Certainly! I'll create a Google Apps Script backend (webapp) that integrates with your HTML page to handle the receipt upload process. This script will:

1. Receive data from the HTML form

2. Save the order number, email address, and additional comments to a Google Sheet

3. Upload the image to Google Drive

4. Save the image link to the same Google Sheet

Here's the Google Apps Script code:

```javascript

// Set these variables according to your Google Sheet and folder setup

const SPREADSHEET\_ID = 'YOUR\_SPREADSHEET\_ID';

const SHEET\_NAME = 'Receipts';

const FOLDER\_ID = 'YOUR\_FOLDER\_ID';

function doPost(e) {

try {

const data = JSON.parse(e.postData.contents);

const orderNumber = data.orderNumber;

const email = data.email;

const comments = data.comments;

const imageData = data.image;

// Upload image to Google Drive

const imageName = `Receipt\_${orderNumber}\_${new Date().getTime()}.jpg`;

const imageBlob = Utilities.newBlob(Utilities.base64Decode(imageData), 'image/jpeg', imageName);

const folder = DriveApp.getFolderById(FOLDER\_ID);

const file = folder.createFile(imageBlob);

const imageUrl = file.getUrl();

// Save data to Google Sheets

const sheet = SpreadsheetApp.openById(SPREADSHEET\_ID).getSheetByName(SHEET\_NAME);

sheet.appendRow([new Date(), orderNumber, email, comments, imageUrl]);

return ContentService.createTextOutput(JSON.stringify({

success: true,

message: 'Receipt uploaded successfully'

})).setMimeType(ContentService.MimeType.JSON);

} catch (error) {

return ContentService.createTextOutput(JSON.stringify({

success: false,

message: 'Error: ' + error.toString()

})).setMimeType(ContentService.MimeType.JSON);

}

}

function doGet() {

return HtmlService.createHtmlOutput('<h1>This is a web app for handling receipt uploads.</h1>');

}

```

To use this Google Apps Script:

1. Create a new Google Apps Script project.

2. Copy and paste the above code into the script editor.

3. Replace `YOUR\_SPREADSHEET\_ID` with the ID of your Google Sheet.

4. Replace `YOUR\_FOLDER\_ID` with the ID of the Google Drive folder where you want to store the images.

5. Deploy the script as a web app, setting the following:

- Execute as: Your account

- Who has access: Anyone (for testing) or limit it as needed for production

Now, let's modify your HTML form to work with this backend. Here are the changes you need to make to your existing HTML:

```tsx

import React, { useState } from 'react';

import { Card, CardHeader, CardContent } from '@/components/ui/card';

import { Button } from '@/components/ui/button';

import { Input } from '@/components/ui/input';

import { Textarea } from '@/components/ui/textarea';

const ReceiptUploadForm = () => {

const [orderNumber, setOrderNumber] = useState('');

const [email, setEmail] = useState('');

const [comments, setComments] = useState('');

const [file, setFile] = useState(null);

const handleSubmit = async (e) => {

e.preventDefault();

if (!file) {

alert('Please select an image file.');

return;

}

const reader = new FileReader();

reader.onloadend = async () => {

const base64Image = reader.result.split(',')[1];

const formData = {

orderNumber,

email,

comments,

image: base64Image

};

try {

const response = await fetch('YOUR\_GOOGLE\_APPS\_SCRIPT\_WEB\_APP\_URL', {

method: 'POST',

body: JSON.stringify(formData),

headers: {

'Content-Type': 'application/json',

},

});

const result = await response.json();

if (result.success) {

alert('Receipt uploaded successfully!');

// Reset form

setOrderNumber('');

setEmail('');

setComments('');

setFile(null);

} else {

alert('Error uploading receipt: ' + result.message);

}

} catch (error) {

alert('Error uploading receipt: ' + error.toString());

}

};

reader.readAsDataURL(file);

};

return (

<Card className="w-full max-w-md mx-auto">

<CardHeader>

<h2 className="text-2xl font-bold">Upload Your Receipt</h2>

</CardHeader>

<CardContent>

<form onSubmit={handleSubmit} className="space-y-4">

<div>

<label htmlFor="orderNumber" className="block text-sm font-medium text-gray-700">Order Number</label>

<Input

type="text"

id="orderNumber"

value={orderNumber}

onChange={(e) => setOrderNumber(e.target.value)}

required

/>

</div>

<div>

<label htmlFor="email" className="block text-sm font-medium text-gray-700">Email Address</label>

<Input

type="email"

id="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

required

/>

</div>

<div>

<label htmlFor="file" className="block text-sm font-medium text-gray-700">Receipt Image</label>

<Input

type="file"

id="file"

accept="image/\*"

onChange={(e) => setFile(e.target.files[0])}

required

/>

</div>

<div>

<label htmlFor="comments" className="block text-sm font-medium text-gray-700">Additional Comments (Optional)</label>

<Textarea

id="comments"

value={comments}

onChange={(e) => setComments(e.target.value)}

rows={3}

/>

</div>

<Button type="submit" className="w-full">Upload Receipt</Button>

</form>

</CardContent>

</Card>

);

};

export default ReceiptUploadForm;

```

To use this updated form:

1. Replace `YOUR\_GOOGLE\_APPS\_SCRIPT\_WEB\_APP\_URL` with the URL of your deployed Google Apps Script web app.

2. Make sure you have the necessary UI components from your design system (Card, Button, Input, Textarea) imported and available.

This setup will now:

1. Capture the order number, email, comments, and image file from the form.

2. Convert the image to a base64 string.

3. Send all the data to your Google Apps Script backend.

4. The backend will save the text data to a Google Sheet and the image to Google Drive.

5. The image URL will be saved alongside the other data in the Google Sheet.

6. The user will receive a success message upon completion.

Remember to test thoroughly and handle potential errors gracefully. Also, consider implementing security measures like authentication and input validation in a production environment.

Certainly! I'll update the Google Apps Script to include email notifications to both the sheet operators and the customer who submitted the form. We'll add this functionality to our existing script.

```javascript

// Set these variables according to your Google Sheet and folder setup

const SPREADSHEET\_ID = 'YOUR\_SPREADSHEET\_ID';

const SHEET\_NAME = 'Receipts';

const FOLDER\_ID = 'YOUR\_FOLDER\_ID';

const OPERATOR\_EMAIL = 'operator@example.com'; // Replace with actual operator email

function doPost(e) {

try {

const data = JSON.parse(e.postData.contents);

const orderNumber = data.orderNumber;

const email = data.email;

const comments = data.comments;

const imageData = data.image;

// Upload image to Google Drive

const imageName = `Receipt\_${orderNumber}\_${new Date().getTime()}.jpg`;

const imageBlob = Utilities.newBlob(Utilities.base64Decode(imageData), 'image/jpeg', imageName);

const folder = DriveApp.getFolderById(FOLDER\_ID);

const file = folder.createFile(imageBlob);

const imageUrl = file.getUrl();

// Save data to Google Sheets

const sheet = SpreadsheetApp.openById(SPREADSHEET\_ID).getSheetByName(SHEET\_NAME);

const timestamp = new Date();

sheet.appendRow([timestamp, orderNumber, email, comments, imageUrl]);

// Send emails

sendOperatorEmail(orderNumber, email, comments, imageUrl, timestamp);

sendCustomerEmail(email, orderNumber, timestamp);

return ContentService.createTextOutput(JSON.stringify({

success: true,

message: 'Receipt uploaded successfully. Confirmation emails have been sent.'

})).setMimeType(ContentService.MimeType.JSON);

} catch (error) {

return ContentService.createTextOutput(JSON.stringify({

success: false,

message: 'Error: ' + error.toString()

})).setMimeType(ContentService.MimeType.JSON);

}

}

function sendOperatorEmail(orderNumber, customerEmail, comments, imageUrl, timestamp) {

const subject = `New Receipt Upload - Order #${orderNumber}`;

const body = `

A new receipt has been uploaded:

Order Number: ${orderNumber}

Customer Email: ${customerEmail}

Timestamp: ${timestamp.toLocaleString()}

Comments: ${comments}

Image URL: ${imageUrl}

Please review the uploaded receipt at your earliest convenience.

`;

MailApp.sendEmail(OPERATOR\_EMAIL, subject, body);

}

function sendCustomerEmail(customerEmail, orderNumber, timestamp) {

const subject = `Receipt Upload Confirmation - Order #${orderNumber}`;

const body = `

Dear Customer,

We have received your receipt upload for Order #${orderNumber} on ${timestamp.toLocaleString()}.

Thank you for submitting your receipt. Our team will review it shortly.

If you have any questions, please don't hesitate to contact our customer support.

Best regards,

TPO Wellness Store Team

`;

MailApp.sendEmail(customerEmail, subject, body);

}

function doGet() {

return HtmlService.createHtmlOutput('<h1>This is a web app for handling receipt uploads.</h1>');

}

```

Here are the key changes and additions to the script:

1. Added a constant `OPERATOR\_EMAIL` at the top of the script. You should replace this with the actual email address of the sheet operators.

2. In the `doPost` function, after saving the data to the Google Sheet, we now call two new functions: `sendOperatorEmail` and `sendCustomerEmail`.

3. The `sendOperatorEmail` function:

- Constructs an email with details of the uploaded receipt, including the order number, customer email, timestamp, comments, and the image URL.

- Sends this email to the operator's email address.

4. The `sendCustomerEmail` function:

- Constructs a confirmation email for the customer, including the order number and timestamp of their submission.

- Sends this email to the customer's email address that was provided in the form.

5. Updated the success message to indicate that confirmation emails have been sent.

To use this updated script:

1. Replace `'YOUR\_SPREADSHEET\_ID'` with your actual Google Sheet ID.

2. Replace `'YOUR\_FOLDER\_ID'` with your actual Google Drive folder ID where images will be stored.

3. Replace `'operator@example.com'` with the actual email address of the sheet operators.

4. Make sure your Google Apps Script project has permission to send emails. You might need to authorize this when you first run the script.

This script now provides a more comprehensive solution:

- It saves the receipt data and image as before.

- It notifies the operators of new submissions, allowing them to quickly review new receipts.

- It sends a confirmation email to customers, improving their experience and providing them with a record of their submission.

Remember to test the script thoroughly, especially the email functionality, to ensure everything works as expected. Also, consider implementing rate limiting or other measures to prevent abuse of the email sending functionality.