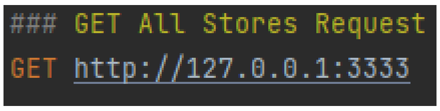
REST api Tutorial

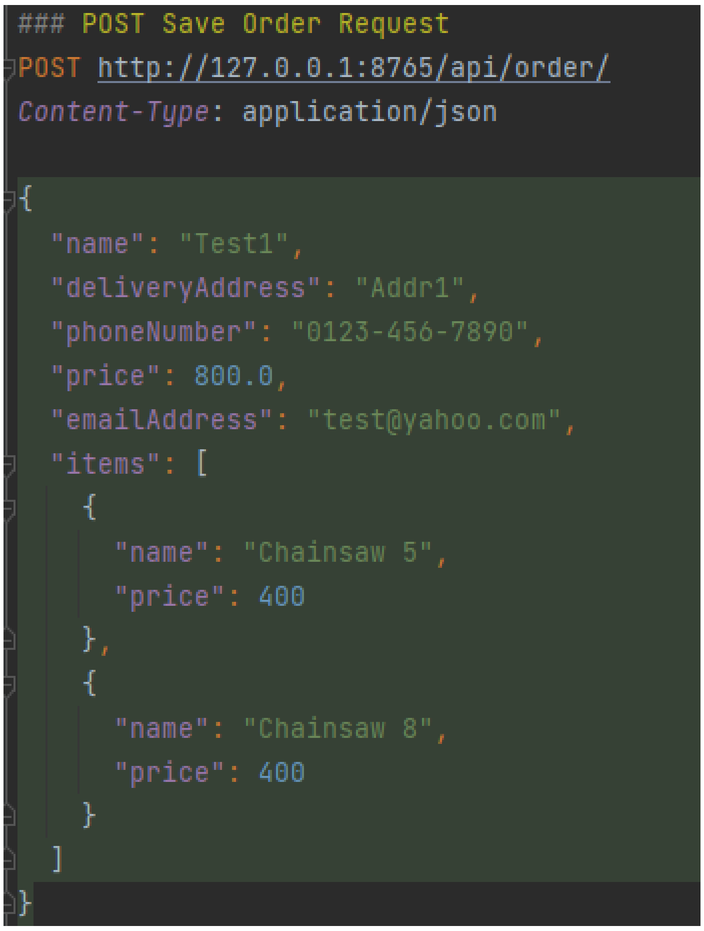
REST, which stands for Representational State Transfer, is an architectural style for designing networked applications. It relies on a stateless, client-server, cacheable communications protocol — and in virtually all cases, the HTTP protocol is used. RESTful applications use HTTP requests to perform four basic operations termed as CRUD (Create, Read, Update, Delete) on data resources:

GET: Retrieve a resource or a collection of resources.

Example: 

```
### GET All Stores Request
GET http://127.0.0.1:3333
```

POST: Create a new resource.

Example: 

```
### POST Save Order Request
POST http://127.0.0.1:8765/api/order/
Content-Type: application/json

{
  "name": "Test1",
  "deliveryAddress": "Addr1",
  "phoneNumber": "0123-456-7890",
  "price": 800.0,
  "emailAddress": "test@yahoo.com",
  "items": [
    {
      "name": "Chainsaw 5",
      "price": 400
    },
    {
      "name": "Chainsaw 8",
      "price": 400
    }
  ]
}
```

PUT: Update an existing resource.

DELETE: Remove a resource.

Key Concepts of REST:

- Resource-Based: In REST, everything is considered a resource. A resource is accessed via a common interface based on the standard HTTP methods. Each resource is identified by a unique URI (Uniform Resource Identifier).
- Stateless: Each request from client to server must contain all the information necessary to understand and complete the request. The server does not store any state about the client session on the server side.
- Cacheable: Resources must be able to be cached for better performance. The server must define which resources can be cached and for how long.
- Layered System: A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.
- Uniform Interface: This constraint is fundamental to the design of any RESTful system. It simplifies and decouples the architecture, which enables each part to evolve independently. The four guiding principles of the uniform interface are:
    - Resource Identification in Requests: Individual resources are identified in requests, for example, using URIs in web-based REST systems.
    - Resource Manipulation through Representations: When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.
    - Self-descriptive Messages: Each message includes enough information to describe how to process the message.
    - Hypermedia as the Engine of Application State (HATEOAS): Clients interact with the application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia.

Advantages of REST:

- Scalability: Due to its stateless nature, RESTful services can handle multiple requests more efficiently.
- Simplicity: Using standard HTTP methods for operations, RESTful APIs are easy to understand and implement.

- Flexibility and Portability: With data being sent and received via standard HTTP, any client can communicate with the server regardless of the underlying programming language.
- Interoperability: REST APIs can be easily integrated with other websites and services to allow for more comprehensive services and functionalities.

Uses of REST APIs:

REST APIs are used in a wide array of applications, from web services that allow for third-party integration, to mobile app backends, and even IoT (Internet of Things) devices. Their simplicity, scalability, and flexibility make them well-suited for modern web applications and services.

In summary, REST is a powerful, efficient, and widely used approach to building APIs that offer services over the web or other networks. By leveraging HTTP and its verbs (GET, POST, PUT, DELETE), RESTful services enable applications to interact seamlessly with one another and share data in a decoupled fashion.