Google Sheet Connection using App Script

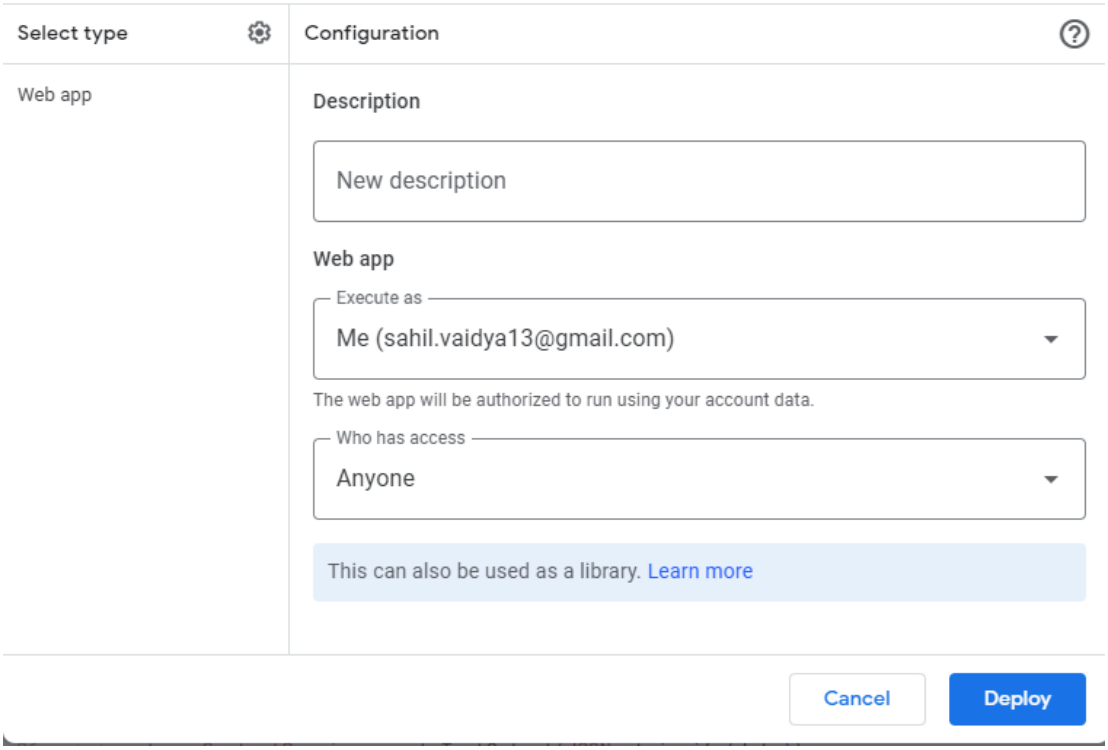
- **Initiating Google Sheet Spreadsheet**
    - Creating Spreadsheet and sharing the sheet as anyone with the link.
    - In the Extension dropdown, select AppScript.
- **Script Code and Deployment**
    - Write the following code(based on functionality needed).

```
var sheetName = 'Sheet1'; //Sheetname
var scriptProp = PropertiesService.getScriptProperties();

function setup() {
  var doc = SpreadsheetApp.getActiveSpreadsheet();
  scriptProp.setProperty('key', doc.getId());
}

function doGet(e) {

  console.log(e.parameter);

  var doc = SpreadsheetApp.openById(scriptProp.getProperty('key'));
  var sheet = doc.getSheetByName(sheetName);


  if ('row' in e.parameter) {
    var row = parseInt(e.parameter.row, 10);
    var data = sheet.getRange(row, 1, 1, sheet.getLastColumn()).getValues();
    return ContentService.createTextOutput(JSON.stringify(data))
      .setMimeType(ContentService.MimeType.JSON);
  } else {

    var data = sheet.getDataRange().getValues();
    return ContentService.createTextOutput(JSON.stringify(data))
      .setMimeType(ContentService.MimeType.JSON);
  }
}

function doPost(e) {
  var response;
  try {
    var lock = LockService.getScriptLock();
    lock.waitLock(30000);
    var scriptProp = PropertiesService.getScriptProperties();
```

```
    var doc = SpreadsheetApp.openById(scriptProp.getProperty('key'));
    var sheet = doc.getSheetByName(sheetName);
    var postData = JSON.parse(e.postData.contents);
    var operation = postData.operation;

    if (operation === "add") {
      var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];
      var rowData = [];


      for (var key in postData.data) {
        var columnIndex = headers.indexOf(key);
        if (columnIndex !== -1) {
          rowData[columnIndex] = postData.data[key];
        }
      }


      sheet.appendRow(rowData);

      response = { result: "success", action: "added", data: rowData };
    } else if (operation === "edit" && postData.row) {

      var rowNumber = postData.row;


      var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];

      var range = sheet.getRange(rowNumber, 1, 1, sheet.getLastColumn());

      var rowValues = range.getValues()[0];
      console.log(postData.data)
      for (var key in postData.data) {
        var columnIndex = headers.indexOf(key);
        if (columnIndex !== -1) {
          rowValues[columnIndex] = postData.data[key];
        }
      }

      range.setValues([rowValues]);

      response = { result: "success", action: "edited", row: rowNumber, data: postData.data
};
    } else if (operation === "delete" && postData.row) {

      var rowNumber = postData.row;

      if (rowNumber < 2 || rowNumber > sheet.getLastRow()) {

        response = { result: "error", message: "Invalid row number provided for deletion."
};
      } else {
```

```
        sheet.deleteRow(rowNumber);
        response = { result: "success", action: "deleted", row: rowNumber };
      }

    } else {
      response = { result: "error", message: "Invalid operation requested." };
    }
  } catch (e) {
    response = { result: "error", message: e.toString() };
  } finally {
    lock.releaseLock();
  }
  return ContentService.createTextOutput(JSON.stringify(response))
    .setMimeType(ContentService.MimeType.JSON);
}
```

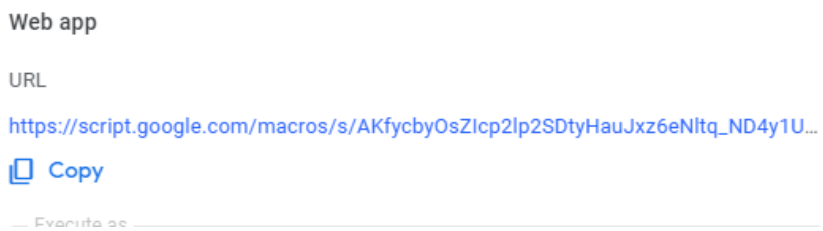
- Click on Deploy as a Web app, choose the account as **Me({email})** and later as **Anyone**.



- It will prompt for the Authorize app.

- (In some cases, it can show a back to safety page, but there click on Advanced and then click on visit App(unsafe) and then click Allow.

- Then it will provide a **URL** for performing operations on the Sheet.



- **For GET Request**

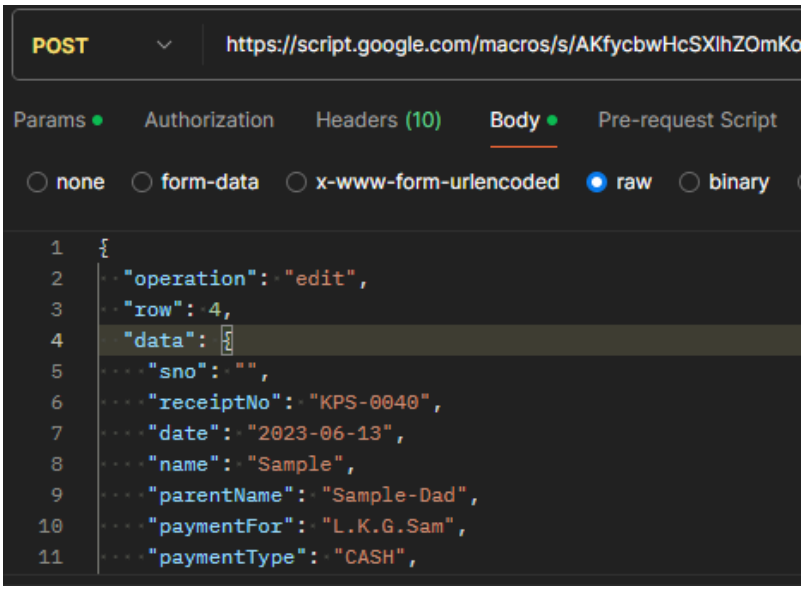
    - For the get requests, simply call using the **GET** method to the route specified.(if bulk response needed.

    - For Single row **GET** request:
        - At the end of the **URL** add the query parameter as **"?row=RowNumber"**

- **For POST Requests**

    - As we have a single function **doPost(e)** in the AppScript which handles all three operations **["add","edit","delete"]**.

    - The App Script API doesn't let us use the **DELETE** method on the objects because of security reasons, for that we use only the **POST** method.
    - The edit method asks for **row, operation** and **data object.**



- The delete method asks for only **operation** and **row**.

- The add method asks for **data,** and **operation.**

- **The API Response**

  - The AppScript URL gives a response in the form of an array containing n-number of arrays as data.
  - It doesn't provide the response as objects containing key-value pairs. The response needs to be manipulated for the desired form for the frontend.

- **Using Express**

  - For handling this different data representation and making the communication smooth, we used Express to fetch frontend requests and parse it according to the script URL API and manipulate the response from AppScript API to be in such so that the frontend can understand.
  - The Express Backend is working as a middleman in this scenario.