PROGRAM CODE CLUB

DJANGO INTERVIEW QUESTION

Introduction of Django:

Django (named after the Django Reinhardt) is a high-level python-based free and open-source web framework that follows the **model-view-template(MVT)** architectural pattern. As of now, the framework is maintained by **Django** Software Foundation (DSF), an independent organization based in the US and established as a 501(c)(3) non-benefit.

It was created in the fall of 2003 when the web programmers at the Lawrence Journal-World newspaper, Adrian Holovaty and Simon Willson. Began using python to build applications. Two years later, in July of 2005, it was released to the public under a BSD license. Later, in June 2008, the fledgling Django Software Foundation(DSF) took over the development of Django.

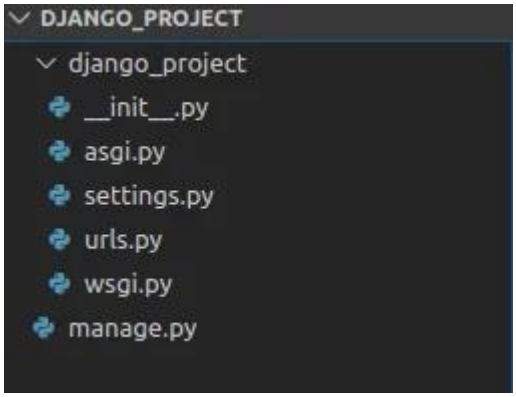
Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as **DRY (Don't Repeat Yourself)** and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

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.

File Structure of Django:



__init.py_

This is an empty file as you can see below in the image. The function of this file is to tell the Python interpreter that this directory is a package and involvement of this __init.py_ file in it makes it a python project.

settings.py

It contains the Django project configuration. The setting.py is the most important file, and it is used for adding all the applications and middleware applications. This is the main setting file of the Django project. This contains several variable names, and if you change the value, your application will work accordingly. It contains sqlite3 as the default database. We can change this database to MySQL, PostgreSQL, or MongoDB according to the web application we create. It contains some pre-installed apps and middleware that are there to provide basic functionality.

urls.py

URL is a universal resource locator; it contains all the endpoints that we should have for our website. It is used to provide you the address of the resources (images, webpages, websites, etc) that are present out there on the internet.

In simpler words, this file tells Django that if a user comes with this URL, direct them to that website or image whatsoever it is.

wsgi.py

When you will complete your journey from development to production, the next task is hosting your application. Here you will not be using the Django web server, but the WSGI server will take care of it. WSGI stands for Web Server Gateway Interface, it describes the way how servers interact with the applications. It is a very easy task; you just have to import

middleware according to the server you want to use. For every server, there is Django middleware available that solves all the integration and connectivity issues.

asgi.py

ASGI works similar to WSGI but comes with some additional functionality. ASGI stands for Asynchronous Server Gateway Interface. It is now replacing its predecessor WSGI.

Manage.py

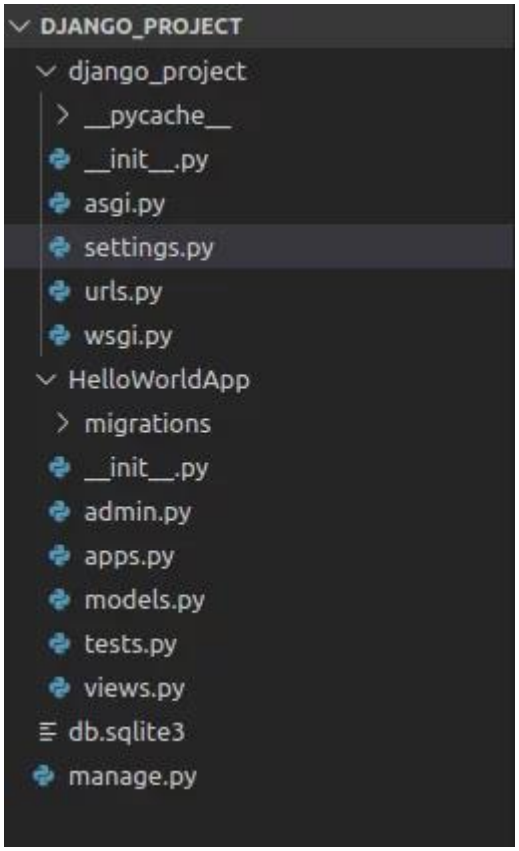
This file is used basically as a command-line utility and for deploying, debugging, or running our web application.

This file contains code for **runserver, or makemigrations or migrations**, etc. that we use in the shell. **Anyway, we do not need to make any changes to the file.**

- **Runserver:** This command is used to run the server for our web application.
- **Migration:** This is used for applying the changes done to our models into the database. That is if we make any changes to our database then we use **migrate** command. This is used the first time we create a database.
- **Makemigration:** this is done to apply new migrations that have been carried out due to the changes in the database.

APPs

Apart from the above file, our project contains all the app directories. Now we will look into the Django app structure in detail



admin.py

As the name suggests, this file is used for registering the models into the Django administration. The models that are present have a superuser/admin who can control the information that is being stored. This admin interface is pre-built and we don't need to create it.

apps.py

This file deals with the application configuration of the apps. The default configuration is sufficient in most of the cases and hence we won't be doing anything here in the beginning.

models.py

This file contains the models of our web applications (usually as classes). Models are basically the blueprints of the database we are using and hence contain the information regarding attributes and the fields etc of the database.

views.py

This file is a crucial one, it contains all the Views (usually as classes). Views.py can be considered as a file that interacts with the client. Views are a user interface for what we see

when we render a Django Web application. We are going to make different types of Views using the concept of serializers in the Django Rest Framework in the further sections.

urls.py

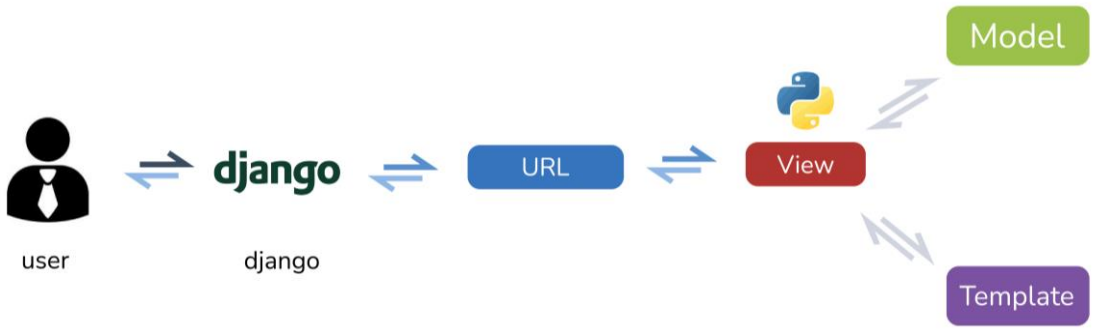
Just like the project urls.py file, this file handles all the URLs of our web application. This file is just to link the Views in the app with the host web URL. The settings urls.py has the endpoints corresponding to the Views.

tests.py

This file contains the code that contains different test cases for the application. It is used to test the working of the application. We won't be working on this file in the beginning and hence it is going to be empty as of now.

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 templates in Django or Django template language?

Templates are an integral part of the Django MVT architecture. They generally comprise HTML, CSS, and js in which dynamic variables and information are embedded with the help of views. Some constructs are recognized and interpreted by the template engine. The main ones are variables and tags.

A template is rendered with a context. Rendering just replaces variables with their values, present in the context, and processes tags. Everything else remains as it is.

The syntax of the Django template language includes the following four constructs :

- Variables
- Tags
- Filters
- Comments

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.

Example:

```
from django.http import HttpResponse
def sample_function(request):
 return HttpResponse("Welcome to Django")
```

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 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 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

What is the use of Middlewares in Django?

Middleware's in Django is a lightweight plugin that processes during request and response execution. It performs functions like security, CSRF protection, session, authentication, etc. Django supports various built-in middleware.

What are Django Exceptions?

An exception is an abnormal event that leads to program failure. Django uses its exception classes and python exceptions as well to deal with such situations.

We define Django core exceptions in "Django.core.exceptions". The following classes are present in this module:

Exception	Description
AppRegistryNotReady	This class raises for using models before loading the app process.
ObjectDoesNotExist	It's a base class for DoesNotExist exceptions.
EmptyResultSet	This exception arises when the query fails to return results.
FieldDoesNotExist	When the requested file does not exist, this exception arises.
MultipleObjectsReturned	It raises by the query multiple objects returned when we expect only one object.
SuspiciousOperation	It raises when the user has performed some operation, which is considered suspicious from a security perspective.
PermissionDenied	It arises when a user does not have permission to execute a specific action requested.
ViewDoesNotExist	When the requested view does not exist, this exception raises.
MiddlewareNotUsed	When there is no middleware in server configuration, this exception arises.
ImproperlyConfigured	When Django configuration is improper, this exception arises.
FieldError	When there is a problem with the model field, this exception arises.
ValidationError	It raises when data validation fails.

Explain Django session

Django uses the session to keep track of the state between the site and a particular browser. Django supports anonymous sessions. The session framework stores and retrieves data on a per-site-visitor basis. It stores the information on the server side and supports sending and receiving cookies. Cookies store the data of session ID but not the actual data itself.

What are Django cookies?

A cookie is a piece of information stored in the client's browser. To set and fetch cookies, Django provides built-in methods. We use the set_cookie() method for setting a cookie and the get() method for getting the cookie.

You can also use the request.COOKIES['key'] array to get cookie values.

Describe Django ORM.

In Django, the most notable feature is Object-Relational Mapper (ORM), which allows you to interact with app data from various relational databases such as SQLite, MySQL, and PostgreSQL.

Django ORM is the abstraction between web application data structure (models) and the database where the data is stored. Without writing any code, you can retrieve, delete, save, and perform other operations over the database.

The main advantage of ORMs is rapid development. ORMs make projects more portable. It's easier to change the database with Django ORM.

When to use iterators in Django ORM?

Iterators are containers in Python containing several elements. Every object in the iterator implements two methods that are __init__() and the __next__() methods.

In Django, the fair use of an iterator is when you process results that take up a large amount of memory space. For this, you can use the iterator() method, which evaluates the QuerySet and returns the corresponding iterator over the results.

What is the "django.shortcuts.render" function?

When a View function returns a web page as HttpResponse instead of a simple string, we use the render function.

Render is a shortcut for passing a data dictionary with a template. This function uses a templating engine to combine templates with a data dictionary.

Finally, the render() returns the HttpResponse with the rendered text, the models' data.

Syntax:

```
render(request, template_name, context=None, content_type=None, status=None,
using=None)
```

The request generates a response. The template name and other parameters pass the dictionary. For more control, specify the content type, the data status you passed, and the render you are returning.

Explain Django Security.

Protecting user's data is an essential part of any website design. Django implements various sufficient protections against several common threats. The following are Django's security features:

- Cross-site scripting (XSS) protection
- SQL injection protection
- Cross-site request forgery (CSRF) protection
- Enforcing SSL/HTTPS
- Session security
- Clickjacking protection
- Host header validation

What is a Django serializers?

Serializers permit complex information, for example, querysets and model cases to be changed over completely to local Python data types that can then be effortlessly delivered into JSON, XML, or other substance types. Serializers likewise give deserialization, permitting parsed information to be changed over once more into complex sorts, after first approving the approaching information.

What is an API?

An API is a set of definitions and protocols for building and integrating application software. API stands for Application Programming Interface. APIs let your product or service communicate with other products and services without having to know how they're implemented. This can simplify app development, saving time and money. The API is not the database or even the server; it's the code that governs the access point(s) for the server. It's sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).

What is a web API?

A web API is a collection of endpoints that expose certain parts of an underlying database. As developers, we control the URLs for each endpoint, what underlying data is available, and what actions are possible via HTTP verbs.

What is a REST API?

A **REST(Representational State Transfer) API (also known as RESTful API)** is an API that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. Every RESTful API: is stateless, like HTTP supports common HTTP verbs (GET, POST, PUT, DELETE, etc.) returns data in either the JSON or XML format.
Any RESTful API must, at a minimum, have these three principles. The standard is important because it provides a consistent way to both design and consumes web APIs.

What is an endpoint?

A web API has endpoints - URLs with a list of available actions (HTTP verbs) that expose data (typically in JSON, which is the most common data format these days and the default for Django REST Framework).

The type of endpoint which returns multiple data resources is known as a collection.

What are HTTP Verbs?

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs.

HTTP Verb	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

What is the difference between HTTP and HTTPS?

HTTPS stands for Hypertext Transfer Protocol Secure (also referred to as HTTP over TLS or HTTP over SSL). HTTPS also uses TCP (Transmission Control Protocol) to send and receive data packets, but it does so over port 443, within a connection encrypted by Transport Layer Security (TLS). Generally, sites running over HTTPS will have a redirect in place so even if you type in http:// it will redirect to deliver over a secured connection.

Key Differences:

- HTTP is unsecured while HTTPS is secured.
- HTTP sends data over port 80 while HTTPS uses port 443.
- HTTP operates at application layer, while HTTPS operates at the transport layer.
- No SSL certificates are required for HTTP, with HTTPS it is required that you
- have an SSL certificate and it is signed by a CA.
- HTTP doesn't require domain validation, whereas HTTPS requires at least
- domain validation and certain certificates even require legal document validation.
- No encryption in HTTP, with HTTPS the data is encrypted before sending.

What are status codes?

HTTP response status codes indicate whether a specific HTTP request has been completed. Responses are grouped into five classes:

- 1xx: Informational – Communicates transfer protocol-level information.
- 2xx: Success – Indicates that the client's request was accepted successfully.
- 3xx: Redirection – Indicates that the client must take some additional action to complete their request.
- 4xx: Client Error – This category of error status codes points the finger at clients.
- 5xx: Server Error – The server takes responsibility for these error status codes.

What is the difference between authentication and authorization?

Authentication confirms that users are who they say they are. Authorization gives those users permission to access a resource. Insecure environments, authorization must always follow authentication. Users should first prove that their identities are genuine before an organization's administrators grant them access to the requested resources.

Let's use an analogy to outline the differences.

Consider a person walking up to a locked door to provide care to a pet while the family is away on vacation. That person needs: Authentication, in the form of a key. The lock on the door only grants access to someone with the correct key in much the same way that a system only grants access to users who have the correct credentials. Authorization, in the form of permissions. Once inside, the person has the authorization to access the kitchen and open the cupboard that holds the pet food. The person may not have permission to go into the bedroom for a quick nap.

What is a browsable API?

Django REST Framework supports generating human-friendly HTML output for each resource when the HTML format is requested. These pages allow for easy browsing of resources, as well as forms for submitting data to the resources using POST, PUT, and DELETE. It facilitates interaction with RESTful web services through any web browser. To enable this feature, we should specify text/html for the Content-Type key in the request

header.

What is CORS?

Cross-Origin Resource Sharing (CORS) is a protocol that enables scripts running on a browser client to interact with resources from a different origin. This is useful because, thanks to the same-origin policy followed by XMLHttpRequest and fetch, JavaScript can only make calls to URLs that live on the same origin as the location where the script is running.

How to fix CORS error in Django?

CORS requires the server to include specific HTTP headers that allow for the client to determine if and when cross-domain requests should be allowed. The easiest way to handle this–-and the one recommended by Django REST Framework–-is to use middleware that will automatically include the appropriate

HTTP headers based on our settings.

We use django-cors-headers :

- add corsheaders to the INSTALLED_APPS
- add CorsMiddleware above CommonMiddleWare in MIDDLEWARE
- create a CORS_ORIGIN_WHITELIST

What is the difference between stateful and stateless?

Stateful applications and processes, however, are those that can be returned to again and again, like online banking or email. They're performed with the context of previous transactions and the current transaction may be affected by what happened during previous transactions. For these reasons, stateful apps use the same servers each time they process a request from a user.

If a stateful transaction is interrupted, the context and history have been stored so you can pick up where you left off. Stateful apps track things like window location, setting preferences, and recent activity. You can think of stateful transactions as an ongoing periodic conversation with the same person.

In terms of authorization,

- **Stateful** = save authorization info on the server-side, this is the traditional way
- **Stateless** = save authorization info on the client-side, along with a signature to ensure integrity, in form of tokens

What is Django Rest Framework?

Django REST Framework is a web framework built over Django that helps to create web APIs which are a collection of URL endpoints containing available HTTP verbs that return JSON. It's very easy to build model-backed APIs that have authentication policies and are browsable.

What are the benefits of using the Django Rest Framework?

Its Web-browsable API is a huge usability win for your developers. Authentication policies include packages for **OAuth1** and **OAuth2**.

Serialization supports both **ORM** and **non-ORM** data sources. It's customizable all the way down. Just use regular function-based views if you don't need the more powerful features.

It has extensive documentation and great community support. It's used and trusted by internationally recognized companies including Mozilla, Red Hat, Heroku, and Eventbrite.

What are serializers?

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML, or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

What are Permissions in DRF?

Permission checks are always run at the very start of the view before any other code is allowed to proceed. Permission checks will typically use the authentication information in the **request.user** and **request.auth** properties to determine if the incoming request should be permitted. Permissions are used to grant or deny access for different classes of users to different parts of the API.

The simplest style of permission would be to allow access to any authenticated user and deny access to any unauthenticated user. This corresponds to the **IsAuthenticated** class in the REST framework.

These can be applied at a **project-level**, a **view-level**, or any **individual model**

level.

How to add login in the browsable API provided by DRF?

Within the project-level urls.py file, add a new URL route that includes

rest_framework.urls .

```
# blog_project/urls.py
from django.urls import include path

urlpatterns = [
...
path('api-auth/', include('rest_framework.urls')), # new
]
```

What are Project-Level Permissions?

Django REST Framework ships with several built-in project-level permissions settings we can use, including:

- **AllowAny** - any user, authenticated or not, has full access
- **IsAuthenticated** - only authenticated, registered users have access
- **IsAdminUser** - only admins/superusers have access
- **IsAuthenticatedOrReadOnly** - unauthorized users can view any page, but only authenticated users have write, edit, or delete privileges

Implementing any of these four settings requires updating the

DEFAULT_PERMISSION_CLASSES setting:

```
# blog_project/settings.py

REST_FRAMEWORK = {
  'DEFAULT_PERMISSION_CLASSES': [
      'rest_framework.permissions.IsAuthenticated', # new
  ]
}
```

How to make custom permission classes?

To make a custom permission class, create a file named **permissions.py** that imports

permissions at the top and then create your custom class, for example **IsAuthorOrReadOnly** which extends **BasePermission** , then we override **has_object_permission .**

What is Basic Authentication?

The most common form of HTTP authentication is known as "Basic" Authentication. When a client makes an HTTP request, it is forced to send an approved authentication credential before access is granted.

The complete request/response flow looks like this:

- The client makes an HTTP request
- The server responds with an HTTP response containing a 401 (Unauthorized) status code and WWW-Authenticate HTTP header with details on how to authorize
- The client sends credentials back via the Authorization HTTP header
- The server checks credentials and responds with either 200 OK or 403 Forbidden status codes. Once approved, the client sends all future requests with the Authorization HTTP header credentials.

Note: The authorization credentials sent are the unencrypted base64 encoded version of <username>:<password> .

What are the disadvantages of Basic Authentication?

Cons of Basic Authentication:

- On every single request, the server must look up and verify the username and password, which is inefficient.
- User credentials are being passed in clear text—not encrypted at all, can be easily captured and reused.

What is session authentication?

At a high level, the client authenticates with its credentials (username/password) and then receives a session ID from the server which is stored as a cookie). This session ID is then passed in the header of every future HTTP request.

When the session ID is passed, the server uses it to look up a session object containing all available information for a given user, including credentials. This approach is stateful because a record must be kept and maintained on both the server (the session object) and the client (the session ID).

Let's review the basic flow:

- A user enters their login credentials (typically username/password)
- The server verifies the credentials are correct and generates a session object that is then stored in the database

- The server sends the client a session ID — not the session object itself—which is stored as a cookie on the browser
- On all future requests, the session ID is included as an HTTP header and if verified by the database, the request proceeds
- Once a user logs out of an application, the session ID is destroyed by both the client and server
- If the user later logs in again, a new session ID is generated and stored as a cookie on the client

Note: The default setting in Django REST Framework is actually a combination of Basic Authentication and Session Authentication. Django's traditional session-based authentication system is used and the session ID is passed in the HTTP header on each request via Basic Authentication.

What are the pros and cons of session authentication?

Pros:

- User credentials are only sent once, not on every request/response cycle as in Basic Authentication.
- It is also more efficient since the server does not have to verify the user's credentials each time, it just matches the session ID to the session object which is a fast look-up.

Cons:

- A session ID is only valid within the browser where log-in was performed; it will not work across multiple domains. This is an obvious problem when an API needs to support multiple front-ends such as a website and a mobile app.
- The session object must be kept up-to-date which can be challenging in large sites with multiple servers.
- The cookie is sent out for every single request, even those that don't require authentication, which is inefficient.

Note: It is generally not advised to use a session-based authentication scheme for any API that will have multiple front-ends.

What is Token Authentication?

Tokens are pieces of data that carry just enough information to facilitate the process of determining a user's identity or authorizing a user to perform an action. All in all, tokens are

artifacts that allow application systems to perform the authorization and authentication process.

Token-based authentication is stateless: once a client sends the initial user credentials to the server, a unique token is generated and then stored by the client as either a cookie or in local storage. This token is then passed in the header of each incoming HTTP request and the server uses it to verify that a user is authenticated. The server itself does not keep a record of the user, just whether a token is valid or not.

What are the pros and cons of token authentication?

Pros:

- Since tokens are stored on the client, scaling the servers to maintain up-to-date session objects is no longer an issue.
- Tokens can be shared amongst multiple front-ends: the same token can represent a user on the website and the same user on a mobile app.

Cons:

- A token contains all user information, not just an id as with a session-id/sessionobject set up.
- Since the token is sent on every request, managing its size can become a performance issue.

What is the difference between cookies vs localStorage?

- **Cookies** are used for reading server-side information.They are smaller (4KB) in size and automatically sent with each HTTP request.
- **LocalStorage** is designed for client-side information.It is much larger (5120KB) and its contents are not sent by default with each HTTP request.

Where should the token be saved - cookie or localStorage?

With token-based auth, you are given the choice of where to store the JWT. Commonly, the JWT is placed in the browser's local storage and this works well formost use cases. There are some issues with storing JWTs in local storage to be aware of. Unlike cookies, local storage is sandboxed to a specific domain and itsdata cannot be accessed by any other domain including sub-domains. You can store the token in a cookie instead, but the max size of a cookie is only 4kb so that may be problematic if you have many claims attached to the token.

Additionally, you can store the token in session storage which is similar to local storage but is cleared as soon as the user closes the browser.

Tokens stored in both cookies and localStorage are vulnerable to XSS attacks. The current best practice is to store tokens in a cookie with the httpOnly and Secure cookie flags.

What are the disadvantages of Django REST Framework's built-in TokenAuthentication?

- It doesn't support setting tokens to expire
- It only generates one token per user

What are JSON Web Tokens(JWTs)?

JSON web token (JWT), is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Because of its relatively small size, a JWT can be sent through a URL, through a POST parameter, or inside an HTTP header, and it is transmitted quickly. A JWT contains all the required information about an entity to avoid querying a database more than once. The recipient of a JWT also does not need to call a server to validate the token.

The benefits of JWT?

- **More compact**: JSON is less verbose than XML, so when it is encoded, a JWT is smaller than a simple token. This makes JWT a good choice to be passed in HTML and HTTP environments.
- **More secure**: JWTs can use a public/private key pair for signing. A JWT can also be symmetrically signed by a shared secret using the HMAC algorithm.
- **More common**: JSON parsers are common in most programming languages because they map directly to objects.
- **Easier to process:** JWT is used at the internet scale. This means that it is easier to process on user's devices, especially mobile.

What is the difference between a session and a cookie?

A cookie is a bit of data stored by the browser and sent to the server with every request. Cookies are used to identify sessions. A session is a collection of data stored on the server and associated with a given user (usually via a cookie containing an id code).

The main difference between a session and a cookie is that session data is stored on the server, whereas cookies store data in the visitor's browser. Sessions are more secure than cookies as it is stored in the server. A cookie can be turned off from the browser. Data stored in cookies can be stored for months or years, depending on the life span of the cookie. But the data in the session is lost when the web browser is closed.

What is the difference between cookies and tokens?

Cookie-based authentication is stateful. This means that an authentication record or session must be kept both server and client-side. The server needs to keep track of active sessions in a database, while on the front-end a cookie is created that holds a session identifier, thus the name cookie-based authentication.

Token-based authentication is stateless. The server does not keep a record of which users are logged in or which JWTs have been issued. Instead, every request to the server is accompanied by a token which the server uses to verify the authenticity of the request. The token is generally sent as an addition Authorization header in the form of Bearer {JWT}, but can additionally be sent in the body of a POST request or even as a query parameter.

What's an access token?

When a user login in, the authorization server issues an access token, which is an artifact that client applications can use to make secure calls to an API server. When a client application needs to access protected resources on a server on behalf of a user, the access token lets the client signal to the server that it has received authorization by the user to perform certain tasks or access certain resources.

What is meant by a bearer token?

A bearer token stands for a token that can be used by those who hold it. The access token thus acts as a credential artifact to access protected resources rather than an identification artifact.

What is the security threat to access tokens?

Malicious users could theoretically compromise a system and steal access tokens, which in turn they could use to access protected resources by presenting those tokens directly to the server. As such, it's critical to have security strategies that minimize the risk of compromising

access tokens. One mitigation method is to create access tokens that have a short lifespan: they are only valid for a short time defined in terms of hours or days

.

What is a refresh token?

For security purposes, access tokens may be valid for a short amount of time. Once they expire, client applications can use a refresh token to "refresh" the access token. That is, a refresh token is a credential artifact that lets a client application get new access tokens without having to ask the user to log in again. The client application can get a new access token as long as the refresh token is valid and unexpired. Consequently, a refresh token that has a very long lifespan could theoretically give infinite power to the token bearer to get a new access token to access protected resources anytime. The bearer of the refresh token could be a legitimate user or a malicious user.

What are the best practices when using token authentication?

Some basic considerations to keep in mind when using tokens:

- Keep it secret. Keep it safe.
- Do not add sensitive data to the payload.
- Give tokens an expiration.
- Embrace HTTPS.
- Consider all of your authorization use cases.
- Store and reuse.

What is cookie-based authentication?

A request to the server is always signed in by an authorization cookie.

Pros:

- Cookies can be marked as "http-only" which makes them impossible to be read on the client-side. This is better for XSS-attack protection.
- Comes out of the box - you don't have to implement any code on the client-side.

Cons:

- Bound to a single domain.
- Vulnerable to XSRF. You have to implement extra measures to make your site protected against cross-site request forgery.
- Are sent out for every single request, (even for requests that don't require authentication).

What are viewsets in DRF?

A viewset is a way to combine the logic for multiple related views into a single class. In other words, one viewset can replace multiple views. It is a class that is simply a type of class-based View, that does not provide any method handlers such as .get() or .post() , and instead provides actions such as .list() and .create() .

What are routers in DRF?

Routers work directly with viewsets to automatically generate URL patterns for us. Django REST Framework has two default routers: **SimpleRouter** and **DefaultRouter**.

- **SimpleRouter** - This router includes routes for the standard set of list, create, retrieve, update, partial_update, and destroy actions.
- **Default Router** - This router is similar to SimpleRouter, but additionally includes a default API root view, that returns a response containing hyperlinks to all the list views. It also generates routes for optional .json style format suffixes.

What is the difference between APIViews and Viewsets in

DRF has two main systems for handling views:

- **APIView :** This provides methods handler for http verbs: get, post, put, patch, and delete.
- **ViewSet :** This is an abstraction over APIView , which provides actions as methods:
    - **list**: read-only, returns multiple resources (http verb: get). Returns a list of dicts.
    - **retrieve**: read-only, single resource (http verb: get, but will expect an id in the url). Returns a single dict.
    - **create**: creates a new resource (http verb: post)
    - **update/partial_update**: edits a resource (http verbs: put/patch)
    - **destroy**: removes a resource (http verb: delete)

What is the difference between GenericAPIView and GenericViewset ?

GenericAPIView : for APIView, this gives you shortcuts that map closely to your database models. Adds commonly required behavior for standard list and detail views. Gives you some attributes like the serializer_class , also gives pagination_class , filter_backend , etc

GenericViewSet : There are many GenericViewSet, the most common being ModelViewSet . They inherit from GenericAPIView and have a full implementation of all of the actions: list, retrieve, destroy, updated, etc.