HTTP GET

The HTTP GET method is **used to retrieve resources from a server**. It is the most common HTTP method and is primarily used for sending queries in the form of URL parameters to the server.

1. Characteristics of GET

- Purpose
  - **Fetch data** (read-only operation).
- Data Transmission
  - Data is appended to the URL as **query parameters**.
  - Format: ?key1=value1&key2=value2.
- **Visible** in URL
  - All query parameters are visible in the URL, which can be cached, logged, or bookmarked.
- **Idempotent**
  - Multiple identical GET requests will produce the **same result**.
- Bookmarkable
  - URLs with query parameters can be saved and reused.

2. GET Request Format

Basic GET Request Example

- URL:

```
https://example.com/search?query=books&category=fiction
```

- Request Headers

```
GET /search?query=books&category=fiction HTTP/1.1
Host: example.com
```

- Query Parameters
  - query: books
  - category: fiction

3. Queries in GET Requests

Structure

- Query strings **start with a ?** after the base URL.
- Multiple **key-value pairs** are separated by &.
- Example

```

```

```
https://example.com/products?item=laptop&price=1000&sort=asc
```

How Queries Are Used

- Search: https://example.com/search?keyword=shoes
- Filters: https://example.com/items?category=electronics&sort=price_low_to_high
- Pagination: https://example.com/page?page=2&limit=20

4. GET vs. POST

Aspect	GET	POST
Data Location	Sent as URL query parameters.	Sent in the HTTP request body.
Visibility	Data is visible in the URL.	Data is hidden in the body.
Length Limit	Limited by browser/server URL length.	No strict limit for data size.
Use Case	Fetching or querying data.	Submitting data (e.g., form submission).
Idempotent	Yes (safe and repeatable).	No (multiple requests may create changes).
Caching	GET requests can be cached.	POST requests are generally not cached.

5. Security Concerns with GET

1. **Data Exposure**

- Sensitive data (e.g., passwords, tokens) should never be sent via GET because query parameters are visible in:
  - Browser history.
  - URL logs on servers.
  - Bookmarks.
  - Proxy or analytics logs.
- Example (Unsafe)

```
https://example.com/login?username=admin&password=1234
```

2. **Caching Risks**

- GET requests can be cached, leading to unintended exposure of sensitive data.

3. **Length Limit**

- Browsers and servers impose limits on URL length (usually around 2000–8000 characters).

4. **Bookmarking**

- GET URLs can be bookmarked, potentially exposing sensitive information.

6. Example of GET Request with Queries

URL with Parameters

```
https://example.com/weather?city=seattle&unit=metric
```

HTTP Request

```
GET /weather?city=seattle&unit=metric HTTP/1.1
Host: example.com
```

Server Response

```
{
  "city": "Seattle",
  "temperature": "15°C",
  "unit": "metric"
}
```

7. Use Cases for GET

1. **Fetching Data**

- Retrieve information from APIs or databases.
- Example: https://api.example.com/users?id=123

2. **Search Queries**

- Pass search keywords or filters.
- Example: https://example.com/search?q=movies&genre=comedy

3. **Bookmarkable URLs**

- Dynamic pages with query parameters can be bookmarked and shared.

4. **Analytics and Tracking**

- Use query strings to track user activity.
- Example: https://example.com/home?utm_source=google

5. **Pagination**

- Fetch specific pages of results.
- Example: https://example.com/products?page=2&limit=10

8. Best Practices for GET Requests

1. **Avoid Sensitive Data**

- Do not send passwords, tokens, or personal information via GET.

2. **URL Length**

- Keep URLs short and manageable to avoid truncation.

3. **Proper Query Parameter Encoding**

- Encode special characters to avoid malformed URLs

```
const query = encodeURIComponent("hello world");
console.log(query); // hello%20world
```

4. **Caching Awareness**

- Understand that GET requests may be cached, and stale data could be served.

5. **Use HTTPS**

- Encrypt URLs to protect query parameters during transmission.

9. Summary

Aspect	Details
Purpose	Retrieve data from the server.
Data Location	Data is sent as query parameters in the URL.
Visibility	Query parameters are visible in the URL.
Use Cases	Fetching data, search queries, filters, pagination, analytics.
Risks	Data exposure, caching risks, and URL length limits.
Best Practices	Avoid sensitive data, encode URLs, and use HTTPS for secure transmission.

The GET method is ideal for retrieving and querying data in a web application. Its visibility in URLs makes it **easy to share and bookmark, but it requires careful handling to avoid exposing sensitive information**. By following best practices like **encoding query parameters and using HTTPS**, developers can use GET securely and efficiently.