NIET Greater Noida
Get Future Ready Autonomous Institute

Course code ACSE0512
Course Name : Python web development with Django

UNIT-IV: Integrating Accounts & Authentication on Django

Database Migrations, Fetch Data From Database, Displaying Data On Templates, Adding Condition On Data, Sending data from url to view, Sending data from view to template, Saving objects into database, Sorting objects, Filtering objects, Deleting objects, Difference between session and cookie, Creating sessions and cookies in Django.

NIET Greater Noida Autonomous Institute

Unit II Objective

- Database Migrations.
- Fetch Data From Database.
- Displaying Data On Templates.
- Adding Condition On Data.
- Sending data from url to view.
- Sending data from view to template.
- Saving objects into database, Sorting objects, Filtering objects, Deleting objects.
- Difference between session and cookie, Creating sessions and cookies in Django.

NIET Greater Noida Autonomous Institute

Unit IV Objective

In Unit IV, the students will be able to find

Definitions of terms Database Migrations.

How Fetch Data From Database & Displaying Data On Templates.

How to Send data from url to view.

How to Send data from view to template.

The idea of a python Library .

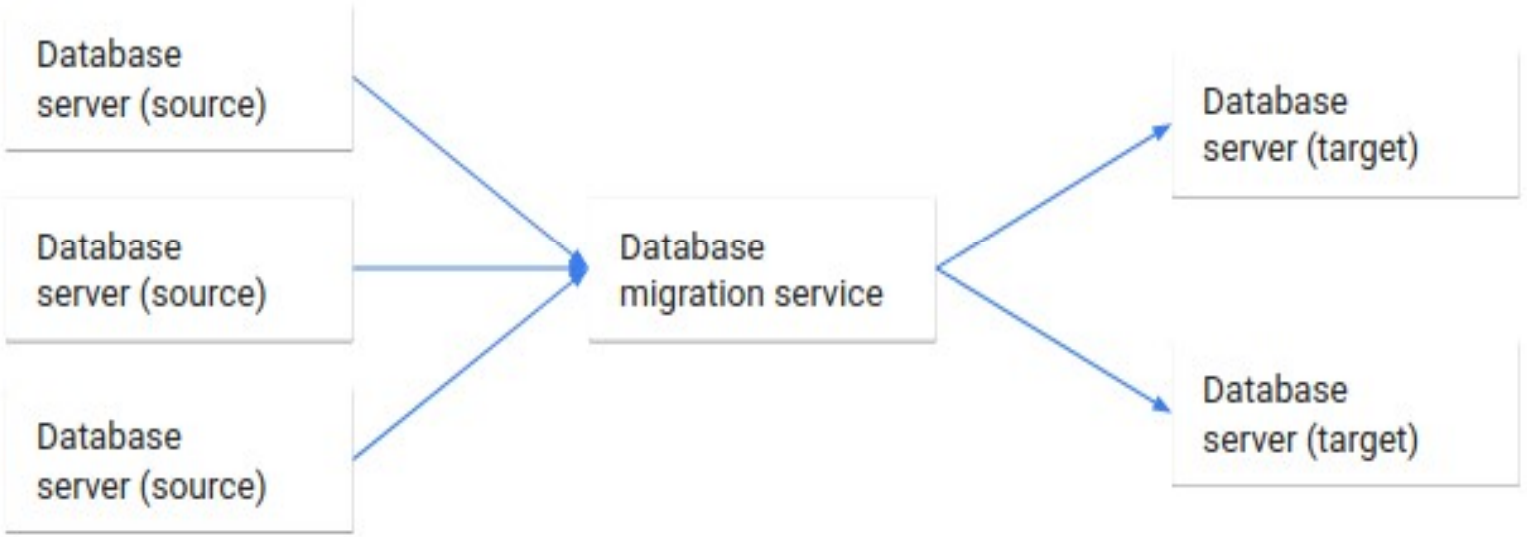
Difference between session and cookie, Creating sessions and cookies in Django

Database migration: Concepts and principles

- Database **migration is the process** of migrating data from one or more source databases to one or more target databases by using a database migration service.
- When a migration is finished, the dataset in the **source databases** resides fully, though possibly restructured, in the **target databases**. Clients that accessed the source databases are then switched over to the target databases, and the source databases are turned down.
- A database migration service runs within **Google Cloud** and accesses both source and target databases.
- Two variants are represented: (a) shows the **migration** from a source database in an **on-premises** data **center or a remote cloud** to a managed database like **Cloud Spanner**; (b) shows a migration to a database on **Compute Engine**

Database Migration Process

The following diagram illustrates this database migration process.

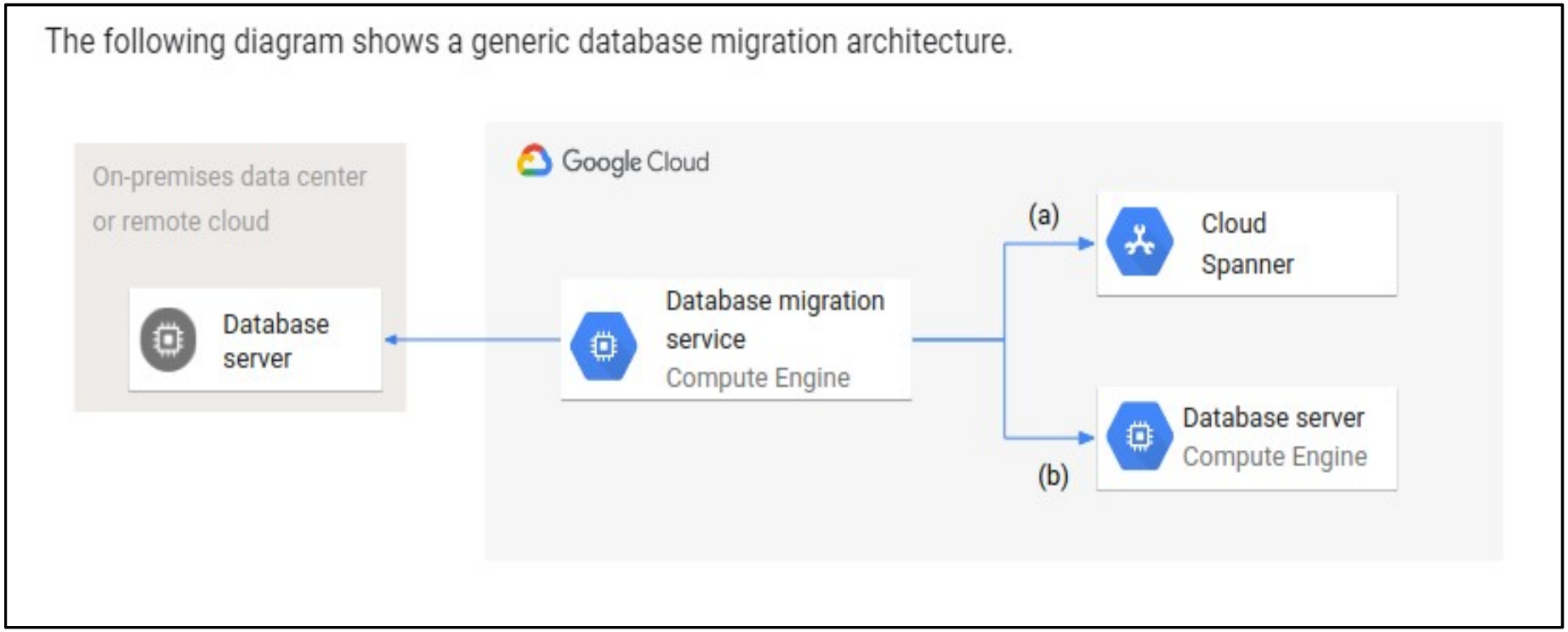


Database Migration Process

We describe Database migration from an architectural stand pointwise

- The services and technologies involved in database migration.
- The differences between **homogeneous and heterogeneous** database migration.
- The **tradeoffs** and selection of a **migration downtime tolerance**.
- A setup architecture that supports a fallback if unforeseen errors occur during a migration.
- This document does not describe how you set up a particular database migration technology. Rather, it introduces database migration in fundamental, conceptual, and principle terms.

Database Migrations



Even though the target **databases** are different in type (managed and unmanaged) and setup, the database migration architecture and configuration is the same for both cases.

Database Migrations - Terminology

The most important data migration terms for these documents are defined as follows:

source database: A database that contains data to be migrated to one or more target databases.

target database: A database that receives data migrated from one or more source databases.

database migration: A migration of data from source databases to target databases with the goal of turning down the source database systems after the migration completes. The entire dataset, or a subset, is migrated.

homogeneous migration: A migration from source databases to target databases where the source and target databases are of the same database management system from the same provider.

heterogeneous migration: A migration from source databases to target databases where the source and target databases are of different database management systems from different providers.

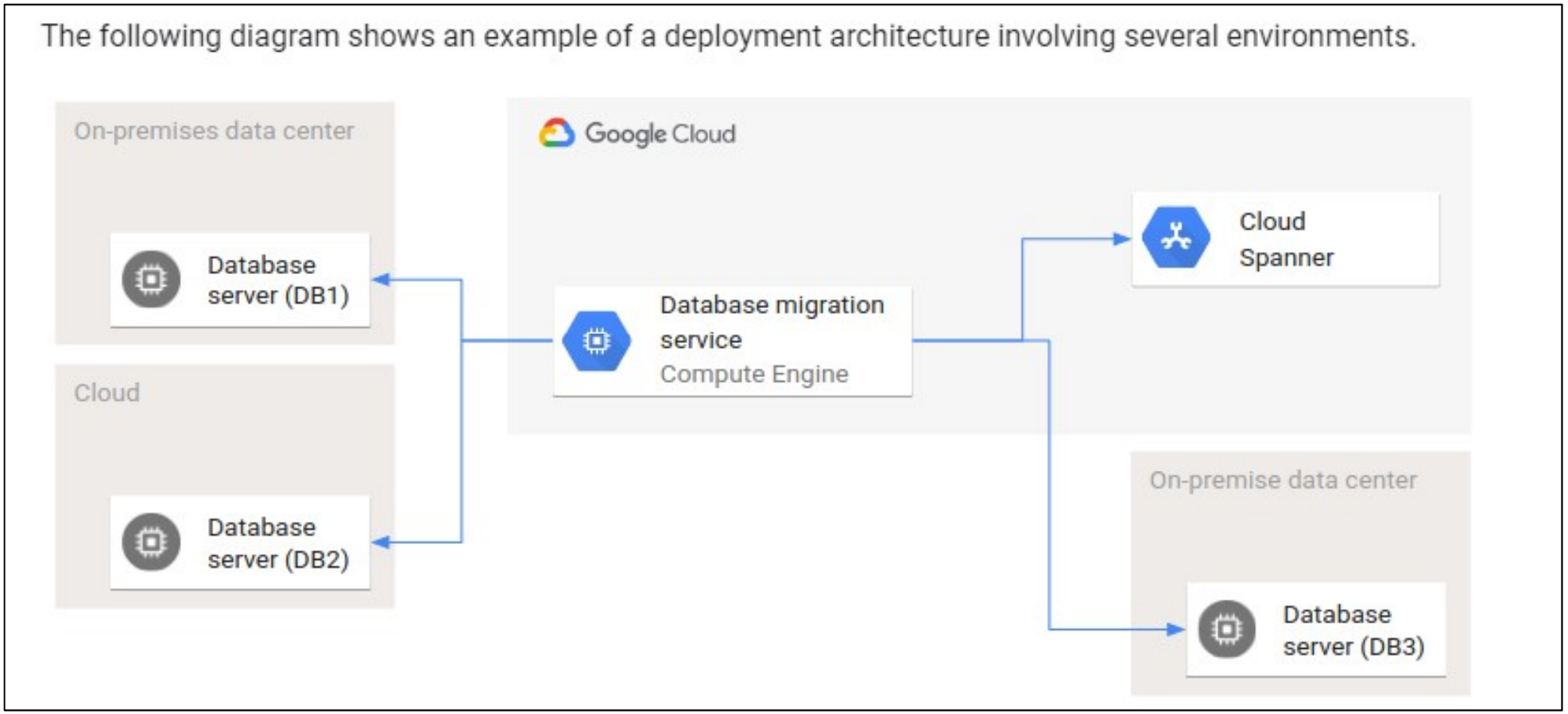
Database Migrations - Terminology

database migration system: A software system or service that connects to source databases and target databases and performs data migrations from source to target databases.

data migration process: A configured or implemented process executed by the data migration system to transfer data from source to target databases, possibly transforming the data during the transfer.

database replication: A continuous transfer of data from source databases to target databases without the goal of turning down the source databases. Database replication (sometimes called *database streaming*) is an ongoing process.

Introduction to Django Authentication System



NIET Greater Noida
Get Future Ready Autonomous Institute

Migrations in Django

Migrations are Django's way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your **database schema**.

They're designed to be mostly automatic, but you'll need to know when to make migrations, when to run them, and the common problems you might run into.

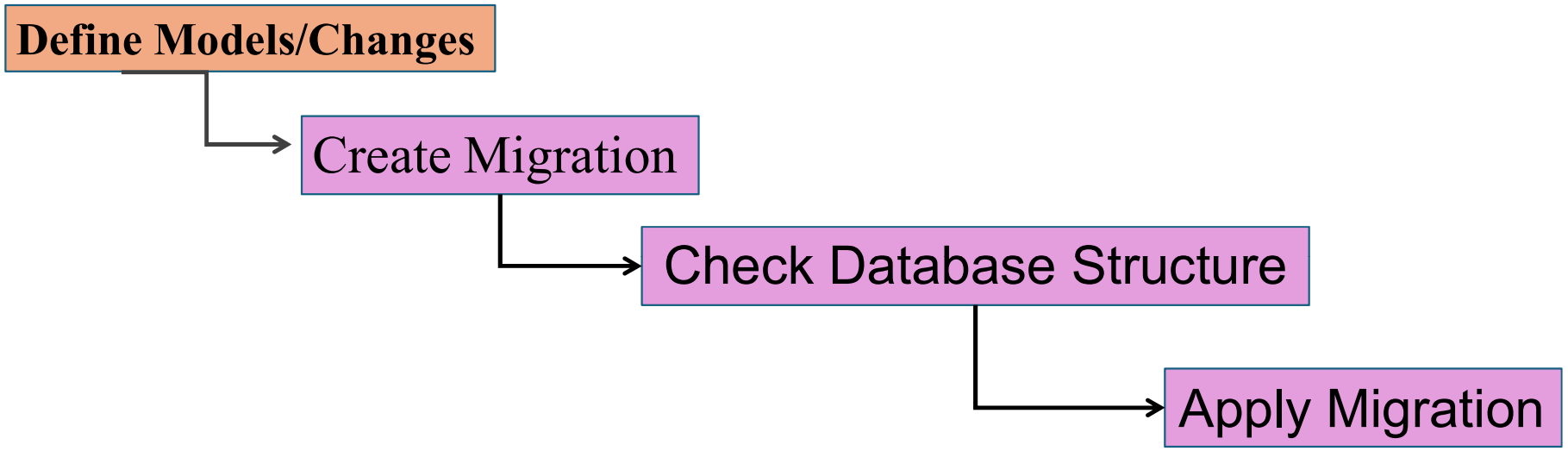
There are several commands to handling database schema in django.

These commands depend on which type of database is used

By **default** django provide **SQLite3**.

Migrations in Django

Django Basic Migration Workflows



Migrations in Django

Migrate -which is responsible for applying and unapplying migrations.

python mange.py migrate

makemigrations basically generates the SQL commands for preinstalled apps (which can be viewed in installed apps in **settings.py**) and your newly created app's model which you add in **installed** apps whereas migrate executes those SQL commands in the database file.

python manage.py makemigrations

Migrations in Django

Migration of custom models in Django

1. **create a django project**

   django-admin startproject myproject

2. now get into the project directory

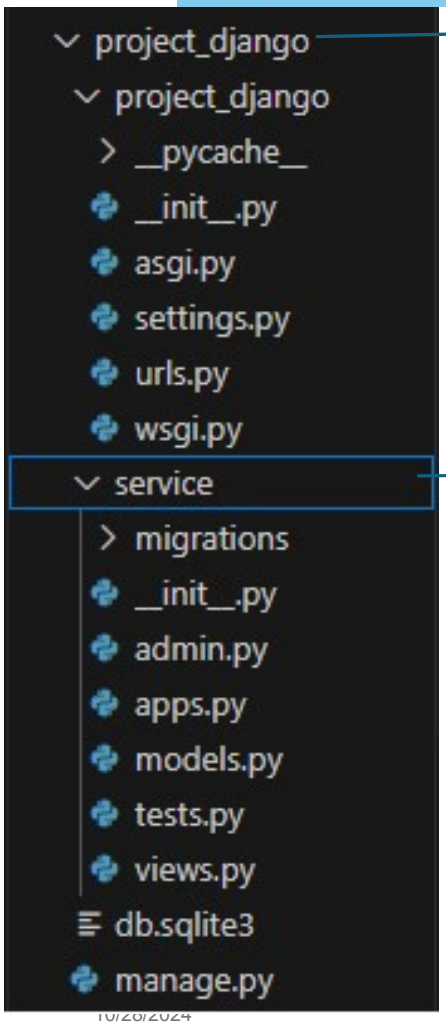
   cd myproject

3. create django app

   python manage.py startapp myapp

Above commands create a structure of your project. In next slide

Migrations in Django



Main project folder

App in django project

Migrations in Django

4. After creating app
   place the create app information in settings.py
   and app name Installed_apps

```
INSTALLED_APPS = [courses,]
```

5. Open models.py from your app folder

```
from django.db import models
class Courses(models.Model):
courseName = models.CharField(max_length=50)
courseBranch = models.CharField(max_length=50)
courseDetails = models.TextField(verbose_name="Course Details")
```

Migrations in Django

6. Create and apply commands to migrations
    1. python manage.py makemigrations
    2. python mange.py migrate

Once these migrations complete you and add your custom model in admin section

Create Superuser and map custom model to admin

1. Create an Admin User (Optional)

   python manage.py createsuperuser

2. Register the Model in Admin(in django app write the code in admin.py)

```
from django.contrib import admin
from courses.models import Courses

class CoursesAdmin(admin.ModelAdmin):
    list_display= "courseName","courseBranch","courseDetails“)
admin.site.register(Courses,CoursesAdmin)
```

Fetch Data From Database

write the code in views.py file of your django app. Example below

```
from django.shortcuts import render
from .models import Data
def courselst(request):
    courseData = Courses.objects.all()
    #print(courseData)
     return
 render(request,'myapp/list_course.html',{'courseData‘:courseData})
```

Once the view processed the data(fetch the data, process logic)

Data pass from view to template in the form of context dictionary.

The context dictionary key used as a variable for the template.

Displaying Data On Templates

We need to add a new folder ie templates to kept UI files of your project

Create a template file at courses/templates/myapp/list_course.html

key used as a variable for the template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home Page</title>
</head>
<body>
    <div>
    <h1>Course Detail</h1>
    <table border="1">
        <thead>
            <tr>
                <th>Course Name</th>
                <th>Course Branch</th>
                <th>Course Details</th>
                <!-- Add more columns for additional fields -->
            </tr>
        </thead>
        <tbody>
            {% for courseData in courseData %}
                <tr>
                    <td>{{ courseData.courseName }}</td>
                    <td>{{ courseData.courseBranch}}</td>
                    <td>{{ courseData.courseDetails }}</td>
                    <!-- Add more cells for additional fields -->
                </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
</body>
</html>
```

Set Up a URL for the View

```
from django.contrib import admin
from django.urls import path
from . import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.courselst, name="course_list"),
  ]
```

*note this files in app folder of django project

NIET Greater Noida Autonomous Institute

Set Up a URL for the View

Now map app urls.py to main project urls.py file

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

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

NIET Greater Noida
Get Future Ready Autonomous Institute

Sending data from url to view

In Django, URL parameters are a way to pass data to views via the URL path.

These parameters allow you to create dynamic web pages that respond to user inputs.

URL parameters are often used to retrieve specific records from a database, perform searches, or filter data based on user preferences.

What Are URL Parameters?

url parameters are the segment that provide the additional information to the server

In Django url parameters are captrued in the URL patterns and then passed to the corresponding views

Sending data from url to view

Benefits of URL

1. Dynamic
2. SEO friendly
3. Clean Url

Creating URL Patterns in Django

Django provide patterns to map **Urls (urls.py)** to views

Steps to create url

1. create a urls.py file in your app. If not exist.
2. Define url patterns : Use django **path()** to create url or route

NIET Greater Noida Autonomous Institute

Sending data from url to view

Example

```
from django.urls import path
from firstproject import views
urlpatterns = [
    path('', views.homePage),
    path('admin/', admin.site.urls),
    path('about-us/', views.aboutUs),
    path('course', views.Course),
    path('course/<slug:course_id>', views.courseDetatils)
  ]
```

Sending data from url to view

Include App URLs in Project URLs

```
urlpatterns = [
    path('', include('courses.urls')),
]
```

Getting URL Parameters

- To capture the parameters is the curical steps to create dynamic django applications.
- It helps user to create more interactive web pages.

How to capture URL Parameters

URL parameters are captured using angle brackets (<>) in the URL pattern

Sending data from url to view

```
# urls.py
    from django.urls import path
    from . import views urlpatterns = [
    path('product/<int:product_id>/', views.product_detail,
    name='product_detail'),
    ]
```

Sending data from url to view

Types of URL Parameters

Various types of URL parameters used in Django

1. String: <str:parameter_name>

   Passing string parameters in the Url . Ex </users/<str:username>

2. Integer: <int:parameter_name>

   only numbers are accepted. Ex </products/<int:product_id>

3. Slug: <slug:parameter_name>

4. Human readable string parameter and support SEO . This mainly contains lowercase letters, numbers, hyphens, and underscores.

Sending data from url to view

Example of slug

/articles/<slug:article_slug>

5. UUID: <uuid:parameter_name>

   Universally Unique Identifier (Ex order_id etc)

   /orders/<uuid:order_id>

   URL: /orders/123e4567-e89b-12d3-a456-426614174000

Sending data from url to view

. Path: <path:parameter_name>

Similar to a string, but can include slashes, allowing for multi-segment paths.

Example of path

/files/<path:file_path>

URL: /files/documents/reports/2024/file.pdf

Sending data from url to view

Accessing URL Parameters

```
# views.py
from django.shortcuts import render, get_object_or_404
from .models import Product
 def product_detail(request, product_id):
        product = get_object_or_404(Product, id=product_id)
        return  render(request, 'product_detail.html', {'product': product})
```

Sending data from url to view

Passing Multiple Parameters

```
# urls.py
urlpatterns = [ path('blog/<int:year>/<int:month>/', views.archive,
name='archive'), ] # views.py
def archive(request, year, month):
        # Access the year and month parameters
        # Perform actions based on the parameters return
        render(request, 'archive.html', {'year': year, 'month': month})
```

Sending data from url to view

Passing Multiple Parameters

```
# urls.py
urlpatterns = [ path('blog/<int:year>/<int:month>/', views.archive,
name='archive'), ] # views.py
def archive(request, year, month):
        # Access the year and month parameters
        # Perform actions based on the parameters return    render(request,
'archive.html', {'year': year, 'month': month})
```

Sending data from url to view

Built-in Parameter Validation

Django's URL patterns **automatically validate parameters** based on their type.

For example, if you define a parameter as

<int:parameter_name>

Django will ensure that the parameter is an integer before passing it to the view function.

Operations in objects

In Python, **saving, sorting, filtering, and deleting objects** in a database is commonly handled through an **Object-Relational Mapping (ORM) system**.

ORMs allow developers to interact with the database using Python objects rather than writing raw SQL queries

Saving Object to database

When we create a **new entities(records**) in database or update existing records.

Django provide **.save()** method, which is part of the Django ORM.

An objects in django does not saved if don't use **.save()**

NIET Greater Noida Autonomous Institute

Operations in objects

```
from django.db
import models class Student(models.Model):
name = models.CharField(max_length=100)
age = models.IntegerField()
# Saving an object to the database
student = Student(name="John", age=21)
student.save() # Saves the object to the database
```

Operations in objects

Sorting an objects

Arranging the records in a specific order, usually based on a field or fields of the object.

Django ORM provides .order_by()

By default ascending **order, for descending** order by **prefixing** the **field name** with a minus (-) sign.

```
# Fetch all students sorted by age in ascending order
students = Student.objects.all().order_by('age')
# Fetch all students sorted by name in descending order
students = Student.objects.all().order_by('-name')
```

Operations in objects

Filtering Objects

Filtering objects involves **retrieving** a **subset** of records that **match certain conditions**.

Django ORM provides the **.filter()** method to query the database for objects that meet certain criteria.

```
# Fetch students who are older than 20
students = Student.objects.filter(age__gt=20)
# Fetch students whose name contains
'John'students = Student.objects.filter(name__icontains='John')
```

Operations in objects

Deleting Objects

To remove objects from database . In django **.delete()** method is used to delete the object.

.delete() method permanently removed from the database.

Examples

```
1. # Fetch a specific student and delete
student = Student.objects.get(name="John")
student.delete()  # Deletes the object from the database

2. # Delete all students older than 20
Student.objects.filter(age__gt=20).delete()
```

Session and Cookie

- **Cookies**, technically called HTTP Cookies are small text files which are created and maintained by your browser on the particular request of Web-Server.
- They are stored locally by **your browser**, and most browser will also show you the **cookies generated in the Privacy and Security settings** of the browser.
- **HTTP is a stateless protocol**. When any request is sent to the server, over this protocol, the server cannot distinguish whether the user is new or has visited the site previously.

Session and Cookie

- Suppose, you are logging in any website, that website will respond the browser with some cookies which will have some **unique identification** of user generated by the server and some more details according to the context of the website.
- Cookies made these **implementations possible** with ease which were previously not possible over **HTTP** implementation.

How do Cookies work?

Cookies work like other **HTTP requests** over the Internet. In a typical web-system, the **browser makes a request to the server**. The server then sends the response along with some cookies, which may contain some login information or some other data.
When the browser makes a new request, the cookie generated previously is also transmitted to the server.
This process is repeated every time a new request is made by the browser.

Session and Cookie

The **browser repeats** the process **until the cookie expires** or the session is closed and the cookie is deleted by the browser itself.

Then, the cookie applies in all sorts of tasks, like when your login to a website or when shopping online on the web. **Google AdSense** and **Google Analytics** can also track you using the **cookies** they generate.
Different websites use **cookies differently** according to their needs.

Creating Cookies in Django

Django **bypasses** lots of work which otherwise would be required when working on cookies.

Django has methods like set_cookie() which we can use to create cookies very easily.

The Django HttpResponse object has a set_cookie() method

set_cookie(key, value='', max_age=None, expires=None, path='/', domain=None, secure=None, httponly=False, samesite=None) :

Attributes of set_cookie() are

Continue……..

Session and Cookie

1. **name**: Name of the cookie
   value: Value you want to store in the cookie. You can **set int** or **string** but it will **return string**.

2. **max_age**: Should be a number of seconds, **or None (default)** . if the cookie should last only as long as the client's browser session. If expires is not specified, it will be calculated.

3. **expires:** Should either be a string in the format "Wdy, DD-Mon-YY HH:MM:SS GMT" or a datetime. datetime object in UTC. If expires is a datetime object, the max_age will be calculated.

Read cookie the **request.COOKIES**

Every Django request object has a **COOKIES attribute** which is a **dictionary**.

We can use **COOKIES** to **read** a cookie value, which **returns a string** even though you **set an integer value**

request.COOKIES['cookie_name']

Session and Cookie

Let's take an example.

Create a view in your `views.py` as below:

```
def test_cookie(request):
    if not request.COOKIES.get('team'):
        response = HttpResponse("Visiting for the first time.")
        response.set_cookie('team', 'barcelona')
        return response
    else:
        return HttpResponse("Your favorite team is
{}".format(request.COOKIES['team']))
```

To read cookie

Create cookie

Now, add the URL for this view in `urls.py`.

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

Delete or update a cookie

Syntax response.delete_cookie('cookie_name')

Important points when using cookies

1. Never ever use cookies to store sensitive data like passwords. Cookies store data in plain text, as a result, anybody can read/modify them.

2. Most browsers don't allow cookies to store more than 4KB of data (i.e. 4KB for each cookie).

Important points when using cookies

3. Further, most browsers accept no more than 30 cookies per website.(exact number of cookies per website)
4. Recall that once the cookie is set in the browser, it will be sent along with each request to the server.
5. Users can delete the cookies at their will. The user can even configure their browsers to not accept cookies at all.

Session and Cookie

What are session?

A session refers to a way of **maintaining state information** about a user's **interactions** with a website or web application.

When a user visits a website, the **server** can **create a session** for that user. Additionally, a session allows the server to keep track of information such as the user's login status, preferences, and any data entered into forms.

The server typically initiates a session when a user logs in to a website. Furthermore, **we can identify a session by a unique session ID.**

Session and Cookie

We pass the **session IDs** as a parameter in URLs or store them in the cookies.

The **session ID** allows the **server** to associate the user's requests with their specific session. Additionally, it also helps to retrieve and update the session data as needed.

How Do Web Sessions Work?

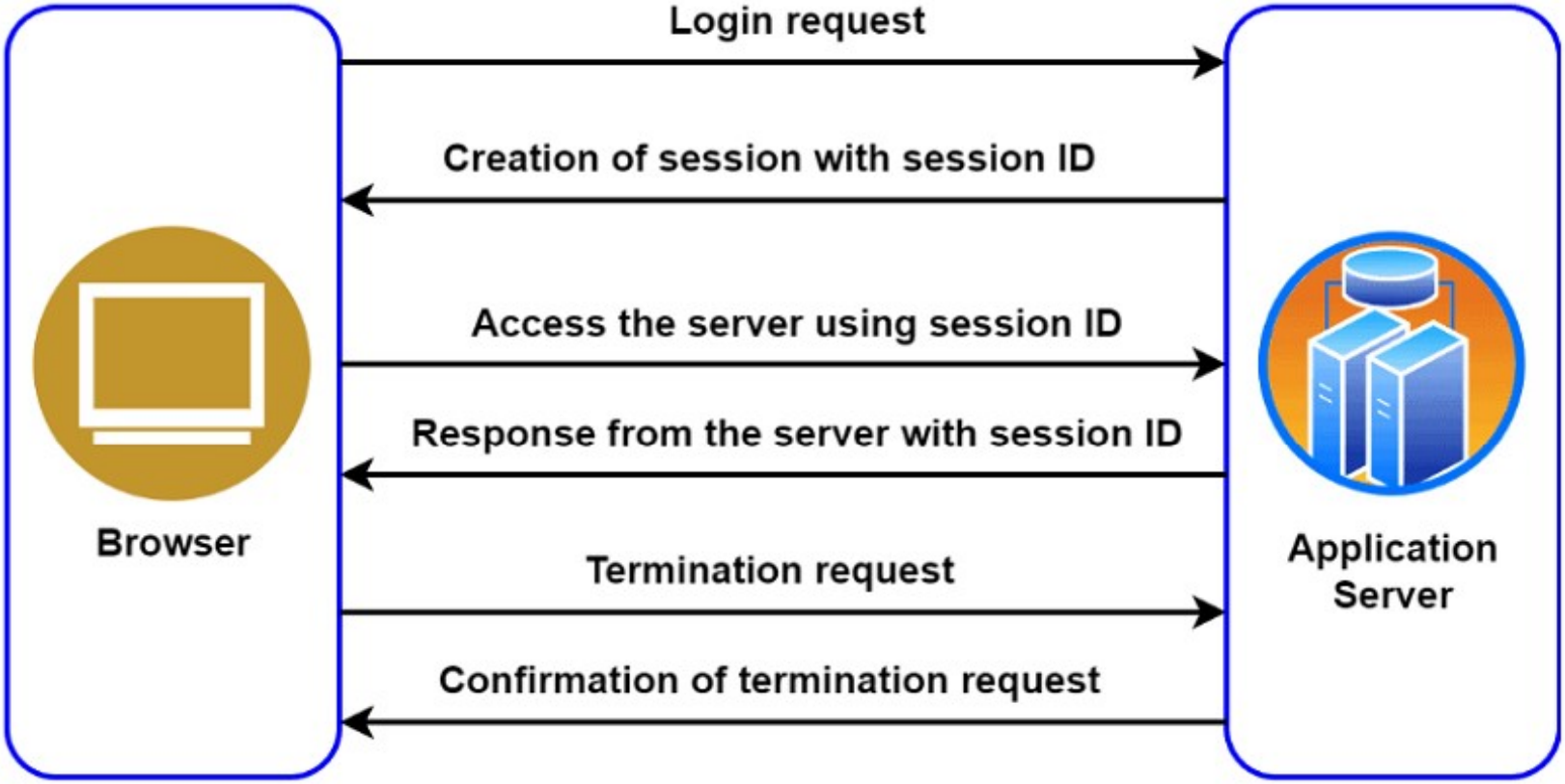
A web session is a period of **interaction between a user and a website**.

Furthermore, the website **maintains state** information about **the user's actions** and **preferences** during a session.

The server can **initiate a session** for a user when they browse through a website. The session **remains active** until the **user logs out**.

next slide

Session and Cookie



Working in web session

NIET Greater Noida
Autonomous Institute

Session and Cookie

Cookies	Session
Cookies are client-side files on a local computer that hold user information.	Sessions are server-side files that contain user data.
Cookies end on the lifetime set by the user.	When the user quits the browser or logs out of the programmed, the session is over.
It can only store a certain amount of info.	It can hold an indefinite quantity of data.
The browser's cookies have a maximum capacity of 4 KB.	We can keep as much data as we like within a session, however there is a maximum memory restriction of 128 MB that a script may consume at one time.
Because cookies are kept on the local computer, we don't need to run a function to start them.	To begin the session, we must use the session start() method.
Cookies are not secured.	Session are more secured compare than cookies.
Cookies stored data in text file.	Session save data in encrypted form.

Creating & Accessing Django Sessions

Django allows you to easily create session variables and manipulate them accordingly.

The request object in Django has a session attribute, which creates, access and edits the session variables.

This attribute acts like a dictionary, i.e., you can define the session names as keys and their value as values.

Session in Django

Step1 : Check middleware is in your settings.py (it is by default but make sure)

1. Ensure that in **settings.py** file **MIDDLEWARE** having this

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',   # Session middleware
    'django.middleware.common.CommonMiddleware',
    # other middleware…
```

Sessions continue…..

Step 2: Storing data in a session

request.session object is to store data into sessions.

```
# views.py
    from django.shortcuts import render, redirect
    def set_session(request):
        # Set session data
        request.session['username'] = 'Shalini'
        request.session['is_authenticated'] = True
        return redirect('get_session‘)
```

Step 3: Retrieve values from the session dictionary in Django request.session.get

```
        # views.py
def get_session(request):
    # Get session data
    username = request.session.get('username', 'Guest')
    is_authenticated = request.session.get('is_authenticated', False)
    #  send session varibales to template from view
    return render(request, 'session_example.html', {
        'username': username,
        'is_authenticated': is_authenticated,
    })
```

Sessions continue…….

Display session data in template (optional)

```
<!-- session_example.html -->
<!DOCTYPE html>
<html>
<body>
   <h1>Welcome {{ username }}</h1>
   <p>Authenticated: {{ is_authenticated }}</p>
</body>
</html>
```

Sessions continue…….

Step 4: Clearing session data .

You can **clear session** data when required, either by **removing specific** items or **clearing the entire session**.

```
# views.py
        def clear_session(request):
            # Clear specific key
            if 'username' in request.session:
                del request.session['username']

            # Clear all session data
            request.session.flush()
            return redirect('get_session')
```

Sessions continue…….

Session Expiration:

We can set the custom time to expire the session valriables

For this we add code in settings.py

```
# Session expires after 30 minutes of inactivity
# 30 minutes
        SESSION_COOKIE_AGE = 60 * 30
         # Expire when browser closes
        SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

NIET
Greater Noida
Autonomous Institute

THANK YOU