Software Engineer Test Solutions

Task 1: JavaScript Solutions

1.1 Extend JS Date Object

The solution adds a `daysTo` method to the Date prototype to calculate the number of full days between two dates.

Code:

```
Date.prototype.daysTo = function (date) {
    const oneDay = 1000 * 60 * 60 * 24;
    const diffDays = Math.round(
      Math.abs((date.getTime() - this.getTime()) / oneDay)
    );

    return diffDays;
  };

const d1 = new Date("2024-12-01");
const d2 = new Date("2024-12-12");

console.log(
    `Days between ${d1.toLocaleDateString()} and
${d2.toLocaleDateString()} :`,
    d1.daysTo(d2)
  );
// Output: 11
```

1.2 Order by Total

A function that takes an array of sales objects, computes the total for each, and sorts the array based on the total values in descending order.

Code:

```
const orderSales = (sales) => {
    const salesWithTotal = sales.map((sale) => ({
      ...sale,
      Total: sale.amount * sale.quantity,
    }));

    salesWithTotal.sort((a, b) => a.Total - b.Total);

    return salesWithTotal;
  };
  const sales = [
    { amount: 10000, quantity: 10 },
    { amount: 5000, quantity: 15 },
  ];

  const orderedSales = orderSales(sales);

  console.log("Original Sales Array: ", sales);
  console.log("Ordered  Sales Array: ", orderedSales);

output

Original Sales Array:  [ { amount: 10000, quantity: 10 }, { amount: 5000,
quantity: 15 } ]
Ordered  Sales Array:  [
  { amount: 5000, quantity: 15, Total: 75000 },
  { amount: 10000, quantity: 10, Total: 100000 }
]
```

1.3 Object Projection

A function that creates a new object with properties that exist in both a source object and a prototype object.

Code:

```
const projectObject = (src, prototype) => {
  const results = {};

  for (let key in prototype) {
    if (src.hasOwnProperty(key)) {
      results[key] = src[key];
    }
  }

  return results;
};

const src = {
  prop11: {
    prop21: 21,
    prop22: {
      prop31: 31,
      prop32: 32,
    },
  },
  prop12: 12,
};

const proto = {
  prop11: {
    prop22: null,
  },
};

const res = projectObject(src, proto);

console.log("res: ", res);

//Output

res:  { prop11: { prop21: 21, prop22: { prop31: 31, prop32: 32 } } }
```

Task 2: REST API Task

2.1 This task involves creating a JavaScript program to return an array of free/busy intervals for a shared Google Calendar within a specified time period.

Code Solution

Below is the JavaScript program to fetch free/busy intervals using the **Google Calendar API**:

```
const { google } = require("googleapis");

const oauth2Client = new google.auth.OAuth2(
    "31007035729-
ihij20jvhrmiveje2qq1sf4anjh35jts.apps.googleusercontent.com",
    "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",
    "http://localhost:3000"
);

oauth2Client.setCredentials({
    access_token:
"ya29.a0ARW5m75lozDPIYD9dQsKVCiEqYplGBxBsGraUurYY0Kbsao-
z2xL5zqdb61b8jjHYIz88f66qxBdivPk3C69-
DjcggAUSpBOxaTVwsn_yxK018wpnlnsBbhmjQPJiRLBui2eX_Eqc8Uas153iOW6BI9LeAUcRb
ZIuG1f9TfLaCgYKAfwSARESFQHGX2MiYGgtIDyP_YJaJlka_6qFwA0175",
});
```

```
async function getBusyIntervals() {
    const calendar = google.calendar({ version: "v3", auth: oauth2Client
});

    try {
        const response = await calendar.freebusy.query({
            requestBody: {
                timeMin: "2024-12-13T8:00:00Z",
                timeMax: "2024-12-13T11:59:59Z",
                items: [{ id: "2018a151@gmail.com" }],
            },
        });

        const busyIntervals =
response.data.calendars["2018a151@gmail.com"].busy;

        console.log("Array of busy intervals:", busyIntervals);
    } catch (error) {
        console.error("Error fetching busy intervals:", error.message);
    }
}

getBusyIntervals();
```

Sequence of REST API Calls

If the program is too complex, the same result can be achieved using the following REST API calls:

- **Endpoint**:
  `POST https://www.googleapis.com/calendar/v3/freeBusy`
- **Headers**:

  Authorization: Bearer
  <ya29.a0ARW5m75lozDPIYD9dQsKVCiEqYplGBxBsGraUurYY0Kbsao-z2xL5zqdb61b8jjHYIz88f66qxBdivPk3C69-DjcggAUSpBOxaTVwsn_yxK018wpnlnsBbhmjQPJiRLBui2eX_Eqc8Uas153iOW6BI9LeAUcRbZIuG1f9TfLaCgYKAfwSARESFQHGX2MiYGgtIDyP_YJaJlka_6qFwA0175>

  Content-Type: application/json

- **Body**:

```
{
{
  "timeMin": "2024-12-01T00:00:00Z",
  "timeMax": "2024-12-31T23:59:59Z",
  "items": [
    {"id": "2018a151@gmail.com"}
  ]
}
```

- **Response**

```
 {

    "kind": "calendar#freeBusy",
    "timeMin": "2024-12-13T00:00:00.000Z",
    "timeMax": "2024-12-13T23:59:59.000Z",
    "calendars": {
        "2018a151@gmail.com": {
            "busy": []
        }
    }
}
```

OAuth Token Details: Token Name: my app calendar Access Token:

ya29.a0ARW5m75lozDPIYD9dQsKVCiEqYplGBxBsGraUurYY0Kbsao-z2xL5zqdb61b8jjHYIz88f66qxBdivPk3C69DjcggAUSpBOxaTVwsn_yxK018wpnlnsBbhmjQPJiRLBui2eX_Eqc8Uas153iOW6BI9LeAUcRbZIuG1f9TfLaCgYKAfwSARESFQHGX2MiYGgtIDyP_YJaJlka_6qFwA0175 Authorization Header: Bearer <Access_Token>

Token Request URL: https://oauth2.googleapis.com/token
Authorization URL: https://accounts.google.com/o/oauth2/auth
Redirect URI: https://oauth.pstmn.io/v1/callback
Client ID: 31007035729-ihij20jvhrmiveje2qq1sf4anjh35jts.apps.googleusercontent.com
Client Secret: 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
Scope: https://www.googleapis.com/auth/calendar.readonly

Task 3: SQL Solutions

3.1 Create Tables and Insert Data

Scripts to create `user`, `group`, and `groupMembership` tables and populate them with sample data.

Code:

```
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'),
(17, 'TEST-team', '2011-01-07');

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 Empty Test Groups

A query to find groups starting with `TEST-` that have no members.

Code:

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

Output:

```
name
TEST-QA-team
```

3.3 Select Specific Users

Select `firstName` and `lastName` of users with the first name `"Victor"` who:

- Are not members of any `"TEST-"` groups.
- May be members of other groups or have no membership in any group.

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

Evaluation:

1. This query correctly filters users with the first name `"Victor"`.
2. The nested `NOT IN` ensures that the user's `id` does not appear in any group membership linked to `"TEST-"` groups.

Output:

```
firstName  lastName

Victor     Tarasenko
```

Task 3.4: Select Users Created Before Their Group

Select users and groups for which the user's `created` date is earlier than the group's `created` date.

Query:

```
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 u.created < g.created;
```

Evaluation:

1. The query joins the `user`, `groupMembership`, and `group` tables, linking users to their group memberships and groups.
2. The `WHERE u.created < g.created` condition ensures only users created before their group are selected.

Output:

```
firstName  lastName    name

Sergiy     Ivanenko    Dev team
```

Final Review of Provided Outputs

Task	Output	Status
3.2	`TEST-QA-team`	**succeed**
3.3	`Victor Tarasenko`	**succeed**
3.4	`Sergiy Ivanenko, Dev team`	**succeed**