How copy the google add-on project

1. Go to https://script.google.com/home and create a new project
2. Give the new project a name then go to view => show manifest file
3. Delete the basic code file that was created with the project
4. Go to File => New => Script file and create 3 different script files named
    a. Common
    b. Calendar
    c. Authentication
5. The code to paste in these files is at the bottom of this word doc. Copy the code below and paste it into the corresponding name in the google project. This also includes the copy and pasting the appsscripts.json file
6. Save the project.

Files to copy
*Note: copy below the ---- to the next _______

____

Appsscripts.json

```
{

  "timeZone": "America/New_York",

  "dependencies": {

    "enabledAdvancedServices": [{

      "userSymbol": "Calendar",

      "serviceId": "calendar",

      "version": "v3"

    }],

    "libraries": [{

      "userSymbol": "OAuth2",

      "libraryId": "1B7FSrk5Zi6L1rSxxTDgDEUsPzlukDsi4KGuTMorsTQHhGBzBkMun4iDF",

      "version": "38"

    }]
```

```
},
"exceptionLogging": "STACKDRIVER",
"oauthScopes": [
  "https://www.googleapis.com/auth/script.external_request",
  "https://www.googleapis.com/auth/calendar.addons.execute",
  "https://www.googleapis.com/auth/calendar.readonly",
  "https://www.googleapis.com/auth/calendar",
  "https://www.googleapis.com/auth/script.locale",
  "https://www.googleapis.com/auth/calendar.addons.current.event.read",
  "https://www.googleapis.com/auth/calendar.events.readonly",
  "https://www.googleapis.com/auth/calendar.addons.current.event.write",
  "https://www.googleapis.com/auth/script.scriptapp"
],
"urlFetchWhitelist": ["https://login.microsoftonline.com/common/oauth2/v2.0/authorize"],
"runtimeVersion": "V8",
"addOns": {
  "common": {
    "name": "Teams Meeting Scheduler",
    "logoUrl": "LOGO URL HERE",
    "layoutProperties": {
      "primaryColor": "#6963ff",
      "secondaryColor": "#1f2933"
    },
    "useLocaleFromApp": true,
    "homepageTrigger": {
      "runFunction": "onHomepage",
      "enabled": true
    },
    "universalActions": [{
        "label": "Log out",
        "runFunction": "resetOAuth"
      }]
    },
    "calendar": {
      "conferenceSolution": [{
        "onCreateFunction": "createConference",
        "id": "msftTeamsMeeting",
        "name": "Microsoft Teams Meeting",
        "logoUrl": "LOGO URL HERE"
      }],
      "eventOpenTrigger": {
        "runFunction": "onCalendarEventOpen"
      },
      "eventUpdateTrigger": {
        "runFunction": "buildFoundMeetingCard"
      },
      "currentEventAccess": "READ_WRITE"
    }
  }
}
```

____

Common.gs

```
---------------------
/**
 * Callback for rendering the homepage card.
 * @return {CardService.Card} The card to show to the user.
 */
```

```
function onHomepage(e) {
  return createCard(true);
}

/**
 * @param {String} text The text to overlay on the image.
 * @param {Boolean} isHomepage True if the card created here is a homepage;
 *      false otherwise. Defaults to false.
 * @return {CardService.Card} The assembled card.
 */
function createCard(isHomepage) {
  // Explicitly set the value of isHomepage as false if null or undefined.
  if (!isHomepage) {
    isHomepage = false;
  }

  var service = getOAuthService();
  var maybeAuthorized = service.hasAccess();
  if(!maybeAuthorized) {
    CardService.newAuthorizationException()
    .setAuthorizationUrl(service.getAuthorizationUrl())
    .setResourceDisplayName("Microsoft Teams Meeting Scheduler")
    .throwException();
  }

  var cardHeader = CardService.newCardHeader()
    .setTitle("Welcome to Teams Meeting Scheduler")
    .setSubtitle("Create a meeting to get started");
  var card = CardService.newCardBuilder()
      .setHeader(cardHeader);

  if (!isHomepage) {
    // Create the header shown when the card is minimized,
    // but only when this card is a contextual card. Peek headers
    // are never used by non-contexual cards like homepages.
    var peekHeader = CardService.newCardHeader()
      .setTitle('Not Home Page')
      .setSubtitle('this is not the homepage');
    card.setPeekCardHeader(peekHeader)
  }

  return card.build();
}
```
____

Calendar.gs

```
function onCalendarEventOpen(e) {
  console.log('onCalenderEventOpen param');
  console.log(e);
  Logger.log(e);
  try{
    if (e.calendar.conferenceData.parameters.addOnParameters.parameters.meetingType === "Microsoft
Teams"){
      return buildFoundMeetingCard(e)
    }
  }
  catch(err){
    console.log("event existed but didn't have a teams meeting");
    console.log(err);
  }
  var calendar = CalendarApp.getCalendarById(e.calendar.calendarId);
  var event = CalendarApp.getEventById(e.calendar.id);
  if (!event) {
    // This is a new event still being created.
    return buildNewMeetingCard();
  }

  return buildNewMeetingCard(event);
}

function buildNewMeetingCard(e) {
  console.log('buildNewMeetingCard param');
  Logger.log(e);
  var service = getOAuthService();
  var maybeAuthorized = service.hasAccess();
  if(!maybeAuthorized) {
    CardService.newAuthorizationException()
    .setAuthorizationUrl(service.getAuthorizationUrl())
    .setResourceDisplayName("G Suite dev")
    .throwException();
  }

  // Assemble the widgets and return the card.
  var cardHeader = CardService.newCardHeader()
  .setTitle("Create a meeting");
  var card = CardService.newCardBuilder()
  .setHeader(cardHeader);

  return card.build();

}

function buildFoundMeetingCard(e) {
  console.log('buildFoundMeetingCard param');
  Logger.log(e);
  var service = getOAuthService();
  var maybeAuthorized = service.hasAccess();
  if(!maybeAuthorized) {
    CardService.newAuthorizationException()
    .setAuthorizationUrl(service.getAuthorizationUrl())
    .setResourceDisplayName("G Suite dev")
    .throwException();
  }

  var openMeetingSettingsButton = CardService.newTextButton()
  .setText('Open Meeting Settings')
.setOpenLink(CardService.newOpenLink().setUrl(e.calendar.conferenceData.parameters.addOnParamet
ers.parameters.meetingSettingsUrl))
  .setTextButtonStyle(CardService.TextButtonStyle.FILLED);

  var openMeetingSettingsButtonSet = CardService.newButtonSet()
  .addButton(openMeetingSettingsButton);
  var cardHeader = CardService.newCardHeader()
  .setTitle("Meeting Settings");

  var section = CardService.newCardSection()
  .addWidget(openMeetingSettingsButtonSet);

  var card = CardService.newCardBuilder()
  .setHeader(cardHeader)
  .addSection(section);

  return card.build();
}

function createConference(arg) {
  const eventData = arg.eventData;
  const calendarId = eventData.calendarId;
  const eventId = eventData.eventId;

  // Retrieve the Calendar event information using the Calendar
  // Advanced service.
  var calendarEvent;
  try {
    //this will work if the event exists already
    calendarEvent = Calendar.Events.get(calendarId, eventId);
  } catch (err) {
    // The calendar event does not exist just yet; just proceed with the
    // given event ID and allow the event details to sync later.
    console.log(err);
    calendarEvent = {
```

```
    id: eventId,
   };
  }

  // Build and return a ConferenceData object, either with conference or
  // error information.
  var dataBuilder = ConferenceDataService.newConferenceDataBuilder();
//  .setNotes("Select Joining Instructions to edit Teams meeting");

  var service = getOAuthService();
  var maybeAuthorized = service.hasAccess();
  if(!maybeAuthorized) {
   var authenticationUrl = getAuthenticationUrl();
   var error = ConferenceDataService.newConferenceError()
    .setConferenceErrorType(ConferenceDataService.ConferenceErrorType.AUTHENTICATION)
    .setAuthenticationUrl(authenticationUrl);

   dataBuilder.setError(error);
   return dataBuilder.build();
  }

  // Create a conference on the third-party service and return the
  // conference data or errors in a custom JSON object.
  var conferenceInfo = create3rdPartyConference(calendarEvent);

  Logger.log('conferenceInfo-teams meeting');
  console.log(conferenceInfo);
  Logger.log(conferenceInfo);
  if (!conferenceInfo.error) {
   // No error, so build the ConferenceData object from the
   // returned conference info.
   const organizerId = conferenceInfo.participants.organizer.identity.user.id;
   const tenantId = conferenceInfo.participants.organizer.identity.user.tenantId;
   const threadId = conferenceInfo.chatInfo.threadId;
   const messageId = conferenceInfo.chatInfo.messageId
   const language = "en-US";
   const settingsUrl = "https://teams.microsoft.com/meetingOptions/?organizerId=" + organizerId +
"&tenantId=" + tenantId + "&threadId=" + threadId + "&messageId=" + messageId + "&language=" +
language;

   var urlEntryPoint = ConferenceDataService.newEntryPoint()
       .setEntryPointType(ConferenceDataService.EntryPointType.VIDEO)
       .setUri(conferenceInfo.joinWebUrl);

   var meetingSettingsParameter = ConferenceDataService.newConferenceParameter()
       .setKey('meetingSettingsUrl')
       .setValue(settingsUrl);

   var meetingTypeParameter = ConferenceDataService.newConferenceParameter()
       .setKey('meetingType')
       .setValue('Microsoft Teams');

   dataBuilder.setConferenceId(eventId)
   .addEntryPoint(urlEntryPoint)
   .addConferenceParameter(meetingSettingsParameter)
   .addConferenceParameter(meetingTypeParameter);
  } else if (conferenceInfo.error === 'AUTH') {
   console.log("auth error");
   var authenticationUrl = getAuthenticationUrl();
   var error = ConferenceDataService.newConferenceError()
       .setConferenceErrorType(
           ConferenceDataService.ConferenceErrorType.AUTHENTICATION)
       .setAuthenticationUrl(authenticationUrl);
   dataBuilder.setError(error);

  } else {
   console.log("not auth error");
   // Other error type;
   var error = ConferenceDataService.newConferenceError()
       .setConferenceErrorType(
           ConferenceDataService.ConferenceErrorType.TEMPORARY);
   dataBuilder.setError(error);
  }

  // Don't forget to build the ConferenceData object.
  return dataBuilder.build();
}

function create3rdPartyConference(calendarEvent) {
  let onlineMeeting = {};
  try{
   onlineMeeting = {
     startDateTime: calendarEvent.start.dateTime,
     endDateTime: calendarEvent.end.dateTime,
     subject: "Microsoft Teams Meeting"
    };
  } catch (err) {
    onlineMeeting = {
      startDateTime: new Date(),
      endDateTime: new Date(),
      subject: "Microsoft Teams Meeting"
    };
  }

  let meResponse = accessProtectedResource('https://graph.microsoft.com/v1.0/me/onlineMeetings',
'post', JSON.stringify(onlineMeeting));
  var data = JSON.parse(meResponse);
  return data;
}

function getAuthenticationUrl() {
  var authorizationUrl;
  var service = getOAuthService();
  var authorizationUrl = service.getAuthorizationUrl();

  return authorizationUrl;
}
```

Authentication.gs

```
function accessProtectedResource(url, method_opt, payload_opt, headers_opt) {
  var service = getOAuthService();
  var maybeAuthorized = service.hasAccess();
  if (maybeAuthorized) {
```

```
// A token is present, but it may be expired or invalid. Make a
// request and check the response code to be sure.

// Make the UrlFetch request and return the result.
var accessToken = service.getAccessToken();
var method = method_opt || 'get';
var headers = headers_opt || {};
var payload = payload_opt || {};
headers['Authorization'] = Utilities.formatString('Bearer %s', accessToken);;
var resp = UrlFetchApp.fetch(url, {
  'headers': headers,
  'method' : method,
  'payload': payload,
  'contentType': 'application/json',
  'muteHttpExceptions': true, // Prevents thrown HTTP exceptions.
});

var code = resp.getResponseCode();
if (code >= 200 && code < 300) {
  return resp.getContentText("utf-8"); // Success
} else if (code == 401 || code == 403) {
   // Not fully authorized for this action.
   maybeAuthorized = false;
} else {
   // Handle other response codes by logging them and throwing an
   // exception.
   console.error("Backend server error (%s): %s", code.toString(),
                 resp.getContentText("utf-8"));
   throw ("Backend server error: " + code);
```

```
    }
  }

  if (!maybeAuthorized) {
    //oauth2 docs stuff
    var authorizationUrl = service.getAuthorizationUrl();
    var template = HtmlService.createTemplate(
        '<a href="<?= authorizationUrl ?>" target="_blank">Authorize</a>. ' +
        'Reopen the sidebar when the authorization is complete.');
    template.authorizationUrl = authorizationUrl;
    var page = template.evaluate();
    console.log(page);

    // Invoke the authorization flow using the default authorization
    // prompt card.
    CardService.newAuthorizationException()
        .setAuthorizationUrl(service.getAuthorizationUrl())
        .setResourceDisplayName("Microsoft Teams Meeting Scheduler")
        .throwException();
  }
}

function getOAuthService() {
  return OAuth2.createService('Microsoft Teams Meeting Scheduler')

.setAuthorizationBaseUrl("https://login.microsoftonline.com/common/oauth2/v2.0/authorize?response
_mode=query&prompt=select_account")// use prompt=select_account to force them to sign in
      .setClientId('CLIENT ID HERE')
      .setTokenUrl('https://login.microsoftonline.com/common/oauth2/v2.0/token')
      .setClientSecret('CLIENT SECRET HERE')
      .setScope("User.Read openid offline_access OnlineMeetings.ReadWrite")
      .setCallbackFunction('authCallback')
      .setCache(CacheService.getUserCache())
      .setPropertyStore(PropertiesService.getUserProperties());
}

function authCallback(callbackRequest) {
  var authorized = getOAuthService().handleCallback(callbackRequest);
  if (authorized) {
    return HtmlService.createHtmlOutput(
      'Success! <script>setTimeout(function() { top.window.close() }, 1);</script>');
  } else {
    return HtmlService.createHtmlOutput('Denied');
  }
}

function resetOAuth() {
  getOAuthService().reset();
  accessProtectedResource();
}
```