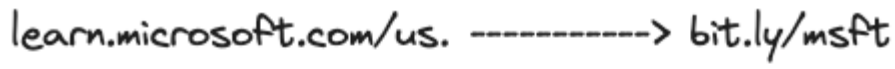
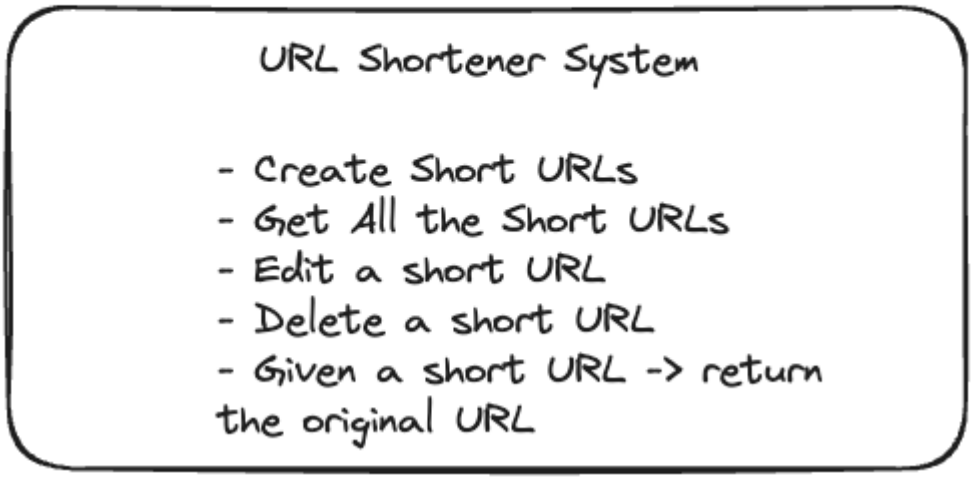
API Designing

What is an API ?





Let's assume we have a URL shortener service, in which we want provide functionalities like creating a short URL, deleting it, editing it etc.

So as an engineer we will be responsible for coding the logic of all these features. And apart from this it will be our responsibility to expose the functionalities of this system so that outer world can consume it. May be some frontend teams in our own organisation or some third party organisation who just need our logics.

And APIs helps us to expose these functionalities.
API stands for Application programming interface. If we have a software and we want to define a contract for this software so that any other system can communicate to this software, then we need to use an API to describe the contract and expose the functionality. Inside the contract you define:

- How to communicate
- What are the input and output expectations
- Any success or error scenarios to take care of

Important points to remember when describing APIs

- Platform Independence

- Service Evolution

Note:

Generally the clients will be consuming your APIs via some network interaction.

Ways to make APIs

The most relevant way now a days to prepare APIs are:

- **Rest API**
- **Graph QL**
- **RPC**
  There are more like SOAP (Simple object access protocol) etc.

REST

Rest stands for Representational State Transfer. REST is a way to prepare APIs. REST is a set of recommendations that provide you architectural style details to designing APIs. REST is independent of any underlying protocols. Mostly people use HTTP/HTTPS with REST but you are free to choose anything else based on your use case.

Recommendations of REST

- REST APIs need to de designed around resources which are any kind of object, data, service or any real life entity.
  - For our URL shortener service, shortUrls and originalUrls can be the resources.
- In REST APIs we need to write the implementation in a way such that every resource has a unique id, that uniquely identifies the data about the resource.
  - If say in an e-commerce we have product resource, then there will some kind of productId. Example: Iphone 11 - ProductId: 33
  - In our URL shortener, every shortUrl and originalUrl will be having an Id associated.
- Now because we will some kind of network interaction in REST APIs, that means we might need to send and receive data. To send and receive this data we use `JSON` as the message format.
  - Example: If we have to fetch all the short and originalUrls from our service then the response payload of the API should be like:

```
{
        shortUrlId: 22,
        originalUrlId: 102,
        shortUrl: "bit.ly/xyz",
        originalUrl: "www.google.com"
}
```

```
{
        orderId:  33,
        userId: 22,
        orderItems: [
                { productId: 1, quantity: 2 },
                { productId: 100, quantity: 5}
        ],
        totalOrderValue: 500,
        orderStatus: "DELIVERED"
}
```

- For network interaction to be done via HTTP on a rest api, we define HTTP methods in a way that they auto describe the APIs. It's not mandatory to follow this and anything else also as it is just recommendation.
  - GET: If a REST Api has GET http method associated then probably you will be fetching some resource
  - POST: If a REST Api has POST http method associated then probably you will be creating some resource
  - DELETE: If a REST Api has DELETE http method associated then probably you will be deleting some resource
  - and more ..
- The URL to be defined for these network interactions are defined using the resource name and the unique identifier.
  Examples:
  For an e-commerce: Resource - Customer, and we want to create, delete, get or update a customer, URLs will be

1. https://www.mykart.com/customers/12 - GET -> This should be fetching the details of customer with ID 12
2. https://www.mykart.com/customers - GET -> This should be fetching the details of all the customers
3. https://www.mykart.com/customers/12 - DELETE -> This should be deleting the details of customer with ID 12

4. https://www.mykart.com/customers - POST -> This should be creating a new customer. Details of the new customer will be sent in a JSON payload inside the request body.
5. https://www.mykart.com/customers/12 - PUT -> This should be updating the details of customer with ID 12. Details to be updated will be sent inside JSON payload in the request body.
6. https://www.mykart.com/customers/12/orders - GET -> This should be fetching the details of all the orders of the customer with ID 12
7. https://www.mykart.com/customers/12/orders/33 - GET -> This should be fetching the details of the order with id 33 of the customer with ID 12

- Resource names mentioned in the URLs are plural. And the hierarchy also should be logical, like an orders exists because of a customer should probably customer comes first.
- To describe an ID in a url (because Id can be a variable value), we generally represent it as:
  - http://www.mykart.com/customers/:customerId/orders/:orderId
  - http://www.mykart.com/customers/{customerId}/orders/{orderId}
- Sometime we might need to send data to filter or sort request output, to do that a lot of times we can use query params in the url.
  - http://www.mykart.com/search?price_start=35000&price_end=75000&category=electronics
  - Everything after question mark is a form of query params, which is key value pairs separated by ampersand.
- If we need to version the APIs, then we also put a version tag in the URLs
  - http://www.mykart.com/v1/customers/{customerId}/orders/{orderId}

RPC

Remote procedure calls. In an RPC, two machine communicate with each other using function calls. The creator of an RPC writes the definition and implementation of the function and consumer calls the function.
Internally these functions are automatically converted to a HTTP request or whatever protocol is used, but we as developer donot need to care about putting url, http method etc in place.

Technologies like gRPC, tRPC etc helps us to make RPCs and consume it.