**package** com.stream;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.function.Consumer;

**import** java.util.stream.Stream;

**public** **class** StreamExample {

**public** **static** **void** main(String[] args) {

List<Integer> nums = Arrays.*asList*(6,4,1,8,9,3);

Stream<Integer> s1 = nums.stream();

Stream<Integer> s2 = s1.sorted();

Stream<Integer> s3 = s2.filter(n -> n%2 == 0);

Stream<Integer> s4 = s3.map(n -> n\*2);

Integer s5 = s4.reduce(0,(a,b) -> a+b);

System.***out***.println(s5);

//s3.forEach(n -> System.out.println(n));

}

}

**int** res = nums.stream()

.filter(n->n%2==0)

.map(n -> n\*2)

.reduce(0, (a,b) -> a+b);

System.***out***.println(res);

List<Integer> acRooms = Arrays.*asList*(1,2,3,4,5,6,7);

List<Integer> allRooms = Arrays.*asList*(1,2,3,8,9,21,20);

**long** acRoomsBooked = allRooms.stream().filter(room -> room >=1 && room <= 6).count();

**long** nonAcRoomsBooked = allRooms.stream().filter(room -> room >=7 && room <= 14).count();

**long** deluxBooked = allRooms.stream().filter(room -> room >=15 && room <= 21).count();

System.***out***.println(acRoomsBooked);

System.***out***.println(nonAcRoomsBooked);

System.***out***.println(deluxBooked);

List<Integer> list = Arrays.*asList*(2,3,4,5,6,7,8);

List<Integer> evenList = **new** ArrayList<>();

evenList =list.stream().filter(num -> num%2==0).map(num -> num+1).sorted().collect(Collectors.*toList*());

System.***out***.println(evenList);

If u don’t want to collect anything into a list we can directly print the every element:

FILTER : true or false

list.stream().filter(num -> num%2==0).map(num -> num+1).sorted().forEach(num->System.***out***.println(num));

list.stream().filter(num -> num%2==0).map(num -> num+1).sorted().forEach(System.***out***::println);

List<String> names = Arrays.*asList*("Shiva","RanguShiva","Srikanth","sakethmedagoni");

List<String> nameList = **new** ArrayList<>();

nameList = names.stream().filter(str->str.length()>4 && str.length()<8).collect(Collectors.*toList*());

System.***out***.println(nameList);

**class** Product{

**int** id;

String name;

**public** Product(**int** id, String name, **int** price) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.price = price;

}

**int** price;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

}

List<Product> products = **new** ArrayList<Product>();

products.add(**new** Product(1, "Hp", 20000));

products.add(**new** Product(2, "Dell", 25000));

products.add(**new** Product(3, "Lenovo", 30000));

products.add(**new** Product(4, "apple", 27000));

products.stream().filter(p->p.getPrice()>25000).forEach(pr->System.***out***.println(pr.getPrice()));

MAP: each element

List<String> names = Arrays.*asList*("Shiva","RanguShiva","Srikanth","sakethmedagoni");

List<String> nameList = **new** ArrayList<>();

nameList= names.stream().map(name->name.toUpperCase()).collect(Collectors.*toList*());

System.***out***.println(nameList);

List<Product> products = Arrays.*asList*(**new** Product(1, "Hp", 20000),

**new** Product(2, "Dell", 23000),

**new** Product(3, "Lenovo", 30000),

**new** Product(4, "apple", 27000)

);

List<Integer> list;

list = products.stream().filter(p -> p.price<25000).map(pr->pr.price+5000).collect(Collectors.*toList*());

System.***out***.println(list);

FLATMAP:  
List of Object

List<Integer> l1 = Arrays.*asList*(1,2);

List<Integer> l2 = Arrays.*asList*(3,4);

List<Integer> l3 = Arrays.*asList*(5,6);

List<List<Integer>> finalList = Arrays.*asList*(l1,l2,l3);

List<Integer> fl = finalList.stream().flatMap(l->l.stream().filter(n->n>3).map(m->m+5)).collect(Collectors.*toList*());

System.***out***.println(fl);

MIN:

List<Integer> lst = Arrays.*asList*(10,9,100,4,5,22);

Optional<Integer> min = lst.stream().min((v1,v2)-> {

**return** v1.compareTo(v2);

});

System.***out***.println(min.get());

COUNT:

**long** c = lst.stream().count();

System.***out***.println(c);

DISTINCT();

**long** c = lst.stream().distinct().count();

Stream<Integer> stream = Stream.of( **new** Integer[]{1,2,3,4,5,6,7,8,9} );

stream.forEach(p -> System.out.println(p));

SORTED: ANYMatch

Sorted(Comapareter.reverseOrder())

Set<String> fruits = **new** HashSet<>();

fruits.add("one apple");

fruits.add("one mango");

fruits.add("two grapes");

fruits.add("three banana");

fruits.add("one apple");

List<String> l = **new** ArrayList<>(fruits);

**boolean** ans = fruits.stream().anyMatch(value -> {**return** value.startsWith("one");});

System.***out***.println(ans);

Set<String> fruit = l.stream().sorted(Comparator.*reverseOrder*()).collect(Collectors.*toSet*());

System.***out***.println(fruit);

@PostMapping("/saveStudent")

**public** ResponseEntity<Student> saveStudent(@Valid @RequestBody Student stud){

Student response = studentService.saveStudent(stud);

**return** ResponseEntity.*ok*().body(response);

}

@PutMapping("/updateStudent/{age}")

**public** ResponseEntity<List<Student>> updateStudent(@PathVariable **int** age, @RequestBody Student stud){

List<Student> response = studentService.updateStudent(age, stud);

**return** ResponseEntity.*ok*(response);

}

**public** Student saveStudent(Student stud) {

Student response = repo.save(stud);

**return** response;

}

**public** List<Student> updateStudent(**int** age, Student stud) {

List<Student> st = repo.findByAge(age);

**if**(st == **null**) {

**throw** **new** ValidationException("mg");

}

**for**(Student s : st) {

s.setStudentName(stud.getStudentName());

s.setMobile(stud.getMobile());

s.setEmail(stud.getEmail());

s.setGender(stud.getGender());

}

List<Student> response = repo.saveAll(st);

**return** response;

}

@DeleteMapping("/deleteStudent")

**public** ResponseEntity<String> deleteStudent(@RequestParam("studentId") **int** studentId){

repo.deleteById(studentId);

**return** ResponseEntity.*ok*("deleted succssfully");

}

Search Service :

package com.hotelmanagement.domin.serviceimpl;

import java.sql.SQLException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.hotelmanagement.domin.dtos.SearchRoomsResponseDto;

import com.hotelmanagement.domin.exceptions.ReservationException;

import com.hotelmanagement.domin.repositories.SearchRoomsRepo;

import com.hotelmanagement.domin.service.SearchRoomsService;

import lombok.extern.slf4j.Slf4j;

@Slf4j

@Service

public class SearchRoomServiceImple implements SearchRoomsService {

@Autowired

SearchRoomsRepo hotelroomsrep;

@Override

public List<SearchRoomsResponseDto> getAvailableRooms(Date fromDate, Date toDate) throws SQLException {

log.info("method name: getAvailableRooms, request to getAvailableRooms from {} to {} ", fromDate, toDate);

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

String fromDateStr = sdf.format(fromDate);

String toDateStr = sdf.format(toDate);

List<SearchRoomsResponseDto> response;

List<Integer> reservedRoomsInRangeOfDates = hotelroomsrep.getAllReservedRoomsInRangeOfFromDateToDate(fromDate,

toDate);

int noOfAcRooms = hotelroomsrep.countRooms("AC");

int noOfNonAcRooms = hotelroomsrep.countRooms("NON\_AC");

int noOfDeluxRooms = hotelroomsrep.countRooms("DELUX");

if (reservedRoomsInRangeOfDates.isEmpty()) {

response = availableRoomsInRangeOfFromDateToDate(noOfAcRooms, noOfNonAcRooms, noOfDeluxRooms);

return response;

}

int reservedAcRooms = (int) reservedRoomsInRangeOfDates.stream().filter(room -> room >= 1 && room <= 7).count();

int reservedNonAcRooms = (int) reservedRoomsInRangeOfDates.stream().filter(room -> room >= 8 && room <= 13).count();

int reservedDeluxRooms = (int) reservedRoomsInRangeOfDates.stream().filter(room -> room >= 14 && room <= 21).count();

if (noOfAcRooms +noOfNonAcRooms + noOfDeluxRooms == reservedAcRooms + reservedNonAcRooms + reservedDeluxRooms) {

log.warn("method name:getAvailableRooms " + "No rooms available from " + fromDateStr + " to " + toDateStr);

throw new ReservationException("No rooms available from " + fromDateStr + " to " + toDateStr);

}

int availableAcRooms =noOfAcRooms - reservedAcRooms;

int availableNonAcRooms = noOfNonAcRooms - reservedNonAcRooms;

int availableDeluxRooms = noOfDeluxRooms - reservedDeluxRooms;

List<SearchRoomsResponseDto> result = availableRoomsInRangeOfFromDateToDate(availableAcRooms,

availableNonAcRooms, availableDeluxRooms);

log.info("successfully returned list of available rooms...");

return result;

}

public List<SearchRoomsResponseDto> availableRoomsInRangeOfFromDateToDate(int acRooms, int nonAcRooms,

int deluxRooms) {

log.info("Method Name:availableRoomsInRangeOfFromDateToDate Running...");

List<SearchRoomsResponseDto> available = new ArrayList<>();

SearchRoomsResponseDto acRoomDto = new SearchRoomsResponseDto();

acRoomDto.setRoomType("AC");

acRoomDto.setPrice(1500);

acRoomDto.setTotalAvailableRooms(acRooms);

SearchRoomsResponseDto nonAcRoomDto = new SearchRoomsResponseDto();

nonAcRoomDto.setRoomType("NON\_AC");

nonAcRoomDto.setPrice(1200);

nonAcRoomDto.setTotalAvailableRooms(nonAcRooms);

SearchRoomsResponseDto deluxRoomDto = new SearchRoomsResponseDto();

deluxRoomDto.setRoomType("DELUX");

deluxRoomDto.setPrice(2000);

deluxRoomDto.setTotalAvailableRooms(deluxRooms);

available.add(acRoomDto);

available.add(nonAcRoomDto);

available.add(deluxRoomDto);

return available;

}

}

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd" );

Date newcheckinD = sdf.parse(bookingdetails.getCheckInDate());

Date newcheckoutD = sdf.parse(bookingdetails.getCheckOutDate());

Calendar checkInDateWithTime = Calendar.*getInstance*();

checkInDateWithTime.setTime(newcheckinD);

checkInDateWithTime.set(Calendar.***HOUR\_OF\_DAY***, 12);

checkInDateWithTime.set(Calendar.***MINUTE***, 0);

checkInDateWithTime.set(Calendar.***SECOND***, 0);

Calendar checkOutDateWithTime = Calendar.*getInstance*();

checkOutDateWithTime.setTime(newcheckoutD);

checkOutDateWithTime.set(Calendar.***HOUR\_OF\_DAY***, 11);

checkOutDateWithTime.set(Calendar.***MINUTE***, 0);

checkOutDateWithTime.set(Calendar.***SECOND***, 0);

SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String formatttedCheckInDate = dateFormat.format( checkInDateWithTime.*getInstance*());

String formatttedCheckOutDate = dateFormat.format( checkOutDateWithTime.*getInstance*());

bookings.setCheckinDate(formatttedCheckInDate);

bookings.setCheckoutDate(formatttedCheckOutDate);

server.port=3307

spring.datasource.url=jdbc:mysql://localhost:3306/hotelReservationSystem

spring.datasource.username=root

spring.datasource.password=Srikanth@1234

spring.jpa.hibernate.ddl-auto=update

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

**package com.room.reservation.api.domain.services;**

**import java.text.SimpleDateFormat;**

**import java.util.Date;**

**import java.util.TimeZone;**

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.stereotype.Service;**

**import com.fasterxml.jackson.databind.ObjectMapper;**

**import com.room.reservation.api.domain.dtos.ApiResponseDto;**

**import com.room.reservation.api.domain.enums.CustomHttpStatus;**

**import com.room.reservation.api.domain.exceptions.ValidationException;**

**import lombok.extern.slf4j.Slf4j;**

**import okhttp3.OkHttpClient;**

**import okhttp3.Request;**

**import okhttp3.Response;**

**@Slf4j**

**@Service**

**public class SearchRoomService {**

**private final OkHttpClient client = new OkHttpClient();**

**public ResponseEntity<?> getAvailableRooms(Date fromDate, Date toDate, int staffId, int custId) {**

**log.info("Method Name:getAvailableRooms communicating with search service , running..");**

**SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");**

**// dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));**

**String formattedFromDate = dateFormat.format(fromDate);**

**String formattedToDate = dateFormat.format(toDate);**

**log.info("{}{}", fromDate, toDate);**

**String url = "http://localhost:8081/api/search-rooms/available?fromDate=" + formattedFromDate + "&toDate="**

**+ formattedToDate;**

**log.info(url);**

**Request request = new Request.Builder().url(url).addHeader("staffId", String.valueOf(staffId))**

**.addHeader("custId", String.valueOf(custId)).build();**

**try (Response response = client.newCall(request).execute()) {**

**if (!response.isSuccessful()) {**

**log.error("Method Name: getAvailableRooms and message:Failed to get response from server");**

**throw new ValidationException("Failed to get response from server",**

**CustomHttpStatus.INTERNAL\_SERVER\_ERROR);**

**}**

**String responseBody = response.body().string();**

**ObjectMapper objectMapper = new ObjectMapper();**

**Object json = objectMapper.readValue(responseBody, Object.class);**

**return ResponseEntity.ok(json);**

**} catch (Exception e) {**

**log.error("Method Name:getAvailableRooms message:Failed to get data from search service");**

**e.printStackTrace();**

**ApiResponseDto apiresponse = new ApiResponseDto("Failed to get data from search service", false);**

**return ResponseEntity.status(HttpStatus.valueOf(CustomHttpStatus.INTERNAL\_SERVER\_ERROR.getCode()))**

**.body(apiresponse);**

**}**

**}**

**}**

View microservice

**package** com.hotelmanagement.domin.service;

**import** java.io.IOException;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Service;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.hotelmanagement.domin.dtos.ViewReservationResponseDto;

**import** com.hotelmanagement.domin.exceptions.ReservationException;

**import** okhttp3.OkHttpClient;

**import** okhttp3.Request;

**import** okhttp3.Response;

@Service

**public** **class** BookingService {

OkHttpClient client = **new** OkHttpClient();

**public** ResponseEntity<ViewReservationResponseDto> reserveRoom(**int** bookingId, **int** custId, **int** staffId) {

// **TODO** Auto-generated method stub

String newUrl = "http://localhost:3307/view-api/view-reservation?bookingId="+bookingId;

//String url = baseUrl+"/search-rooms/available?fromDate=" + formattedFromDate + "&toDate="

// + formattedToDate;

Request req = **new** Request.Builder().url(newUrl)

.addHeader("custId", String.*valueOf*(custId))

.addHeader("staffId",String.*valueOf*(staffId)).build();

**try** {

Response res = client.newCall(req).execute();

**if**(!res.isSuccessful()) {

**throw** **new** ReservationException("failed to get data");

}

String response = res.body().string();

ObjectMapper mapper = **new** ObjectMapper();

ViewReservationResponseDto dto = mapper.readValue(response, ViewReservationResponseDto.**class**);

**return** ResponseEntity.*ok*(dto);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **null**;

}

}

Booking micro

**package** com.hotelmanagement.domin.service;

**import** java.io.IOException;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Service;

**import** com.fasterxml.jackson.core.JsonProcessingException;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.hotelmanagement.domin.dtos.RoomsReservationRequestDto;

**import** com.hotelmanagement.domin.enums.CustomHttpStatus;

**import** com.hotelmanagement.domin.exceptions.ReservationException;

**import** jakarta.validation.ValidationException;

**import** okhttp3.MediaType;

**import** okhttp3.OkHttpClient;

**import** okhttp3.Request;

**import** okhttp3.RequestBody;

**import** okhttp3.Response;

@Service

**public** **class** BookingService {

OkHttpClient client = **new** OkHttpClient();

@Value("${baseurl}")

String baseUrl;

**public** ResponseEntity<String> reserveRoom(RoomsReservationRequestDto request, **int** custId, **int** staffId) **throws** JsonProcessingException {

MediaType JSON = MediaType.*parse*("application/json; charset=utf-8");

String json = **new** ObjectMapper().writeValueAsString(request);

RequestBody body = RequestBody.*create*(json, JSON);

String newUrl = baseUrl + "/reservations/api/reserve-rooms";

Request req = **new** Request.Builder().url(newUrl).addHeader("custId", String.*valueOf*(custId))

.addHeader("staffId", String.*valueOf*(staffId)).post(body).build();

**try** {

Response res = client.newCall(req).execute();

**if** (!res.isSuccessful()) {

**throw** **new** ReservationException("failed to get data");

}

String response = res.body().string();

// ObjectMapper mapper = new ObjectMapper();

// ViewReservationResponseDto dto = mapper.readValue(response, ViewReservationResponseDto.class);

**return** ResponseEntity.*ok*(response);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** ValidationException("failed server" + CustomHttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

}

}

Calling req=obj and res=obj

**package** com.hotelmanagement.domin.service;

**import** java.io.IOException;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.stereotype.Service;

**import** com.fasterxml.jackson.core.JsonProcessingException;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.hotelmanagement.domin.dtos.RoomsReservartionResponseDto;

**import** com.hotelmanagement.domin.dtos.RoomsReservationRequestDto;

**import** com.hotelmanagement.domin.enums.CustomHttpStatus;

**import** com.hotelmanagement.domin.exceptions.ReservationException;

**import** jakarta.validation.ValidationException;

**import** okhttp3.MediaType;

**import** okhttp3.OkHttpClient;

**import** okhttp3.Request;

**import** okhttp3.RequestBody;

**import** okhttp3.Response;

@Service

**public** **class** BookingService {

OkHttpClient client = **new** OkHttpClient();

@Value("${baseurl}")

String baseUrl;

**public** ResponseEntity<RoomsReservartionResponseDto> reserveRoom(RoomsReservationRequestDto request, **int** custId, **int** staffId) **throws** JsonProcessingException {

MediaType JSON = MediaType.*parse*("application/json; charset=utf-8");

String json = **new** ObjectMapper().writeValueAsString(request);

RequestBody body = RequestBody.*create*(json, JSON);

String newUrl = baseUrl + "/reservations/api/reserve-rooms";

Request req = **new** Request.Builder().url(newUrl).addHeader("custId", String.*valueOf*(custId))

.addHeader("staffId", String.*valueOf*(staffId)).post(body).build();

**try** {

Response res = client.newCall(req).execute();

**if** (!res.isSuccessful()) {

**throw** **new** ReservationException("failed to get data");

}

String response = res.body().string();

ObjectMapper mapper = **new** ObjectMapper();

RoomsReservartionResponseDto dto = mapper.readValue(response, RoomsReservartionResponseDto.**class**);

**return** ResponseEntity.*ok*(dto);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** ValidationException("failed server" + CustomHttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

}

}

**Swapping Pairs of Characters in a String in Java**

String s = "shivapatel";

char[] ch = s.toCharArray();

for(int i=0;i<ch.length-1;i+=2){

char temp = ch[i];

ch[i]=ch[i+1];

ch[i+1]=temp;

}

String s1 = new String(ch);

System.out.println(ch);

# Split a String into a Number of Substrings in Java

import java.util.\*;

public class Main

{

public static void main(String[] args) {

String s = "shiva";

List<String> list = new ArrayList<>();

for(int i=0;i<s.length();i++){

for(int j=i+1;j<=s.length();j++){

list.add(s.substring(i,j));

}

}

System.out.println(list);

}

}

Swagger:

<dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.0.2</version>  
</dependency>

File:

package com.atm.management.api.domain.swaggerconfig;  
  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import io.swagger.v3.oas.models.OpenAPI;  
import io.swagger.v3.oas.models.info.Info;  
  
@Configuration  
public class SwaggerConfig {  
  
 @Bean  
 public OpenAPI myOpenAPI() {  
  
 Info info = new Info().title("ATM Management")  
 .description("This API exposes endpoints to ATM Services.");  
  
 return new OpenAPI().info(info);  
 }  
}

Application.properties:

springdoc.swagger-ui.path = /swagger-ui.html

Config Server:

Dependency : In micro serviece1

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
 <version>4.1.0</version>  
</dependency>

Application.properties:

spring.config.import=optional:configserver:http://localhost:3311

Dependency in config server :

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-config-server</artifactId>  
</dependency>

Main lo @EnableConfigServer

Application properties

server.port=3311  
spring.cloud.config.server.git.uri=https://github.com/shivapatelrangu/AtmConfigServer.git

If two microservices unte:

Git lo file ni microservice name tho create cheyali :

spring.application.name=configServer

microservise name : AtmManagement

file ni AtmManagement.properties name tho create cheyali.

package com.hotelmanagement.domin.controller;

import java.sql.SQLException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.format.annotation.DateTimeFormat;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import com.hotelmanagement.domin.dtos.SearchRoomsResponseDto;

import com.hotelmanagement.domin.kafka.KafkaConsumer;

import com.hotelmanagement.domin.kafka.KafkaProd;

import jakarta.validation.Valid;

import lombok.extern.slf4j.Slf4j;

@RestController

@RequestMapping("/api")

@Slf4j

public class SearchCont {

@Autowired

KafkaProd producer;

@Autowired

KafkaConsumer cons;

@GetMapping("/rooms")

public ResponseEntity<List<SearchRoomsResponseDto>> searchAvailableRooms(

@Valid @RequestParam("fromDate") @DateTimeFormat(pattern = "yyyy-MM-dd") String fromDate,

@Valid @RequestParam("toDate") @DateTimeFormat(pattern = "yyyy-MM-dd") String toDate, @RequestHeader("staffId" ) String staffId, @RequestHeader("custId") String customerId) throws ParseException, SQLException {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

//dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));

Date from\_date = dateFormat.parse(fromDate);

Date to\_date = dateFormat.parse(toDate);

log.info("/search/available method name: searchAvailableRooms"

+ "recieved request to search rooms from {} to {}, staffId={} and customerId={}", fromDate, toDate, staffId , customerId );

producer.sendMessage(fromDate,toDate);

List<SearchRoomsResponseDto> res = cons.listen(fromDate,toDate);

return ResponseEntity.ok(res);

}

}

Overfetching and underfetching

REST APIs always return a whole dataset. For example, from a *person* object in the REST API, you would receive the person’s name, date of birth, address, and phone number. You would get all of this data even if you only needed a phone number.

Similarly, if you wanted to know a person’s phone number and last purchase, you would need multiple REST API requests. The URL */person* would return the phone number and the URL */purchase* would return purchase history.

Social media developers had to write a lot of code just to process API requests, which affected performance and user experience.

GraphQL emerged as a query-based solution. Queries can return the exact data in only one API request and response exchange.

->  GraphQL operates over a single endpoint using HTTP.

-> In addition, REST development has been more focused on making new APIs. Meanwhile, GraphQL’s focus has been on API performance and flexibility.

**Client-side request**

Here’s what a REST request uses to work:

* HTTP verbs that determine the action
* A URL that identifies the resource on which to action the HTTP verb
* Parameters and values to parse, if you want to create or modify an object within an existing server-side resource

For example, you use *GET* to get read-only data from a resource, *POST* to add a new resource entry, or *PUT* to update a resource.

In contrast, here’s what GraphQL requests use:

* Query for getting read-only data
* Mutation for modifying data
* Subscription to receive event-based or streaming data updates

A *data format*describes how you would like the server to return the data, including objects and fields that match the server-side schema. You can also input new data. Internally, GraphQL sends every client request as a *POST* HTTP request.

* You have limited bandwidth, and you want to minimize the number of requests and responses
* You have multiple data sources, and you want to combine them at one endpoint
* You have client requests that vary significantly, and you expect very different responses

On the other hand, REST is is likely a better choice if you have these considerations:

* You have smaller applications with less complex data
* You have data and operations that all clients use similarly
* You have no requirements for complex data querying

JBOSS:

**WildFly**,[[2]](https://en.wikipedia.org/wiki/WildFly#cite_note-2) formerly known as **JBoss AS**, or simply **JBoss**, is an [application server](https://en.wikipedia.org/wiki/Application_server) written by [JBoss](https://en.wikipedia.org/wiki/JBoss_(company)), now developed by [Red Hat](https://en.wikipedia.org/wiki/Red_Hat). WildFly is written in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) and implements the [Java Platform, Enterprise Edition](https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition) (Java EE) specification.[[3]](https://en.wikipedia.org/wiki/WildFly#cite_note-FOOTNOTEStancapiano20178%E2%80%939Chapter_%C2%A71_Introducing_Java_EE_and_Configuring_the_Development_Environment-3) It runs on [multiple platforms](https://en.wikipedia.org/wiki/Cross-platform).

WildFly is [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software),[[3]](https://en.wikipedia.org/wiki/WildFly#cite_note-FOOTNOTEStancapiano20178%E2%80%939Chapter_%C2%A71_Introducing_Java_EE_and_Configuring_the_Development_Environment-3) subject to the requirements of the [GNU Lesser General Public License](https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License) (LGPL), version 2.1.

ORIGIN:  
In 1999, [Marc Fleury](https://en.wikipedia.org/wiki/Marc_Fleury) started a free software project named *EJB-OSS* (stands for *Enterprise Java Bean Open Source Software*) implementing the [EJB](https://en.wikipedia.org/wiki/Enterprise_JavaBeans) [API](https://en.wikipedia.org/wiki/API) from [J2EE (Java 2 Enterprise Edition)](https://en.wikipedia.org/wiki/Java_EE). [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems) asked the project to stop using the trademarked *EJB* within its name. *EJB-OSS* was then renamed to *JBOSS*, then *JBoss* later.[[4]](https://en.wikipedia.org/wiki/WildFly#cite_note-FOOTNOTEJamaeJohnson20104-4)

On November 20, 2014, JBoss Application Server was renamed WildFly. The JBoss Community and other Red Hat JBoss products like [JBoss Enterprise Application Platform](https://en.wikipedia.org/wiki/JBoss_Enterprise_Application_Platform) were not renamed.[[5]](https://en.wikipedia.org/wiki/WildFly#cite_note-5)

[The transition from JBoss to WildFly was primarily driven by the need for a more modern, lightweight, and flexible platform1](https://www.endpointdev.com/blog/2017/07/jboss-456-to-wildfly-migration-tips/). Here are some reasons for the update:

1. [**End of Life Support**: Older versions of JBoss no longer have life support, which necessitates an upgrade to a supported platform like WildFly2](https://stackoverflow.com/questions/59199185/migration-from-jboss-4-2-3-to-wildfly-8-is-it-worth-the-effort).
2. [**Modern Technologies**: WildFly supports the latest technologies and standards, including Jakarta EE and Eclipse MicroProfile, which are essential for developing modern applications3](https://docs.wildfly.org/26/Getting_Started_Guide.html).
3. [**Performance**: WildFly is designed to be exceptionally fast, lightweight, and powerful, making it a better choice for hosting highly transactional Java applications and services](https://docs.wildfly.org/26/Getting_Started_Guide.html)

**Uses of WildFly:**

1. [**Enterprise Applications**: WildFly is primarily used for deploying and running Java-based enterprise applications5](https://blog.servermania.com/wildfly-server). [This includes web applications, microservices, messaging systems, and batch processing5](https://blog.servermania.com/wildfly-server).
2. [**Database Connection**: WildFly provides a Java web application an extension to the JVM with a complete runtime environment that will create the connection of database on one end to the web client on the other6](https://www.jrebel.com/blog/wildfly).
3. [**Development and Production**: As an open-source community project, WildFly can be used in both development and production level deployments6](https://www.jrebel.com/blog/wildfly). [It’s often chosen for innovative/pilot projects or generally low-risk projects](https://www.mastertheboss.com/articles/faqs/what-is-wildfly/)
4. [**Fast Deployments**: WildFly is known for its fast deployments and its ability to support enterprise applications if needed6](https://www.jrebel.com/blog/wildfly). [It also provides features like managed domain mode that allows the user to deploy several WildFly instances6](https://www.jrebel.com/blog/wildfly).

[In summary, WildFly is a powerful, flexible, and lightweight application server that helps you build amazing applications8](https://www.wildfly.org/). [It provides all the necessary features to run a Java web application1](https://opensource.com/article/21/7/wildfly). [It’s designed and maintained by Red Hat and was formally known as JBoss AS1](https://opensource.com/article/21/7/wildfly).

JBOSS: Java Bean Open Source Software

* Founded by Marc fleury in 1999
* Till JBOSS 7 it is JBOSS after(from 8) it will become wildfly. JBOSS 8 == wildfly.
* Not one person will develop this, multiple people will develop it, if iam familiar with some kind of topic I will offer it(anybody can offer , suppose I will develop framework for logging, Jboss will accept it and releases it with the next community edition).
* In 2006 The RedHat bought Jboss Inc.
* Jboss is a Community edition
* Jboss EAP is a EE
* If somebody asks you what server is using in your production u cant say Joss, Say Jboss EAP
* If you have a issue with the EE you raise a req to Redhat they will provide a support for u. but in community edition no one will provide you support.

COMPONENTS OF Jboss

* Wildfly 10.1.2
* Undertow – is a webserver which is packaged along with Jboss.
* Jgroups & Infinispan – help you to do clustering in Jboss
* Iron jacamar – 3rd party component helps you to do EJB level clustering.
* Narayan(Trxn) – is a transaction sw which will help you to do transactions.
* Active MQ Artemis –jboss product, helps you do messaging in jboss.
* Jboss Logging
* Hall
* Jboss Security
* Picket pox maintain pswd securly
* Netty – http based protocal connector

Operating Modes:

1. Standalone Mode
2. Domain Mode