1. **What is REST?**

   REST stands for Representational State Transfer. It transfers state(data) in global format(representational) between two different applications running on different servers.

   In this process of transfer of state(Data),

   The data requestee is called Consumer/ Client. The data provider is called Producer application.

   REST is an architectural style which follows set of rules to create a webservice.

   Webservices provide reusable data to multiple applications and interoperability between them on the internet.

   Web services that conform to the REST architectural style, called RESTful Web Services.

2. **How will you define a development of REST API?**

   RestController is a mandatory class.

   It acts as a front-end controller(client code).

   It contains several methods that return BODY and STATUS as a ResponseEntity object.

   BODY refers to data in the form of String, Object, Collections etc.

   STATUS refers to the HttpResponse Status (200,404,405,500 etc)

3. **Project Structure**

   Project structure:

Package/ location	Class/ Interface/ file name	Purpose
Src/main/resources	Application.properties	Properties file to declare common properties in the project
Com.spring.restWebServicesAPI.entity	Invoice.java	Model/Entity class with Database table mapping
Com.spring.restWebServicesAPI.repo	InvoiceRepository.java	Repository interface which extends JpaRepository interface

Com.spring.restWebServceesAPI.service	IInvoiceService.java	Serviceinterface with database related methods
Com.spring.restWebServicesAPI.impl	InvoiceServiceImpl.java	Service class contains implementations of methods declared in service interface for database related operations
Com.spring.restWebServicesAPI.error	ErrorType.java	Helper class used in InvoiceErrorHandler.java
Com.spring.restWebServicesAPI.exception	InvoiceNotFoundException.java	To define custom exception if any invoice not found
Com.spring.restWebServicesAPI.handler	InvoiceErrorHandler.java	To handle the error in case InvoiceNotFoundException is thrown from any controller
Com.spring.restWebServicesAPI.utility	InvoiceUtil.java	Utility class to maximize code reusability and minimize code redundancy
Com.spring.restWebServicesAPI.controller	InvoiceRestController.java	Rest Controller with CRUD. It accepts all requests coming from client and handover to respective methods for processing.

Project Details:

1. Application.properties

```
#  DB Connection Properties
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/REST_INVOICE_API
spring.datasource.username=root
spring.datasource.password=root

# JPA Properites
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

Data definition is –> ddl-auto = update.

2. Entity class is Invoice. It contains the fields which map to columns in table.
There is a primary key, auto-incremented.

```java
package com.spring.restWebServiceAPI.entity;

import jakarta.annotation.Nonnull;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;

@Entity
@Table(name="invoices")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Invoice {

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name="invoice_id")
    private Long id;

    @Column(name="payee_name")
    private String payee;

    @Column(name="receiver_name")
    private String receiver;

    @Column(name="amount")
    private Double amount;

}
```

3. To enable the data transfer between server and client. We have, DTO class which accepts information in string data-type for entity/ table Invoice.

```
package com.spring.restWebServiceAPI.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class InvoiceDTO {

    private String invoice_id;
    private String payee_name;
    private String receiver_name;
    private String invoice_amount;

}
```

4. To enable the mapping of DTO to Entity and vice versa and to map the fields by name for source and target, we use mapstruct mapper.
   Mapstruct mapper is used to avoid exposing the entity class and database information to client code.
   In Business implementation class we make use of mapper to segregate the entity layer and db layer information from client side.
   NOTE: Mapper is injected in service implementation class as spring bean using parameter 'componentModel' with value = 'spring' along with @Mapper.
   As interfaces and abstract classes don't qualify to become spring beans by default.

```
package com.spring.restWebServiceAPI.mapper;

import com.spring.restWebServiceAPI.dto.InvoiceDTO;
import com.spring.restWebServiceAPI.entity.Invoice;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.factory.Mappers;
import org.springframework.stereotype.Component;

@Mapper(componentModel = "spring")
public interface InvoiceMapper {

    InvoiceMapper MAPPER = Mappers.getMapper(InvoiceMapper.class);

    @Mapping(source = "id", target = "invoice_id")
    @Mapping(source = "payee", target="payee_name")
    @Mapping(source="receiver", target="receiver_name")
    @Mapping(source="amount", target="invoice_amount")
    InvoiceDTO mapToInvoiceDTO(Invoice invoice);
    @Mapping(source = "invoice_id", target = "id")
    @Mapping(source = "payee_name", target="payee")
    @Mapping(source="receiver_name", target="receiver")
    @Mapping(source="invoice_amount", target="amount")
    Invoice mapToInvoiceEntity(InvoiceDTO invoiceDTO);
}
```

5. Custom exception:
   a. An helper class to define the fields for custom exception created. ErrorType.java

```
package com.spring.restWebServiceAPI.error;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorType {

    private String time;
    private String status;
    private String message;
}
```

   b. The class to define the exception. InvoiceNotFoundException.java

```
package com.spring.restWebServiceAPI.exception;

public class InvoiceNotFoundException extends RuntimeException{

    private static final long serialVersionUID = 1L;

    public InvoiceNotFoundException(){}

    public InvoiceNotFoundException(String
message){super(message);}
}
```

   c. A @RestControllerAdvice annotated class as an advice which defines the structure of the InvoiceNotFoundException. i.e., return type, and clubs InvoiceNotFoundException to it using @ExceptonHander so it can be used globally in the project. InvoiceErrorHandler.java
   This enables us to have a dignified return type in global format(responseentity object) and this way InvoiceNotFoundException can be used in any controller method.
   The literal for response entity is helper class, ErrorType and response contains value for the described fields in that class, along with HttpStatus

```
package com.spring.restWebServiceAPI.handler;

import com.spring.restWebServiceAPI.error.ErrorType;
import com.spring.restWebServiceAPI.exception.InvoiceNotFoundException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.util.Date;

@RestControllerAdvice
public class InvoiceErrorHandler {

    /**
     * In case of InvoiceNotFoundException is thrown
     * from any controller method, this logic gets
     * executed which behaves like re-usable and
     * clear code (Code Modularity)
     * @param nfe
     * @return ResponseEntity
     */
    @ExceptionHandler(InvoiceNotFoundException.class)
    public ResponseEntity<ErrorType>
handleNotFound(InvoiceNotFoundException nfe){
        return new ResponseEntity<ErrorType>(
                new ErrorType(
                        new
Date(System.currentTimeMillis()).toString(),
                        "404- NOT FOUND",
                        nfe.getMessage()),
                        HttpStatus.NOT_FOUND
                );
    }
}
```

d. Utility class: It is the helper class. To perform some operations on the entity class in business implementation layer.

```
package com.spring.restWebServiceAPI.utility;

import com.spring.restWebServiceAPI.dto.InvoiceDTO;
import com.spring.restWebServiceAPI.entity.Invoice;
import org.springframework.stereotype.Component;

@Component
public class InvoiceUtil {

    public Invoice calculateFinalAmountIncludingGST (Invoice inv){
        var amount = inv.getAmount();
        var gst = 0.18;
        var finalAmount = amount + (amount*gst);
        inv.setAmount(finalAmount);
        return inv;
    }

    public void copyNonNullValues(Invoice req, Invoice db){

        if(req.getAmount() != null){
            db.setAmount(req.getAmount());
        }

        if(req.getPayee() != null){
            db.setPayee(req.getPayee());
        }

        if(req.getReceiver() != null){
            db.setReceiver(req.getReceiver());
        }

        if(req.getId() != null){
            db.setId(req.getId());
        }
    }
}
```

e. Service Interface: It contains all abstract methods to be implemented in business layer.

```
package com.spring.restWebServiceAPI.service;

import com.spring.restWebServiceAPI.dto.InvoiceDTO;
import com.spring.restWebServiceAPI.entity.Invoice;

import java.util.List;

public interface InvoiceService {

    InvoiceDTO saveInvoice(InvoiceDTO inv);

    void updateInvoice (Long id,InvoiceDTO inv);

    void deleteInvoice (Long id);

    InvoiceDTO getOneInvoice(Long id);

    List<InvoiceDTO> getAllInvoices();

    boolean isInvoiceExist(Long id);

    Integer updateInvoiceAmountById(Double amount, Long id );
}
```

f. Service layer implementation:
   Here, we can see all our CRUD methods implementation

```
package com.spring.restWebServiceAPI.serviceImpl;

import com.spring.restWebServiceAPI.dto.InvoiceDTO;
import com.spring.restWebServiceAPI.entity.Invoice;
import com.spring.restWebServiceAPI.exception.InvoiceNotFoundException;
import com.spring.restWebServiceAPI.mapper.InvoiceMapper;
import com.spring.restWebServiceAPI.repo.InvoiceRepository;
import com.spring.restWebServiceAPI.service.InvoiceService;
import com.spring.restWebServiceAPI.utility.InvoiceUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.List;

@Service
public class InvoiceServiceImpl implements InvoiceService {

    @Autowired
    private InvoiceRepository invoiceRepository;

    @Autowired
    private InvoiceUtil invoiceUtil;

    @Autowired
    private InvoiceMapper invoiceMapper;

    @Override
    public InvoiceDTO saveInvoice(InvoiceDTO inv) {

        Invoice invoice = invoiceMapper.mapToInvoiceEntity(inv);
        invoice =
invoiceUtil.calculateFinalAmountIncludingGST(invoice);
        invoiceRepository.save(invoice);
        return invoiceMapper.mapToInvoiceDTO(invoice);
    }

    @Override
    public void updateInvoice(Long id,InvoiceDTO inv) {

        inv = getOneInvoice(id);
        Invoice invoice =
invoiceUtil.calculateFinalAmountIncludingGST(invoiceMapper.mapToInvoice
Entity(inv));
        invoiceRepository.save(invoice);
    }

    @Override
    public void deleteInvoice(Long id) {

        Invoice invoice = invoiceRepository.findById(id)
                .orElseThrow(()->new InvoiceNotFoundException("Invoice
doesnot exist"));
        invoiceRepository.delete(invoice);
    }

    @Override
    public InvoiceDTO getOneInvoice(Long id) {
        Invoice invoice = invoiceRepository.findById(id)
                .orElseThrow(()->new InvoiceNotFoundException(new
StringBuffer().append("Invoice not found for id ")
                        .append(id).toString()));
        return invoiceMapper.mapToInvoiceDTO(invoice);
    }

    @Override
    public List<InvoiceDTO> getAllInvoices() {
        List<Invoice> invoices = invoiceRepository.findAll();
        List<InvoiceDTO> invoicesDTO = new ArrayList<>();
        invoices.forEach(invoice -> {
            invoicesDTO.add(invoiceMapper.mapToInvoiceDTO(invoice));
        });

        return invoicesDTO;
    }

    @Override
    public boolean isInvoiceExist(Long id) {
        return invoiceRepository.existsById(id);
    }

    @Override
    @Transactional
    public Integer updateInvoiceAmountById(Double amount, Long id) {
        if(invoiceRepository.existsById(id)){
            return
invoiceRepository.updateInvoiceAmountById(amount,id);
        }else{
            throw new InvoiceNotFoundException("Invoice doesnot
exist");
        }
    }
}
```

We put @Transactional over updateInvoiceAmountById as it is defined explicitly in repository layer. It is a NON-SELECT QUERY.
By default, all other repository methods are transactional in nature.

In Service implementation layer, we can see the usage of mapper as all methods accept DTO as param, and convert it into entity as per requirement, like in repo.save(S entity)

We can also see the usage of custom exception INVOICENOTFOUNDEXCEPTION used along with lambda expression.

We can also see the usage for the utility class.

g. Repository layer

```
package com.spring.restWebServiceAPI.repo;

import com.spring.restWebServiceAPI.entity.Invoice;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;

public interface InvoiceRepository extends
JpaRepository<Invoice,Long> {

    //Non Select operation
    @Modifying
    @Query("UPDATE Invoice SET amount=:amount WHERE id=:id")
    Integer updateInvoiceAmountById(Double amount, Long id);
}
```

h. Controller layer

Its notable to see how, the response entity object is returnable in all of the methods to make it a globally supported format over network.

It also notable to see, how exceptions are considered and written taking their existence in service layer.
It is notable to see, all implementation of db, entity and business logic are not to be seen directly in controller layer.
i.e there is no Invoice.java, InvoiceRepository.java, InvoiceServiceImpl.java in controller code.

```
package com.spring.restWebServiceAPI.controller;

import com.spring.restWebServiceAPI.dto.InvoiceDTO;
import com.spring.restWebServiceAPI.exception.InvoiceNotFoundException;
import com.spring.restWebServiceAPI.service.InvoiceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api")
public class InvoiceRestController {

    @Autowired
    private InvoiceService invoiceService;

    @GetMapping("/invoices/{id}")
    public ResponseEntity<?> getInvoiceById(@PathVariable Long id){
        ResponseEntity<?> resp = null;
        try{
            InvoiceDTO inv = invoiceService.getOneInvoice(id);
            resp = new ResponseEntity<InvoiceDTO>(inv, HttpStatus.OK);
        }catch(InvoiceNotFoundException ex){
            throw ex;
        }catch(Exception e){
            e.printStackTrace();
            resp = new ResponseEntity<String>("Unable to find Invoice",
HttpStatus.NOT_FOUND);
        }
        return resp;
    }

    @GetMapping("/invoices")
    public ResponseEntity<?> getInvoices(){
        ResponseEntity<?> resp = null;
        try{
            List<InvoiceDTO> list = invoiceService.getAllInvoices();
            resp = new ResponseEntity<List<InvoiceDTO>>(list,
HttpStatus.OK);
        }catch(Exception e){
            e.printStackTrace();
            resp = new ResponseEntity<String>("Invoices not found",
HttpStatus.NOT_FOUND);
        }
        return resp;
    }

    @PostMapping(value="/invoices", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<?> saveInvoice(@ModelAttribute InvoiceDTO
invoiceDTO){
        ResponseEntity<?> response = null;
        try{
            invoiceDTO=invoiceService.saveInvoice(invoiceDTO);
            response = new
ResponseEntity<InvoiceDTO>(invoiceDTO,HttpStatus.OK);
        }catch(Exception e){
            e.printStackTrace();
            response = new ResponseEntity<String>("Invoice not saved",
HttpStatus.CONFLICT);
        }
        return response;
    }

    @DeleteMapping("/invoices/{id}")
    public ResponseEntity<String> deleteInvoice(@PathVariable Long id){
        ResponseEntity<String> response = null;
        try{
            invoiceService.deleteInvoice(id);
            response = new ResponseEntity<String>(new StringBuffer()
                    .append("Invoice with id ")
                    .append(id)
                    .append(" is deleted.").toString(), HttpStatus.OK);
        }catch(InvoiceNotFoundException nfe){
            throw nfe;
        }catch(Exception e){
            e.printStackTrace();
            response = new ResponseEntity<String>(new StringBuffer()
                    .append("Invoice could-not be ")

.append("Deleted").toString(),HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return response;
    }

    @PutMapping(value = "/invoices/{id}", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> updateInvoice(@PathVariable Long id,
@ModelAttribute InvoiceDTO invoiceDTO){
        ResponseEntity<String> response = null;
        try{
            invoiceService.updateInvoice(id,invoiceDTO);
            response = new ResponseEntity<String>("Invoice updated",
HttpStatus.OK);
        }catch(Exception nfe){
            nfe.printStackTrace();
            response = new ResponseEntity<String>("not updated",
HttpStatus.INTERNAL_SERVER_ERROR);

        }
        return response;
    }

    @PatchMapping("/invoices/{id}/{amount}")
    public ResponseEntity<String> updateAmount(
            @PathVariable Long id,
            @PathVariable Double amount
    )
    {
        ResponseEntity<String> response = null;
        try{
            invoiceService.updateInvoiceAmountById(amount,id);
            response = new ResponseEntity<String>("AMount updated",
HttpStatus.OK);
        }catch(InvoiceNotFoundException nfe){
            throw nfe;
        }catch(Exception e){
            e.printStackTrace();
            response = new ResponseEntity<String>("Invoice amount not
updated", HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return response;
    }



}
```