Answers for Screening Test.

1. Java script

1.1 Extend JS Date object with a method daysTo() which returns number of complete days between any pair of JS date objects: d1.daysTo(d2) should return quantity of complete days from d1 to d2.

```
Date.prototype.daysTo = function(second_day){
  const msforDay = 24* 60 * 60 * 1000;  // milliseconds for one day
  const date_diff = second_day - this;
  if (date_diff < 0)
    return -1;       // if the second_day is earlier than the current day, return -1
  return Math.floor(date_diff / msforDay);  // used floor to get lower integer value
};
// inputs
const d1 = new Date('2024-12-15');
const d2 = new Date('2024-12-10');
console.log(d1.daysTo(d2));
```

1.2 Please order by Total Develop a program which produces ordered array of sales. Input: array of objects with the following structure {amount: 10000, quantity: 10}. Output: new array of ordered sales. Array element structure should be: {amount: 10000, quantity: 10, Total: 100000}, where Total = amount * quantity. Please order by Total and note that input array shall remain intact.

```
// input array
const ordered = [
  {amount:1000, quantity:10},
  {amount:2000, quantity:5},
  {amount:1500, quantity:8},
  {amount:3000, quantity:3},
  {amount:2500, quantity:7}
]
const arrayWithTotal = []   // initiate new array
for(let i = 0; i < ordered.length; i++){
  const {amount, quantity} = ordered[i]        // get variable
  const total = amount * quantity
  arrayWithTotal.push({...ordered[i], total})  // assign to array with total
};
orderedSales = arrayWithTotal.sort((a,b)=> b.total - a.total); // sort descending by total
// console output
console.log(arrayWithTotal);
console.log(orderedSales);
```

1.3 Develop a program “Object Projection”. Input: any JSON object; prototype object. Output: projected object. Projected object structure shall be intersection of source object and prototype object structures. Values of properties in projected object shall be the same as values of respective properties in source object.

```
function projectObject(src,proto) {
   const result = {};
   for (let key in proto) {
      if(src.hasOwnProperty(key)){          // checking the selected property is property of the source
         if (typeof proto[key] === 'object' && proto[key] !== null){        // checking proto value is not null and weather it is an object
            if(src[key] && src[key]=== 'object' && src[key] !== null){   // checking source value is not null and weather it is an object
               result[key] = projectObject(src[key],proto[key]);       // continue as neasted loop
         } else {
            result[key] = src[key]   // assign values to the 'result' object
           };
      };
   }
}
   return result;
};
// input objects
const src = {
   prop11:{
      prop21: 21,
      prop22:{
         prop31: 31,
         prop32: 32
      }
   },
   prop12 : 12
};
const proto = {
   prop11:{
      prop22: null
   }
};
const resultObject = projectObject(src,proto);   //test case
// console output
console.log(resultObject);
```

2. REST API

Objective:
Retrieve free/busy intervals for a shared Google over a specified time using REST API calls

Step 1: Authentication

- Enable Google Calander API on Google Cloud Console
- Create credentials: select OAuth 2.0 Client IDs , Specify the application type

API Keys

Name	Creation date	Restrictions	Actions
API key 1	Dec 17, 2024	HTTP referrers, 1 API	SHOW KEY

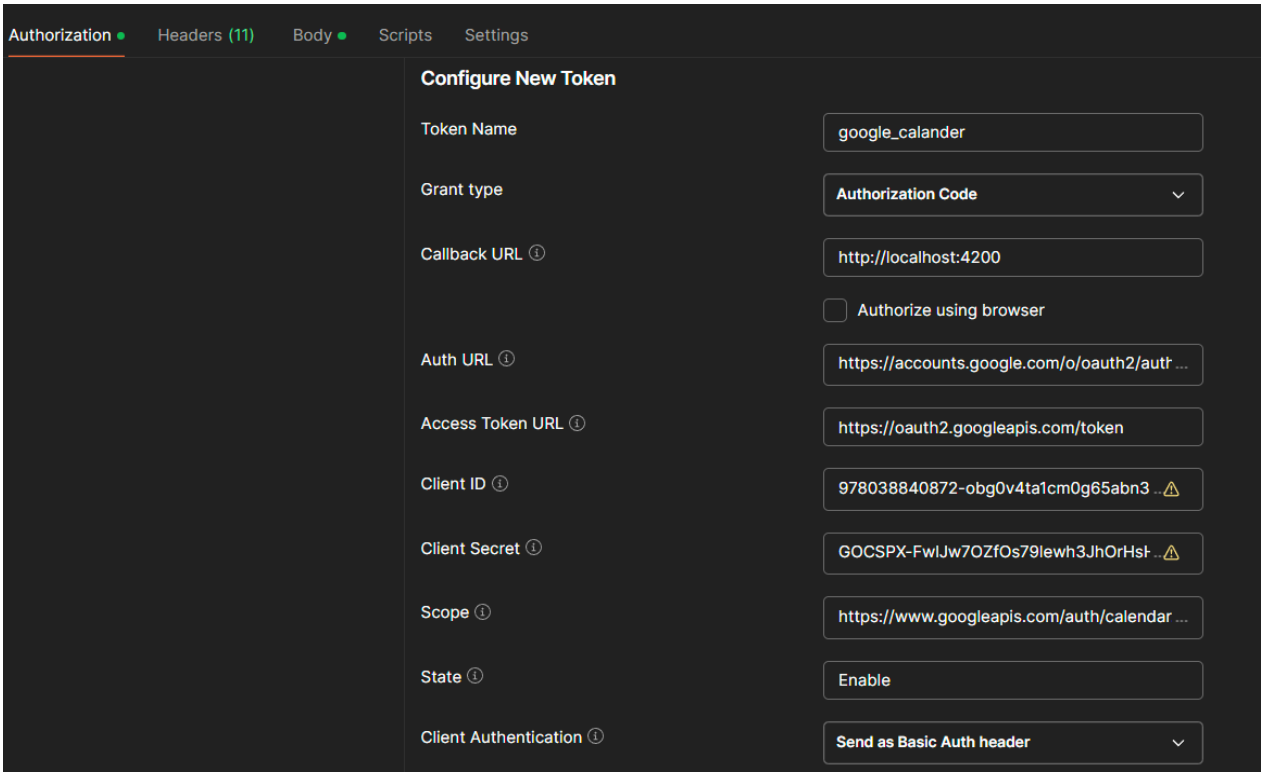
OAuth 2.0 Client IDs

Name	Creation date	Type	Client ID	Actions
demo-calander	Dec 17, 2024	Web application	978038840872-obg0...	

Service Accounts

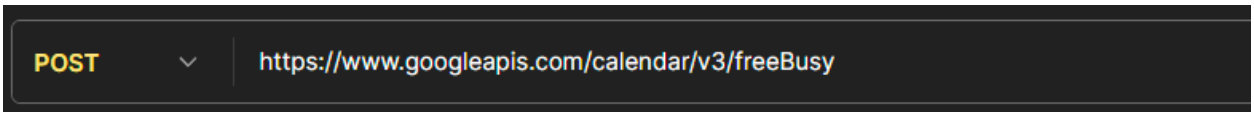
Manage service accounts

- Setup OAuth in postman (Authorization tab)
    - Header Prefix - Bearer

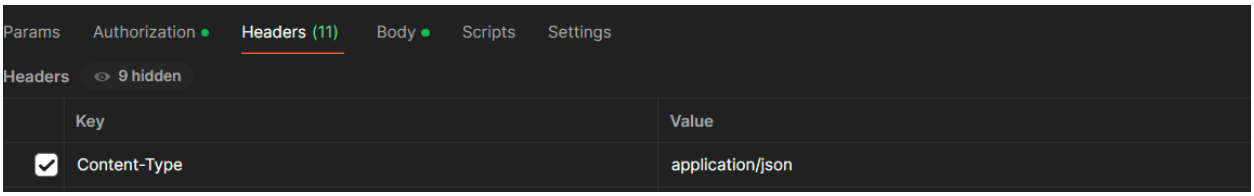


Step 2: Make a API Request

- Create new request in postman
  - Method – POST
  - URL - https://www.googleapis.com/calendar/v3/freeBusy

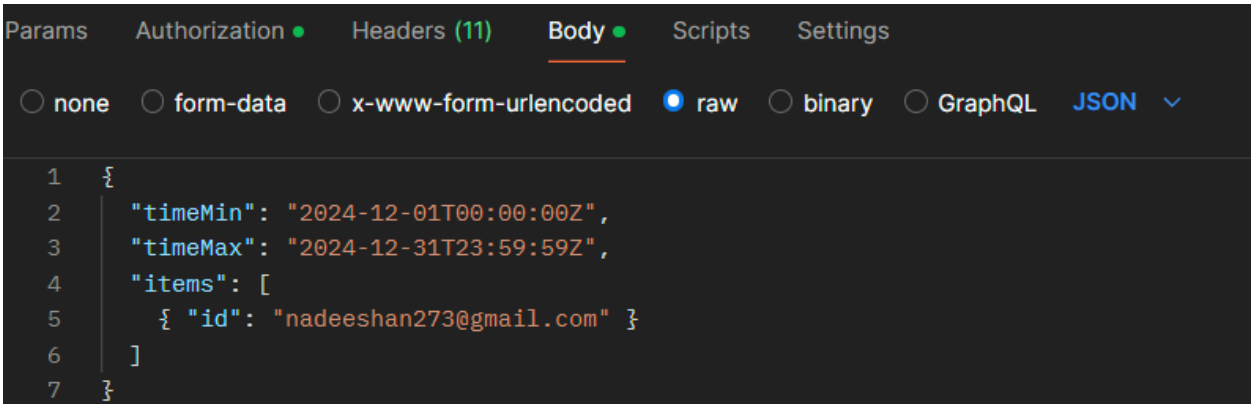


- Set the Headers
  Content – type : application/json



- Define the Request Body

```
{
 "timeMin": "2024-12-18T00:00:00Z",
 "timeMax": "2024-12-19T00:00:00Z",
 "items": [
   { "id": "nadeeshan273@gmail.com" }
 ]
}
```



- After send the request response will provide as a array.

Body Cookies Headers (16) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "kind": "calendar#freeBusy",
    "timeMin": "2024-12-01T00:00:00.000Z",
    "timeMax": "2024-12-31T23:59:59.000Z",
    "calendars": {
        "nadeeshan273@gmail.com": {
            "busy": [
                {
                    "start": "2024-12-25T18:30:00Z",
                    "end": "2024-12-26T18:30:00Z"
                }
            ]
        }
    }
}
```

3. SQL

3.1 Create tables and insert data

```
CREATE TABLE user(
  id INT,
  firstName VARCHAR(255),
  lastName VARCHAR(255),
  email VARCHAR(255),
  cultureID INT,
  deleted BIT,
  country VARCHAR(255),
  isRevokeAccess BIT,
  created DATETIME
);
```

```
INSERT INTO user
VALUES   (1, 'Victor', 'Shevchenko', 'vs@ gmail.com', 1033, 1, 'US', 0, '2011-04-05'),
         (2, 'Oleksandr', 'Petrenko', 'op@ gmail.com', 1034, 0, 'UA', 0, '2014-05-01'),
         (3, 'Victor', 'Tarasenko', 'vt@gmail.com', 1033, 1, 'US', 1, '2015-07-03'),
         (4, 'Sergiy', 'Ivanenko', 'sergiy@gmail.com', 1046, 0, 'UA', 1, '2010-02-02'),
         (5, 'Vitalii', 'Danilchenko', 'shumko@ gmail.com', 1031, 0, 'UA', 1, '2014-05-01'),
         (6, 'Joe', 'Dou', 'joe@ gmail.com', 1032, 0, 'US', 1, '2009-01-01'),
         (7, 'Marko', 'Polo', 'marko@gmail.com', 1033, 1, 'UA', 1, '2015-07-03');
```

```
CREATE TABLE `group` (
   id INT,
   name VARCHAR(255),
   created DATETIME
);
```

```
INSERT INTO `group`
VALUES (10, 'Support', '2010-02-02'),
   (12, 'Dev team', '2010-02-03'),
   (13, 'Apps team', '2011-05-06'),
   (14, 'TEST- dev team', '2013-05-06'),
   (15, 'Guest', '2014-02-02'),
   (16, 'TEST-QA-team', '2014-02-02'),
```

```
CREATE TABLE  groupMembership (
   id INT,
   userId INT,
   groupId INT,
   created DATETIME
);
```

```
INSERT INTO groupMembership
VALUES (110, 2, 10, '2010-02-02'),
   (112, 3, 15, '2010-02-03'),
   (114, 1, 10, '2014-02-02'),
   (115, 1, 17, '2011-05-02'),
   (117, 4, 12, '2014-07-13'),
   (120, 5, 15, '2014-06-15');
```

3.2 Select names of all empty test groups (group name starts with “TEST-”).

Query - SELECT name FROM `group` g
WHERE name LIKE 'TEST-%' AND
id NOT IN (SELECT groupID FROM groupmembership);

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
TEST-dev team
TEST-QA-team

3.3 Select user first names and last names for the users that have Victor as a first name and are not members of any test groups (they may be members of other groups or have no membership in any groups at all).

Query -
```
SELECT firstName , lastName FROM user
    WHERE firstName = 'Victor' AND id  IN
        (SELECT userID FROM groupmembership
            WHERE groupID IN (SELECT id FROM `group`
                WHERE name NOT LIKE 'TEST-%'));
```

	firstName	lastName
▶	Victor	Shevchenko
	Victor	Tarasenko

3.4 Select users and groups for which user was created before the group for which he(she) is member of.

Queary -
```
SELECT u.firstName,u.lastName ,g.name
FROM user u
JOIN groupmembership gm ON u.id = gm.userId
JOIN `group` g ON gm.groupID = g.id
WHERE g.created > u.created;
```

Result Grid | Filter Rows:

	firstName	lastName	name
▶	Sergiy	Ivanenko	Dev team

(TRY-SQL didn’t allow to create table because of that used mysql workbench to implement sql queries)