# Web Development

## Introduction to Web Development

Web development is a multifaceted discipline encompassing the creation, enhancement, and maintenance of websites and web applications. It involves a combination of front-end and back-end technologies, databases, and various tools to bring digital experiences to life.

### Key Components:

- HTML (Hypertext Markup Language):
    - *Description:* HTML forms the structural foundation of web pages, defining the elements and content structure. It includes tags for headings, paragraphs, images, links, and more.
- CSS (Cascading Style Sheets):
    - *Description:* CSS is responsible for styling HTML elements, ensuring a visually appealing and consistent layout. It controls aspects like colors, fonts, spacing, and responsiveness.
- JavaScript and Front-End Frameworks:
    - *Description:* JavaScript is a dynamic scripting language that enables interactive features on the client side. Front-end frameworks like React, Angular, and Vue.js streamline the development of complex user interfaces.
- Back-End Development:
    - *Languages:* Back-end languages like Python, Ruby, PHP, Java, and Node.js handle server-side logic and interact with databases.
    - *Frameworks:* Frameworks such as Django, Ruby on Rails, Laravel, and Express provide structures for efficient server-side development.
- Databases:
    - *Relational Databases:* MySQL, PostgreSQL, and SQLite are used for structured data storage.
    - *NoSQL Databases:* MongoDB, Cassandra, and Firebase handle unstructured or semi-structured data.
- APIs (Application Programming Interfaces):
    - *RESTful APIs:* Follow the REST architecture for communication between the front-end and back-end.
    - *GraphQL:* An alternative to REST, GraphQL optimizes data fetching for clients.
- Version Control:
    - *Git:* A distributed version control system facilitating collaborative development and code versioning.

- Web Development Tools:
    - *Text Editors/IDEs:* Tools like Visual Studio Code, Sublime Text, and Atom provide coding environments.
    - *Browser Developer Tools:* Browsers offer tools for debugging, inspecting elements, and monitoring network activity.
- Web Hosting and Deployment:
    - *Hosting Services:* AWS, Heroku, Netlify, etc., provide infrastructure for hosting web applications.
    - *CI/CD:* Continuous Integration/Continuous Deployment tools automate testing and deployment processes.

## Web API project

- In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building software and applications.
- API is some kind of interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services.
- Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc.
- The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view. It is like a web service or WCF service but the exception is that it only supports HTTP protocol.

### Project Components:

- **Controllers:**
    - Controllers are the heart of the Web API project and are responsible for handling incoming HTTP requests.
    - Each controller class typically represents a resource or a collection of related resources.
    - Actions within controllers respond to specific HTTP methods (GET, POST, PUT, DELETE) and define the behavior for processing requests.
- **Models:**
    - Models represent the data entities used within the Web API project.
    - These can be simple DTOs (Data Transfer Objects) or more complex entities, depending on the requirements.
    - Models define the structure of the data exchanged between the client and the API.
- **Routing:**
    - Routing in ASP.NET Web API maps HTTP requests to the appropriate controller actions.
    - The routing configuration defines the URL patterns and parameter constraints for the API's endpoints.
    - It plays a crucial role in directing requests to the correct controllers and actions.

- **Action Results:**
    - Actions within controllers return action results, which represent the HTTP response to the client.
    - Common action results include OkResult, CreatedAtActionResult, NotFoundResult, etc.
    - These results are responsible for generating the appropriate HTTP status codes and response bodies.
- **Filters:**
    - Filters are attributes or classes that can be applied to controllers or actions to perform logic before or after the execution of the action.
    - Examples include authorization filters, exception filters, and action filters.
- **Middleware:**
    - Middleware components are part of the HTTP request processing pipeline in ASP.NET.
    - They can be used to perform tasks such as authentication, logging, or custom processing before or after the request is handled by the controller.
- **Dependency Injection:**
    - ASP.NET Web API supports dependency injection, allowing for the injection of services or dependencies into controllers.
    - This promotes a more modular and testable design.

## Building Web API

Building a Web API with ASP.NET allows developers to create scalable and flexible HTTP services that can be consumed by various clients. This documentation outlines the essential steps involved in constructing a Web API using ASP.NET.

### Steps to Build a Web API:

- **Create a New ASP.NET Web API Project:**
  - Open Visual Studio and create a new project.
  - Select the "ASP.NET Web Application" template and choose "Web API" as the project type.
- **Define Models:**
  - Create models to represent the data entities used in your API.
  - Models define the structure of the data exchanged between the client and the API.
- **Create Controllers:**
  - Add controllers to your project. Controllers handle incoming HTTP requests and define actions for processing requests.
  - Each controller class typically corresponds to a resource or a collection of related resources.
- **Implement Actions:**
  - Within controllers, implement actions that correspond to different HTTP methods (GET, POST, PUT, DELETE).
  - Actions contain the logic for processing requests and generating responses.
- **Configure Routing:**
  - Configure routing to map incoming requests to the appropriate controllers and actions.
  - Define URL patterns and parameter constraints to structure your API's endpoints.
- **Handle Request and Response Formats:**
  - Use attributes like [FromBody] and [FromUri] to handle request formats (JSON, XML, query parameters).
  - Return appropriate action results to generate HTTP responses.
- **Implement Middleware:**
  - Implement middleware components for additional processing before or after the request is handled by the controller.
  - Examples include authentication middleware and custom logging middleware.

- **Apply Filters:**
    - Apply filters to controllers or actions to perform logic before or after the execution of an action.
    - Filters can include authorization filters, exception filters, and action filters.
- **Test Your API:**
    - Use tools like Postman or curl to test your API endpoints.
    - Ensure that the API behaves as expected and returns the correct responses.
- **Deploy Your API:**
    - Choose a hosting environment (e.g., IIS) and deploy your Web API to make it accessible over the internet.
    - Consider securing your API using appropriate authentication mechanisms.

## Action Method Response (HTTP status code etc.)

### ➔ Common HTTP Status Codes:

- **200 OK**: The request was successful, and the server returns the requested data.

```
{
  "status": "success",
  "data": [Response Data]
}
```

- **201 Created**:The request has been fulfilled, and a new resource has been created.

```
{
  "status": "success",
  "data": [Created Resource Data]
}
```

- **204 No Content**: The request was successful, but there is no additional information to send back (e.g., for a DELETE request).

### ➔ Client Error Status Codes:

- **400 Bad Request:** The request cannot be processed due to client error (e.g., invalid input, missing parameters).

```
{
  "status": "error",
  "message": "Invalid input parameters"
}
```

- **401 Unauthorized**: The request requires authentication, and the provided credentials are missing or invalid.

```
{
  "status": "error",
  "message": "Unauthorized. Please provide valid credentials."
}
```

- **403 Forbidden**: The server understood the request but refuses to authorize it.

```
{
  "status": "error",
  "message": "Forbidden. You do not have permission to access this resource."
}
```

- **404 Not Found:** The requested resource could not be found on the server.

```
{
  "status": "error",
  "message": "Resource not found."
}
```

### ➔ Server Error Status Codes:

- **500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.

```
{
  "status": "error",
  "message": "Internal Server Error. Please try again later."
}
```

### ➔ Additional HTTP Status Codes:

- **202 Accepted:** The request has been accepted but is not yet processed.

- **401 Unauthorized:** Similar to 401, but indicates that the client must authenticate to gain network access.

```
{
  "status": "error",
  "message": "Unauthorized. Please authenticate to access this resource."
}
```

- **422 Unprocessable Entity:** The server understands the content type of the request entity, and the syntax is correct, but it cannot process the contained instructions.

```
{
  "status": "error",
  "message": "Unprocessable Entity. Invalid input data."
}
```

The following table lists all the methods of ApiController class that returns an object of a class that implements IHttpActionResult interface.

| ApiController Method | Description |
|---|---|
| BadRequest() | Creates a BadRequestResult object with status code 400. |
| Conflict() | Creates a ConflictResult object with status code 409. |
| Content() | Creates a NegotiatedContentResult with the specified status code and data. |
| Created() | Creates a CreatedNegotiatedContentResult with status code 201 Created. |
| CreatedAtRoute() | Creates a CreatedAtRouteNegotiatedContentResult with status code 201 created. |
| InternalServerError() | Creates an InternalServerErrorResult with status code 500 Internal server error. |
| NotFound() | Creates a NotFoundResult with status code404. |
| Ok() | Creates an OkResult with status code 200. |
| Redirect() | Creates a RedirectResult with status code 302. |
| RedirectToRoute() | Creates a RedirectToRouteResult with status code 302. |
| ResponseMessage() | Creates a ResponseMessageResult with the specified HttpResponseMessage. |
| StatusCode() | Creates a StatusCodeResult with the specified http status code. |
| Unauthorized() | Creates an UnauthorizedResult with status code 401. |

## Security (CORS, Authentication, Authorization,Exception, JWT token etc.)

Security in ASP.NET Web API is a crucial aspect that involves implementing measures to protect your web API from various potential threats and ensuring that only authorized users or systems can access and interact with your API.

### A. CORS

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to control and restrict web page scripts from making requests to a different domain than the one that served the web page. This is a crucial security measure to prevent potential security vulnerabilities, such as cross-site request forgery (CSRF) and unauthorized data access.

When you are working with a web application developed using .NET, you might need to deal with CORS if your application makes requests to a different domain than the one hosting your web application.

- **Purpose:**

  The primary purpose of CORS is to prevent unauthorized web pages from making requests to a different domain, thereby mitigating potential security threats.

- **Key Concepts:**
  - **Cross-Origin Requests:** A request made by a web page to a domain different from the one hosting the web page.
  - **Same-Origin Policy:** Browsers, by default, enforce the Same-Origin Policy, which restricts web pages from making requests to a different domain.
  - **CORS Headers:** To enable cross-origin requests, servers must include specific HTTP headers in their responses. The headers include:
    - **Access-Control-Allow-Origin:** Specifies which origins are allowed to access the resource.
    - **Access-Control-Allow-Methods:** Specifies the HTTP methods (e.g., GET, POST) allowed when accessing the resource.
    - **Access-Control-Allow-Headers:** Specifies which headers can be used when making the actual request.

- **Access-Control-Allow-Credentials:** Indicates whether the browser should include credentials (such as cookies) when making the request.

## B. Authentication

Authentication is a crucial aspect of web development that ensures that users and entities interacting with an application are who they claim to be. It helps protect sensitive information and resources from unauthorized access. In the context of web development, authentication is commonly used to control access to web pages, APIs, or other resources.

Authentication is a fundamental aspect of security protocols, forming the first line of defense against unauthorized access. It guarantees that users are who they claim to be and plays a crucial role in protecting data, maintaining privacy, and preventing security breaches.

- **Authentication in Web Applications:**
    - In web applications, authentication is often handled through login forms, cookies, and sessions.
    - Technologies like OAuth and OpenID Connect are commonly used for third-party authentication.

- **Authentication in APIs:**
    - For API authentication, methods like API keys, OAuth tokens, or JWTs (JSON Web Tokens) are commonly used.
    - Token-based authentication is prevalent for stateless communication between clients and APIs.

## C. Authorization

Authorization allows a website user to grant and restrict permissions on Web pages, functionality, and data. Authorization checks whether a user is allowed to perform an action or has access to some functionality. For example, having the permission to get data and post data is a part of authorization.

Web API uses authorization filters to implement authorization. The Authorization filters run before the controller action. If the request is not authorized, the filter returns an error response, and the action is not invoked.

Web API provides a built-in authorization filter, Authorize Attribute. This filter checks whether the user is authenticated. If not then it returns the HTTP status code 401 (Unauthorized), without invoking the action.

- **Identity Verification:**
  After the authentication process, where the user's identity is established, authorization comes into play to determine what the authenticated user is allowed to do.
- **Role-Based Access Control (RBAC):**
  RBAC is a widely used authorization model where access permissions are tied to roles, and users are assigned to one or more roles. This simplifies the management of permissions, especially in large systems.

- **Authorization in Web Applications:**
  - In web applications, authorization is typically handled by server-side logic that checks the user's role and permissions before granting access to specific pages or functionalities.
- **Authorization in APIs:**
  - APIs often use token-based authentication combined with scopes or claims to enforce authorization. The API validates the user's token and checks whether the associated permissions allow the requested action.

## D. Exception

Exception are the errors that happens at the run time. If any error thrown by web api, it is translated into Http Response with status code 500 - ‘’Internal server error”.

There are many ways to handle Exception:

- **HttpError:**

  The HttpError class is part of the System.Web.Http namespace and is used to represent HTTP errors. It allows you to create structured error responses with details about the error, such as an error message, error code, and exception details.

- **HttpResponseException:**

  The HttpResponseException class is used to create an HTTP response with a specific status code and content. It allows you to directly return an HTTP response without using the IHttpActionResult interface.

- **Exception Handler:**

  Exception handlers are used to handle unhandled exceptions globally in the Web API application. This can be achieved by implementing an IExceptionHandler and registering it in the HttpConfiguration.

- **Exception Filter:**

  Exception filters are attributes that can be applied to specific action methods or controllers. They allow you to catch and handle exceptions that occur during the processing of a request.

| Filter Type | Interface | Class | Description |
|---|---|---|---|
| Simple Filter | IFilter | - | Defines the methods that are used in a filter |
| Action Filter | IActionFilter | ActionFilterAttribute | Used to add extra logic before or after action methods execute. |
| Authentication Filter | IAuthenticationFilter | - | Used to force users or clients to be authenticated before action methods execute. |
| Authorization Filter | IAuthorizationFilter | AuthorizationFilterAttribute | Used to restrict access to action methods to specific users or groups. |
| Exception Filter | IExceptionFilter | ExceptionFilterAttribute | Used to handle all unhandled exception in Web API. |
| Override Filter | IOverrideFilter | - | Used to customize the behaviour of other filter for individual action method. |

## E. JWT Token

JSON Web Tokens (JWT) is a compact, URL-safe means of representing claims between two parties. It's designed to be a lightweight and self-contained method for securely transmitting information between parties as a JSON object. JWTs are often used for authentication and authorization purposes in web development and APIs.

### Components of a JWT:

- **Header:**
  The header typically consists of two parts: the type of the token (JWT) and the signing algorithm being used, such as HMAC SHA256 or RSA.
- **Payload:**
  The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data.
  - Common claims include:
    - iss (issuer)
    - sub (subject)
    - aud (audience)
    - exp (expiration time)
    - nbf (not before)
    - iat (issued at)
    - jti (JWT ID)
- **Signature:**
  To create the signature part, you need to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

**Advantages of JWT:**

- **Compact and Efficient:**
  - JWTs are designed to be compact, which makes them efficient for transmission and processing.
- **Self-Contained:**
  - JWTs contain all the necessary information within the token, reducing the need for constant database queries.
- **Easy to Verify:**
  - The signature in a JWT allows for easy verification of the token's integrity.

- **Standardized:**
    - JWT is an open standard (RFC 7519), which means it is well-defined and widely supported.

## Security Considerations:

- **Encryption:**
    - While JWTs are signed to ensure integrity, sensitive information should be encrypted within the JWT.
- **Expiration:**
    - JWTs should have a reasonable expiration time to mitigate the risk of token misuse.
- **Secure Transmission:**
    - Use HTTPS to secure the transmission of JWTs over the network.
- **Secret Management:**
    - Protect the JWT signing secret to prevent unauthorized parties from creating fake tokens.
- **Validation:**
    - Always validate incoming JWTs, checking the signature and expiration time.

## HTTP caching

Caching is a technique of storing frequently used data or information in a local memory, for a certain time period. So, next time, when the client requests the same information, instead of retrieving the information from the database, it will give the information from the local memory. The main advantage of caching is that it improves the performance by reducing the processing burden. HTTP caching is a mechanism used in the Hypertext Transfer Protocol (HTTP) to store and reuse previously fetched resources, such as web pages, images, and other types of content.

### Key Concepts in HTTP Caching:

- **Cache-Control Header:**
    - The Cache-Control header is a crucial part of HTTP caching. It includes directives that guide how caches should behave. Common directives include:
        - **max-age:** Specifies the maximum amount of time a resource is considered fresh.
        - **no-cache:** Forces caches to validate the resource with the server before using a cached copy.
        - **no-store:** Instructs caches not to store the resource under any circumstances.

    **ETag (Entity Tag):**

    - The ETag is an identifier assigned by the server to a specific version of a resource. When a client requests a resource, it can include the If-None-Match header with the ETag of its cached copy. If the resource hasn't changed (matching ETag), the server responds with a "304 Not Modified" status.
- **Last-Modified:**
    - The Last-Modified header indicates the date and time when the resource was last modified. If the client has a cached copy, it can include the If-Modified-Since header in the request. If the resource hasn't been modified since the specified date, the server responds with a "304 Not Modified" status.

## Versioning

Web API Versioning is required as the business grows and business requirement changes with the time. As Web API can be consumed by multiple clients at a time, Versioning of Web API will be necessarily required so that Business changes in the API will not impact the client that are using/consuming the existing API.

### Web API Versioning Ways

Web API Versioning can be done by using the following methods:

- **URI**:In URI versioning, the version information is included directly in the URI.
  - Pros:
    - Explicit and easy to understand.
    - Straightforward for clients to select the desired version.
  - Cons:
    - URI can become cluttered with version information.
    - May not be the most RESTful approach as URI should ideally identify resources.

- **QueryString parameter** : In QueryString parameter versioning, the version information is included as a parameter in the query string
  - Pros:
    - Keeps the URI cleaner than URI versioning.
    - Allows clients to easily switch versions.
  - Cons:
    - The version information is not as visible in the URI.

- **Custom Header parameter** : In Custom Header parameter versioning, the version information is included in a custom header.
    - Pros:
        - Keeps URI clean.
        - Allows for a separation of concerns, as versioning is a metadata concern.
    - Cons:
        - Requires clients to be aware of and send the custom header.

- **Accept Header parameter** : In Accept Header parameter versioning, the version information is included in the Accept header of the HTTP request.
    - Pros:
        - Aligns with content negotiation principles.
        - Provides flexibility for versioning different media types.
    - Cons:
        - Requires clients to set the Accept header correctly.

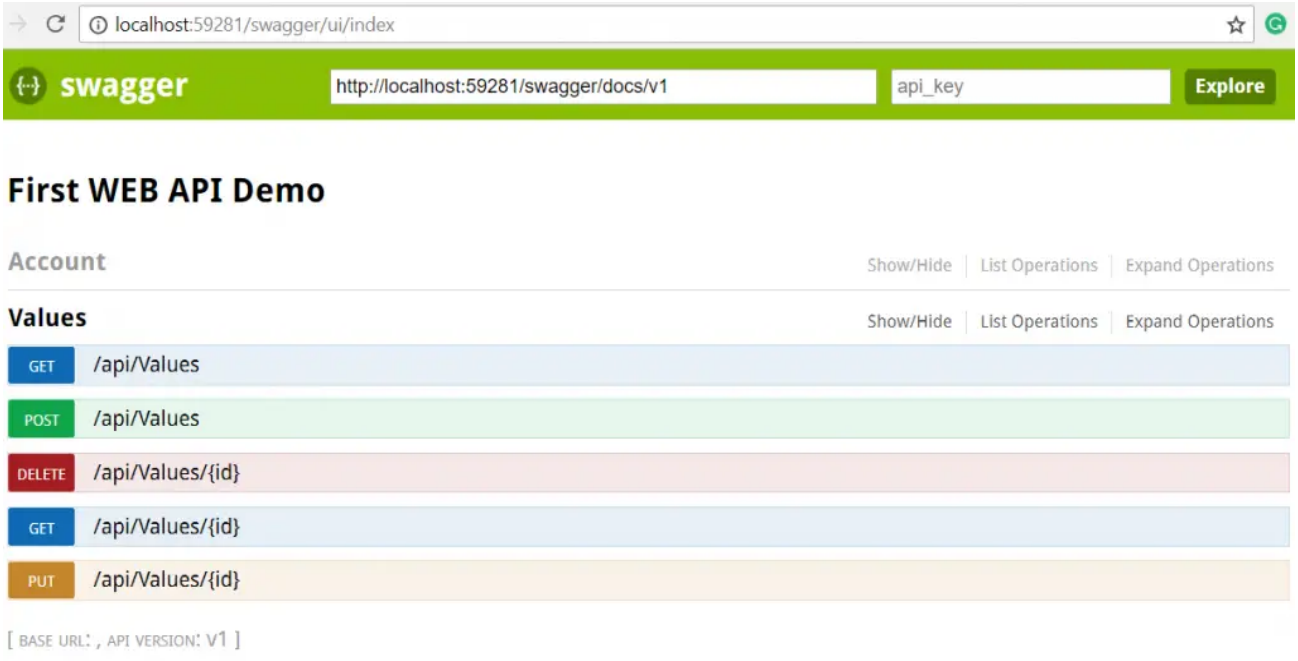
## Use of Swagger

The Swagger is a simple but powerful representation of the RESTful API. Nowadays, most developers use Swagger in almost every modern programming language and deployment environment to document. With a Swagger-enabled Web API, you will get interactive documentation, client SDK generation, and discoverability.

Swagger (OpenAPI) is a language-agnostic specification for describing REST APIs. It allows both computers and humans to understand the capabilities of a REST API without direct access to the source code. Its main goals are to:

- Minimize the amount of work needed to connect decoupled services.
- Reduce the amount of time needed to accurately document a service.

Swagger UI offers a web-based UI that provides information about the service, using the generated OpenAPI specification. Both Swashbuckle and NSwag include an embedded version of Swagger UI, so that it can be hosted in your ASP.NET Core app using a middleware registration call.



## Use of POSTMAN

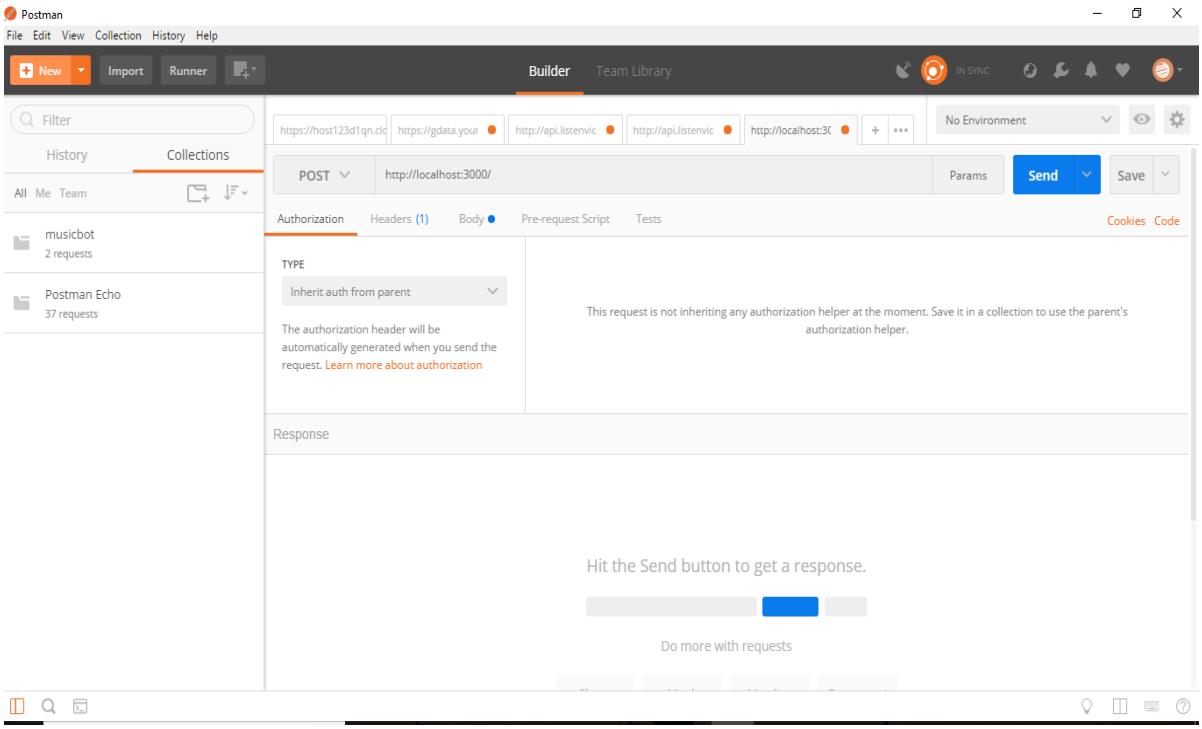
Postman is one of the most popular software testing tools which is used for API testing. With the help of this tool, developers can easily create, test, share, and document APIs.

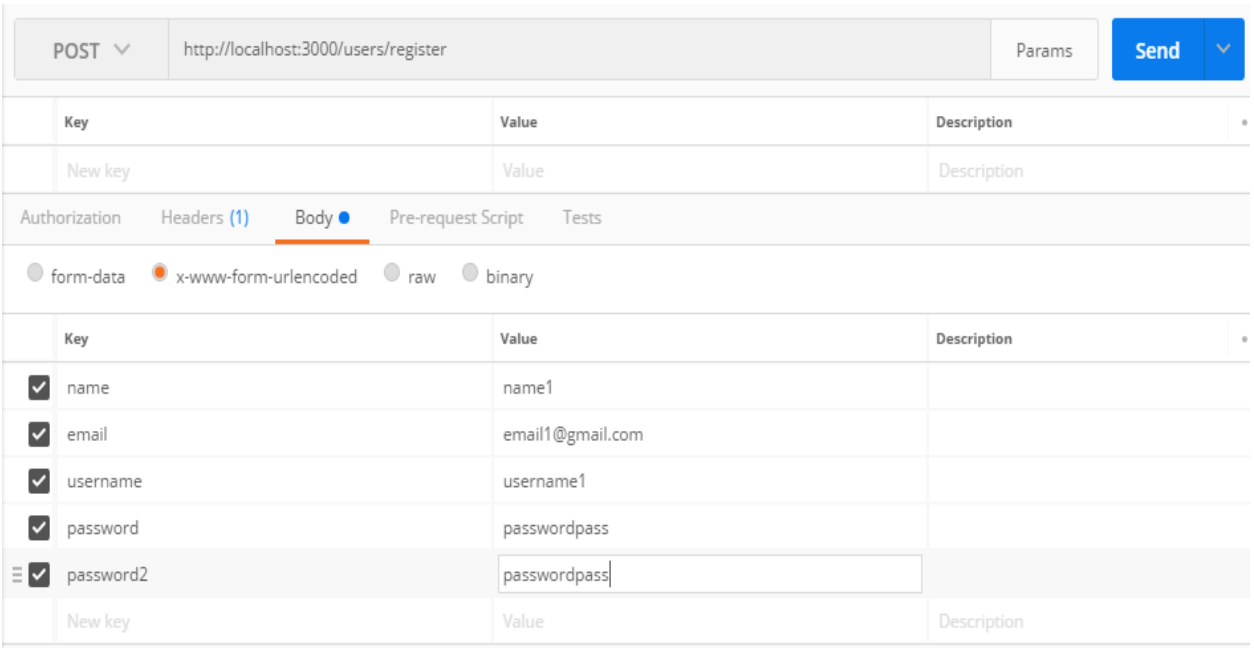
- Postman is a standalone software testing API (Application Programming Interface) platform to build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.
- While using Postman, for testing purposes, one doesn't need to write any HTTP client network code. Instead, we build test suites called collections and let Postman interact with the API.
- In this tool, nearly any functionality that any developer may need is embedded. This tool has the ability to make various types of HTTP requests like GET, POST, PUT, PATCH, and convert the API to code for languages like JavaScript and Python.

### Why use Postman?

- **Accessibility-** One can use it anywhere after installing Postman into the device by simply logging in to the account.
- **Use Collections**-Postman allows users to build collections for their API-calls. Every set can create multiple requests and subfolders. It will help to organize the test suites.
- **Test development**- To test checkpoints, verification of successful HTTP response status shall be added to every API- calls.
- **Automation Testing**-Tests can be performed in several repetitions or iterations by using the Collection Runner or Newman, which saves time for repeated tests.
- **Creating Environments**- The design of multiple environments results in less replication of tests as one can use the same collection but for a different setting.

- **Debugging-** To effectively debug the tests, the postman console helps to track what data is being retrieved.
- **Collaboration-** You can import or export collections and environments to enhance the sharing of files. You may also use a direct connection to share the collections.
- **Continuous integration-**It can support continuous integration.




POST
http://localhost:3000/users/register
Params
Send
Key
Value
Description
New key
Value
Description
Authorization
Headers (1)
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Description
name
name1
email
email1@gmail.com
username
username1
password
passwordpass
password2
passwordpass
New key
Value
Description

## Deployment

Deploying a Web API involves making the application accessible on a server or a hosting environment. Below is a descriptive document outlining the steps for deploying a Web API. Note that the specifics can vary based on your application requirements, hosting environment, and technology stack.

### Prerequisites:

- Completed development and testing of your Web API.
- A hosting environment or server where the Web API will be deployed.
- Appropriate permissions and credentials for server access.
- Dependencies and configurations required for the Web API.