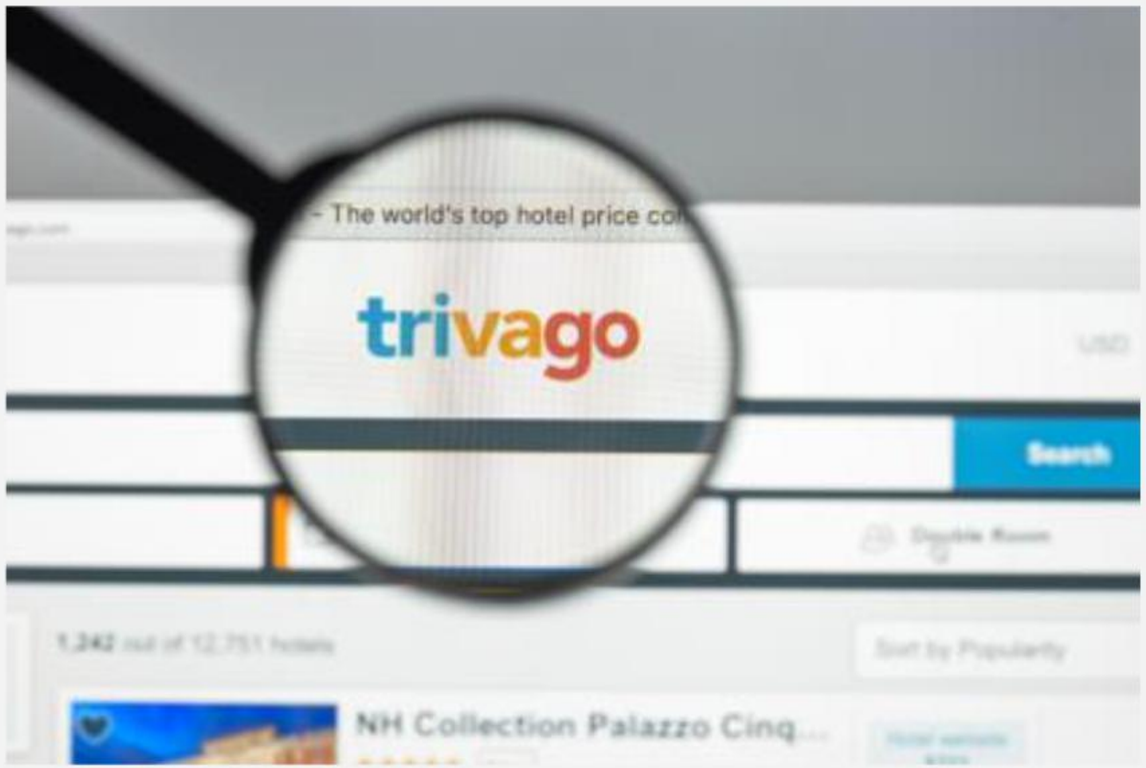


Time to Think

- The Trivago application can be used to book flights, hotels, buses, and cabs.
  - Where do you think the data for bookings come from?
  - Does it come from multiple vendors or just one vendor?
  - What is the format of the data?
  - Do you think the airline vendor gives the entire database structure to this application?
  - How do they communicate?

Slide Note

Think and Tell

- Will it be successful if a new airline enters the market and sells tickets only on its website but does not merge the data with third-party API vendors like Trivago? Or will they go out of business?
- Do the airline and third-party vendors have their applications on the same or different technology?
- The airline vendor application may be written in Java, and the third-party vendor application may be written in .NET. How does communication occur between them?

Slide Note

Think and Tell (Cont'd)

- The number of servers needs to be increased to handle requests from a huge number of clients.
- Clients do not need to know which server is serving the request.
- Servers also do not need to know from which client the request is received.
- How efficiently can a client's request be processed in such a complex environment?
- Is there an approach that will help to design a program that establishes communication between client and server without increasing the server's overhead and is also language agnostic?

Slide Note

Develop RESTful Services by Using Spring Boot by Using JPA

Slide Note

Learning Objectives

- Describe the REST API
- Explore applications of RESTful services
- Define components of RESTful services
- Implement layers of the RESTful API

Slide Note

Describe REST API

Menu

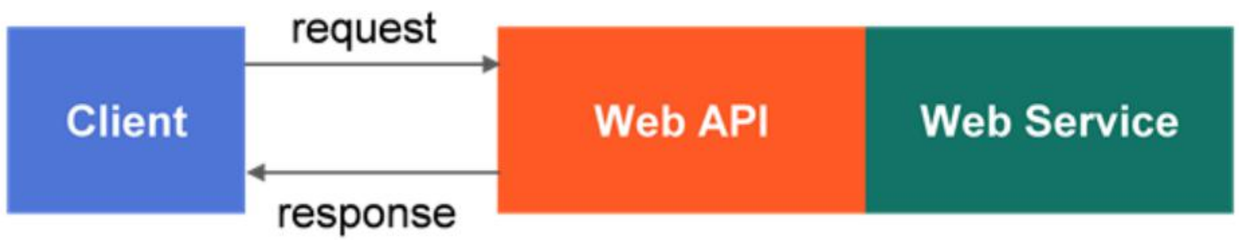
00:06:00:00:46

06/ 46

Slide Note

Web Applications and Services

- Nowadays, a client-server model is used to make web applications.
- The web application is software that runs on a web server to fulfill requests from the clients.
- Clients use application programming interfaces (APIs) to communicate with web services.
- APIs expose a set of data and functions to facilitate interactions between computer programs and allow them to exchange information.
- A Web API is the face of a web service, directly listening to and responding to client requests.



Slide Note

Design Criteria for Web API

- The API to be designed for client-server applications should:
  - Follow the HTTP protocol since it is stateless.
  - Accept a request from a client for required information.
  - Enable the server to send HTTP responses in a format that is language-independent.
- Is there a design pattern that will help designers design APIs that meet these requirements?

Slide Note

REST can be used to view or modify resources on the server without performing any operations on the server side.

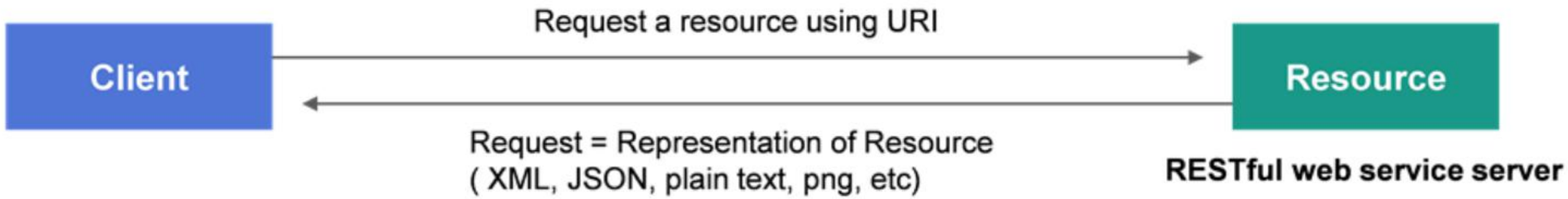
It works by identifying the two main entities: Client and Server.

A client sends the request to the web server, and the server sends back the response to the client.

The response is the representational state of the resource as it presents on the server.

RESTful Webservices

- The REST architectural style is commonly applied to the design of APIs for modern web services.
- A Web API conforming to the REST architectural style is a REST API.
- Having a REST API makes a web service RESTful.
- A REST API consists of an assembly of interlinked resources.



Slide Note

When a RESTful API is called, the server transfers the representation of the state of the requested resource to the client, usually in JSON (language-independent) text-based format.

REST – Representational State Transfer

- REST is an acronym for Representational State Transfer
- It is a design pattern or architectural style for designing web APIs.
- REST relies on HTTP, and therefore it is stateless.
- The client makes an HTTP request to a RESTful API for the requested information, referred to as a resource.
  - For e.x., products, orders, and shipping addresses are examples of resources on an online shopping website.
- At a given instance, the values of the resource determine its state.
  - For e.x., if a user has successfully logged in, their state of login property would be logged in, or else logged out.
- When a RESTful API is called, the server sends the request or data to the client, usually in JSON (language-independent) text-based format.
  - When a client requests the delivery status of an order, for example, the server sends the current state of the delivery along with track information.

Slide Note

Need of REST API

- Unlike other architectures, REST clearly defines the role of a client and a server.
- The user interface is separated from database services.
- Developers can focus on only one aspect when developing client-server applications.
- REST is independent of the underlying platform.
- Any platform such as PHP, Python, and Node.js can be used to implement a REST API.
- REST allows communication between the server and the client by using HTTP, regardless of the platform used..
- REST APIs are highly scalable and flexible.

Slide Note

Explore Applications of RESTful Services

Slide Note

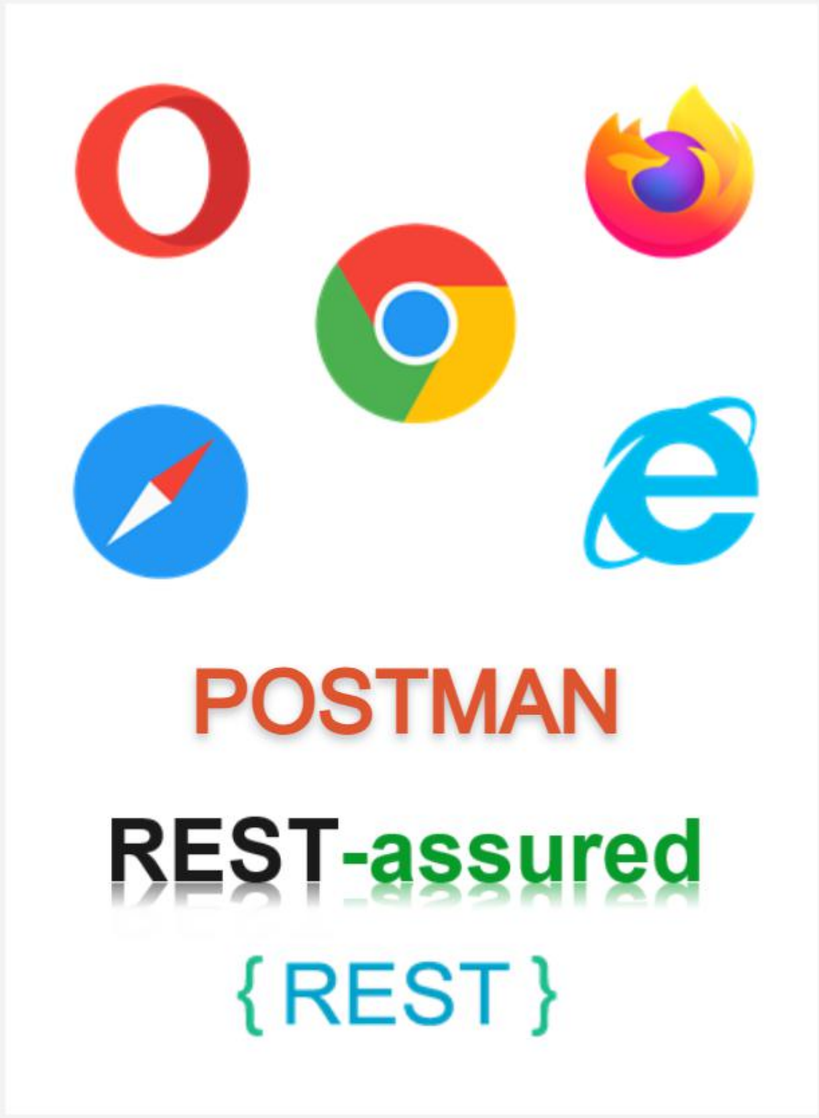
Applications of RESTful Services

- Tech giants like Twitter, YouTube, and Spotify owe their business growth to REST APIs.
- REST is used to improve the portability of an interface to other types of platforms.
- It increases the scalability of the projects.

Slide Note

REST Clients

- REST Client is a method or a tool to invoke a REST service API.
- REST APIs are exposed for communication by any system or service.
- For e.x., if an API is exposed to get real-time traffic information about a route from Google, the software/tool that invokes the Google traffic API is called the REST client.
- Opera, Google Chrome, Safari, Postman, Katalan, etc. are all examples of REST clients.



Slide Note

Define Components of RESTful Services

Slide Note

Explain the components in detail:

Resources: The main key component of a RESTful service is an asset. Let's think about how a web application on a server has records of a few representatives. Then let's accept that the URL of the web application is **http://demo.xyz.com**. Currently, to get to a worker record asset by means of REST, one can issue the directive, **http://demo.xyz.com/worker/1**. This request advises the webserver to give the information of the worker whose worker number is 1.

Request Verbs: These portray what you need to do with the asset. A program issues a GET action word to train the endpoint it needs to retrieve information. There are also several other different action words that are accessible, including POST, PUT, and DELETE. So, for **http://demo.xyz.com/worker/1**, the internet browser is issuing a GET request verb since it needs to get the details of the worker record.

Request Headers: These are extra guidelines sent with the request. These might characterize the kind of reaction required or the approval of the details.

Request Body: Data that is sent with the request. Information is regularly sent when a POST demand is made to the REST web administration. In a POST call, the customer tells the web administration that it needs to add an asset to the server. Thus, the request body gets the details of the asset, which is required to be added to the server.

Reaction Body: This is the fundamental body of the reaction. In our model, if we somehow managed to inquiry the web server through the request of **http://demo.xyz.com/worker/1**, the web server may restore an XML record with the details of the worker in the response body.

Response Status Codes: These codes are the general codes that are returned alongside the reaction from the web server. Our model is the code 200, which is typically returned if there is no mistake while restoring a reaction to the customer.

Note: All of the above links are for demonstration purposes, None of them are linked to an actual website nor are they functional.

Components of RESTful Services

Slide Note

Request to REST API – HTTP Request

- The HTTP request is made by a client to a server through the REST API to access a resource on the server.
- The components of an HTTP request are:
  - Endpoint: It's the URL of the REST server that enables access to a resource on the server.
  - Method: The kind of operation the client wants a server to perform on the resource.
    - GET: Get the resource from the server
    - POST: Create the resource on the server
    - PUT: Update the existing resource on the server
    - DELETE: Delete the existing resource on the server
  - Headers: Additional information provided for communication by the client to the server.
  - Data: Information that needs to be sent to the server.

Slide Note

Response From REST API – HTTP Response

- The response from the REST API is an HTTP response.
- The HTTP response is an object that contains the response that the server wants to send to the client.
- The response object contains:
  - Status: An HTTP Status code that indicates the status of the request
  - Status Message: A short description of the status code
  - Headers: Additional information for the response sent by the server to the client
  - Body: Optional, and if present, contains the fetched response in JSON format

Slide Note

Request and Response Flow

To request the resource **Products** from the server, the HTTP Request with **GET** method should be made to URL **http://eshop.com/products**

View products

CLIENT

REST API

SERVER

The HTTP Response received from the server will contain Products data in the response body with Status Code 200 and Status Message OK

Slide Note

HTTP Status Code

- HTTP response status codes indicate the status of the completion of the HTTP request.
- Responses are grouped into five classes:
  - 100 – 199: Informational responses
    - For e.x., the status code 102 indicates the server has received and is processing the request. The response has not yet been generated.
  - 200 – 299: Successful responses
    - For e.x., the status code 200 for a Get request indicates the request was successful in fetching the requested data.

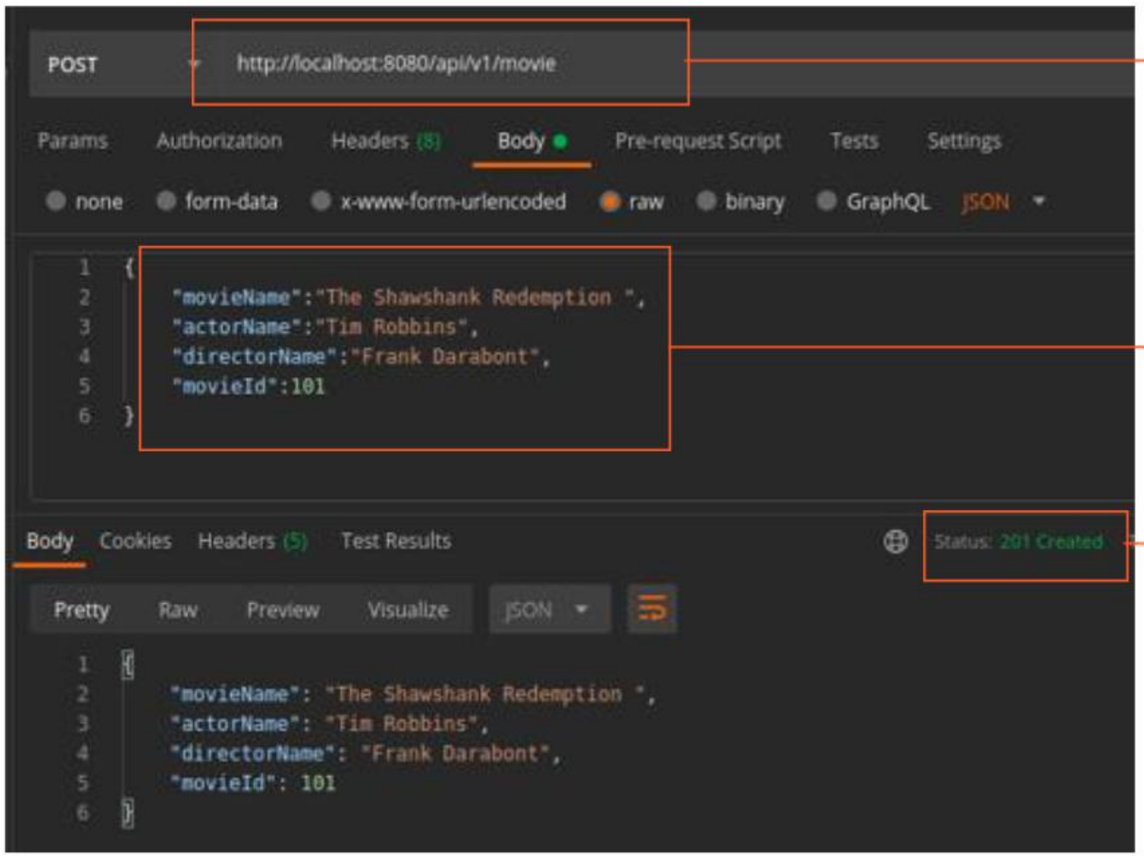
Slide Note

HTTP Status Code (Cont'd)

- 300 – 399: Redirection responses
  - For e.x., the status code 301 indicates the request has been changed permanently, and a new URL is given in the response.
- 400 – 499: Client errors
  - For e.x., the status code 404 indicates the server could not find the requested resource.
- 500 – 599: Server errors
  - For e.x., the status code 500 indicates an internal server error that the server does not know how to handle. For instance, a resource with a duplicate ID is sent to the server, and the server is not allowed to store a resource with a duplicate ID.

Slide Note

Request Headers and Body



URL

Request Body

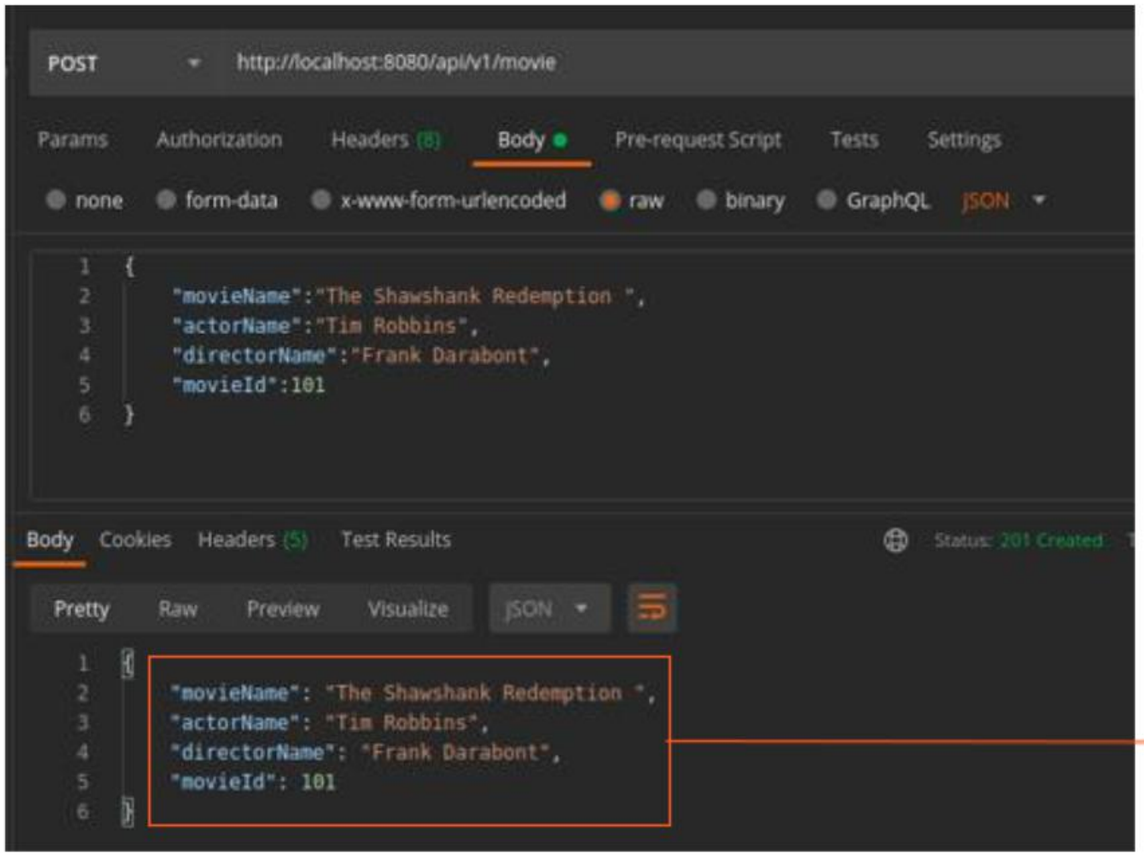
HTTP Status code

```
POST /api/v1/movie HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 134

{
    "movieName":"The Shawshank Redemption ",
    "actorName":"Tim Robbins",
    "directorName":"Frank Darabont",
    "movieId":101
}
```

Request Header

Slide Note

Response Headers and Body (Cont'd)



Response Header

Response Body

Menu

00:23:00:00:46

23/ 46

Slide Note

Quick Check

Which of the following HTTP Status codes means NOT FOUND?

1. 400
2. 401
3. 403
4. 404

Slide Note

Quick Check: Solution

Which of the following HTTP Status codes means NOT FOUND?

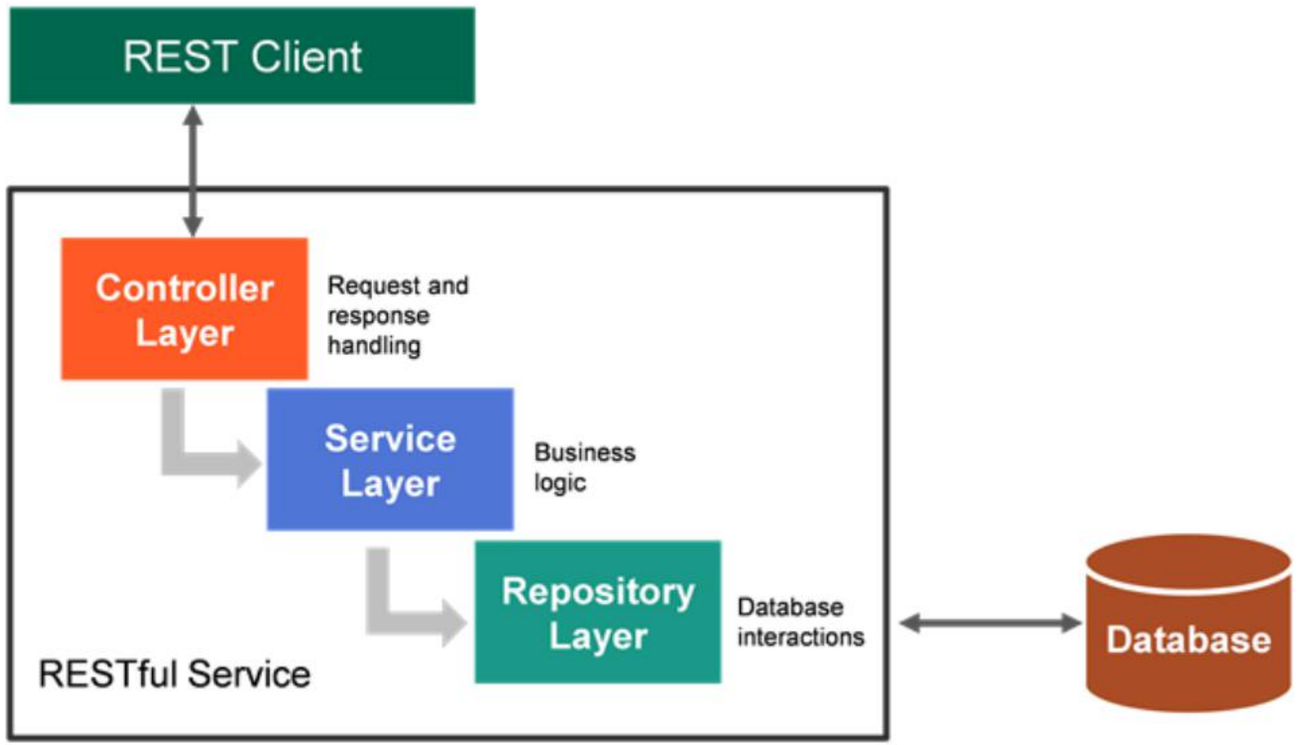
1. 400
2. 401
3. 403
4. 404

Slide Note

Implement Layers of the RESTful API

Slide Note

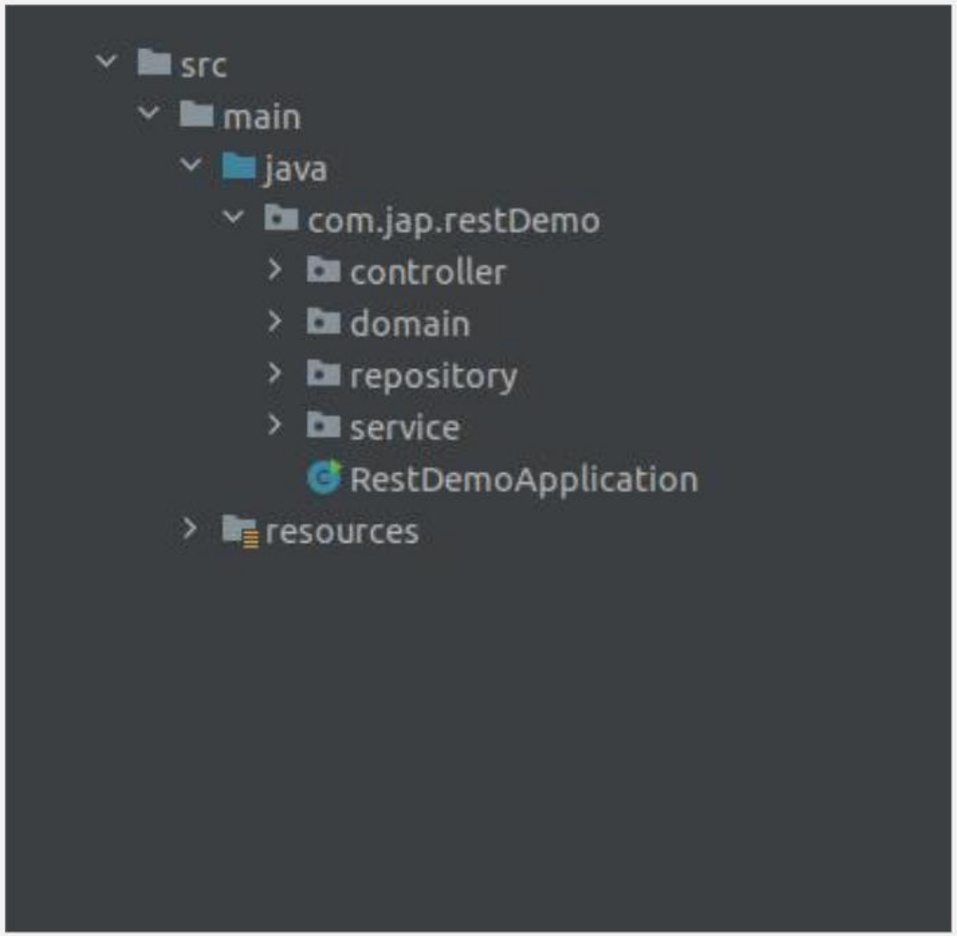
RESTful Layers



A RESTful service is typically composed of three layers:

- Controller layer is responsible for handling requests and sending back a proper response.
- Service layer holds the business logic of the service – for example, tax calculation, bill generation, etc.
- Repository layer, also known as D-A-O (Data Access Object) layer, is responsible for interacting with the database and performing operations on the data.

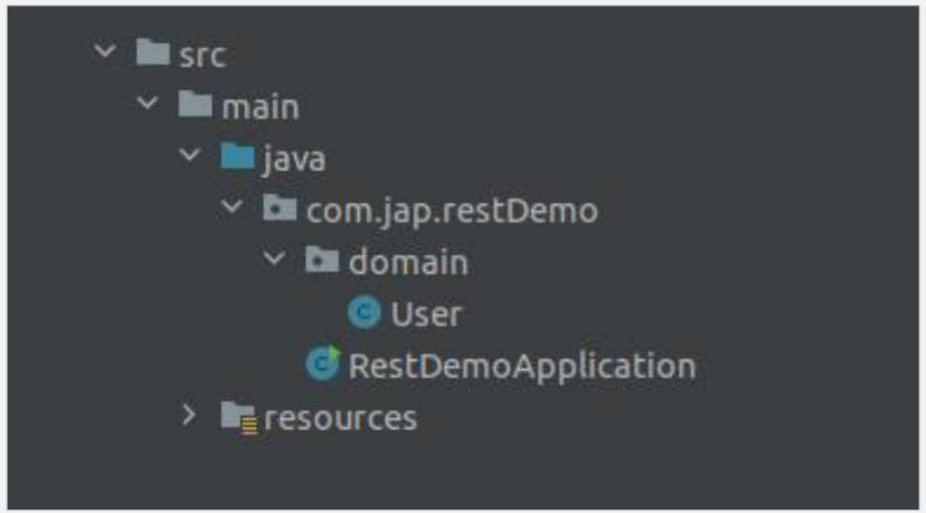
Slide Note



RESTful Layers (cont'd)

- Here, `controller`, `domain`, `repository`, and `service` are user-defined packages created inside the root package, i.e., `com.jap.restDemo`.
- As per standard, controller, service, and repository should be the package names for the controller, service, and repository layers.
- The controller package will have the `controller` class to handle the request and send the response.
- The service package will have a `service` class containing the business logic.
- The repository package will have the `repository` class with all DAO information.
- The `Domain` classes are POJO (Plain Old Java Object) classes that contain the attributes of each object.

Slide Note

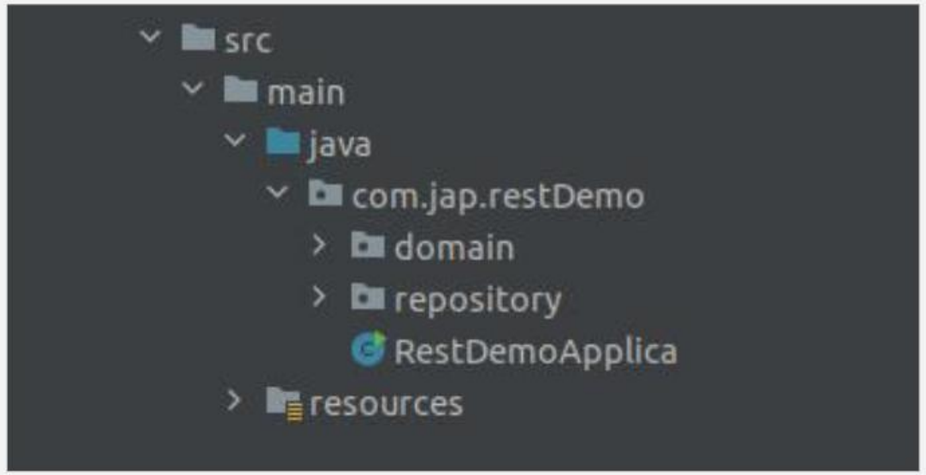



```
@Entity
public class User {

    private String name;
    @Id
    private int userId;
     //getter and setters
}
```

Domain Class

- The objects of the application are defined in the domain classes (For e.x. - `User, Product, Movie`).
- The Domain classes are POJO (Plain Old Java Object) classes that contain the attributes of each object.
- These are persisted in the database and are also called Entity classes.
- Represented using the `@Entity` annotation.
- All transactions are done on this object.
- `@Id` for userId means that this is going to be the primary key in the table.

Slide Note




```
@Repository
public interface UserRepository extends CrudRepository<User,Integer> {
}
```

Repository Layer

- The repository layer is where the data access logic is written.
- The `@Repository` annotation marks the interface as a repository, although it's not mandatory, but it's good practice.
- A repository is a mechanism for encapsulating storage, retrieval, and search behavior for objects.
- This interface extends `CrudRepository`.

Note – CrudRepository will be discussed in more detail in the next Sprint.

Slide Note

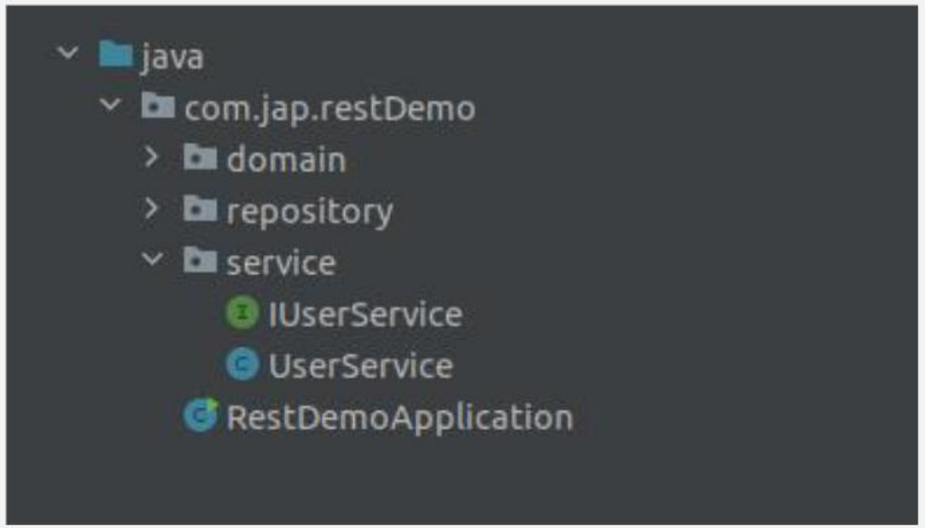
Repository Layer (cont'd)

- There are two generic parameters of the `CrudRepository` that must be mentioned mandatorily:
  - The first is the domain class, i.e., the class annotated with `@Entity`.
  - The second parameter is the datatype of the attribute that is marked with the `@Id` annotation in the domain or entity class.
- At runtime, a `UserRepository` implementation class will be created by Spring Data JPA that will implement the `UserRepository` interface.
- Since `UserRepository` implements the `CrudRepository`, the implementation for the CRUD methods will be automatically provided by Spring Data JPA.
- Therefore, you do not need to write an implementation class for the UserRepository interface.

```
@Repository
public interface UserRepository extends CrudRepository<User,Integer> {
}
```

Note – Spring Data JPA will be discussed in the next Sprint

Slide Note




```java
public interface IUserService {

    User saveUser(User user);
    List<User> getAllUsers();

}
```

Service Layer

- The service layer contains the business logic.
- This package has two Java files: one is the interface and the other is its implementation class.
- The Interface defines what functionalities are to be provided.
- Here, the `saveUser()` methods will save user data in the database.
- The `getAllUsers()` will retrieve data from the database.

Slide Note

```java
@Service
public class UserService implements IUserService{

    private UserRepository userRepository;
    @Autowired
    UserService(UserRepository userRepository){
        this.userRepository = userRepository;
    }
    @Override
    public User saveUser(User user) {
        return userRepository.save(user);
    }
    @Override
    public List<User> getAllUsers() {
        return (List<User>) userRepository.findAll();
    }
}
```

Service Class

This layer is annotated with the @Service annotation. During the component scanning, Spring looks at this annotation and treats it as a Spring Container managed Object.

- If a new user is registered, the information must be saved in the database, and the service layer utilizes the methods of the Repository layer to perform the save functionality.
- There is no code written by the programmer to explicitly save the user object in the database, but Spring Data JPA provides the functionality implicitly when the Repository layer extends the CRUDRepository.

Slide Note

Service Class (cont'd)

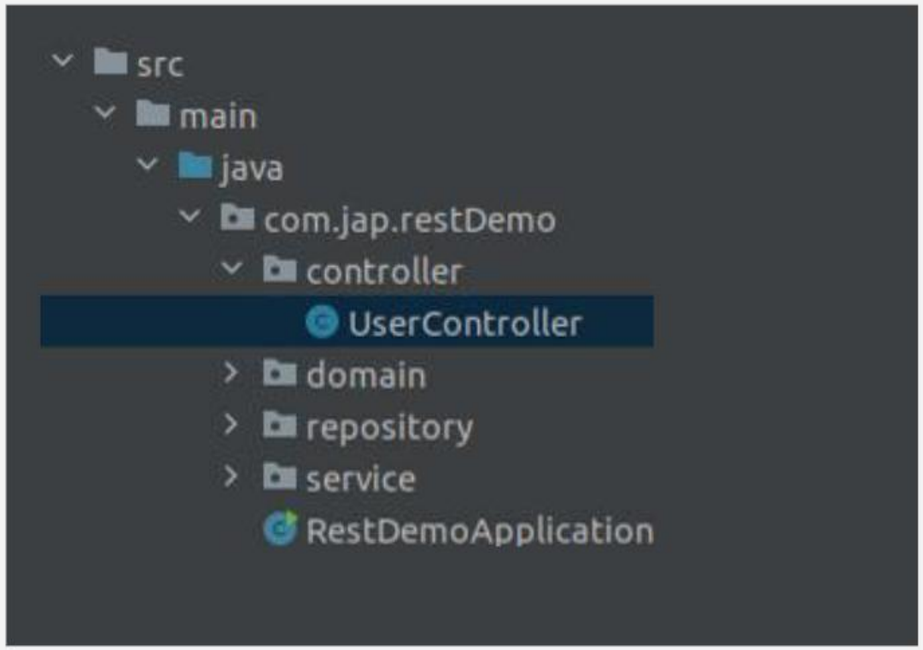
```java
@Service
public class UserService implements IUserService{

    private UserRepository userRepository;
    @Autowired
    UserService(UserRepository userRepository){
        this.userRepository = userRepository;
    }
    @Override
    public User saveUser(User user) {
        return userRepository.save(user);
    }
    @Override
    public List<User> getAllUsers() {
        return (List<User>) userRepository.findAll();
    }
}
```

The Service layer utilizes the methods of the repository layer to get data from the database.

The object of the repository layer must be autowired in the service layer.

Here, the `save()` and `findAll()` methods are of CrudRepository

Slide Note




```
@RestController
public class UserController {

}
```

Controller Layer

- This Controller layer communicates with the client.
- It is responsible for handling user requests and returning the response.
- The Controller is used for making restful web services with the help of the @`RestController` annotation.
- The @`RestController` annotation is responsible for returning the response.
- The response is returned as a JSON object.

Slide Note

```java
@RestController
@RequestMapping("/api/v1")
public class UserController {

    private UserService userService;
    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }
    @PostMapping("/user")
    public ResponseEntity<?> saveUser(@RequestBody User user){
        User user1 = userService.saveUser(user);
            return new ResponseEntity<>(user1, HttpStatus.CREATED);
    }
    @GetMapping("/users")
    public ResponseEntity<?> getAllUsers() {
        List list = userService.getAllUsers();
        return new ResponseEntity<>(list, HttpStatus.OK);
    }}
```

Controller Layer (cont'd)

- The Controller layer utilizes the methods of the Service layer to generate a response to send back to the client.
- Therefore, the object of the service layer must be `autowired` in the controller layer.
- The `@RequestMapping` annotation is used to define the Request URI to access the REST Endpoints
- The `@RequestMapping` annotation provides a standard path, api/v1, at the class level.
  - api – stands for API's
  - V1 – stands for version 1
- The controller methods `saveUser`() and `getAllUsers`() are called request handler methods for respective POST and GET requests.

Slide Note

Controller Layer (cont'd)

- The `@PostMapping` and `@GetMapping` annotation are annotations for mapping HTTP POST and GET requests onto Specific handler methods.
- A `ResponseEntity` object is returned from the handler methods of the controller.
- In Spring REST, a `ResponseEntity` represents a complete HTTP response and includes the headers, the response body, and the status code.
- The `@RequestBody` annotation is used to define the request body content type.
- The `@RequestBody` annotation is the parameter for the handler methods.
- `@ResponseBody` annotations are used to bind the HTTP request/response body with a domain object in the method parameter.

```java
@RestController
@RequestMapping("/api/v1")
public class UserController {

    private UserService userService;
    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }
    @PostMapping("/user")
    public ResponseEntity<?> saveUser(@RequestBody User user){
        User user1 = userService.saveUser(user);
            return new ResponseEntity<>(user1, HttpStatus.CREATED);
    }
    @GetMapping("/users")
    public ResponseEntity<?> getAllUsers() {
        List list = userService.getAllUsers();
        return new ResponseEntity<>(list, HttpStatus.OK);
    }}
```

Slide Note

The application.properties File

- In this application, you are using the H2 database. The H2 database is an in-memory database.
- H2 database is very fast, open source, and uses the JDBC API.
- It has a browser-based console application.
- There is no configuration information required, yet the application works.
- This is the magic of Spring Boot autoconfiguration, i.e., `@SpringBootApplication`.
- On running the application, Spring Boot finds the H2 dependency in the pom.xml and generates the default H2 configurations to use in the H2 embedded database.
- Only add one line in the application.properties to enable the H2 database:
  `spring.h2.console.enabled=true`.

Slide Note

Dependency for the Project

- `spring-boot-starter-data-jpa` **dependency connects Spring application with relational databases efficiently.**
  - **This dependency internally uses JDBC.**
- `spring-boot-starter-web` **dependency is a starter for building web applications, including RESTful applications. It uses Tomcat as the default embedded container.**
- `H2` **database dependency is added to work with embedded or in-memory database H2.**
- **Running this application will run on an inbuilt Tomcat server. By default, Tomcat server runs on port 8080.**

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
</dependencies>
```

Slide Note

Postman - REST Client

- Postman is an HTTP client used for testing web services.
- It can be used to build, test, and share REST API calls.
- It's easy to use and has an intuitive interface that helps us access most of its features with just one click.
- It stores the API calls in the history, saving time and avoiding retaking API calls.
- It allows us to test our REST API without writing the code for testing or using any other additional setup.
- Install Postman.

Slide Note

GET and POST Mapping Using the H2 Database

User details like name and userId need to be saved for any application that we create.

Let us take an example of a user entity and see how to save the user object in a database and list all the users present in the system.

Run the application by writing in the terminal the command -

```
mvn spring-boot:run
```

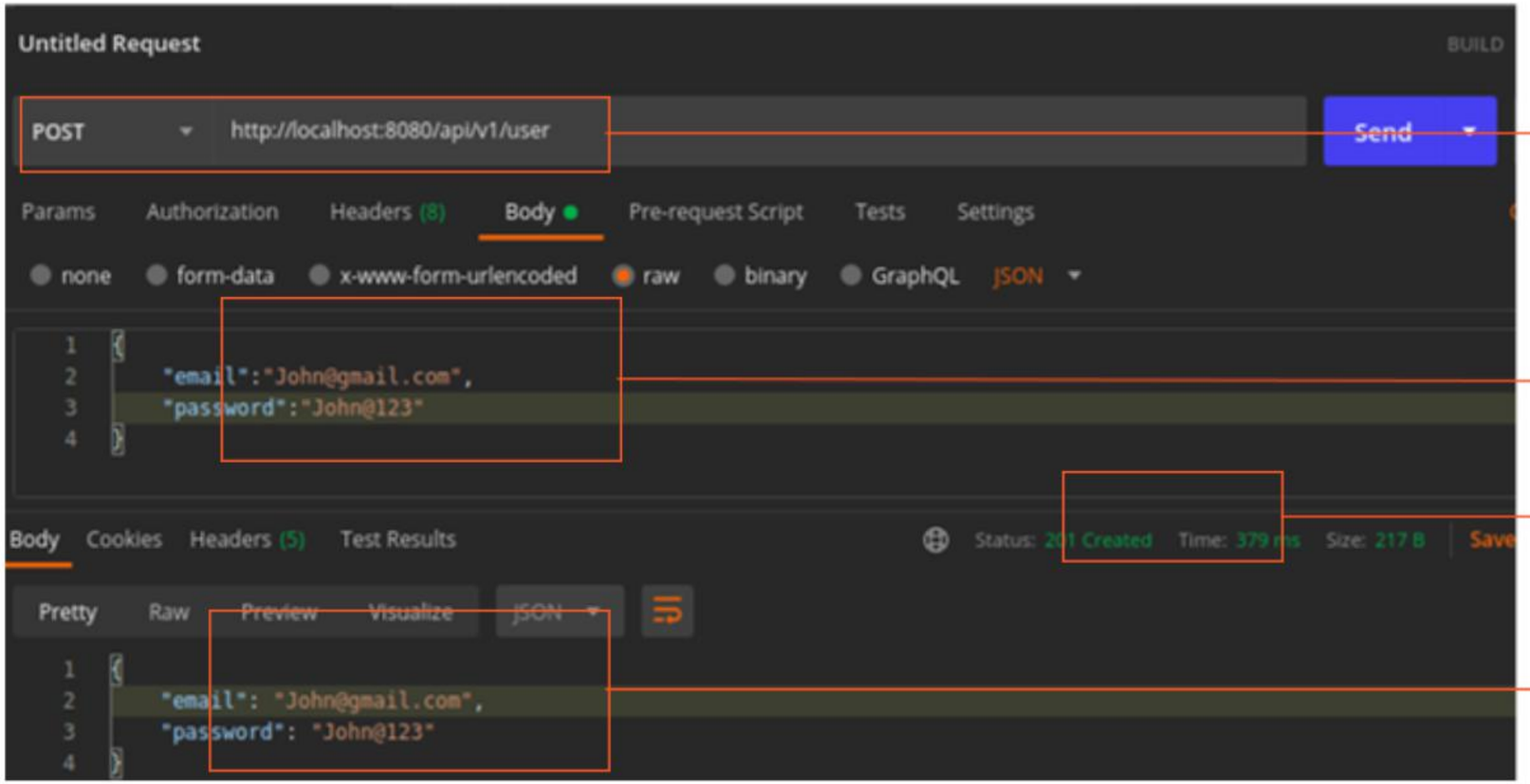
Click here for the solution.

DEMO

Slide Note

This URL should map with the @Postmapping of the handler method in the controller.

POSTMAN Output – Save the User



This URL should map with @Postmapping of the handler method in the controller

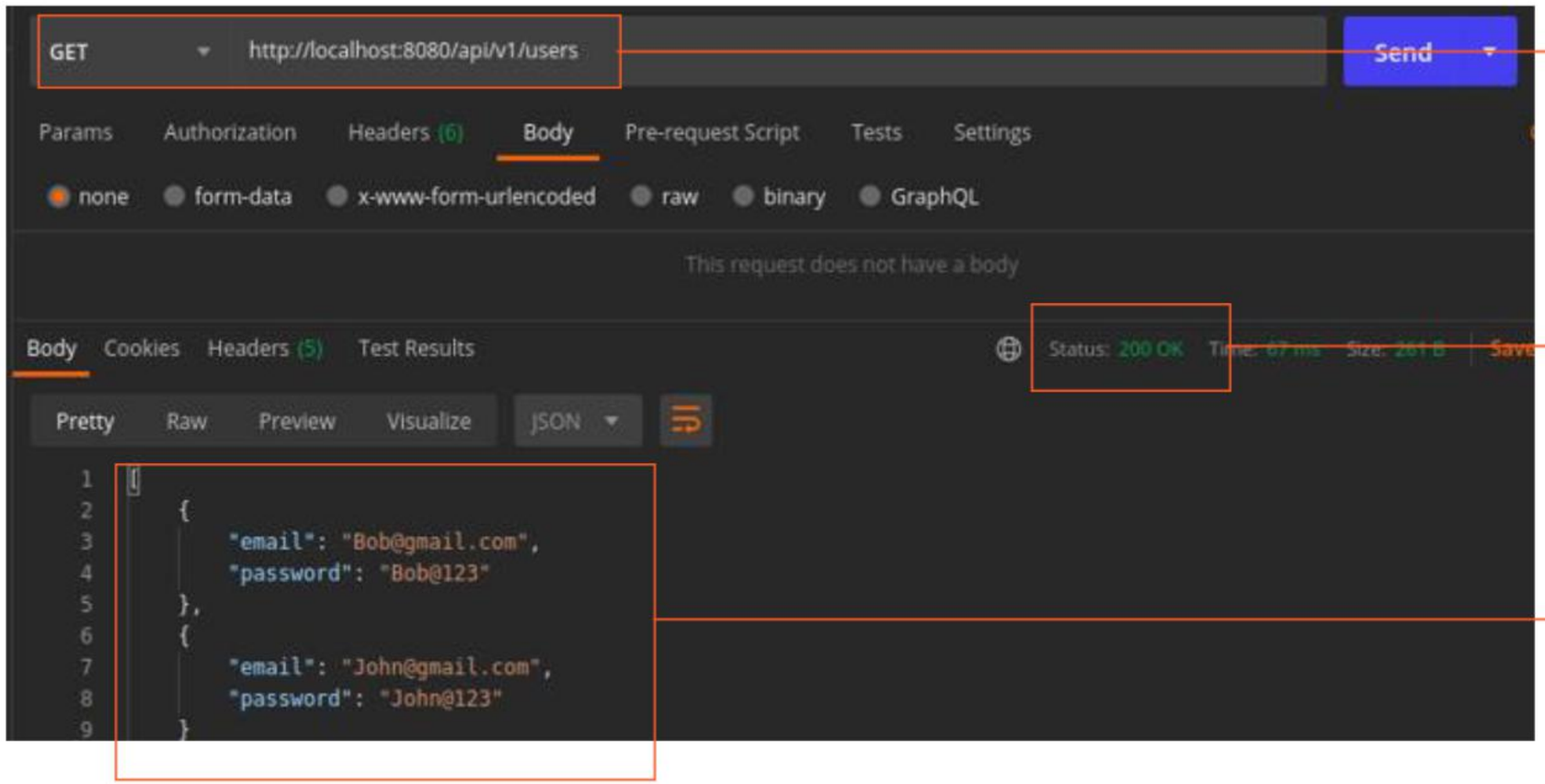
Request Body

HTTP status code

Response Body

Slide Note

POSTMAN Output - GetAll User



This URL should map with @Getmapping of the handler method in the controller

HTTP status code

Response Body

Slide Note

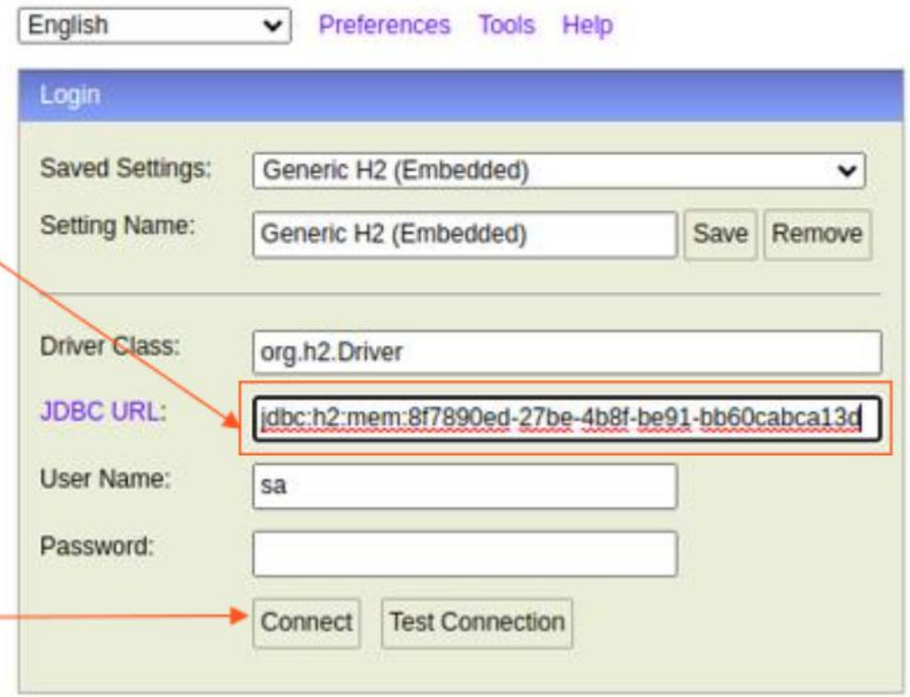
The H2 Console

- If you want to have a look at the tables created, the H2 console can be accessed at localhost:8080/h2-console,
- To establish a connection with the database, Spring Boot itself creates a path.

main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:8f7890ed-27be-4b8f-be91-bb60cabca13d'

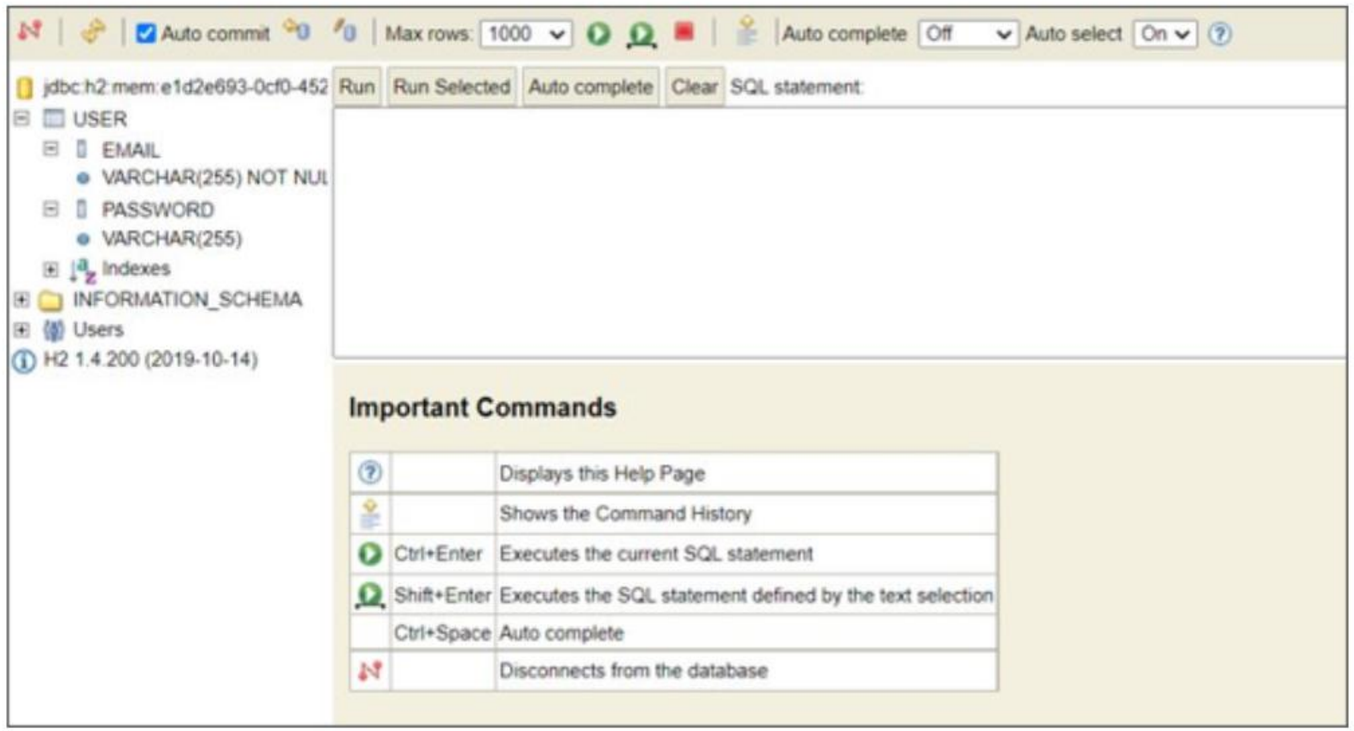
Paste the database available into the H2 console

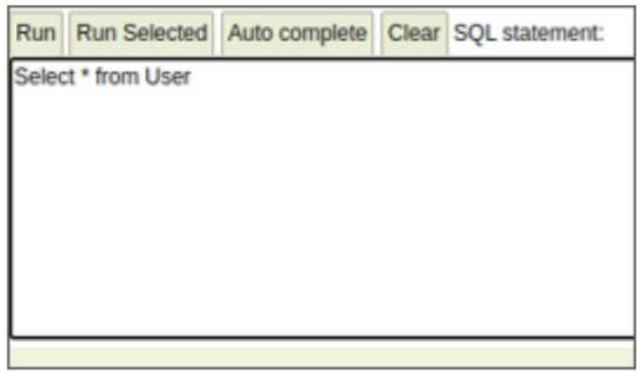
Click on the Connect button



Slide Note

The H2 Console (Cont'd)





Write the Select Query and Click on the Run button.

EMAIL	PASSWORD
John@gmail.com	John@123
Bob@gmail.com	Bob@123

(2 rows, 2 ms)

Menu

00:45:00:00:46 45/ 46