**API Cheat Sheet**

**1. What is an API?**

* **API (Application Programming Interface)**: A set of rules that allows one piece of software or application to interact with another.
* **Types of APIs**:
  + **REST (Representational State Transfer)**: Most common, stateless, uses HTTP.
  + **SOAP (Simple Object Access Protocol)**: More rigid, XML-based messaging protocol.
  + **GraphQL**: Query language for APIs, allows for precise data fetching.
  + **gRPC**: Open-source, RPC framework that uses HTTP/2, suited for real-time services.

**2. HTTP Methods (for REST APIs)**

* **GET**: Retrieve data from a server.
* **POST**: Send data to create a resource on the server.
* **PUT**: Update or replace an existing resource.
* **PATCH**: Apply partial modifications to a resource.
* **DELETE**: Remove a resource from the server.

**3. REST API Design Principles**

* **Statelessness**: Each request from client to server must contain all the information the server needs.
* **Client-Server Separation**: Frontend (UI) and backend (logic) are decoupled.
* **Uniform Interface**: APIs should be consistent across endpoints.
* **Layered System**: Different parts of the system (client, server, database) are separated.
* **Cacheability**: Responses must be explicitly marked as cacheable or non-cacheable.

**4. URL Structure in REST APIs**

* **Base URL**: The starting point for all API requests (e.g., https://api.example.com).
* **Endpoint**: The specific path you’re accessing (e.g., /users).
* **Query Parameters**: Additional parameters in the URL for filtering/sorting (e.g., ?page=2).
* **Path Parameters**: Dynamic parts of the URL (e.g., /users/{id}).

**Example:**

bash

Copy code

GET https://api.example.com/users?page=2

GET https://api.example.com/users/{userId}/orders

**5. Status Codes**

* **1xx** (Informational): Request received, continuing process.
* **2xx** (Success): The request was successfully received.
  + **200 OK**: The request was successful.
  + **201 Created**: A new resource has been successfully created.
  + **204 No Content**: Request succeeded, but there’s no content to return.
* **3xx** (Redirection): Further action is needed to complete the request.
  + **301 Moved Permanently**: The resource has been moved to a new URL.
  + **304 Not Modified**: Cached version is still up to date.
* **4xx** (Client Error): The request contains bad syntax or cannot be fulfilled.
  + **400 Bad Request**: The server could not understand the request.
  + **401 Unauthorized**: Authentication is required.
  + **403 Forbidden**: The request is understood, but it’s refused.
  + **404 Not Found**: The requested resource could not be found.
  + **429 Too Many Requests**: Rate-limiting; client sent too many requests.
* **5xx** (Server Error): The server failed to fulfill a valid request.
  + **500 Internal Server Error**: The server encountered an unexpected error.
  + **502 Bad Gateway**: Invalid response from the upstream server.
  + **503 Service Unavailable**: The server is temporarily unavailable.

**6. Authentication & Authorization**

* **API Keys**: A token used to authenticate and authorize API requests.
* **OAuth 2.0**: Open standard for access delegation (e.g., allowing apps to access user data).
* **JWT (JSON Web Tokens)**: Encoded JSON tokens used for securely transmitting data between parties.

**7. Versioning**

* API versioning helps maintain backward compatibility.
  + **In URL**: https://api.example.com/v1/users
  + **In Headers**: Accept: application/vnd.example.v1+json
  + **In Query Params**: https://api.example.com/users?version=1

**8. API Documentation**

* **Swagger/OpenAPI**: Tools for creating interactive API documentation.
* **Postman**: A tool for testing API endpoints and generating documentation.

**9. Rate Limiting**

* **Throttling**: Restricting the number of API requests allowed per user over a certain period.
* **Common HTTP Headers**:
  + **X-RateLimit-Limit**: Maximum number of allowed requests.
  + **X-RateLimit-Remaining**: Number of remaining requests in the window.
  + **X-RateLimit-Reset**: Time when the limit will reset.

**10. Pagination**

* Large datasets are often paginated in APIs. Common methods:
  + **Offset/Limit**: e.g., GET /users?offset=20&limit=10
  + **Cursor-based Pagination**: e.g., GET /users?cursor=abc123
  + **Link Headers**: Provide rel="next" and rel="prev" links in headers.

**11. Error Handling**

* **Standardized Response Format**: APIs should return errors in a standardized format, often in JSON, e.g.:

json

Copy code

{

"error": {

"code": 404,

"message": "Resource not found"

}

}

* **Use Descriptive Error Messages**: Return meaningful and useful error descriptions.

**12. CORS (Cross-Origin Resource Sharing)**

* Browsers enforce CORS to restrict web applications from making requests to a different domain.
* **Headers**:
  + **Access-Control-Allow-Origin**: Controls which origins are allowed.
  + **Access-Control-Allow-Methods**: Specifies allowed HTTP methods.

**13. API Security Best Practices**

* **HTTPS**: Always use HTTPS to encrypt API traffic.
* **Token Expiration**: Set expiration dates for tokens (like JWT).
* **Input Validation**: Sanitize inputs to avoid SQL injection and XSS.
* **Rate Limiting**: Implement to prevent abuse.
* **Authentication & Authorization**: Use OAuth2, JWT, etc., appropriately.