**Advantages of using Django:**

* Rich Ecosystem: It comes with numerous third-party apps which can be easily integrated as per the requirements of the project.
* Maturity:  Django has been in use for over a decade. In the time frame, a lot of features are added and enhanced to make it a Robust framework. Apart from that, there are a large number of developers who are using Django.
* Admin panel: Django provides an admin dashboard that we can use to do basic CRUD operations over the models.
* Plugins: Allow programmers to add various features to applications and leave sufficient space for customization.
* Libraries: Due to the large development community there is an ample number of libraries for every task.
* ORM: It helps us with working with data in a more object-oriented way.

### Explain Django Architecture?

Django follows the MVT (Model View Template) pattern which is based on the Model View Controller architecture. It’s slightly different from the MVC pattern as it maintains its own conventions, so, the controller is handled by the framework itself. The template is a presentation layer. It is an HTML file mixed with Django Template Language (DTL). The developer provides the model, the view, and the template then maps it to a URL, and finally, Django serves it to the user.

### What are models in Django?

### A model in Django refers to a class that maps to a database table or database collection. Each attribute of the Django model class represents a database field. They are defined in app/models.py

### What are views in Django?

A view function, or “view” for short, is simply a Python function that takes a web request and returns a web response. This response can be HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image, etc.

There are two types of views:

* **Function-Based Views:**In this, we import our view as a function.
* **Class-based Views**: It’s an object-oriented approach.

### What is Django ORM?

This ORM (an acronym for Object Relational Mapper) enables us to interact with databases in a more pythonic way like we can avoid writing raw queries, it is possible to retrieve, save, delete and perform other operations over the database without ever writing any SQL query. It works as an abstraction layer between the models and the database.

### Define static files and explain their uses?

Websites generally need to serve additional files such as images. Javascript or CSS. In Django, these files are referred to as “static files”, Apart from that Django provides django.contrib.staticfiles to manage these static files.

### What is Django Rest Framework(DRF)?

Django Rest Framework is an open-source framework based upon Django which lets you create RESTful APIs rapidly.

**What is django-admin and manage.py and explain its commands?**

django-admin is Django’s command-line utility for administrative tasks. In addition to this, a manage.py file is also automatically created in each Django project. Not only does it perform the same purpose as the django-admin but it also sets the DJANGO\_SETTINGS\_MODULE environment variable to point to the project's settings.py file.

* django-admin help - used to display usage information and a list of the commands provided by each application.
* django-admin version - used to check your Django version.
* django-admin check - used to inspect the entire Django project for common problems.
* django-admin compilemessages - Compiles .po files created by makemessages to .mo files for use with the help of built-in gettext support.
* django-admin createcachetable - Creates the cache tables for use in the database cache backend.
* django-admin dbshell - Runs the command-line client for the database engine specified in your ENGINE setting(s), with the connection parameters (USER, PASSWORD, DB\_NAME, USER etc.) specified settings file.
* django-admin diffsettings - Shows the difference between the existing settings file and Django’s default settings.
* django-admin dumpdata - Used to the dumpdata from the database.
* django-admin flush - Flush all values from the database and also re-executes any post-synchronization handlers specified in the code.
* django-admin inspectdb - It generates django models from the existing database tables.
* django-admin loaddata - loads the data into the database from the fixture file.
* django-admin makemessages - Used for translation purpose and it generates a message file too.
* django-admin makemigrations - Generates new migrations as per the changes detected to your models.
* django-admin migrate - Executes SQL commands after which the database state with the current set of models and migrations are synchronized.
* django-admin runserver - Starts a light-weight Web server on the local machine for development. The default server runs on port 8000 on the IP address 127.0.0.1. You can pass a custom IP address and port number explicitly if you want.
* django-admin sendtestemail - This is used to confirm email sending through Django is working by sending a test email to the recipient(s) specified.
* django-admin shell - Starts the Python interactive interpreter.
* django-admin showmigrations - Shows all migrations present in the project.
* django-admin sqlflush - Prints the SQL statements that would be executed for the flush command mentioned above.
* django-admin sqlmigrate - Prints the SQL statement for the named migration.
* django-admin sqlsequencereset - output the SQL queries for resetting sequences for the given app name(s).
* django-admin squashmigrations - Squashes a range of migrations for a particular app\_label.
* django-admin startapp - Creates a new Django app for the given app name within the current directory or at the given destination.
* django-admin startproject - Creates a new Django project directory structure for the given project name within the current directory or at the given destination.
* django-admin test - Runs tests for all installed apps.
* django-admin testserver - Runs a Django development server (which is also executed via the runserver command) using data from the given fixture(s).
* django-admin changepassword - offers a method to change the user's password.
* django-admin createsuperuser - Creates a user account with all permissions(also known as superuser account).
* django-admin remove\_stale\_contenttypes - removes stale content types (from deleted models) in your database.
* django-admin clearsessions - Can be used to clean out expired sessions or as a cron job.

**10. What is Jinja templating?**

Jinja Templating is a very popular templating engine for Python, the latest version is Jinja2.

Some of its features are:

* Sandbox Execution - This is a sandbox (or a protected) framework for automating the testing process
* HTML Escaping - It provides automatic HTML Escaping as <, >, & characters have special values in templates and if using a regular text, these symbols can lead to XSS Attacks which Jinja deals with automatically.
* Template Inheritance
* Generates HTML templates much faster than default engine
* Easier to debug as compared to the default engine.

**11. What are Django URLs?**

URLs are one of the most important parts of a web application and Django provides you with an elegant way to design your own custom URLs with help of its module known as URLconf (URL Configuration). The basic functionality of this python module is to   
You can design your own URLs in Django in the way you like and then map them to the python function (View function). These URLs can be static as well as dynamic. These URLs as present in the urls.py where they are matched with the equivalent view function.

Basic Syntax:

**from** django.urls **import** path

**from** . **import** views

urlpatterns = [

path('data/2020/', views.data\_2020),

path('data/<int:year>/', views.data\_year)

]

### What is the difference between a project and an app in Django?

In simple words Project is the entire Django application and an app is a module inside the project that deals with one specific use case.   
For eg, payment system(app) in the eCommerce app(Project).

**What are different model inheritance styles in the Django?**

* **Abstract Base Class Inheritance**: Used when you only need the parent class to hold information that you don’t want to write for each child model.
* **Multi-Table Model Inheritance:**  Used when you are subclassing an existing model and need each model to have its own table in the database.
* **Proxy Model Inheritance:**  Used when you want to retain the model's field while altering the python level functioning of the model.

**What are Django Signals?**

Whenever there is a modification in a model, we may need to trigger some actions. Django provides an elegant way to handle these in the form of signals. The signals are the utilities that allow us to associate events with actions. We can implement these by developing a function that will run when a signal calls it.

**List of built-in signals in the models:**

| **Signals** | **Description** |
| --- | --- |
| django.db.models.pre\_init & django.db.models.post\_init | Sent before or after a models’s \_init\_() method is called |
| django.db.models.signals.pre\_save & django.db.models.signals.post\_save | Sent before or after a model’s save() method is called |
| django.db.models.signals.pre\_delete & django.db.models.signals.post\_delete | Sent before or after a models’ delete() method or queryset delete() method is called |
| django.db.models.signals.m2m\_changed | Sent when a ManyToManyField is changed |
| django.core.signals.request\_started & django.core.signals.request\_finished | Sent when an HTTP request is started or finished |

**Explain the caching strategies in the Django?**

Caching refers to the technique of storing the output results when they are processed initially so that next time when the same results are fetched again, instead of processing again those already stored results can be used, which leads to faster accessing as well us less resource utilization. Django provides us with a robust cache system that is able to store dynamic web pages so that these pages don’t need to be evaluated again for each request.   
  
**Some of the caching strategies in Django are listed below:**

| **Strategy** | **Description** |
| --- | --- |
| Memcached | A memory-based cache server is the fastest and most efficient |
| FileSystem Caching | Values of the cache are stored as separate files in a serialized order |
| Local-memory Caching | This is used as the default cache strategy by Django if you haven’t set anything. It is per-process as well as thread-safe. |
| Database Caching | Cache data will be stored in the database and works very well if you have a fast and well-indexed DB server. |

**Explain user authentication in Django?**

Django comes with a built-in user authentication system, which handles objects like users, groups, user-permissions, and few cookie-based user sessions. Django User authentication not only authenticates the user but also authorizes him.  
  
**The system consists and operates on these objects:**

* Users
* Permissions
* Groups
* Password Hashing System
* Forms Validation
* A pluggable backend system

**Explain Django Response lifecycle?**

Whenever a request is made to a web page, Django creates an HttpRequest object that contains metadata about the request. After that Django loads the particular view, passing the HttpRequest as the first argument to the view function. Each view will be returning an HttpResponse object.  
On the big picture following steps occur when a request is received by Django:

1. First of the Django settings.py file is loaded which also contain various middleware classes (MIDDLEWARES)
2. The middlewares are also executed in the order in which they are mentioned in the MIDDLEWAREST
3. From here on the request is now moved to the URL Router, who simply gets the URL path from the request and tries to map with our given URL paths in the urls.py.
4. As soon as it has mapped, it will call the equivalent view function, from where an equivalent response is generated
5. The response also passes through the response middlewares and send back to the client/browser.

### What's the use of a session framework?

Using the session framework, you can easily store and retrieve arbitrary data based on the pre-site-visitors. It stores data on the server-side and takes care of the process of sending and receiving cookies. These cookies just consist of a session ID, not the actual data itself unless you explicitly use a cookie-based backend.

### What’s the use of Middleware in Django?

Middleware is something that executes between the request and response. In simple words, you can say it acts as a bridge between the request and response. Similarly In Django when a request is made it moves through middlewares to views and data is passed through middleware as a response.

### What is context in the Django?

Context is a dictionary mapping template variable name given to Python objects in Django. This is the general name, but you can give any other name of your choice if you want.

### What is django.shortcuts.render function?

When a view function returns a webpage as HttpResponse instead of a simple string, we use render(). Render function is a shortcut function that lets the developer easily pass the data dictionary with the template. This function then combines the template with a data dictionary via templating engine. Finally, this render() returns as HttpResponse with the rendered text, which is the data returned by models. Thus, Django render() bypasses most of the developer’s work and lets him use different template engines.  
The basic syntax:  
render(request, template\_name, context=None, content\_type=None, status=None, using=None)  
The request is the parameter that generates the response. The template name is the HTML template used, whereas the context is a dict of the data passed on the page from the python. You can also specify the content type, the status of the data you passed, and the render you are returning.

### What’s the significance of the settings.py file?

As the name suggests this file stores the configurations or settings of our Django project, like database configuration, backend engines, middlewares, installed applications, main URL configurations, static file addresses, templating engines, main URL configurations, security keys, allowed hosts, and much more.

**How to view all items in the Model?**

ModelName.objects.all()

**How to filter items in the Model?**

ModelName.objects.filter(field\_name=”term”)

### How to use file-based sessions?

To use the same, you need to set the SESSION\_ENGINE settings to "django.contrib.sessions.backends.file"

### What is mixin?

Mixin is a type of multiple inheritances wherein you can combine behaviors and attributes of more than one parent class. It provides us with an excellent way to reuse code from multiple classes. One drawback of using these mixins is that it becomes difficult to analyze what a class is doing and which methods to override in case of its code being too scattered between multiple classes.

### What is Django Field Class?

'Field' refers to an abstract class that represents a column in the database table.   
The Field class is just a subclass of RegisterLookupMixin. In Django, these fields are used to create database tables (db\_types()) which are used to map Python types to the database using get\_prep\_value() and the other way round using from\_db\_value() method. Therefore, fields are fundamental pieces in different Django APIs such as models and querysets.

### Why is permanent redirection not a good option?

Permanent redirection is used only when you don’t want to lead visitors to the old URLs. The response of the permanent redirections is cached by the browser so when you try to redirect to something else it will cause issues. Since this is a browser-side operation if your user wants to move to a new page it will load the same page.

### Difference between Django OneToOneField and ForeignKey Field?

Both of them are of the most common types of fields used in Django. The only difference between these two is that ForeignKey field consists of on\_delete option along with a model’s class because it’s used for many-to-one relationships while on the other hand, the OneToOneField, only carries out a one-to-one relationship and requires only the model’s class.

### How can you combine multiple QuerySets in a View?

Initially, Concatenating QuerySets into lists is believed to be the easiest approach. Here’s an example of how to do that:  
from itertools import chain  
result\_list = list(chain(model1\_list, model2\_list, model3\_list))

### How to get a particular item in the Model?

ModelName.objects.get(id=”term”)  
Note: If there are no results that match the query, get() will raise a **DoesNotExist** exception. If more than one item matches the given get() query. In this case, it’ll raise **MultipleObjectsReturned**, which is also an attribute of the model class itself.

### How to obtain the SQL query from the queryset?

print(queryset.query)

### What are the ways to customize the functionality of the Django admin interface?

There are multiple ways to customize the functionality of the Django admin interface. You can piggyback on top of an add/change form that’s automatically generated by Django, you can add JavaScript modules using the js parameter. This parameter is basically a list of URLs that point to the JavaScript modules that are to be included in your project within a <script> tag. You can also write views for the admin if you want.

### Difference between select\_related and prefetch\_related?

Though both the functions are used to fetch the related fields on a model but their functioning is bit different from each other. In simple words, select\_related uses a foreign key relationship, i.e. using join on the query itself while on the prefetch\_related there is a separate lookup and the joining on the python side. Let’s try to illustrate this via an example:

**from** django.db **import** models

**class** **Country**(models.Model):

country\_name = models.CharField(max\_length=5)

**class** **State**(models.Model):

state\_name = models.CharField(max\_length=5)

country = model.ForeignKey(Country)

>> states = State.objects.select\_related(‘country’).all()

>> **for** state **in** states:

… print(state.state\_name)

```Query Executed

SELECT state\_id, state\_name, country\_name FROM State INNER JOIN Country ON (State.country\_id = Country.id)

```

>> country = Country.objects.prefetch\_related(‘state’).get(id=1)

>> **for** state **in** country.state.all():

… print(state.state\_name)

```Query Executed

SELECT id, country\_name FROM country WHERE id=1;

SELECT state\_id, state\_name WHERE State WHERE country\_id IN (1);

```

### Explain Q objects in Django ORM?

Q objects are used to write complex queries, as in filter() functions just `AND` the conditions while if you want to `OR` the conditions you can use Q objects. Let’s see an example:

**from** django.db **import** models

**from** django.db.models **import** Q

>> objects = Models.objects.get(

Q(tag\_\_startswith='Human'),

Q(category=’Eyes’) | Q(category=’Nose’)

)

```Query Executed

SELECT \* FROM Model WHERE tag LIKE ‘Human%’ AND (category=’Eyes’ OR category=’Nose’)

```

### What are Django exceptions?

In addition to the standard Python exceptions, Django raises of its own exceptions.List of the exceptions by Django (https://docs.djangoproject.com/en/3.1/ref/exceptions/)