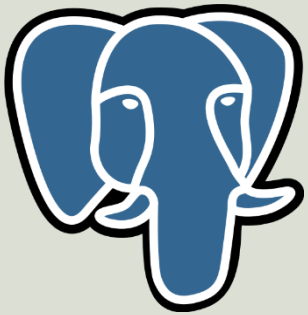
https://www.halvorsen.blog



Django and PostgreSQL

Hans-Petter Halvorsen

Contents

Introduction

Hans-Petter Halvorsen

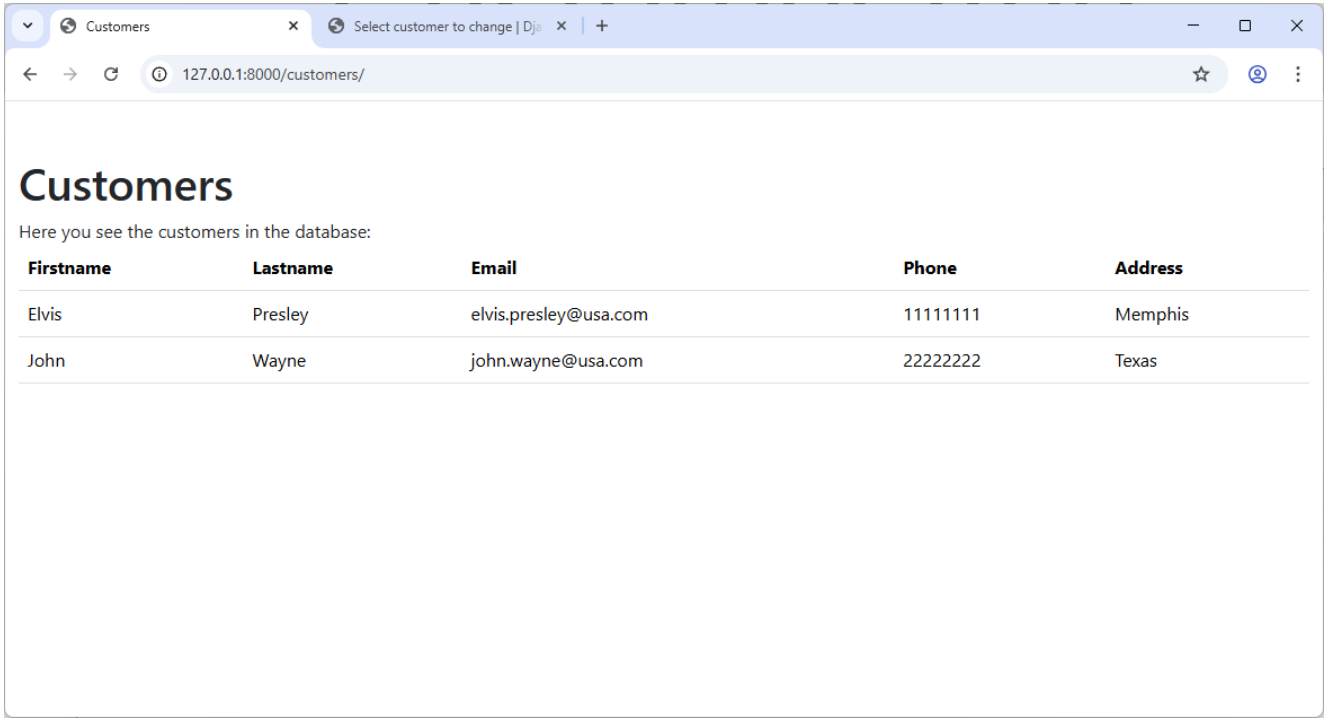
Table of Contents

Introduction

- We will create a Django Project and a Django Web Application.
- Django has support for databases like SQLite, MySQL/MariaDB and PostgreSQL.
- Django includes an SQLite database, but for the others you need to install the database system you want to use from their respective websites and in addition install a Python package/driver for the specific database system.
- In this Tutorial we will see how we can use **Django** in combination with **PostgreSQL**.
- We will create a simple "**Customer App**" that retrieves a list of Customers stored in the PostgreSQL database.
- Finally, we will use the built-in "**Django Admin**" module to create **CRUD** functionality, i.e., the possibility to Create, Read, Update and Delete Customers in the PostgreSQL database.

Customer App

We shall create a basic Django Web Application like this where the data are stored in a PostgreSQL database:



Software and Tools

- Python
  - We need to have Python installed to install and use Django.
- Django
  - Django is basically a Python package installed using PIP which is part of Python.
- PostgreSQL database
- Visual Studio Code

Django

Table of Contents

Hans-Petter Halvorsen

Django

- Django is a Python web development framework.
- Django is a back-end/server-side web framework.
- Django has support for databases like SQLite, MySQL/MariaDB and PostgreSQL.
- Django includes an SQLite database, but for the others you need to install the database system you want to use from their respective websites and in addition install a Python package/driver for the specific database system.
- Django is free, open source and written in Python.
- Homepage: https://www.djangoproject.com

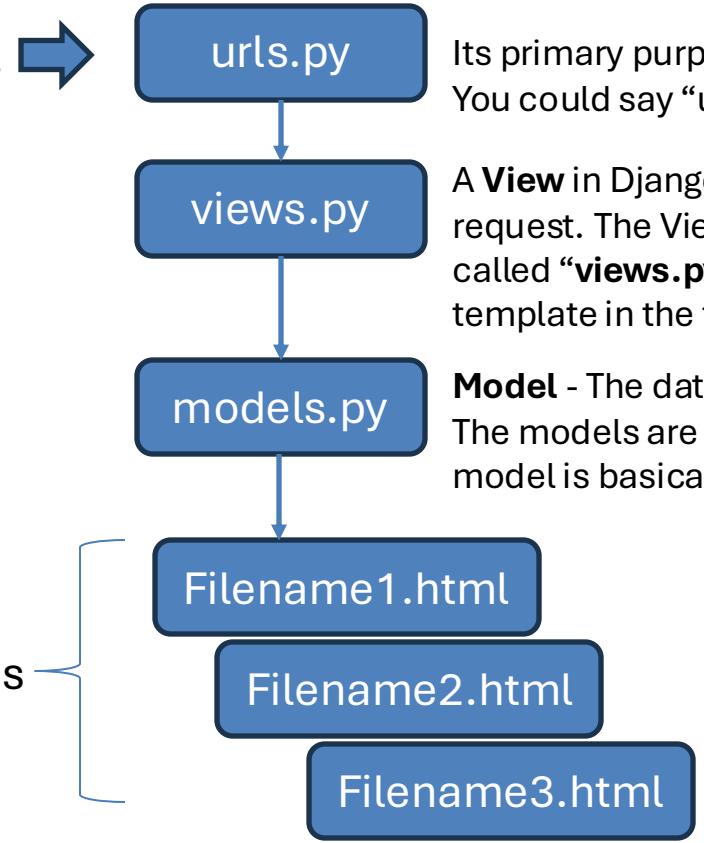
Model View Template (MVT)

Django follows the MVT design pattern (Model View Template).

- **Model** - The data you want to present, usually data from a database. The models are usually located in a file called "**models.py**".
- **View** - A request handler that returns the relevant template and content - based on the request from the user. The views are usually located in a file called "**views.py**".
- **Template** – A HTML file containing the layout of the web page, with logic on how to display the data. The HTML template files are stored in a subfolder called "**templates**"

Django Files and Folders

URL entered in the Web Browser



Its primary purpose is to map URL patterns to their corresponding views. You could say “urls.py” is the table of contents for the Django project.

A **View** in Django is basically a Python function that handles an HTTP request. The Views (Python Functions) are usually located in a file called “**views.py**”. The view then sends the data to a specified template in the template folder.

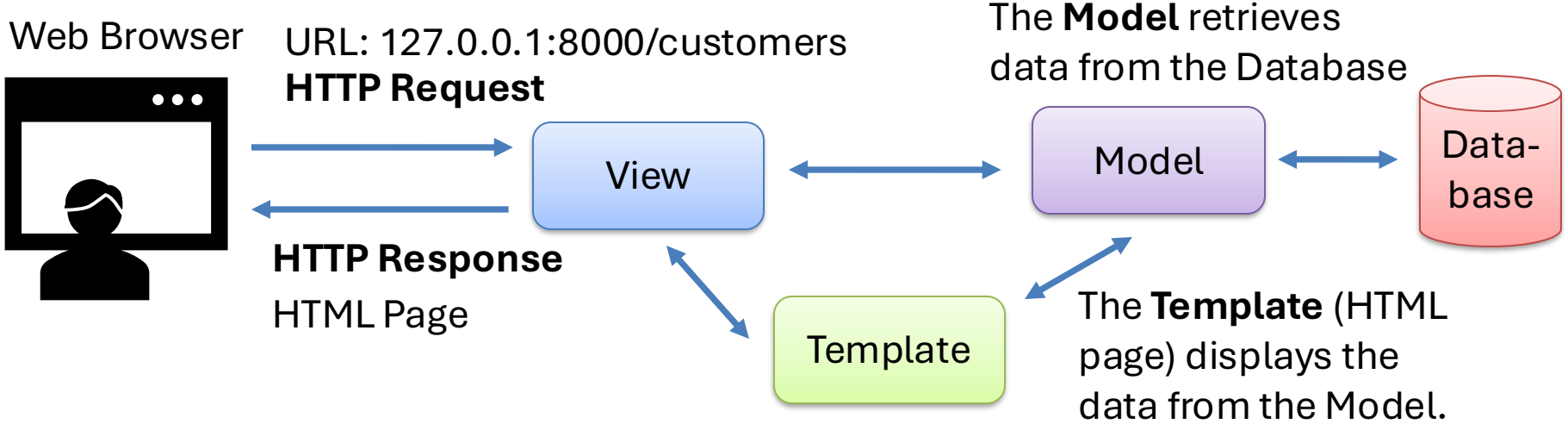
Model - The data you want to present, usually data from a database. The models are usually located in a file called “**models.py**”. A Django model is basically a table in your database.

Template Files

Templates - Django Templates are HTML files containing the layout of the web pages, with logic on how to display the data
The HTML template files are stored in a subfolder called “**templates**”

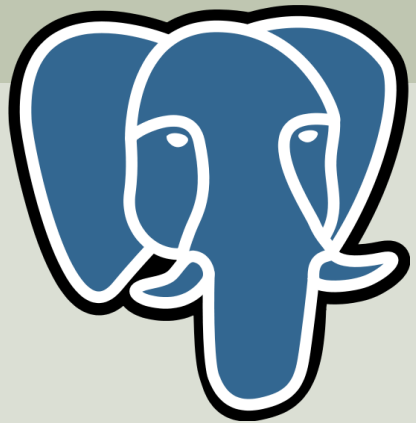
Django workflow

Django find which View to call based on information in “urls.py”.



The **View** handles the HTTP request from the Web Browser and sends data to the specific Template. Then the HTTP response is sent back to the Web Browser.

https://www.halvorsen.blog



PostgreSQL

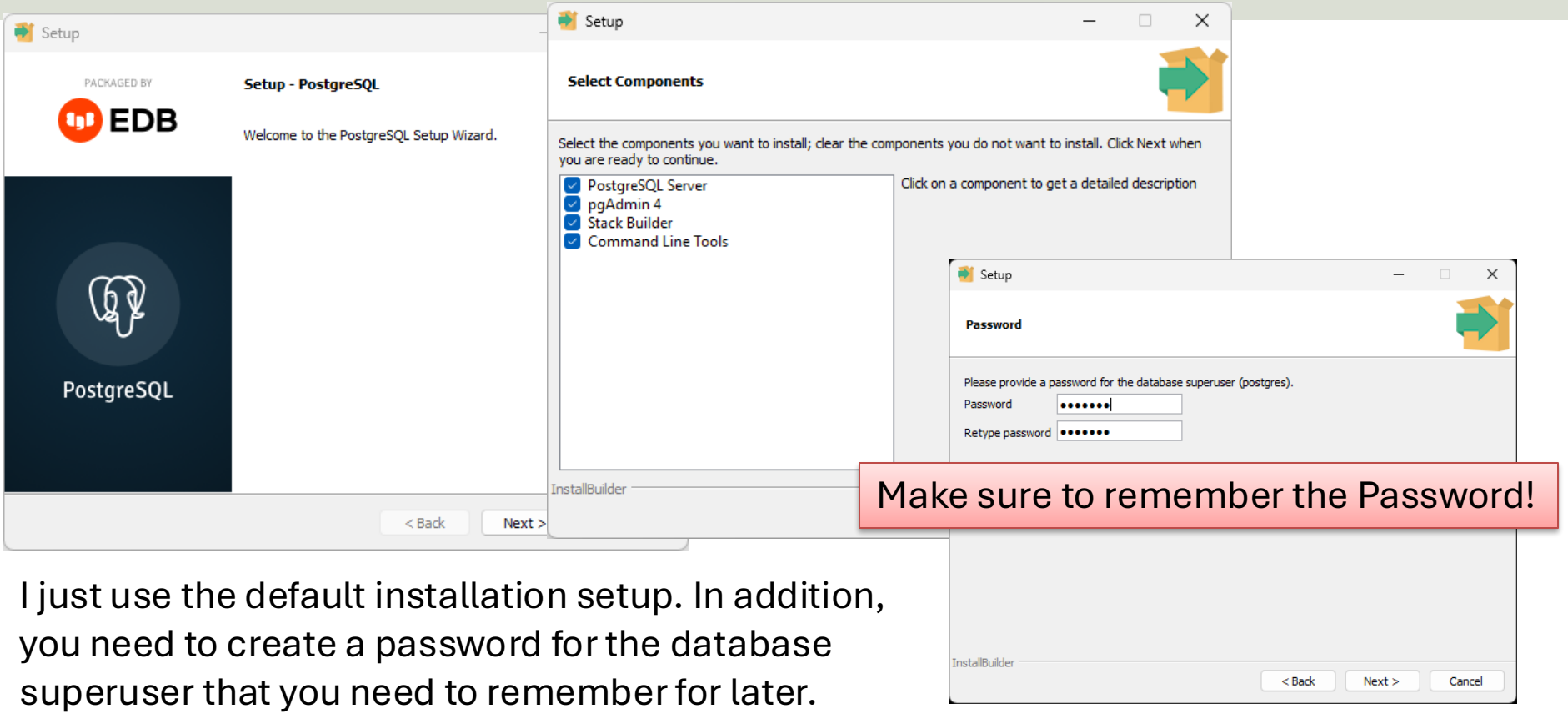
Table of Contents

Hans-Petter Halvorsen

PostgreSQL

- PostgreSQL is an open-source relational database system.
- PostgreSQL exists for Windows, macOS and Linux.
- Homepage: https://www.postgresql.org
- EnterpriseDB (EDB) is the company that is one of the largest contributor to PostgreSQL and responsible for the installer.
- EDB offer paid services for enterprises, but PostgreSQL itself is free.
- ERD Download Page: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

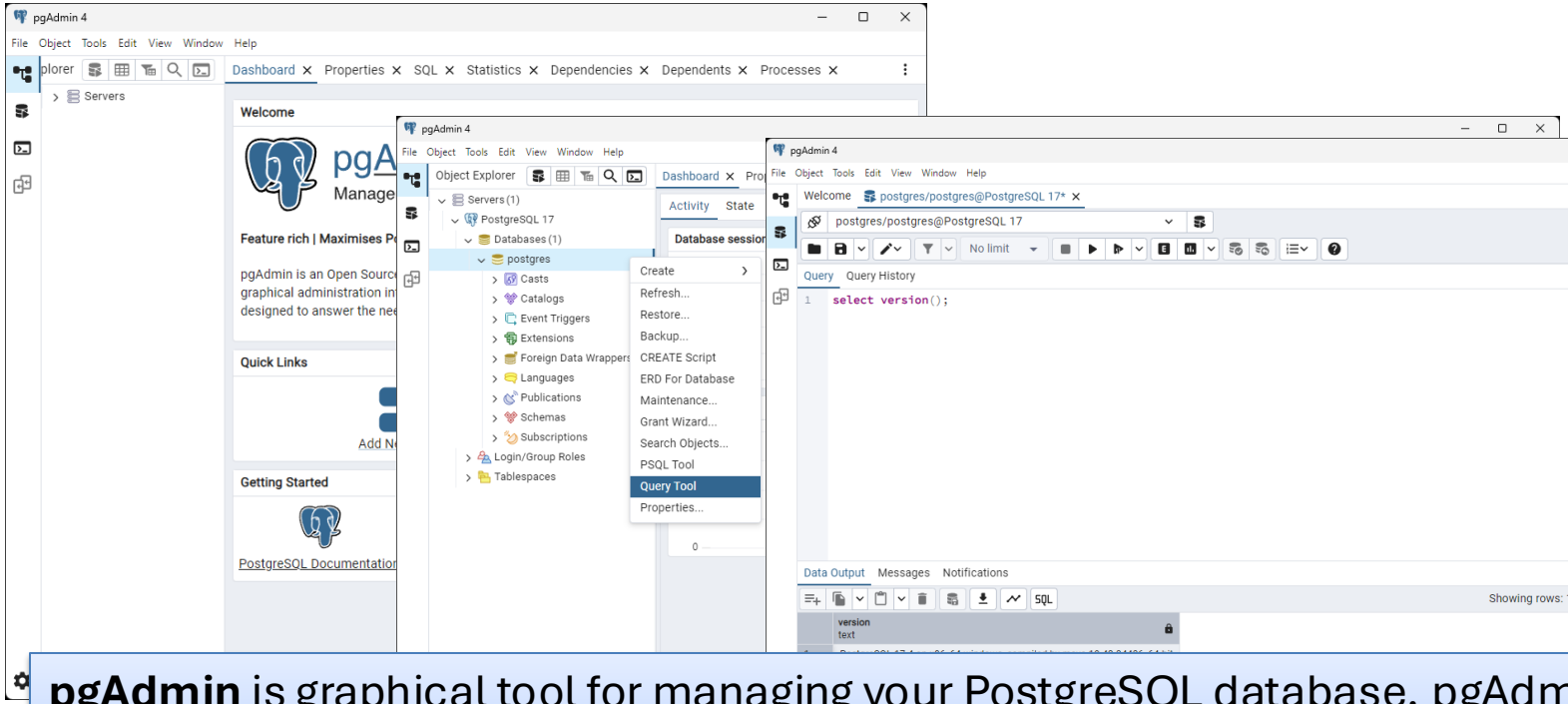
Installation



I just use the default installation setup. In addition, you need to create a password for the database superuser that you need to remember for later.

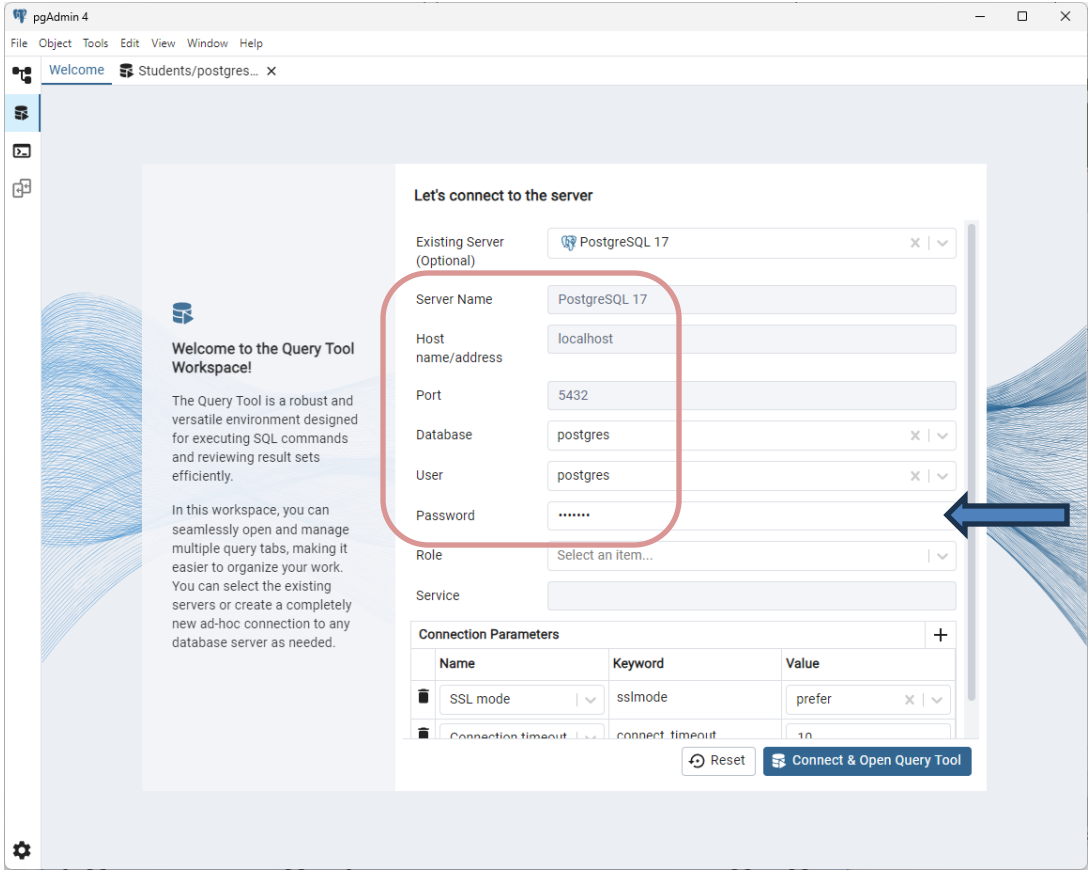
https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

pgAdmin



pgAdmin is graphical tool for managing your PostgreSQL database. pgAdmin is part of the installer from EDB. Here you can create new Database, new Tables, execute SQL Scripts, etc.

PostgreSQL Connection



The Password you entered during installation of the PostgreSQL database

https://www.halvorsen.blog

Create Django Project and App

Hans-Petter Halvorsen

Table of Contents

Create Django Project and App

1. Create and Activate Virtual Python Environment

```
>python –m venv venvname
```

```
C:\...\venvname\Scripts>activate.bat
```

2. Install Django

```
...>python -m pip install Django
```

3. Create Django Project

```
...>django-admin startproject projectname
```

4. Run the Django Project

```
...>python manage.py runserver
```

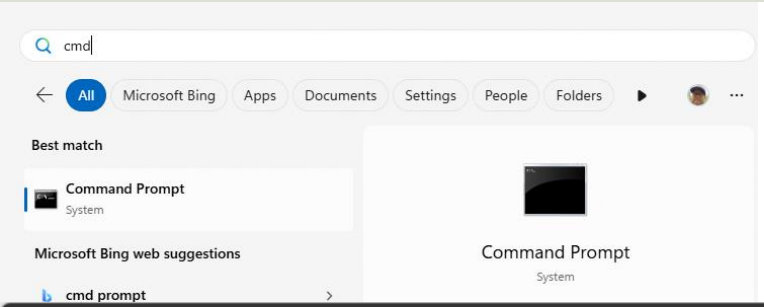
5. Create a Django App

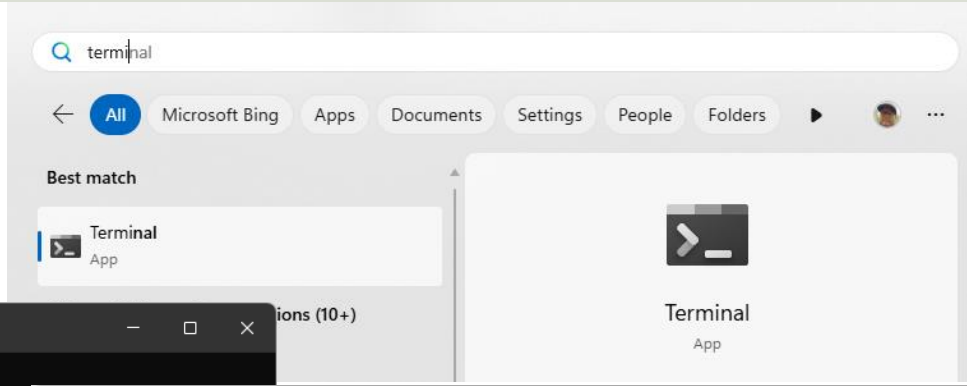
```
...>python manage.py startapp appname
```

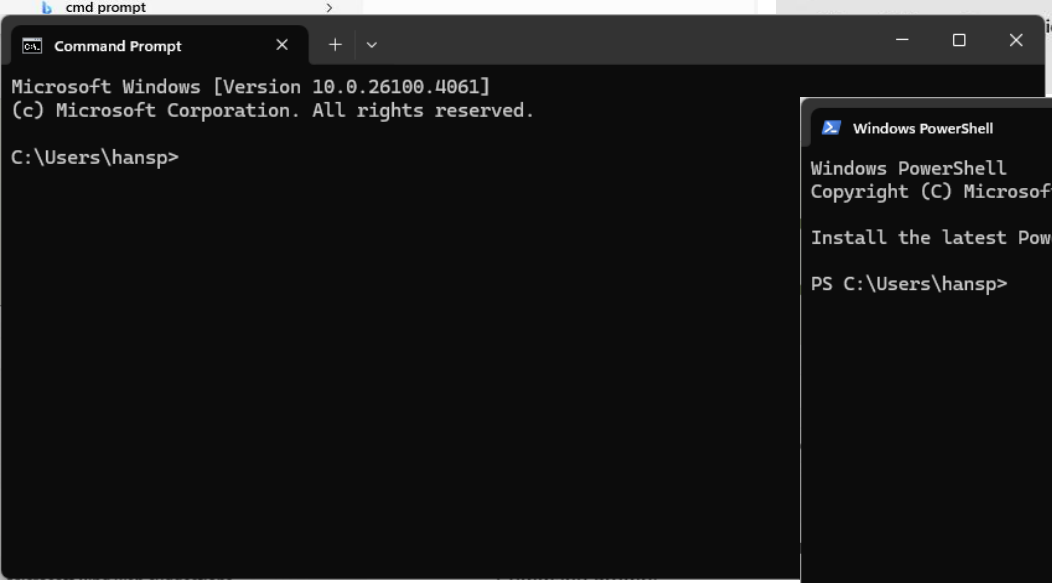
We need to do these steps to create a virtual Python environment (which is recommended), then install Django. Then you need to create a new Django Project and a Django App.

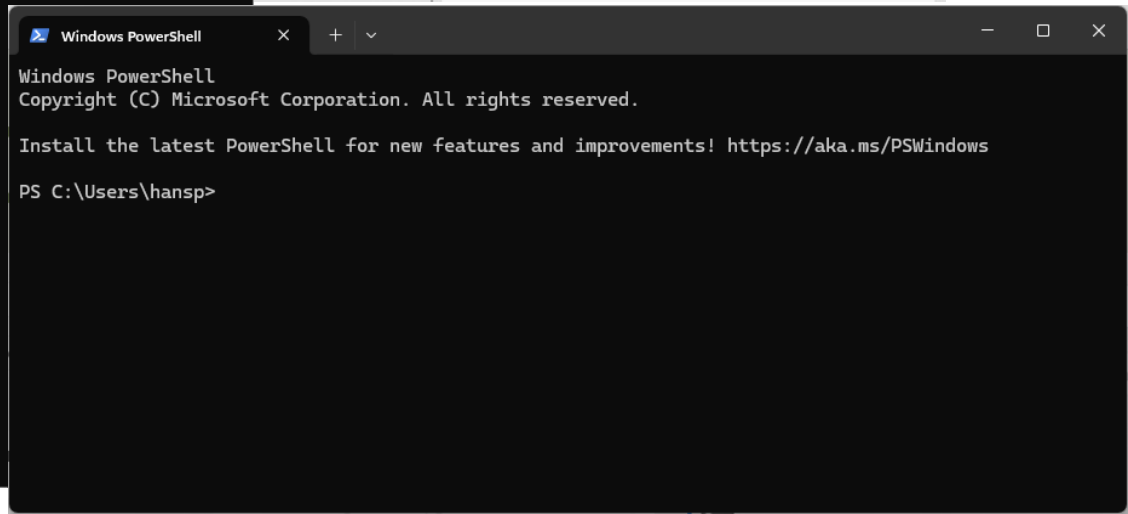
We can use the Command Prompt or the Terminal in Windows. We can also use the built-in Terminal in Visual Studio Code.

Command Prompt and Terminal





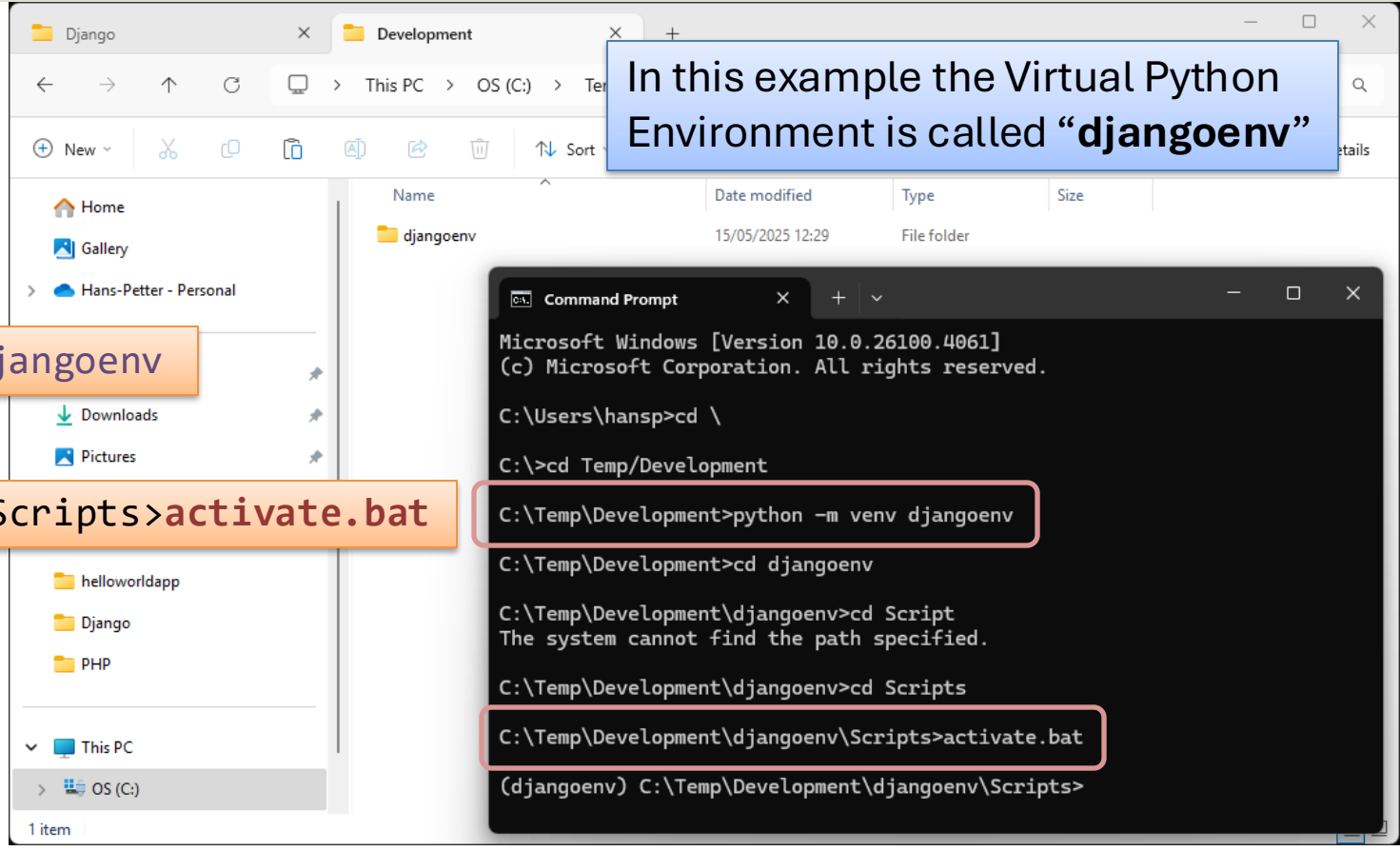




Create and Activate Virtual Python Environment

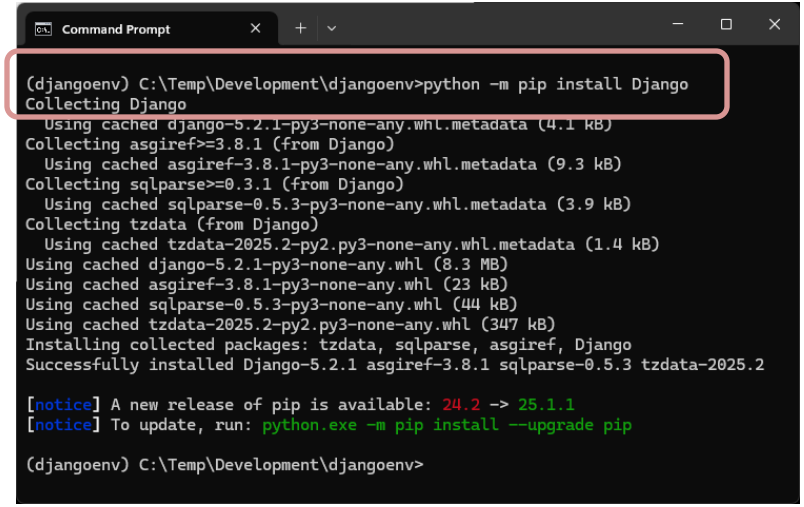
```
>python -m venv djangoenv
```

```
C:\...\djangoenv\Scripts>activate.bat
```

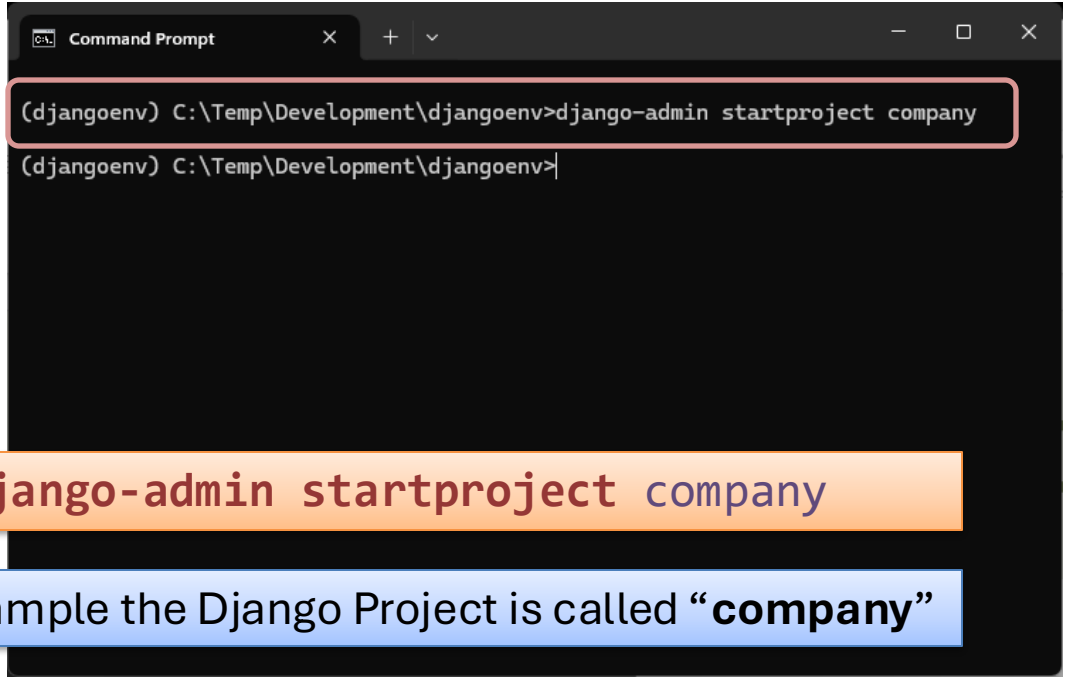


Install Django and Create Django Project

Install Django:




```
...>python -m pip install Django
```

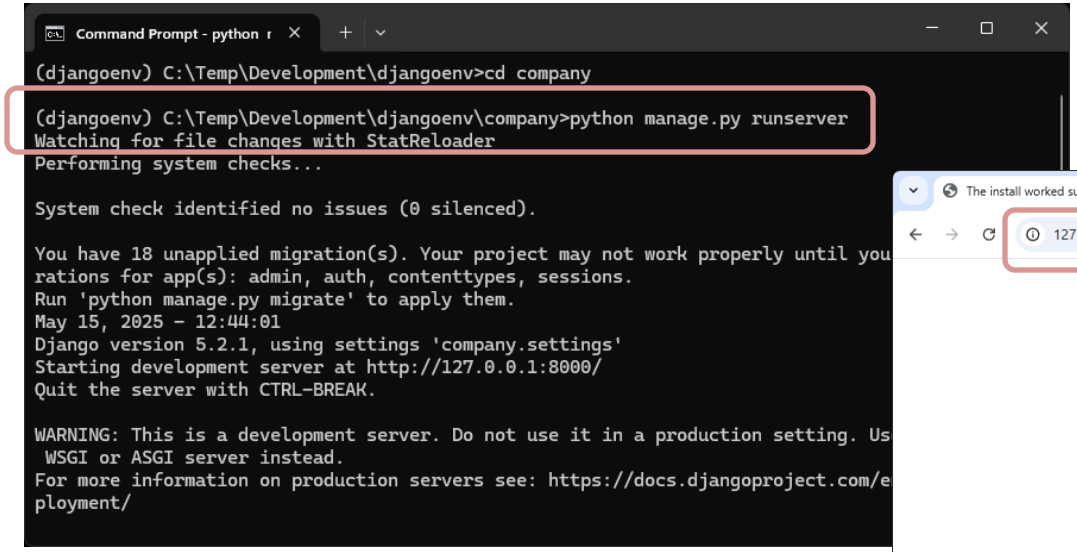


Create Django Project:

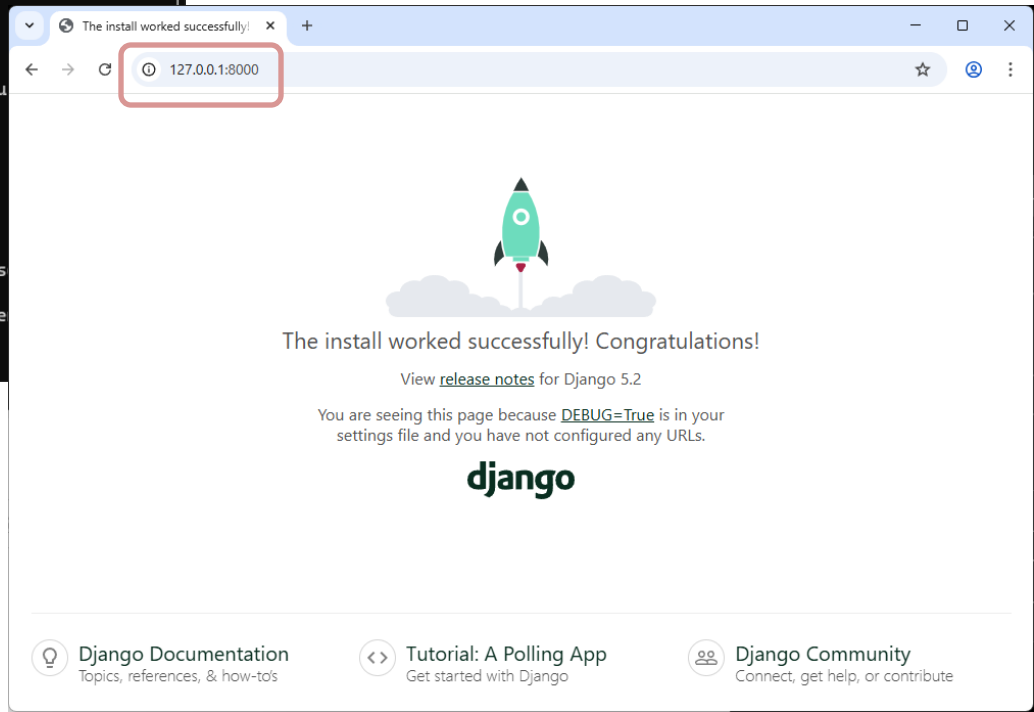
```
...>django-admin startproject company
```

In this example the Django Project is called “**company**”

Run the Django Project



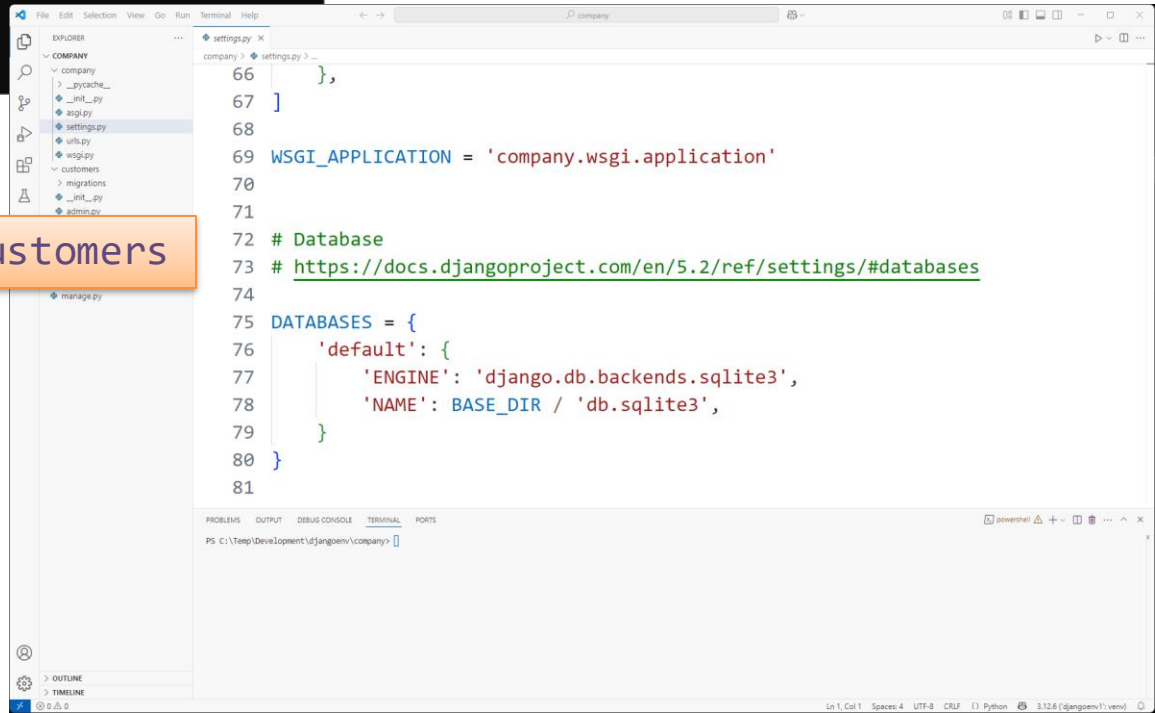

```
...>python manage.py runserver
```



Create a Django App

In this example the Django Project is called "**customers**"

```
...>python manage.py startapp customers
```



We can then open the project in Visual Studio Code:

Create a Django App

After creating the Django App, you need to insert the “customers” App in “**settings.py**”:

settings.py

company > settings.py > ...

```
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'customers'
41  ]
```

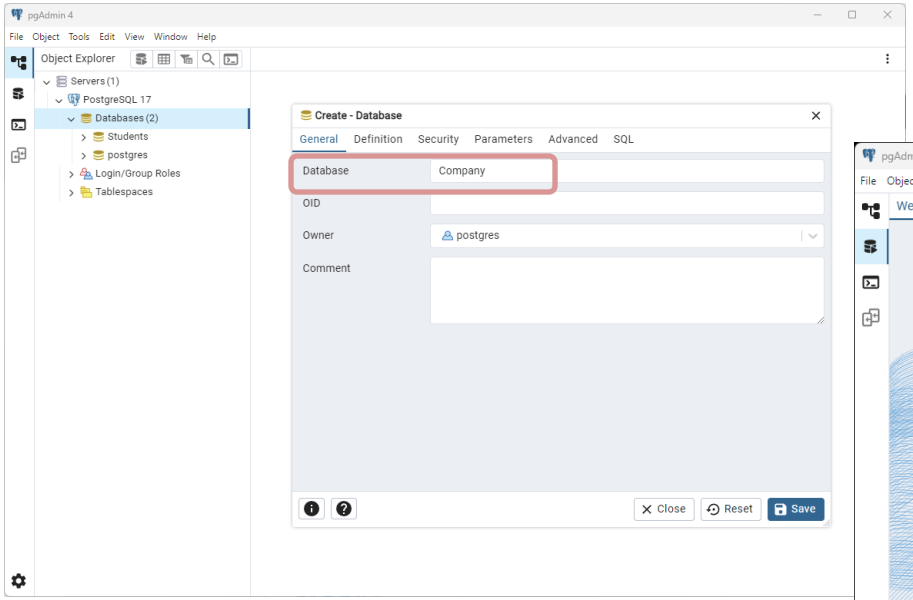
https://www.halvorsen.blog

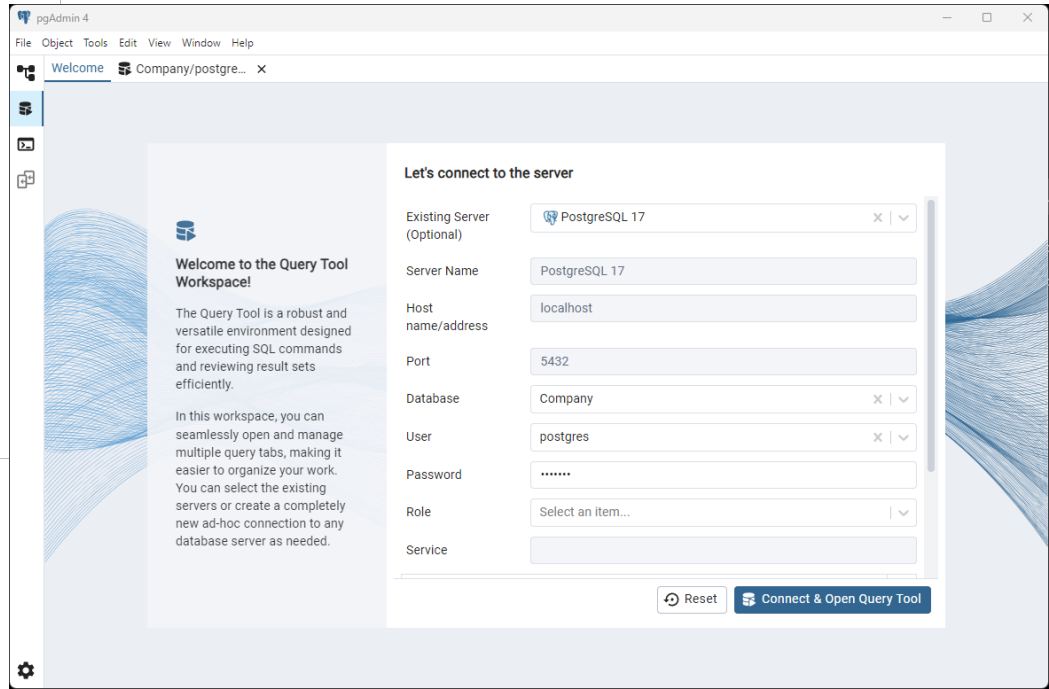
Django + PostgreSQL

Hans-Petter Halvorsen

PostgreSQL

We can create a new Database called “**Company**”

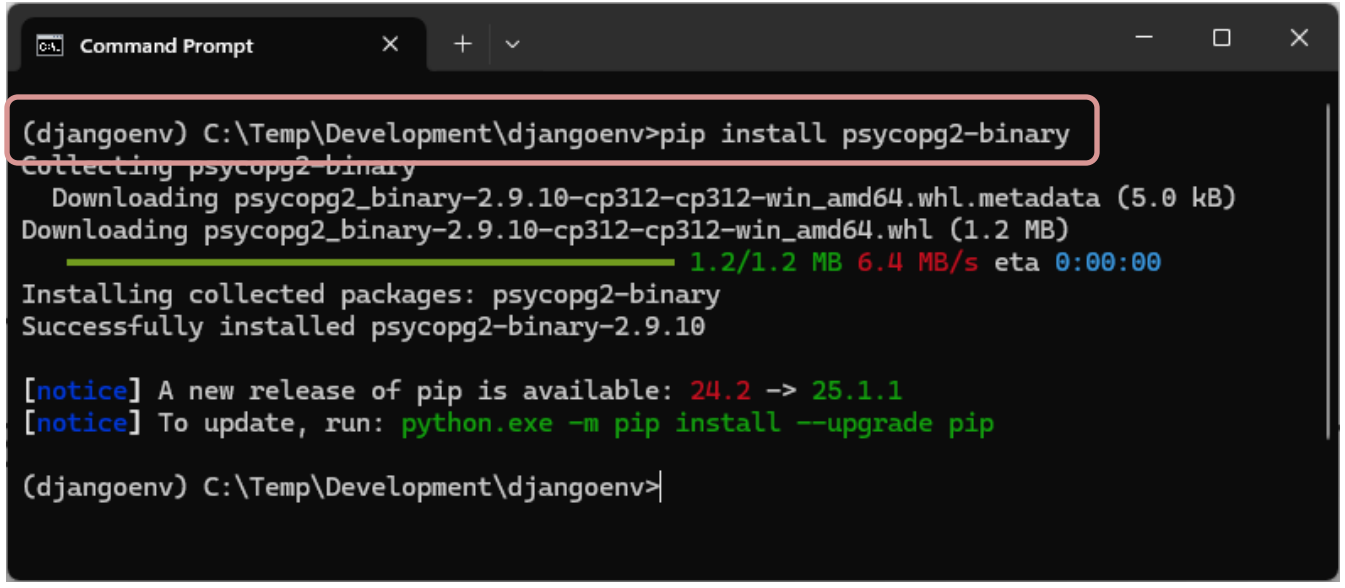




psycopg2

I order to use PostgreSQL with Django we need to install the “**psycopg2**” Python package in your virtual Python Environment where you are using Django.

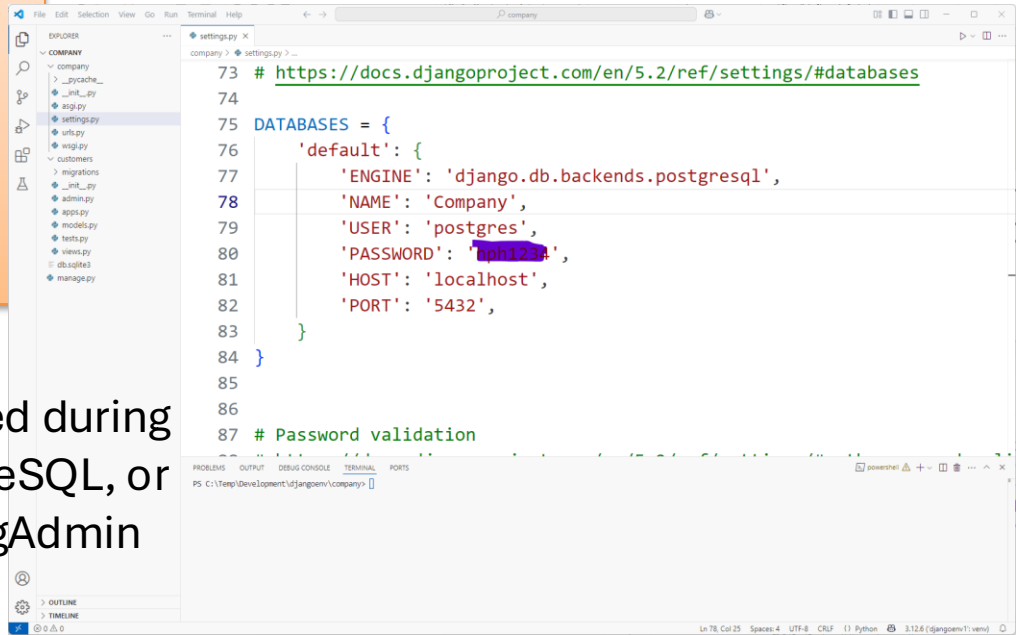
>pip install psycopg2-binary



“settings.py”

Then change the Database Connection Setting in “**settings.py**”. In this file the default database is the SQLite database that comes preinstalled with Django.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'Company',
        'USER': 'postgres',
        'PASSWORD': 'xxxxxx',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```



The Password you created during the installation of PostgreSQL, or a password created in pgAdmin

Migrate

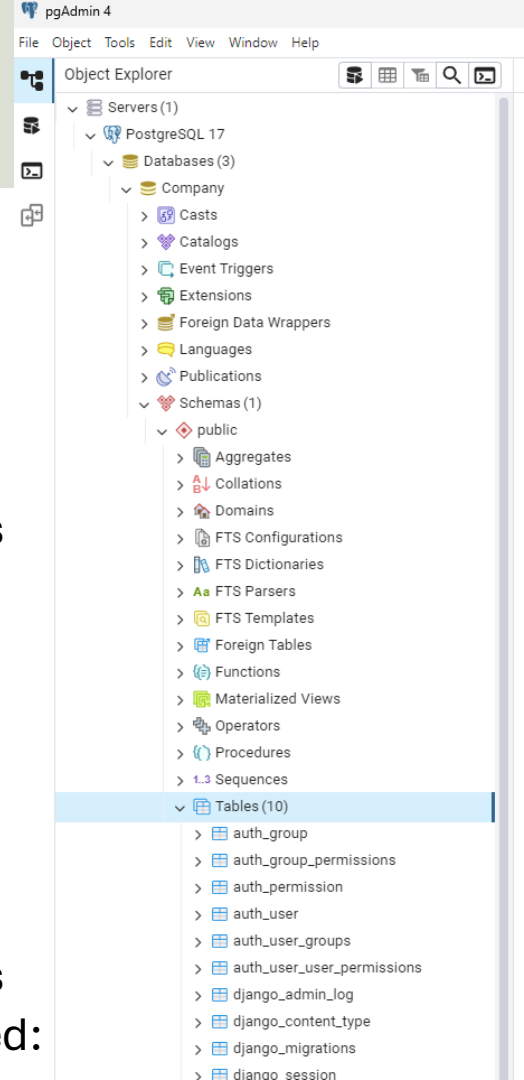
```
..>python manage.py makemigrations
```

```
..>python manage.py migrate
```

```
Command Prompt

(djangoenv) C:\Temp\Development\djangoenv\company>python manage.py makemigrations
No changes detected

(djangoenv) C:\Temp\Development\djangoenv\company>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

(djangoenv) C:\Temp\Development\djangoenv\company>
```

The migration process updates the PostgreSQL database.

Here we see the default Django database tables has been created:



https://www.halvorsen.blog

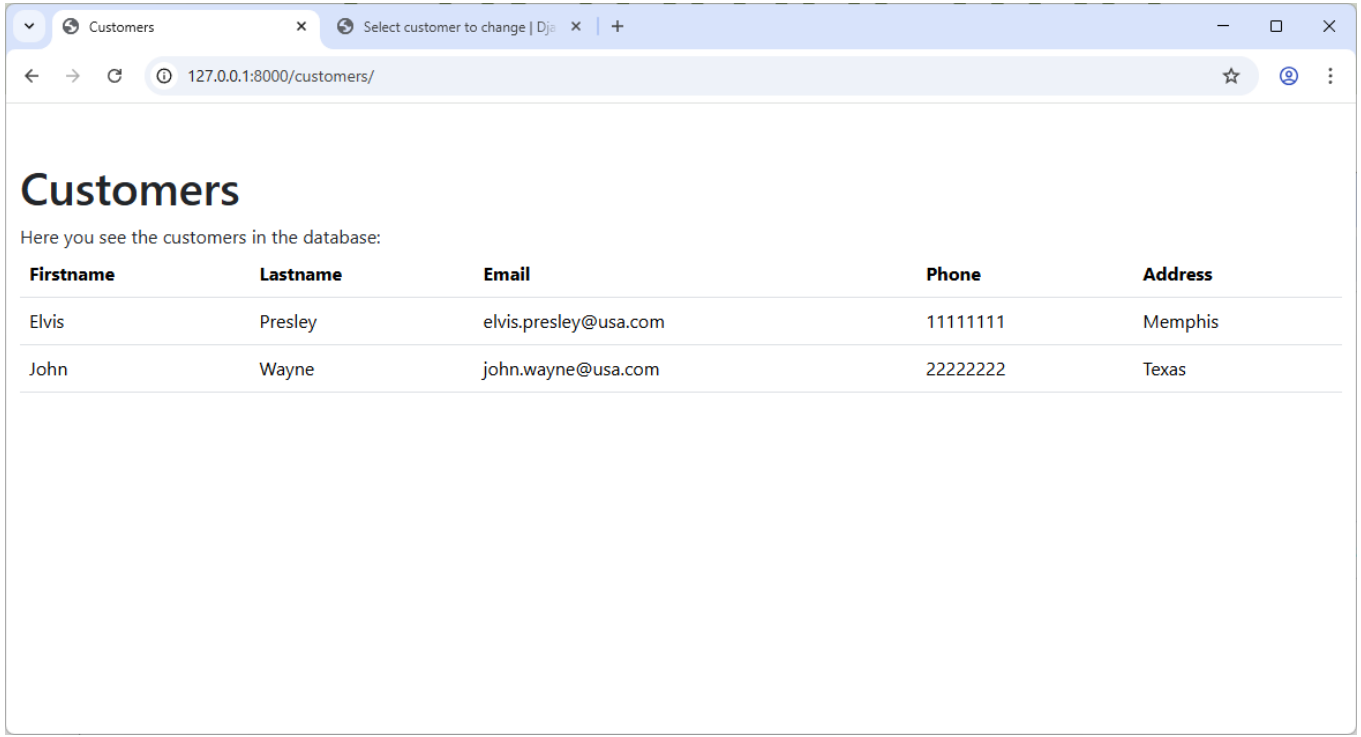
Customers App

Hans-Petter Halvorsen

Table of Contents

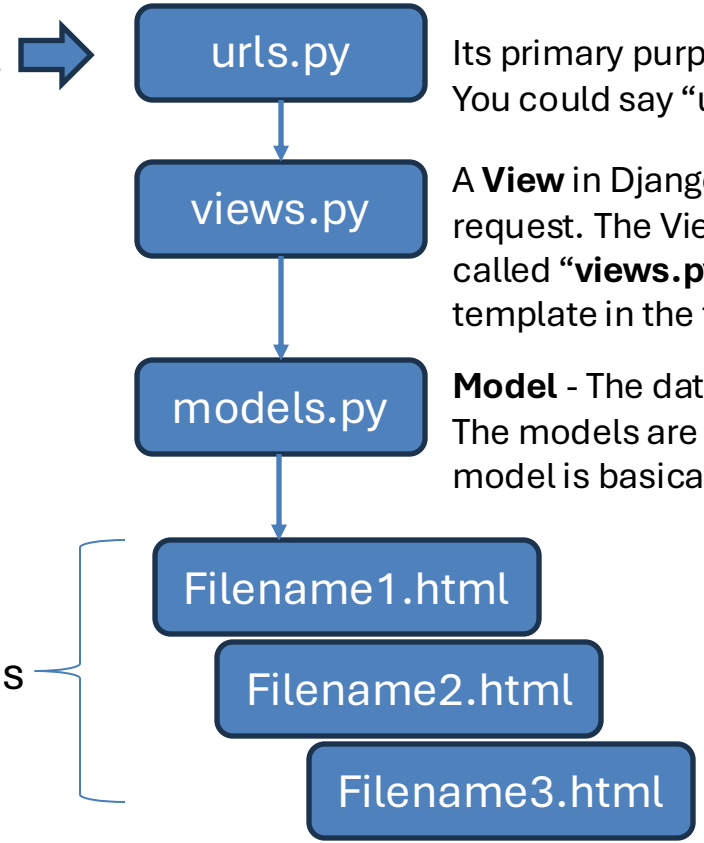
Customer App

We shall create a basic Django Web Application like this where the data are stored in the PostgreSQL database:



Django Files and Folders

URL entered in the Web Browser



Its primary purpose is to map URL patterns to their corresponding views. You could say "urls.py" is the table of contents for the Django project.

A **View** in Django is basically a Python function that handles an HTTP request. The Views (Python Functions) are usually located in a file called "**views.py**". The view then sends the data to a specified template in the template folder.

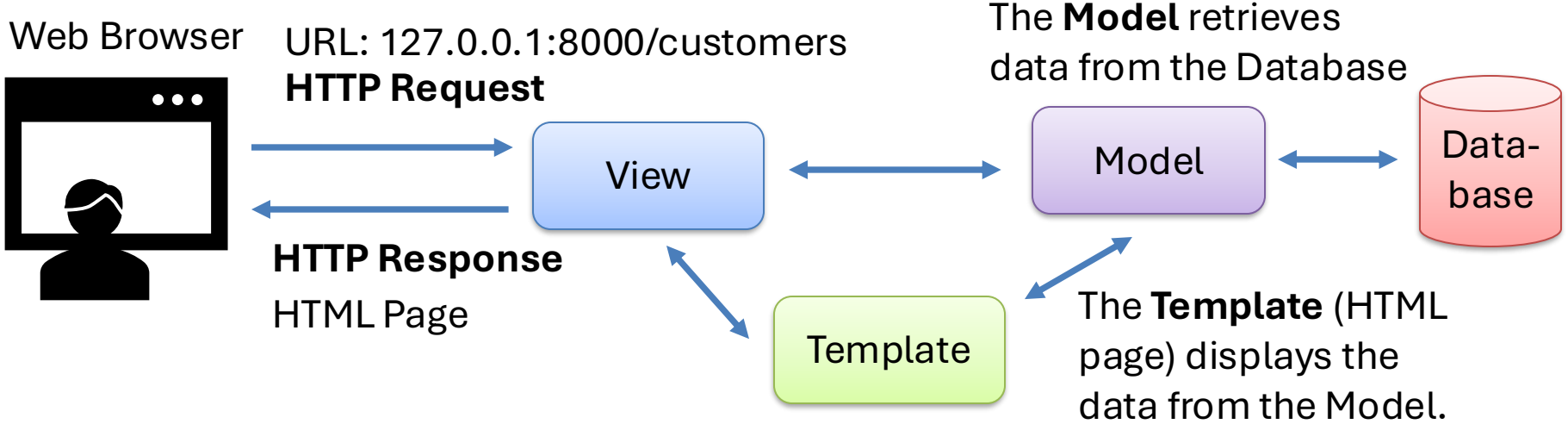
Model - The data you want to present, usually data from a database. The models are usually located in a file called "**models.py**". A Django model is basically a table in your database.

Template Files

Templates - Django Templates are HTML files containing the layout of the web pages, with logic on how to display the data
The HTML template files are stored in a subfolder called "**templates**".

Django workflow

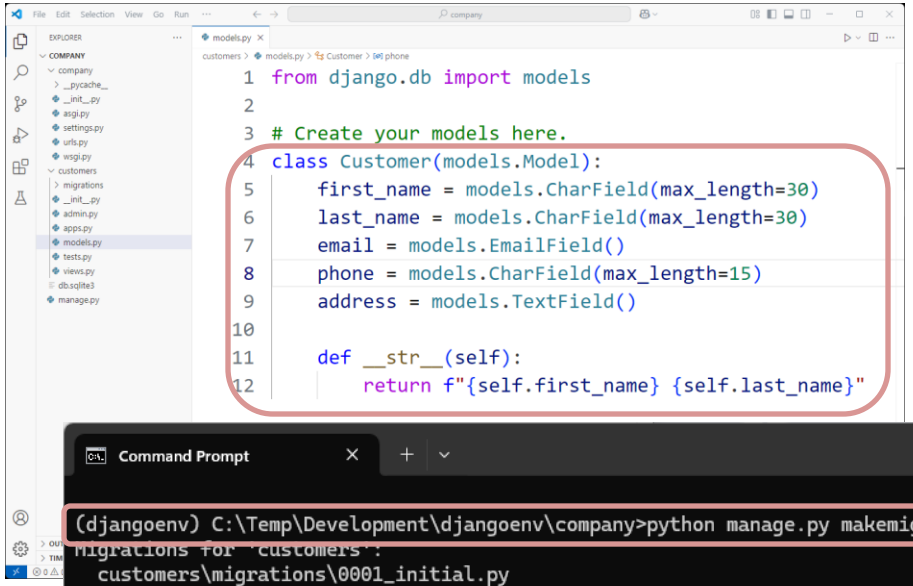
Django find which View to call based on information in “urls.py”.



The **View** handles the HTTP request from the Web Browser and sends data to the specific Template. Then the HTTP response is sent back to the Web Browser.

Create the **Model** (Table)

Let's create a “Customer” Model (Table). We start by creating the Model in “**models.py**”:

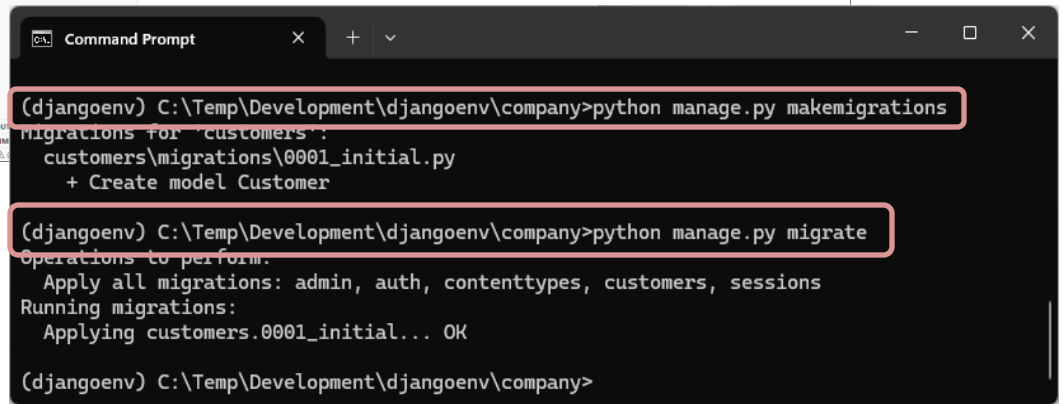


```python
from django.db import models

# Create your models here.
class Customer(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()
    phone = models.CharField(max_length=15)
    address = models.TextField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Then “Migrate” the Model, i.e., create the Table in the PostgreSQL database:

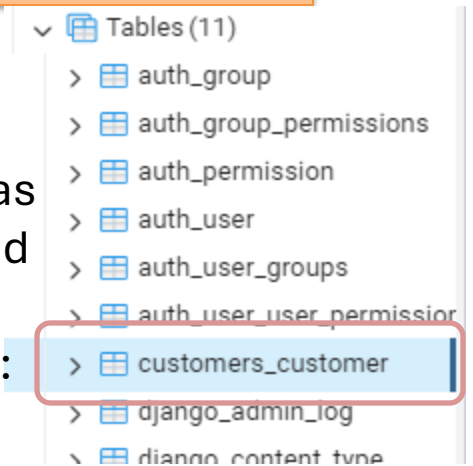
```
...>python manage.py makemigrations
```

```
...>python manage.py migrate
```



```
Command Prompt

(djangoenv) C:\Temp\Development\djangoenv\company>python manage.py makemigrations
Migrations for 'customers':
  customers\migrations\0001_initial.py
    + Create model Customer

(djangoenv) C:\Temp\Development\djangoenv\company>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, customers, sessions
Running migrations:
  Applying customers.0001_initial... OK

(djangoenv) C:\Temp\Development\djangoenv\company>
```

The Table has been created in PostgreSQL:

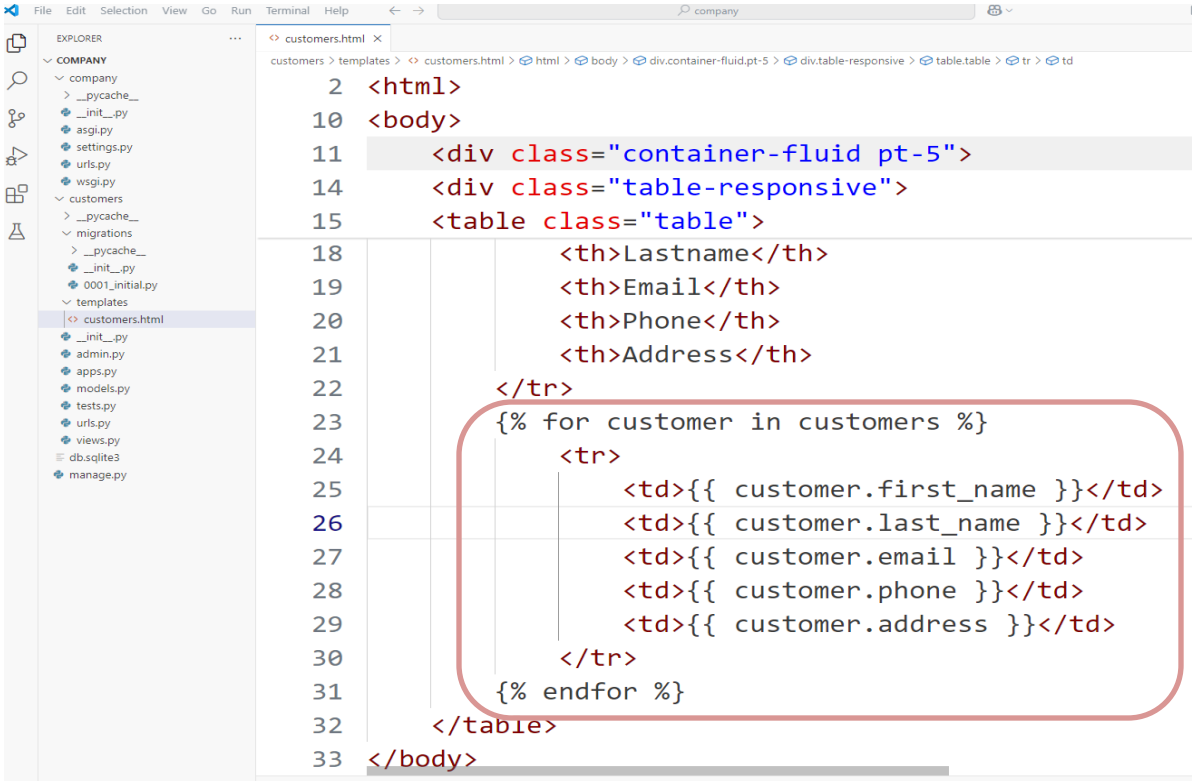


“models.py”

```python
from django.db import models

# Create your models here.
class Customer(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()
    phone = models.CharField(max_length=15)
    address = models.TextField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Create the **Template**

Create a new folder called “**templates**”. Then create a new Template, i.e., a HTML File called “**customers.html**” in the /templates/ folder:



Templates in Django are basically just HTML files.

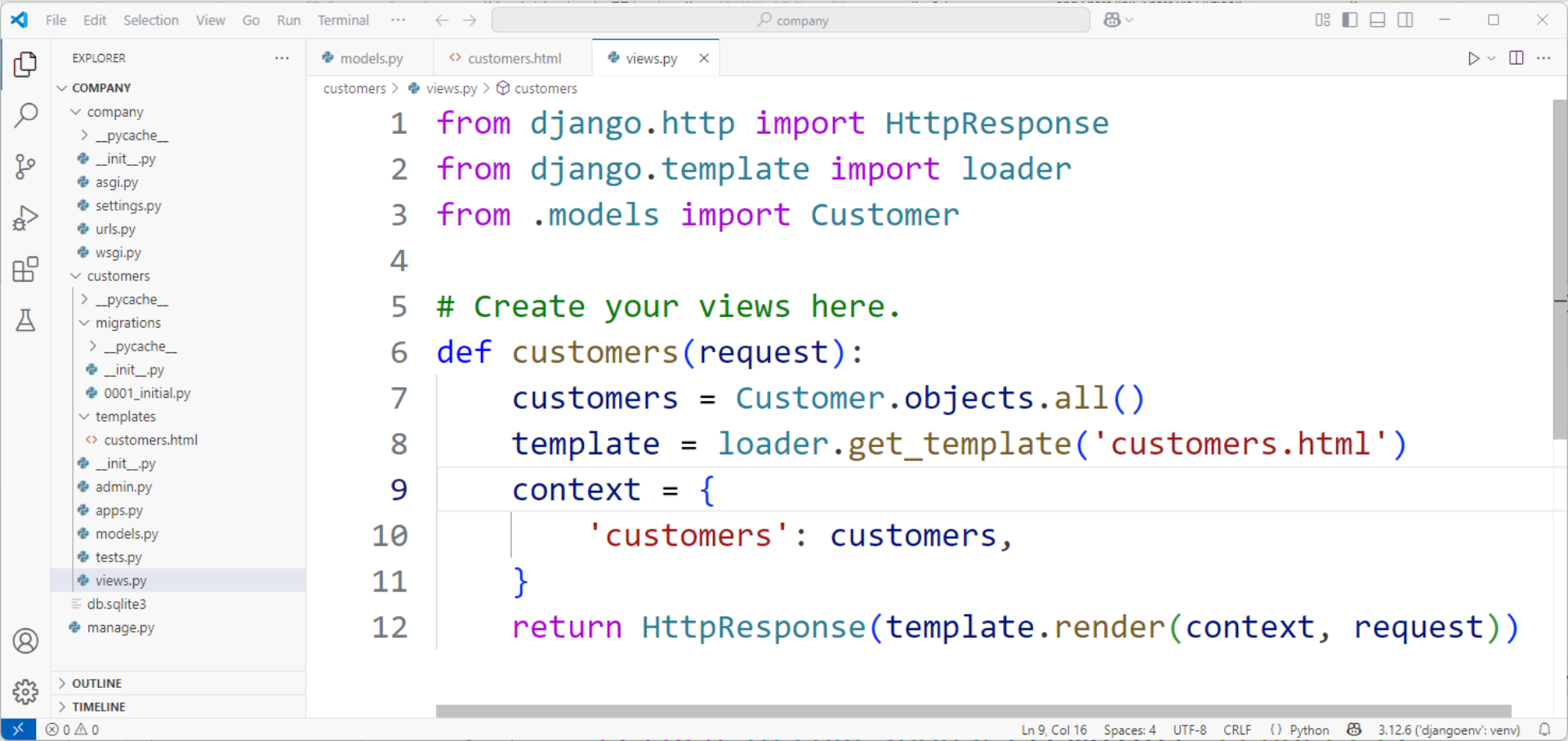
“customers.html”

```
<!DOCTYPE html>
<html>
    <head>
        <title>Customers</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
    </head>
<body>
    <div class="container-fluid pt-5">
    <h1>Customers</h1>
    Here you see the customers in the database:
    <div class="table-responsive">
    <table class="table">
        <tr>
            <th>Firstname</th>
            <th>Lastname</th>
            <th>Email</th>
            <th>Phone</th>
            <th>Address</th>
        </tr>
        {% for customer in customers %}
            <tr>
                <td>{{ customer.first_name }}</td>
                <td>{{ customer.last_name }}</td>
                <td>{{ customer.email }}</td>
                <td>{{ customer.phone }}</td>
                <td>{{ customer.address }}</td>
            </tr>
        {% endfor %}
    </table>
</body>
</html>
```

Note! **Bootstrap** is used to boost up the appearance of the web page.

Django code that retrieves data from the “Customer” Model, which again retrieves data from the customer table in the PostgreSQL database.

Create the **View**



“view.py”

```
from django.http import HttpResponse
from django.template import loader
from .models import Customer

# Create your views here.
def customers(request):
    customers = Customer.objects.all()
    template = loader.get_template('customers.html')
    context = {
        'customers': customers,
    }
    return HttpResponse(template.render(context, request))
```

“urls.py”

Create a “**urls.py**” file for the “**customers**” Django App:

models.py | customers.html | views.py | urls.py customers | urls.py company | settings.py

customers > urls.py > ...

```python
from django.urls import path
from . import views

urlpatterns = [
    path('customers/', views.customers, name='customers'),
]
```

Update the existing “**urls.py**” file for the “**company**” Django Project:

omers | urls.py company

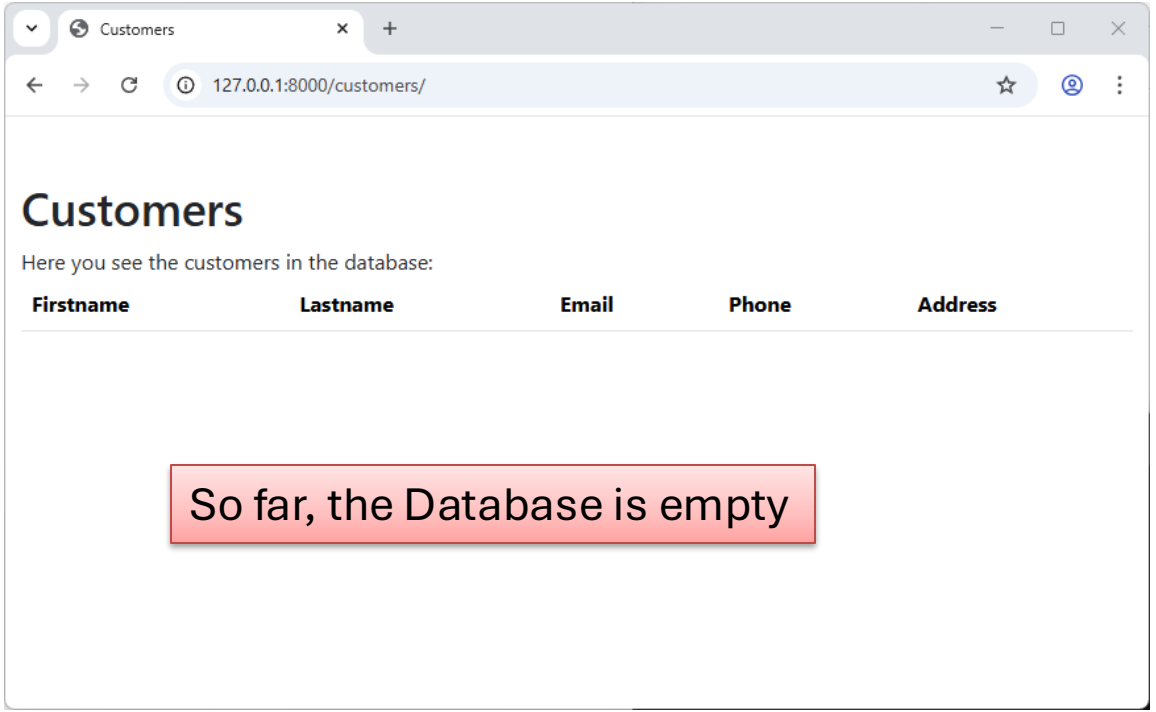
company > urls.py > ...

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('customers.urls')),
    path('admin/', admin.site.urls),
]
```

Customer App

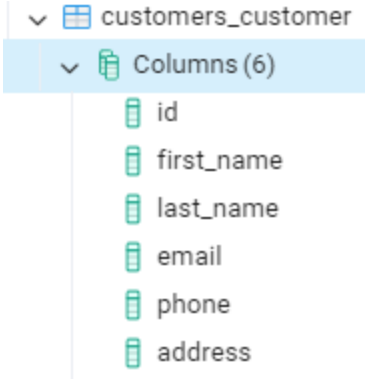
Then run your Django Project: `...>python manage.py runserver`

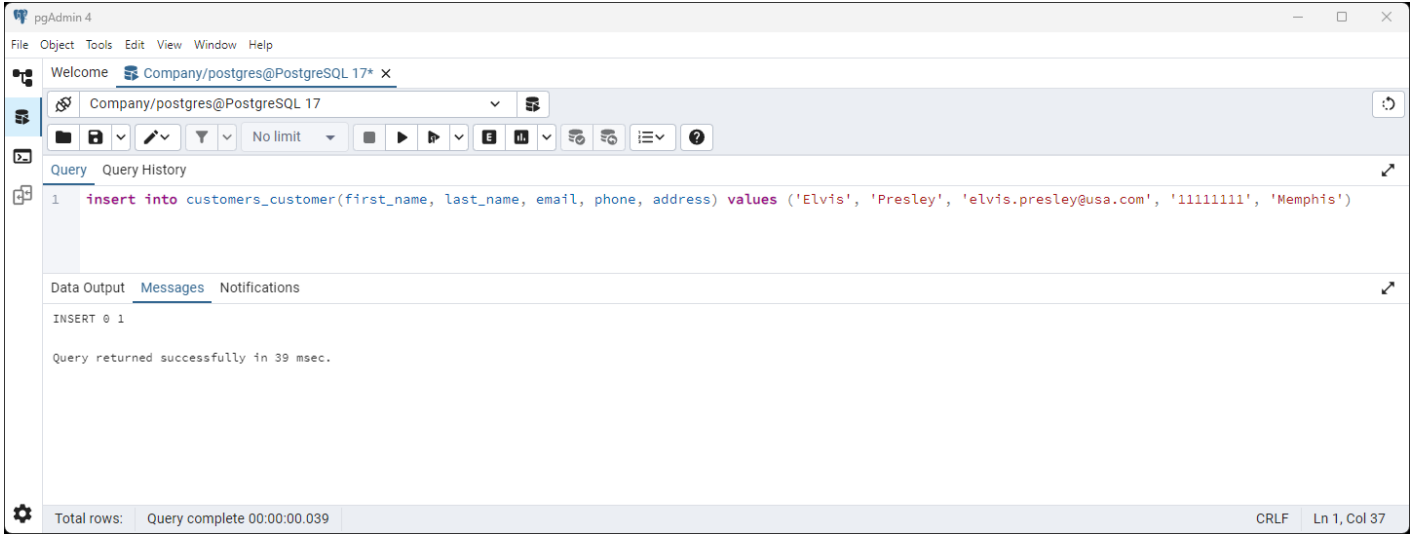
And open your Web Browser and enter the URL: **127.0.0.1:8000/customers**



Add some Data

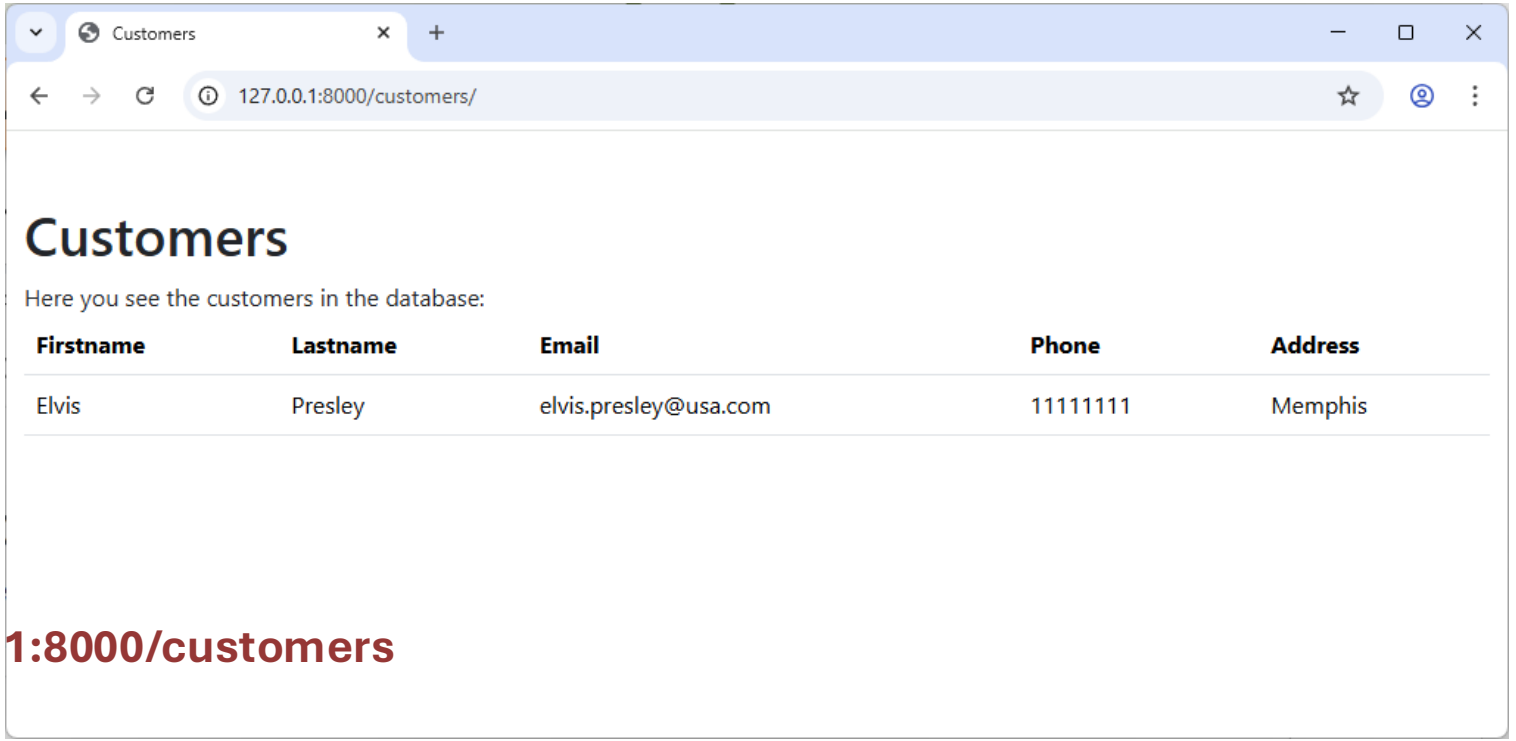
We can use, e.g., pgAdmin to insert some data into the table:





Customer App

We now get data that we inserted in the PostgreSQL database (using “pgAdmin”) in our Customers App:



URL: **127.0.0.1:8000/customers**

https://www.halvorsen.blog

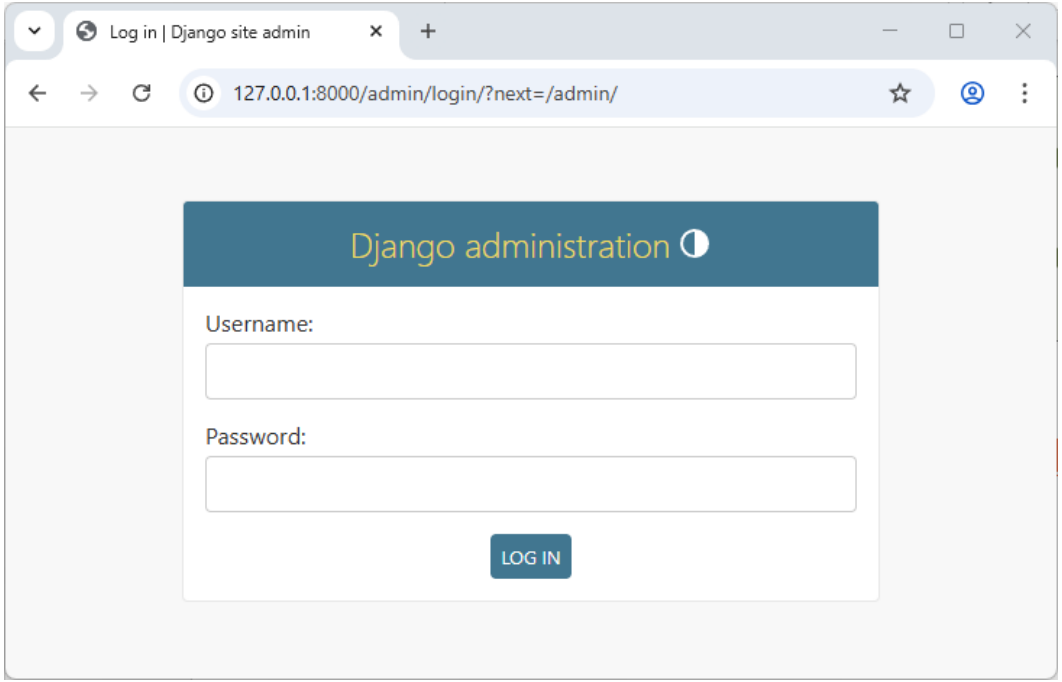
Django Admin

Hans-Petter Halvorsen

Table of Contents

Django Admin

http://127.0.0.1:8000/admin



"Django Admin" interface is included with Django.

Django Admin

- “Django Admin” is included with Django.
- “Django Admin” offers a CRUD user interface for all your Models.
- CRUD is short for Create, Read, Update and Delete.
- Note! The Admin Interface is typically intended for “superusers” and no for ordinary users of your application.
- To use it you need to create a User.

Django Admin - Create User

Type the following to create a new User:

```
...>python manage.py createsuperuser
```

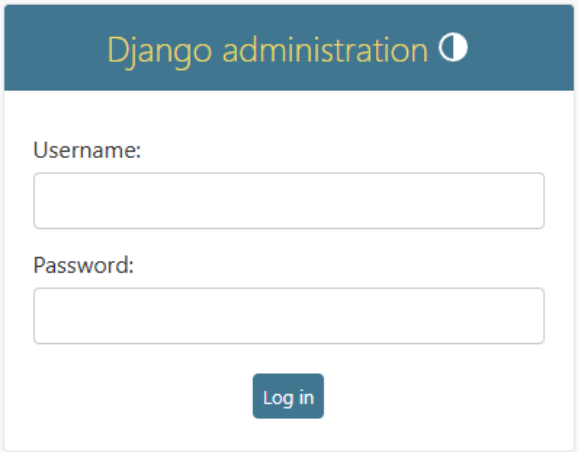
Then you are asked to create a Username and a Password.

```
(djangoenv) PS C:\Temp\Development\djangoenv\company> python manage.py createsuperuser
Username (leave blank to use 'hansp'):
Email address: hans.p.halvorsen@usn.no
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(djangoenv) PS C:\Temp\Development\djangoenv\company>
```

Then run the server:

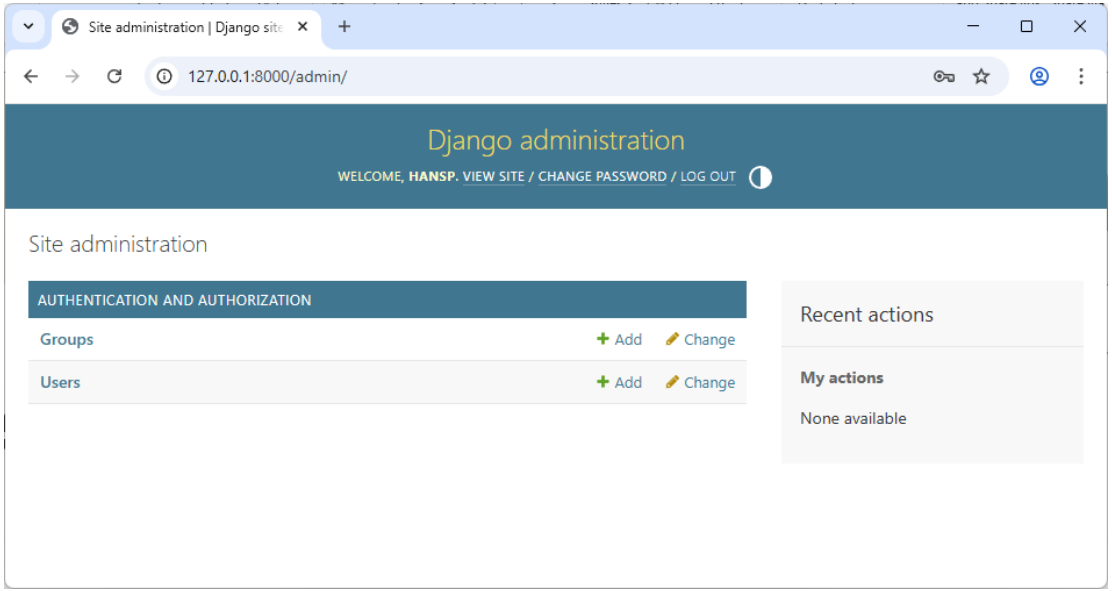
```
...>python manage.py runserver
```

Goto http://127.0.0.1:8000/admin
and enter your Username and Password:



Django Admin

The following will appear:

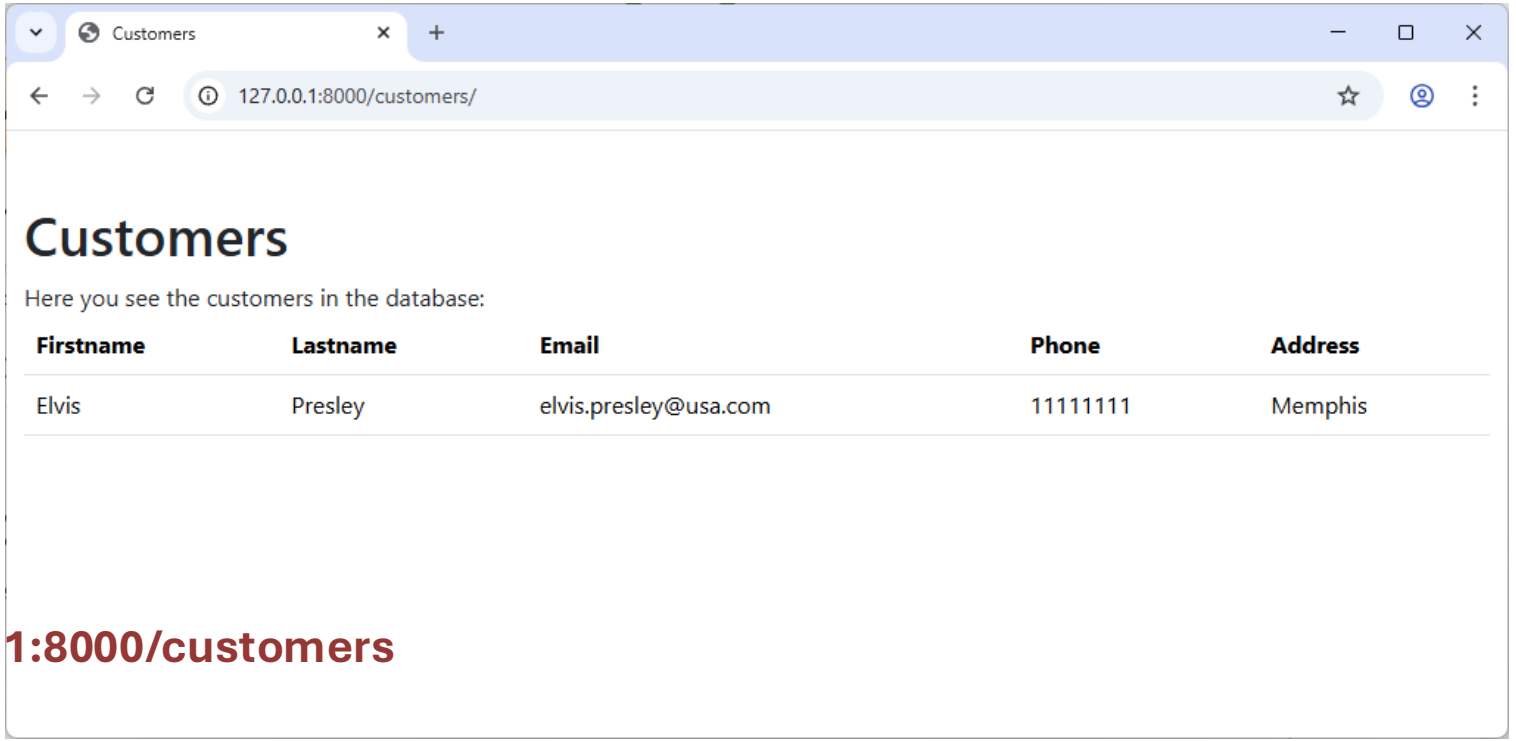


Here you can Add, Change and Delete (CRUD functionality) **Groups** and **Users**.

How can we add Customers?

Customer App

We now get data that we inserted in the PostgreSQL database (using “pgAdmin”) in our Customers App:



URL: **127.0.0.1:8000/customers**

Add Customer Model

Django Admin offers a CRUD user interface for all your Models.
Let's add CRUD functionality for our Customer Model.

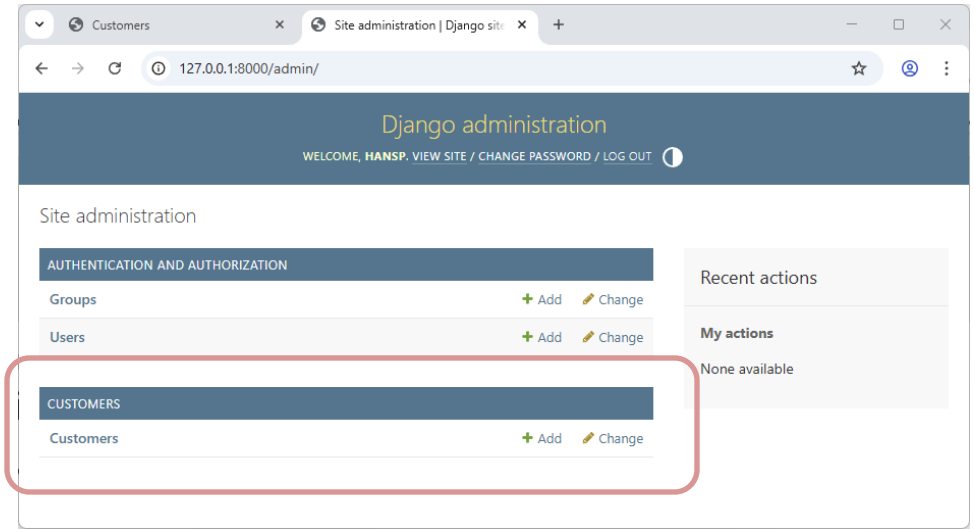
You need to register the Customer Model in the file "**admin.py**":

admin.py

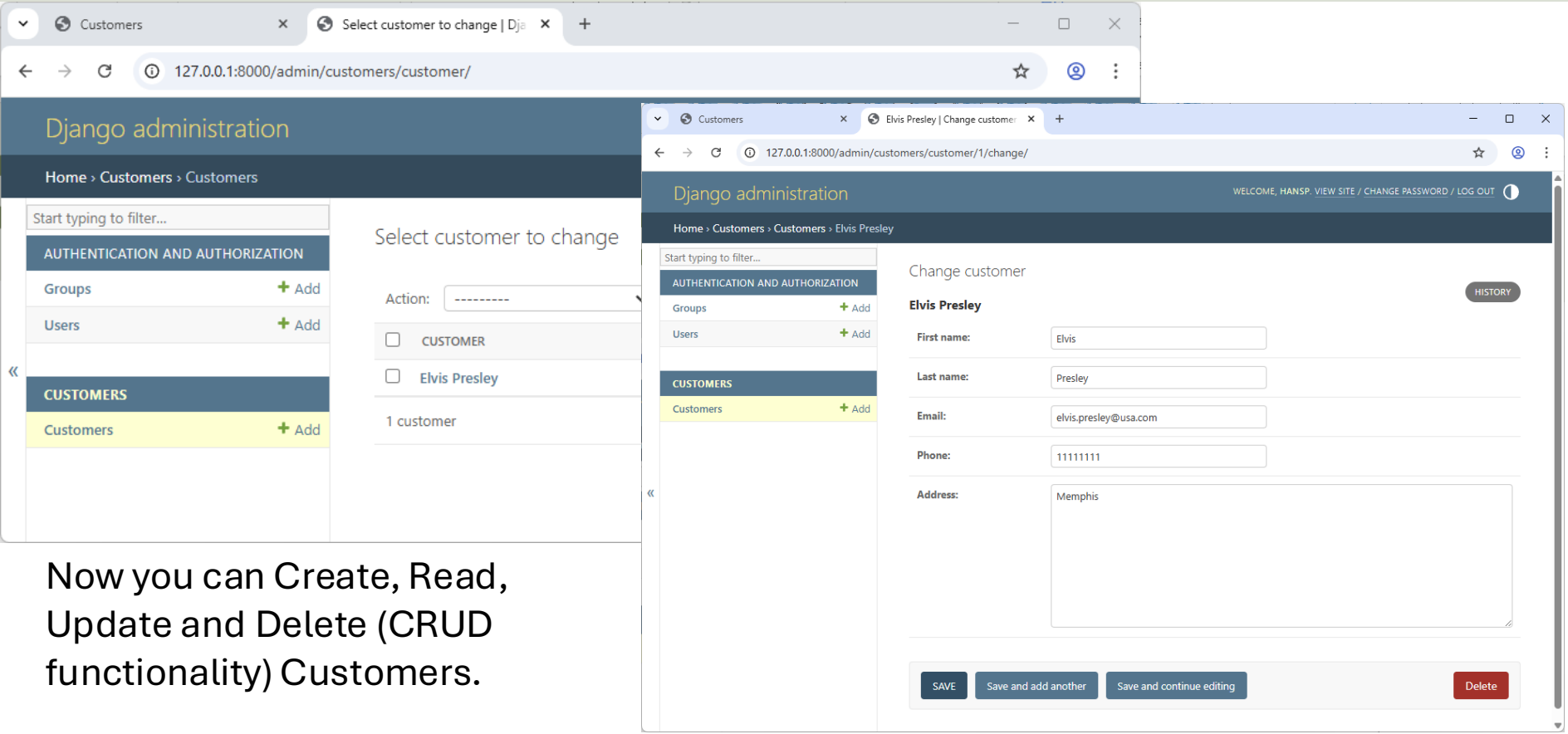
helloworldapp > admin.py

```
1  from django.contrib import admin
2  from .models import Customer
3
4  # Register your models here.
5  admin.site.register(Customer)
```

Then go back to
http://127.0.0.1:8000/admin



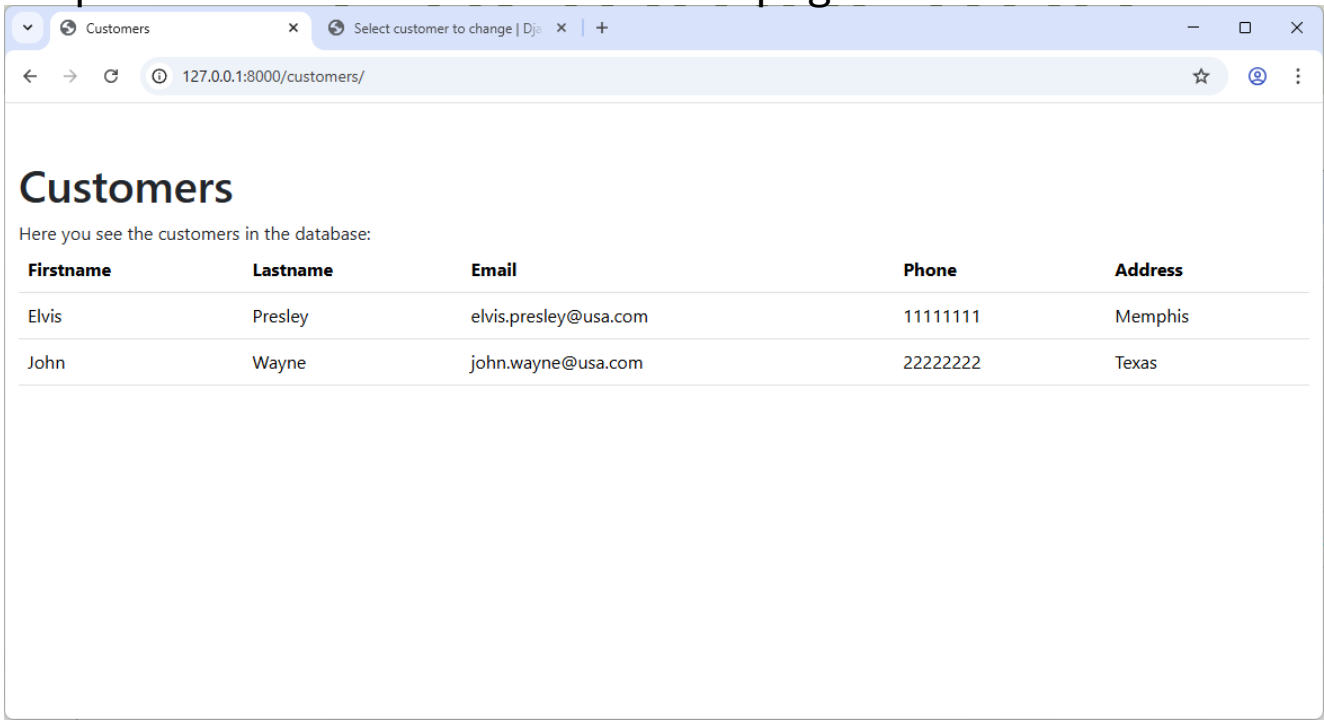
Customer CRUD Interface



Now you can Create, Read, Update and Delete (CRUD functionality) Customers.

Customer App

Here is our final Customer App where the end-user see a list of available Customers, while a superuser or administrator can Create, Read, Update and Delete Customers in the PostgreSQL database and presented in the Customers web page.



Summary

- We created a Django Project and a Django Web Application.
- Django has support for databases like SQLite, MySQL/MariaDB and PostgreSQL.
- Django includes an SQLite database, but for the others you need to install the database system you want to use from their respective websites and in addition install a Python package/driver for the specific database system.
- In this Tutorial we used **Django** in combination with **PostgreSQL**.
- We created a simple "**Customer App**" that retrieves a list of Customers stored in the PostgreSQL database.
- Finally, we used the built-in "**Django Admin**" module to create **CRUD** functionality, i.e., the possibility to Create, Read, Update and Delete Customers in the PostgreSQL database.

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog