1. **Ensure that the Virtual Environment is Activated**
   Before proceeding, verify that you have activated the virtual environment. This is crucial to ensure that the correct dependencies are used for the project.

```
PS D:\W3\inventory_management> python -m venv venv
>>
PS D:\W3\inventory_management> venv\Scripts\activate
(venv) PS D:\W3\inventory_management>
```

```
(venv) PS D:\W3\inventory_management> cd inventory_management
(venv) PS D:\W3\inventory_management\inventory_management> docker-compose build
[+] Building 191.2s (13/13) FINISHED
```

```
(venv) PS D:\W3\inventory_management\inventory_management> docker-compose up
[+] Running 3/3
 ✓ Network inventory_management_default  Created
 ✓ Container inventory_management-db-1   Created
 ✓ Container inventory_management-web-1  Created
```

```
(venv) PS D:\W3\inventory_management\inventory_management> docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                    NAMES
1f65373e1e2f   inventory_management-web "python manage.py ru…"   9 minutes ago   Up 9 minutes   0.0.0.0:8000->8000/tcp   inventory_management-web-1
01fd9094794c   postgis/postgis:15-3.3   "docker-entrypoint.s…"   9 minutes ago   Up 9 minutes   0.0.0.0:5432->5432/tcp   inventory_management-db-1
```

2. **You have to create an admin or superuser in the web container who can manage all the tables.**

```
(venv) PS D:\W3\inventory_management\inventory_management> docker exec -it inventory_management-web-1 bash
root@1f65373e1e2f:/app# python manage.py createsuperuser
Username (leave blank to use 'root'): Sultana
Email address: sultana@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
root@1f65373e1e2f:/app#
```

- **Property Owner Sign-Up Page**

  The **Sign-Up** page at http://localhost:8000/properties allows **Property Owners** to register by providing a unique username and email address.

  - Upon successful registration, a confirmation message will be displayed, indicating that the user has been successfully registered.
  - If the provided username or email is already in use or does not meet the required validation criteria, an error message will be shown, specifying the issue (e.g., username or email already taken).
  - This ensures that each **Property Owner** has a unique account within the system.

localhost:8000/properties/

Sign Up

Username:

Email:

Password:

Sign Up

Sign Up

Your account has been created! You can now log in.

Username:

Email:

Password:

Sign Up

Sign Up

Username:

Sultana

This username is already taken. Please choose a different one.

Email:

This email is already taken. Please choose a different one.

Password:

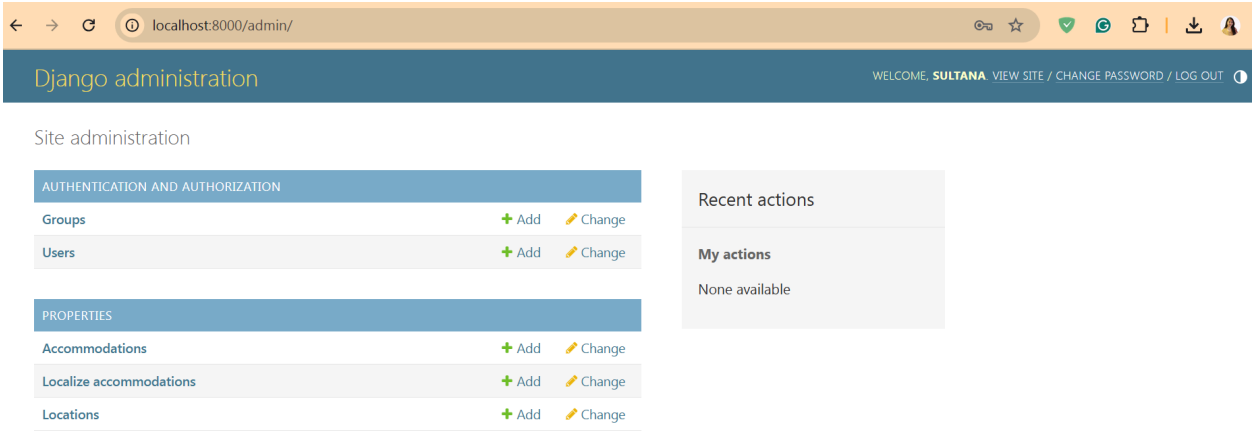
•••••

Sign Up

- **Django Admin Interface**

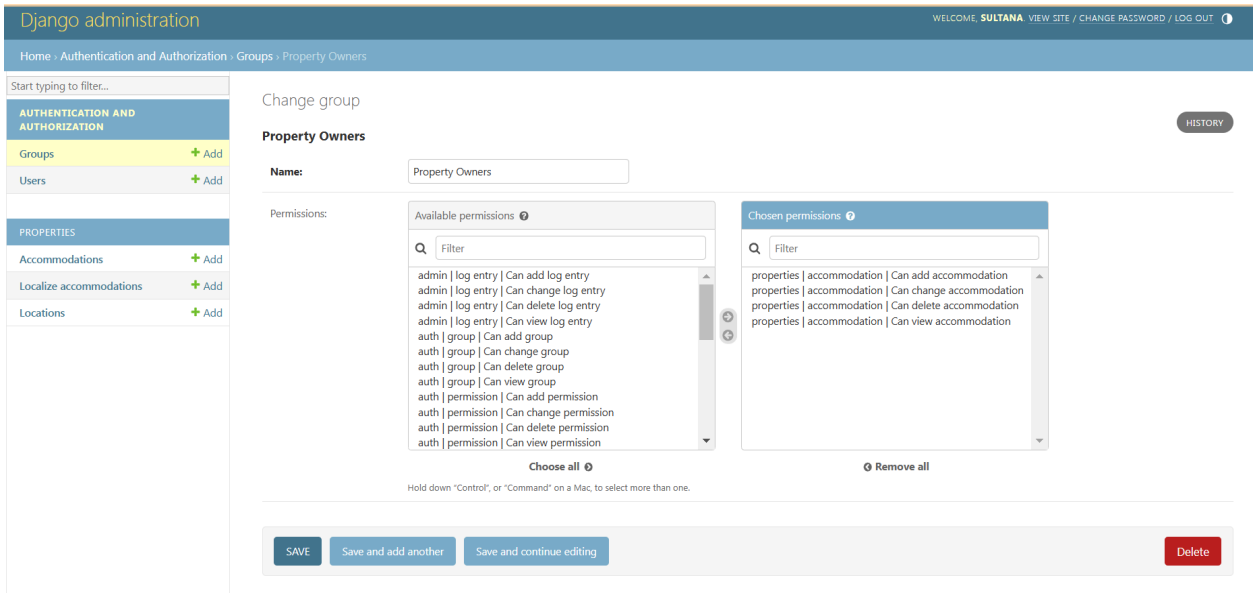
  The **Django Admin Interface** is accessible at http://localhost:8000/admin, where users can log in using their username and password.

  - On your first login, use the **Admin/Superuser** credentials that were created during the setup process.
  - After logging in successfully, you will be presented with an administrative interface that allows you to manage various aspects of the application, such as locations, accommodations, and user permissions.



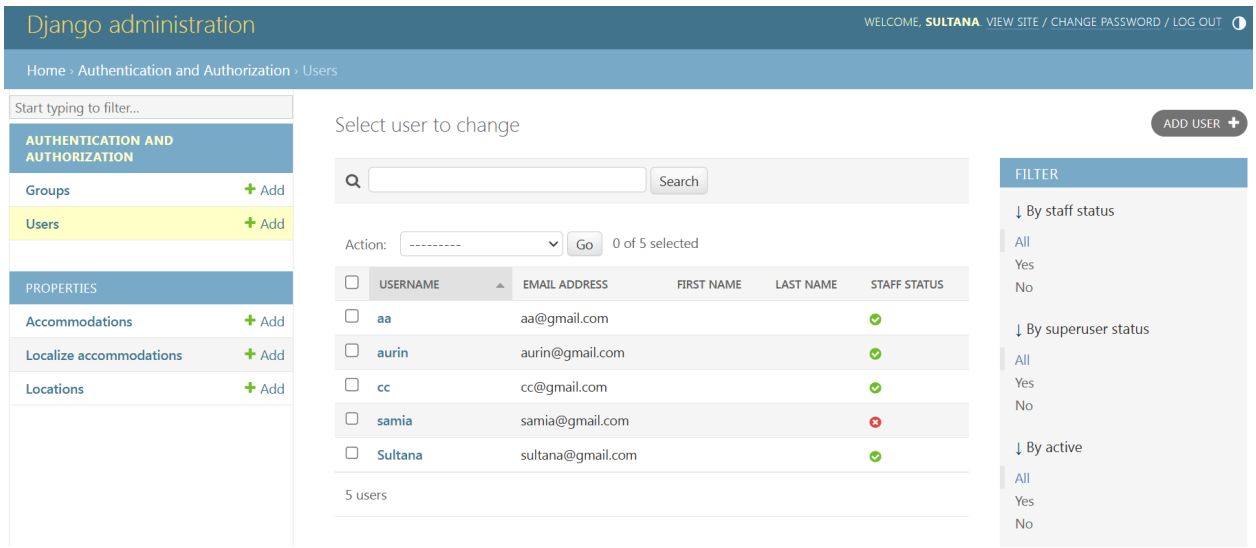
Managing User Groups and Permissions

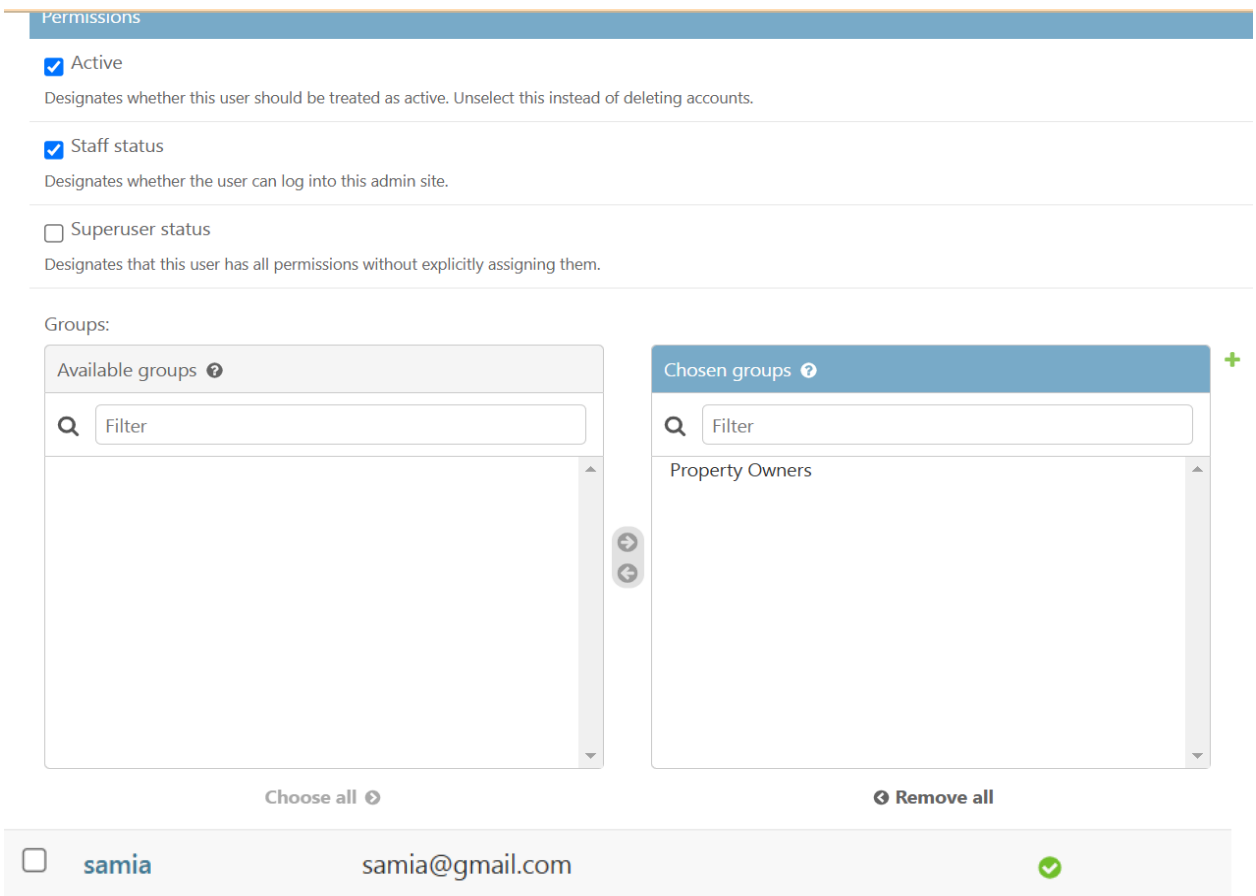
1. **Create Property Owners Group**:
   - In the Django Admin Interface, navigate to **Groups** and create a new group named **Property Owners**.
   - Assign the **Accommodations** table to this group, ensuring that users in this group can only access and manage their own properties.
   - Save the group configuration. As a result, each user in the **Property Owners** group will only be able to view and manage their own accommodations; they will not have access to other users' properties.



2. **Manage User Access**:
   - Navigate to the **Users** section in the Admin Interface.
   - You will see all registered users, but newly registered users will have a red cross under the **Staff status** column.
   - To enable a user to log in, click on the user, check the **Staff status** box, and save the changes.
   - Only users with the **Staff status** enabled will be able to log into the Admin interface. Users without this status will not be able to access the system.

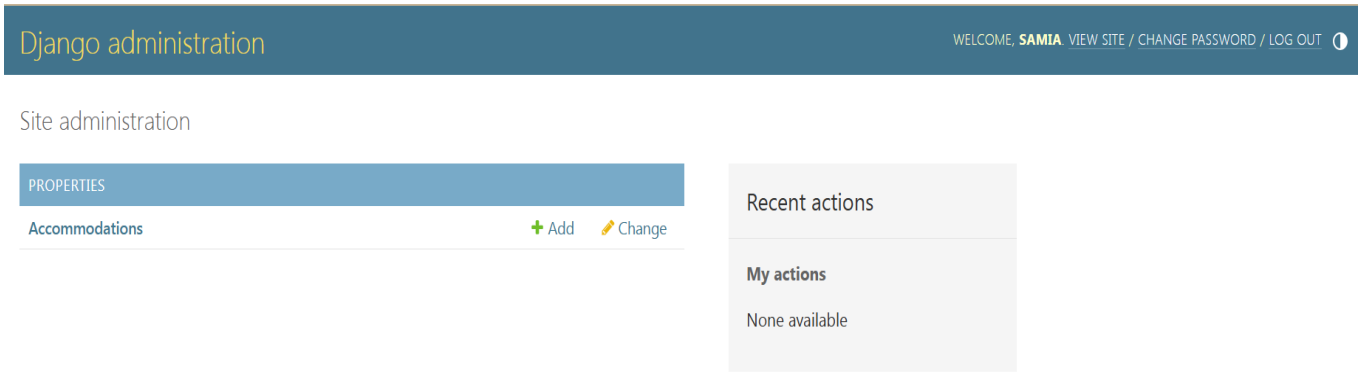
These steps ensure proper user management and access control, providing each user with the appropriate permissions based on their role.





- **Staff Status Activation**:
  - When a new user, such as **Samia**, registers, the **Staff status** will initially be unchecked, represented by a red cross.
  - Once the **Admin** enables the **Staff status** for **Samia**, a green checkmark will appear, indicating that she now has access to the Admin interface.

- **Logging in as the New User**:
  - After the **Admin** has granted staff access, ensure that you **log out** of the Admin interface if you're currently logged in as the Admin.
  - **Samia** can then log in with her credentials at http://localhost:8000/admin.
  - Upon logging in, **Samia** will have access only to the **Accommodations** table, where she can manage her own properties. She will not have access to other sections or user data, ensuring role-based access control.



➢ **Property owners** can add their properties to the **Accommodations Table** by providing the following information:

1. **id**: A unique identifier for the property (Primary Key, string, max 20 characters).
2. **feed**: Feed number (unsigned small integer), with a default value of 0.
3. **title**: The name of the property (string, max 100 characters), required.
4. **country_code**: The ISO country code (string, max 2 characters), required.
5. **bedroom_count**: The number of bedrooms (unsigned integer).
6. **review_score**: The review score for the property (numeric, 1 decimal place), default is 0.
7. **usd_rate**: The price rate in USD (numeric, 2 decimal places).
8. **center**: The geolocation of the property, stored as a PostGIS point.
9. **images**: An array of image URLs, with each URL having a maximum of 300 characters.
10. **location_id**: A foreign key referencing the **Location** table. To enable the selection of locations in the dropdown menu, the admin must first add entries to the **Location** table. Log in as an admin and ensure that the required locations are added before attempting to use this feature.
11. **amenities**: A JSONB array of amenities, with each amenity having a maximum of 100 characters.

Additionally, the following column is automatically added to associate the property with its owner:
12. **user_id**: A foreign key referencing Django's **auth_user** table, which is automatically set to the logged-in user when the property is created.

Add accommodation

Id:	3
Feed:	0
Title:	house3
Country code:	FR
Bedroom count:	3
Review score:	4
Usd rate:	100

Images:

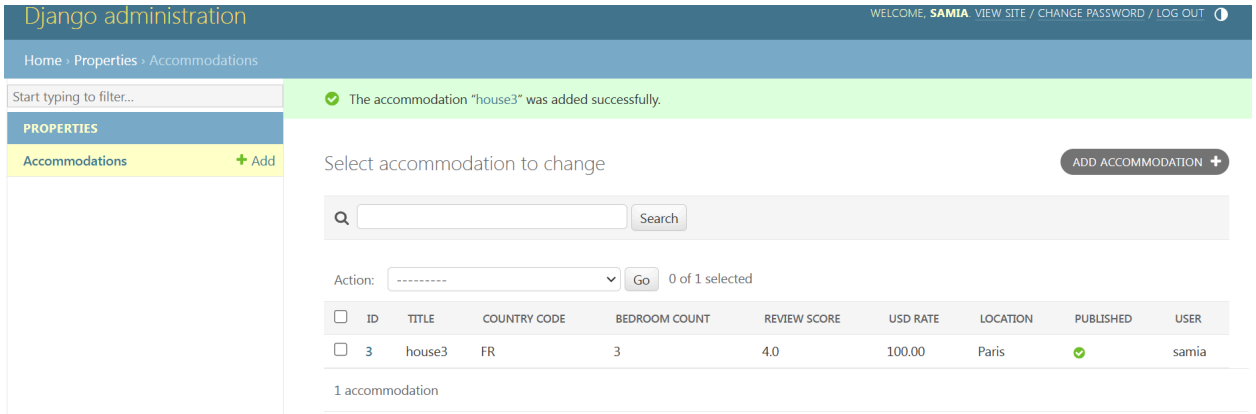
```
[
"https://example.com/images/beach_sunset.jpg",
"https://example.com/images/forest_pathway.jpg"
]
```

Location: Paris

Amenities:

```
[
"wifi", "free-furnitures"
]
```

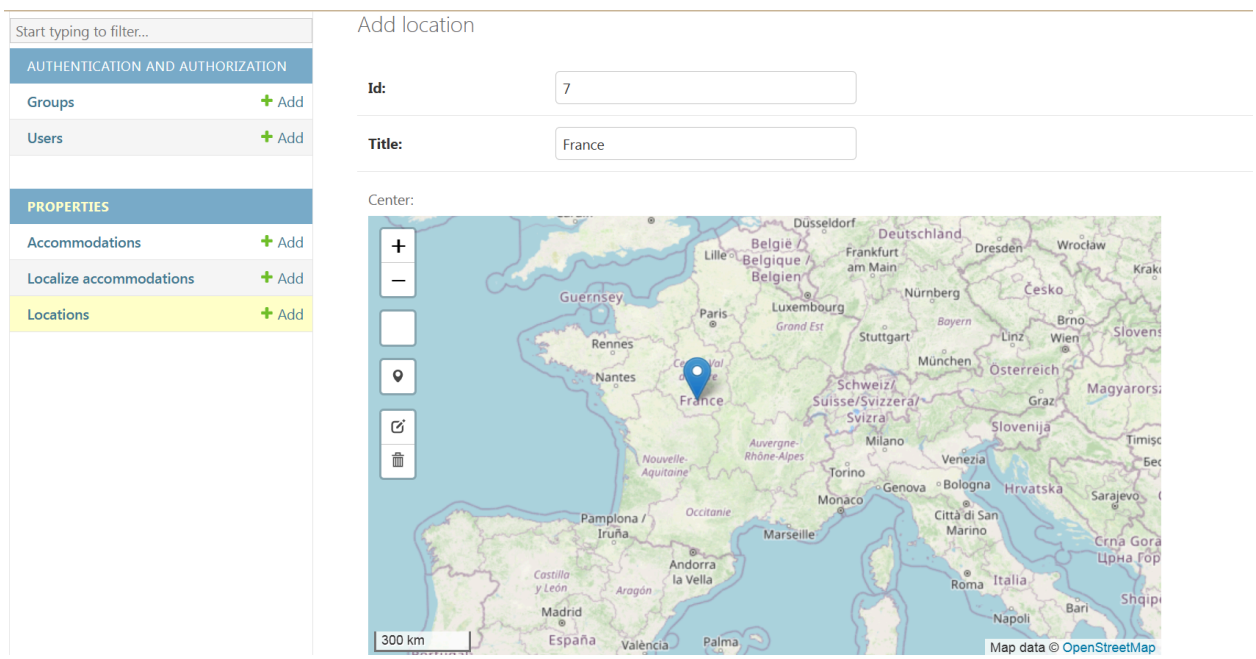
☑ Published

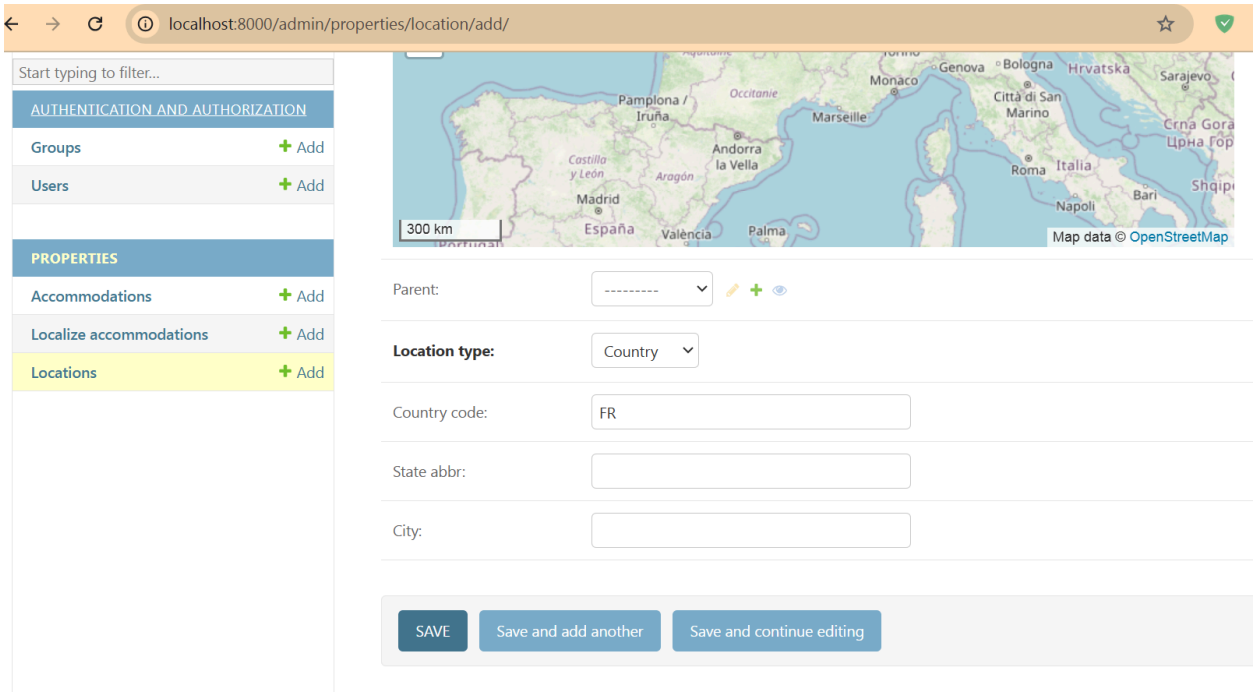


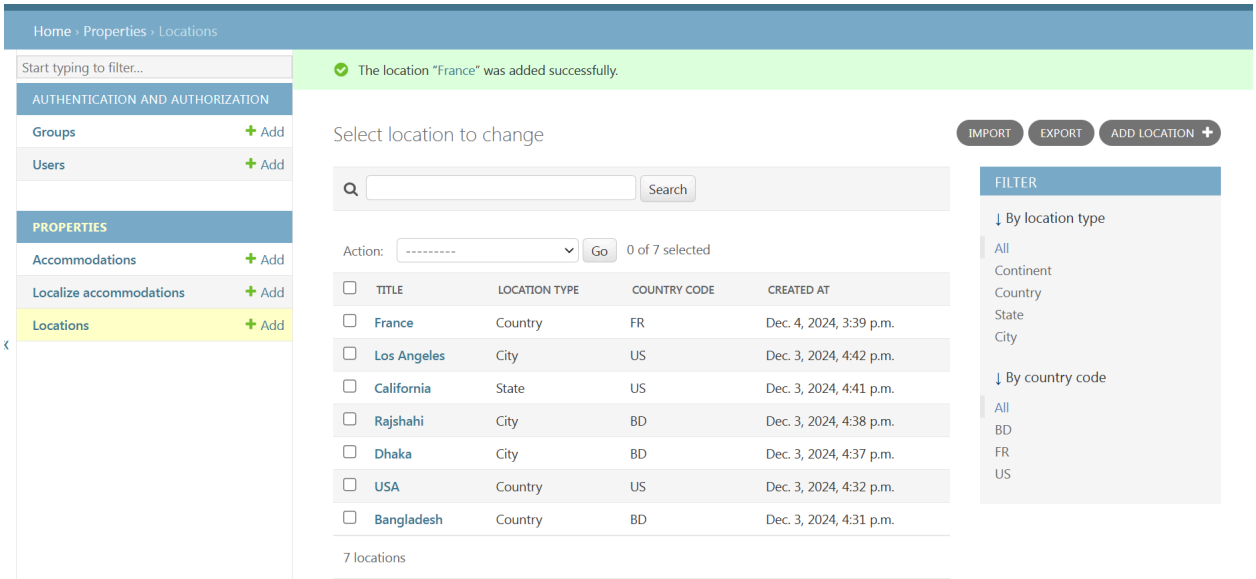
Managing Locations Table

Only Admin/Superuser can manage the Locations table. The Location model is designed to handle hierarchical data, including continents, countries, states, and cities.

1. ***Add Countries***:
   - Start by adding all the countries to the system. For example, to add France as a country:
     - Navigate to the Locations table in the Admin interface.
     - Select Add Location and input France as a Country (Click the green '+' sign)
     - Leave the Parent field empty if France does not require a parent location (or select Continent as its parent if needed).



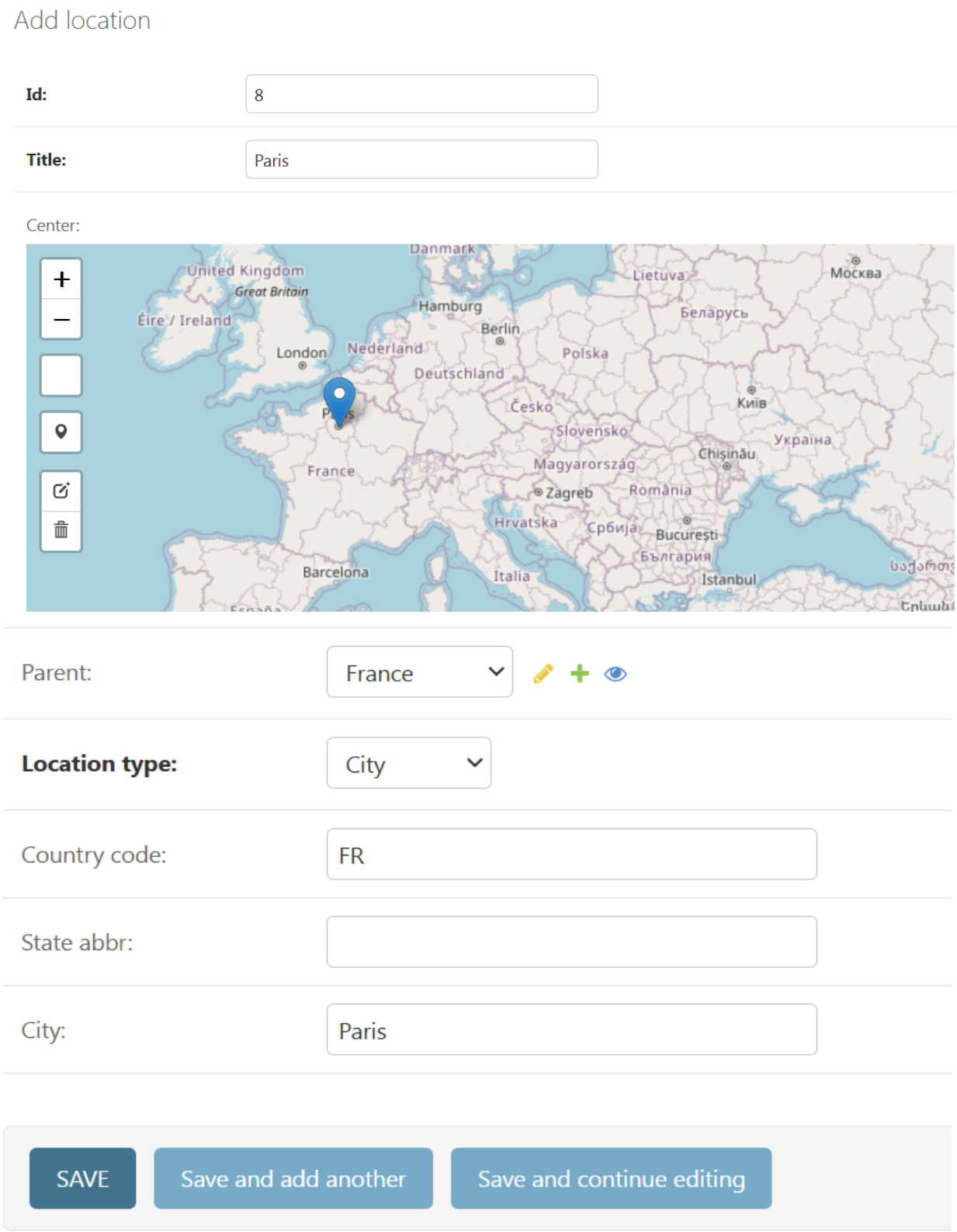




Countries Without States:

For countries like **Bangladesh or France**, which do not have states, you can directly add **cities** under the country. In this case, the **Parent** of the city will be the **Country** itself.

- For example, add **Paris** as a **City** under **France** with **France** as the **Parent**.
- The Id will be unique (string, (max 20 characters))



Countries With States and Cities:

If the country has states, add each **state** as a separate entry. For instance, for **USA**:

- Add **Texas** as a **State** under **USA** by selecting **State** as the **Location type** and setting **USA** as the **Parent**.
- After adding the state, you can then add **cities** under that state. For example, add **Houston** as a **City** with **Texas** as the **Parent**.

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

PROPERTIES

Accommodations + Add

Localize accommodations + Add

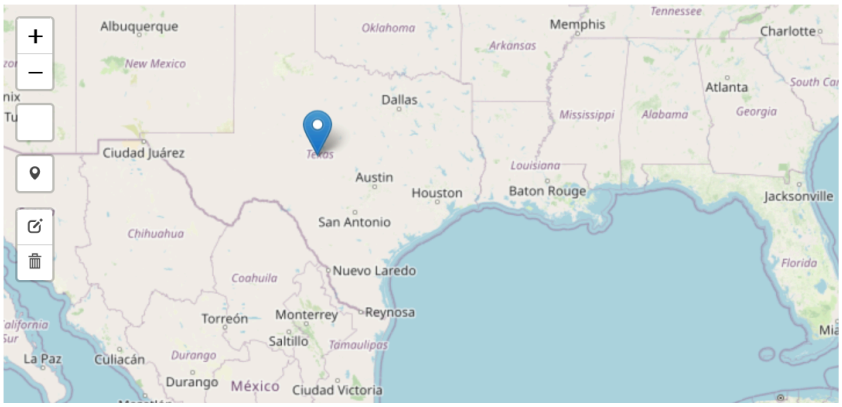
Locations + Add

Add location

Id: 9

Title: Texas

Center:



Parent: USA

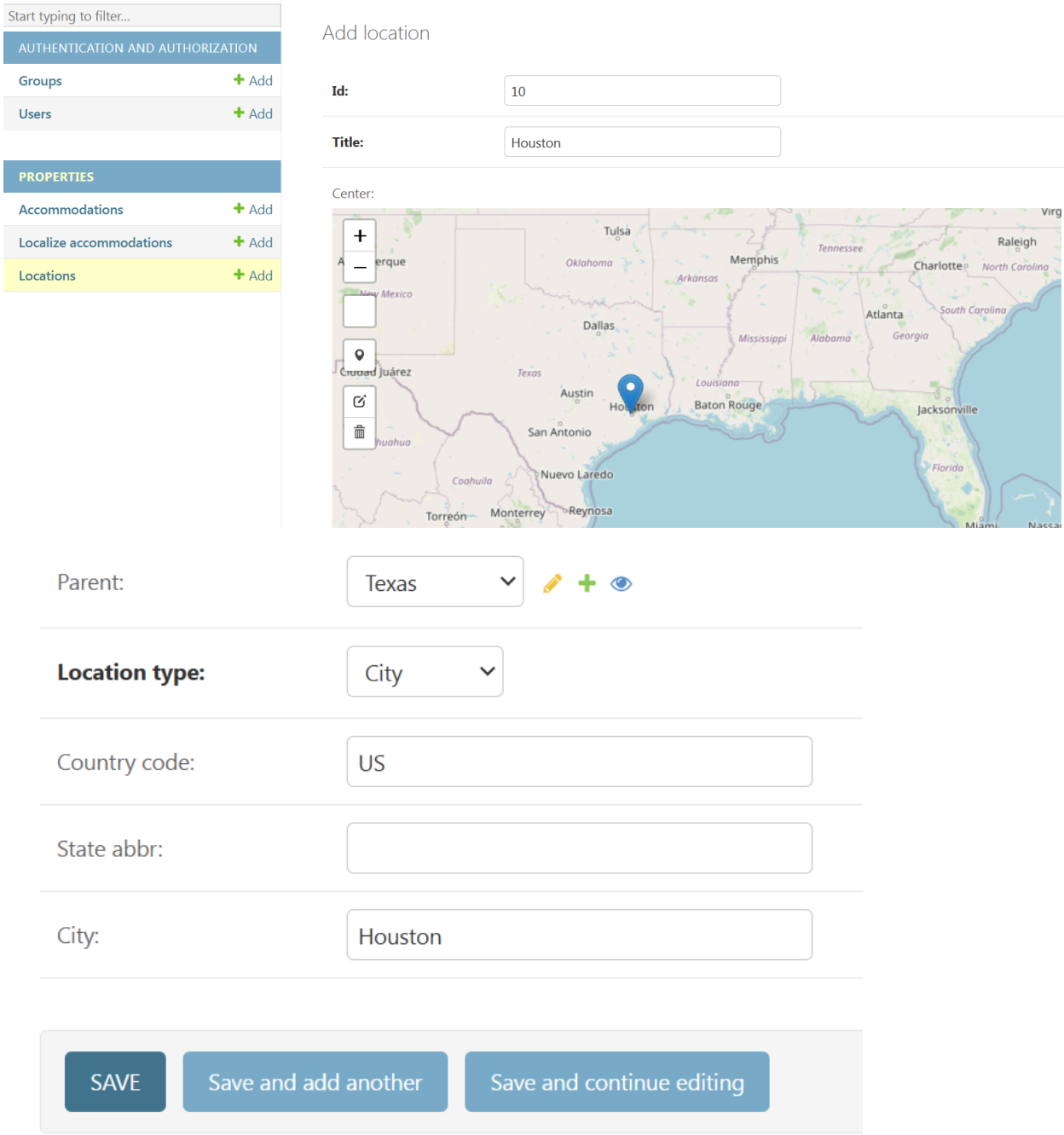
Location type: State

Country code: US

State abbr: TX

City: Texas

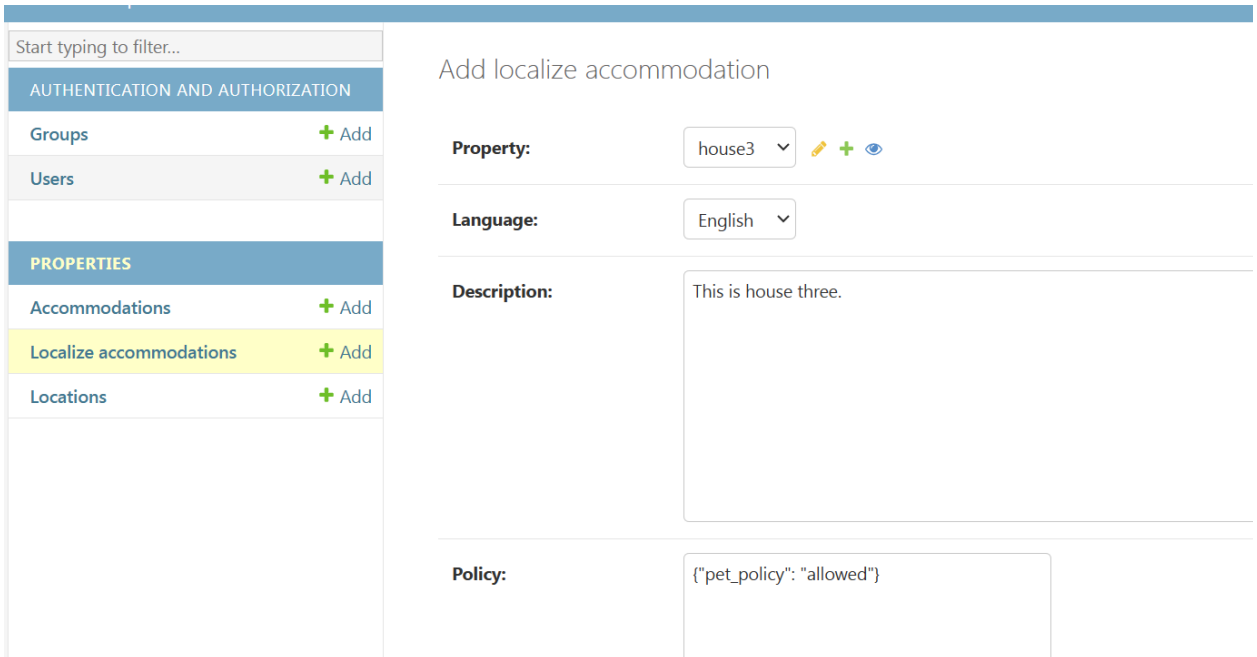
SAVE | Save and add another | Save and continue editing



2. ***Update & Delete Countries***:

- Navigate to the **Locations** table in the Django Admin interface.
- Click on the desired entry you wish to update.
- Edit the relevant fields.
- Once you have made the necessary changes, click **Save** to apply the updates.
- In the **Locations** table, select the entry you wish to delete.
- In the **Actions** dropdown menu, choose the **Delete** option.
- Click **Go** to confirm the deletion.
- A confirmation prompt will appear to finalize the deletion. Confirm the action to remove the entry.

Managing Localize accommodations Table



➢ The **LocalizeAccommodation** table is designed to store localized information for properties listed in the **Accommodations Table**. The following fields are included:

1. **id**: Primary key, auto-incremented.
2. **property_id**: Foreign key referencing the **Accommodation** table.
3. **language**: Language code (string, max 2 characters).
4. **description**: Localized description (text).
5. **policy**: A JSONB dictionary (e.g., `{"pet_policy": "value"}`).

- The **LocalizeAccommodation** table allows administrators or superusers to provide localized content for properties listed in the **Accommodations Table**, such as translating descriptions or adding location-specific policies. This table ensures that accommodations are accessible and understandable in multiple languages, enhancing the user experience. Administrators can manage this table directly through the admin interface, as demonstrated in the accompanying example image.
- The **LocalizeAccommodation** model enables the localization of accommodation details, such as descriptions and policies, in different languages. The model includes a `language` field with dropdown choices (e.g., English, French, Spanish) defined by the `LANGUAGE_CHOICES`. It ensures that each accommodation has unique localized content for each language, as enforced by the `unique_together` constraint on `property` and `language`.
- To validate the correctness of language-specific entries, the `clean` method uses the **langid** library (which must be imported) to detect the language of the `description` and each string within the `policy` JSON. If the detected language does not match the selected language, a **ValidationError** is raised. This ensures accurate and consistent localization across all accommodations.
- Demo screenshots given below will demonstrate how incorrect language entries trigger validation errors and how the dropdown for `language` makes selection intuitive.

localhost:8000/admin/properties/localizeaccommodation/7/change/

Django administration

Home › Properties › Localize accommodations › house5 - ar

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups + Add
- Users + Add

PROPERTIES

- Accommodations + Add
- Localize accommodations + Add
- Locations + Add

Change localize accommodation

house5 - ar

Please correct the error below.

The description is not in the expected language (ar). It is in en.

Property: house5

Language: Arabic

Description: this is house five.

Policy: {"policy": "value"}

SAVE | Save and add another | Save and continue editing

Testing

```
(venv) w3e55@w3e55:~/Assignments/inventory_management/inventory_management$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                                       NAMES
e5ec43010a5b   inventory_management-web   "python manage.py ru…"   7 minutes ago   Up 7 minutes   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   inventory_management-web-1
b31cdfdf93f5   postgis/postgis:15-3.3     "docker-entrypoint.s…"   47 hours ago    Up 7 minutes   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   inventory_management-db-1
(venv) w3e55@w3e55:~/Assignments/inventory_management/inventory_management$ docker exec -it inventory_management-web-1 bash
root@e5ec43010a5b:/app# pip install coverage
Collecting coverage
  Downloading coverage-7.6.8-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (234 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 234.5/234.5 kB 1.0 MB/s eta 0:00:00
Installing collected packages: coverage
Successfully installed coverage-7.6.8
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual envi
ronment instead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
root@e5ec43010a5b:/app# coverage run --source='.' manage.py test
```

```
root@e5ec43010a5b:/app# coverage run --source='.' manage.py test
Found 11 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...........
----------------------------------------------------------------------
Ran 11 tests in 3.128s

OK
Destroying test database for alias 'default'...
```

```
root@e5ec43010a5b:/app# coverage report
Name                                                    Stmts   Miss  Cover
-------------------------------------------------------------------------
inventory_management/__init__.py                            0      0   100%
inventory_management/asgi.py                                4      4     0%
inventory_management/settings.py                           21      0   100%
inventory_management/urls.py                                4      0   100%
inventory_management/wsgi.py                                4      4     0%
manage.py                                                  11      2    82%
properties/__init__.py                                      0      0   100%
properties/admin.py                                        66     25    62%
properties/apps.py                                          4      0   100%
properties/forms.py                                        18      0   100%
properties/migrations/0001_initial.py                       8      0   100%
properties/migrations/0002_alter_accommodation_center.py    5      0   100%
properties/migrations/__init__.py                           0      0   100%
properties/models.py                                       67     12    82%
properties/tests.py                                        60      0   100%
properties/urls.py                                          4      0   100%
properties/views.py                                        16      1    94%
-------------------------------------------------------------------------
TOTAL                                                     292     48    84%
root@e5ec43010a5b:/app#
```