ARBOVERSE Documentation

Creating Data Models

Creating new data models is a straightforward process. They are defined in models.py. Specifically, you will want to look at arboverse\_updated/models.py. There are other models defined for admin functions and utilities which likely will not need to be edited at any time.

To create another model, simply edit the models.py file by adding one matching the formatting there. Define a python class object with a name and type models.Model. This is a built-in Django method for use with tables in PGSQL or other relational databases.

There are many data types for columns defined in the models. A list of them can be found here: <https://docs.djangoproject.com/en/5.2/ref/models/fields/>

Most of the columns throughout the project were defined as TextField or ForeignKey. It is likely that this will be the case for future entries. Foreign keys are briefly described in the Editing Data Models section.

When data models are created, you need to enter the terminal and run the following commands to update the models used by the site:

python manage.py makemigrations

python manage.py migrate

Additionally, you need to make edits to /arboverse\_updated/admin.py to add a new line for an added model to make it API accessible.

Ex: admin.site.register(ModelName, ExportCsvAdmin)

Editing Data Models

Django utilized data models to define class objects based on the data entered into your PGSQL table. These need to match the formatting of the table in order for the website to correctly pull data from the table. A mismatch between the two will result in broken site functionality.

Firstly, I will introduce the data models which are defined as class objects in python. The following is one example within the ARBOVERSE project:  
  
A computer screen shot of a program code

AI-generated content may be incorrect.

This class utilized the models.Model structure within the Django models framework to define a data model based on the PGSQL table arboverse\_updated\_vectorsubfamily. The table consists of two columns:

“name” refers to the name of the sub family.

“family” is a foreign key ID which refers to another table in the database arboverse\_updated\_vectorfamily. This foreign key ID maps to a row in the aforementioned table and can be used to get the resulting family name mapped to the subfamily listed in the name column.

A foreign key is defined as a reference to the id number of a row in another table in the database. Instead of including duplicated data, we reference the row since there may be many subfamilies to a single family.

This class example was chosen as it well outlines the complexity of creating good data models. Since many of the tables in the ARBOVERSE database are written to use foreign key maps, we need to ensure that not only is the current table defined and matching our intended specifications, but that any tables on which it depends match as well.

Any of the column headers or “keys” are fixed values determined when the table is created. These can be changed, but ensure that any changed values are updated in the data model. Anywhere that those values may be used to collect or utilize data from the table may need updated as well. Most of the usages of these values will be found in arboverse\_updated/templates/pages/arboverse.html

Note: If you are using PyCharm from JetBrains as your IDE, you can use the CTRL+SHIFT+F macro to open a project-wide search tool. Make sure you select the project tab to search the entire project, and then include the value that you have updated. This should outline any locations which are referencing that value and can be updated to match your new spec. It is likely other IDEs have similar functionality, but I can confirm that this method is great for tracking usages across the project.

When data models are edited, you need to enter the terminal and run the following commands to update the models used by the site:

python manage.py makemigrations

python manage.py migrate

Populating New or Edited Tables

The script schema\_populator.py is used to read in data from the excel/csv documents that were initially provided. These assume that the format of any documents used to populate the tables will match those. The script will need to be edited if the format of the excel or CSV documents change.

When new data models are added, or their keys have been edited, you need to ensure that the matching class in schema\_populator.py is edited or created. If not, no data will be added to those classes.

Schema\_populator.py uses sqlalchemy to interact with the PGSQL database the site depends on. This means that the class objects defined here are similar but different from the ones outlined in the models.py file which has been discussed above. Follow the examples outlined in the file to create new classes for tables whose data need to be populated.

Downloading CSVs from Admin

When a user with Admin access logs into the admin portal on ARBOVERSE, they can download a CSV document which contains all the data in the PGSQL database. If you want to export portions of tables, you can enter list view by clicking on one of the tables. Select the rows you are interested in, and then in the action dropdown menu you can select download csv. Pressing the Go button will download the CSV.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Uploading CSVs from Admin

If a user with Admin access wants to add new data by uploading a CSV, they can do so from the admin portal on the ARBOVERSE site. The CSV will need to match the formatting expected by the schema\_populator.py file in order to read the data into the tables.

This can be updated by editing the insert\_\* functions outlined in schema\_poulator.py, although they are complicated. It is likely easier to continue on using the existing format than to change that code to meet a different CSV.

The original files provided are included in the repo and should be used as the standard for formatting.

Virus data is collected from “The global distribution of arbovirus diversity – Official.xlsx” on the sheet named “main\_arbovirus”.

Main Vector data is pulled from the same file but found on the sheet named “main\_vector”.

Vector data is pulled from “Arbovector\_database.csv”.

There are columns which are unused in the application and can be ignored. The schema\_populator.py can be considered as an exhaustive list of which elements are used at the time of this document’s creation.

Creating Updated Virus Maps

Map overlays require special geojson files which are extremely dense specially formatted json which describes the way that regions of maps should be treated and interacted with. A script is included in the repo called **virus\_geojson\_builder.py** which can create the regional overviews by virus. It depends on two files:

Arbovirus\_distribution-use-for-layer.xlsx, which contains the mappings. Future updates should follow this same format. Relevant data is pulled from the sheet named “Binary ISO alpha 3 – virus”.

And a shapefile. Any map shapefile will work, but the one that I used for the layer’s initial creation is “ne\_50m\_admin\_0\_countries.shp”. This is a free-to-use shapefile provided by [Natural Earth](https://www.naturalearthdata.com/). There are other versions available here, but this one was the easiest to use, and closely matched the visual of the rest of the map.

From there, the resulting geojson file must be uploaded to MapBox. This will give the user a key to access the layer. The layer is defined in /arboverse\_updated/static/js/mapboxutil.js. If you wish to update an existing layer, you should find the matching layer in mapboxutl.js and replace the url with the one given by mapbox for your new layer.