**PatientSky Backend Developer Test**

Input:

Calendar Ids: Person1, Person2, Person3

For each Calendar Id:

1. a) Find the available time for each calendar Id.
2. Assumption: Interval startTime/endTime is exactly matching with timeslot startTime/endTime
3. Remove the time slot which already have appointment.
4. Merge the available timeslots as per given duration.
5. a) Find common available time across calender Ids.

b) Create a combined timeSlot when the break is found in endTime and startTime of consecutive timeslots.

c) Use the combined timeslot after the break as StartTime for a next combined slot.

3) a) Check for Duration in available timeslots.

b) Create a split till endTime of each Split is not crossing the endTime of combined Timeslot.

……………………………………………………………………………………………….

|  |
| --- |
| 30 min |
|  |
| P1 |
| 0 = {TimeSlot@1956} "TimeSlot{start=2019-04-23T08:15, end=2019-04-23T08:30}" |
| 1 = {TimeSlot@1960} "TimeSlot{start=2019-04-23T08:30, end=2019-04-23T08:45}" |
| 2 = {TimeSlot@1961} "TimeSlot{start=2019-04-23T08:45, end=2019-04-23T09:00}" |
| 3 = {TimeSlot@1962} "TimeSlot{start=2019-04-23T09:00, end=2019-04-23T09:15}" |
| 4 = {TimeSlot@1963} "TimeSlot{start=2019-04-23T09:15, end=2019-04-23T09:45}" |
| 5 = {TimeSlot@1964} "TimeSlot{start=2019-04-23T09:45, end=2019-04-23T10:15}" |
| 6 = {TimeSlot@1965} "TimeSlot{start=2019-04-23T10:15, end=2019-04-23T10:45}" |
| 7 = {TimeSlot@1966} "TimeSlot{start=2019-04-23T10:45, end=2019-04-23T11:15}" |
| 8 = {TimeSlot@1967} "TimeSlot{start=2019-04-23T11:15, end=2019-04-23T11:45}" |
| 9 = {TimeSlot@1968} "TimeSlot{start=2019-04-23T12:15, end=2019-04-23T12:45}" |
| 10 = {TimeSlot@1969} "TimeSlot{start=2019-04-23T12:45, end=2019-04-23T13:15}" |
| 11 = {TimeSlot@1970} "TimeSlot{start=2019-04-23T13:15, end=2019-04-23T13:45}" |
| 12 = {TimeSlot@1971} "TimeSlot{start=2019-04-23T13:45, end=2019-04-23T14:15}" |
| 13 = {TimeSlot@1972} "TimeSlot{start=2019-04-23T14:30, end=2019-04-23T14:45}" |
| 14 = {TimeSlot@1973} "TimeSlot{start=2019-04-23T14:45, end=2019-04-23T15:00}" |

P2

|  |
| --- |
| 0 = {TimeSlot@2066} "TimeSlot{start=2019-04-23T08:00, end=2019-04-23T08:30}" |
| 1 = {TimeSlot@2067} "TimeSlot{start=2019-04-23T08:30, end=2019-04-23T09:00}" |
| 2 = {TimeSlot@2068} "TimeSlot{start=2019-04-23T09:00, end=2019-04-23T09:30}" |
| 3 = {TimeSlot@2069} "TimeSlot{start=2019-04-23T09:30, end=2019-04-23T10:00}" |
| 4 = {TimeSlot@2070} "TimeSlot{start=2019-04-23T10:00, end=2019-04-23T10:30}" |
| 5 = {TimeSlot@2071} "TimeSlot{start=2019-04-23T10:30, end=2019-04-23T10:45}" |
| 6 = {TimeSlot@2072} "TimeSlot{start=2019-04-23T10:45, end=2019-04-23T11:00}" |
| 7 = {TimeSlot@2073} "TimeSlot{start=2019-04-23T11:00, end=2019-04-23T11:15}" |
| 8 = {TimeSlot@2074} "TimeSlot{start=2019-04-23T11:15, end=2019-04-23T11:30}" |
| 9 = {TimeSlot@2075} "TimeSlot{start=2019-04-23T11:30, end=2019-04-23T11:45}" |
| 10 = {TimeSlot@2076} "TimeSlot{start=2019-04-23T11:45, end=2019-04-23T12:00}" |
| 11 = {TimeSlot@2077} "TimeSlot{start=2019-04-23T12:00, end=2019-04-23T12:15}" |
| 12 = {TimeSlot@2078} "TimeSlot{start=2019-04-23T12:30, end=2019-04-23T12:45}" |
| 13 = {TimeSlot@2079} "TimeSlot{start=2019-04-23T12:45, end=2019-04-23T13:00}" |
| 14 = {TimeSlot@2080} "TimeSlot{start=2019-04-23T13:00, end=2019-04-23T13:30}" |
| 15 = {TimeSlot@2081} "TimeSlot{start=2019-04-23T13:30, end=2019-04-23T14:00}" |
| 16 = {TimeSlot@2082} "TimeSlot{start=2019-04-23T14:00, end=2019-04-23T14:30}" |
| 17 = {TimeSlot@2083} "TimeSlot{start=2019-04-23T14:30, end=2019-04-23T15:00}" |
| 18 = {TimeSlot@2084} "TimeSlot{start=2019-04-23T15:00, end=2019-04-23T15:30}" |
| 19 = {TimeSlot@2085} "TimeSlot{start=2019-04-23T15:30, end=2019-04-23T16:00}" |

P3

|  |
| --- |
| 0 = {TimeSlot@2030} "TimeSlot{start=2019-04-23T08:15, end=2019-04-23T08:30}" |
| 1 = {TimeSlot@2031} "TimeSlot{start=2019-04-23T08:30, end=2019-04-23T08:45}" |
| 2 = {TimeSlot@2032} "TimeSlot{start=2019-04-23T08:45, end=2019-04-23T09:00}" |
| 3 = {TimeSlot@2033} "TimeSlot{start=2019-04-23T09:00, end=2019-04-23T09:15}" |
| 4 = {TimeSlot@2034} "TimeSlot{start=2019-04-23T10:15, end=2019-04-23T10:45}" |
| 5 = {TimeSlot@2035} "TimeSlot{start=2019-04-23T10:45, end=2019-04-23T11:15}" |
| 6 = {TimeSlot@2036} "TimeSlot{start=2019-04-23T11:15, end=2019-04-23T11:45}" |
| 7 = {TimeSlot@2037} "TimeSlot{start=2019-04-23T12:15, end=2019-04-23T12:45}" |
| 8 = {TimeSlot@2038} "TimeSlot{start=2019-04-23T12:45, end=2019-04-23T13:15}" |
| 9 = {TimeSlot@2039} "TimeSlot{start=2019-04-23T14:30, end=2019-04-23T14:45}" |
| 10 = {TimeSlot@2040} "TimeSlot{start=2019-04-23T14:45, end=2019-04-23T15:00}" |
|  |

………………………………………………………………………………………………………………

**Code:**

package practice;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.DeserializationFeature;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;

import java.io.IOException;

import java.io.InputStream;

import java.time.LocalDateTime;

import java.time.temporal.ChronoUnit;

import java.util.\*;

/\*\*

\* Hello world!

\*

\*/

public class App{

Map<String, CalendarInfo> calendarIds;

public static void main( String[] args )

{

App app = new App();

try {

app.loadCalendarData();

app.init();

} catch (IOException e) {

System.out.println(e);

}

}

void init() throws IOException {

findAvailableTime(Arrays.asList("48644c7a-975e-11e5-a090-c8e0eb18c1e9","48cadf26-975e-11e5-b9c2-c8e0eb18c1e9","452dccfc-975e-11e5-bfa5-c8e0eb18c1e9"), 30,

new Interval(

LocalDateTime.of(2019,4,23,8,15,0),

LocalDateTime.of(2019,4,23, 15,45,0)

),

""

);

}

private void findAvailableTime(List<String> calendarIds, int duration, Interval interval, String timeSlotType) {

List<List<TimeSlot>> availabletimeslotList = new ArrayList<>();

//1. Find available time for each Calendar Id

for (String eachCalenderId : calendarIds) {

CalendarInfo calendarInfo = this.calendarIds.get(eachCalenderId);

List<Appointment> appointments = calendarInfo.getAppointments();

List<TimeSlot> timeslots = calendarInfo.getTimeslots();

List<TimeSlot> availableTime = findAvailableTime(timeslots, appointments, interval);

availabletimeslotList.add(availableTime);

}

//2. Find common available time across calendar Ids

List<TimeSlot> primaryTimeSlot = availabletimeslotList.get(0);

primaryTimeSlot = createChunks(primaryTimeSlot);

for (int i = 1; i < availabletimeslotList.size() ; i++) {

List<TimeSlot> secondaryTimeSlots = availabletimeslotList.get(i);

secondaryTimeSlots = createChunks(secondaryTimeSlots);

findCommonTime(primaryTimeSlot, secondaryTimeSlots);

}

System.out.println("Combined common time between calendarIds : "+primaryTimeSlot);

//3. Check for Duration in available timeslots.

List<TimeSlot> splitTimeSlotAsPerDuration = splitTimeSlotAsPerDuration(primaryTimeSlot, duration);

System.out.println("TimeSlots Available between all calendarIds : "+splitTimeSlotAsPerDuration);

}

private List<TimeSlot> createChunks(List<TimeSlot> timeslotList) {

List<TimeSlot> combinedTimeSlot = new ArrayList<>();

TimeSlot startTime = timeslotList.get(0);

for (int i = 0; i < timeslotList.size()-1; i++) {

TimeSlot timeSlot1 = timeslotList.get(i);

TimeSlot timeSlot2 = timeslotList.get(i + 1);

System.out.println("Comparing and Combining : "+timeSlot1+" :: "+timeSlot2);

if (!timeSlot1.getEnd().equals(timeSlot2.getStart())){

// Create a combined timeSlot when the break is found in endTime and startTime of consecutive timeslots.

combinedTimeSlot.add(new TimeSlot(startTime.getStart(), timeSlot1.getEnd()));

// Use the combined timeslot after the break as StartTime for a next combined slot.

startTime = timeslotList.get(i+1);

}

}

combinedTimeSlot.add(new TimeSlot(startTime.getStart(), timeslotList.get(timeslotList.size()-1).getEnd()));

return combinedTimeSlot;

}

private List<TimeSlot> splitTimeSlotAsPerDuration(List<TimeSlot> combinedTimeSlot, int duration) {

List<TimeSlot> splitedTimeSlots = new ArrayList<>();

for (TimeSlot timeSlot : combinedTimeSlot) {

long minutes = ChronoUnit.MINUTES.between(timeSlot.getStart(), timeSlot.getEnd());

if(minutes> duration) {

LocalDateTime startTime = timeSlot.getStart();

LocalDateTime endTime = null;

do {

endTime = startTime.plusMinutes(duration);

splitedTimeSlots.add(new TimeSlot(startTime, endTime));

startTime = endTime;

}while (endTime.isBefore(timeSlot.getEnd()));

//Create a split till endTime of each Split is not crossing the endTime of combinedTimeslot.

} else {

splitedTimeSlots.add(new TimeSlot(timeSlot.getStart(), timeSlot.getEnd()));

}

}

return splitedTimeSlots;

}

private void findCommonTime(List<TimeSlot> primaryTimeSlot, List<TimeSlot> timeSlots) {

Iterator<TimeSlot> iterator = primaryTimeSlot.iterator();

while (iterator.hasNext()) {

TimeSlot timeSlot = iterator.next();

boolean ifTimeExistsWithinRange = false;

for (TimeSlot slot : timeSlots) {

ifTimeExistsWithinRange = findIfTimeExistsWithinRange(timeSlot, slot);

if(ifTimeExistsWithinRange)

break;

// System.out.println(ifTimeExistsWithinRange);

}

if(!ifTimeExistsWithinRange){

System.out.println("Removing the timeframe: "+timeSlot+" from primary timeslot list");

iterator.remove();

}

}

}

private boolean findIfTimeExistsWithinRange(TimeSlot timeSlot, TimeSlot slot) {

LocalDateTime start1 = timeSlot.getStart();

LocalDateTime end1 = timeSlot.getEnd();

long timePeriod1 = ChronoUnit.MINUTES.between(start1, end1);

LocalDateTime start2 = slot.getStart();

LocalDateTime end2 = slot.getEnd();

long timePeriod2 = ChronoUnit.MINUTES.between(start2, end2);

System.out.println("Comparing timePeriod1 : "+timeSlot+ " :: timePeriod2 : "+slot);

if(timePeriod1 > timePeriod2){

// Check timePeriod2 contains in timePeriod1

return ( start2.isAfter(start1) || start2.isEqual(start1) ) && (end2.isBefore(end1) || end2.isEqual(end1) );

} else {

// Check timePeriod1 contains in timePeriod2

return ( start1.isAfter(start2) || start1.isEqual(start2) ) && (end1.isBefore(end2) || end1.isEqual(end2) );

}

}

/\*

Assumption : Interval startTime/endTime is exactly matching with timeslot startTime/endTime

\*/

private List<TimeSlot> findAvailableTime(List<TimeSlot> timeslots, List<Appointment> appointments, Interval interval) {

Iterator<TimeSlot> iterator = timeslots.iterator();

while(iterator.hasNext()){

TimeSlot timeslot = iterator.next();

if((timeslot.getEnd().isBefore(interval.getStart()) ||timeslot.getEnd().isEqual(interval.getStart()))

|| (timeslot.getStart().isAfter(interval.getEnd()) ||timeslot.getStart().isEqual(interval.getEnd()))){

iterator.remove();

continue;

}

for (Appointment appointment : appointments) {

if (timeslot.getStart().equals(appointment.getStart())

&& timeslot.getEnd().equals(appointment.getEnd())) {

//Remove the timeSlots which already have appointment

iterator.remove();

break;

}

}

}

return timeslots;

}

void loadCalendarData() throws IOException {

InputStream resourceAsStream = App.class.getClassLoader().getResourceAsStream("John.json");

ObjectMapper objectMapper = new ObjectMapper();

objectMapper.configure(DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES, false);

objectMapper.configure(DeserializationFeature.READ\_DATE\_TIMESTAMPS\_AS\_NANOSECONDS, false);

objectMapper.registerModule(new JavaTimeModule());

calendarIds = objectMapper.readValue(resourceAsStream,

new TypeReference<Map<String, CalendarInfo>>() {});

}}

……………………………………………………………………………………………………………….

**//DTO created**

package practice;

import java.time.LocalDateTime;

public class Appointment {

private String id; //": "2faea810-8e14-4d45-b075-4d78528183f3",

private String calendar\_id;//": "452dccfc-975e-11e5-bfa5-c8e0eb18c1e9",

private LocalDateTime start;//": "2019-04-23T08:00:00",

private LocalDateTime end;//": "2019-04-23T08:30:00",

private String time\_slot\_type\_id;//": "4529821e-975e-11e5-bbaf-c8e0eb18c1e9",

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getCalendar\_id() {

return calendar\_id;

}

public void setCalendar\_id(String calendar\_id) {

this.calendar\_id = calendar\_id;

}

public LocalDateTime getStart() {

return start;

}

public void setStart(LocalDateTime start) {

this.start = start;

}

public LocalDateTime getEnd() {

return end;

}

public void setEnd(LocalDateTime end) {

this.end = end;

}

public String getTime\_slot\_type\_id() {

return time\_slot\_type\_id;

}

public void setTime\_slot\_type\_id(String time\_slot\_type\_id) {

this.time\_slot\_type\_id = time\_slot\_type\_id;

}

// "patient\_id": "1cfeee58-9751-11e5-9c8d-c8e0eb18c1e9",

// "patient\_comment": null,

// "note": null,

// "type\_id": null,

// "state": 0,

// "out\_of\_office\_location": "",

// "out\_of\_office": false,

// "completed": true,

// "is\_scheduled": false

}

…………………………………………………………………………………………………………….

package practice;

import java.util.Map;

class CalendarIds {

Map<String, CalendarInfo> calenderInfo;

}

………………………………………………………………………………………………………………..

package practice;

import java.util.List;

public class CalendarInfo {

List<Appointment> appointments;

List<TimeSlot> timeslots;

public List<Appointment> getAppointments() {

return appointments;

}

public void setAppointments(List<Appointment> appointments) {

this.appointments = appointments;

}

public List<TimeSlot> getTimeslots() {

return timeslots;

}

public void setTimeslots(List<TimeSlot> timeslots) {

this.timeslots = timeslots;

}

}

…………………………………………………………………………………………………..

package practice;

import java.time.LocalDateTime;

class Interval{

LocalDateTime start;

LocalDateTime end;

public Interval(LocalDateTime start, LocalDateTime end) {

this.start = start;

this.end = end;

}

public LocalDateTime getStart() {

return start;

}

public void setStart(LocalDateTime start) {

this.start = start;

}

public LocalDateTime getEnd() {

return end;

}

public void setEnd(LocalDateTime end) {

this.end = end;

}

}

…………………………………………………………………………………………………….

package practice;

import java.time.Instant;

public class TimePeriod {

Instant startTime;

Instant endTime;

}

…………………………………………………………………………………………………….

package practice;

import java.time.LocalDateTime;

public class TimeSlot {

private String id;//": "2dfb4ae6-6715-4e19-b0d5-a14e5b62ef93"

private String calendar\_id;//": "48644c7a-975e-11e5-a090-c8e0eb18c1e9",

//type\_id": "452935de-975e-11e5-ae1a-c8e0eb18c1e9",

private LocalDateTime start;//": "2019-04-23T08:00:00",

private LocalDateTime end;//": "2019-04-23T08:15:00",

private boolean public\_bookable;//": true,

private boolean out\_of\_office;//": false

public TimeSlot() {

}

public TimeSlot(LocalDateTime start, LocalDateTime end) {

this.start = start;

this.end = end;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getCalendar\_id() {

return calendar\_id;

}

public void setCalendar\_id(String calendar\_id) {

this.calendar\_id = calendar\_id;

}

public LocalDateTime getStart() {

return start;

}

public void setStart(LocalDateTime start) {

this.start = start;

}

public LocalDateTime getEnd() {

return end;

}

public void setEnd(LocalDateTime end) {

this.end = end;

}

public boolean isPublic\_bookable() {

return public\_bookable;

}

public void setPublic\_bookable(boolean public\_bookable) {

this.public\_bookable = public\_bookable;

}

public boolean isOut\_of\_office() {

return out\_of\_office;

}

public void setOut\_of\_office(boolean out\_of\_office) {

this.out\_of\_office = out\_of\_office;

}

@Override

public String toString() {

return "TimeSlot{" +

"start=" + start +

", end=" + end +

'}';

}

}