Title: Postman Fundamentals (Day 2)

Slide 1: Agenda

1. API Fundamentals
2. Introduction to Postman
3. Postman Basic Workflow
4. Postman Advanced Features
5. Summary
6. Hand-On Labs
7. Q&A

Slide 2: Introduction to APIs

- **API Concepts**
  - API: Application Programming Interface; enables interaction between systems using standards and protocols.
  - Acts as a bridge between various applications/systems.

- **REST APIs**
  - REST: Representational State Transfer, an architectural style for web applications.
  - HTTP Methods: Used to transfer data (e.g., GET, POST, PUT, DELETE).

Slide 3: Anatomy of an HTTP Request

- **URL/URI**

  - URL: Uniform Resource Locator - Identifies the resource.
    - Example: `http://api.example.com/v1/helloworld`
  - URI: Uniform Resource Identifier - Identifies which resource to access.
    - Example: `http://api.example.com/v2/helloworld?lang=en`
  - Base URL: Common part of all requests.
    - Example: `http://api.example.com/v1/`

Slide 4: Anatomy of an HTTP Request

- **Request Body**
  - Data sent for creating/updating a resource.

```
{
    "greetings": "Hello World!"
}
```

- **Request Parameters**
  - **Query Parameters**: Variables in the URI for filtering resources.
    - Example: `http://api.example.com/orders?startDate=06/08/2024&endDate=06/12/2024`
  - **Path Parameters**: Variables in the URI path for specific resources.
    - Example: `http://api.example.com/order/1/items`

Slide 5: Anatomy of an HTTP Request

- **Headers**:
  - Additional information sent in a request (e.g., authentication tokens, content type).
  - **Common Headers**:
    - `Content-type`: MIME type of the response body.
    - `Authorization`: Client credentials for protected resources.
    - `User-Agent`: Identifies the client application.
    - `Content-length`: Specified the length of the response body in bytes.
    - `Cache-Control`: Specify the cache behavior of the response
    - `Location`: Specifies the URI of the resources that can be used to retrieve the request resources.
    - `Server`: Specifies the name of the name and version of the server software that generated the response.
    - `Accept`: The Accept header defines the media types that the client can accept from the server.

Slide 6: Anatomy of an HTTP Request

- **HTTP Methods**

  - **GET**: Retrieve a resource.
  - **POST**: Create a new resource.
  - **PUT**: Update an existing resource.
  - **DELETE**: Remove a resource.
  - **PATCH**: Partially update a resource.

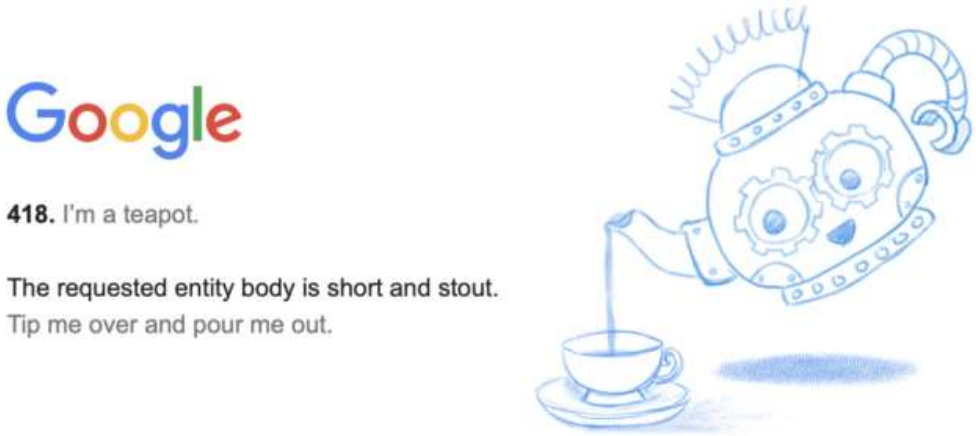
Slide 7: Anatomy of an HTTP Request

- HTTP Status Code
  - **1XX**: Informational (e.g., 100 - Continue)
  - **2XX**: Success (e.g., 200 - OK)

- **3XX**: Redirection (e.g., 301 - Moved Permanently)
- **4XX**: Client Errors (e.g., 404 - Not Found)
- **5XX**: Server Errors (e.g., 503 - Service Unavailable)

Slide 8: What is HTTP 418 Status?

Hint: "I'm a teapot"



Slide 9: API Design and Architecture

- **Design Patterns**
  - REST: Uses HTTP requests to perform CRUD operations.
  - SOAP: Simple Object Access Protocol for exchanging structured data.
  - GraphQL: Query language for APIs.

Slide 10: Use-Case of API design patterns

Case Study I: Twitter API
Overview

- **Twitter API**: A RESTful API used by developers to interact with Twitter data.
- **Functionality**: Allows developers to read and write Twitter data, including tweets, user profiles, and trends.

Features

- Endpoints: Multiple endpoints for different resources (e.g., tweets, users, trends).

- Authentication: OAuth 1.0a for secure access.
- Data Format: JSON for easy integration with web and mobile apps.

Example Use Case

- Tweet Retrieval: Fetching recent tweets containing a specific hashtag.
- GET https://api.twitter.com/2/tweets/search/recent?query=%Postman

Case Study II: PayPal API

Overview

- PayPal API: A SOAP-based API used for processing online payments.
- Functionality: Handles various payment operations such as transactions, refunds, and subscription management.

Features

- Endpoints: Single endpoint for different operations, specified in the SOAP body.
- Authentication: API credentials (username, password, and signature).
- Data Format: XML for structured data exchange.

Example Use Case

- Payment Processing: Making a payment through a PayPal account.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ebay:api:PayPalAPI">
  <soapenv:Header/>
  <soapenv:Body>
    <urn:DoExpressCheckoutPaymentReq>
      <urn:DoExpressCheckoutPaymentRequest>
        <!-- Payment details here -->
      </urn:DoExpressCheckoutPaymentRequest>
    </urn:DoExpressCheckoutPaymentReq>
  </soapenv:Body>
</soapenv:Envelope>
```

Case Study II: GitHub API

Overview

- **GitHub API**: A GraphQL-based API used for interacting with GitHub data.
- **Functionality**: Provides detailed queries and mutations to access and modify GitHub resources such as repositories, issues, and user profiles.

Features

- **Endpoint**: Single endpoint for all operations (https://api.github.com/graphql).
- **Authentication**: Personal access tokens for secure access.

Slide 12: Architectural Styles

- **Microservices**: Small, independent services.
- **Serverless**: Cloud provider manages infrastructure and resource allocation.
- **SOA**: Service-Oriented Architecture for distributed systems.
- **Event-Driven**: Data flow triggered by events.

Slide 11: API Security

- **Security Standards**
  - SSL/TSL: Secure connection protocols.

    - API Key: Secret token for authenticating requests.
    - OpenID Connect: Authentication layer on top of OAuth.
    - JWT Token: JSON Web Token for secure data transmission.
    - OAuth: Open standard for authorization.
    - CORS: Cross-Origin Resource Sharing for requesting resources from different domains.

Slide 14: API Testing Tools

- **Popular Tools**
    - Postman: Tool for testing and debugging APIs.
    - SOAPUI: Tests SOAP and REST web services.
    - Swagger: Designs, builds, and tests APIs.
    - JMeter: Tests performance of APIs.
    - Test Rail: Manages API testing.
    - Dredd: Command-line tool for testing API documentation.
    - REST Assured: Java library for testing RESTful APIs.
    - Karate DSL: Testing framework using Gherkin syntax.

Slide 15: Using Postman

- **Introduction to Postman**
    - Simplifies API development with tools for testing, documentation, and monitoring.
    - **Key Features**:
        - Collections: Organize requests into folders.
        - Environments: Manage different sets of variables.
        - Tests: Validate responses with assertions.
        - Mock Servers: Simulate API endpoints.
        - Monitors: Automated testing schedules.

Slide 16: Postman Basic Workflow

1. **Create a Request**: Specify URL, method, headers, and body.
2. **Send Request**: Execute and observe the response.
3. **Analyze Response**: Inspect status code, headers, and body.
4. **Write Tests**: Create scripts to validate response data.
5. **Save and Organize**: Store requests in collections for reuse.

Slide 17: Postman Advanced Features

- **Automated Testing**: Create detailed test suites.
- **Load Testing:** For Load Testing / Benchmarking API performance
- **API Documentation**: Generate and share API documentation.
- **Collaboration**: Share collections and workspaces with team members.

Slide 18: Summary

- **Understanding APIs**: Essential concepts and components.
- **API Design & Security**: Best practices and standards.
- **Postman Usage**:
    - Key features,
    - Advanced Capabilities.
    - Test Suite
    - Functional Testing
    - Performance Testing

Slide 19: Q&A

- **Open the floor for questions.**
- **Encourage discussion and feedback.**