1. Java Script

////Q1.1 : Extend Date object with daysTo() method to calculate the to number of days between two given dates

```
Date.prototype.daysTo = function (otherDate) {

    const date1 = new Date(this.getFullYear(), this.getMonth(), this.getDate());
    const date2 = new Date(otherDate.getFullYear(), otherDate.getMonth(), otherDate.getDate());

    const diffInMilliseconds = Math.abs(date2 - date1);

    const millisecondsInADay = 1000 * 60 * 60 * 24;
    const completeDays = Math.floor(diffInMilliseconds / millisecondsInADay);

    return completeDays;
  };
  const d1 = new Date('2022-04-18');
  const d2 = new Date('2023-05-22');
  // console.log(d1.daysTo(d2)); // 399
  document.write('<h2>'+'Q1.1 : Quantity of complete days from date1 to date2 is : ' +
d1.daysTo(d2)+'</h2>');
```

////Q1.2 : Function to calculate the Total from Input array of sales is sorted order

```
// Input array
const sales = [
  { amount: 10000, quantity: 10 },
  { amount: 5000, quantity: 2 },
  { amount: 3000, quantity: 5 },
  { amount: 2000, quantity: 3 },
  { amount: 8000, quantity: 1 }
];


// New array with Total sales
const orderedSales = sales.map(sale => {
  return {
    amount: sale.amount,
    quantity: sale.quantity,
    Total: sale.amount * sale.quantity
  };
});
// Sorting the new array
orderedSales.sort((a, b) => a.Total - b.Total);
const jsonSales = JSON.stringify(orderedSales);

document.write('<h2>'+'Q1.2 : Array with total property sorted by total sales : '+ '</h2>');

for (let i = 0; i < orderedSales.length; i++) {
  document.write('<h3>'+ JSON.stringify(orderedSales[i])+'</h3>');
}
```

// //Q1.3   : Object Projection

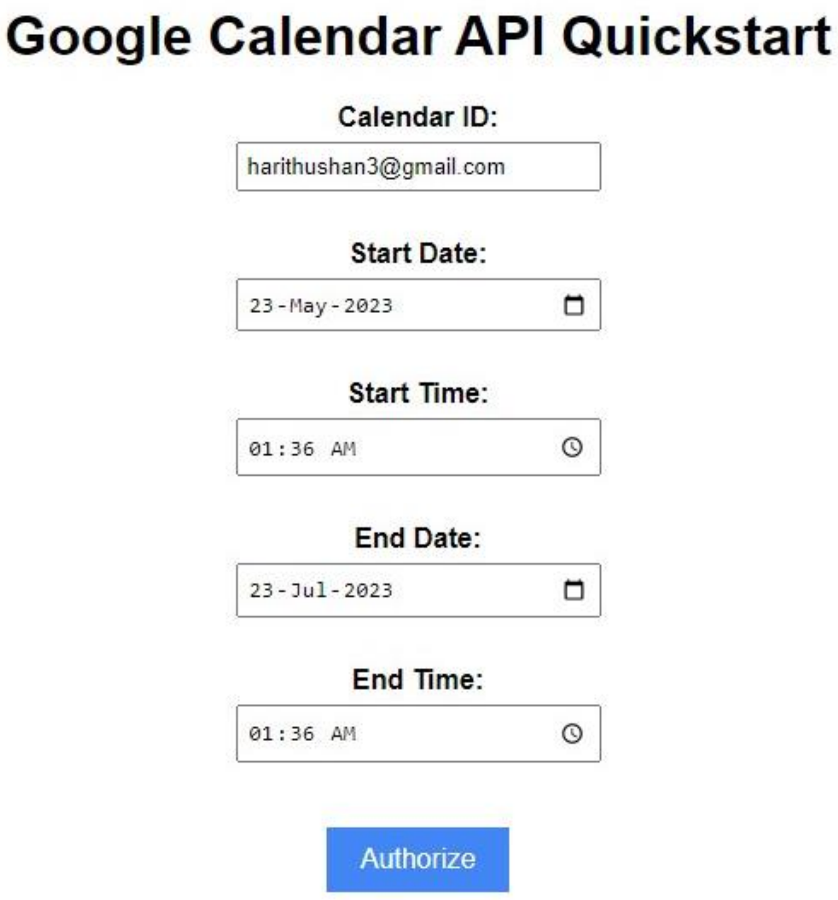
```
function projectObject(source, prototype) {
  // Check if the source and prototype are both objects
  if (typeof source !== 'object' || typeof prototype !== 'object') {
    return {};
  }
  const projected = {};

  // Iterate over prototype object
  for (const key in prototype) {
    // if the property exists in the source object
    if (key in source) {
      if (typeof prototype[key] === 'object' && prototype[key] !== null) {
        const projectedValue = projectObject(source[key], prototype[key]);
        // adding the projected property only if it has at least one property
        if (Object.keys(projectedValue).length > 0) {
          projected[key] = projectedValue;
        }
      } else {
        // If the property is not an object or is null, assign its value from the source object
        projected[key] = source[key];
      }
    }
  }
  return projected;
}

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

// Input proto object
const proto = {
  prop11: {
    prop22: null,
  },
};
```

```
const projectedObject = projectObject(src, proto);
const jsontext = JSON.stringify(projectedObject);
document.write('<h2>'+'Q1.3  Projected object res : '+ '</h2>');
document.write(jsontext);
```

2. REST API

Github repository :https://github.com/harithushan/Google_calendar_API_automation

The following code returns the busy periods in a table after authorization


Google Calendar API Quickstart

Calendar ID: harithushan3@gmail.com

Start Date: 23-May-2023

Start Time: 01:36 AM

End Date: 23-Jul-2023

End Time: 01:36 AM

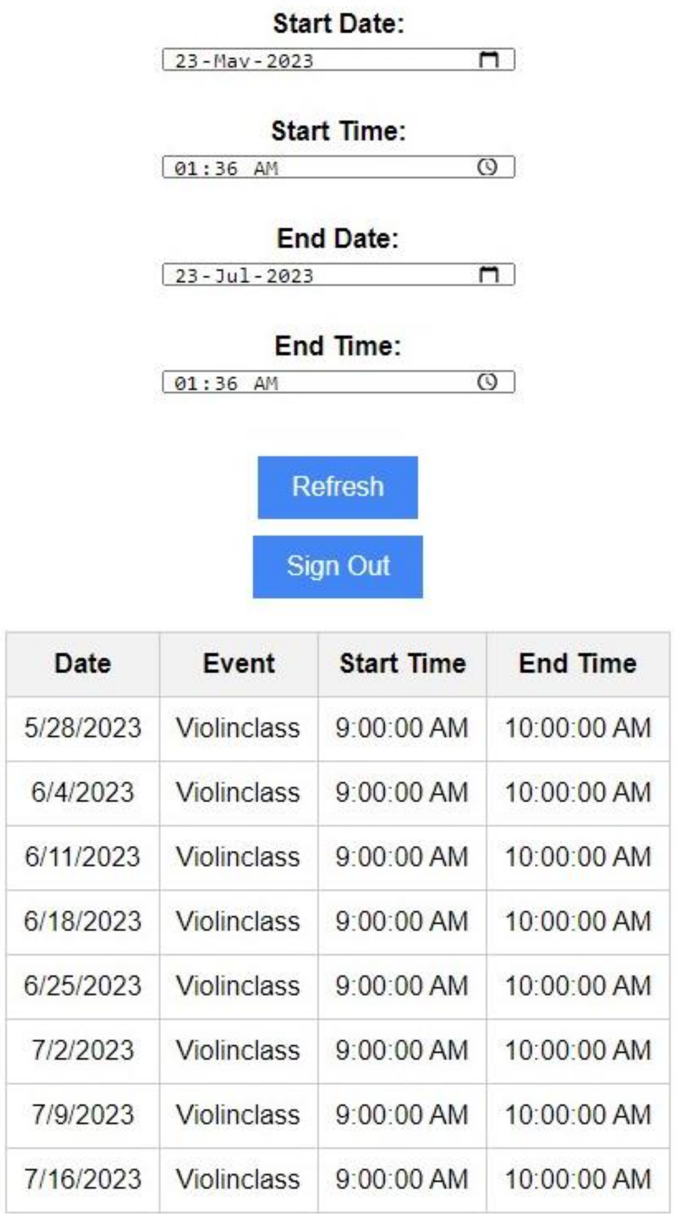
CODE :

```
<!DOCTYPE html>
<html>
<head>
  <title>Google Calendar API Quickstart</title>
  <meta charset="utf-8" />
  <style>
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      flex-direction: column;
      font-family: Arial, sans-serif;
    }

    h1 {
      margin-bottom: 20px;
    }

    label {
      font-weight: bold;
      margin-bottom: 5px;
    }

    input[type="text"],
    input[type="date"],
    input[type="time"] {
      width: 200px;
      padding: 5px;
      margin-bottom: 10px;
    }

    button {
      padding: 10px 20px;
      background-color: #4285f4;
      border: none;
      color: #fff;
      font-size: 16px;
      cursor: pointer;
      margin-top: 10px;
    }

    #busy_periods {
      margin-top: 20px;
      text-align: center;
      border-collapse: collapse;
    }

    #busy_periods th,
    #busy_periods td {
      padding: 10px;
      border: 1px solid #ccc;
    }

    #busy_periods th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>
```

```
  <h1>Google Calendar API Quickstart</h1>

  <label for="calendar_id">Calendar ID:</label>
  <input type="text" id="calendar_id" required><br>

  <label for="start_date">Start Date:</label>
  <input type="date" id="start_date" required><br>

  <label for="start_time">Start Time:</label>
  <input type="time" id="start_time" required><br>

  <label for="end_date">End Date:</label>
  <input type="date" id="end_date" required><br>

  <label for="end_time">End Time:</label>
  <input type="time" id="end_time" required><br>

  <button id="authorize_button" onclick="handleAuthClick()">Authorize</button>
  <button id="signout_button" onclick="handleSignoutClick()" style="margin-top: 10px;">Sign
Out</button>

  <table id="busy_periods" style="margin-top: 20px;"></table>

  <script type="text/javascript">
    /* exported gapiLoaded */
    /* exported gisLoaded */
    /* exported handleAuthClick */
    /* exported handleSignoutClick */

    // TODO(developer): Set to client ID and API key from the Developer Console
    const CLIENT_ID = '949300763919-
o6i0m9jdcobh9903pvvrdpan5qv9nlnh.apps.googleusercontent.com';
    const API_KEY = 'AIzaSyDbq7hHbfEbMbKa_3QsKaaBmDWpkwZ1j30';

    // Discovery doc URL for APIs used by the quickstart
    const DISCOVERY_DOC = 'https://www.googleapis.com/discovery/v1/apis/calendar/v3/rest';

    // Authorization scopes required by the API; multiple scopes can be
    // included, separated by spaces.
    const SCOPES = 'https://www.googleapis.com/auth/calendar.readonly';

    let tokenClient;
    let gapiInited = false;
    let gisInited = false;

    document.getElementById('authorize_button').style.visibility = 'hidden';
    document.getElementById('signout_button').style.visibility = 'hidden';
```

```
/**
 * Callback after api.js is loaded.
 */
function gapiLoaded() {
  gapi.load('client', initializeGapiClient);
}

/**
 * Callback after the API client is loaded. Loads the
 * discovery doc to initialize the API.
 */
async function initializeGapiClient() {
  await gapi.client.init({
    apiKey: API_KEY,
    discoveryDocs: [DISCOVERY_DOC],
  });
  gapiInited = true;
  maybeEnableButtons();
}

/**
 * Callback after Google Identity Services are loaded.
 */
function gisLoaded() {
  tokenClient = google.accounts.oauth2.initTokenClient({
    client_id: CLIENT_ID,
    scope: SCOPES,
    callback: '', // defined later
  });
  gisInited = true;
  maybeEnableButtons();
}

/**
 * Enables user interaction after all libraries are loaded.
 */
function maybeEnableButtons() {
  if (gapiInited && gisInited) {
    document.getElementById('authorize_button').style.visibility = 'visible';
  }
}

/**
 *  Sign in the user upon button click.
 */
function handleAuthClick() {
  tokenClient.callback = async (resp) => {
    if (resp.error !== undefined) {
      throw (resp);
    }
    document.getElementById('signout_button').style.visibility = 'visible';
    document.getElementById('authorize_button').innerText = 'Refresh';
    await listBusyPeriods();
  };

  if (gapi.client.getToken() === null) {
    // Prompt the user to select a Google Account and ask for consent to share their data
    // when establishing a new session.
    tokenClient.requestAccessToken({ prompt: 'consent' });
  } else {
    // Skip display of account chooser and consent dialog for an existing session.
    tokenClient.requestAccessToken({ prompt: '' });
  }
}

/**
 *  Sign out the user upon button click.
 */
function handleSignoutClick() {
  const token = gapi.client.getToken();
  if (token !== null) {
    google.accounts.oauth2.revoke(token.access_token);
    gapi.client.setToken('');
    document.getElementById('busy_periods').innerText = '';
    document.getElementById('authorize_button').innerText = 'Authorize';
    document.getElementById('signout_button').style.visibility = 'hidden';
  }
}

/**
 * Fetches and displays the busy periods between the specified start and end dates/times.
 */
async function listBusyPeriods() {
  const calendarId = document.getElementById('calendar_id').value;
  const startDate = document.getElementById('start_date').value;
  const startTime = document.getElementById('start_time').value;
  const endDate = document.getElementById('end_date').value;
  const endTime = document.getElementById('end_time').value;

  const startDateTime = new Date(`${startDate} ${startTime}`).toISOString();
  const endDateTime = new Date(`${endDate} ${endTime}`).toISOString();

  let response;
  try {
    const request = {
      'calendarId': calendarId,
        'timeMin': startDateTime,
        'timeMax': endDateTime,
        'showDeleted': false,
        'singleEvents': true,
        'orderBy': 'startTime',
      };
      response = await gapi.client.calendar.events.list(request);
    } catch (err) {
      console.error(err);
      return;
    }

    const events = response.result.items;
    if (!events || events.length === 0) {
      document.getElementById('busy_periods').innerText = 'No busy periods found.';
      return;
    }

    const table = document.getElementById('busy_periods');
    table.innerHTML = '';
    const headerRow = document.createElement('tr');
    const dateHeader = document.createElement('th');
    dateHeader.innerText = 'Date';
    headerRow.appendChild(dateHeader);
    const eventHeader = document.createElement('th');
    eventHeader.innerText = 'Event';
    headerRow.appendChild(eventHeader);
    const startTimeHeader = document.createElement('th');
    startTimeHeader.innerText = 'Start Time';
    headerRow.appendChild(startTimeHeader);
    const endTimeHeader = document.createElement('th');
    endTimeHeader.innerText = 'End Time';
    headerRow.appendChild(endTimeHeader);
    table.appendChild(headerRow);

    events.forEach((event) => {
      const row = document.createElement('tr');
      const dateCell = document.createElement('td');
      dateCell.innerText = new Date(event.start.dateTime || event.start.date).toLocaleDateString();
      row.appendChild(dateCell);
      const eventCell = document.createElement('td');
      eventCell.innerText = event.summary;
      row.appendChild(eventCell);
      const startTimeCell = document.createElement('td');
      startTimeCell.innerText = event.start.dateTime ? new
Date(event.start.dateTime).toLocaleTimeString() : 'All day';
      row.appendChild(startTimeCell);
      const endTimeCell = document.createElement('td');
```

```
        endTimeCell.innerText = event.end.dateTime ? new
Date(event.end.dateTime).toLocaleTimeString() : 'All day';
        row.appendChild(endTimeCell);
        table.appendChild(row);
      });
    }
  </script>
  <script async defer src="https://apis.google.com/js/api.js" onload="gapiLoaded()"></script>
  <script async defer src="https://accounts.google.com/gsi/client" onload="gisLoaded()"></script>
</body>
</html>
```


Start Date:
23-May-2023

Start Time:
01:36 AM

End Date:
23-Jul-2023

End Time:
01:36 AM

Date	Event	Start Time	End Time
5/28/2023	Violinclass	9:00:00 AM	10:00:00 AM
6/4/2023	Violinclass	9:00:00 AM	10:00:00 AM
6/11/2023	Violinclass	9:00:00 AM	10:00:00 AM
6/18/2023	Violinclass	9:00:00 AM	10:00:00 AM
6/25/2023	Violinclass	9:00:00 AM	10:00:00 AM
7/2/2023	Violinclass	9:00:00 AM	10:00:00 AM
7/9/2023	Violinclass	9:00:00 AM	10:00:00 AM
7/16/2023	Violinclass	9:00:00 AM	10:00:00 AM

3. SQL

3.1

```
# creating user table
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 the data into the table
INSERT INTO user (id, firstName, lastName, email, cultureID, deleted, country, isRevokeAccess,
created)
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');

# creating table group_table
CREATE TABLE group_table (
  id INT,
  name VARCHAR(255),
  created DATETIME
);

# Insert the data into the group_table
INSERT INTO group_table (id, name, created)
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');
```

```
# creating table groupMembership
CREATE TABLE groupMembership (
  id INT,
  userID INT,
  groupID INT,
  created DATETIME
);

# Insert the data into the groupMembership table
INSERT INTO groupMembership (id, userID, groupID, created)
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.**<u>Query1</u>**

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

3.3. **<u>Query2</u>**

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

3.4. **<u>Query3</u>**

```
SELECT u.firstName, u.lastName, g.name
FROM user u
JOIN groupMembership gm ON u.id = gm.userID
JOIN group_table g ON gm.groupID = g.id
WHERE u.created < g.created;
```