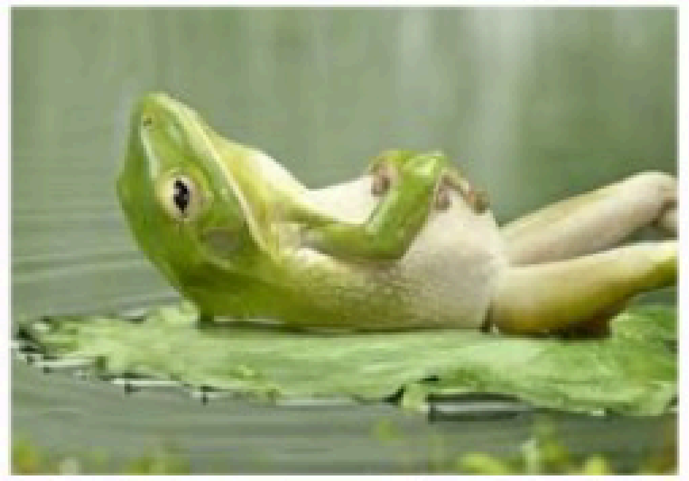
REST web service JAX RS fundamentals

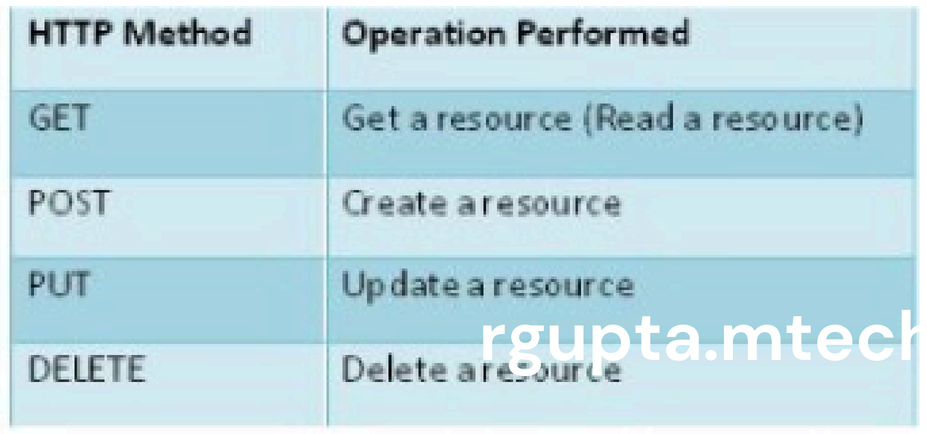
Rajeev Gupta

rgupta.mtech@gmail.com

https://www.linkedin.com/in/rajeev
guptajavatrainer

REST web service





HTTP Method	Operation Performed
GET	Get a resource (Read a resource)
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

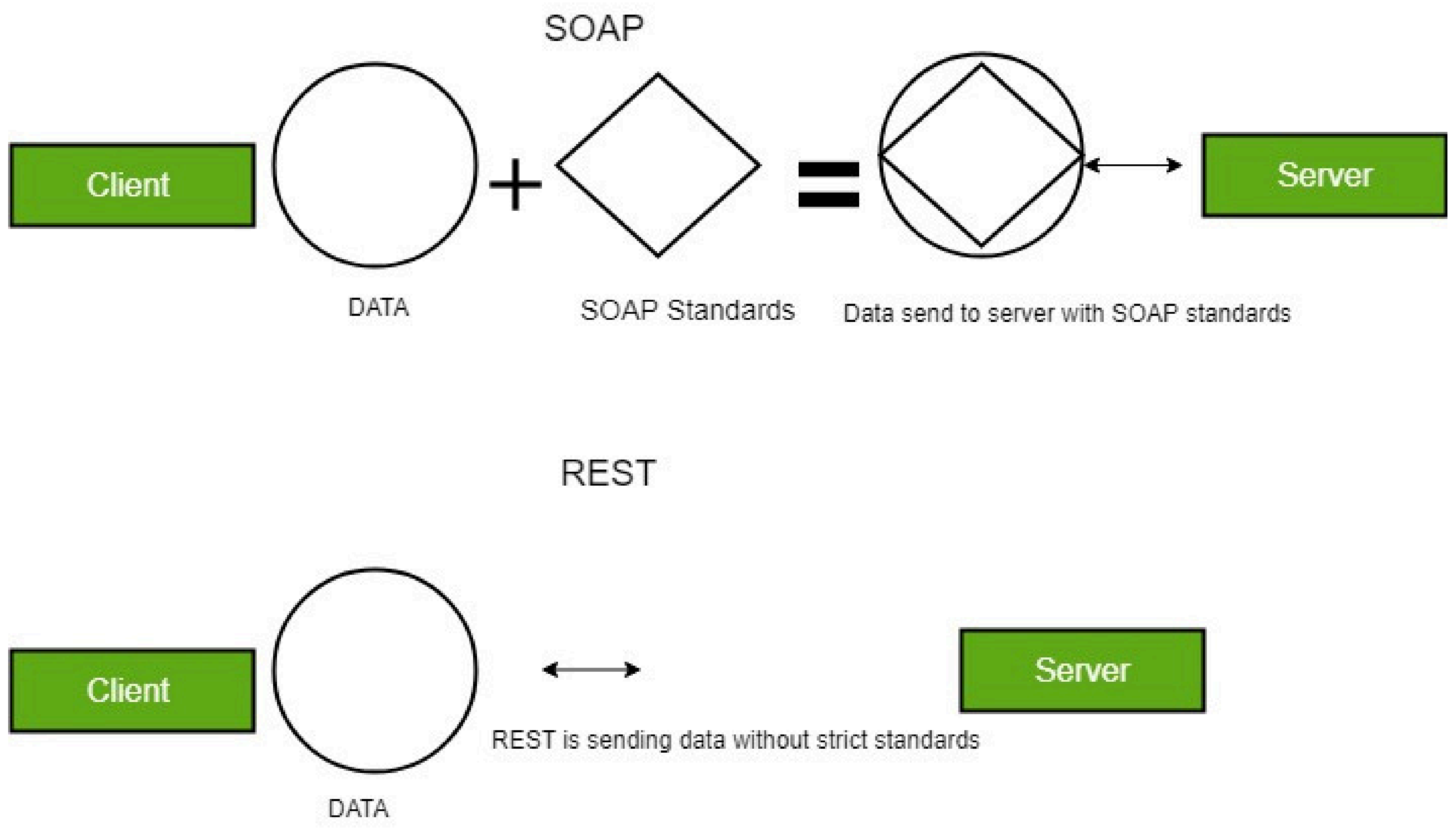
Jersey

RESTful Web Services in Java.

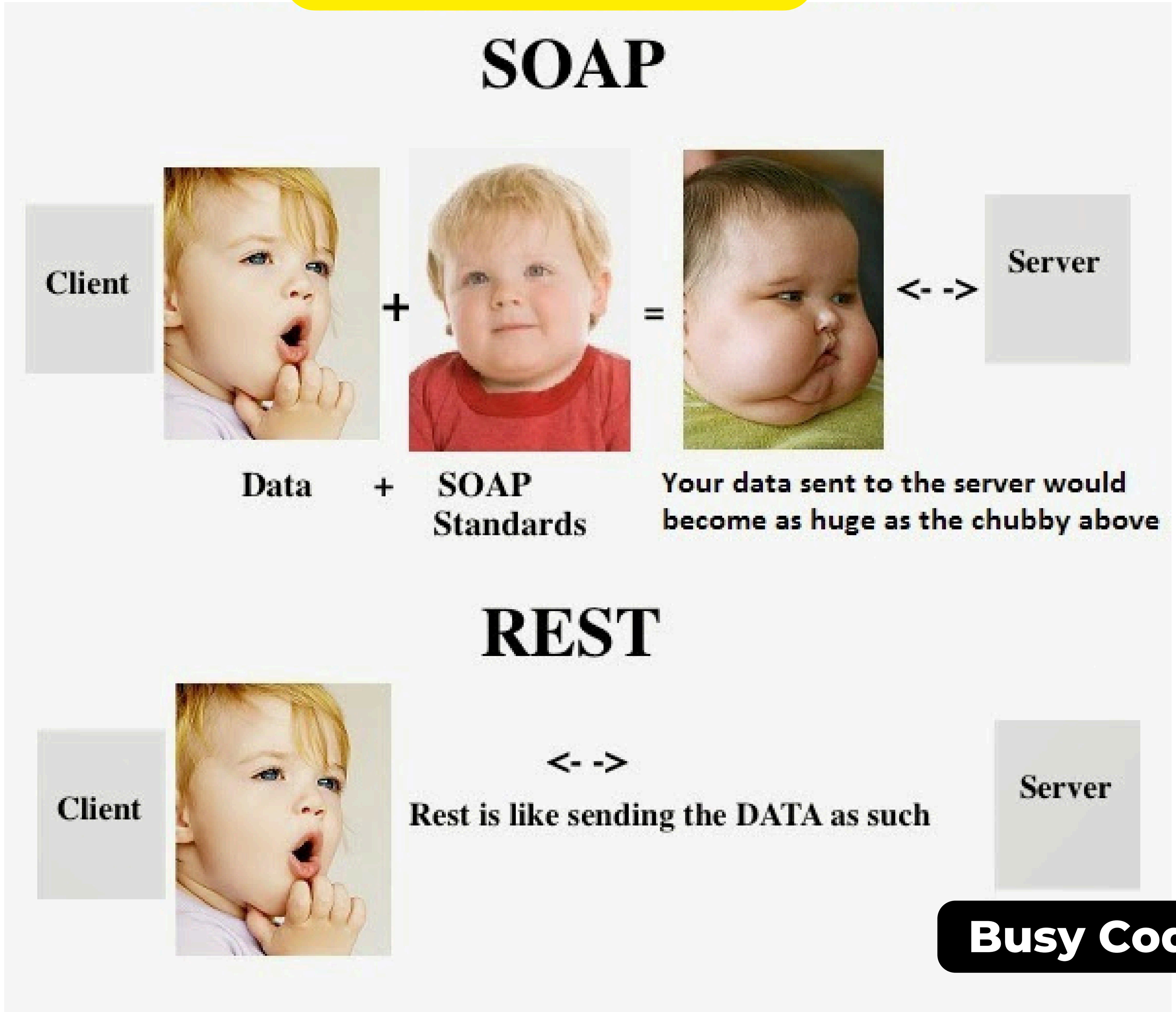
Introduction

- *REST: Representational State Transfer*
- *Introduced by Roy Fielding (2000)*
- *Architectural style (not a strict standard)*
- *Uses existing standards like HTTP*
- *REST = client-server communication*
- *REST is about manipulating resources*

SOAP vs REST


SOAP
Client
DATA
SOAP Standards
Data send to server with SOAP standards
Server
REST
Client
DATA
REST is sending data without strict standards
Server

SOAP vs REST


SOAP
Client
+
=
<- ->
Server
Data
+
SOAP Standards
Your data sent to the server would become as huge as the chubby above
REST
Client
<- ->
Rest is like sending the DATA as such
Server

Busy Coder Academy

SOAP vs REST

REST vs SOAP

Aspect	REST	SOAP
Type	Style	Standard
Transport	HTTP/HTTPS only	HTTP/HTTPS or others
Data Format	JSON, XML, Text	XML only
Request Format	URI	XML
JavaScript Usage	Easy	Hard
JSON Support	Native, lightweight	XML parsing is heavy
Service Call	Via URL path	RPC method calls
Readability	Human-readable (JSON)	Not human-readable

Busy Coder Academy

REST Basics

- *Client requests a specific resource from the server.*
- *Server delivers the requested resource.*
- *Server is stateless (does not store client information).*
- *Two identical requests from the same client are treated independently.*

Resources

- *REST server provides access to resources.*
- *REST client accesses & presents resources.*
- *Each resource identified by URI (Uniform Resource Identifier).*
- *Resources represented in formats like Text, JSON, XML.*

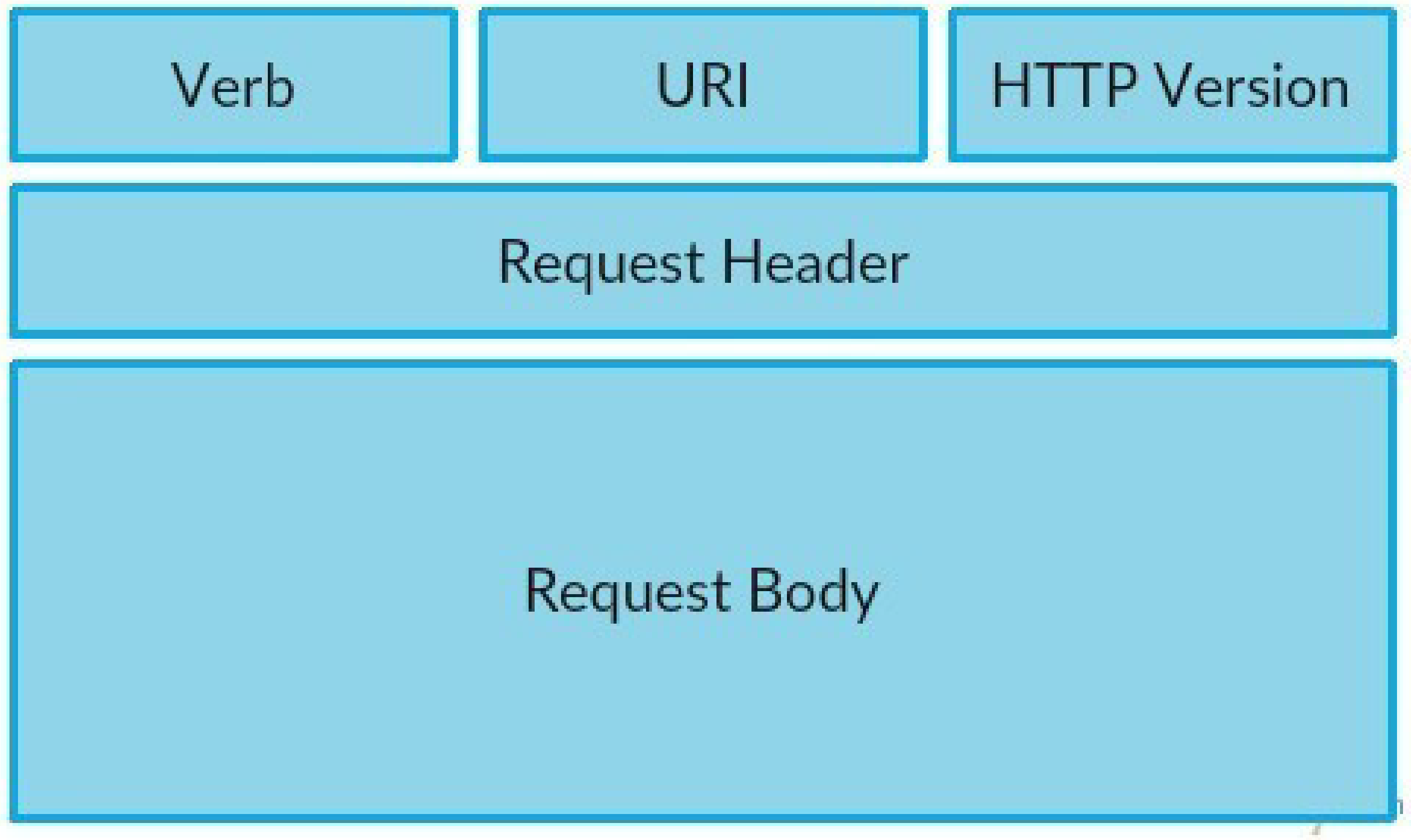
Example URI

- *http://localhost:9999/restapi/books/{id}*
- *GET → Fetch book by ID*
- *PUT → Update book by ID*
- *DELETE → Delete book by ID*

REST Communication

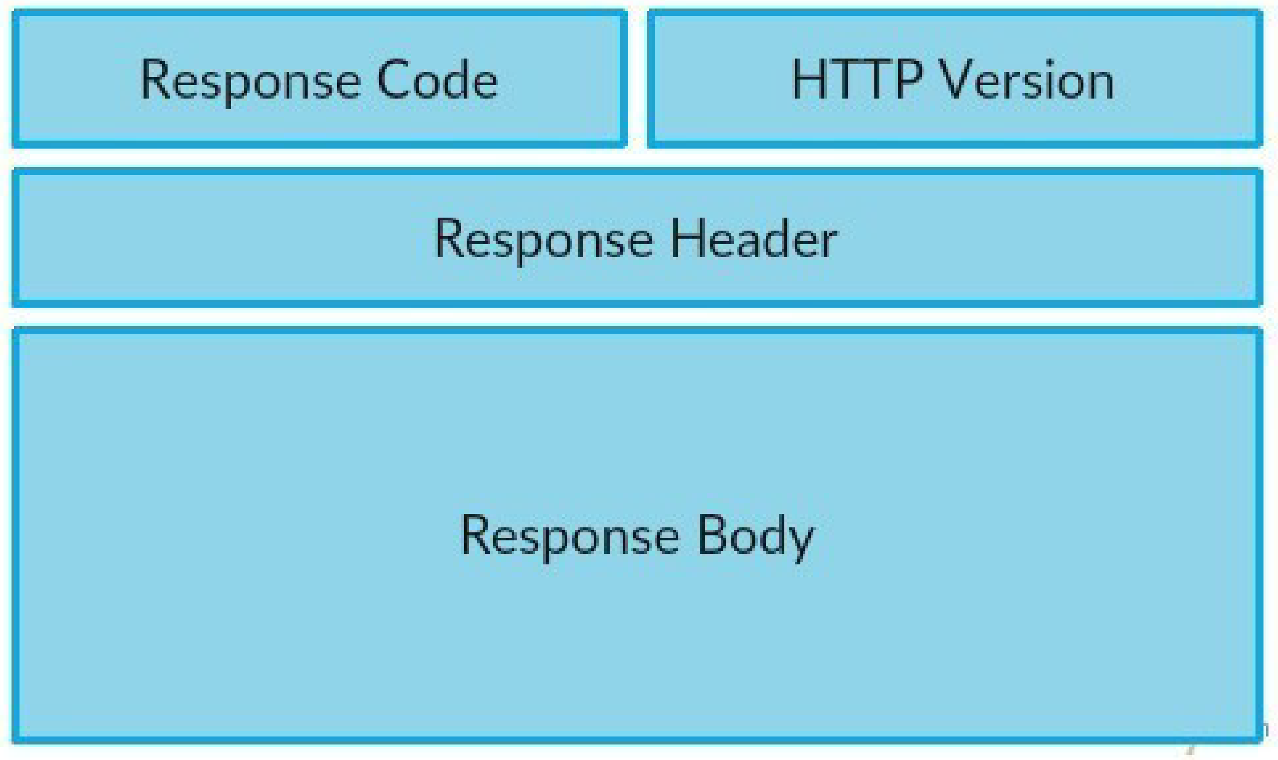
- *Uses HTTP protocol.*
- *Client sends HTTP Request, server replies with HTTP Response.*
- *Known as messaging.*

HTTP Request



- *Verb: GET, POST, PUT, DELETE etc.*
- *URI: Identifies the resource.*
- *HTTP Version: e.g., HTTP 1.1*
- *Request Header: Metadata (key-value pairs).*
- *Request Body: Content or resource representation.*

HTTP Response



- *Status/Response Code: Server status for request.*
- *HTTP Version*
- *Response Header: Metadata like content-type, length, date.*
- *Response Body: Actual content.*

Addressing

Format:

<protocol>://<servicename>/<ResourceType>/<ResourceID>

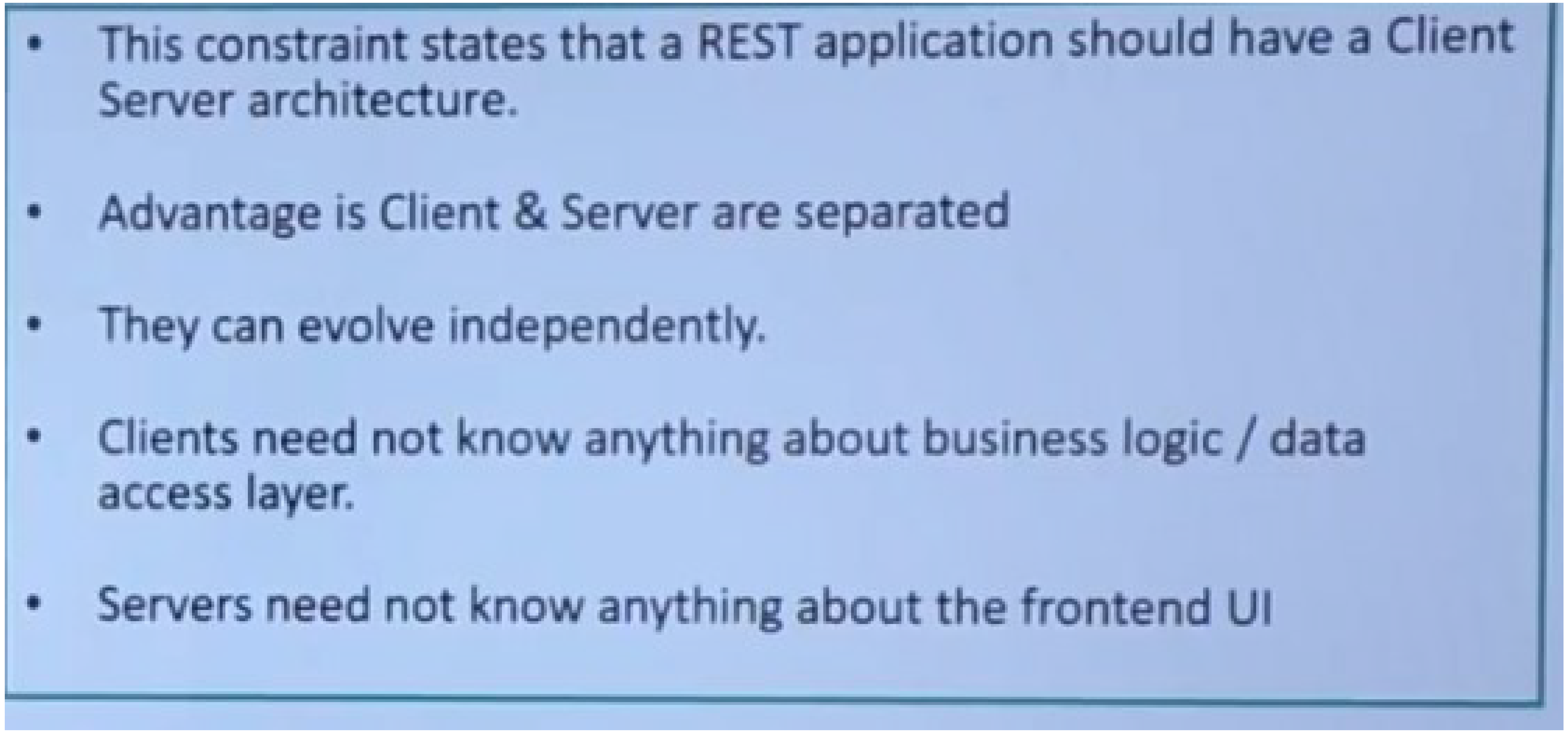
Example Methods:

- *GET – Read users*
- *POST – Create/Update user*
- *PUT – Insert user*
- *DELETE – Delete user*
- *OPTIONS – List supported operations*
- *HEAD – Returns header only*

REST Constraints

- *Client-Server*
- *Stateless*
- *Cache*
- *Uniform Interface*
- *Layered Architecture*
- *Code on Demand*
- *HATEOAS (Hypermedia as the Engine of Application State)*

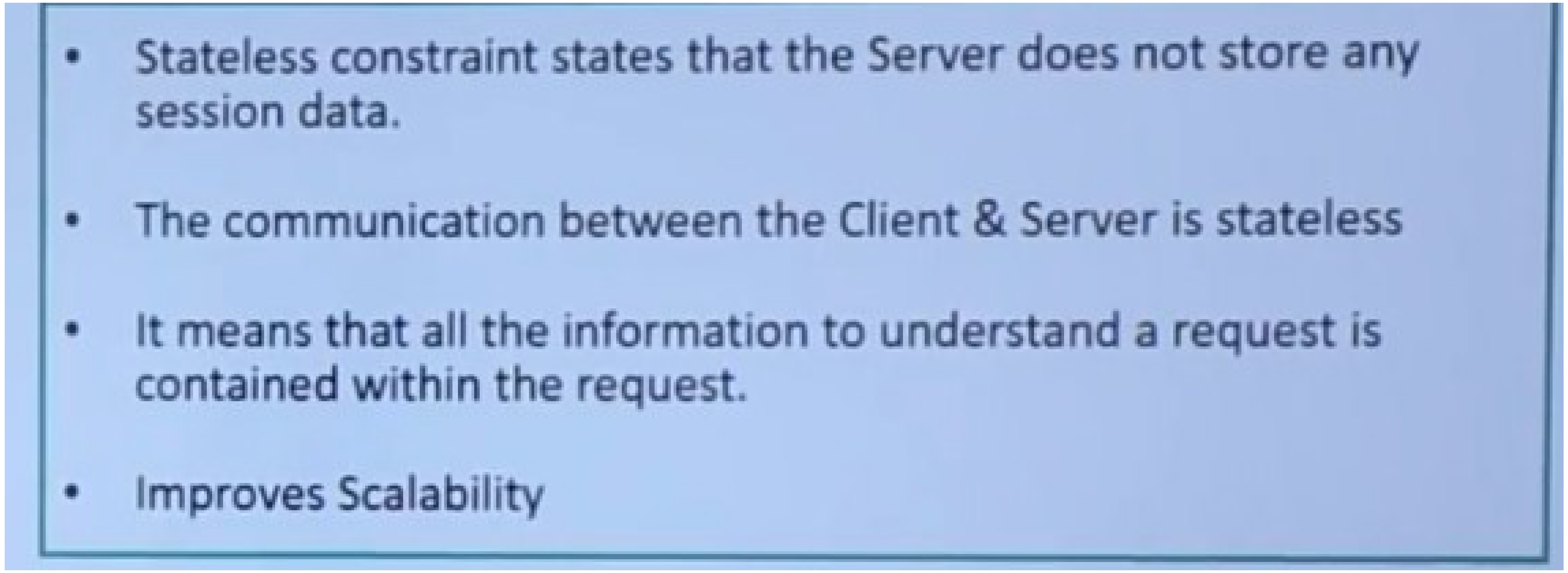
REST Constraints-Client-Server



- This constraint states that a REST application should have a Client Server architecture.
- Advantage is Client & Server are separated
- They can evolve independently.
- Clients need not know anything about business logic / data access layer.
- Servers need not know anything about the frontend UI

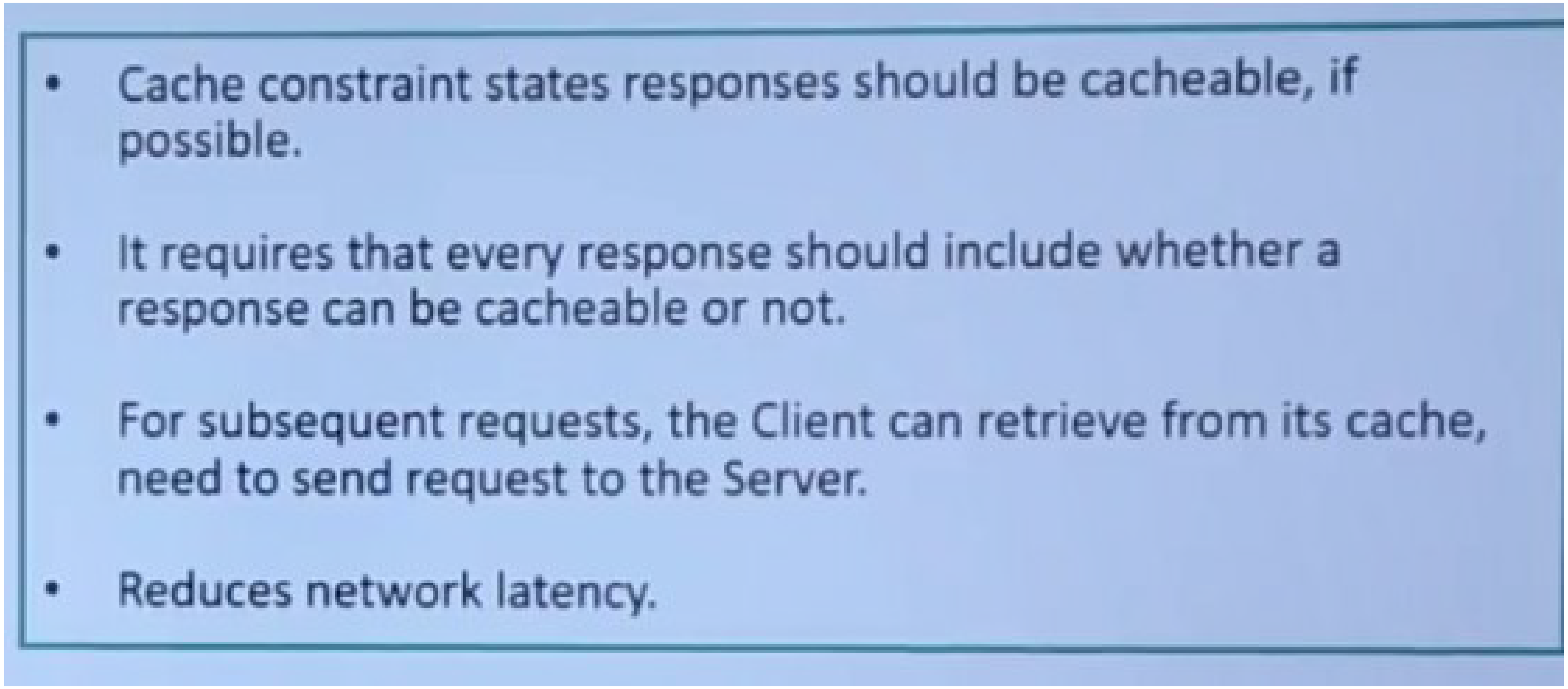
Busy Coder Academy

REST Constraints-Stateless



- Stateless constraint states that the Server does not store any session data.
- The communication between the Client & Server is stateless
- It means that all the information to understand a request is contained within the request.
- Improves Scalability

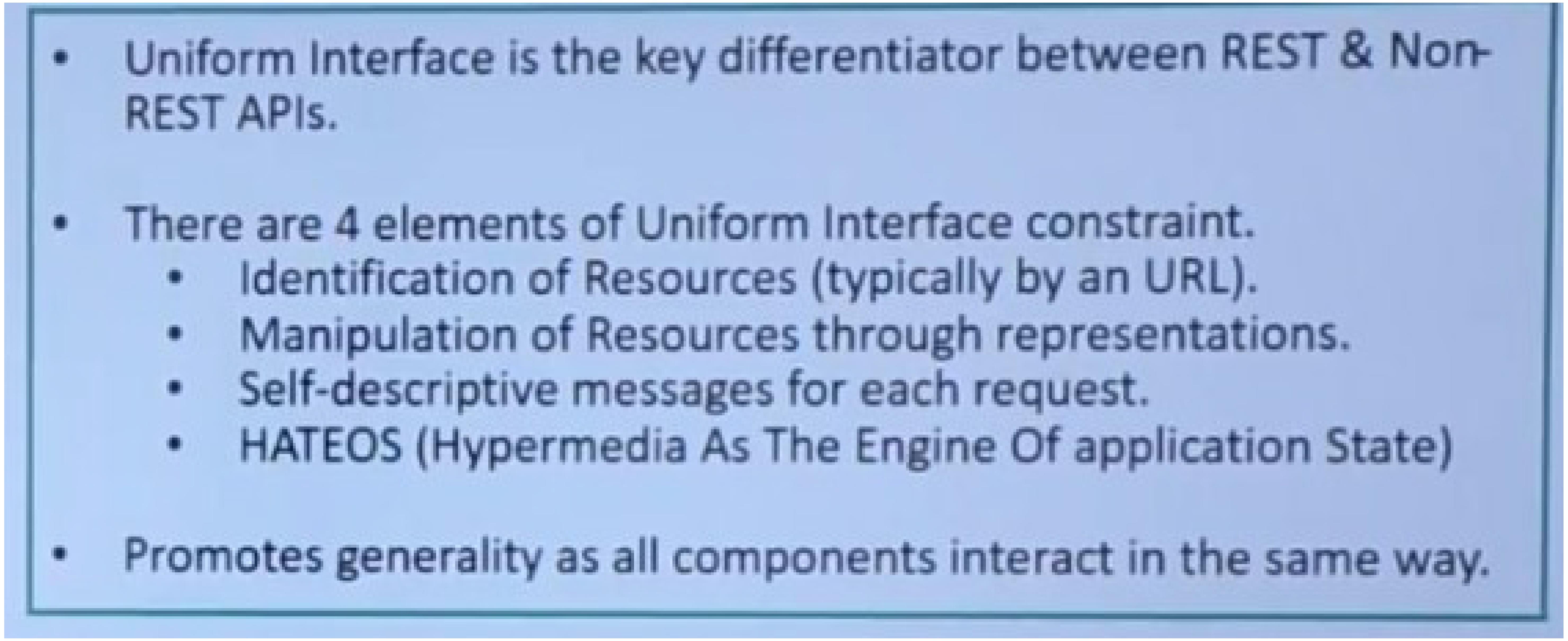
REST Constraints-Cache Constraints



- Cache constraint states responses should be cacheable, if possible.
- It requires that every response should include whether a response can be cacheable or not.
- For subsequent requests, the Client can retrieve from its cache, need to send request to the Server.
- Reduces network latency.

Busy Coder Academy

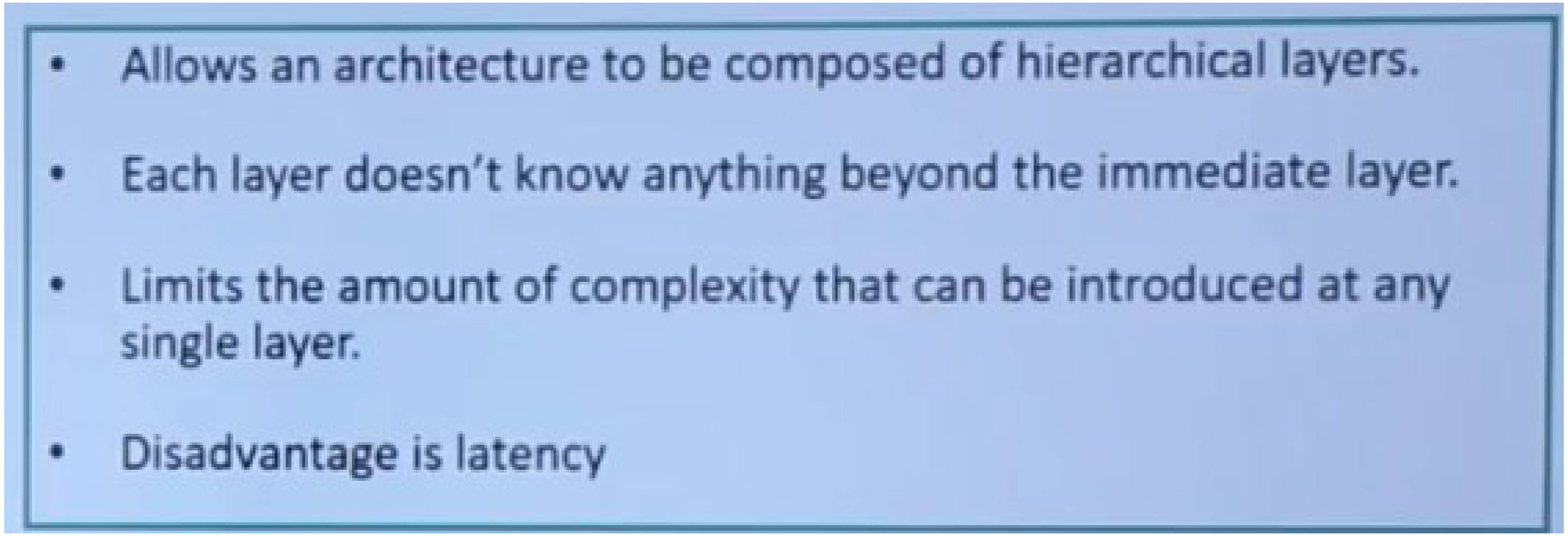
REST Constraints-Uniform Interface



- Uniform Interface is the key differentiator between REST & Non-REST APIs.
- There are 4 elements of Uniform Interface constraint.
    - Identification of Resources (typically by an URL).
    - Manipulation of Resources through representations.
    - Self-descriptive messages for each request.
    - HATEOS (Hypermedia As The Engine Of application State)
- Promotes generality as all components interact in the same way.

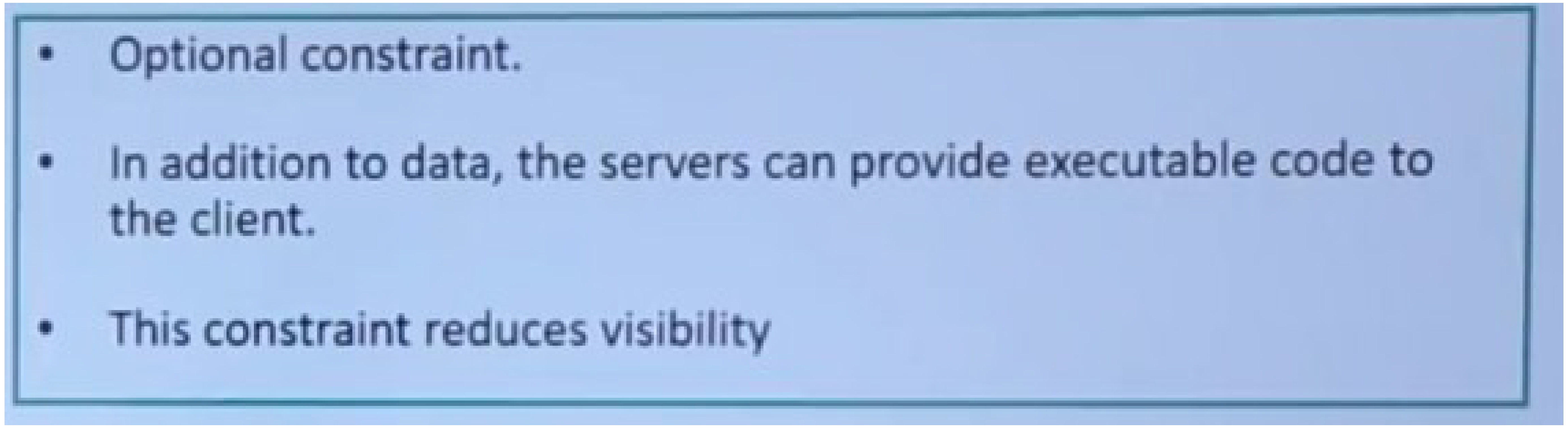
Busy Coder Academy

REST Constraints-Layered Arch



- Allows an architecture to be composed of hierarchical layers.
- Each layer doesn't know anything beyond the immediate layer.
- Limits the amount of complexity that can be introduced at any single layer.
- Disadvantage is latency

REST Constraints-Code on demand



- Optional constraint.
- In addition to data, the servers can provide executable code to the client.
- This constraint reduces visibility

Busy Coder Academy

HATEOAS

- *Used to discover locations & operations dynamically.*
- *Client doesn’t need to know URLs in advance.*
- *Response includes links with:*
- *href (target, mandatory)*
- *rel (relationship, mandatory e.g., "details", "payment")*
- *type (optional, content type)*
- *method (optional, HTTP method)*

JAX-RS

- *Java API for RESTful Web Services*

- *Java-based API & specification for REST services.*
-
- *Uses annotations to simplify development.*

- *Common Annotations*
- *@Path – Relative path of resource*
- *@GET – Fetch resource*
- *@POST – Create/update resource*
- *@DELETE – Delete resource*
- *@PUT – Create resource*
- *@HEAD – Check method availability*
- *@PathParam – Bind value from URI path*
- *@QueryParam – Bind value from query string*
- *@FormParam – Bind form value*
- *@CookieParam – Bind cookie value*
- *@HeaderParam – Bind HTTP header*

Implementations

- *Apache CXF*
- *Jersey (Reference Implementation by Sun/Oracle)*
- *RESTEasy (JBoss)*
- *Restlet*
- *WebSphere Application Server (IBM)*

HTTP Status Codes

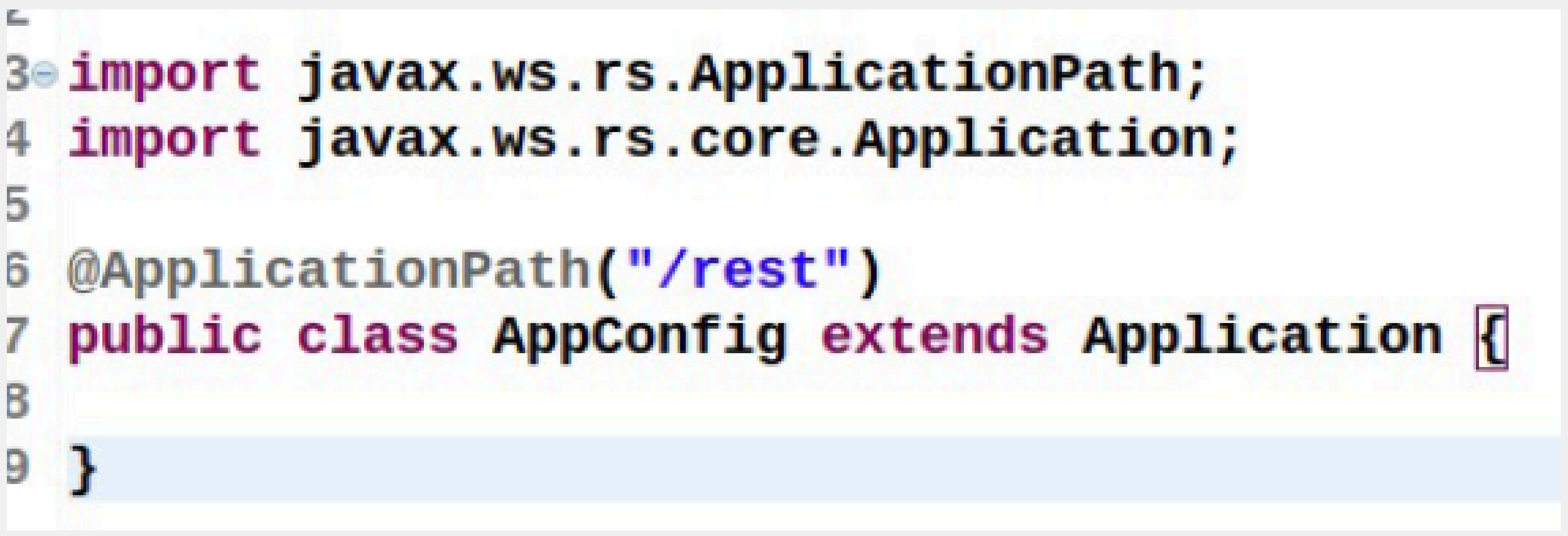
HTTP Status Codes

- **1xx** – Informational
- **2xx** – Success
- **3xx** – Redirection
- **4xx** – Client Error
- **5xx** – Server Error

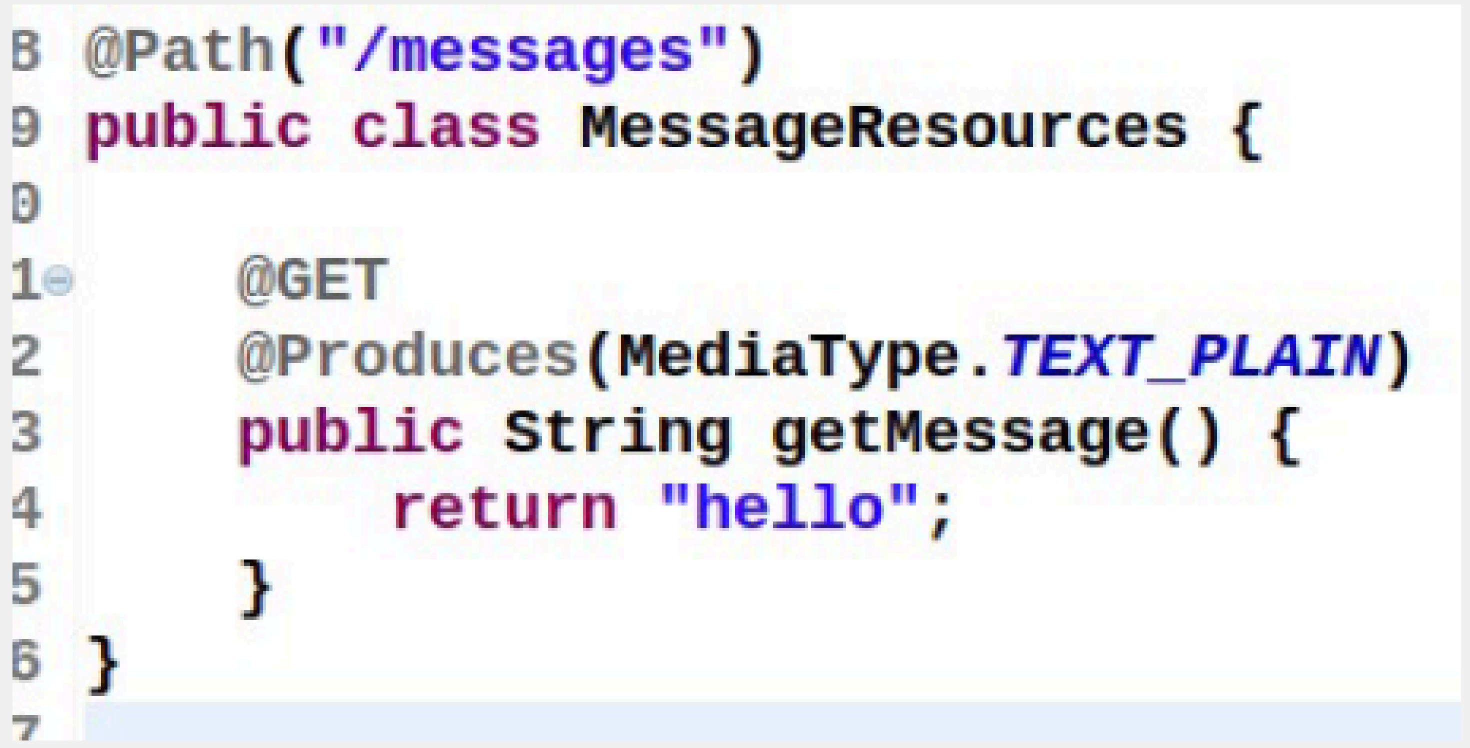
Common Codes

- **200 OK** – Success
- **201 Created** – Resource created
- **301 Moved Permanently** – URI changed
- **307 Temporary Redirect** – Redirect to another URI
- **308 Permanent Redirect** – Permanently moved
- **400 Bad Request** – Invalid syntax
- **403 Forbidden** – Access denied
- **404 Not Found** – Resource not found
- **500 Internal Server Error** – Unexpected server issue
- **503 Service Unavailable** – Server not ready
- **505 HTTP Version Not Supported**

JAX RS Hello World



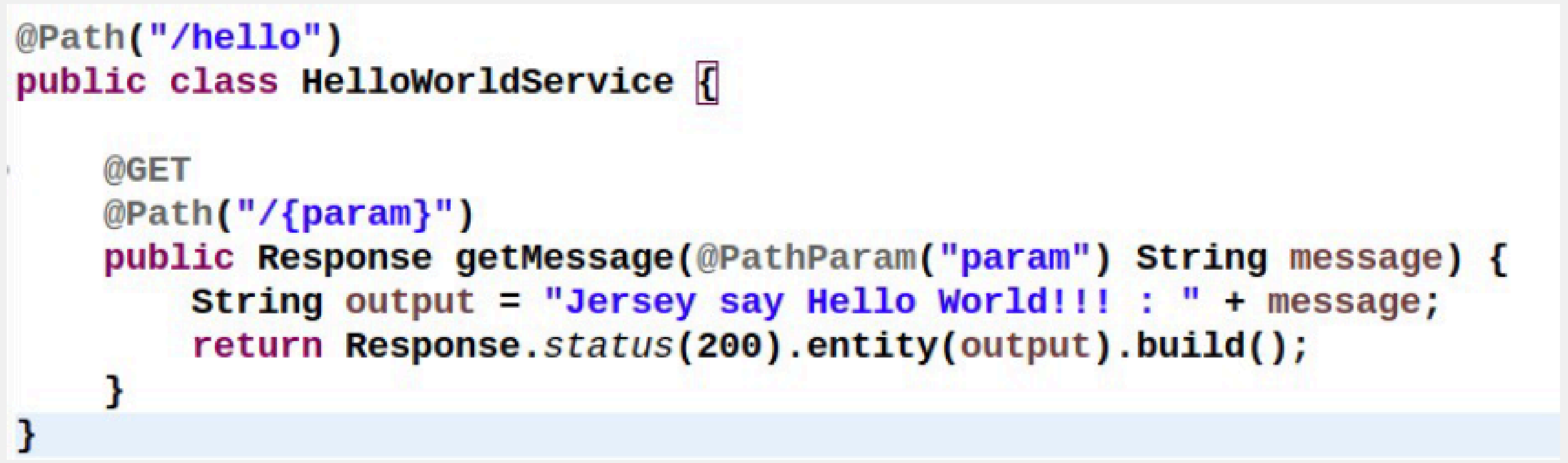
```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest")
public class AppConfig extends Application {

}
```



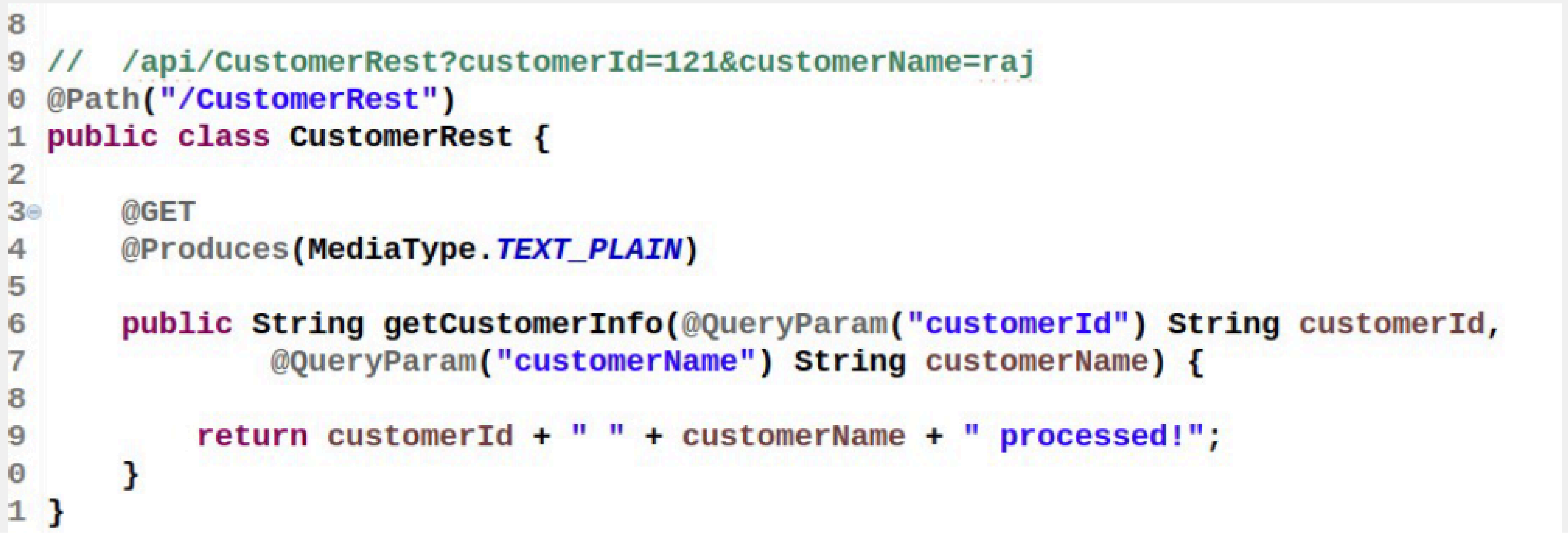
```
@Path("/messages")
public class MessageResources {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getMessage() {
        return "hello";
    }
}
```

Busy Coder Academy

JAX RS Example



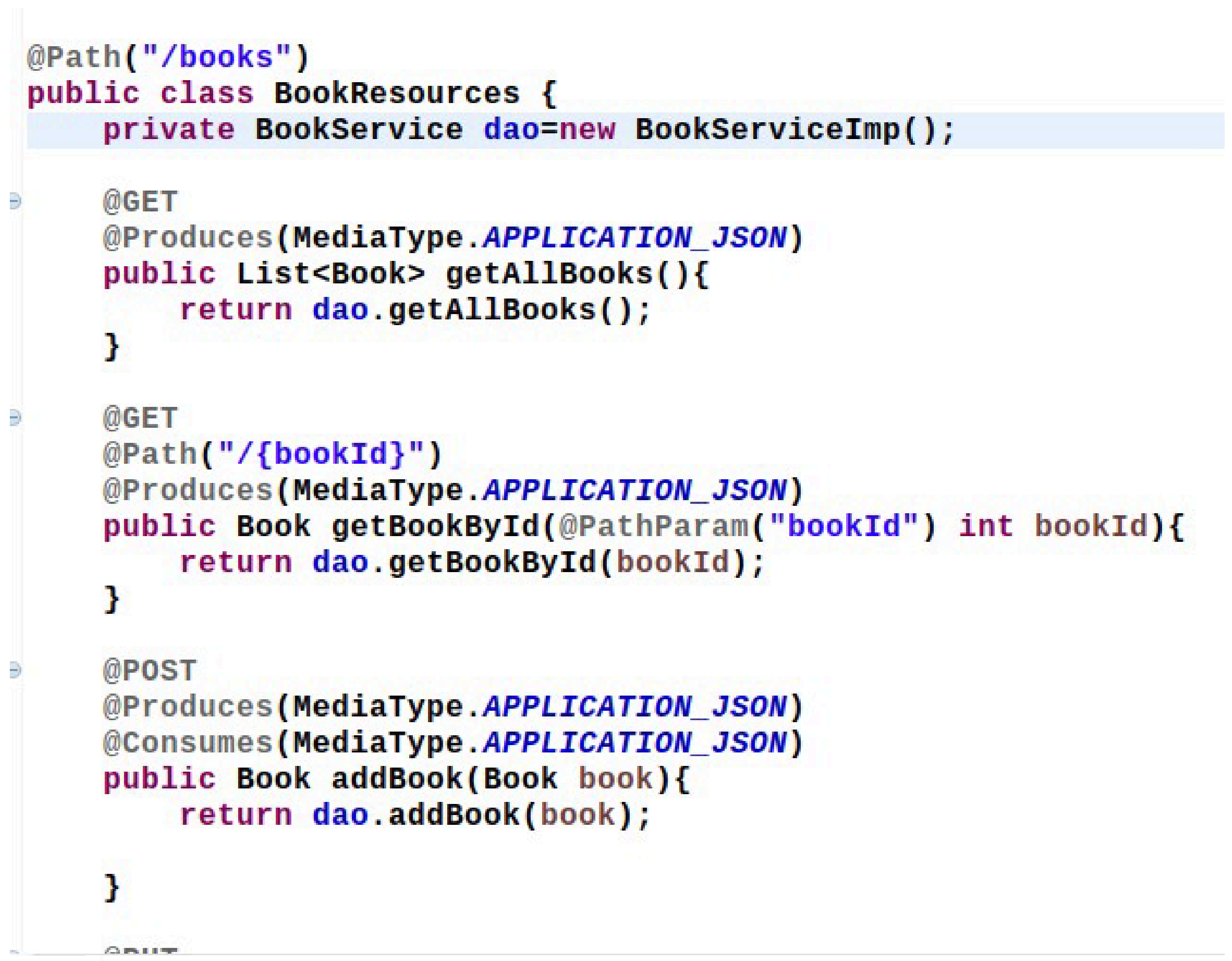
```java
@Path("/hello")
public class HelloWorldService {

    @GET
    @Path("/{param}")
    public Response getMessage(@PathParam("param") String message) {
        String output = "Jersey say Hello World!!! : " + message;
        return Response.status(200).entity(output).build();
    }
}
```



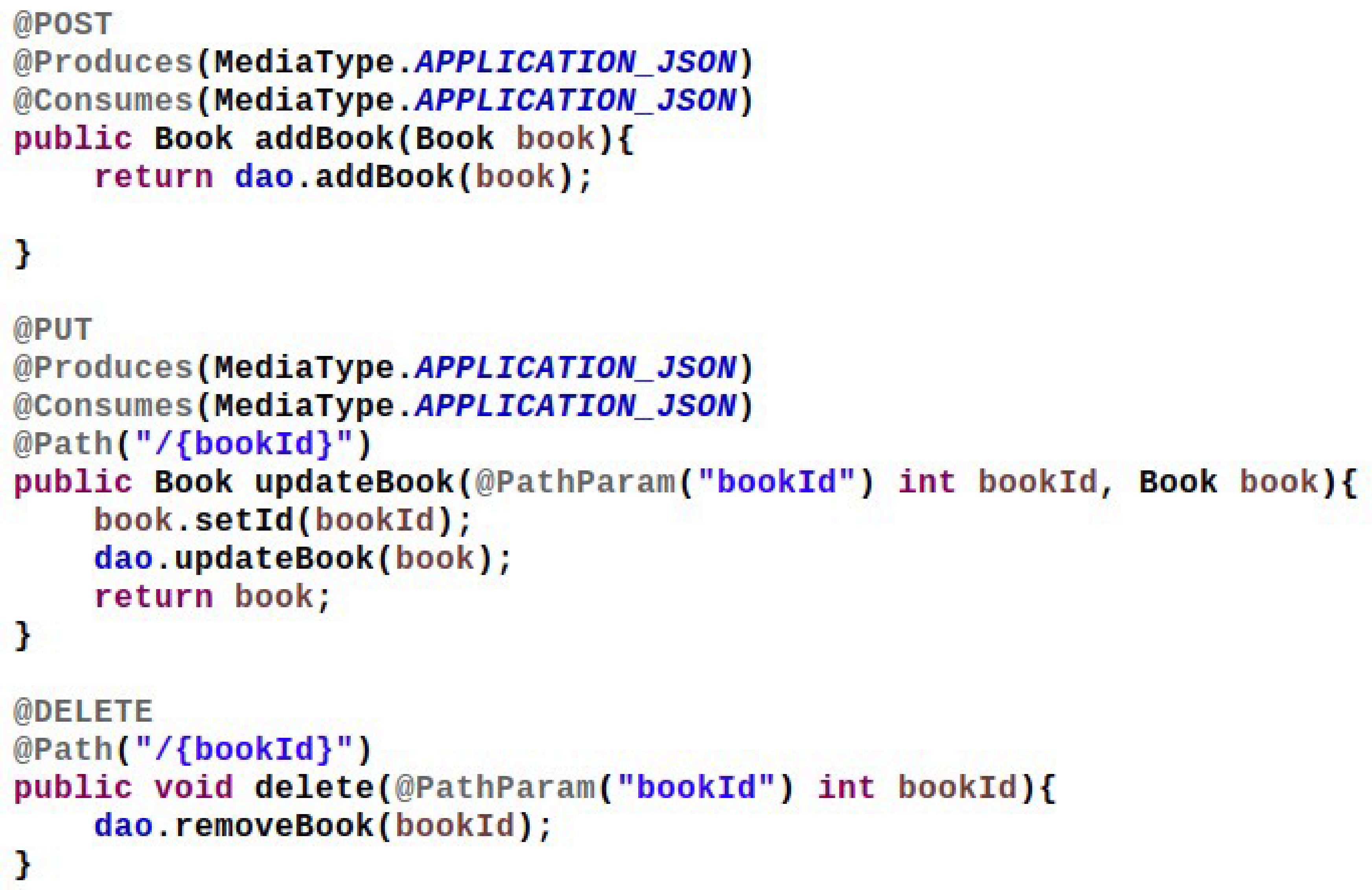
```java

//  /api/CustomerRest?customerId=121&customerName=raj
@Path("/CustomerRest")
public class CustomerRest {

    @GET
    @Produces(MediaType.TEXT_PLAIN)

    public String getCustomerInfo(@QueryParam("customerId") String customerId,
            @QueryParam("customerName") String customerName) {

        return customerId + " " + customerName + " processed!";
    }
}
```

Busy Coder Academy

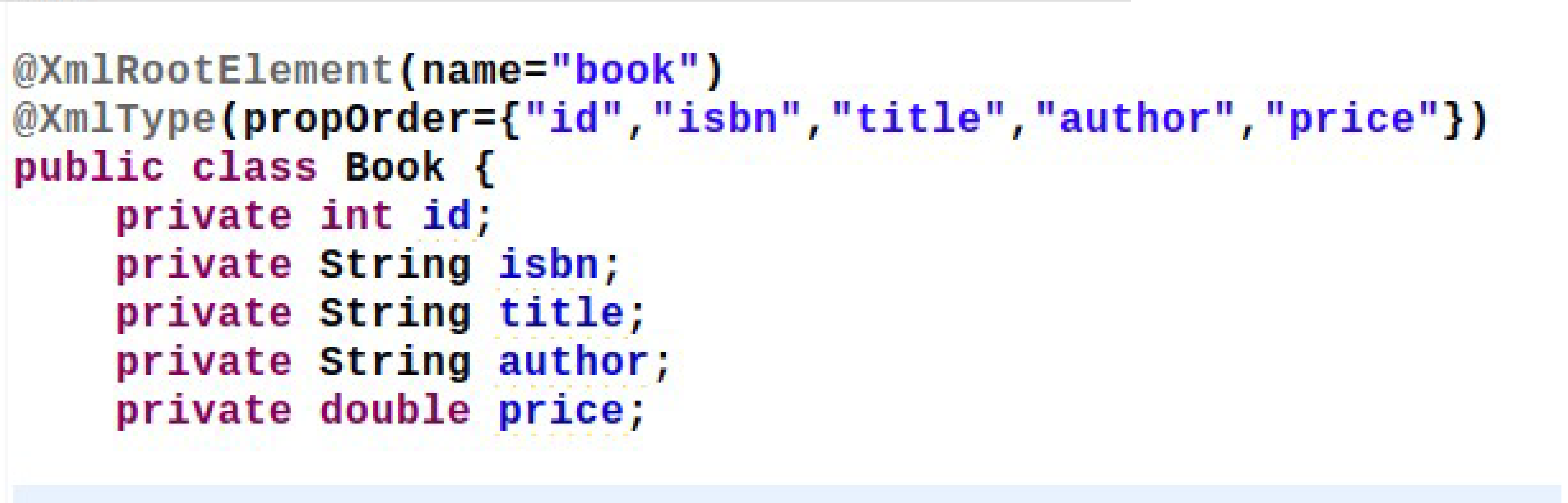
JAX RS Example



```java
@Path("/books")
public class BookResources {
    private BookService dao=new BookServiceImp();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Book> getAllBooks(){
        return dao.getAllBooks();
    }

    @GET
    @Path("/{bookId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Book getBookById(@PathParam("bookId") int bookId){
        return dao.getBookById(bookId);
    }

    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Book addBook(Book book){
        return dao.addBook(book);

    }

    @PUT
```



```java
@POST
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public Book addBook(Book book){
    return dao.addBook(book);

}

@PUT
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/{bookId}")
public Book updateBook(@PathParam("bookId") int bookId, Book book){
    book.setId(bookId);
    dao.updateBook(book);
    return book;
}

@DELETE
@Path("/{bookId}")
public void delete(@PathParam("bookId") int bookId){
    dao.removeBook(bookId);
}
```



```java
@XmlRootElement(name="book")
@XmlType(propOrder={"id","isbn","title","author","price"})
public class Book {
    private int id;
    private String isbn;
    private String title;
    private String author;
    private double price;
```

Busy Coder Academy