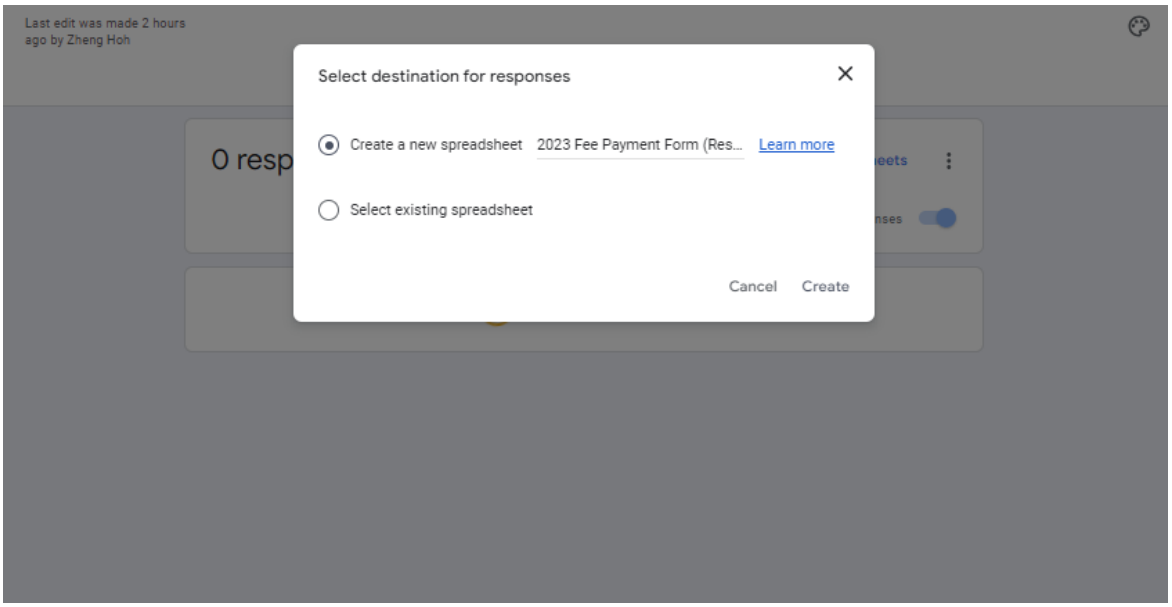
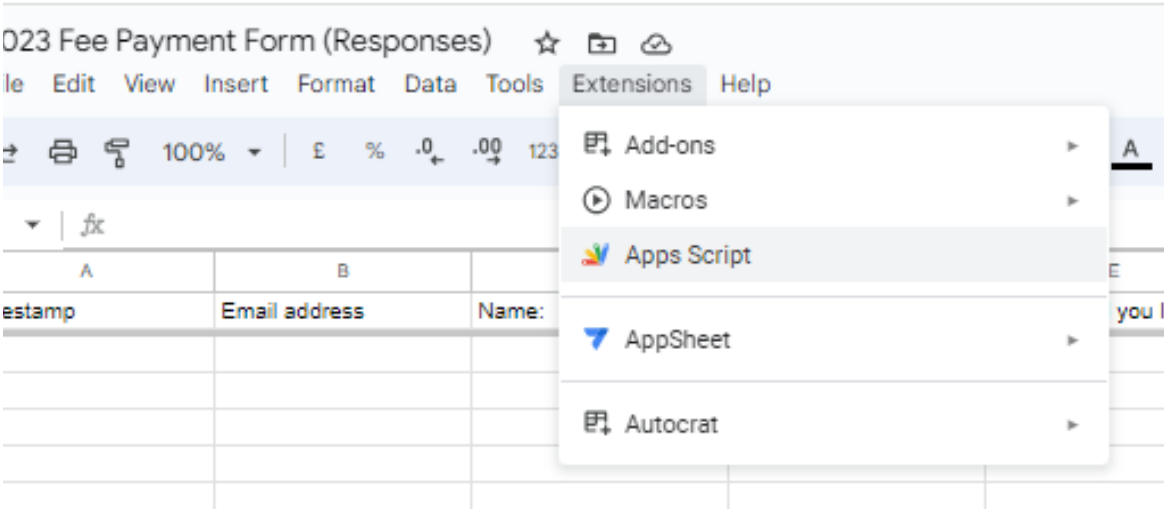
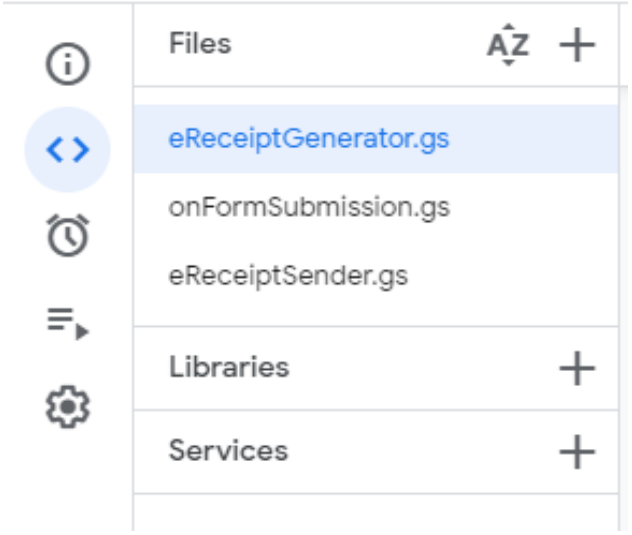
1. **Open the Google Form** and **create a linked sheet** if haven't



2. Head to ***'App Scripts'*** under ***'Extension'***



3. Create three Google Scripts of the following names.



4. Copy and paste the following codes to their respective Google Scripts.

eReceiptGenerator

```
let spreadsheetForGenerator =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
let lastColumnForGenerator =
spreadsheetForGenerator.getLastColumn();
let headersForGenerator = spreadsheetForGenerator.getRange(1, 1, 1,
lastColumnForGenerator).getValues()[0];

function eReceiptGenerator() {

  const pdfFolderForEReceipt =
DriveApp.getFolderById("1rBeH0_-pwmUL9AAU59zH_0svOqenoadf");
  const tempFolderForEReceipt =
DriveApp.getFolderById("1eAH76eRX1F0O6HDFuH8OxSgvQejFtpNn");
  const eReceiptTemplateDoc =
DriveApp.getFileById("1EAjJy_VsDXUC_1_ptZA6CkTYNigTjSIyfC_lX8tJaS4"
);



  let lastRow = spreadsheetForGenerator.getLastRow();

  for (let i = 2; i <= lastRow; i++) {
    if (!hasGeneratedEReceipt(spreadsheetForGenerator.getRange(i,
getColumnGeneratedEReceipt(headersForGenerator)).getValue()) &&
!(spreadsheetForGenerator.getRange(i,
getColumnEReceiptSent(headersForGenerator)).getValue()) &&
(isValid(spreadsheetForGenerator.getRange(i,
getColumnValidation(headersForGenerator)).getValue()))) {
      let currentDate = spreadsheetForGenerator.getRange(i,
getColumnDate(headersForGenerator)).getValue();
      let currentName = spreadsheetForGenerator.getRange(i,
getColumnName(headersForGenerator)).getValue();
      let currentPaidMonth = spreadsheetForGenerator.getRange(i,
getColumnPaidMonth(headersForGenerator)).getValue();
      let currentPaymentPurpose =
```

```
spreadsheetForGenerator.getRange(i,
getColumnPaymentPurpose(headersForGenerator)).getValue();
      let currentFeePaid = spreadsheetForGenerator.getRange(i,
getColumnFeePaid(headersForGenerator)).getValue().split(",");
      // Trim leading and trailing spaces from each element before
splitting
      currentFeePaid = currentFeePaid.map(feePaid =>
feePaid.trim());
      let currentPaymentMethod =
spreadsheetForGenerator.getRange(i,
getColumnPaymentMethod(headersForGenerator)).getValue();
      let currentRemark = spreadsheetForGenerator.getRange(i,
getColumnRemark(headersForGenerator)).getValue();

      let currentPaymentInfo = {
        'name': currentName,
        'date': formatDate(currentDate),
        'description': formatDescription(currentPaidMonth,
currentFeePaid, currentPaymentPurpose),
        'feePaid': formatFeePaid(currentPaymentPurpose,
currentFeePaid, currentPaymentMethod),
        'remarks': currentRemark
      }
      Logger.log(currentPaymentInfo);

      const newTempFileForEReceipt =
eReceiptTemplateDoc.makeCopy(tempFolderForEReceipt);

      const openEReceiptTemplateDoc =
DocumentApp.openById(newTempFileForEReceipt.getId());
      const eReceiptBody = openEReceiptTemplateDoc.getBody();

      eReceiptBody.replaceText("{name}",
currentPaymentInfo["name"]);
      eReceiptBody.replaceText("{date}",
currentPaymentInfo["date"]);
      eReceiptBody.replaceText("{description}",
currentPaymentInfo["description"]);
```

```
      eReceiptBody.replaceText("{payment}",
currentPaymentInfo["feePaid"]);
      eReceiptBody.replaceText("{remarks}",
currentPaymentInfo["remarks"]);
      openEReceiptTemplateDoc.saveAndClose();

      const blobEReceiptPDF =
newTempFileForEReceipt.getAs(MimeType.PDF);
      const pdfFileForEReceipt =
pdfFolderForEReceipt.createFile(blobEReceiptPDF).setName(`${current
PaymentInfo["name"]} | Receipt:
${currentPaymentInfo["description"]}`);
      const pdfIDForEReceipt = pdfFileForEReceipt.getId();

      spreadsheetForGenerator.getRange(i,
getColumnGeneratedEReceipt(headersForGenerator)).setValue(pdfFileFo
rEReceipt.getUrl());
      spreadsheetForGenerator.getRange(i,
getColumnGeneratedEReceiptID(headersForGenerator)).setValue(pdfIDFo
rEReceipt);
    }
    Logger.log("Done " + i);
  }
}

function hasGeneratedEReceipt(value) {
  return value != "-";
}

function isValid(value) {
  return value == "Valid";
}

function getColumnEReceiptSent(headersForGenerator) {
  return headersForGenerator.indexOf("Receipt Sent?") + 1;
}

function getColumnGeneratedEReceipt(headersForGenerator) {
```

```
  return headersForGenerator.indexOf("Generated e-Receipt") + 1;
}

function getColumnGeneratedEReceiptID(headersForGenerator) {
  return headersForGenerator.indexOf("Generated e-Receipt ID") + 1;
}

function getColumnDate(headersForGenerator) {
  return headersForGenerator.indexOf("Timestamp") + 1;
}

function getColumnEmail(headersForGenerator) {
  return headersForGenerator.indexOf("Email address") + 1;
}

function getColumnName(headersForGenerator) {
  return headersForGenerator.indexOf("Name:") + 1;
}

function getColumnPaidMonth(headersForGenerator) {
  return headersForGenerator.indexOf("Fee payment for the Month
of:") + 1;
}

function getColumnPaymentPurpose(headersForGenerator) {
  return headersForGenerator.indexOf("What would you like to pay
for?") + 1;
}

function getColumnFeePaid(headersForGenerator) {
  return headersForGenerator.indexOf("For The Payment Of: (You may
click on both membership and training fee to pay for them both) ")
+ 1;
}

function getColumnPaymentMethod(headersForGenerator) {
  return headersForGenerator.indexOf("Payment Method ") + 1;
}
```

```
function getColumnProofOfPayment(headersForGenerator) {
  return headersForGenerator.indexOf("Proof of payment upload:") +
1;
}

function getColumnValidation(headersForGenerator) {
  return headersForGenerator.indexOf("Response Validity") + 1;
}

function getColumnRemark(headersForGenerator) {
  return headersForGenerator.indexOf("Remark") + 1;
}

function formatDate(dateString) {
  var date = new Date(dateString);
  var formattedDate = Utilities.formatDate(date, "GMT+08:00",
"dd-MM-yyyy");
  return formattedDate;
}

function formatDescription(currentPaidMonth, currentFeePaid,
currentPaymentPurpose) {
  let description = "";
  if (currentPaymentPurpose == "Monthly Training Fee with/without
the Membership Fee.") {
    description = currentPaidMonth + " Training Fee ";

    for (let fee of currentFeePaid) {
      if (fee == "RM60 - Monthly Pass (UNLIMITED)") {
        description = description + "[Monthly Pass - RM 60]";
      }
      else if (fee == "RM30 - 2 Trainings Pass (2 trainings per
month only)") {
        description = description + "[2 Trainings Pass - RM 30]";
      }
    }
```

```
    if (currentFeePaid.includes("RM15 - Club Membership Fee")) {
      description = description + " and Membership Fee - RM 15";
    }
  }
  else {
    description = "Membership Fee - RM 15";
  }

  description = description + "."
  return description
}

function formatFeePaid(currentPaymentPurpose, currentFeePaid,
currentPaymentMethod) {
  if (currentPaymentPurpose == "Just the Membership Fee.") {
    return "RM 15.00";
  }

  // Define the fee amounts for each string
  const feeAmounts = {
    "RM30 - 2 Trainings Pass (2 trainings per month only)": 30,
    "RM60 - Monthly Pass (UNLIMITED)": 60,
    "RM15 - Club Membership Fee": 15
  };

  // Initialize the total amount to 0
  let totalAmount = 0;

  // Iterate through the currentFeePaid array and sum up the fee
amounts
  for (let fee of currentFeePaid) {
    if (feeAmounts.hasOwnProperty(fee)) {
      totalAmount += feeAmounts[fee];
    }
  }

  // Return the formatted fee amount
  return `RM ${totalAmount.toFixed(2)} (${currentPaymentMethod})`;
```

```
}
```

onFormSubmission

```
let spreadsheet =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
let lastColumn = spreadsheet.getLastColumn();
let headers = spreadsheet.getRange(1, 1, 1,
lastColumn).getValues()[0];

async function onFormSubmission(e) {
  await setUpHeaders();
  setUpCells(e);
}

async function setUpHeaders() {
  // Check if the columns already exist, and add them if not
  if (!hasColumnValidation(headers)) {
    lastColumn++;
    spreadsheet.insertColumns(lastColumn);
    spreadsheet.getRange(1, lastColumn).setValue("Response
Validity").setBackground("#d0e0e3");
  }

  if (!hasColumnRemark(headers)) {
    lastColumn++;
    spreadsheet.insertColumns(lastColumn);
    spreadsheet.getRange(1,
lastColumn).setValue("Remark").setBackground("#d0e0e3");
  }

  if (!hasColumnGeneratedEReceipt(headers)) {
    lastColumn++;
    spreadsheet.insertColumns(lastColumn);
    spreadsheet.getRange(1, lastColumn).setValue("Generated
e-Receipt").setBackground("#d0e0e3");
  }
```

```
  if (!hasColumnGeneratedEReceiptID(headers)) {
    lastColumn++;
    spreadsheet.insertColumns(lastColumn);
    spreadsheet.getRange(1, lastColumn).setValue("Generated
e-Receipt ID").setBackground("#d0e0e3");
  }

  if (!hasColumnReceiptSent(headers)) {
    lastColumn++;
    spreadsheet.insertColumns(lastColumn);
    spreadsheet.getRange(1, lastColumn).setValue("Receipt
Sent?").setBackground("#d0e0e3");
  }

  // Update the headers variable after adding new columns
  headers = spreadsheet.getRange(1, 1, 1,
lastColumn).getValues()[0];

}

function hasColumnReceiptSent(headers) {
  return headers.indexOf("Receipt Sent?") != -1;
}
function hasColumnValidation(headers) {
  return headers.indexOf("Response Validity") != -1;
}
function hasColumnRemark(headers) {
  return headers.indexOf("Remark") != -1;
}
function hasColumnGeneratedEReceipt(headers) {
  return headers.indexOf("Generated e-Receipt") != -1;
}
function hasColumnGeneratedEReceiptID(headers) {
  return headers.indexOf("Generated e-Receipt ID") != -1;
}

function setUpCells(triggerEvent) {
```

```
  let colOfReceiptSent = headers.indexOf("Receipt Sent?") + 1;
  let colOfValidInvalid = headers.indexOf("Response Validity") + 1;
  let colOfRemark = headers.indexOf("Remark") + 1;
  let colOfGeneratedEReceipt = headers.indexOf("Generated
e-Receipt") + 1;
  let colOfPaymentDetails = headers.indexOf("For The Payment Of:
(You may click on both membership and training fee to pay for them
both) ") + 1;
  let rowOfNewSubmission = triggerEvent.range.rowEnd;

  // Set the data validation for cell of receipt written to require
a boolean value; the value is rendered as a checkbox.
  let cellOfReceiptSent = spreadsheet.getRange(rowOfNewSubmission,
colOfReceiptSent);
  let checkboxValidationRule =
SpreadsheetApp.newDataValidation().requireCheckbox().build();
  cellOfReceiptSent.setDataValidation(checkboxValidationRule);

  // Set the data validation for cell of Response Validity to
require a a list of selections.
  let validInvalidChoices = ["-", "Valid", "Invalid"];
  let cellOfValidInvalid = spreadsheet.getRange(rowOfNewSubmission,
colOfValidInvalid);

  let valueOfPaymentDetails =
spreadsheet.getRange(rowOfNewSubmission,
colOfPaymentDetails).getValue();

  if (isPaymentDetailsValid(valueOfPaymentDetails)) {
    cellOfValidInvalid.setValue("-"); // Set default value to "-"
if Payment Details are valid
  }
  else {
    cellOfValidInvalid.setValue("Invalid"); // Set default value to
"Invalid" if Payment Details are valid
  }
  let validInvalidValidationRule =
SpreadsheetApp.newDataValidation().requireValueInList(validInvalidC
```

```
hoices).build();
  cellOfValidInvalid.setDataValidation(validInvalidValidationRule);

  // Set colors based on the value of "Validity"
  let validityValue = cellOfValidInvalid.getValue();
  if (validityValue === "Invalid") {
    cellOfValidInvalid.setBackground('#ff0000'); // Red
  }

  let cellOfRemark = spreadsheet.getRange(rowOfNewSubmission,
colOfRemark);
  cellOfRemark.setValue("-"); // Set default value to "-"

  let cellOfGeneratedEReceipt =
spreadsheet.getRange(rowOfNewSubmission, colOfGeneratedEReceipt);
  cellOfGeneratedEReceipt.setValue("-"); // Set default value to
"-"
}

function isPaymentDetailsValid(paymentDetails) {
  // Return true if both strings are present in the array,
otherwise false
  let paymentDetailsArr = paymentDetails.split(",");
  // Trim leading and trailing spaces from each element before
splitting
  paymentDetailsArr = paymentDetailsArr.map(pd => pd.trim());
  return !(paymentDetailsArr.includes("RM30 - 2 Trainings Pass (2
trainings per month only)") && paymentDetailsArr.includes("RM60 -
Monthly Pass (UNLIMITED)"));
}

function onValidInvalidEdit(triggerEvent) {
  let range = triggerEvent.range;
  let colOfValidInvalid = headers.indexOf("Response Validity") + 1;

  // Check if the edited cell is within the Validity column
  if (range.getColumn() === colOfValidInvalid) {
    let cellValue = range.getValue();
```

```
    if (cellValue === "Valid") {
      range.setBackground('#00ff00');
    } else if (cellValue === "Invalid") {
      range.setBackground('#ff0000');
    }
    else {
      range.setBackground('#ffffff');
    }
  }
}
```

eReceiptSender

```
let spreadsheetForSender =
SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();
let lastColumnForSender = spreadsheet.getLastColumn();
let headersForSender = spreadsheet.getRange(1, 1, 1,
lastColumnForSender).getValues()[0];

function eReceiptSender() {
  let lastRow = spreadsheetForSender.getLastRow();

  for (let i = 2; i <= lastRow; i++) {
    if (hasGeneratedEReceipt(spreadsheetForSender.getRange(i,
getColumnGeneratedEReceipt(headersForSender)).getValue()) &&
!(spreadsheetForSender.getRange(i,
getColumnEReceiptSent(headersForSender)).getValue()) &&
(isValid(spreadsheetForSender.getRange(i,
getColumnValidation(headersForSender)).getValue()))) {
      let currentEmail = spreadsheetForSender.getRange(i,
getColumnEmail(headersForSender)).getValue();
      let currentName = spreadsheetForSender.getRange(i,
getColumnName(headersForSender)).getValue();
      let currentPaidMonth = spreadsheetForSender.getRange(i,
getColumnPaidMonth(headersForSender)).getValue();
      let currentPaymentPurpose = spreadsheetForSender.getRange(i,
getColumnPaymentPurpose(headersForSender)).getValue();
```

```
      let currentFeePaid = spreadsheetForSender.getRange(i,
getColumnFeePaid(headersForSender)).getValue().split(",");
      // Trim leading and trailing spaces from each element before
splitting
      currentFeePaid = currentFeePaid.map(feePaid =>
feePaid.trim());

      const pdfFileID = spreadsheetForSender.getRange(i,
getColumnGeneratedEReceiptID(headersForSender)).getValue();
      const pdfFileForEReceipt = getFileFromDriveById(pdfFileID);



      GmailApp.sendEmail(currentEmail, currentName + " | " +
formatDescription(currentPaidMonth, currentFeePaid,
currentPaymentPurpose), 'Thank you for your payment! Please see the
attached file.', {
        attachments: [pdfFileForEReceipt],
        name: 'MTC e-Receipt Engine'
      });

      // Set e-Receipt sent to True
      spreadsheetForSender.getRange(i,
getColumnEReceiptSent(headersForSender)).setValue(true);

    }
  }
}

function getFileFromDriveById(fileId) {
  try {
    const file = DriveApp.getFileById(fileId);
    return file;
  } catch (error) {
    // Handle the case if the file is not found or there's an error
    console.error("File not found or error:", error);
    return null;
  }
}
```

```
function getColumnGeneratedEReceiptID(headersForSender) {
  return headersForSender.indexOf("Generated e-Receipt ID") + 1;
}

function formatDescription(currentPaidMonth, currentFeePaid,
currentPaymentPurpose) {
  let description = "";
  if (currentPaymentPurpose == "Monthly Training Fee with/without
the Membership Fee.") {
    description = currentPaidMonth + " Training Fee ";

    for (let fee of currentFeePaid) {
      if (fee == "RM60 - Monthly Pass (UNLIMITED)") {
        description = description + "[Monthly Pass - RM 60]";
      }
      else if (fee == "RM30 - 2 Trainings Pass (2 trainings per
month only)") {
        description = description + "[2 Trainings Pass - RM 30]";
      }
    }

    if (currentFeePaid.includes("RM15 - Club Membership Fee")) {
      description = description + " and Membership Fee - RM 15";
    }
  }
  else {
    description = "Membership Fee";
  }

  description = description + "."
  return description
}

function hasGeneratedEReceipt(value) {
  return value != "-";
}
```

```
function isValid(value) {
  return value == "Valid";
}

function getColumnEReceiptSent(headersForSender) {
  return headersForSender.indexOf("Receipt Sent?") + 1;
}

function getColumnGeneratedEReceipt(headersForSender) {
  return headersForSender.indexOf("Generated e-Receipt") + 1;
}

function getColumnValidation(headersForSender) {
  return headersForSender.indexOf("Response Validity") + 1;
}
```

5. Make sure the **e-Receipts Folders and e-Receipt Template Files' ID** are correct in ***eReceiptGenerator.gs***

   For example, open each of those folders and look into their URLs

eReceiptWord (Temp || for conversion to pdf)
https://drive.google.com/drive/folders/1xcI8guAL4K7LYlKkpJjgmf9vh18jRdtv
eReceiptPDFs
https://drive.google.com/drive/folders/1jS--asuhfahDmIJ6-RggGwXgq5ahSzPp
MTC 2023 Official Receipt Template
https://docs.google.com/document/d/1DQv0cLaAvYvZXroK3lzbZOqniMGRtUaOwt7DFY8POH4/edit

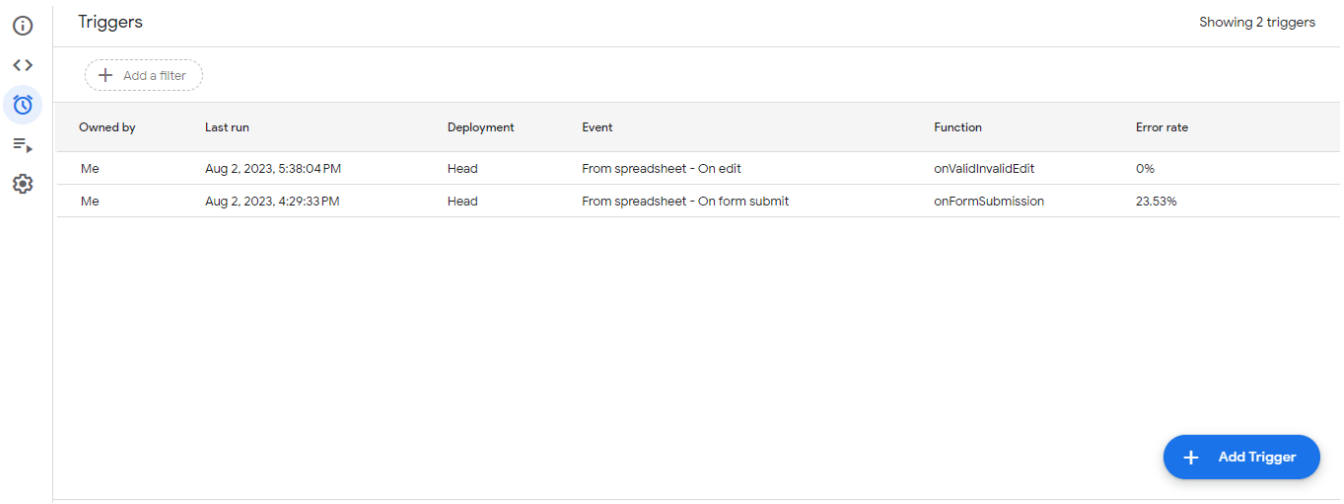
   Then, **copy the sections in red** from each of the **URLs**.

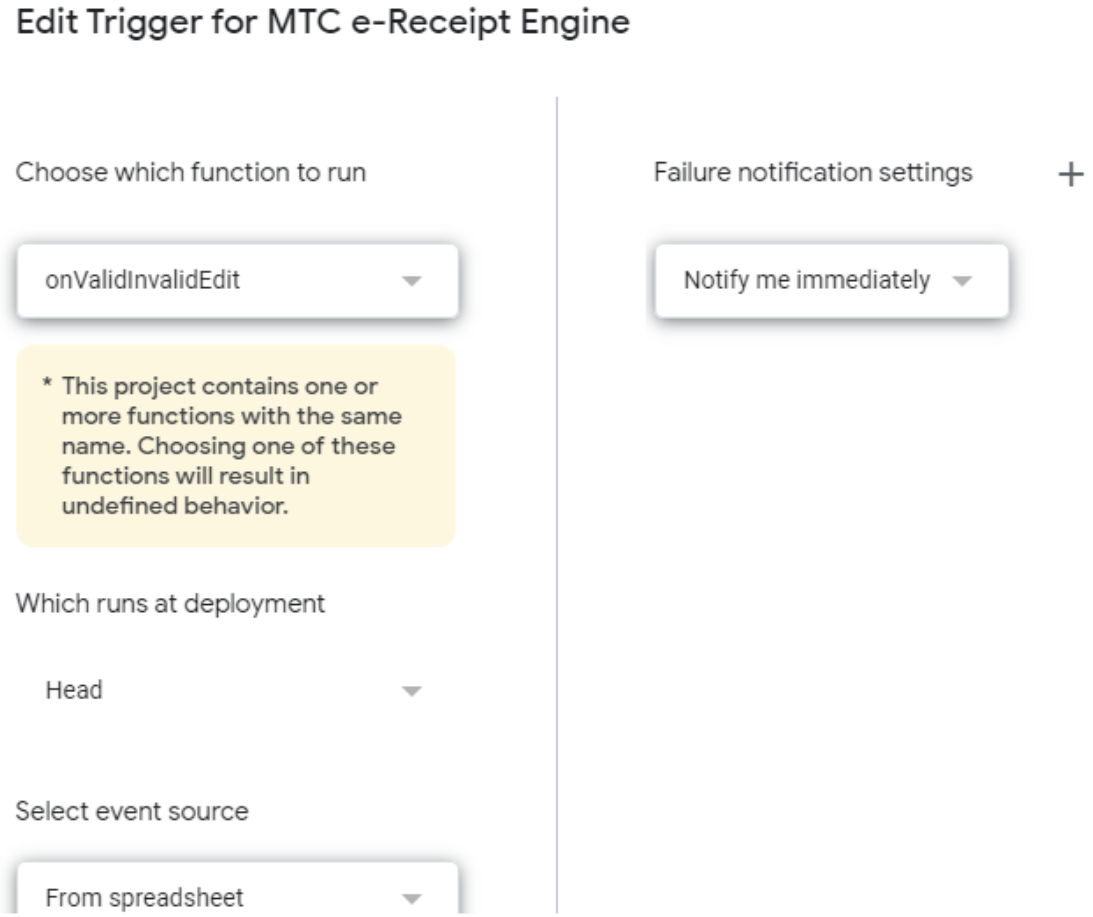
   Paste them into the three variables from the **eReceiptGenerator**

```
  const pdfFolderForEReceipt =
```

```
DriveApp.getFolderById("eReceiptPDFs");
  const tempFolderForEReceipt =
DriveApp.getFolderById("eReceiptWord (Temp || for conversion to pdf)");
  const eReceiptTemplateDoc = DriveApp.getFileById("MTC 2023 Official
Receipt Template");
```

6. Create **two triggers:**


Triggers
Showing 2 triggers
Add a filter
Owned by
Last run
Deployment
Event
Function
Error rate
Me
Aug 2, 2023, 5:38:04 PM
Head
From spreadsheet - On edit
onValidInvalidEdit
0%
Me
Aug 2, 2023, 4:29:33 PM
Head
From spreadsheet - On form submit
onFormSubmission
23.53%
Add Trigger


Edit Trigger for MTC e-Receipt Engine
Choose which function to run
onValidInvalidEdit
* This project contains one or more functions with the same name. Choosing one of these functions will result in undefined behavior.
Which runs at deployment
Head
Select event source
From spreadsheet
Failure notification settings
Notify me immediately

Edit Trigger for MTC e-Receipt Engine

undefined behavior.

Which runs at deployment

Head

Select event source

From spreadsheet

Select event type

On edit

Cancel Save

Edit Trigger for MTC e-Receipt Engine

Choose which function to run

onFormSubmission

* This project contains one or more functions with the same name. Choosing one of these functions will result in undefined behavior.

Which runs at deployment

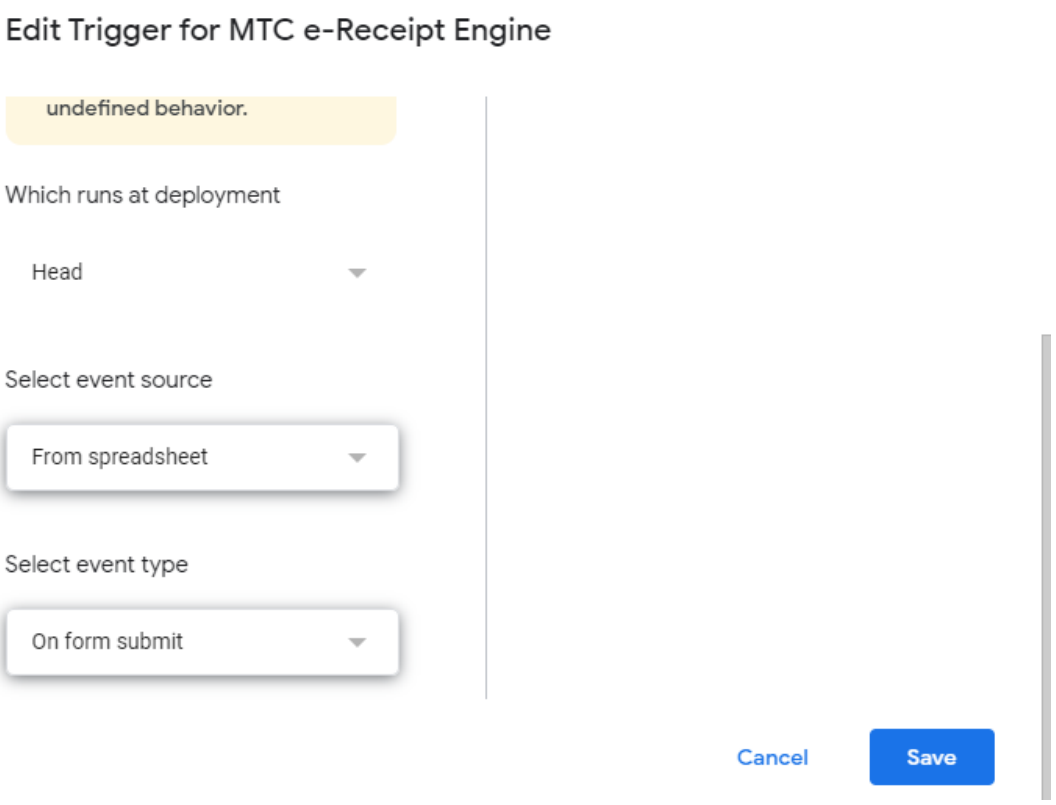
Head

Select event source

From spreadsheet

Failure notification settings +

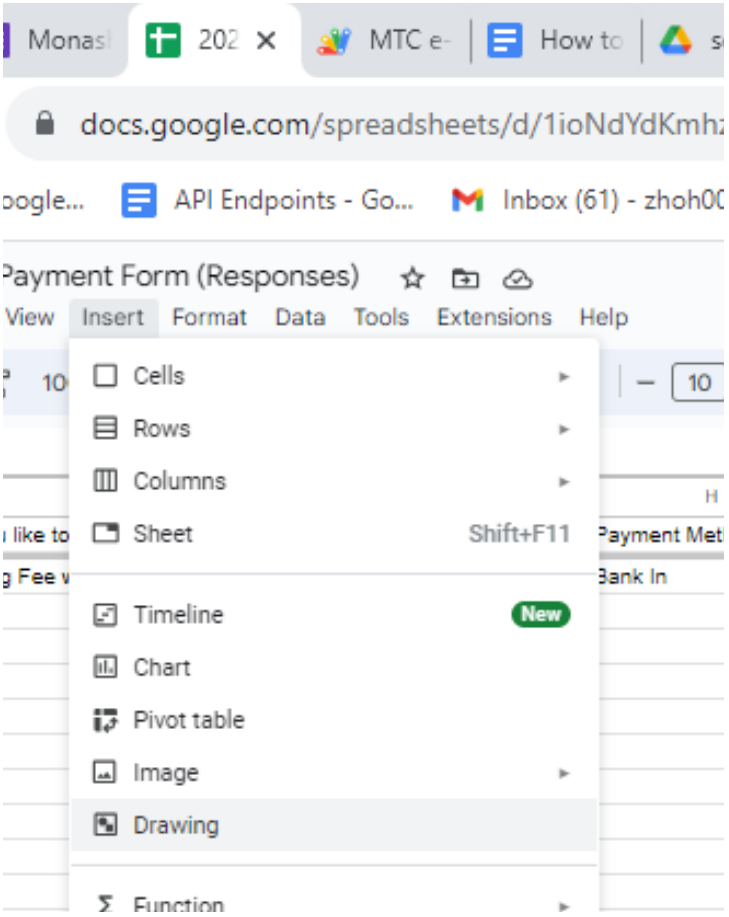
Notify me immediately

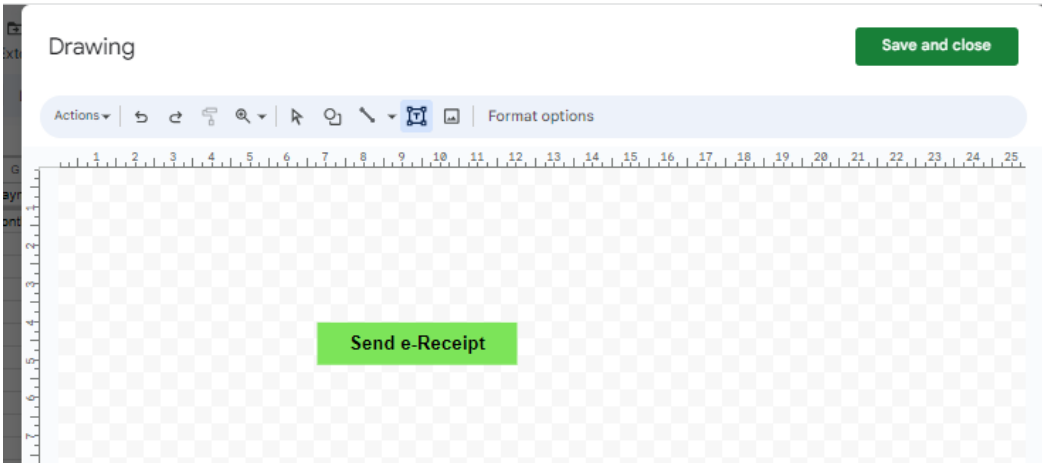


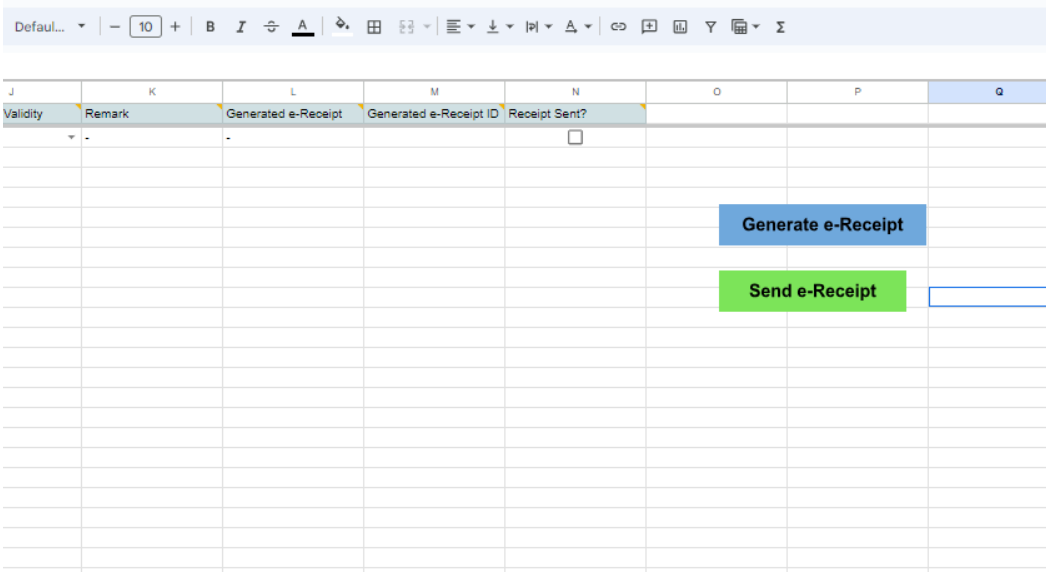
7. **Create a dummy response** to render the columns to work with.

H	I	J	K	L	M	N
ent Method	Proof of payment upload	Response Validity	Remark	Generated e-Receipt	Generated e-Receipt ID	Receipt Sent?
in	https://drive.google.com/	-	-	-		☐

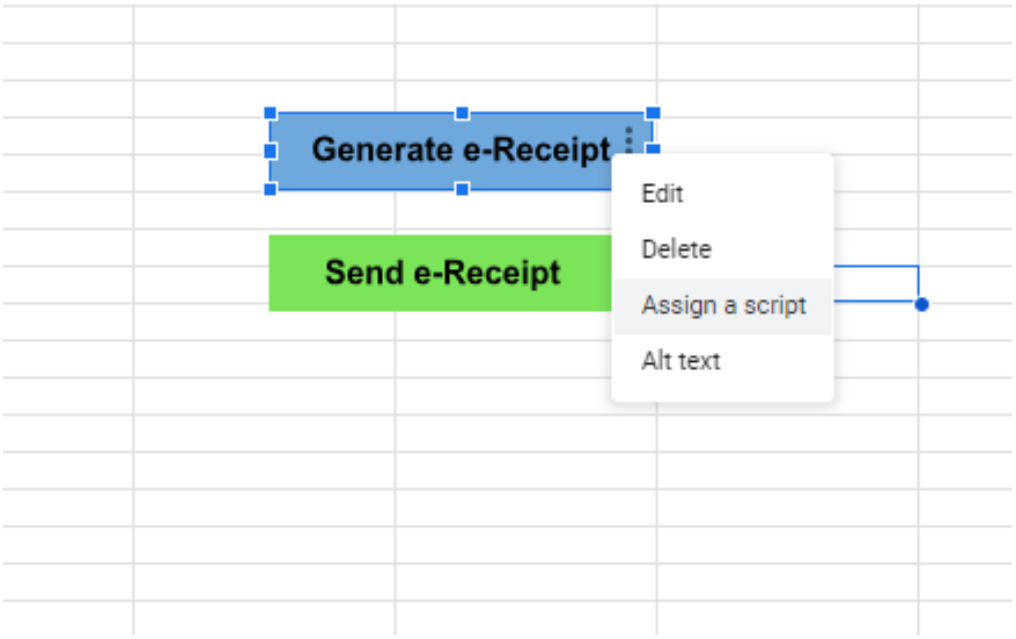
8. Insert **Two Drawings** (i.e. Text Boxes of **Generate and Send e–Receipt**)

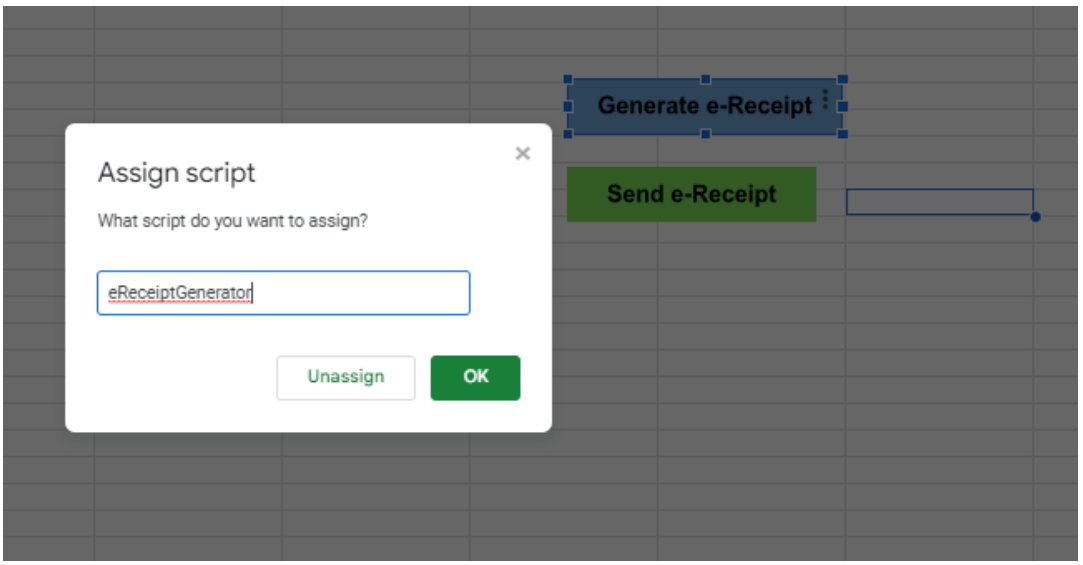

docs.google.com/spreadsheets/d/1ioNdYdKmh
API Endpoints - Go...
Inbox (61) - zhoh00
Payment Form (Responses)
View
Insert
Format
Data
Tools
Extensions
Help
Cells
Rows
Columns
Sheet
Shift+F11
Timeline
New
Chart
Pivot table
Image
Drawing
Function

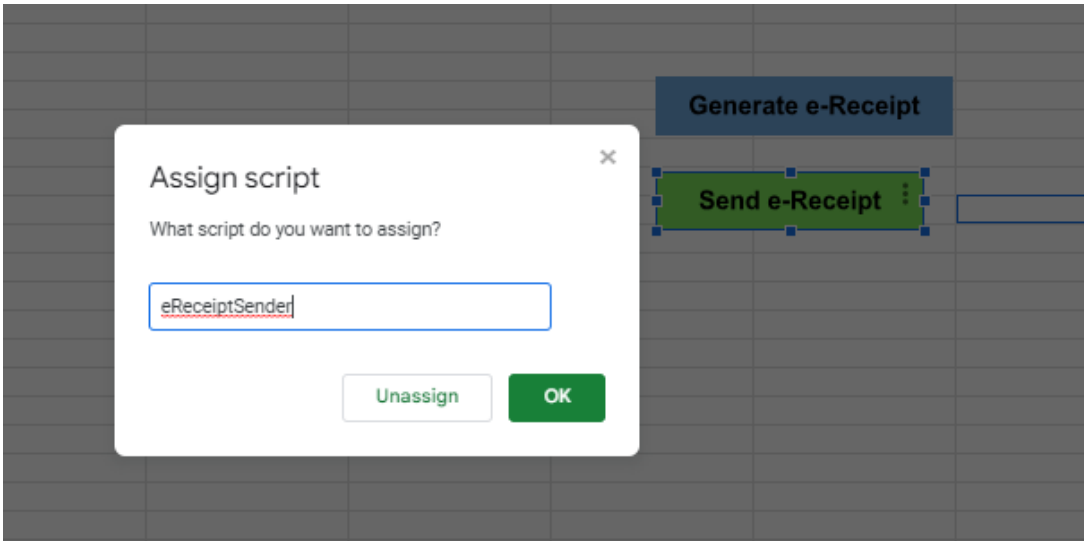

Drawing
Save and close
Actions
Format options
Send e-Receipt



9. Finally, **link their respective Google Scripts.**




Generate e-Receipt
Assign script
What script do you want to assign?
eReceiptGenerator
Unassign
OK
Send e-Receipt


Generate e-Receipt
Assign script
What script do you want to assign?
eReceiptSender
Unassign
OK
Send e-Receipt