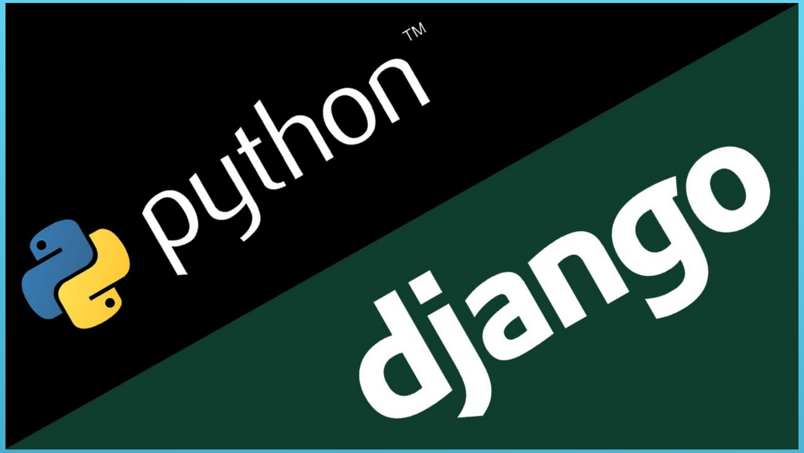
DJANGO

SQL-Injection

Allenspach Peter, Helbling Samuel, Keller Stefan, Sala Nicolò, Vanzella Matteo

OVERVIEW

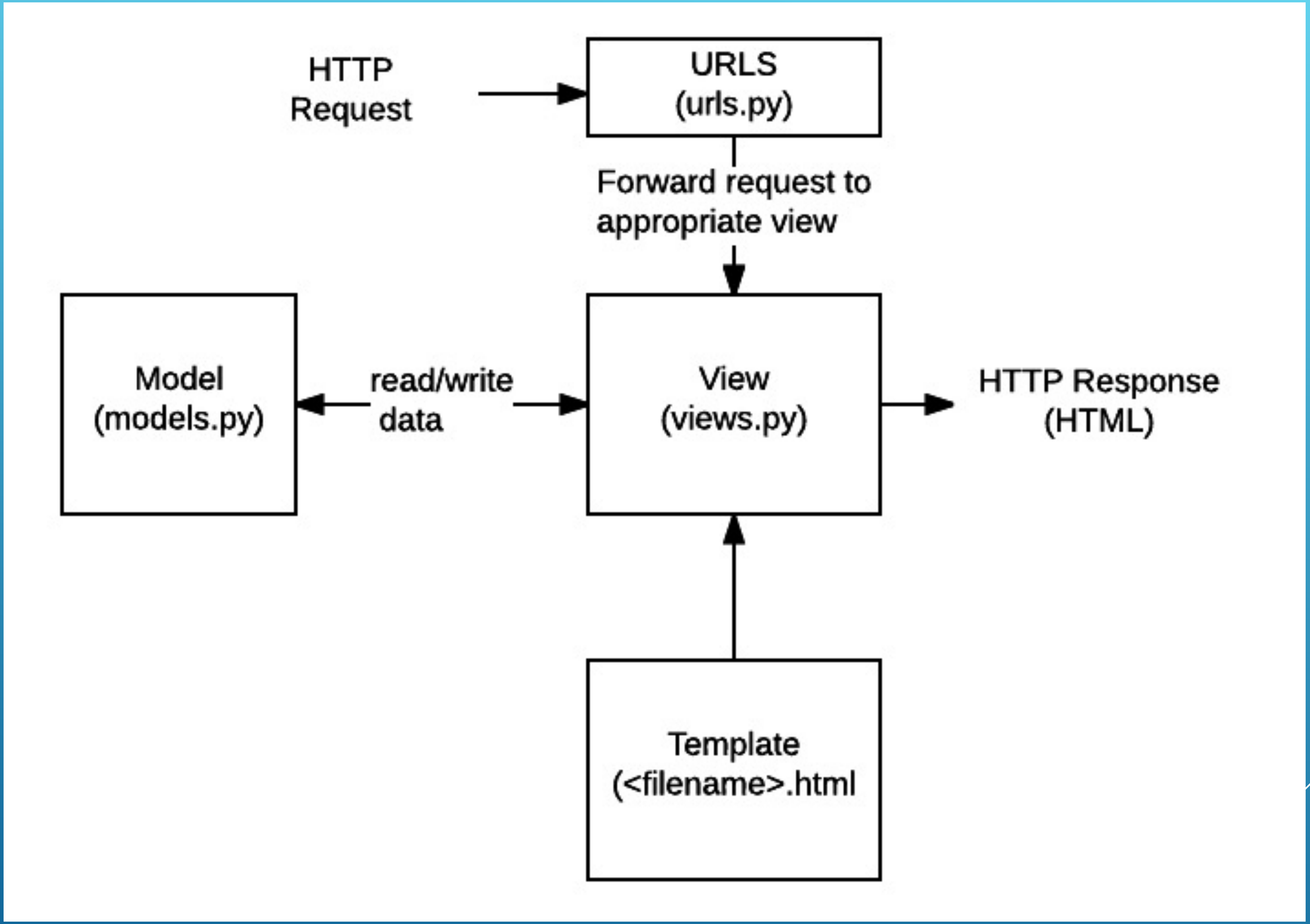


- A high-level Python web framework
- Encourages rapid development and clean, pragmatic design
- Includes a default object-relational mapping layer (ORM)
- Sites using Django: Instagram, Spotify, Nasa, Dropbox, Mozilla, Pinterest

SECURITY

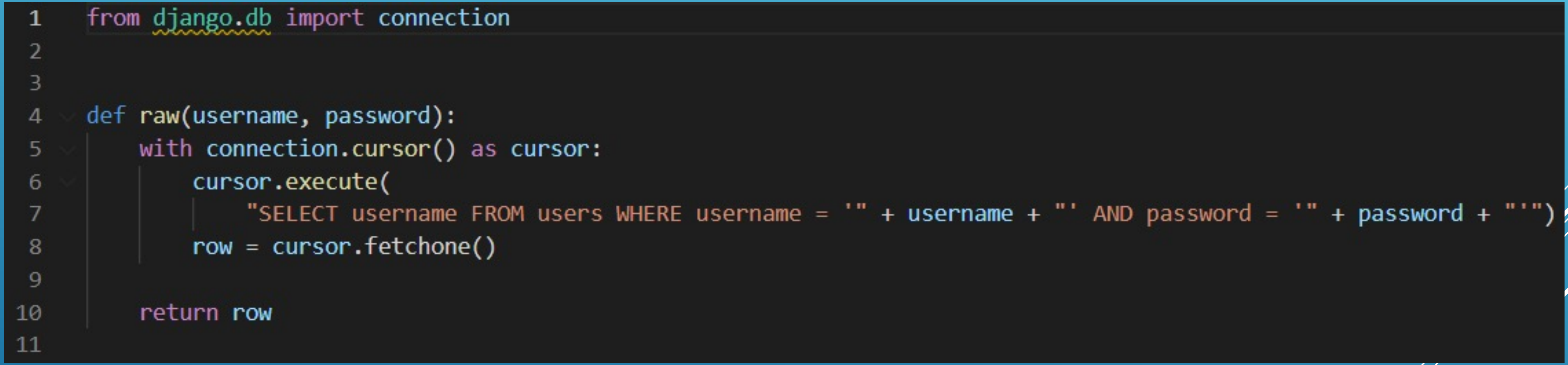
- Security features by default
- Provides protection to developers who have no security experience
- Makes it difficult to write insecure code
- Security Features:
  - User Management
  - Authorization
  - SQL Injection
  - Cross Site Scripting (XSS)
  - Cross Site Request Forgery (CSRF)
  - Clickjacking
  - E-mail Header Injection
  - Cryptography

MVT


HTTP Request
URLS (urls.py)
Forward request to appropriate view
Model (models.py)
read/write data
View (views.py)
HTTP Response (HTML)
Template (<filename>.html

BAD WAY

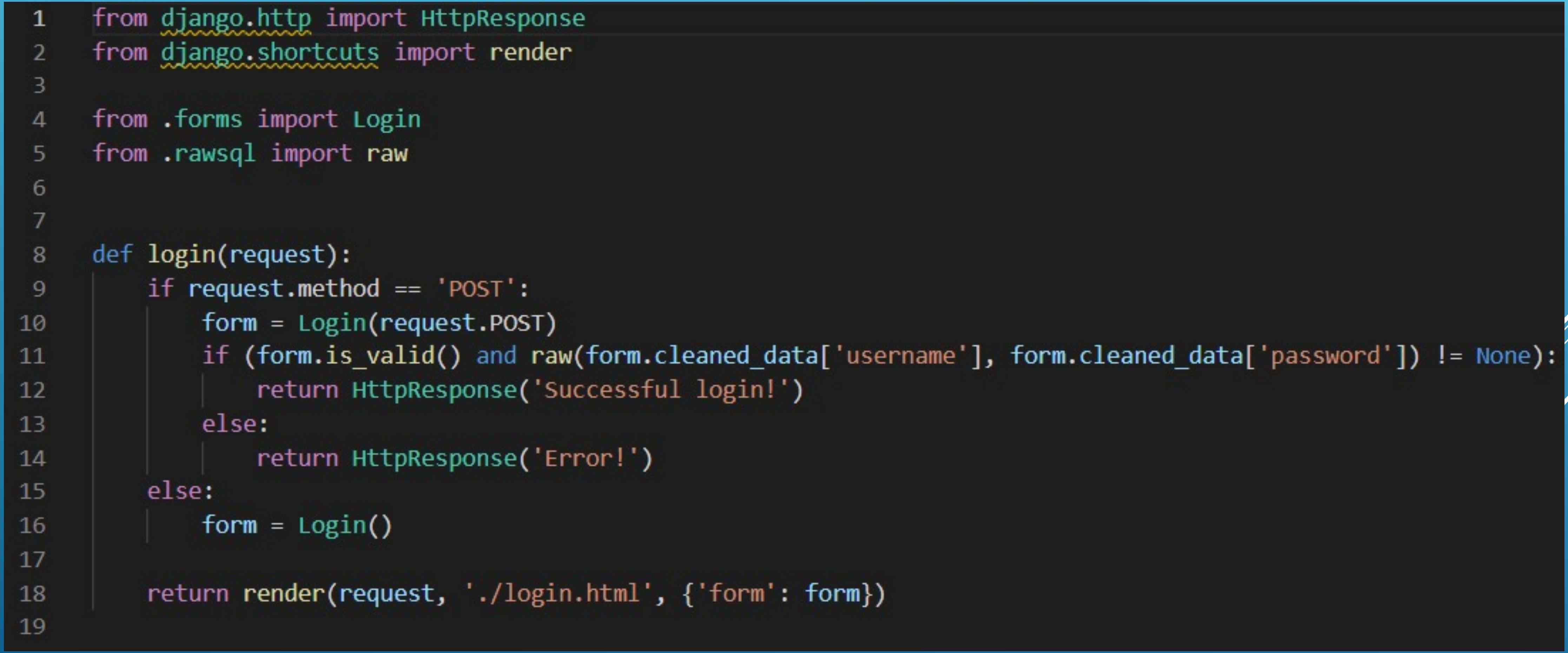
rawsql.py



```python
from django.db import connection


def raw(username, password):
    with connection.cursor() as cursor:
        cursor.execute(
            "SELECT username FROM users WHERE username = '" + username + "' AND password = '" + password + "'")
        row = cursor.fetchone()

    return row

```

BAD WAY

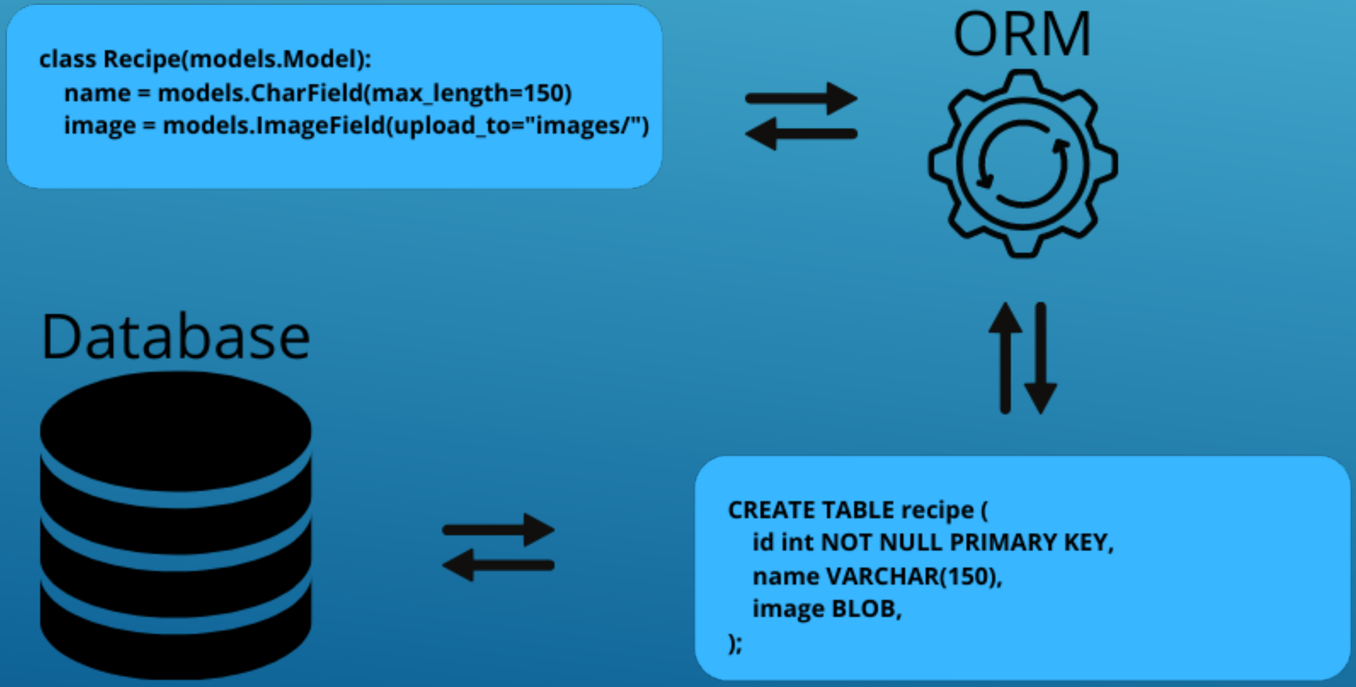
views.py



```python
from django.http import HttpResponse
from django.shortcuts import render

from .forms import Login
from .rawsql import raw


def login(request):
    if request.method == 'POST':
        form = Login(request.POST)
        if (form.is_valid() and raw(form.cleaned_data['username'], form.cleaned_data['password']) != None):
            return HttpResponse('Successful login!')
        else:
            return HttpResponse('Error!')
    else:
        form = Login()

    return render(request, './login.html', {'form': form})

```

SOLUTION 1: ORM

- Object-relational mapping
- Transmit data between a relation database and application model
- ORM automates this transmission, such that the developer doesn't have to write any SQL



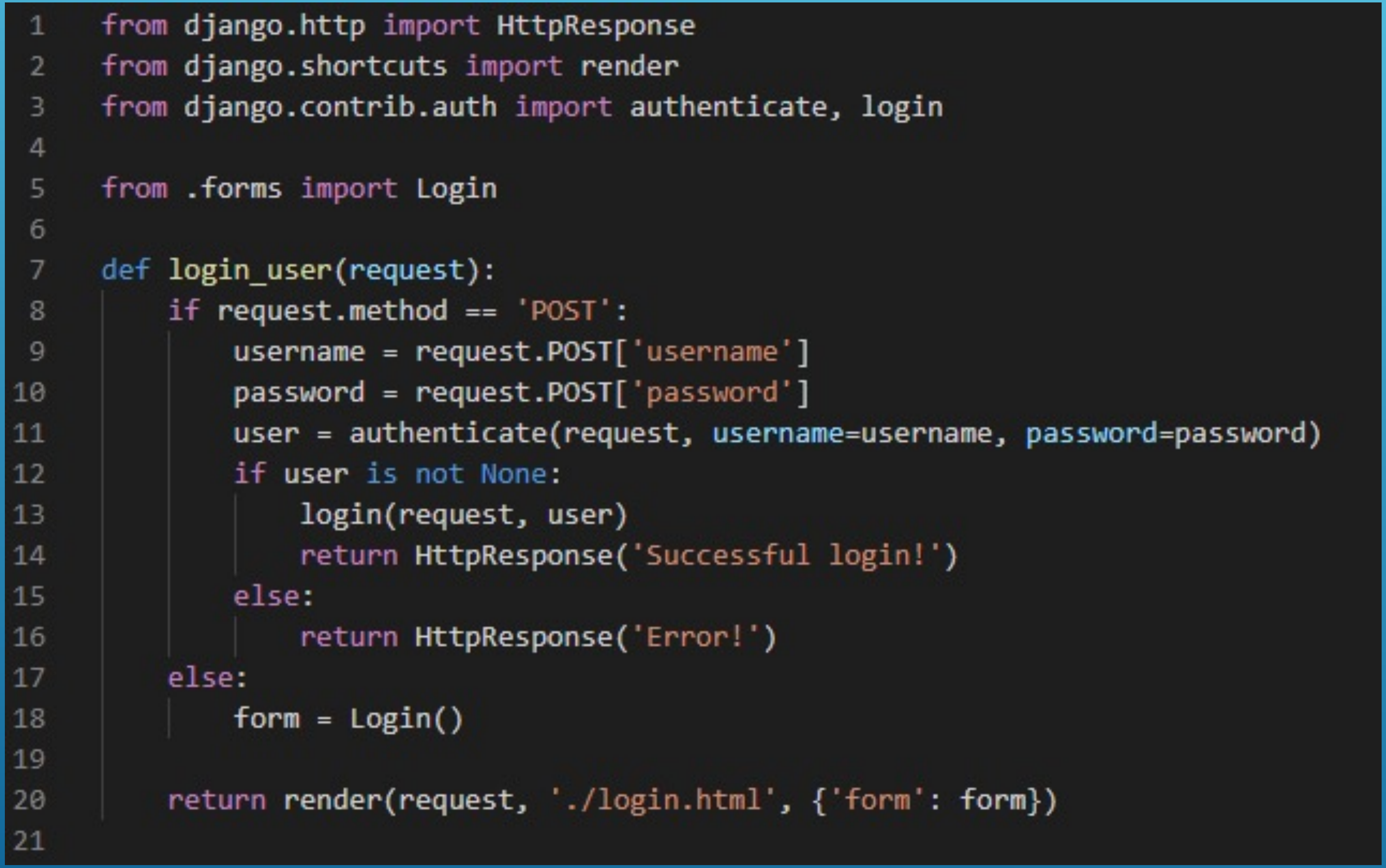
```
class Recipe(models.Model):
    name = models.CharField(max_length=150)
    image = models.ImageField(upload_to="images/")
```

```
CREATE TABLE recipe (
    id int NOT NULL PRIMARY KEY,
    name VARCHAR(150),
    image BLOB,
);
```

Source: https://engineertodeveloper.com/wp-content/uploads/2021/01/django_orm_diagram_1.png

SOLUTION 1: ORM

views.py



```python
from django.http import HttpResponse
from django.shortcuts import render
from django.contrib.auth import authenticate, login

from .forms import Login

def login_user(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return HttpResponse('Successful login!')
        else:
            return HttpResponse('Error!')
    else:
        form = Login()

    return render(request, './login.html', {'form': form})

```

SOLUTION 2: PARAMETERIZED SQL QUERIES

```
Person.objects.raw('SELECT * FROM myapp_person WHERE last_name = %s', [lname])
```

%s placeholder will be replaced with parameters from the params argument (“lname”).

SQL INJECTION PROTECTION

“ Django’s querysets are protected from SQL injection since their queries are constructed using query parameterization. A query’s SQL code is defined separately from the query’s parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver. “

Source: https://docs.djangoproject.com/en/3.2/topics/security/#sql-injection-protection