

<https://www.halvorsen.blog>

django

Django CRUD Web Application

Hans-Petter Halvorsen



Contents

CRUD is short for **C**reate, **R**ead, **U**ppdate and **D**elate, meaning Create (Insert), Read (Select), Update and Delete data from a Database.

- Introduction to Django and CRUD
 - We will create a **CRUD** Web Application (we will call it “Company” App) with Django from scratch and step by step.
- Create the Django “Company” App
- Read, or Select Data from the Database
- Create, or Insert Data into the Database
- Update existing Data in the Database
- Delete existing Data from the Database
- Django Admin – A built-in feature in Django

Resources

- Django Web Programming (Webpage):
<https://www.halvorsen.blog/documents/programming/web/django.php>
- **Django Tutorial** (YouTube):
<https://www.youtube.com/playlist?list=PLdb-TcK6Aqj0enrpIQ0P4ISI8ofhk8Sza>
- Django Admin (YouTube): :
<https://www.youtube.com/watch?v=hOG1-JiCaG8>
- Django and PostgreSQL (YouTube): :
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj10g4Kn-jwDDVt2r_WwqS4G

<https://www.halvorsen.blog>

django

Django CRUD Web Application

Introduction to Django and CRUD



[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>

CRUD

- **CRUD** is short for **C**reate, **R**ead, **U**ppdate and **D**eflete, meaning create/insert, read/select, update and delete data from a database.
- All applications today typically have or need to have CRUD functionality.
- “Django Admin” interface offers a CRUD user interface for all your Django Models.
- “Django Admin” is included with Django, and you can use it “out of the box”.
- The “Django Admin“ interface is intended for “superusers” and not for ordinary users of your application.
- Very often we need to create our own CRUD functionality that suits our needs and have the same look and feel as the rest of the application.
- So, the focus is this tutorial is to create CRUD functionality from “scratch”.

Django and Databases

- Django has support for databases like SQLite, MySQL/MariaDB and PostgreSQL.
- In Django, the data is delivered as an **Object Relational Mapping (ORM)**, which is a technique designed to make it “easier” to work with databases without using complex SQL statements. It’s a bit like the “Entity Framework” used in C#.
- Django includes an **SQLite** database and has built-in support for this.
- For the others you need to install the database system you want to use from their respective websites.
 - In addition, you need to install a Python package/driver for the specific database system.
- For small applications SQLite is good enough, but for larger applications with lots of data and many users’ a database like MySQL/MariaDB or PostgreSQL should be used.
- You can start with SQLite and then easily change to another Database System later, because in Django you don’t need to deal with database specific details. The Python/Django code will remain the same even if you switch from one database system to another.

CRUD

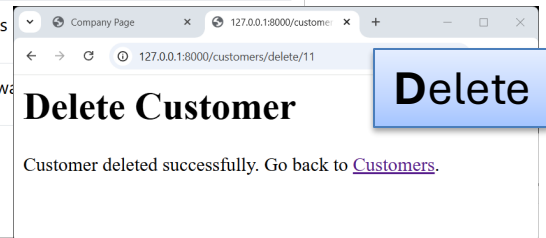
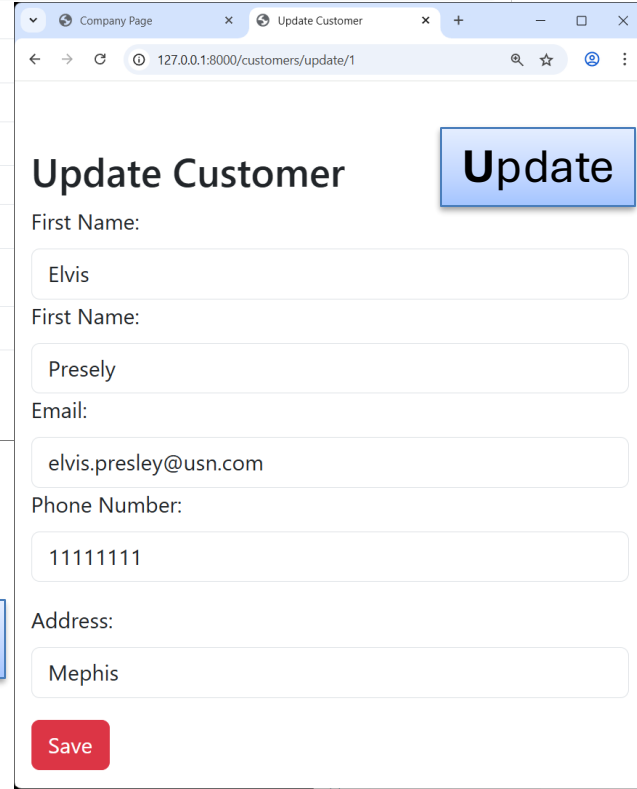
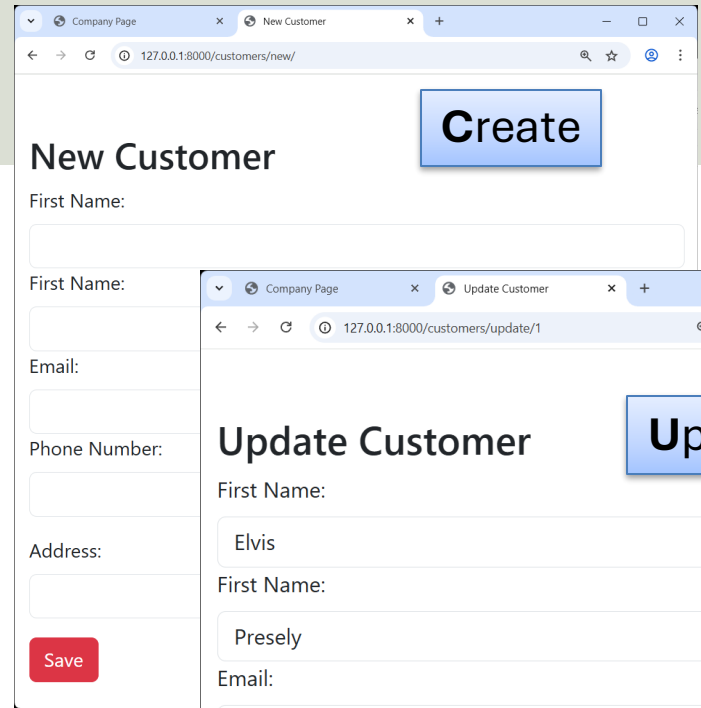
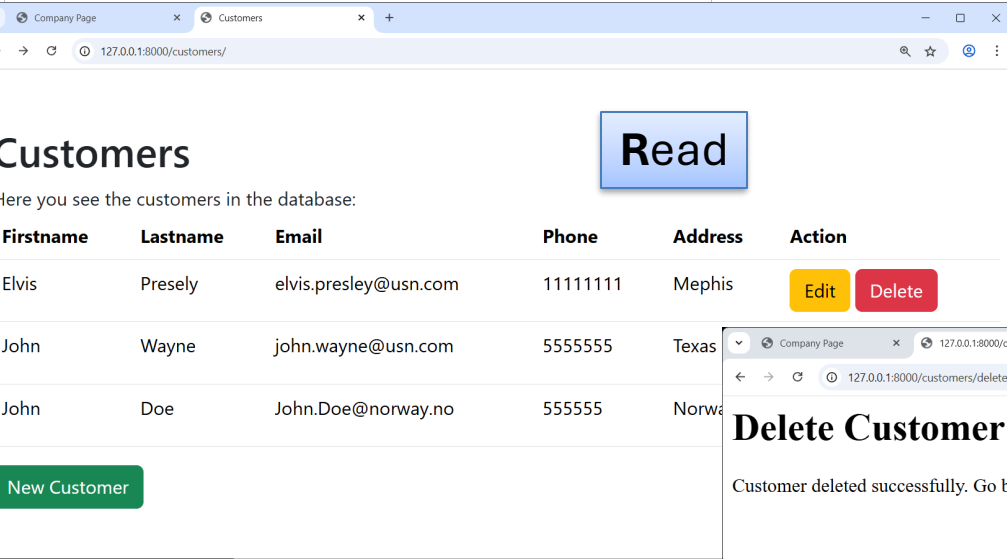
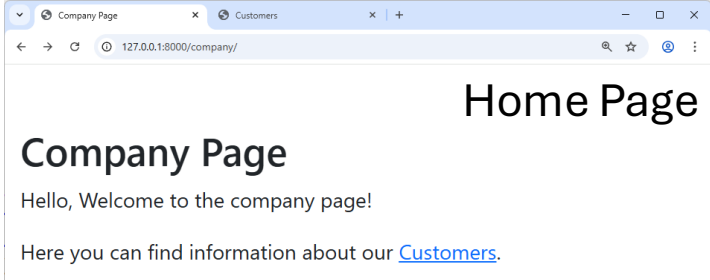
CRUD is short for:

- **C**reate or Insert Data into the Database
- **R**ead or Select Data from the Database
- **U**ppdate existing Data in the Database
- **D**elete existing Data in the Database

=> In this Tutorial we will create a Django Web Application with CRUD functionality. We will call it “Company” App.

“Company” App

We will create a Django Web Application with **CRUD** functionality:



<https://www.halvorsen.blog>



Django CRUD Web Application

Create Django "Company" App

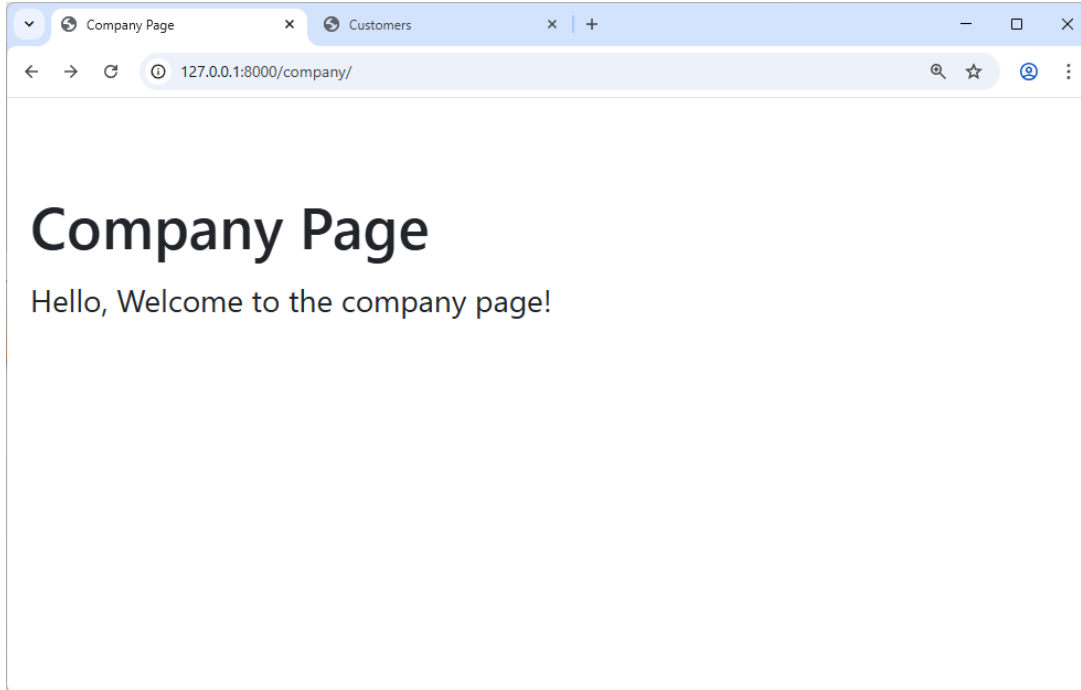


[Table of Contents](#)

Hans-Petter Halvorsen

Django “Company” App

We start by creating a basic “Company” App with a start/home page like this:



Then we will add the CRUD functionality step by step from scratch.

Create Django Project and App

Step 1. Create and Activate **Virtual Python Environment**:

```
>python -m venv venvname
```

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

Step 2. Install **Django**:

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

Step 3. Create **Django Project**:

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

Step 4. Run the Django Project:

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

Step 5. Create a **Django App**:

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

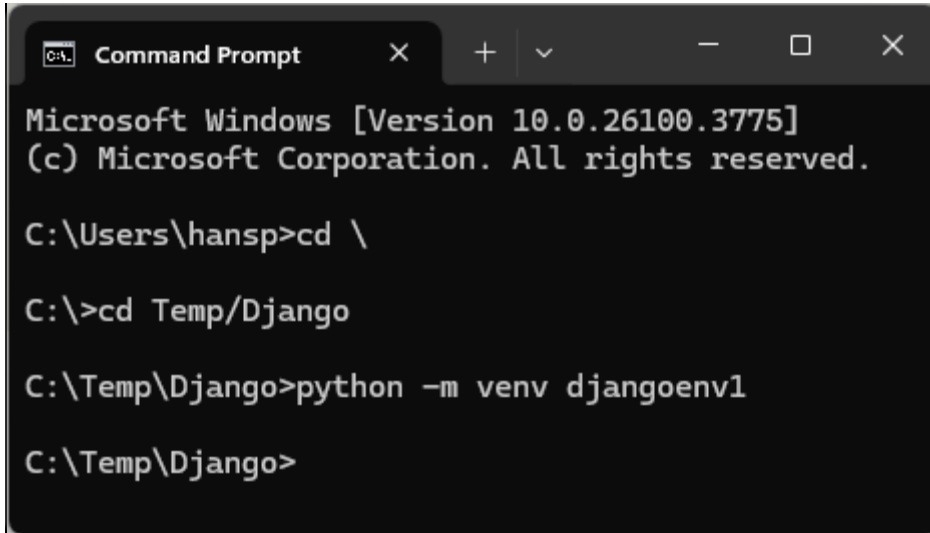
We need to do these steps to create a **Virtual Python environment** (which is recommended), then install Django. Then you need to create a new **Django Project** and a **Django App**.

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

Create **Virtual** Python Environment

We start by creating a Folder where we want to create the Virtual Python Environment and the Django Application.

For example, you can create a folder C:/Temp/Django/



```
Command Prompt
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hansp>cd \

C:\>cd Temp/Django

C:\Temp\Django>python -m venv djangoenv1

C:\Temp\Django>
```

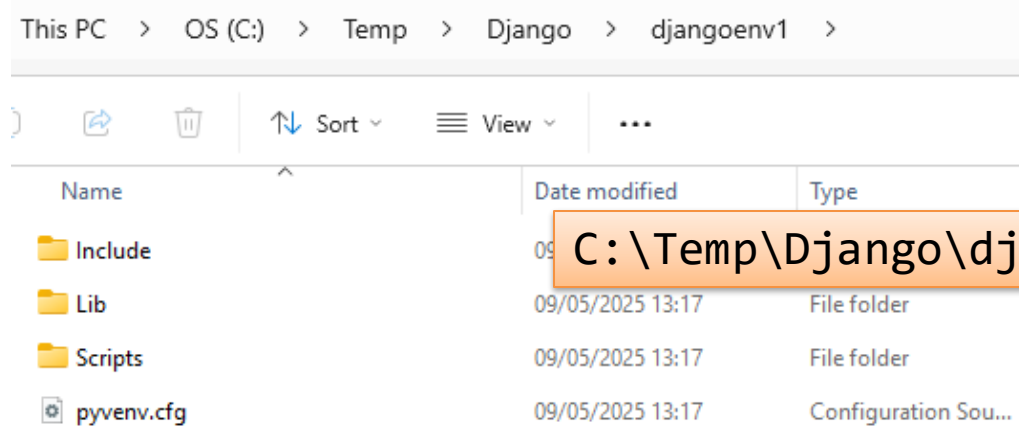
`>python -m venv djangoenv1`



This command will create a Virtual Python Environment called “djangoenv1”.

Activate Virtual Python Environment

Here we see the files created as part of the Virtual Python Environment:



C:\Temp\Django\djangoenv1\Scripts>activate.bat

Next, we need to **activate** the Virtual Python Environment by running the “**activate.bat**” file located in the **/Scripts** folder:

Note! You need to activate it every time you shall work on your project from the command prompt.



```
Command Prompt
C:\Users\hansp>cd \
C:\>cd Temp/Django
C:\Temp\Django>python -m venv djangoenv1
C:\Temp\Django>cddjangoenv1
'cddjangoenv1' is not recognized as an internal or external command,
operable program or batch file.
C:\Temp\Django>cd djangoenv1
C:\Temp\Django\djangoenv1>cd Scripts
C:\Temp\Django\djangoenv1\Scripts>activate.bat
(djangoenv1) C:\Temp\Django\djangoenv1\Scripts>
```

Install Django

```
(djangoenv1)..>python -m pip install Django
```

```
Command Prompt
C:\Temp\Django\djangoenv1>cd Scripts
C:\Temp\Django\djangoenv1\Scripts>activate.bat
(djangoenv1) C:\Temp\Django\djangoenv1\Scripts>cd ..
(djangoenv1) C:\Temp\Django\djangoenv1>python -m pip install Django
Collecting Django
  Using cached django-5.2.1-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<=3.8.1 (from Django)
  Using cached asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from Django)
  Using cached sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting tzdata (from Django)
  Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Using cached django-5.2.1-py3-none-any.whl (8.3 MB)
Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
Using cached sqlparse-0.5.3-py3-none-any.whl (44 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-5.2.1 asgiref-3.8.1 sqlparse-0.5.3 tzdata-2025.2

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(djangoenv1) C:\Temp\Django\djangoenv1>
```

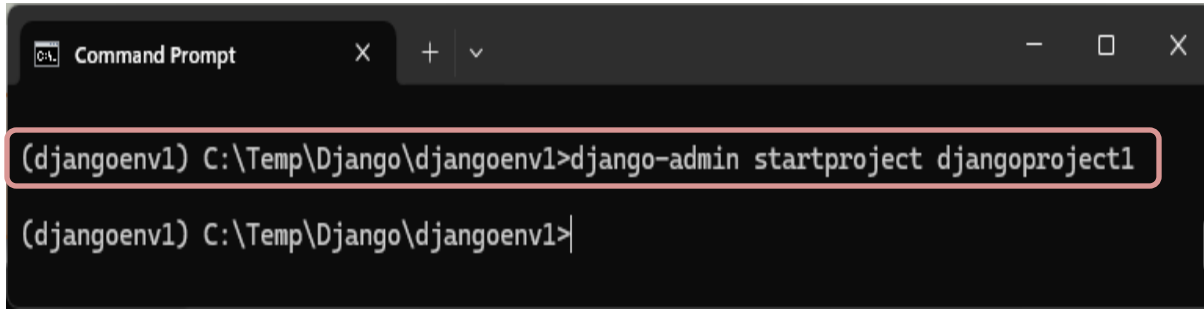
This PC > OS (C:) > Temp > Django > djangoenv1 > Lib > site-packages >

Name	Date modified	Type	Size
asgiref	09/05/2025 13:21	File folder	
asgiref-3.8.1.dist-info	09/05/2025 13:21	File folder	
django	09/05/2025 13:21	File folder	
django-5.2.1.dist-info	09/05/2025 13:21	File folder	
pip	09/05/2025 13:17	File folder	
pip-24.2.dist-info	09/05/2025 13:17	File folder	
sqlparse	09/05/2025 13:21	File folder	
sqlparse-0.5.3.dist-info	09/05/2025 13:21	File folder	
tzdata	09/05/2025 13:21	File folder	
tzdata-2025.2.dist-info	09/05/2025 13:21	File folder	

We see that Django is installed:

Create a Django Project

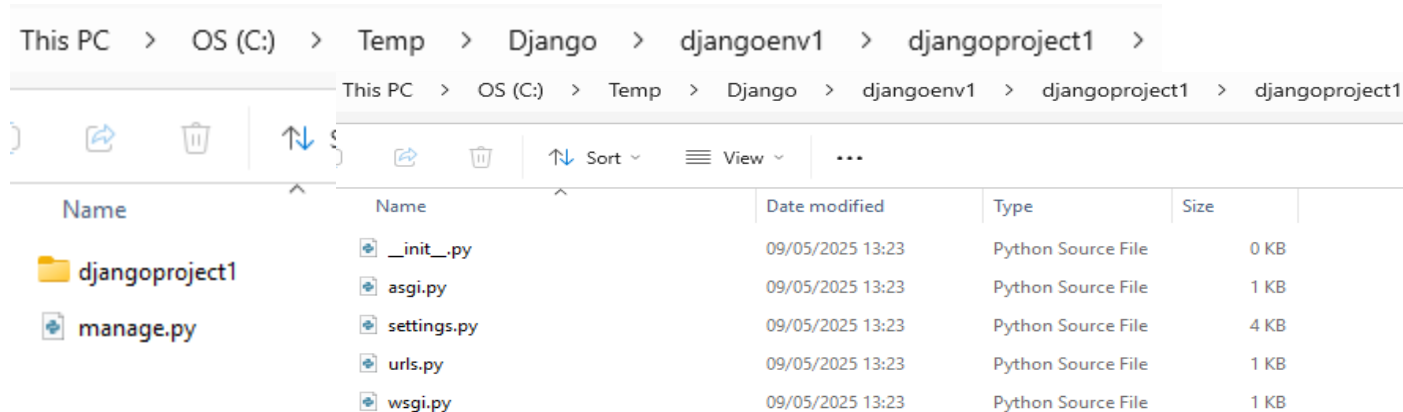
```
(djangoenv1) C:\>django-admin startproject.djangoproject1
```



```
Command Prompt
(djangoenv1) C:\Temp\Django\djangoenv1>django-admin startproject.djangoproject1
(djangoenv1) C:\Temp\Django\djangoenv1>
```

This command will create a Django Project called “djangoproject1”.

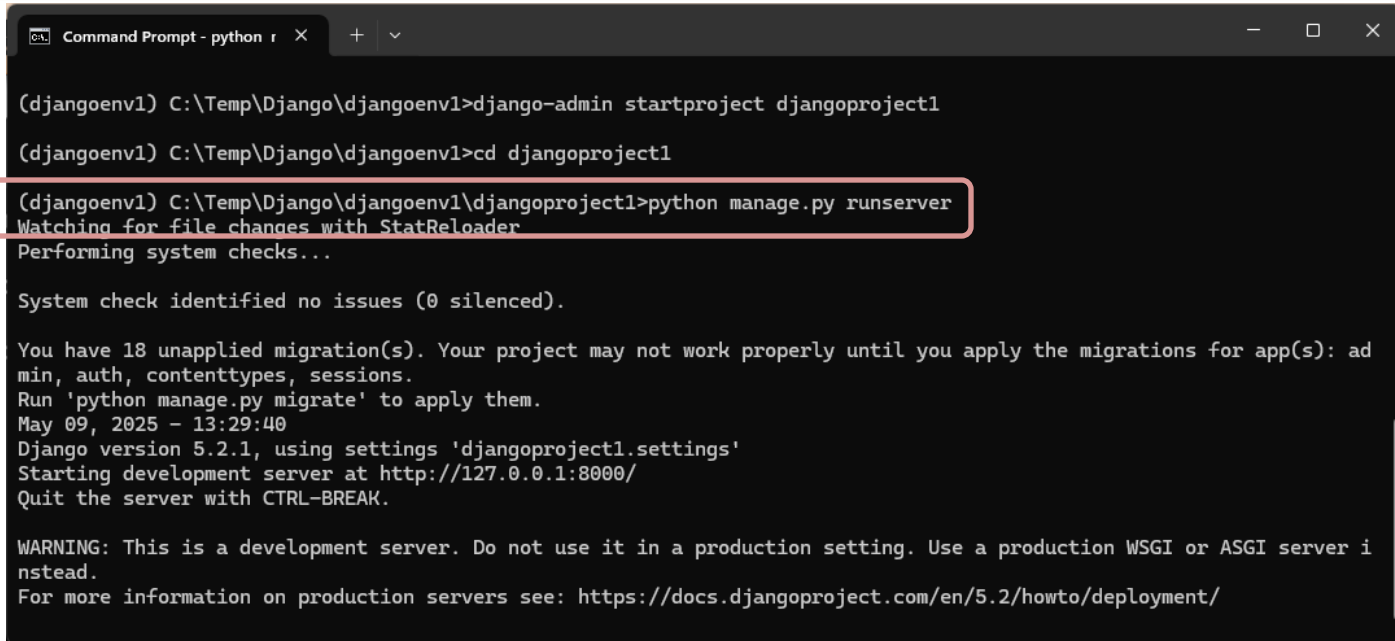
The following folders and files are created:



Name	Name	Date modified	Type	Size
Folder	djangoproject1			
File	manage.py			
File	__init__.py	09/05/2025 13:23	Python Source File	0 KB
File	asgi.py	09/05/2025 13:23	Python Source File	1 KB
File	settings.py	09/05/2025 13:23	Python Source File	4 KB
File	urls.py	09/05/2025 13:23	Python Source File	1 KB
File	wsgi.py	09/05/2025 13:23	Python Source File	1 KB

Run the Django Project

```
(djangoenv1) C:\Temp\Django\djangoenv1\django\project1>python manage.py runserver
```



```
Command Prompt - python r x + v
(djangoenv1) C:\Temp\Django\djangoenv1>django-admin startproject django\project1
(djangoenv1) C:\Temp\Django\djangoenv1>cd django\project1
(djangoenv1) C:\Temp\Django\djangoenv1\django\project1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

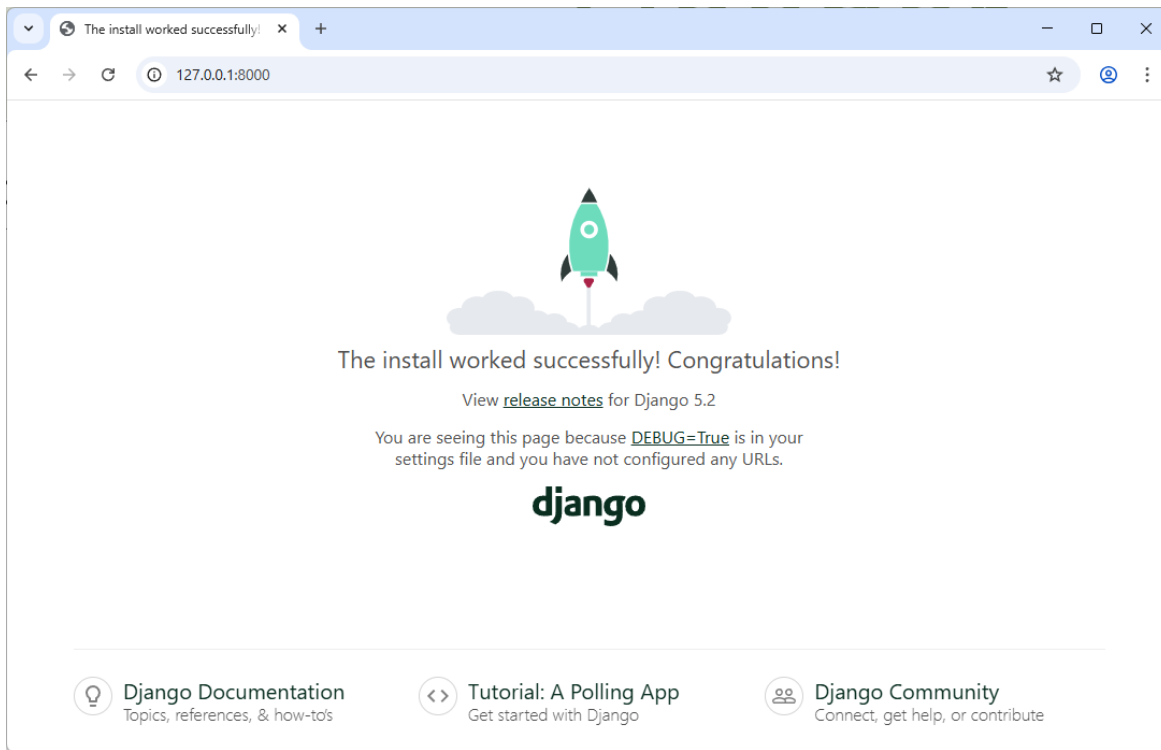
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 09, 2025 - 13:29:40
Django version 5.2.1, using settings 'django\project1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Then open your Web Browser and enter the URL: **127.0.0.1:8000**

Run the Django Project

Open your Web Browser and enter the URL: **127.0.0.1:8000**

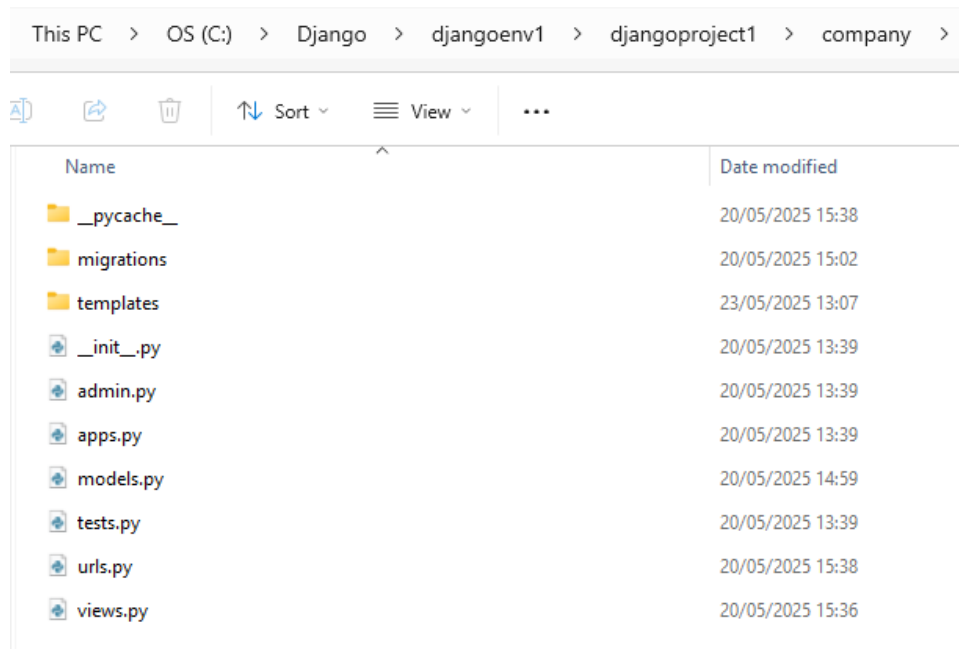


Create a Django App

```
(djangoenv1) C:\>python manage.py startapp company
```



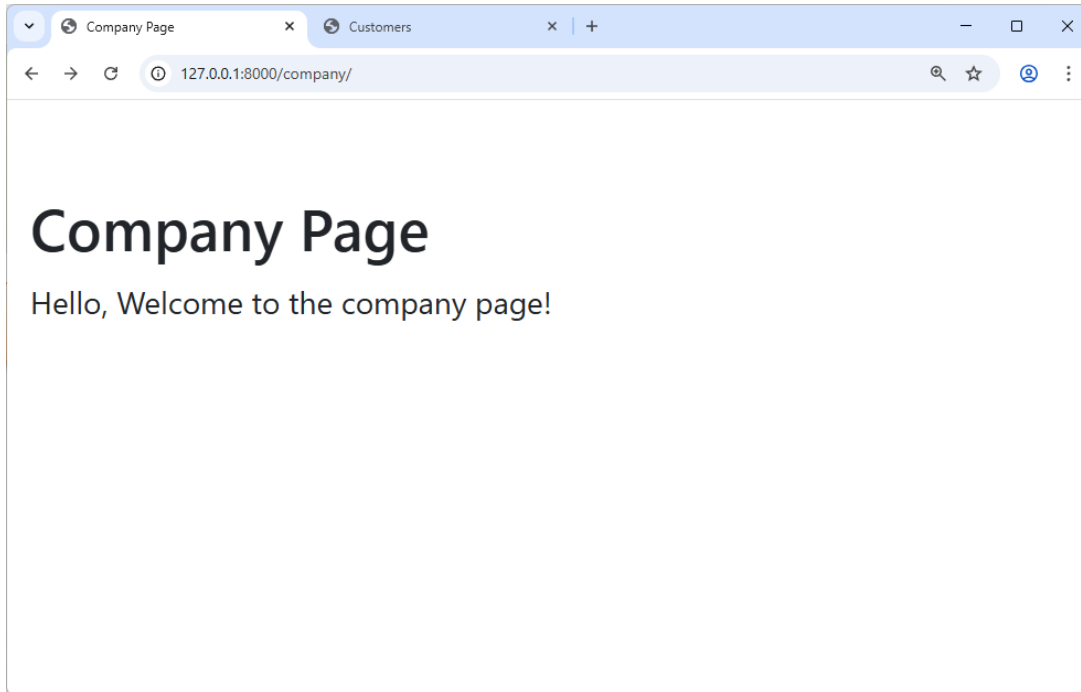
This command will create a Django App called “company”.



Name	Date modified
__pycache__	20/05/2025 15:38
migrations	20/05/2025 15:02
templates	23/05/2025 13:07
__init__.py	20/05/2025 13:39
admin.py	20/05/2025 13:39
apps.py	20/05/2025 13:39
models.py	20/05/2025 14:59
tests.py	20/05/2025 13:39
urls.py	20/05/2025 15:38
views.py	20/05/2025 15:36

Django “Company” App

We start by creating a basic “Company” App with a start/home page like this:



Then we will add the CRUD functionality step by step from scratch.

Create Template (“company.html”)

We start by creating a Template called “**company.html**”:

```
<!DOCTYPE html>
<html>
<body>
  <h1>Company page</h1>
  <p>Hello, Welcome to the company page!>
</body>
</html>
```

We create a subfolder called /templates and put the “company.html” inside that folder.

Create View (“views.py”)

We create a View called “**company**” in “**views.py**”:

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import loader

def company(request):
    template = loader.get_template('company.html')
    return HttpResponse(template.render())
```

Update urls.py

We add the “company” view in the “urls.py” for “company” Django App:

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

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

We also need to include the “company” in “urls.py” for “django project1” Django Project:

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

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

Update Settings (settings.py)

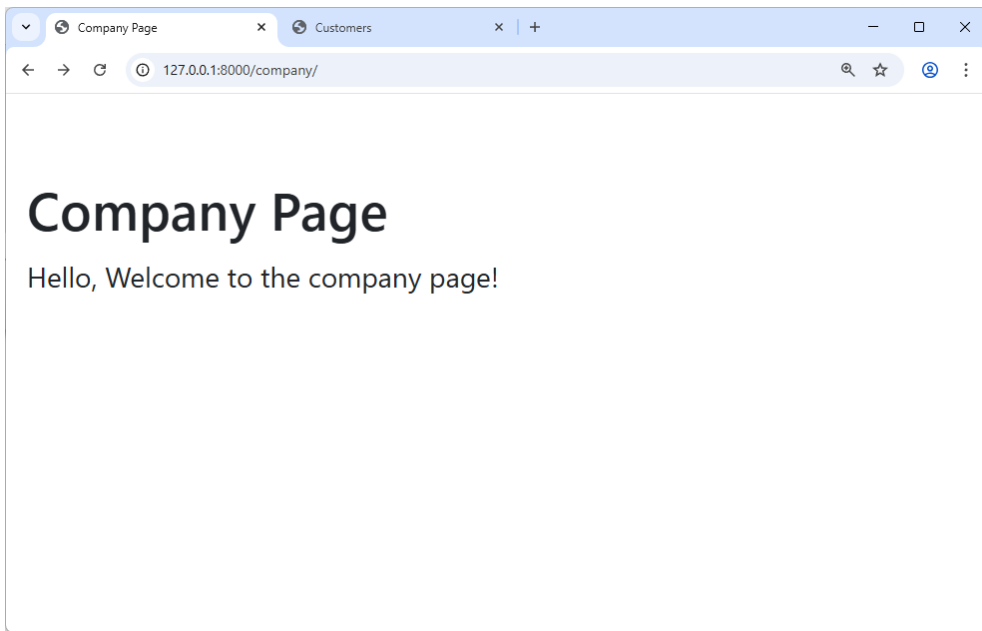
Then we need to modify the “settings.py” file:

```
settings.py ×
django project1 > settings.py > ...
31 # Application definition
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     'company'
41 ]
```

Run “Company” App

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

Then open your Web Browser and enter the URL: **127.0.0.1:8000/company**



<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Read Data

CRUD – Create, **Read**, Update and Delete data from a database

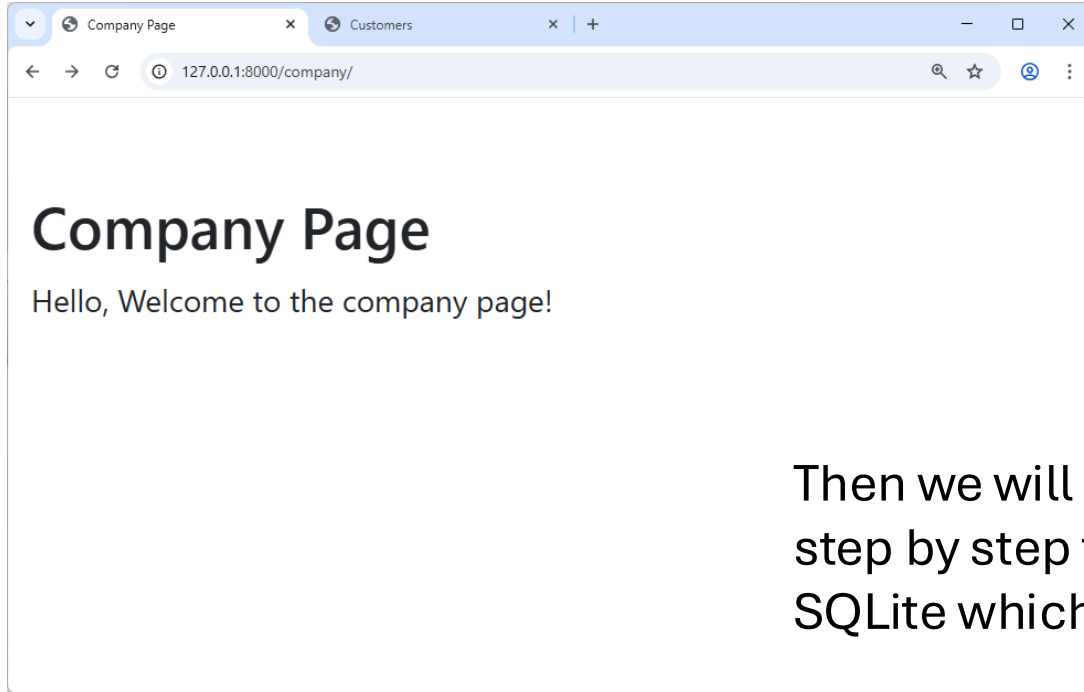
[Table of Contents](#)

Hans-Petter Halvorsen



Django “Company” App

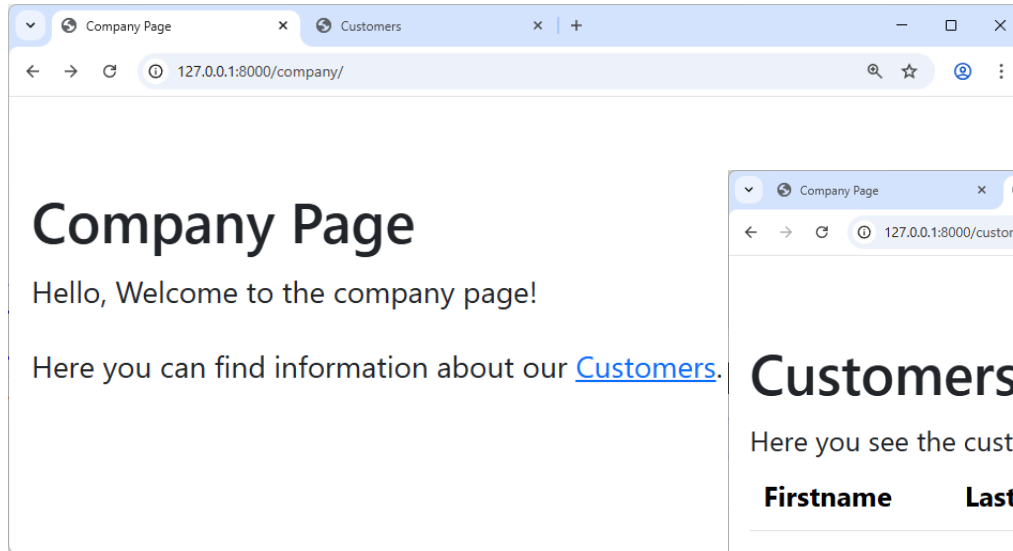
We started by creating a basic “Company” App with a start/home page like this:



Then we will add the CRUD functionality step by step from scratch. We will use SQLite which is included with Django.

Django “Company” App

Main “Company” App:




A browser window showing the main company page. The address bar displays '127.0.0.1:8000/company/'. The page content includes a heading 'Company Page', a welcome message, and a link to the 'Customers' subpage.

Company Page

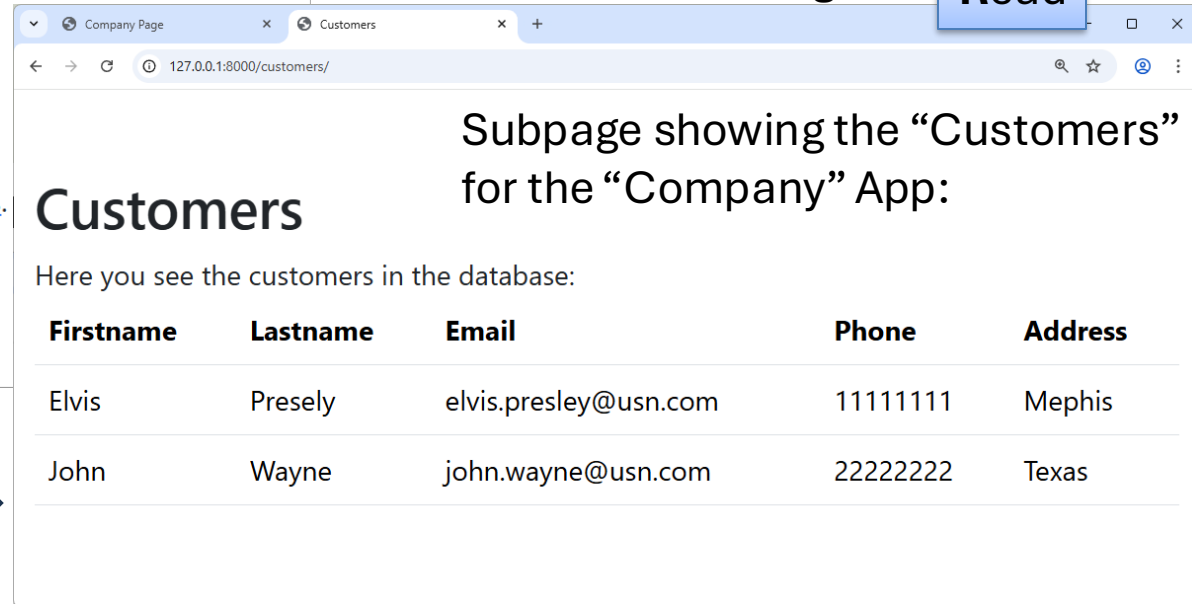
Hello, Welcome to the company page!

Here you can find information about our [Customers](#).

Reading Data from
the Database 

In the “Read” part we
will create the following:

Read



A browser window showing the 'Customers' subpage. The address bar displays '127.0.0.1:8000/customers/'. The page content includes a heading 'Customers', a description, and a table of customer data.

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	22222222	Texas

Subpage showing the “Customers”
for the “Company” App:

Create Model (“models.py”)

```
from django.db import models

# Create your models here.
class Customer(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField()
    phone = models.CharField(max_length=15)
    address = models.TextField()

    def __str__(self):
        return self.first_name + ' ' + self.last_name
```

By default, all Models created in the Django project will be created as tables in the default SQLite database (“db.sqlite3”) that is installed with Django.

Create View (“views.py”)

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

# Create your views here.
def company(request):
    template = loader.get_template('company.html')
    return HttpResponse(template.render())

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

Update urls.py

We need to update “urls.py” for the “company” Django App:

```
urls.py ×
company > urls.py > ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('company/', views.company, name='company'),
6     path('customers/', views.customers, name='customers'),
7 ]
```

Template (“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>
          <tr>
            <td>{{ customer.first_name }}</td>
            <td>{{ customer.last_name }}</td>
            <td>{{ customer.email }}</td>
            <td>{{ customer.phone }}</td>
            <td>{{ customer.address }}</td>
          </tr>
        </table>
      </div>
    </div>
  </body>
</html>
```

We create a **Template** (“customers.html”)

The Template will look like this in the Web Browser:

Firstname	Lastname	Email	Phone	Address
Elvis	Presley	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	22222222	Texas

Migrate

```
(djangoenv1) ..>python manage.py makemigrations
```

```
Command Prompt
(djangoenv1) C:\Temp\Django\djangoenv1\djangoproject1>python manage.py makemigrations
Migrations for 'helloworldapp':
  helloworldapp\migrations\0001_initial.py
  + Create model Customer
(djangoenv1) C:\Temp\Django\djangoenv1\djangoproject1>
```

```
(djangoenv1) ..>python manage.py migrate
```

```
Command Prompt
(djangoenv1) C:\Temp\Django\djangoenv1\djangoproject1>python manage.py makemigrations
Migrations for 'helloworldapp':
  helloworldapp\migrations\0001_initial.py
  + Create model Customer

(djangoenv1) C:\Temp\Django\djangoenv1\djangoproject1>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, helloworldapp, sessions
Running migrations:
  Applying helloworldapp.0001_initial... OK
(djangoenv1) C:\Temp\Django\djangoenv1\djangoproject1>
```

We need to use “Migrate” to update the Database and make the necessary Database Tables in the Database.

DB Browser for SQLite

The screenshot displays the DB Browser for SQLite application window. The main area shows a table named 'company_customer' with the following data:

id	first_name	last_name	email	phone	address
1	Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	John	Wayne	john.wayne@usn.com	5555555	Texas
3	John	Doe	John.Doe@norway.no	555555	Norway

The interface includes a menu bar (File, Edit, View, Tools, Help), a toolbar with actions like 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', 'Undo', 'Open Project', 'Save Project', and 'Attach Database'. The 'Browse Data' tab is active, showing the table structure and data. The 'Edit Database Cell' panel on the right is open, showing 'Mode: Text' and 'Editing row=3, column=1'. The 'Remote' panel is also visible, showing options for 'DBHub.io', 'Local', and 'Current Database'. The status bar at the bottom indicates '1 - 3 of 3' rows and 'Go to: 1'.

<https://sqlitebrowser.org>

SQLiteStudio

The screenshot shows the SQLiteStudio 3.4.17 interface. The main window displays a database connection to 'Django (SQLite 3)'. The left sidebar shows the database structure, including tables like 'auth_group', 'auth_group_permissions', 'auth_permission', 'auth_user', 'auth_user_groups', 'auth_user_user_permissions', and 'company_customer'. The 'company_customer' table is selected, and its columns (id, first_name, last_name, email, phone, address) are visible. The main query editor shows the query: `select * from company_customer`. The results are displayed in a table view, showing 3 rows of data. The status bar at the bottom indicates the query finished in 0.001 second(s).

SQLiteStudio (3.4.17) - [SQL editor 1]

Database Structure View Tools Help

Databases Django

Filter by name

- Django (SQLite 3)
 - Tables (11)
 - auth_group
 - auth_group_permissions
 - auth_permission
 - auth_user
 - auth_user_groups
 - auth_user_user_permissions
 - company_customer
 - Columns (6)
 - id
 - first_name
 - last_name
 - email
 - phone
 - address
 - Indexes
 - Triggers
 - django_admin_log
 - django_content_type
 - django_migrations
 - django_session
 - Views

Query History

```
1 select * from company_customer
```

Grid view Form view

Total rows loaded: 3

id	first_name	last_name	email	phone	address
1	Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	John	Wayne	john.wayne@usn.com	5555555	Texas
3	4 John	Doe	John.Doe@norway.no	555555	Norway

Status [09:37:38] Query finished in 0.001 second(s).

SQL editor 1

<https://sqlitestudio.pl>

DB Browser for SQLite

The screenshot shows the DB Browser for SQLite interface. The main pane displays the database structure, including a table named "helloworldapp_customer" which is highlighted with a red circle. The table has columns: id (integer), firstname (varchar(100)), lastname (varchar(100)), address (varchar(100)), city (varchar(100)), email (varchar(100)), and phone (varchar(20)). A blue callout box contains the text: "We see that the 'customer' has been created in the Database. The table names are actually created as 'appname_modelname', so in our case 'company_customer'". The SQL editor shows the CREATE TABLE statement for "helloworldapp_customer".

Database Structure

Tables (12)

- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissions
- django_admin_log
- django_content_type
- django_migrations
- django_session
- helloworldapp_customer**
 - id integer
 - firstname varchar(100)
 - lastname varchar(100)
 - address varchar(100)
 - city varchar(100)
 - email varchar(100)
 - phone varchar(20)
- sqlite_sequence

Indices (15)

- auth_group_permissions_group_id_b120cbf9
- auth_group_permissions_group_id_permission_id_0cd3
- auth_group_permissions_permission_id_84c5c92e
- auth_permission_content_type_id_2f476e4b
- auth_permission_content_type_id_codename_01ab375e
- auth_user_groups_group_id_97559544
- auth_user_groups_user_id_6a12ed8b
- auth_user_groups_user_id_group_id_94350c9c

```
CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUTO...
CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY KEY AUT...
CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTO...
CREATE TABLE "django_session" ("session_key" varchar(40) NOT NULL PRIMAR...
CREATE TABLE "helloworldapp_customer" ("id" integer NOT NULL PRIMARY KEY...
CREATE TABLE sqlite_sequence(name,seq)
```

SQL Log Plot DB Schema Remote

UTF-8

Insert Data

You can insert data using Python. To open a Python shell:

```
(djangoenv1) ...>python manage.py shell
```

```
>>> Customer.objects.all().values()  
<QuerySet []>
```

We see there is no data

We can insert data like this:

```
>>> customer = Customer(first_name='Hans-Petter', last_name='Halvorsen',  
email='hph@usn.no', phone='12345678', address='Norway')  
>>> customer.save()
```

Then we can look for the data:

```
>>> Customer.objects.all().values()  
<QuerySet [{'id': 1, 'first_name': 'Hans-Petter', 'last_name': 'Halvorsen',  
'email': 'hph@usn.no', 'phone': '12345678', 'address': 'Norway'}]>
```

Insert Data using DB Browser

You can insert data using Python or you can insert data using DB Browser.

The screenshot shows the DB Browser for SQLite interface. The 'company_customer' table is displayed with the following data:

id	first_name	last_name	email	phone	address
1	Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	John	Wayne	john.wayne@usn.com	22222222	Texas

The 'Write Changes' button in the toolbar is highlighted with a red box. A red callout box contains the text: "Note! Make sure to click 'Write Changes' to update the Database!!!".

The SQL editor shows the following query:

```
1 select * from company_customer
```

A second red callout box contains the text: "Note! The Table name is actually 'company_customer'".

In DB Browser you can use the GUI to enter data or a plain insert statement.

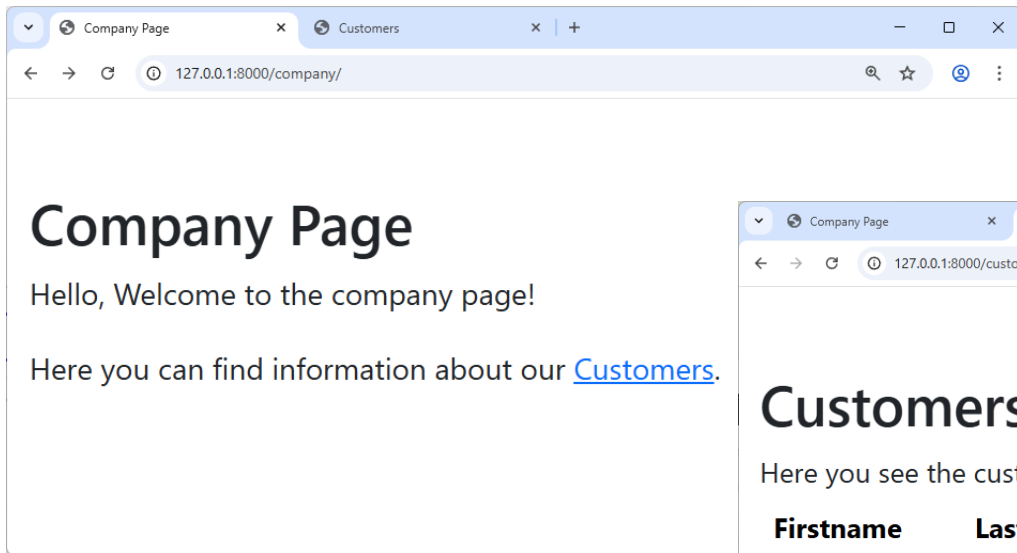
	id	first_name	last_name	email	phone	address
1	1	Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	2	John	Wayne	john.wayne@usn.com	22222222	Texas

Updated “company.html”

```
<!DOCTYPE html>
<html>
  <head>
    <title>Company App</title>
    <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>Company Home Page</h1>
      <p>Hello, Welcome to our Company!</p>
      <p>Here you see our <a href="/customers">customers</a>.</p>
    </div>
  </body>
</html>
```

Testing Django “Company” App

Main “Company” App:



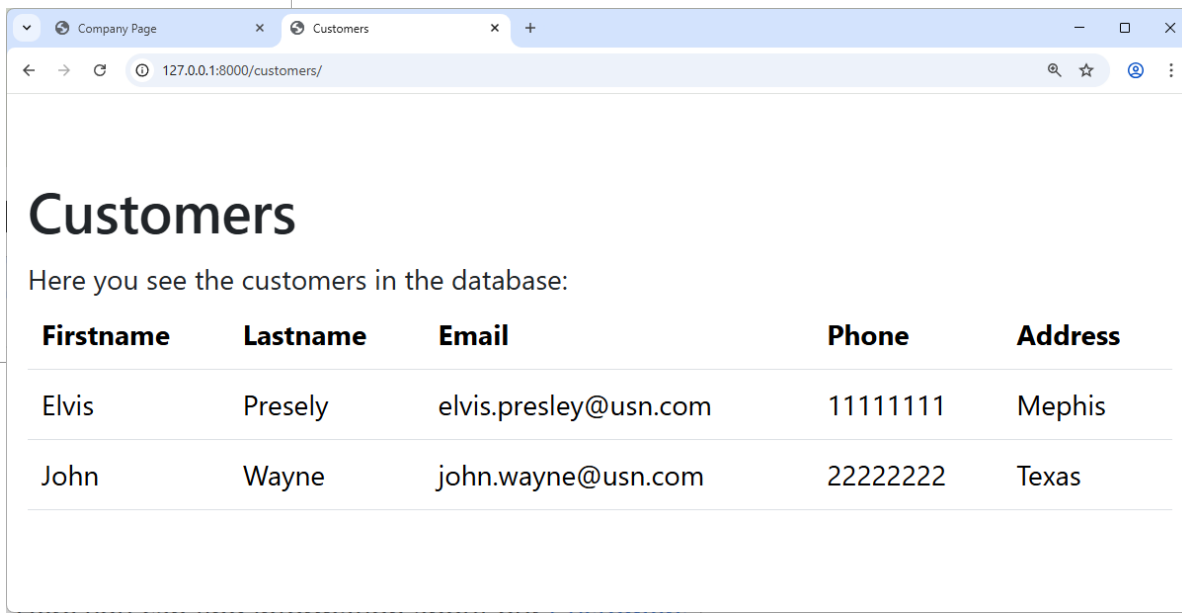
A browser window showing the main 'Company Page'. The address bar displays '127.0.0.1:8000/company/'. The page content includes a heading 'Company Page', a welcome message, and a link to the 'Customers' subpage.

Company Page

Hello, Welcome to the company page!

Here you can find information about our [Customers](#).

Subpage showing the “Customers” for the “Company” App:



A browser window showing the 'Customers' subpage. The address bar displays '127.0.0.1:8000/customers/'. The page content includes a heading 'Customers', a message about the database, and a table listing customer information.

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	22222222	Texas

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Create Data

CRUD – **Create**, Read, Update and Delete data from a database

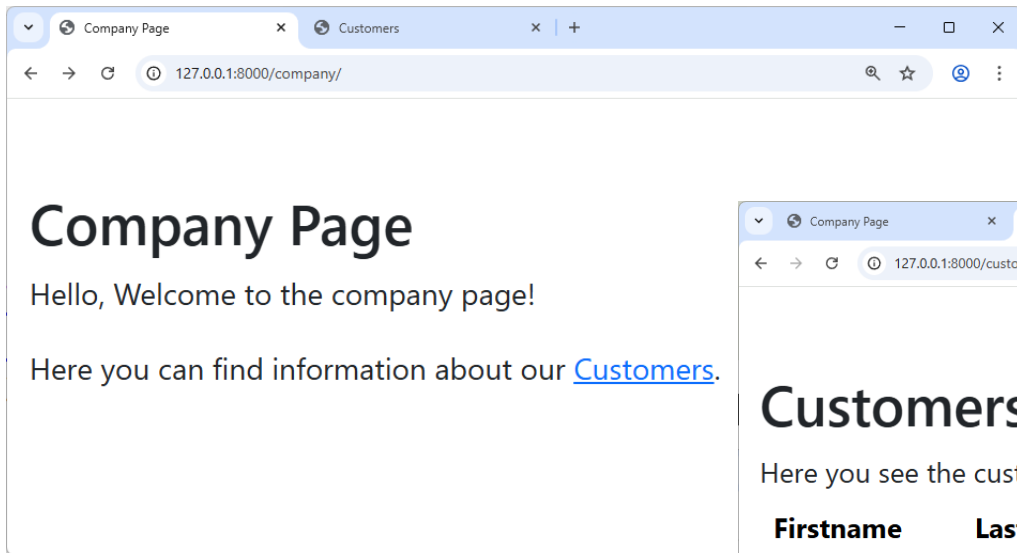
[Table of Contents](#)

Hans-Petter Halvorsen



Django “Company” App

Main “Company” App:

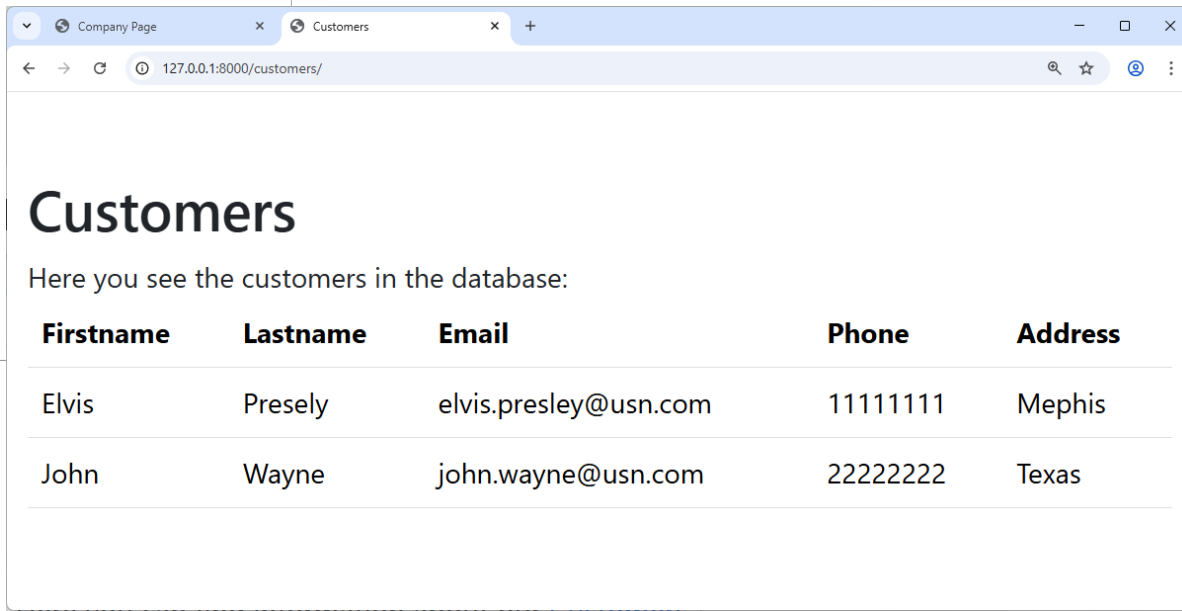


Company Page

Hello, Welcome to the company page!

Here you can find information about our [Customers](#).

Subpage showing the “Customers”
for the “Company” App:



Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	22222222	Texas

“Company” App – Create Customer

In the “Create” part we will create the following:

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway

New Customer

New Customer

First Name:

First Name:

Email:

Phone Number:

Address:

Save

Create View (views.py)

We create a new View called “customernew” in “views.py”:

```
def customernew(request):
    data = Customer()

    data.first_name = "John"
    data.last_name = "Doe"
    data.email = "John.Doe@norway.no"
    data.phone = "555555"
    data.address = "Norway"

    data.save()

    message = "<h1>New Customer</h1><p>New Customer successfully created.</p>"
    return HttpResponseRedirect(message)
```

View (views.py)

You can also write like this if you prefer:

```
def customernew(request):  
    data = Customer(first_name="John", last_name="Doe",  
                    email="John.Doe@norway.no", phone="555555", address="Norway")  
    data.save()  
    message = "<h1>New Customer</h1><p>New Customer successfully created.</p>"  
    return HttpResponseRedirect(message)
```

Update urls.py

The screenshot shows a code editor with the following content:

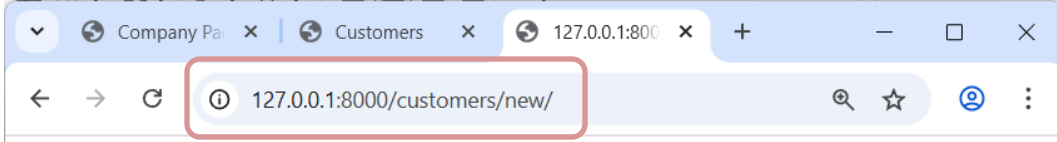
```
company > urls.py > ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('company/', views.company, name='company'),
6     path('customers/', views.customers, name='customers'),
7     path('customers/new/', views.customernew, name='new'),
8 ]
```

Note! We need to register the View in **urssl.py** as well

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
System check identified no issues (0 silenced).
May 27, 2025 - 15:34:07
Django version 5.2.1, using settings 'djangoproject1.settings'
Starting development server at http://127.0.0.1:8000/
```

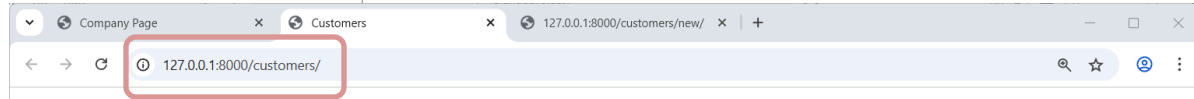
Testing



New Customer

New Customer successfully created.

We see a new Customer has been created



Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Create with Forms

CRUD – **Create**, Read, Update and Delete data from a database



[Table of Contents](#)

Hans-Petter Halvorsen

Create with Forms

- In the previous “Create” example we just created new data in the database from the View without any GUI.
- Typically, we want the user of the application to create these data from a GUI inside the application.
- Here we will create and use a **HTML Form**.
- So, we create a new Django Template, which will basically be a HTML page with a HTML Form.
- We also need to update the “customernew” view.

“Company” App – Create Customer

In the “Create” part we will create the following:

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway

New Customer

New Customer

First Name:

First Name:

Email:

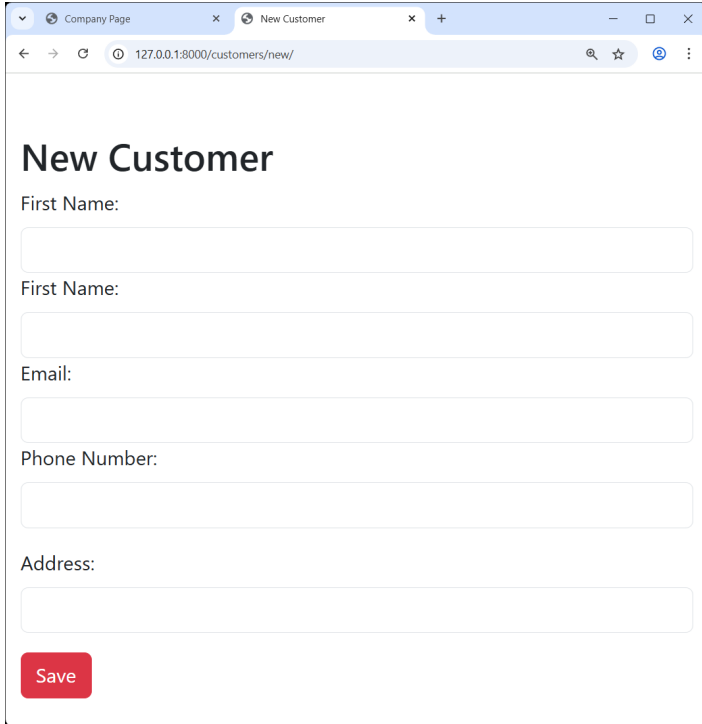
Phone Number:

Address:

Save

Template (customer_new.html)

We create a new Template called, e.g., “customer_new.html”:



The screenshot shows a web browser window with the URL 127.0.0.1:8000/customers/new/. The page title is "New Customer". The form contains the following fields and a button:

- First Name: [text input]
- First Name: [text input]
- Email: [text input]
- Phone Number: [text input]
- Address: [text input]
- Save: [red button]

```
models.py views.py company.html customers.html customer_new.html x
company > templates > customer_new.html > html > head > body > div.container-fluid.pt-5 > div.form-group > form
 2 <html lang="en">
 3 <head>
 9 <body>
10 <div class="container-fluid pt-5">
12
13 <div class="form-group">
14 <form action="..../new/" method="post">
15 {% csrf_token %}
16
17 <label for="first_name" class="form-label">First Name:</label>
18 <input id="first_name" type="text" name="first_name" class="form-control">
19
20 <label for="last_name" class="form-label">First Name:</label>
21 <input id="last_name" type="text" name="last_name" class="form-control">
22
23 <label for="email" class="form-label">Email:</label>
24 <input id="email" type="email" name="email" class="form-control">
25
26 <label for="phone" class="form-label">Phone Number:</label>
27 <input id="phone" type="tel" name="phone" class="form-control"></p>
28
29 <label for="address" class="form-label">Address:</label>
30 <input id="address" type="text" name="address" class="form-control"></p>
31
32 <input type="submit" value="Save" class="btn btn-danger">
33 </form>
34 </div>
35
36 </div>
```

Template (customer_new.html)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>New Customer</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <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>New Customer</h1>

      <div class="form-group">
        <form action="../new/" method="post">
          {% csrf_token %}

          <label for="first_name" class="form-label">First Name:</label>
          <input id="first_name" type="text" name="first_name" class="form-control">

          <label for="last_name" class="form-label">First Name:</label>
          <input id="last_name" type="text" name="last_name" class="form-control">

          <label for="email" class="form-label">Email:</label>
          <input id="email" type="email" name="email" class="form-control">

          <label for="phone" class="form-label">Phone Number:</label>
          <input id="phone" type="tel" name="phone" class="form-control"></p>

          <label for="address" class="form-label">Address:</label>
          <input id="address" type="text" name="address" class="form-control"></p>

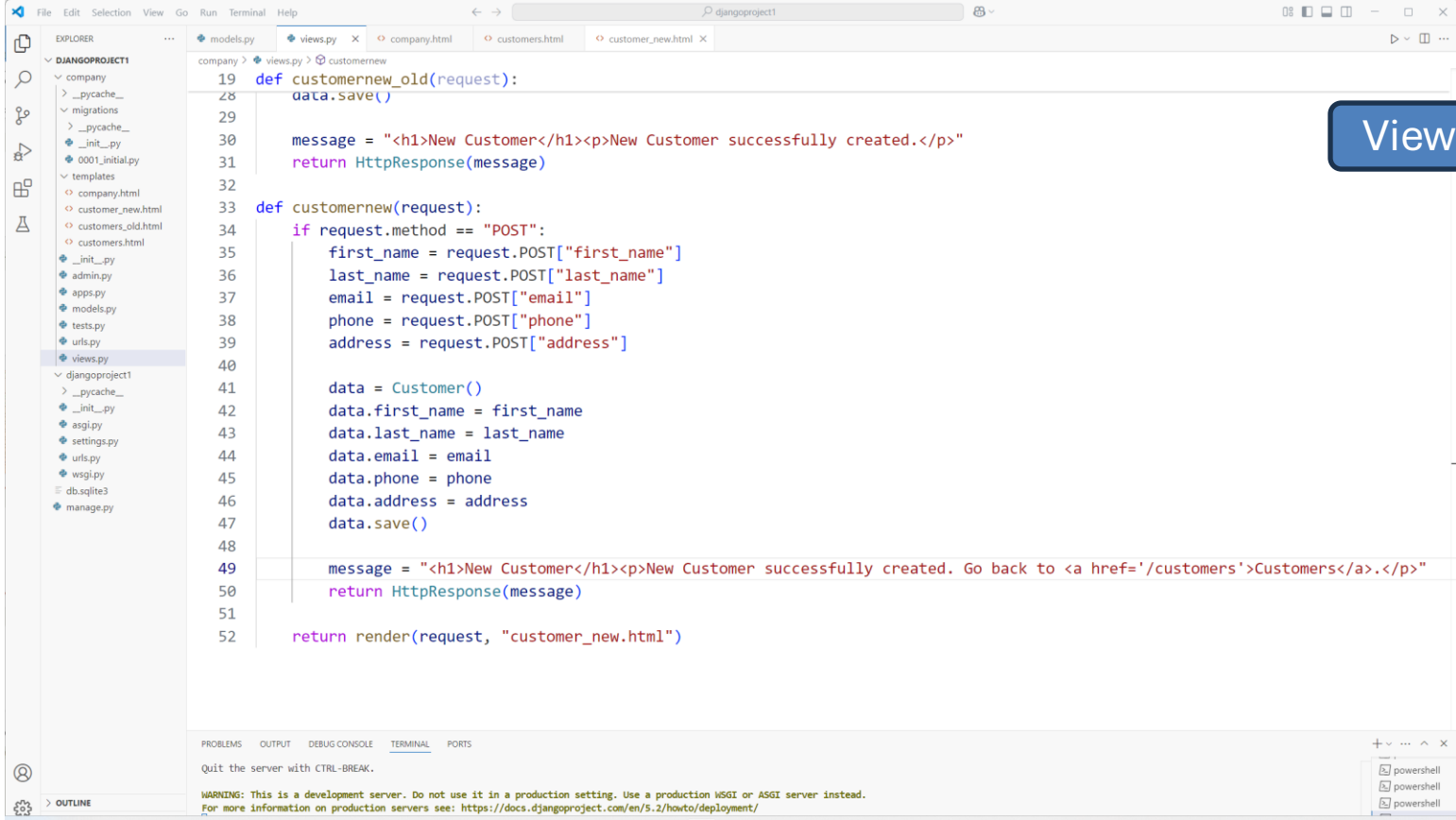
          <input type="submit" value="Save" class="btn btn-danger">
        </form>
      </div>

    </div>
  </body>
</html>
```

{% csrf_token %}

- When creating a POST form we need to worry about Cross Site Request Forgeries (CSRF).
- Django comes with builtin functionality for protecting against CSRF.
- All POST forms should use the {% csrf_token %} template tag.

Updated View (“customernew”)



```
19 def customernew_old(request):
20     data.save()
21
22     message = "<h1>New Customer</h1><p>New Customer successfully created.</p>"
23     return HttpResponseRedirect(message)
24
25 def customernew(request):
26     if request.method == "POST":
27         first_name = request.POST["first_name"]
28         last_name = request.POST["last_name"]
29         email = request.POST["email"]
30         phone = request.POST["phone"]
31         address = request.POST["address"]
32
33         data = Customer()
34         data.first_name = first_name
35         data.last_name = last_name
36         data.email = email
37         data.phone = phone
38         data.address = address
39         data.save()
40
41     message = "<h1>New Customer</h1><p>New Customer successfully created. Go back to <a href='/customers'>Customers</a>.</p>"
42     return HttpResponseRedirect(message)
43
44     return render(request, "customer_new.html")
```

Views.py

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Quit the server with CTRL-BREAK.
WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: <https://docs.djangoproject.com/en/5.2/howto/deployment/>

powershell
powershell
powershell

Updated View (“customernew”)

```
def customernew(request):
    if request.method == "POST":
        first_name = request.POST["first_name"]
        last_name = request.POST["last_name"]
        email = request.POST["email"]
        phone = request.POST["phone"]
        address = request.POST["address"]

        data = Customer()
        data.first_name = first_name
        data.last_name = last_name
        data.email = email
        data.phone = phone
        data.address = address
        data.save()

        message = "<h1>New Customer</h1><p>New Customer successfully created. Go back to <a
                    href='/customers'>Customers</a>.</p>"
        return HttpResponseRedirect(message)

return render(request, "customer_new.html")
```

Views.py

Updated Template “customers.html”

```
models.py views.py x company.html customers.html x customer_new.html
company > templates > customers.html > html > body > div.container-fluid.pt-5 > div.table-responsive > table.table > tr > td
 2 <html>
10 <body>
11 <div class="container-fluid pt-5">
14 <div class="table-responsive">
15 <table class="table">
16 <tr>
17 <th>Firstname</th>
18 <th>Lastname</th>
19 <th>Email</th>
20 <th>Phone</th>
21 <th>Address</th>
22 </tr>
23 {% for customer in customers %}
24 <tr>
25 <td>{{ customer.first_name }}</td>
26 <td>{{ customer.last_name }}</td>
27 <td>{{ customer.email }}</td>
28 <td>{{ customer.phone }}</td>
29 <td>{{ customer.address }}</td>
30 </tr>
31 {% endfor %}
32 </table>
33
34 <a href="/customers/new" class="btn btn-success">New Customer</a>
35
36 </body>
37 </html>
```



We add a “New Customer” Button

Testing

Customers

Here you see the customers in the database:

Firstname	Lastname	Email
Elvis	Presley	elvis.presley@usn.com
John	Wayne	john.wayne@usn.com
John	Doe	John.Doe@norway.no

New Customer

New Customer

First Name:

First Name:

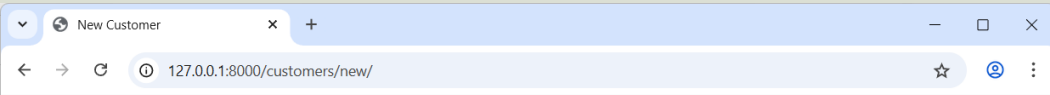
Email:

Phone Number:

Address:

Save

Testing



New Customer

First Name:

Hans-Petter

First Name:

Halvorsen

Email:

hans.p.halvorsen@usn.no

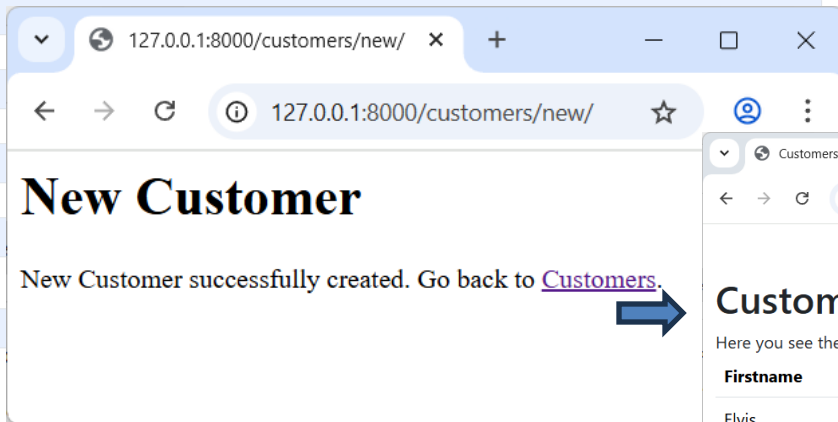
Phone Number:

99999999

Address:

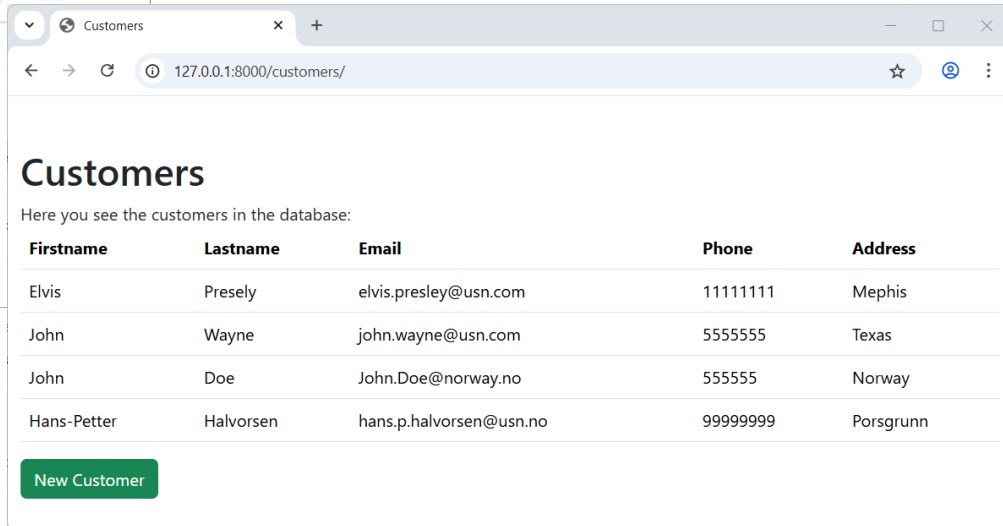
Porsgrunn

Save



New Customer

New Customer successfully created. Go back to [Customers](#).



Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn

New Customer

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Create with “ModelForm”

CRUD – **C**reate, Read, Update and Delete data from a database

[Table of Contents](#)

Hans-Petter Halvorsen



ModelForm

- Django has a ModelForm class that automatically renders a HTML form with its structure matching with the attributes of a model class.
- This is an alternative way compared to previous example.
- I guess this way of doing it has both pros and cons.
- I would recommend doing it the “hard way” (as shown in the previous example) because you lose some control by doing it this way.

Django Forms (forms.py)

We need to create a new file called “forms.py” where we define our Form;

forms.py ×

company > forms.py > CustomerForm > Meta

```
1 from django import forms
2 from .models import Customer
3
4 class CustomerForm(forms.ModelForm):
5     class Meta:
6         model = Customer
7         fields = "__all__"
```

```
from django import forms
from .models import Customer

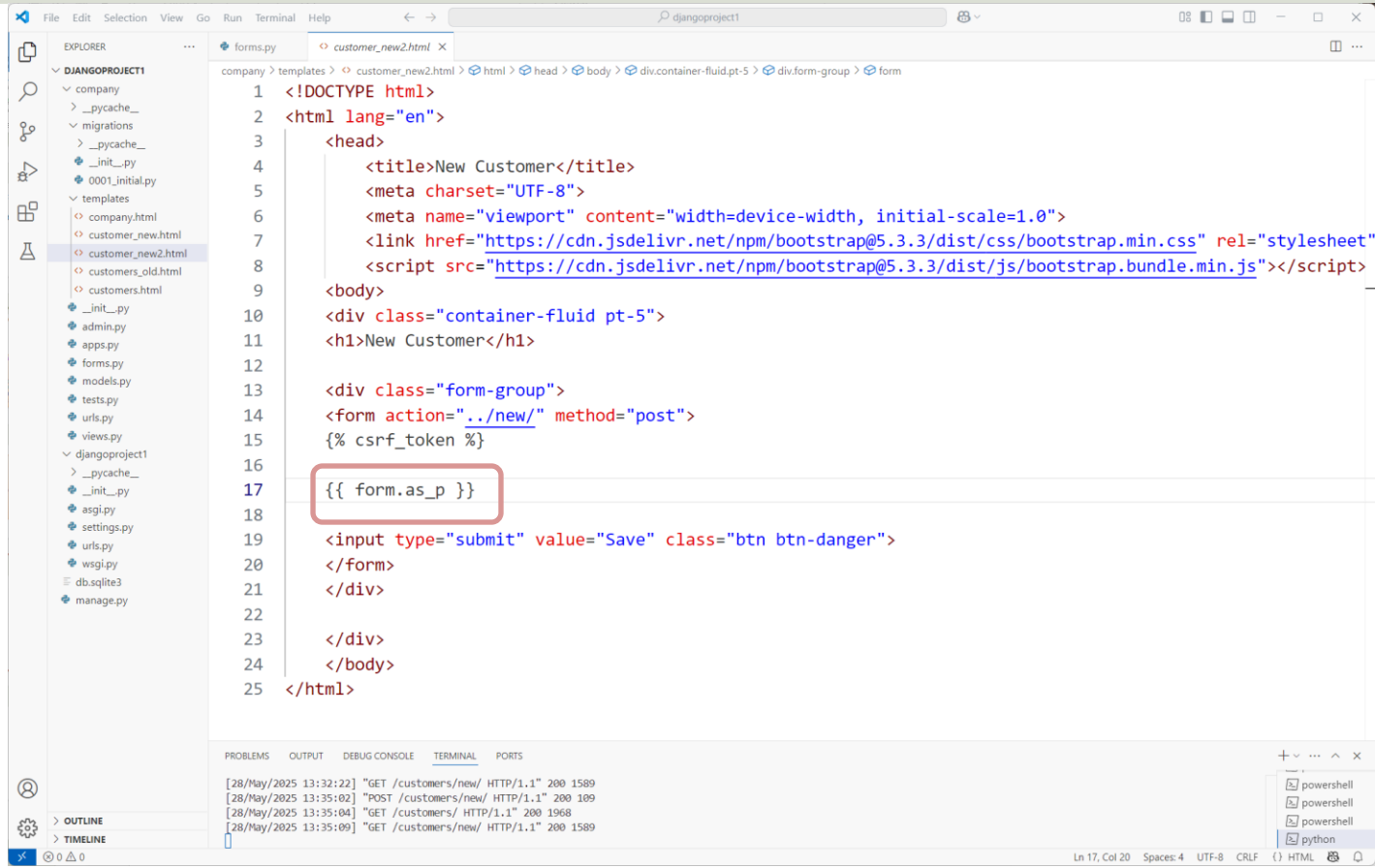
class
CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = "__all__"
```

Updated View ("customernew")

```
def customernew(request):
    if request.method == "POST":
        form = CustomerForm(request.POST)
        if form.is_valid():
            form.save()
            message = "<h1>New Customer</h1><p>New Customer successfully
                created. Go back to <a href='/customers'>Customers</a>.</p>"
            return HttpResponseRedirect(message)

    context = {'form' : CustomerForm}
    return render(request, "customer_new2.html", context)
```

Template (customer_new2.html)



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>New Customer</title>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
8     <script src="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
9   </head>
10  <div class="container-fluid pt-5">
11    <h1>New Customer</h1>
12
13    <div class="form-group">
14      <form action="..new/" method="post">
15        {% csrf_token %}
16        {{ form.as_p }}
17      </form>
18    </div>
19    <input type="submit" value="Save" class="btn btn-danger">
20  </div>
21 </body>
22 </div>
23 </body>
24 </html>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[28/May/2025 13:32:22] "GET /customers/new/ HTTP/1.1" 200 1589
[28/May/2025 13:35:02] "POST /customers/new/ HTTP/1.1" 200 109
[28/May/2025 13:35:04] "GET /customers/ HTTP/1.1" 200 1968
[28/May/2025 13:35:09] "GET /customers/new/ HTTP/1.1" 200 1589
```

powerShell
powerShell
powerShell
python

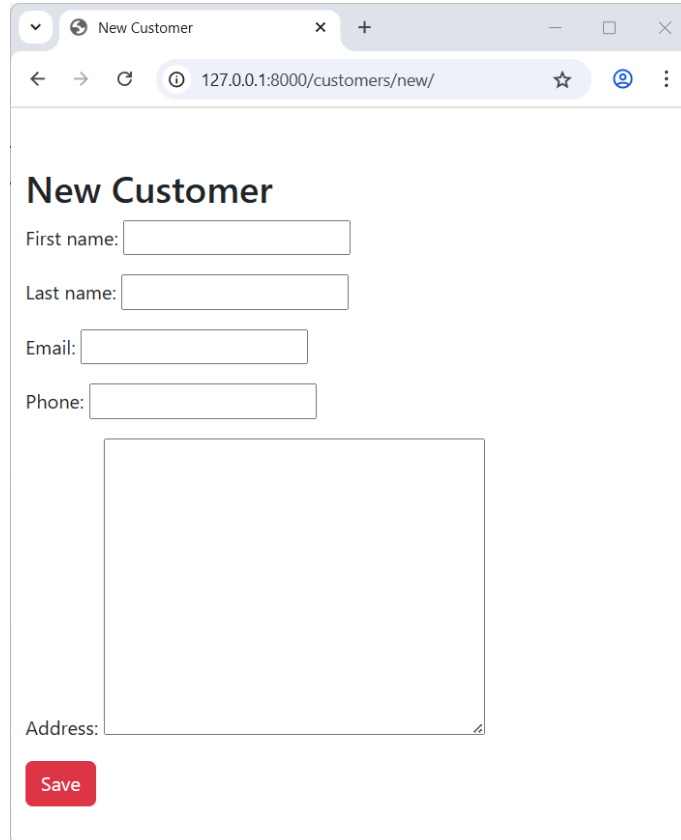
Ln 17, Col 20 Spaces: 4 UTF-8 CRLF HTML

{{ form.as_p }}

- The `{{ form.as_p }}` method render the contents of “form” as paragraphs.
- For more information:
- https://www.geeksforgeeks.org/form-as_p-render-django-forms-as-paragraph/

Testing

We see that the Form contents is automatically generated:



The screenshot shows a web browser window with the title 'New Customer' and the URL '127.0.0.1:8000/customers/new/'. The form is titled 'New Customer' and contains the following fields:

- First name:
- Last name:
- Email:
- Phone:
- Address:

At the bottom of the form is a red 'Save' button.

I guess this way of doing it has both pros and cons. I would recommend doing it the “hard way” (as shown in the previous example) because you lose some control by doing it this way.

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Update Data

CRUD – Create, Read, **Update** and Delete data from a database

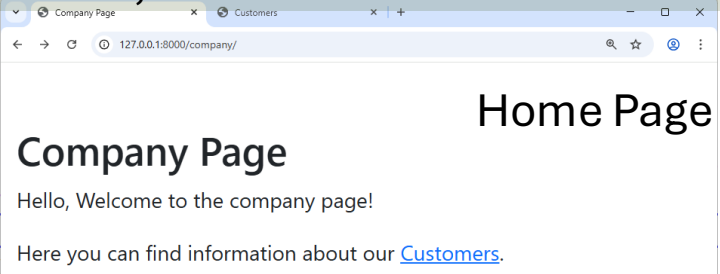


[Table of Contents](#)

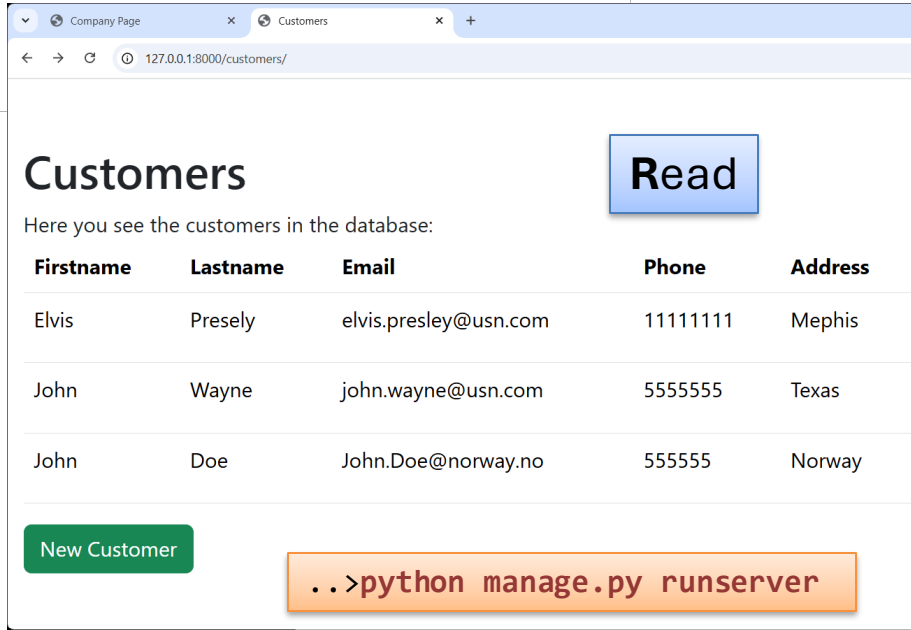
Hans-Petter Halvorsen

“Company” App

So far, we have created the following:



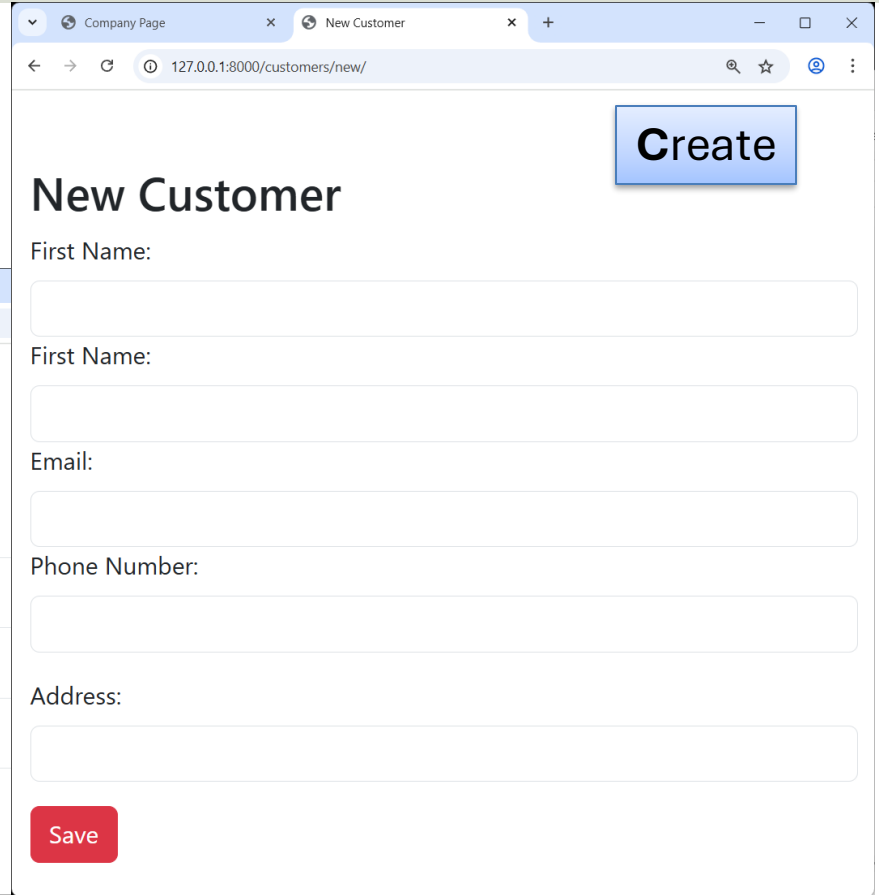
A browser window showing the Home Page. The address bar contains '127.0.0.1:8000/company/'. The page title is 'Company Page'. The main content includes the heading 'Company Page', a welcome message 'Hello, Welcome to the company page!', and a link 'Here you can find information about our [Customers](#).'



A browser window showing the Customers page. The address bar contains '127.0.0.1:8000/customers/'. The page title is 'Customers'. It features a 'Read' button, a message 'Here you see the customers in the database:', and a table with customer data. At the bottom, there is a 'New Customer' button and a terminal command box.

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway

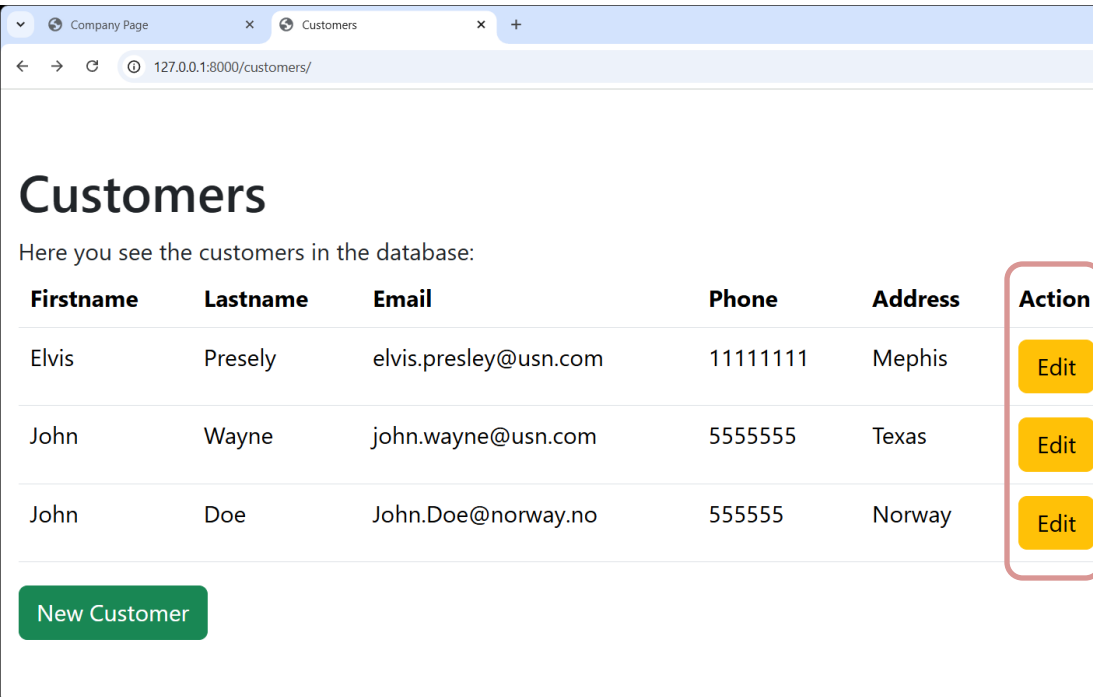
```
..>python manage.py runserver
```



A browser window showing the 'New Customer' form. The address bar contains '127.0.0.1:8000/customers/new/'. The page title is 'New Customer'. It features a 'Create' button, a 'First Name:' label, an input field, an 'Email:' label, an input field, a 'Phone Number:' label, an input field, an 'Address:' label, an input field, and a 'Save' button.

“Company” App – Update Customer

In the “Update” part we will create the following:

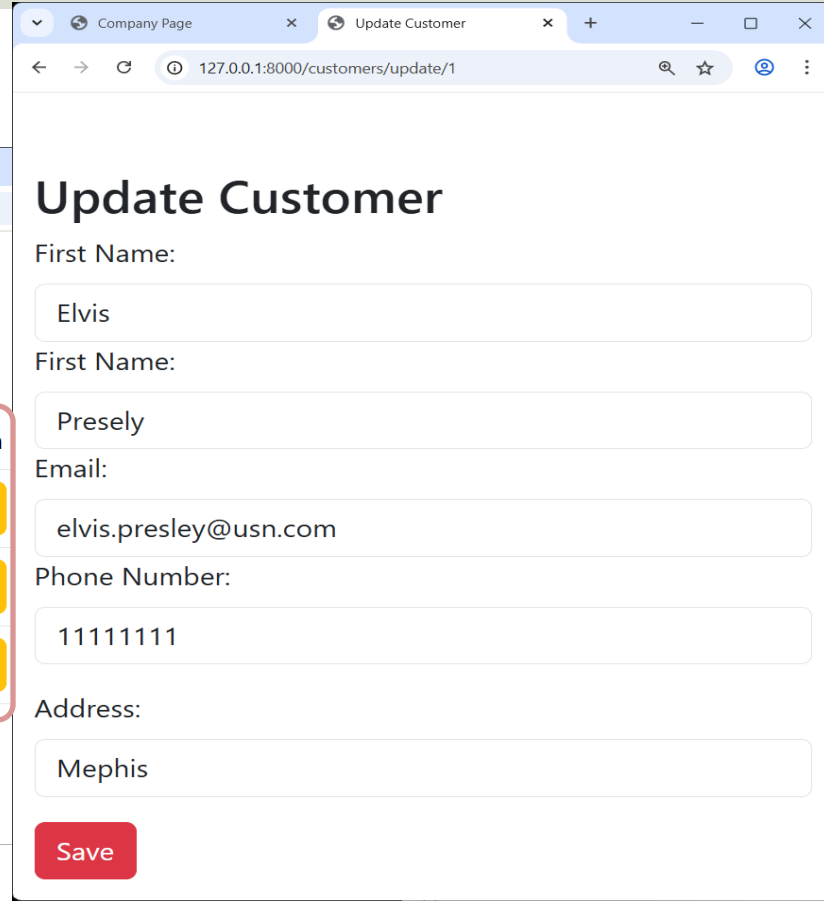


Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit
John	Doe	John.Doe@norway.no	555555	Norway	Edit

New Customer



Update Customer

First Name:
Elvis

First Name:
Presely

Email:
elvis.presley@usn.com

Phone Number:
11111111

Address:
Mephis

Save

Create View (views.py)

We create a new View called “**customerupdate**” in “**views.py**”:

```
def customerupdate(request, pk):  
    data = Customer.objects.get(pk=pk)  
  
    data.first_name = "Jens"  
    data.save()
```

In this first basic example, we just hardcode the “first_name” to be updated to “Jens”.

```
message = "<h1>Update Customer</h1><p>Customer updated successfully.</p>"  
return HttpResponseRedirect(message)
```

Update urls.py

Note! We need to register the View in **urssl.py** as well

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

urlpatterns = [
    path('company/', views.company, name='company'),
    path('customers/', views.customers, name='customers'),
    path('customers/new/', views.customernew, name='new'),
    path('customers/update/<int:pk>', views.customerupdate, name='update'),
]
```



SQLiteStudio

We can use, e.g., SQLiteStudio to see the existing data in the database and find the Primary Keys (PK)

The screenshot shows the SQLiteStudio (3.4.17) interface. The left sidebar displays the database structure for 'Django (SQLite 3)', with the 'company_customer' table selected. The main window shows a query: `select * from company_customer`. Below the query, the 'Grid view' displays 5 rows of data. The 'Status' bar at the bottom indicates the query finished in 0.001 second(s).

id	first_name	last_name	email	phone	address
1	1 Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	2 John	Wayne	john.wayne@usn.com	5555555	Texas
3	4 John	Doe	John.Doe@norway.no	555555	Norway
4	11 Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn
5	12 Knut	Hamsun	knut.hamsun@norway.no	4444444	Norway

Testing

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

We can, e.g., enter a URL like this: <http://127.0.0.1:8000/customers/update/12>

The screenshot shows a web browser window with the address bar containing `127.0.0.1:8000/customers/update/12`. The main content area displays the heading "Update Customer" and the message "Customer updated successfully." To the right, a query editor shows the SQL query `select * from company_customer`. Below the query editor, a table view displays the data for the `company_customer` table. The table has columns for `id`, `first_name`, `last_name`, `email`, `phone`, and `address`. The row with `id` 12 is highlighted with a red box, indicating it is the record that was updated.

	id	first_name	last_name	email	phone	address
1	1	Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
2	2	John	Wayne	john.wayne@usn.com	5555555	Texas
3	4	John	Doe	John.Doe@norway.no	555555	Norway
4	11	Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn
5	12	Jens	Hamsun	knut.hamsun@norway.no	4444444	Norway

"12" is the Primary Key (PK) for this record

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Update with Forms

CRUD – Create, Read, **Update** and Delete data from a database



[Table of Contents](#)

Hans-Petter Halvorsen

Update with Forms

- In the previous “Update” example we just updated existing data in the database from the View without any GUI. We just hardcoded the data that we updated in the code.
- Typically, we want the user of the application to update these data from a GUI inside the application.
- Here we will create and use a **HTML Form** – just like we did for Create earlier in this Tutorial.
- So, we create a new Django Template, which will basically be a HTML page with a HTML Form.
- We also need to update the “customerupdate” view.

Updated View (“customerupdate”)

Views.py

We change the “customerupdate” to this to start with:

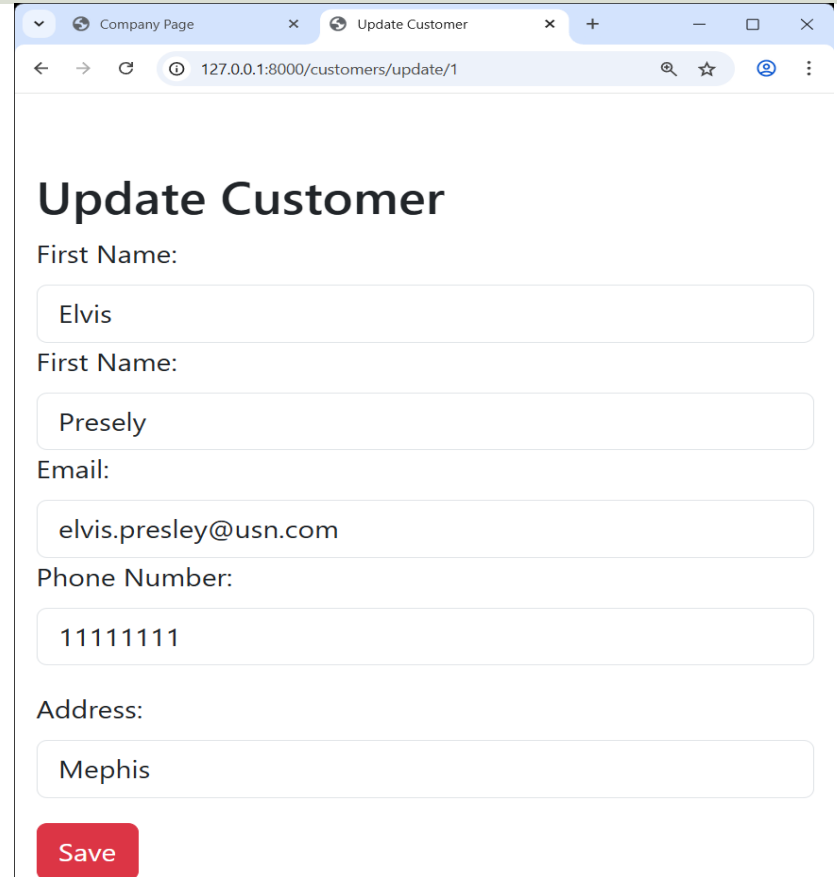
```
def customerupdate(request, pk):  
    data = Customer.objects.get(pk=pk)  
  
    context = {"data": data}  
    return render(request, "customer_update.html", context)
```



Next, we need to create the “customer_update.html” Template. See next slide.

Create Template (customer_update.html)

We create a new Template called, e.g., “customer_update.html”:



The screenshot shows a web browser window with two tabs: 'Company Page' and 'Update Customer'. The address bar displays '127.0.0.1:8000/customers/update/1'. The page content includes a heading 'Update Customer' and a form with the following fields:

- First Name: Elvis
- First Name: Presely
- Email: elvis.presley@usn.com
- Phone Number: 11111111
- Address: Mephis

A red 'Save' button is located at the bottom of the form.

The HTML code is on the next PP slide.

Template (customer_update.html)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Update Customer</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <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>Update Customer</h1>

      <div class="form-group">
        <form action="../update/{{ data.pk }}" method="post">
          {% csrf_token %}
          <label for="first_name" class="form-label">First Name:</label>
          <input id="first_name" type="text" name="first_name" value="{{ data.first_name }}" class="form-control">

          <label for="last_name" class="form-label">First Name:</label>
          <input id="last_name" type="text" name="last_name" value="{{ data.last_name }}" class="form-control">

          <label for="email" class="form-label">Email:</label>
          <input id="email" type="email" name="email" value="{{ data.email }}" class="form-control">

          <label for="phone" class="form-label">Phone Number:</label>
          <input id="phone" type="tel" name="phone" value="{{ data.phone }}" class="form-control"></p>

          <label for="address" class="form-label">Address:</label>
          <input id="address" type="text" name="address" value="{{ data.address }}" class="form-control"></p>

          <input type="submit" value="Save" class="btn btn-danger">
        </form>
      </div>
    </div>
  </body>
</html>
```

Testing

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

We can, e.g., enter a URL like this: <http://127.0.0.1:8000/customers/update/12>

The screenshot shows two browser windows. The left window displays the 'Customers' page at <http://127.0.0.1:8000/customers/>. It features a table of customer records and a 'New Customer' button.

Firstname	Lastname	Email
Elvis	Presely	elvis.presley@usn.com
John	Wayne	john.wayne@usn.com
John	Doe	John.Doe@norway.no
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no
Jens	Hamsun	knut.hamsun@norway.no

The right window displays the 'Update Customer' form at <http://127.0.0.1:8000/customers/update/12>. The form contains the following fields:

- First Name: Jens
- First Name: Hamsun
- Email: knut.hamsun@norway.no
- Phone Number: 4444444
- Address: Norway

A red 'Save' button is located at the bottom of the form.

Updated View (“customerupdate”)

```
def customerupdate(request, pk):
    data = Customer.objects.get(pk=pk)

    if request.method == "POST":
        first_name = request.POST["first_name"]
        last_name = request.POST["last_name"]
        email = request.POST["email"]
        phone = request.POST["phone"]
        address = request.POST["address"]

        data.first_name = first_name
        data.last_name = last_name
        data.email = email
        data.phone = phone
        data.address = address
        data.save()

        message = "<h1>Update Customer</h1><p>Customer updated successfully. Go back to <a
                    href='/customers'>Customers</a>.</p>"
        return HttpResponseRedirect(message)

    data = Customer.objects.get(pk=pk)
    context = {"data": data}
    return render(request, "customer_update.html", context)
```

Views.py

Updated Template “customers.html”

```
customers.html ×
company > templates > customers.html > html > body > div.container-fluid.pt-5 > div.table-responsive > table.table > tr > td > a.btn.btn-warning
2  <html>
10 <body>
11   <div class="container-fluid pt-5">
14     <div class="table-responsive">
15       <table class="table">
16         <tr>
17           <th>Firstname</th>
18           <th>Lastname</th>
19           <th>Email</th>
20           <th>Phone</th>
21           <th>Address</th>
22           <th>Action</th>
23         </tr>
24         {% for customer in customers %}
25           <tr>
26             <td>{{ customer.first_name }}</td>
27             <td>{{ customer.last_name }}</td>
28             <td>{{ customer.email }}</td>
29             <td>{{ customer.phone }}</td>
30             <td>{{ customer.address }}</td>
31             <td><a href="/customers/update/{{ customer.pk }}" class="btn btn-warning">Edit</a></td>
32           </tr>
33         {% endfor %}
34       </table>
35
36       <a href="/customers/new" class="btn btn-success">New Customer</a>
37
38 </body>
39 </html>
```

We can add an “Edit” button in in the “customers.html” Template

Testing

Company Page Customers New Customer Update Customer

127.0.0.1:8000/customers/update/12

Update Customer

First Name:

First Name:

Email:

Phone Number:

Address:

Save

Customer updated successfully. Go back to [Customers](#).

Company Page Customers New Customer

127.0.0.1:8000/customers/upd...

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis
John	Wayne	john.wayne@usn.com	5555555	Texas
John	Doe	John.Doe@norway.no	555555	Norway
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn
Knut	Hamsun	knut.hamsun@norway.no	4444444	Norway

New Customer

Testing

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit
John	Doe	John.Doe@norway.no	555555	Norway	Edit
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn	Edit
Knut	Hamsun	knut.hamsun@norway.no	4444444	Norway	Edit

New Customer

Update Customer

First Name:

First Name:

Email:

Phone Number:

Address:

Save

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Update with “ModelForm”

CRUD – Create, Read, **Update** and Delete data from a database



Hans-Petter Halvorsen

[Table of Contents](#)

Django Forms (forms.py)

We need to update the “CustomerForm” in “forms.py”:

```
class CustomerForm(forms.ModelForm):  
    class Meta:  
        model = Customer  
        fields = "__all__"  
  
    def __init__(self, *args, **kwargs):  
        super(CustomerForm, self).__init__(*args, **kwargs)
```

Updated View ("customerupdate")

We also need to update the "customerupdate" View in "views.py":

```
def customerupdate(request, pk):
    data = Customer.objects.get(pk=pk)

    if request.method == "POST":
        form = CustomerForm(request.POST)
        if form.is_valid():
            form.save()
            message = "<h1>Update Customer</h1><p>Customer updated
            successfully. Go back to <a href='/customers'>Customers</a>.</p>"
            return HttpResponseRedirect(message)

    data = Customer.objects.get(pk=pk)
    context = {'data' : CustomerForm(instance=data), "pk": data.pk}
    return render(request, "customer_update2.html", context)
```

Template (customer_update2.html)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Update Customer</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <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>
  <body>
    <div class="container-fluid pt-5">
      <h1>Update Customer</h1>

      <div class="form-group">
        <form action="../update/{{ pk }}" method="post">
          {% csrf_token %}

          {{ data.as_p }}

          <input type="submit" value="Update" class="btn btn-danger">
        </form>
      </div>

    </div>
  </body>
</html>
```

We also create a new Template called “customer_update3.html” in the templates folder:

{{ data.as_p }}

- The `{{ data.as_p }}` method render the contents of “data” as paragraphs.
- For more information:
- https://www.geeksforgeeks.org/form-as_p-render-django-forms-as-paragraph/

Testing

We see that the Form contents is automatically generated:

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit
John	Doe	John.Doe@norway.no	555555	Norway	Edit
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.com	99999999	Porsgrunn	Edit
Knut	Hamsun	knut.hamsun@norway.no	4444444	Norway	Edit

New Customer

Update Customer

First name:

Last name:

Email:

Phone:

Address:

I guess this way of doing it has both pros and cons. I would recommend doing it the “hard way” (as shown in the previous example) because you lose some control by doing it this way.

<https://www.halvorsen.blog>

The Django logo, featuring the word "django" in a white, lowercase, sans-serif font on a dark green rectangular background.

Django CRUD Web Application

Delete Data

CRUD – Create, Read, Update and **Delete** data from a database



Hans-Petter Halvorsen

[Table of Contents](#)

“Company” App – Delete Customer

In the “Delete” part we will create the following:

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit Delete
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit Delete
John	Doe	John.Doe@norway.no	555555	Norway	Edit Delete

New Customer

Company Page 127.0.0.1:8000/customer

127.0.0.1:8000/customers/delete/11

Delete Customer

Customer deleted successfully. Go back to [Customers](#).

Create View (views.py)

We create a new View called “**customerdelete**” in “**views.py**”:

```
def customerdelete(request, pk):  
    data = Customer.objects.get(pk=pk)  
    data.delete()  
  
    message = "<h1>Delete Customer</h1><p>Customer deleted successfully.  
              Go back to <a href='/customers'>Customers</a>.</p>"  
    return HttpResponseRedirect(message)
```

Update urls.py

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

Note! We need to register the View in **urssl.py** as well

```
urlpatterns = [
    path('company/', views.company, name='company'),
    path('customers/', views.customers, name='customers'),
    path('customers/new/', views.customernew, name='new'),
    path('customers/update/<int:pk>', views.customerupdate, name='update'),
    path('customers/delete/<int:pk>', views.customerdelete, name='delete'),
]
```



Updated Template “customers.html”

```
customers.html x
company > templates > customers.html > html > body > div.container-fluid.pt-5 > div.table-responsive > table.table > tr > td
 2 <html>
10 <body>
11 <div class="container-fluid pt-5">
14 <div class="table-responsive">
15 <table class="table">
16 <tr>
17 <th>Firstname</th>
18 <th>Lastname</th>
19 <th>Email</th>
20 <th>Phone</th>
21 <th>Address</th>
22 <th>Action</th>
23 </tr>
24 {% for customer in customers %}
25 <tr>
26 <td>{{ customer.first_name }}</td>
27 <td>{{ customer.last_name }}</td>
28 <td>{{ customer.email }}</td>
29 <td>{{ customer.phone }}</td>
30 <td>{{ customer.address }}</td>
31 <td>
32 <a href="/customers/update/{{ customer.pk }}" class="btn btn-warning">Edit</a>
33 <a href="/customers/delete/{{ customer.pk }}" class="btn btn-danger">Delete</a>
34 </td>
35 </tr>
36 {% endfor %}
37 </table>
38
39 <a href="/customers/new" class="btn btn-success">New Customer</a>
```

Testing

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit Delete
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit Delete
John	Doe	John.Doe@norway.no	555555	Norway	Edit Delete
Hans-Petter	Halvorsen	hans.p.halvorsen@usn.no	99999999	Porsgrunn	Edit Delete
Knut	Hamsun	knut.hamsun@norway.no	4444444	Norway	Edit Delete

New Customer

Company Page

127.0.0.1:8000/customer

127.0.0.1:8000/customers/delete/11

Delete Customer

Customer deleted successfully. Go back to [Customers](#).

Improve with JavaScript

In the Delete example so far you just push the Delete and then the customer is deleted immediately without any warning or anything. We can, e.g., implement a simple JavaScript function that asks if you want to delete or not.

Step 1: We make a JavaScript Function in “customers.html”:

```
<script>
function deleteCustomer(id)
{
    let message = "Do you really want to delete customer #" + id + "?";

    if (confirm(message) == true)
    {
        window.location.href = "/customers/delete/" + id
    }
}
</script>
```

Step 2: We update the code for the Delete button

```
<a href="javascript:deleteCustomer({{ customer.pk }})" class="btn btn-danger">Delete</a>
```

customers.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Customers</title>
  <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>
<script>
function deleteCustomer(id)
{
  let message = "Do you really want to delete customer #" + id + "?";

  if (confirm(message) == true)
  {
    window.location.href = "/customers/delete/" + id
  }
}
</script>
<body>
  <div class="container-fluid pt-5">
    <h1>Customers</h1>
    <p>Here you see a list of customers from our database:</p>
    <div class="table_responsive">
      <table class="table">
        <thead>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Email</th>
          <th>Phone</th>
          <th>Address</th>
          <th>Action</th>
        </thead>
        <tbody>
          <tr>
            <td>{% for customer in customers %}</td>
            <td>{{ customer.first_name }}</td>
            <td>{{ customer.last_name }}</td>
            <td>{{ customer.email }}</td>
            <td>{{ customer.phone }}</td>
            <td>{{ customer.address }}</td>
            <td>
              <a href="/customers/update/{{ customer.pk }}" class="btn btn-warning">Edit</a>
              <a href="javascript:deleteCustomer('{{ customer.pk }})" class="btn btn-danger">Delete</a>
            </td>
          </tr>
          <tr>
            <td colspan="6">{% endfor %}</td>
          </tr>
        </tbody>
      </table>
      <a href="/customers/new" class="btn btn-success">New Customer</a>
    </div>
  </body>
</html>
```

Another Delete Alternative

We create a new template called “**customer_delete.html**”

```
<!DOCTYPE html>
<html>
  <head>
    <title>Customers</title>
    <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>Delete Customer</h1>

      <div class="form-group">
        <form action=" ../delete/{{ data.pk }}" method="post">
          {% csrf_token %}

          <p>Do you want to delete the customer '{{ data.first_name }} {{ data.last_name }}'?</p>

          <input type="submit" value="Delete" class="btn btn-danger">
          <a href = " ../.. /customers"><input type="button" value="Cancel" class="btn btn-success"/></a>
        </form>
      </div>
    </div>
  </body>
</html>
```

Updated View (“customerdelete”)

```
def customerdelete(request, pk):
    data = Customer.objects.get(pk=pk)

    if request.method == "POST":
        data.delete()
        message = "<h1>Delete Customer</h1><p>Customer deleted successfully.
                  Go back to <a href='/customers'>Customers</a></p>"
        return HttpResponseRedirect(message)

    context = {"data": data}
    return render(request, "customer_delete.html", context)
```

customers.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Customers</title>
  <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>
    <p>Here you see a list of customers from our database:</p>

    <div class="table_responsive">
      <table class="table">
        <tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Email</th>
          <th>Phone</th>
          <th>Address</th>
          <th>Action</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>
          <td>
            <a href="/customers/update/{{ customer.pk }}" class="btn btn-warning">Edit</a>
            <a href="/customers/delete/{{ customer.pk }}" class="btn btn-danger">Delete</a>
          </td>
        </tr>
        {% endfor %}
      </table>
      <a href="/customers/new" class="btn btn-success">New Customer</a>
    </div>
  </body>
</html>
```

<https://www.halvorsen.blog>

django

Django CRUD Web Application

Django Admin



Hans-Petter Halvorsen

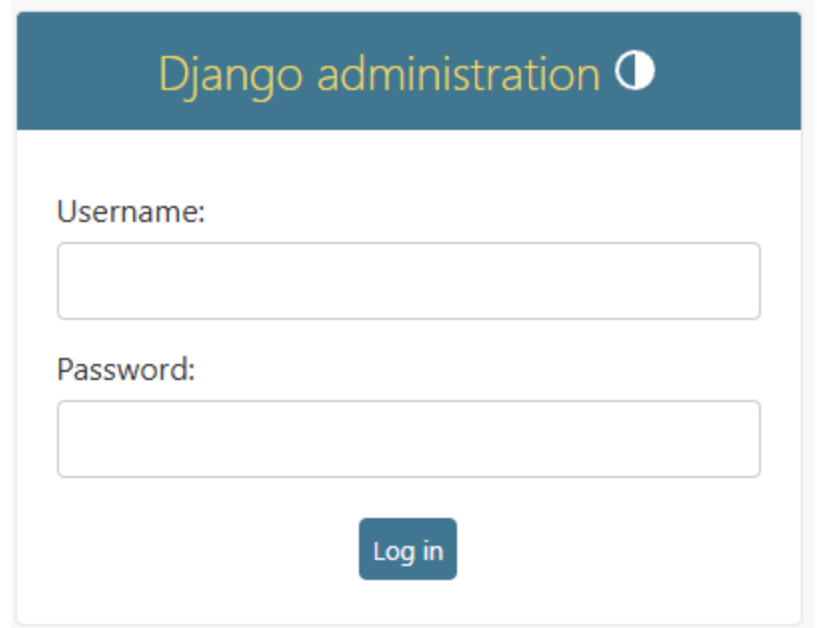
[Table of Contents](#)

Django Admin

- “Django Admin” is included with Django.
- Django provides a ready-to-use user interface for administrative activities. Django automatically generates admin UI based on your Django Models.
- “Django Admin” offers a CRUD user interface for all your Models.
- CRUD is short for Create, Read, Update and Delete.
- Note! The “Django Admin“ interface is intended for “superusers” and no for ordinary users of your application.
- To use it you need to create a User.

Django Admin

Go to <http://127.0.0.1:8000/admin>



The image shows a screenshot of the Django administration interface. At the top, there is a dark blue header with the text "Django administration" and a small white circle icon. Below the header, the page is white and contains a login form. The form has two input fields: "Username:" and "Password:". Below the "Password:" field, there is a dark blue button with the text "Log in".

Django Admin - Create User

Type the following to create a new User:

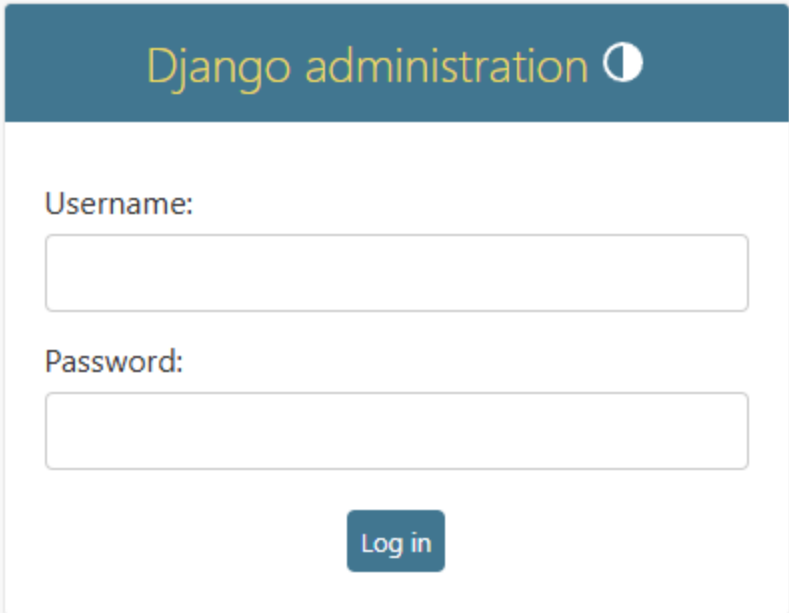
```
...>python manage.py createsuperuser
```

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

Then run the server:

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

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



The screenshot shows the Django administration login interface. At the top, there is a dark blue header with the text "Django administration" and a circular icon. Below the header, the page is white. There are two input fields: "Username:" followed by a text input box, and "Password:" followed by a text input box. At the bottom right, there is a dark blue button with the text "Log in".

Add “Customer” Model to Admin

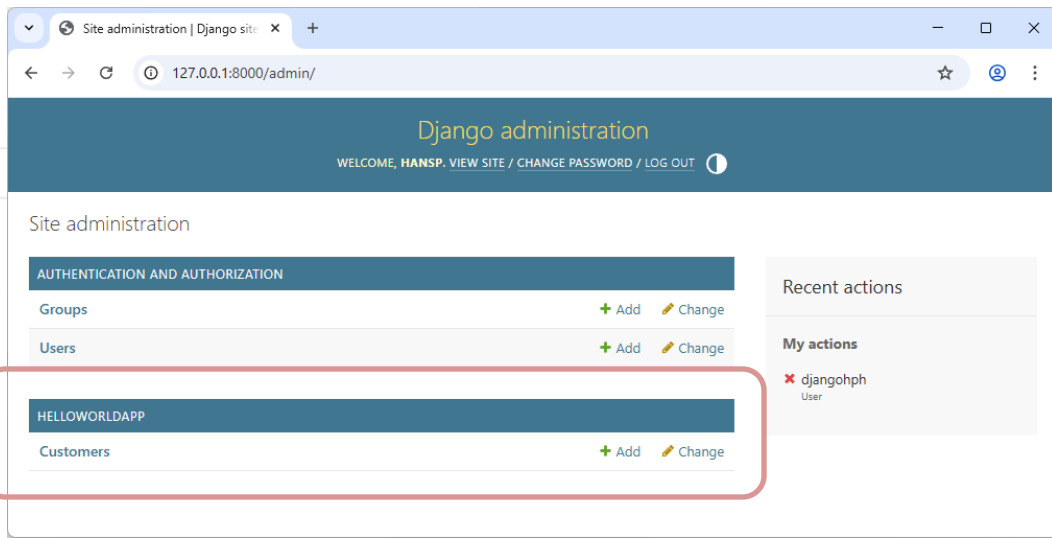
Django Admin offers a CRUD user interface for all your Models.
Let's add CRUD functionality for our Customer Model.

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

```
admin.py x
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)
```

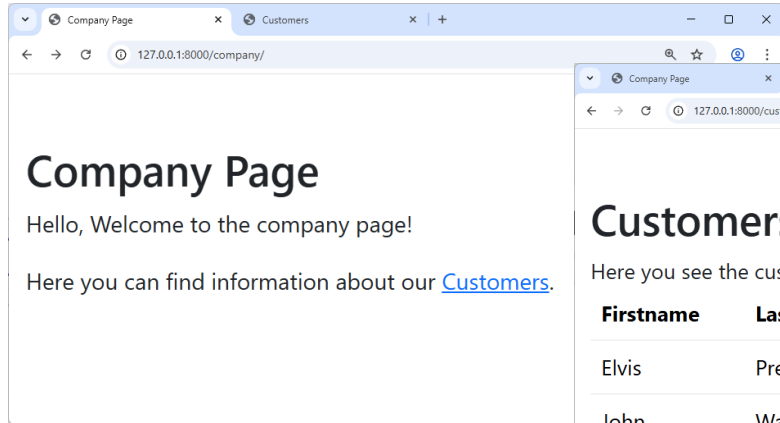
Then go back to

<http://127.0.0.1:8000/admin>

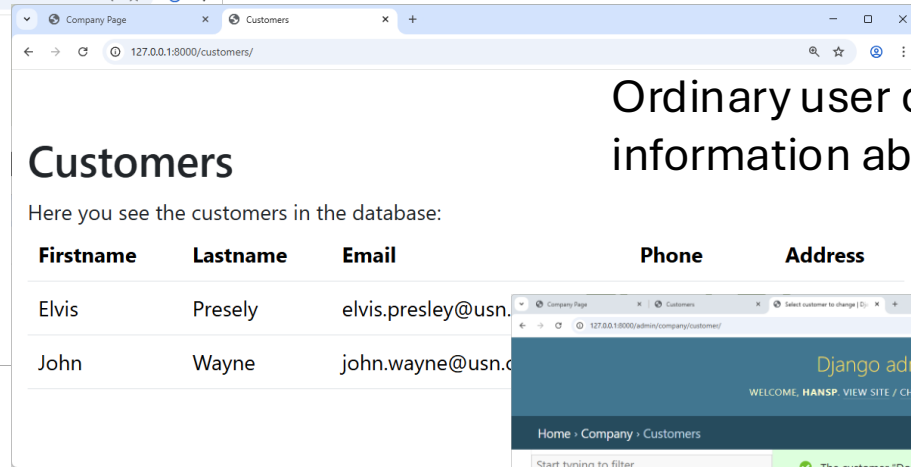


“Company” App with “Django Admin”

Main “Company” App

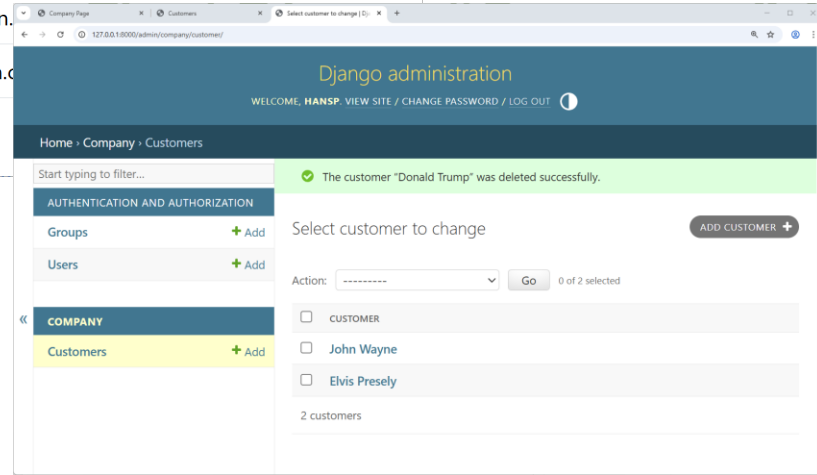


Subpage showing the “Customers” for the “Company” App:



Ordinary user can only see information about the customers

Admin interface for creating, changing and deleting Customers (this part is only available for superusers, and administrators and you need to login):



Summary

We have created a Django Web Application with **CRUD** functionality:

Read

Customers

Here you see the customers in the database:

Firstname	Lastname	Email	Phone	Address	Action
Elvis	Presely	elvis.presley@usn.com	11111111	Mephis	Edit Delete
John	Wayne	john.wayne@usn.com	5555555	Texas	Edit Delete Save
John	Doe	John.Doe@norway.no	555555	Norway	Edit Delete

[New Customer](#)

This is just a basic example. There are lots of improvements to be made!

Create

New Customer

First Name:

First Name:

Email:

Phone Number:

Address:

Delete

Delete Customer

Customer deleted successfully. Go back to [Customers](#).

Django Admin

Django administration

WELCOME HANS! VIEW SITE / CHANGE PASSWORD / LOG OUT

Home - Company - Customers

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups [+ Add](#)
- Users [+ Add](#)

COMPANY

- Customers [+ Add](#)

Select customer to change

ADD CUSTOMER +

Actions: [dropdown] Go 0 of 2 selected

- CUSTOMER
- John.Wayne
- EN
- 2 custom

Update

Update Customer

First Name:

Elvis

First Name:

Presely

Email:

elvis.presley@usn.com

Phone Number:

11111111

Address:

Mephis

Save

Resources

- Working with forms:
<https://docs.djangoproject.com/en/5.2/topics/forms/>
- Django Tutorial Part 9: Working with forms:
https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Forms
- Django Models:
https://www.tutorialspoint.com/django/django_models.htm

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

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

Web: <https://www.halvorsen.blog>

