FLUTTER TUTORIALS

BY: **TECMAN**

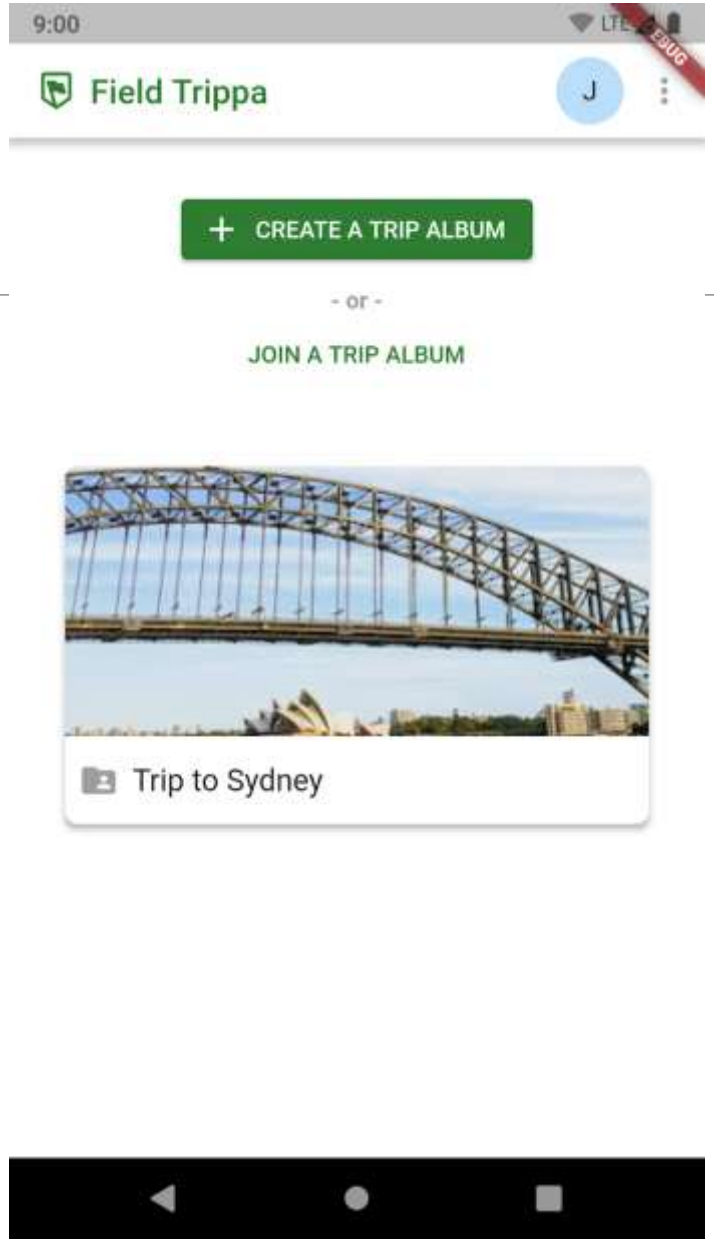
TECMAN Lesson 11

Build a Photo Sharing app with Google Photos and Flutter

What you will build

In this codelab, you'll build a field trip app, Field Trippa, that enables users to share photos.

Learn how to use the Google Photos Library API to back a media sharing experience in your own application.



TECMAN LIMITED 2019

Build a Photo Sharing app with Google Photos and Flutter

What you'll learn

How to use the Google Photos Library API to upload media and share albums

How to use Google Sign-In in Flutter

How to make Google API calls from Flutter

TECMAN LIMITED 2019

Build a Photo Sharing app with Google Photos and Flutter

What you'll need

Flutter development environment

Two Google user accounts set up on different emulators or devices that have access to Google Photos, so you can test sharing between users

TECMAN LIMITED 2019

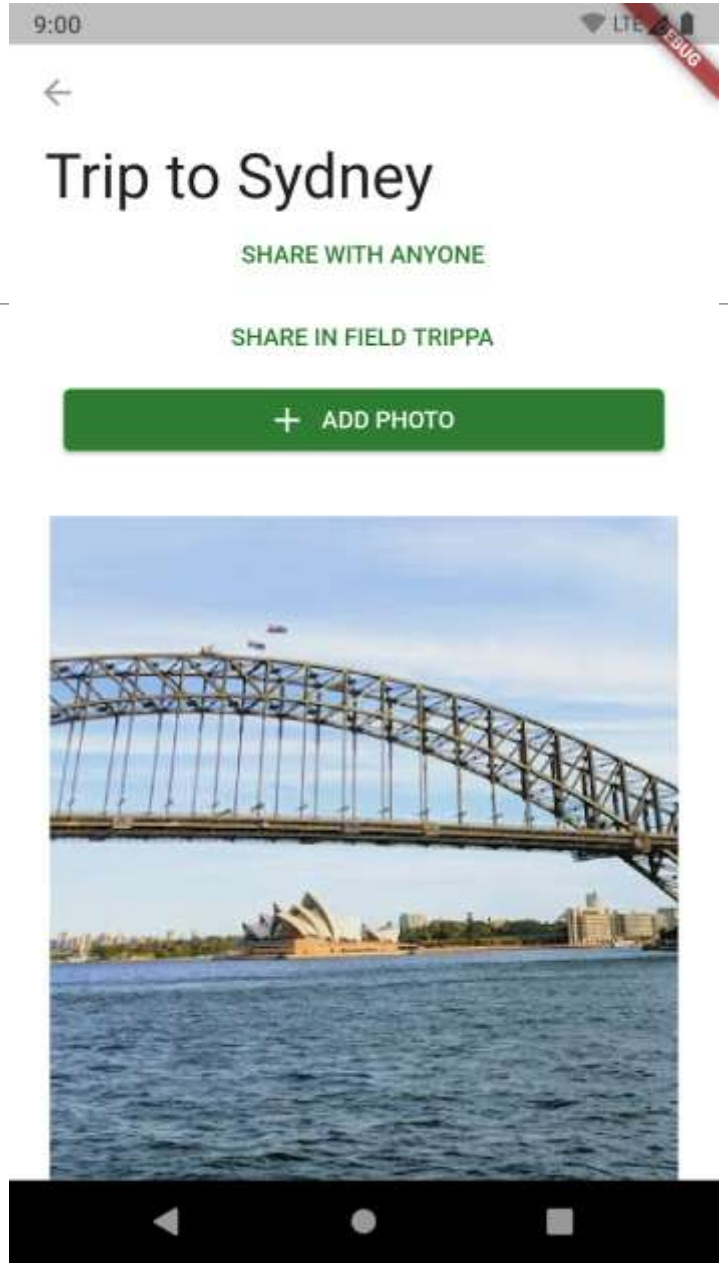
Overview over the Field Trip Gallery App - "Field Trippa"

In this codelab you will build an app to share photos for an excursion or field trip, built using the Google Photos Library API.

The user logs in using Google Sign-In and authorizes the application to use the Google Photos Library API.

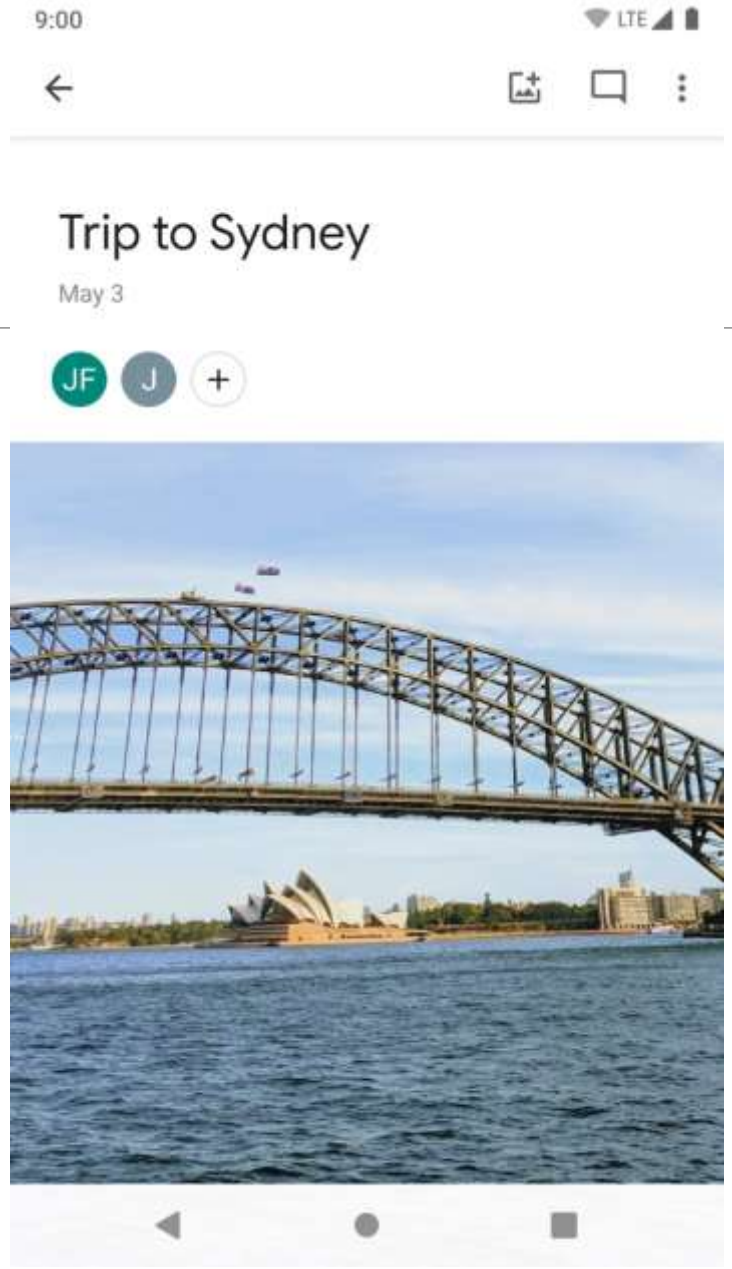
Then, the user can create a trip for uploading photos with a description. Each trip can be shared with other members of the application, who can also contribute photos.

TECMAN LIMITED 2019



Overview over the Field Trip Gallery App - "Field Trippa"

Under the hood, each trip is stored as a *shared album* inside Google Photos. The app handles sharing and uploading to this album, but you can also share the album with others who do not have the app directly through a URL to Google Photos.



TECMAN LIMITED 2019

Download the Code

Download the source code for this codelab:

Download source code Browse on GitHub

(The starting app code is available in the master branch in the repository.)

Unpack the downloaded zip file. This creates a root folder, photos-sharing-master, that contains all the code and resources you need to get started.

Open the extracted folder in your preferred Flutter IDE, for example VSCode or Android Studio with the Dart and Flutter plugins installed.

Download the Code

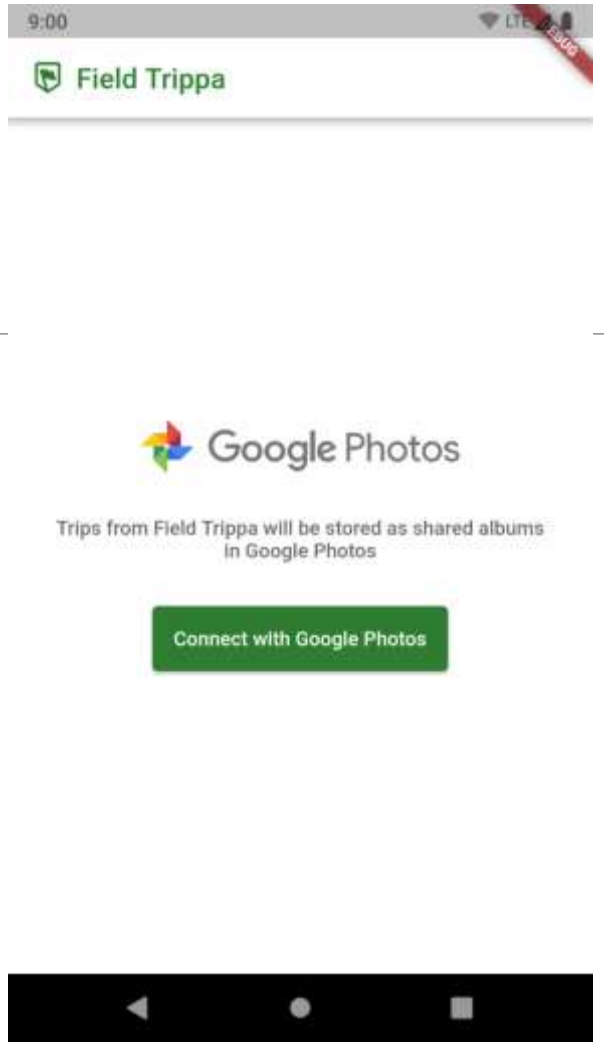
Final Implementation Code

The following link points to the final version of the application that is fully implemented. You can use this if you get stuck or to compare your implementation:

Download final source code Browse final source code on GitHub

(The final app code is available in the final branch in the repository.)

TECMAN LIMITED 2019



You should see the "**Connect with Google Photos**" screen:

Set up the Google Photos Library API

The Google Photos Library API requires you to authenticate your users using OAuth 2.0. Users sign into the application and authorize it to interact with the API on their behalf.

TECMAN LIMITED 2019

Set up the Google Photos Library API

Create a new Firebase project and register your app

Only create a new Firebase project and download the Android or iOS configuration files as described in this codelab.

(The required Firebase SDK is already included by the Flutter package that handles sign in.)

You do not need to add the Firebase SDK or add any dependencies to the project. Skip these steps in the Firebase console.

Set up the Google Photos Library API

- Go to the Firebase console and select "**+ Add Project**". Enter a project name and select "**Create Project**" to continue. Do not follow any other steps in the Firebase console. Instead, return to this codelab and continue to the "Android" or "iOS" parts below to configure the application.

Set up the Google Photos Library API

Android only: If you are running the app on Android, register an Android app:

- Click the **Android icon** to open the Android app registration screen.
- For **package**, enter: com.google.codelab.photos.sharing
- Retrieve the **signing certificate SHA-1** from your machine: Run the following command:

Set up the Google Photos Library API

```
keytool -alias androiddebugkey -keystore ~/.android/debug.keystore -list -v -storepass a
```

TECMAN LIMITED 2019

Set up the Google Photos Library API

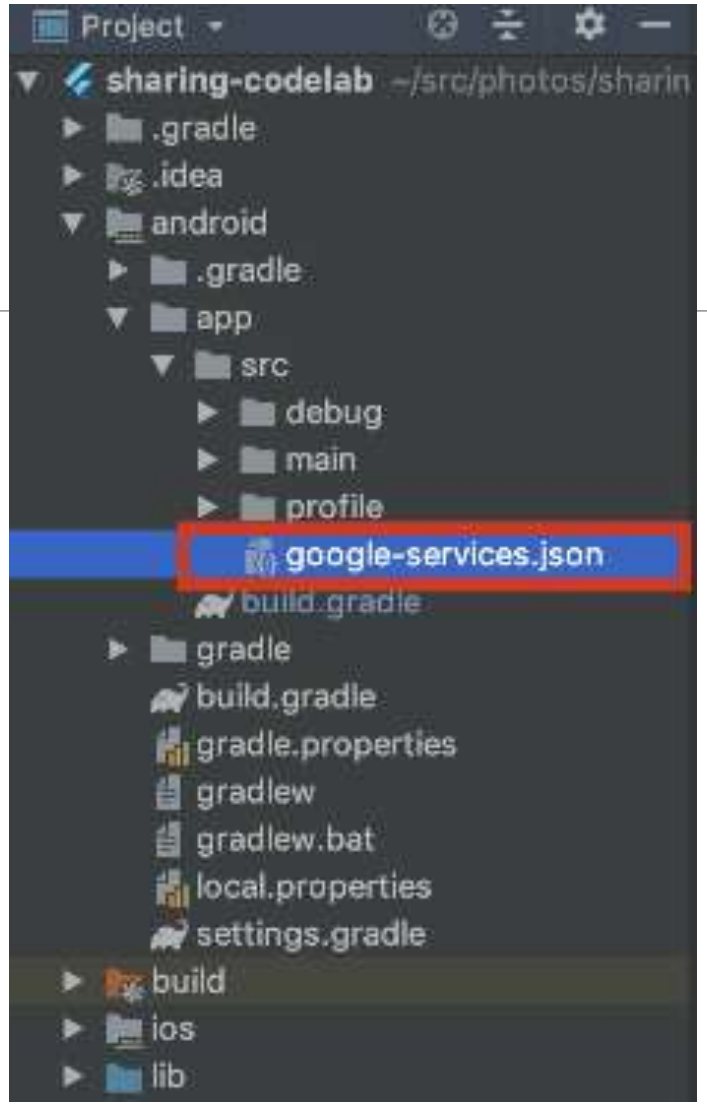
> The signing certificate is used to sign the Android application. The debug certificate is usually located at `<home>/.android/debug.keystore`. You can also retrieve the certificate by running this command from inside the `android/` directory: `./gradlew signingReport`
>
> See here for more detail on finding the SHA1.

TECMAN LIMITED 2019

Set up the Google Photos Library API

- Click "**next**" to continue.
- Download the google-service.json file to your computer and move it into the directory "android/app/". *(Tip: In Android Studio, you can drag the downloaded file directly into the correct location in the **project** side panel.)*

This file contains the project configuration for the Firebase and Google Developers project you have just set up.



(See the documentation for the package google_sign_in for details.)

TECMAN LIMITED 2019

Enable the Google Photos Library API

- Open the API screen in the Google Developers console and enable the "**Google Photos Library API**". (You may have to select the Firebase project at the top of the screen first if the "**enable**" button is disabled.)
- Open the OAuth consent screen configuration in the Google Developers console to add the Google Photos Library API scopes and your email address. (This configuration is required for the OAuth verification review for any scopes used by the Google Photos Library API.)

Enable the Google Photos Library API

- Enter the **Application Name"**, for example Field Trippa Codelab
- Select "**Add Scope**", then "**manually paste scopes"** to enter the following scopes:

```
https://www.googleapis.com/auth/photoslibrary
https://www.googleapis.com/auth/photoslibrary.sharing
```

- Select a "**support email address**"
- Select **Save**

Run the app and sign in

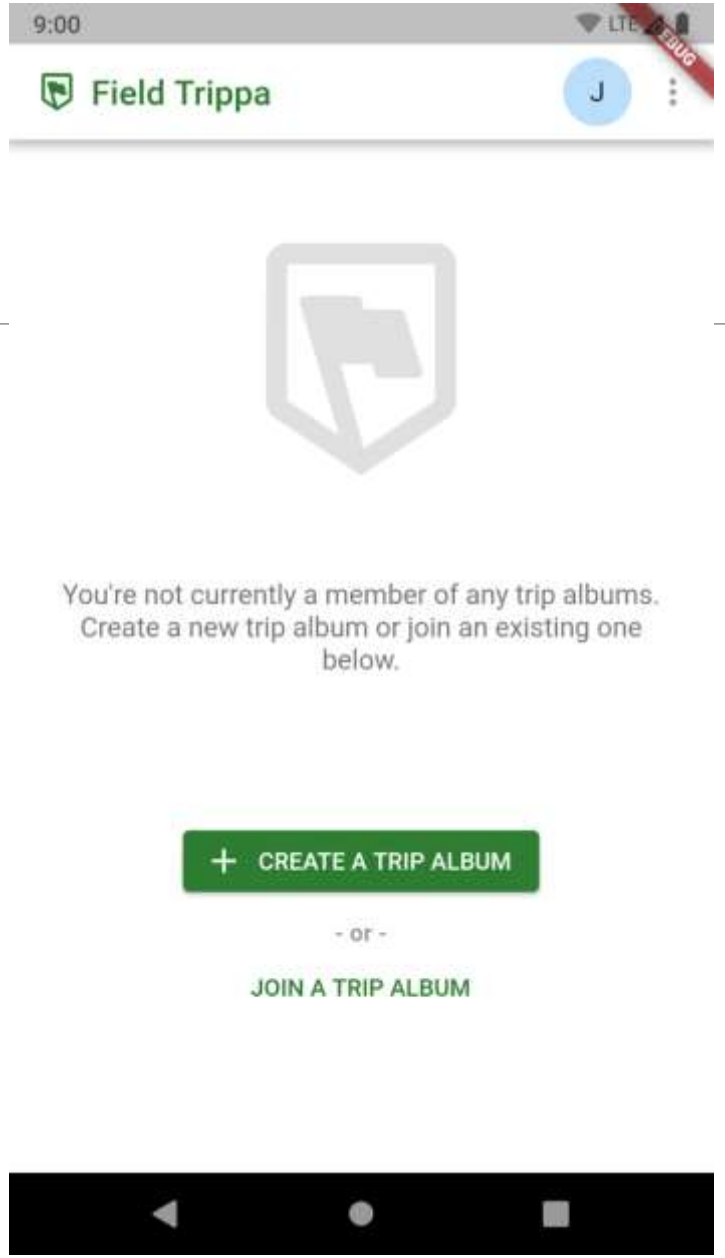
Google Sign-In has already been implemented using the google_sign_in flutter package. This package requires the google-services.json or GoogleService-Info.plist files that you have already copied into the project.

TECMAN LIMITED 2019

Run the app and sign in

- Run the application again and select "**Connect to Google Photos**". You'll be prompted to select a user account and accept the authentication scopes.

  If everything has been set up successfully, you'll see an empty list on the next screen.



TECMAN LIMITED 2019

Create an album

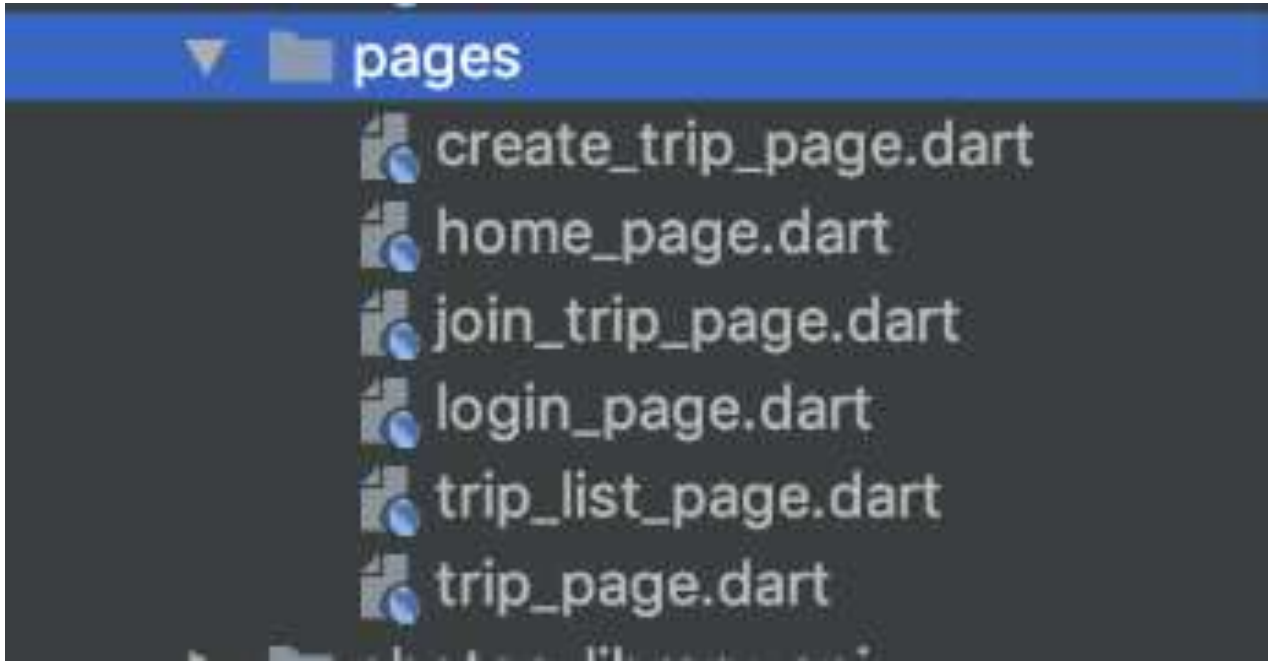
Before you implement the first API call to the Google Photos Library API, let's walk through the data architecture that the "Field Trippa" app uses.

TECMAN LIMITED 2019

Create an album

App Architecture

- Each screen is implemented as a separate page:



Create an album

- The PhotosLibraryApiModel describes the data model of the application and abstracts the Google Photos Library API calls away.
- The HTTPS REST calls to the Library API are implemented in the PhotosLibraryApiClient. Each call provided by this class takes a *Request object where you specify parameters and options for a call.
- The Library API requires user authentication via OAuth2. The access token required to be included in all API calls is set directly by the google_sign_in package on the PhotosLibraryApiClient.

Implement the create albums API call

Each trip is stored as an album in Google Photos. When you select the **"CREATE A TRIP ALBUM"** button, you should prompt the user for the name of the trip and create a new album with this name as its title.

TECMAN LIMITED 2019

Implement the create albums API call

- In create_trip_page.dart, write the logic that makes a request to the Library API to create the album. Implement the _createTrip(...) method at the end of the file to call the PhotosLibraryApiModel with the name of the trip the user entered.

Implement the create albums API call

lib/pages/create_trip_page.dart

```
Future<void> _createTrip(BuildContext context) async {
  // Display the loading indicator.
  setState(() => _isLoading = true);

  await ScopedModel.of<PhotosLibraryApiModel>(context)
      .createAlbum(tripNameFormController.text)
      .then((Album album) {

    // Hide the loading indicator.
    setState(() => _isLoading = false);
    Navigator.pop(context);
  });
}
```

Implement the create albums API call

- Implement the call to the Library API that creates the album. In the API model, implement the createAlbum(...) method that takes the title of the album as a parameter. It makes a call to the PhotosLibraryApiClient where the actual REST call is made.

Implement the create albums API call

lib/model/photos_library_api_model.dart

```
Future<Album> createAlbum(String title) async {
  return client
      .createAlbum(CreateAlbumRequest.fromTitle(title))
      .then((Album album) {
    updateAlbums();
    return album;
  });
}
```

Implement the create albums API call

- Implement the REST call to create the album in photos_library_api_client.dart. Remember that the CreateAlbumRequest already contains the title property-required for this call.

The following encodes it as JSON and adds the authentication headers to authorize the request. Finally, return the album created by the API.

Implement the create albums API call

lib/photos_library_api/photos_library_api_client.dart

```dart
Future<Album> createAlbum(CreateAlbumRequest request) async {
  return http
      .post(
    'https://photoslibrary.googleapis.com/v1/albums',
    body: jsonEncode(request),
    headers: await _authHeaders,
  )
      .then(
    (Response response) {
      if (response.statusCode != 200) {
        print(response.reasonPhrase);
        print(response.body);
      }
      return Album.fromJson(jsonDecode(response.body));
    },
  );
}
```

Implement the create albums API call

All other REST calls required for the Library API are implemented for you. This is the only place where you need to update the `PhotosLibraryApiClient` and the model.

You can find the rest of the implementation in the `lib/photos_library_api/photos_library_api_client.dart` and `lib/model/photos_library_api_model.dart` files.

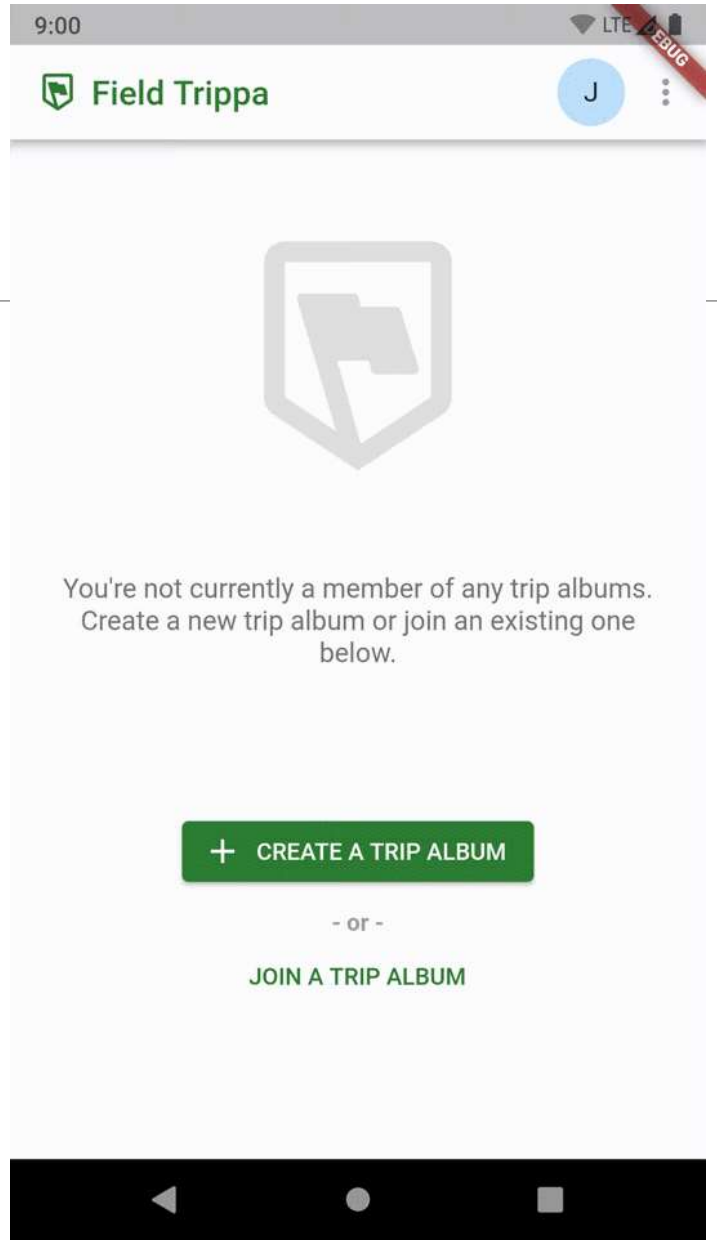
TECMAN LIMITED 2019

Implement the create albums API call

Try it out!

Deploy the app and select "**+ Create Trip**".

TECMAN LIMITED 2019



TECMAN LIMITED 2019

Only show your app's albums

You may have noticed that the trips list shows other albums from Google Photos that were not created by your app. (If you do not have any other albums in Google Photos and want to see this behaviour, open the Google Photos app and create an album. However, this is not required to continue in this codelab.)

Only show your app's albums

Remember that each trip is stored as an album in Google Photos. However, it does not make sense to show any other albums from Google Photos that were created through other means - Field Trippa should only show trips that the app has created.

You can use the API to restrict the list of trips that are displayed to show only those created by the app.

Only show your app's albums

- Modify the method listAlbums() method (NOT listSharedAlbums()!) in photos_library_api_client.dart. This method makes the REST call to retrieve a list of albums. Add the parameter excludeNonAppCreatedData=true that restricts the returned data to exclude albums that were not created by this app.

Only show your app's albums

lib/photos_library_api/photos_library_api_client.dart

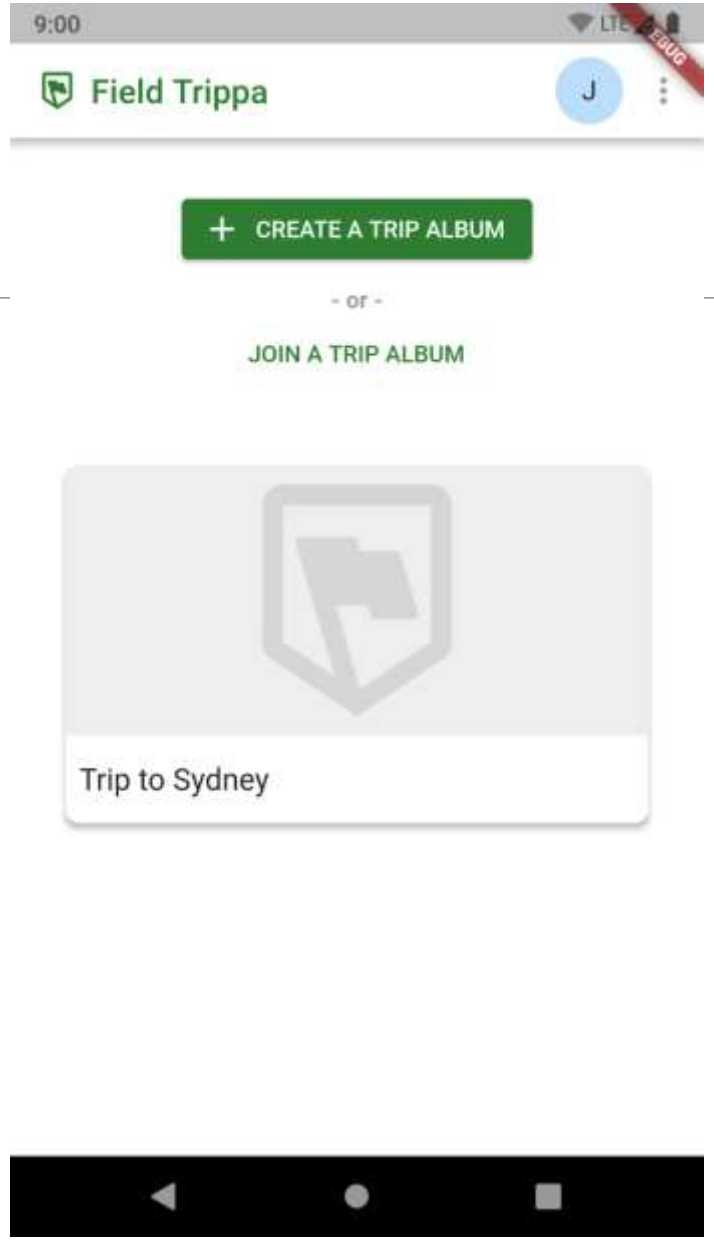
```
Future<ListAlbumsResponse> listAlbums() async {
  return http
      .get(
          'https://photoslibrary.googleapis.com/v1/albums?'
          'excludeNonAppCreatedData=true&pageSize=50',
          headers: await _authHeaders)
       ...
}
```

Only show your app's albums

Try it out!

The first page now only shows trips that were created by the app.

TECMAN LIMITED 2019



TECMAN LIMITED 2019

Upload Photos

The next step is to upload photos to a trip. The data is stored in your user's Google Photos account, so you don't have to worry about storage or processing the data yourself.

TECMAN LIMITED 2019

Upload Photos

Taking a photo in Flutter

First, implement the method _getImage(...) in the contribute photo dialog. This method is called when the user clicks the "**+ADD PHOTO**" button.

Upload Photos

The following code uses the image_picker package to take a photo, update the UI and call the API model to upload the image. (You'll implement this in the next step.) The _getImage(...) call stores an upload token needed later to create the photo in Google Photos.

Upload Photos

lib/components/contribute_photo_dialog.dart

TECMAN LIMITED 2019

Upload Photos

```
Future _getImage(BuildContext context) async {
  // Use the image_picker package to prompt the user for a photo from their
  // device.
  final File image = await ImagePicker.pickImage(
    source: ImageSource.camera,
  );

  // Store the image that was selected.
  setState(() {
    _image = image;
    _isUploading = true;
  });

  // Make a request to upload the image to Google Photos once it was selected.
  final String uploadToken =
      await ScopedModel.of<PhotosLibraryApiModel>(context)
          .uploadMediaItem(image);

  setState(() {
    // Once the upload process has completed, store the upload token.
    // This token is used together with the description to create the media
    // item later.
    _uploadToken = uploadToken;
    _isUploading = false;
  });
}
```

Upload Photos

Implement Library API call to upload the image to get an upload token

Uploading photos and videos to the Library API is done in two steps:

- Upload the media bytes to receive an *upload token*
- Create a *media item* in the user's library from the *upload token*

Upload Photos

Implement the REST request to upload media. You need to set some headers to specify the type of upload request and the filename. In the file photos_library_api_client.dart implement the method uploadMediaItem(...) where the file is uploaded, returning the *upload token* that the HTTP call returns:

Upload Photos

lib/photos_library_api/photos_library_api_client.dart

TECMAN LIMITED 2019

Upload Photos

```
Future<String> uploadMediaItem(File image) async {
  // Get the filename of the image
  final String filename = path.basename(image.path);
  // Set up the headers required for this request.
  final Map<String, String> headers = <String,String>{};
  headers.addAll(await _authHeaders);
  headers['Content-type'] = 'application/octet-stream';
  headers['X-Goog-Upload-Protocol'] = 'raw';
  headers['X-Goog-Upload-File-Name'] = filename;
  // Make the HTTP request to upload the image. The file is sent in the body.
  return http
      .post(
    'https://photoslibrary.googleapis.com/v1/uploads',
    body: image.readAsBytesSync(),
    headers: await _authHeaders,
  )
      .then((Response response) {
    if (response.statusCode != 200) {
      print(response.reasonPhrase);
      print(response.body);
    }
    return response.body;
  });
}
```

Upload Photos

Create a media item from an upload token

Next, implement the creation of a media item in the user's library from the upload token.

Creating a media item requires the upload token, an optional description (for example, the caption of the photo or video) and the optional identifier of an album. Field Trippa always adds the uploaded photo directly to a trip album.

Upload Photos

Implement the call to the photos_library_api_client that calls mediaItems.batchCreate with the upload token, description, and album ID. In the API model, implement the method createMediaItem(...) that calls the Library API. This method returns a media item.

(The photos_library_client for this call is already implemented.)

Upload Photos

lib/model/photos_library_api_model.dart

TECMAN LIMITED 2019

Upload Photos

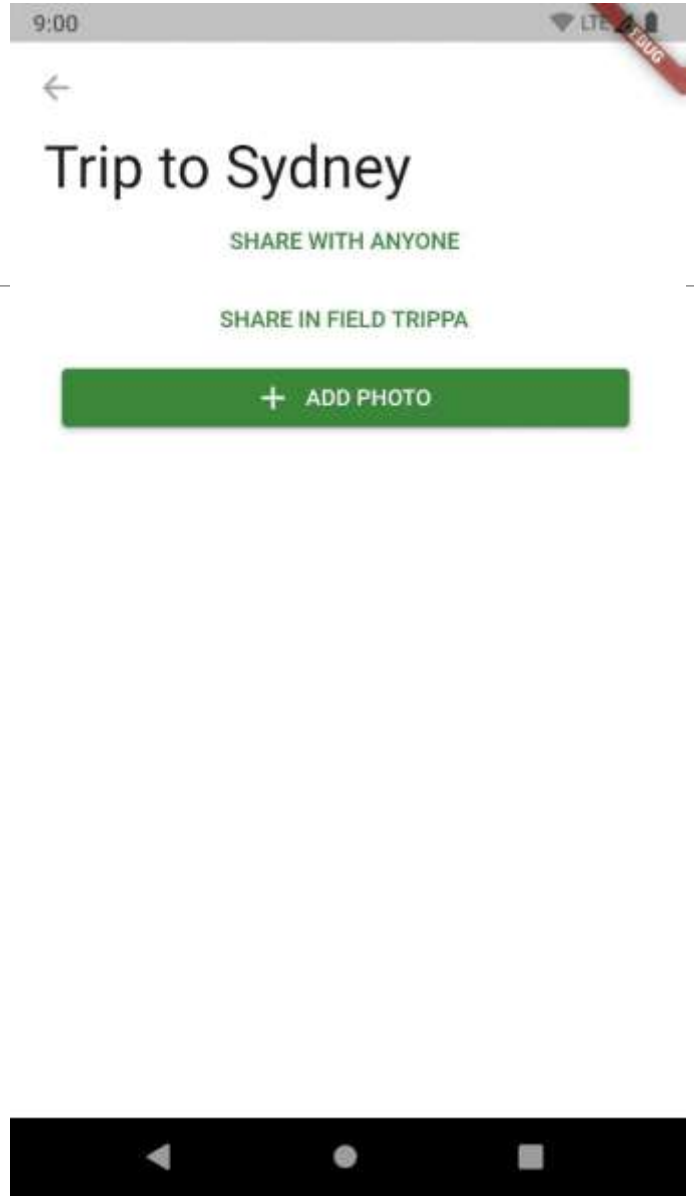
```
Future<BatchCreateMediaItemsResponse> createMediaItem(
    String uploadToken, String albumId, String description) {
  // Construct the request with the token, albumId and description.
  final BatchCreateMediaItemsRequest request =
      BatchCreateMediaItemsRequest.inAlbum(uploadToken, albumId, description);

  // Make the API call to create the media item. The response contains a
  // media item.
  return client
      .batchCreateMediaItems(request)
      .then((BatchCreateMediaItemsResponse response) {
    // Print and return the response.
    print(response.newMediaItemResults[0].toJson());
    return response;
  });
}
```

Upload Photos

Try it out!

Open the app and select a trip. Click **contribute** and select a photo that you have taken previously. Enter a description and select **upload**. The image should appear in the trip after a few seconds.

Open the album in the Google Photos app - you'll see the new image in the album of this trip.



TECMAN LIMITED 2019

Share albums with non-app users

So far you have implemented the functionality to create a trip and upload photos with a description into it. In the backend, each trip is stored as an album in Google Photos.

Next, you will share a trip with others who are not using your application.

Each trip is backed by an album in Google Photos, therefore you can 'share' an album via a URL and make it available to anyone who has this URL.

Share albums with non-app users

Implement the call to share an album

Albums are shared from the trip page when a **share** button at the top of the album is pressed.

TECMAN LIMITED 2019

Share albums with non-app users

- Implement the asynchronous call _shareAlbum(...) that calls to the model to share the album and then reloads the displayed album. By reloading the album, the shareInfo property is propagated which contains the shareableUrl that you'll show the user in a dialog later.

Share albums with non-app users

lib/pages/trip_page.dart

```
Future<void> _shareAlbum(BuildContext context) async {
  // Show the loading indicator
  setState(() {
    _inSharingApiCall = true;
  });
  final SnackBar snackBar = SnackBar(
    duration: Duration(seconds: 3),
    content: const Text('Sharing Album...'),
  );
  Scaffold.of(context).showSnackBar(snackBar);
  // Share the album and update the local model
  await ScopedModel.of<PhotosLibraryApiModel>(context).shareAlbum(album.id);
  final Album updatedAlbum =
      await ScopedModel.of<PhotosLibraryApiModel>(context).getAlbum(album.id);
  print('Album has been shared.');
  setState(() {
    album = updatedAlbum;
    // Hide the loading indicator
    _inSharingApiCall = false;
  });
}
```

Share albums with non-app users

Implement the method _showShareableUrl(...) that is called when the user clicks the "**SHARE WITH ANYONE**" button at the top of the page. First, check if the album has already been shared and call the method _showUrlDialog(...) once it has been shared.

Share albums with non-app users

lib/pages/trip_page.dart

```
void _showShareableUrl(BuildContext context) {
  if (album.shareInfo == null || album.shareInfo.shareableUrl == null) {
    print('Not shared, sharing album first.');
    // Album is not shared yet, share it first, then display dialog
    _shareAlbum(context).then((_) {
      _showUrlDialog(context);
    });
  } else {
    // Album is already shared, display dialog with URL
    _showUrlDialog(context);
  }
}
```

Share albums with non-app users

Implement the method _showUrlDialog(...) that shows the shareableUrl in a dialog.

TECMAN LIMITED 2019

Share albums with non-app users

lib/pages/trip_page.dart

```
void _showUrlDialog(BuildContext context) {
  print('This is the shareableUrl:\n${album.shareInfo.shareableUrl}');

  _showShareDialog(
      context,
      'Share this URL with anyone. '
      'Anyone with this URL can access all items.',
      album.shareInfo.shareableUrl);
}
```

Share albums with non-app users

Try it out!

The app only lists trips that are **not** shared yet on the main screen. Don't worry, we'll implement that in the next step. For now, you can simply create a new trip if you navigate away from the screen.

Open the app and select a trip. Select "**SHARE WITH ANYONE**" at the top of the screen and open the returned URL in your browser. (Tip: the URL is also printed to the log, so you can easily copy it on your computer. In Android Studio, the log is displayed in the "**Run**" tab.)

Share albums with non-app users

TECMAN LIMITED 2019

Share albums inside your app

In Google Photos, albums can be shared via a URL that anyone with access to the URL can access. Through the Library API you can also share albums via *share tokens*. A share token is a string that is used inside your application to join users to a shared album via the API.

Share albums inside your app

The process for sharing an album by your application via the Library API looks like this:

1. User A logs into your application and authorizes the Library API
2. Create the album
3. Share the album using the identifier of the album
4. Transfer the *share token* to another User

TECMAN LIMITED 2019

Share albums inside your app

The joining process is similar:

5. User B logs into your application and authorizes the Library API
6. Retrive the *share token* for the album the user should join
7. Join the album using the *share token*

TECMAN LIMITED 2019

Share albums inside your app

How *share tokens* are exchanged between users to join them to an album depends on your application architecture and backend.

In this codelab, the token is displayed on screen and the app asks the user to copy and paste it from the log output. In a real world application you might consider sharing tokens via URLs to your app, or via other means inside your application.

Share albums inside your app

Shared albums are shown inside Google Photos on the "sharing" tab.

TECMAN LIMITED 2019

Share albums inside your app

Display the share token

In the previous step you already implemented the method _shareAlbum(...) that shares an album.
The shareInfo property also contains the "*share token*" that will be shown on screen.

On the trip page, implement the method _showShareToken(...) that is called when the user presses the "**SHARE WITH FIELD TRIPPA**" button on screen.

Share albums inside your app

lib/pages/trip_page.dart

```
void _showShareToken(BuildContext context) {
  if (album.shareInfo == null) {
    print("Not shared, sharing album first.");
    // Album is not shared yet, share it first, then display dialog
    _shareAlbum(context).then((_) {
      _showTokenDialog(context);
    });
  } else {
    // Album is already shared, display dialog with token
    _showTokenDialog(context);
  }
}
```

Share albums inside your app

Next, implement the display of the "share token" in the method _showTokenDialog(...). The token is part of the shareInfo property of an album.

TECMAN LIMITED 2019

Share albums inside your app

lib/pages/trip_page.dart

```
void _showTokenDialog(BuildContext context) {
  print('This is the shareToken:\n${album.shareInfo.shareToken}');

  _showShareDialog(
      context, 'Use this token to share', album.shareInfo.shareToken);
}
```

TECMAN LIMITED 2019

Share albums inside your app

List shared albums

The application currently only lists albums that are *owned* by the user, but not shared albums.

Only albums that the user has created or explicitly added to their Google Photos library are shown on the "Albums" screen inside the Google Photos app. Only these albums are returned when calling albums.list in the Library API. However, in our app the user can join other user's shared albums, which are only returned in the call to list shared albums. You need to change the way the list of trips (albums) are retrieved from the Library API to include both *owned* and *shared* albums.

Share albums inside your app

Albums are loaded and cached in the API Model. Change the implementation of updateAlbums() in the model to load albums and shared albums, before storing them as one list.

This implementation uses multiple Futures to list the albums asynchronously before combining them into the list of cached albums. Delete the old implementation and comment out the new code.

Share albums inside your app

lib/model/photos_library_api_model.dart

TECMAN LIMITED 2019

Share albums inside your app

```
void updateAlbums() async {
  // Reset the flag before loading new albums
  hasAlbums = false;
  // Clear all albums
  _albums.clear();
  // Add albums from the user's Google Photos account
  // var ownedAlbums = await _loadAlbums();
  // if (ownedAlbums != null) {
  //   _albums.addAll(ownedAlbums);
  // }

  // Load albums from owned and shared albums
  final List<List<Album>> list =
      await Future.wait([_loadSharedAlbums(), _loadAlbums()]);

  _albums.addAll(list.expand((a) => a ?? []));

  notifyListeners();
  hasAlbums = true;
}
```

Share albums inside your app

Join a shared album

You can join users of your application to an album by using the share token. This is done through a simple text dialog in this codelab.

Implement the _joinTrip method on the join trip page that calls the API model with the share token the user has entered. First, display the loading indicator, then make the call to join the shared album with the input from the text form, before hiding the loading indicator and returning back to the previous screen.

Share albums inside your app

lib/pages/join_trip_page.dart

```dart
Future<void> _joinTrip(BuildContext context) async {
  // Show loading indicator
  setState(() => _isLoading = true);

  // Call the API to join an album with the entered share token
  await ScopedModel.of<PhotosLibraryApiModel>(context)
      .joinSharedAlbum(shareTokenFormController.text);

  // Hide loading indicator
  setState(() => _isLoading = false);

  // Return to the previous screen
  Navigator.pop(context);
}
```

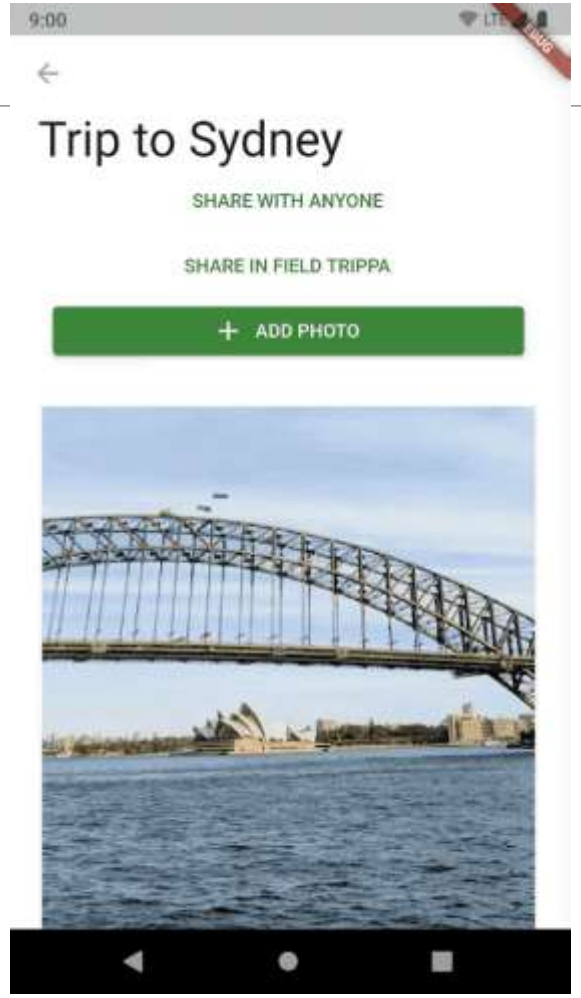
Share albums inside your app

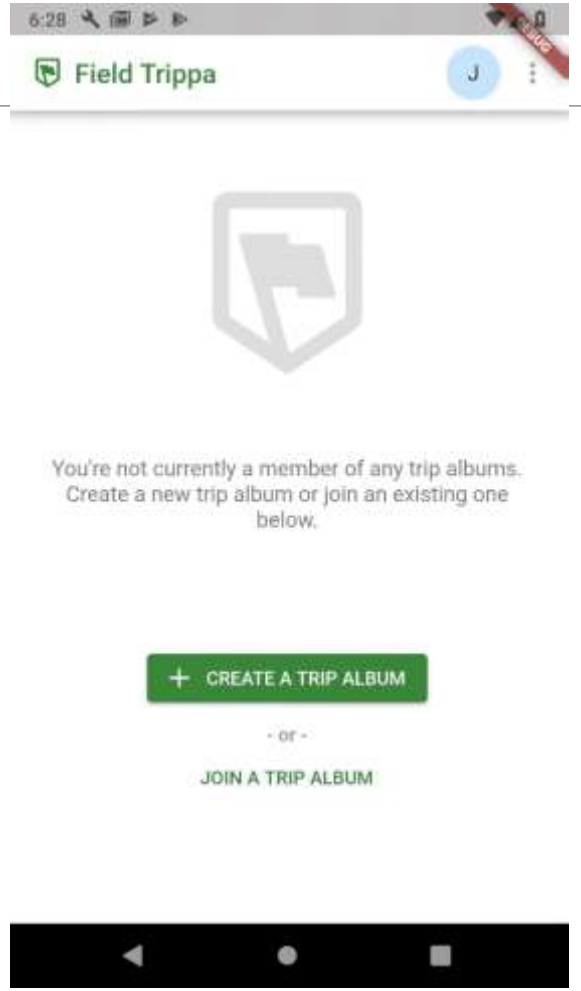
Try it out!

You need a second device or emulator with a different user account to try out this part of the codelab.

Create and share a trip under one user, then select the "**SHARE IN FIELD TRIPPA**" option to retrieve the *share token*. Copy this share token to the other device or emulator and enter it via the "**JOIN A TRIP ALBUM**" option on the home page. (Tip: The clipboard between your emulators and your host computer is shared.)

Share albums inside your app





Summary

What you have built

- Implemented sharing functionality into your application, backed by Google Photos
- Create your own photo and video sharing experiences on top of the Google Photos Library API, without having to worry about infrastructure or storage
- Using the sharing functionality that is part of the API in interesting and novel ways to share content directly to your users.
- Used some key parts of the Library API:
- Created new albums and uploaded new photos
- Listed shared albums, limited to albums created by your application

THANK YOU

TECMAN LIMITED 2019