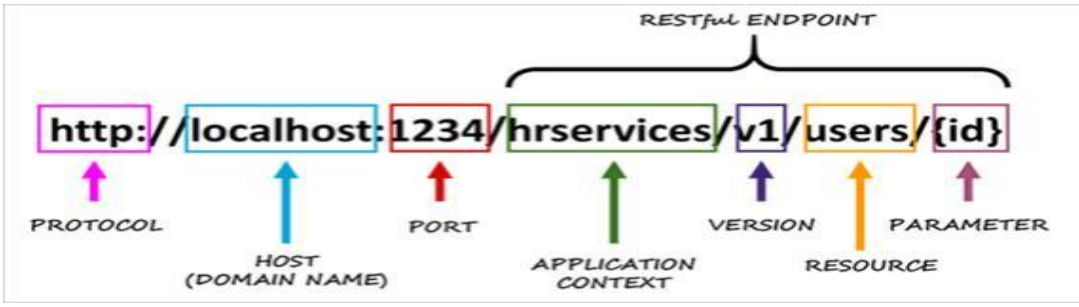
**API Development:**

The API was developed following the guidelines of REST architecture and uses JSON for data transport. Here is an overview of the steps followed in developing the API:




1. Resource identification: The first step was to identify the resources that the API would expose. Resources represent the entities or objects that the API will interact with. For example, in studio room booking website resources could include user, user's booking list, user's shared booking list, etc.

2. Endpoint design: Once the resources were identified, the next step was to design the API endpoints. Endpoints are the URLs that clients use to interact with the API's resources. Each endpoint is associated with a specific resource and represents a unique URI.
(For e.g. "http://localhost:5000/user/signup" here '/user/signup' is endpoint of this URI

3. HTTP methods: REST architecture utilizes HTTP methods (GET, POST, PUT, DELETE, etc.) to perform different operations on resources. The API was designed to use appropriate HTTP methods for each endpoint. For example, GET method was used for retrieving resource information, POST for creating new resources, PUT for updating existing resources, and DELETE for deleting resources.

4. URL structure: The API followed a meaningful and hierarchical URL structure for its endpoints. For example, a common convention is to have the resource name followed by its unique identifier (e.g., /users/123) or a collection of related resources (e.g., /posts/456/comments).

5. Data representation: JSON (JavaScript Object Notation) was chosen as the data transport format. JSON is lightweight, human-readable, and extensively supported across various programming

languages. All data sent or received by the API is in JSON format, including request and response bodies.

6. Schema definition: A schema definition for JSON data was created to provide a clear structure and validation rules. This helped in ensuring consistency and understanding of the data passed through the API.

7. Response codes: The API followed standard HTTP response codes to convey the status and outcome of each request. For example, 200 for successful responses, 404 for resource not found, 201 for successful resource creation, etc.

8. Error handling: The API implemented appropriate error handling mechanisms. Errors were returned in JSON format with relevant information about the error type, message, and any additional details.

9. Documentation: Comprehensive documentation was prepared to describe each endpoint, supported HTTP methods, request/response formats, and example usage scenarios.