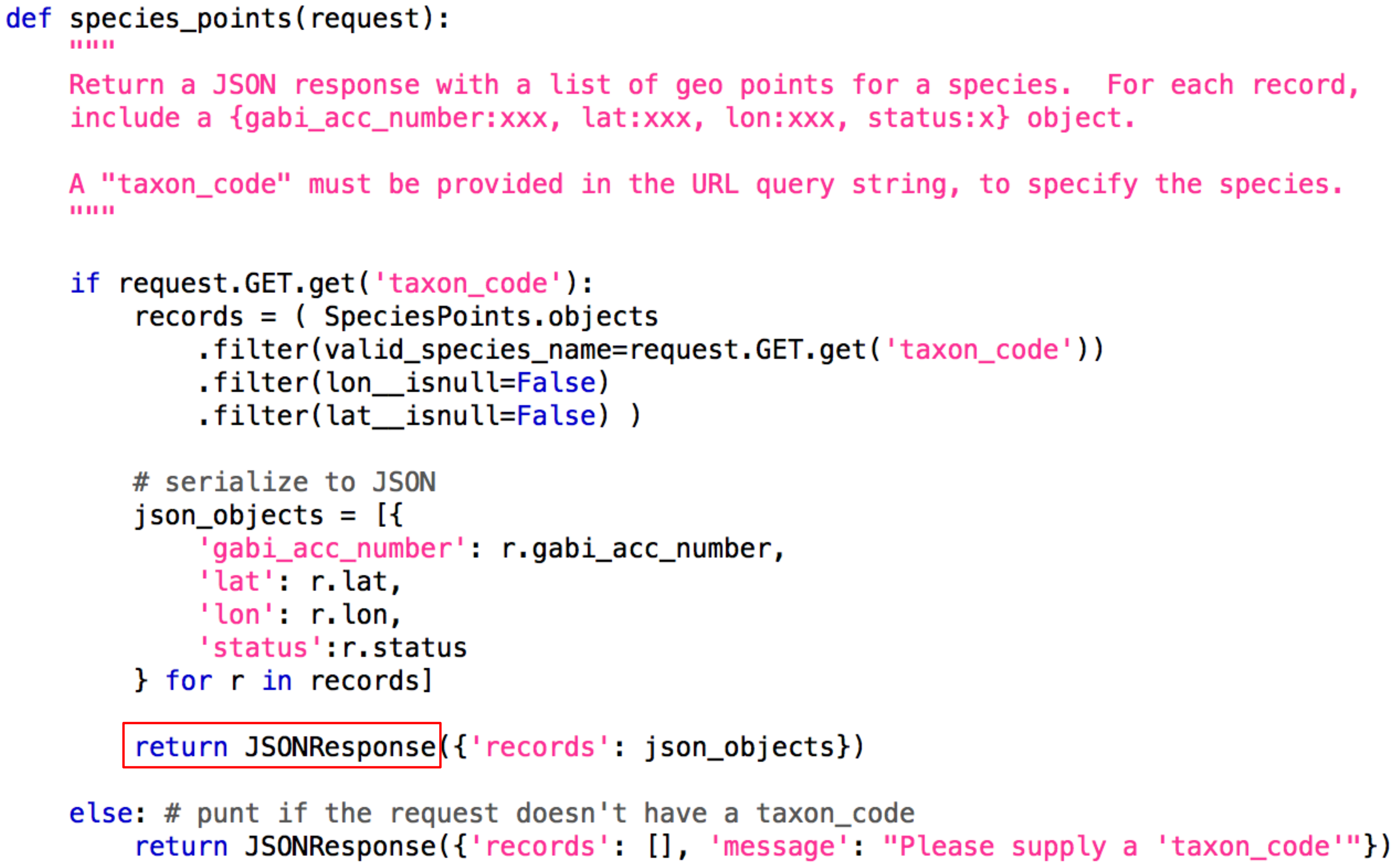
gabi_acc_number	valid_species_nam	bentity2_id	dec_lat	dec_long	category
GABI_00668153	Pheidole.megacep	BEN20310	-33.93333054	151.1333313	E
GABI_00668156	Pheidole.megacep	BEN20310	-33.91667175	151.1499939	E
GABI_00010073	Pheidole.megacep	BEN20310	-33.916	151.183	E
GABI_00668151	Pheidole.megacep	BEN20310	-33.83332825	151.25	E
GABI_00656619	Pheidole.megacep	BEN20432	-32.13333893	116	E
GABI_00656611	Pheidole.megacep	BEN20432	-32.04999924	115.7667007	E
GABI_00656617	Pheidole.megacep	BEN20432	-32.04999924	115.8000031	E
GABI_00656608	Pheidole.megacep	BEN20432	-32	115.9332962	E
GABI_00506541	Pheidole.megacep	BEN20426	-30.4	29.61666667	N
GABI_00656643	Pheidole.megacep	BEN20310	-30.29999924	153.1333313	E
GABI_00664078	Pheidole.megacep	BEN20310	-30.05861092	152.9897156	E

valid_species_name contains Pheidole.megacephala

HTTP cycle

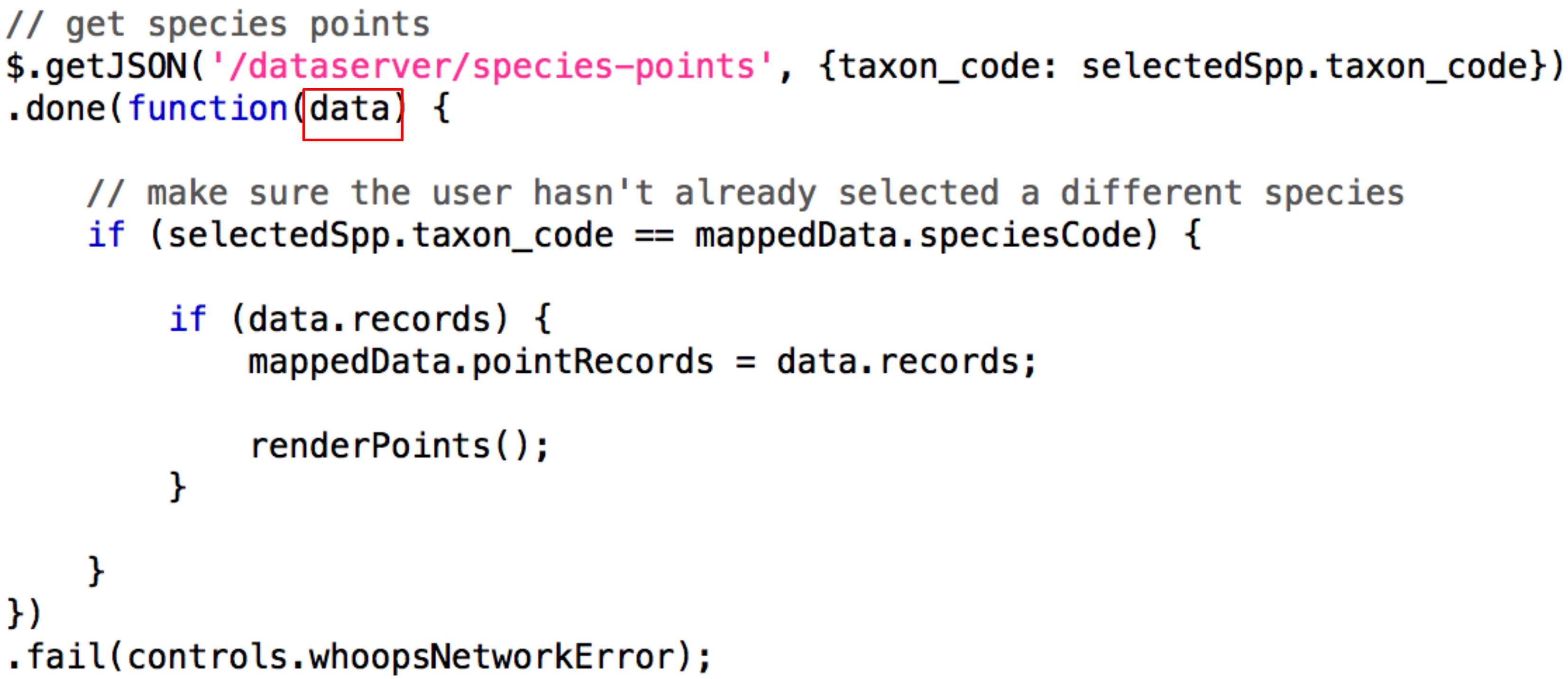

DATABASE
models.py
views.py
urls.py
HTTP Request
HTTP Response
Browser

views.py

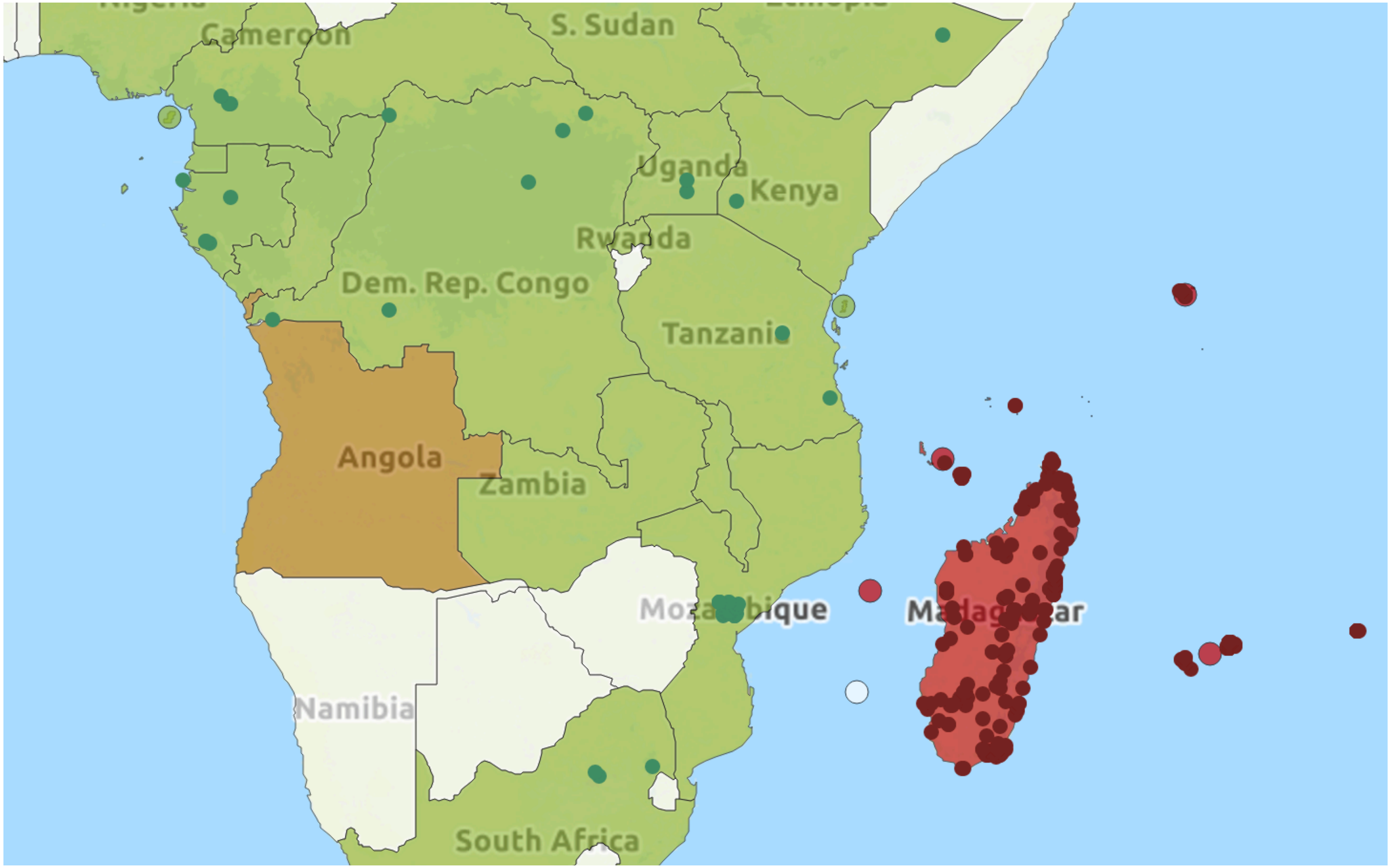


```
def species_points(request):
    """
    Return a JSON response with a list of geo points for a species.  For each record,
    include a {gabi_acc_number:xxx, lat:xxx, lon:xxx, status:x} object.

    A "taxon_code" must be provided in the URL query string, to specify the species.
    """

    if request.GET.get('taxon_code'):
        records = ( SpeciesPoints.objects
            .filter(valid_species_name=request.GET.get('taxon_code'))
            .filter(lon__isnull=False)
            .filter(lat__isnull=False) )

        # serialize to JSON
        json_objects = [{
            'gabi_acc_number': r.gabi_acc_number,
            'lat': r.lat,
            'lon': r.lon,
            'status':r.status
        } for r in records]

        return JSONResponse({'records': json_objects})

    else: # punt if the request doesn't have a taxon_code
        return JSONResponse({'records': [], 'message': "Please supply a 'taxon_code'"})
```

speciesMode.js



```
// get species points
$.getJSON('/dataserver/species-points', {taxon_code: selectedSpp.taxon_code})
.done(function(data) {

    // make sure the user hasn't already selected a different species
    if (selectedSpp.taxon_code == mappedData.speciesCode) {

        if (data.records) {
            mappedData.pointRecords = data.records;

            renderPoints();
        }

    }
})
.fail(controls.whoopsNetworkError);
```


Cameroon
S. Sudan
Uganda
Kenya
Rwanda
Dem. Rep. Congo
Tanzania
Angola
Zambia
Mozambique
Madagascar
Namibia
South Africa

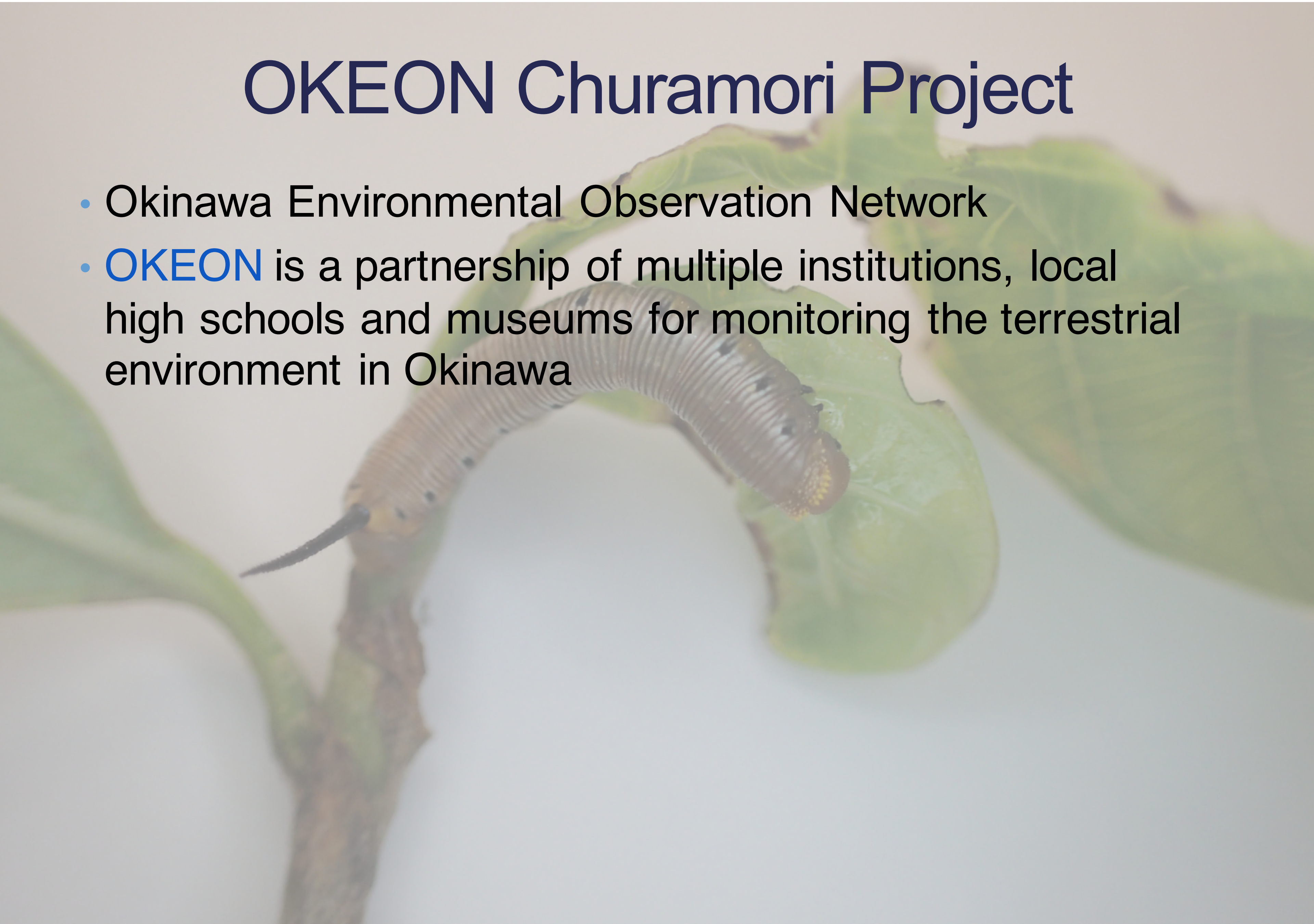
Performance

- Materialized Views
- Indexes

Other databases in our lab

- OKEON database
- Specimen database

OKEON Churamori Project

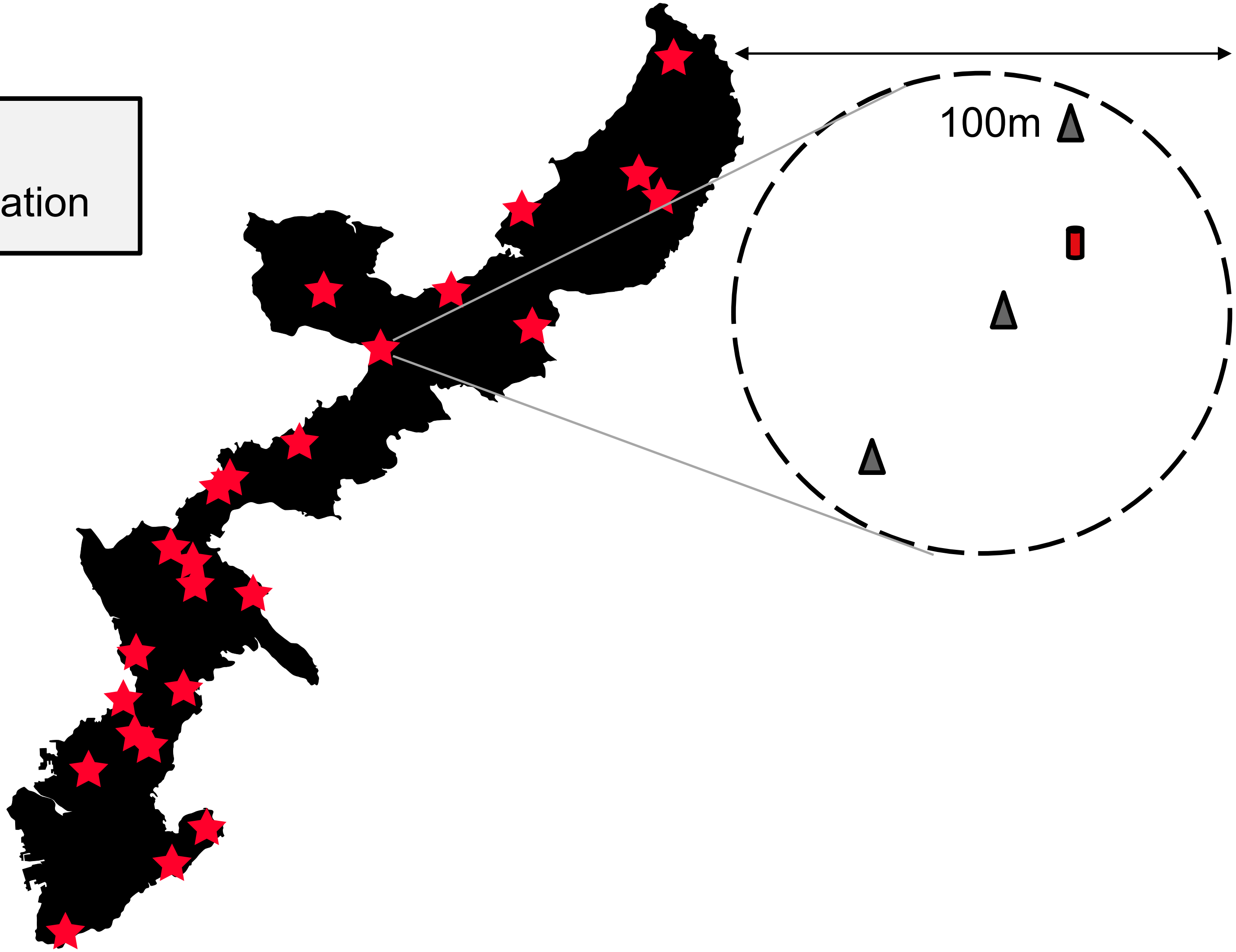


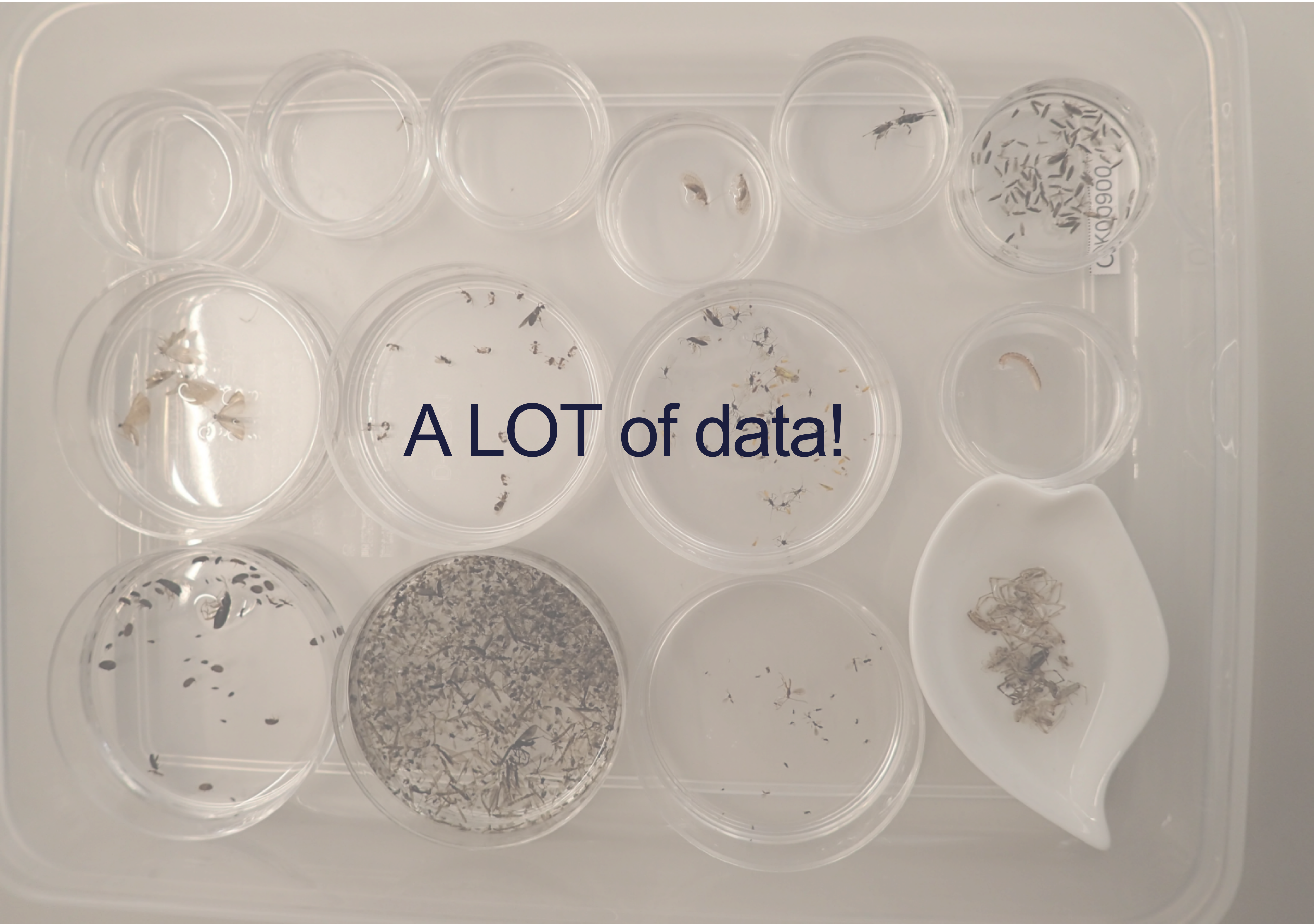
- Okinawa Environmental Observation Network
- OKEON is a partnership of multiple institutions, local high schools and museums for monitoring the terrestrial environment in Okinawa

Field Network

▲ SLAM trap

🟥 Weather station





A LOT of data!

- 24 sites
- 72 traps
  - change every two weeks (each is a **collection event**)
  - Sort into **lots** (first pass is sorted into major insect orders)
- 24 weather stations (collect every 15 minutes)

site → locality → trap → collection_event → lot → specimen