**Week 4: Principles of RESTful API Design, HTTP Methods, and Error Handling**

**Session 1: Theory**

**1. Principles of RESTful API Design**

**1.1 What is REST?**

* REST (**Representational State Transfer**) is an **architectural style** used for designing **scalable and stateless APIs**.
* It follows the principles of **client-server communication** over HTTP.
* REST APIs are widely used for web and mobile applications due to their simplicity and flexibility.

**1.2 Key Principles of RESTful APIs**

| **Principle** | **Description** |
| --- | --- |
| **Statelessness** | The server does not store client session data. Each request is independent. |
| **Client-Server Architecture** | The client and server operate independently. |
| **Uniform Interface** | Consistent and predictable API design (e.g., using standard HTTP methods). |
| **Resource-Based** | Everything is treated as a resource (/users, /products, etc.). |
| **Representation of Resources** | Resources can be represented in formats like JSON or XML. |
| **Cacheability** | API responses can be cached to improve performance. |
| **Layered System** | APIs can have multiple layers (e.g., authentication, business logic, database). |
| **HATEOAS (Hypermedia as the Engine of Application State)** | API responses contain links to guide clients on next actions. |

**1.3 Designing a RESTful API**

* **Use meaningful resource names** (e.g., /users instead of /getUsers).
* **Follow resource hierarchy** (e.g., /users/{id}/tasks).
* **Use plural nouns for collections** (e.g., /tasks instead of /task).
* **Use standard HTTP methods (GET, POST, PUT, DELETE, PATCH)**.
* **Include proper status codes** (e.g., 201 Created for successful resource creation).
* **Implement pagination and filtering** for large datasets.



**2. HTTP Methods and Status Codes**

**2.1 Common HTTP Methods in REST APIs**

| **HTTP Method** | **Description** | **Example** |
| --- | --- | --- |
| **GET** | Retrieve a resource | GET /users/1 |
| **POST** | Create a new resource | POST /users |
| **PUT** | Update an entire resource | PUT /users/1 |
| **PATCH** | Update part of a resource | PATCH /users/1 |
| **DELETE** | Remove a resource | DELETE /users/1 |

**2.2 Common HTTP Status Codes**

| **Code** | **Meaning** | **Description** |
| --- | --- | --- |
| **200 OK** | Success | Request was successful. |
| **201 Created** | Resource Created | A new resource was successfully created. |
| **204 No Content** | Success | The request was successful, but no data is returned. |
| **400 Bad Request** | Client Error | The request is invalid or missing parameters. |
| **401 Unauthorized** | Authentication Error | User is not authenticated. |
| **403 Forbidden** | Authorization Error | User is authenticated but lacks permission. |
| **404 Not Found** | Client Error | The requested resource does not exist. |
| **500 Internal Server Error** | Server Error | Unexpected error on the server. |

A close-up of a white rectangular object

AI-generated content may be incorrect.

**3. Error Handling in APIs**

**3.1 Importance of Error Handling**

* Provides meaningful feedback to clients.
* Prevents exposing sensitive system details.
* Ensures API consistency and usability.

**3.2 Best Practices for API Error Handling**

1. **Use standard HTTP status codes** (e.g., 400 Bad Request, 500 Internal Server Error).
2. **Return consistent error response formats**.
3. **Include helpful error messages**.
4. **Log errors on the server for debugging**.
5. **Validate user inputs and return appropriate errors**.
6. **Avoid exposing sensitive system details**.

**3.3 Example Error Response Format (JSON)**

A close-up of a message

AI-generated content may be incorrect.

**3.4 Handling Errors in Django REST Framework (DRF)**



**Conclusion**

**RESTful APIs should be stateless, structured, and resource-based**.  
**Use standard HTTP methods and status codes for consistency**.  
**Implement robust error handling to improve API usability**.