Table of Contents

Core Concepts

1. 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). For example, the API design for a weather service could specify that the user supply a zip code and that the producer reply with a 2-part answer, the first being the high temperature, and the second being the low.

⬆ Back to Top

2. 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.

⬆ Back to Top

3. 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 consume web APIs.

⬆ Back to Top

4. 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.**

⬆ Back to Top

5. 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 Request Method	Description
GET	Used to fetch data and shouldn't be used to modify data. Eg: Retrieve list of facebook friends
POST	Used to create/upload data to the server and results in change of state of the server. Eg: Create a new post in facebook
PUT	Used to completely update an already existing data. Eg: Update your profile page
DELETE	Used to delete existing data in the server. Eg: Delete one of your facebook posts
PATCH	Used to partially update an already existing date. Eg: Update your relationship status only in the profile page
OPTIONS	Used to check the HTTP methods a server allows
CONNECT	Used to open a tunnel to a server Eg: When the request is to be sent over a proxy server
HEAD	Works similar to GET request but doesn't retrieve data but only the response status line and the response headers. Eg: To check the size of a file to be downloaded using the **Content-length** header
TRACE	Returns the request sent and is used for troubleshooting purpose

⬆ Back to Top

6. 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 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, where as 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.

⬆ Back to Top

7. What are status codes?

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in 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 in order 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.

⬆ Back to Top

8. 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. In secure 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.

⬆ Back to Top

9. 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 service through any web browser. To enable this feature, we should specify text/html for the Content-Type key in the request header.

⬆ Back to Top

10. 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.

⬆ Back to Top

11. 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`

⬆ Back to Top

12. What is the difference between stateful and stateless?

A stateless process or application can be understood in isolation. There is no stored knowledge of or reference to past transactions. Each transaction is made as if from scratch for the first time.

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

⬆ Back to Top

Django Rest Framework

1. 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.

⬆ Back to Top

2. What are benefits of using 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.

⬆ Back to Top

3. 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.

⬆ Back to Top

4. 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 REST framework.

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

⬆ Back to Top

5. 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
]
```

⬆ Back to Top

6. What are Project-Level Permissions?

Django REST Framework ships with a number of 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
 ]
}
```

⬆ Back to Top

7. How to make custom permission classes?

To make 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`.

⬆ Back to Top

8. 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:

i. Client makes an HTTP request
ii. Server responds with an HTTP response containing a `401 (Unauthorized)` status code and `WWW-Authenticate` HTTP header with details on how to authorize
iii. Client sends credentials back via the `Authorization` HTTP header
iv. Server checks credentials and responds with either `200 OK` or `403 Forbidden` status code. 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>`.

⬆ Back to Top

9. 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.

⬆ Back to Top

10. 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:

i. A user enters their log in credentials (typically username/password)
ii. The server verifies the credentials are correct and generates a session object that is then stored in the database
iii. The server sends the client a session ID — not the session object itself—which is stored as a cookie on the browser
iv. On all future requests the session ID is included as an HTTP header and, if verified by the database, the request proceeds
v. Once a user logs out of an application, the session ID is destroyed by both the client and server
vi. 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.

⬆ Back to Top

11. 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.

⬆ Back to Top

12. 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.

⬆ Back to Top

13. What are 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/session object set up. Since the token is sent on every request, managing its size can become a performance issue.

⬆ Back to Top

14. 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.

⬆ Back to Top

15. Where should 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 for most 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 its data 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.

⬆ Back to Top

16. ## What are disadvantages of Django REST Framework's built-in TokenAuthentication?

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

⬆ Back to Top

17. ## 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.

⬆ Back to Top

18. ## What are 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 internet scale. This means that it is easier to process on user's devices, especially mobile.

⬆ Back to Top

19. ## What is the difference between a session and 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 server. Cookie can be turned off from browser. Data stored in cookie 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.

⬆ Back to Top

20. What is the difference between cookie 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.

> Read more here

⬆ Back to Top

21. What's an access token?

When a user logins 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.

⬆ Back to Top

22. 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.

⬆ Back to Top

23. What is the security threat to access token?

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.

⬆ Back to Top

24. 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.

⬆ Back to Top

25. 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.

README MIT license

26. What is cookie-based authentication?

A request to the server is always signed in by 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).

⬆ Back to Top

27. 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()`.

⬆ Back to Top

28. 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.

⬆ Back to Top

29. What is the difference between APIViews and Viewsets in DRF?

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)

⬆ Back to Top

30. 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.

⬆ Back to Top