Project 5: Wasteagram

Due Mar 14 by 11:59pm **Points** 15
Submitting a website url, a media recording, or a file upload
Available Mar 1 at 8am - Mar 16 at 11:59pm 16 days

This assignment was locked Mar 16 at 11:59pm.

Introduction

Demonstrate proficiency of prior material, and challenge yourself to demonstrate how to apply new concepts to meet functional requirements. Build an application for recording food waste. Practice applying the concepts of location services, camera / image picker, permissions, forms, navigation, lists, asynchronous programming, streams, and Firebase backend services. Enhance your application with analytics, crash reporting, accessibility, internationalization, debugging and automated testing.

Note: This is the portfolio assignment. You are allowed to use another name for your app, rather than Wasteagram, as long as the functionality remains exactly the same.

Learning Outcomes

1. Obtain device location information and integrate the use of the camera or photo gallery. (Module 9 MLO 1)
2. Demonstrate persistence with remote storage services, such as Firebase Cloud Storage and a Firestore database. (Module 9 MLOs 2 & 3)
3. Invoke asynchronous methods, employ navigation, capture form data, and display data in ListView components and detail screens. (Module 7 MLOs 2 - 4)
4. Implement unit tests to validate application behavior. (Module 10 MLO 1)
5. Incorporate the Semantics widget to facilitate accessibility features of native platforms. (Module 10 MLO 2)
6. Demonstrate the use of analytics, crash reporting, and debugging tools (Module 10 MLOs 3 & 4)

Scenario

Your client, Matthew Peter, is the owner of TwentySix Cafe, a Portland coffee shop.

"Man, I am so tired of these wasted bagels and pastries we have at the end of every day!" he says. "I'm losing money, and it's so wasteful... I feel like there's an episode of Portlandia about this. I mean, why waste a donut? A *donut*!"

Mr. Peter wants his employees to run an application that, "is like Instagram, but for food waste," he says. Every night the person closing the shop can gather up the leftover baked goods, take out their phone, start the app, and create a post consisting of a photo of the wasted food and the number of leftover items.

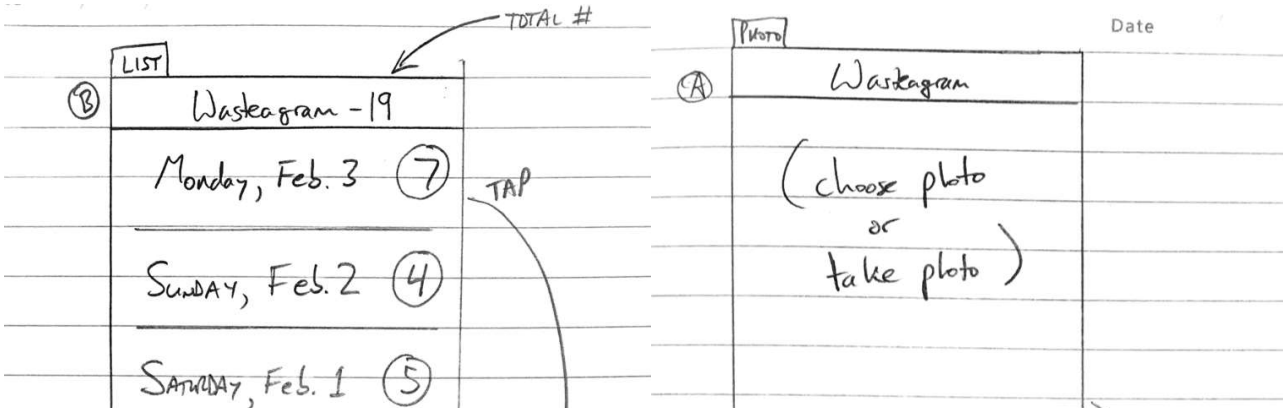
leftover items.

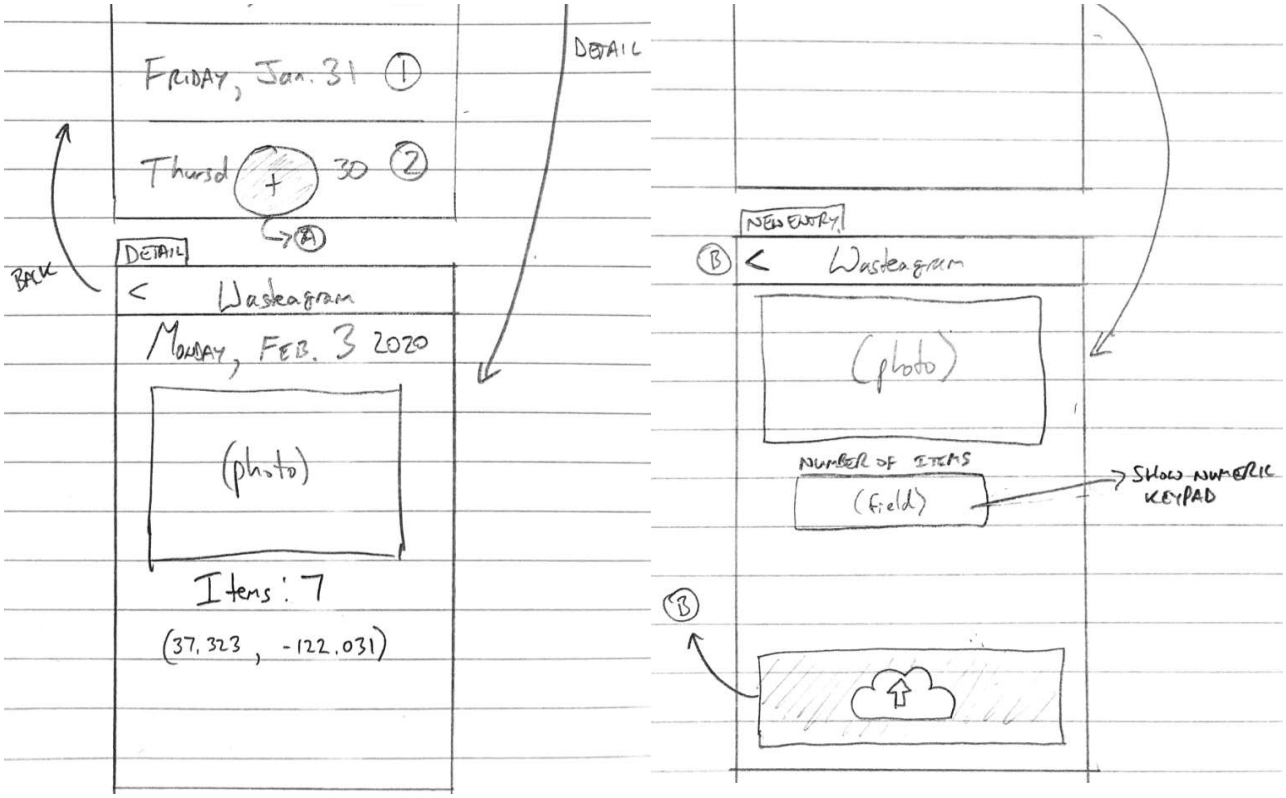
"If only I could see a list of these posts over time, then at least I'd know how much money I'm losing, and I could make adjustments to my pastry orders," he says, dreamily. "No more forsaken donuts!"

You have engaged Matthew Peter in a paid contract to develop a functioning version of the application that he and his employees can try out at the coffee shop. "Hey, I know," he says, "Let's call it *Wasteagram*."

What to Do

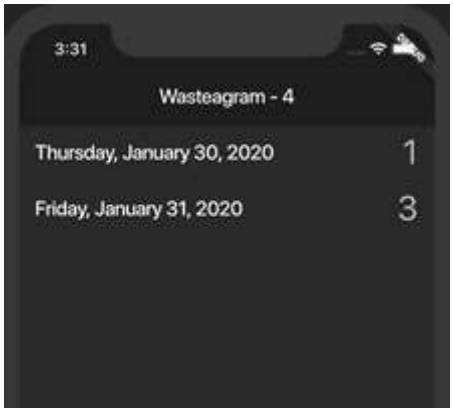
Implement *Wasteagram*, a mobile app that enables coffee shop employees to document daily food waste in the form of "posts" consisting of a photo, number of leftover items, the current date, and the location of the device when the post is created. The application should also display a list of all previous posts. After discussing the requirements with the client and sketching out the UX flow, your design notebook describes the app like this:





Download full-size pdf (https://oregonstate.instructure.com/courses/1799015/files/82696122/download?download_frd=1)

Here is a demonstration of a generic version of such an app:



Download Larger Version

The **functional requirements** are:

1. Display a circular progress indicator when there are no previous posts to display in the List Screen.
2. The List Screen should display a list of all previous posts, with the most recent at the top of the list.
3. Each post in the List Screen should be displayed as a date, representing the date the post was created, and a number, representing the total number of wasted items recorded in the post.
4. Tapping on a post in the List Screen should cause a Detail Screen to appear. The Detail Screen's *back* button should cause the List Screen to appear.
5. The Detail Screen should display the post's date, photo, number of wasted items, and the latitude and longitude that was recorded as part of the post.
6. The List Screen should display a large button at the center bottom area of the screen.
7. Tapping on the large button enables an employee to capture a photo, or select a photo from the device's photo gallery.
8. After taking a new photo or selecting a photo from the gallery, the New Post screen appears.
9. The New Post screen displays the photo of wasted food, a *Number of Items* text input field for entering the number of wasted items, and a large *upload* button for saving the post.
10. Tapping on the *Number of Items* text input field should cause the device to display its numeric keypad.

11. In the New Post screen, tapping the *back* button in the app bar should cause the List Screen to appear.
12. In the New Post screen, tapping the large *upload* button should cause the List Screen to appear, with the latest post now appearing at the top of the list.
13. In the New Post screen, if the *Number of Items* field is empty, tapping the *upload* button should cause a sensible error message to appear.

In addition to the functional requirements above, your application should meet the following **technical requirements**:

1. Use the *location, image_picker, cloud_firestore, and firebase_storage* packages to meet the functional and technical requirements.
2. Incorporate the use of Firebase Cloud Storage and Firebase Cloud Firestore for storing images and post data.
3. Data should not be stored locally on the device.
4. On the List Screen, the application should display the posts stored in the Firestore database.
5. On the Detail Screen, the application should display the image stored in the Cloud Storage bucket.
6. On the New Post screen, tapping the large *upload* button should store a new post in the Firestore database.
7. Each "post" in Firestore should have the following attributes: *date, imageURL*, *quantity*, *latitude* and *longitude*.
8. The application should incorporate the Semantics widget in multiple places, such as interactive widgets like buttons, to aid accessibility.
9. The codebase should incorporate a model class.
10. The codebase should incorporate a few (two or three) simple unit tests that test the model class.

The functional and technical requirements specifically exercise the Exploration content and our module learning outcomes. In addition, here are some **optional extra credit requirements that can add up to 4% to your overall class grade**:

1. The app bar of the List Screen should display the total sum of the number of wasted items in all posts (extra 1% added to class grade)
2. Add integration tests that verify any one particular part of the UX flow (extra 1% added to class grade)
3. Integrate the use of in-app analytics (Analytics) to monitor application usage (extra 1% added to class grade)
4. Integrate the use of crash reporting (Sentry or Crashlytics) to record application crashes (extra 1% added to class grade)

Consider the following steps, and consult the ***Tips*** below.

▶ 1

Create a new flutter application via:

```
flutter create --org edu.oregonstate.YOURONID wasteagram
```

This ensure that your app has an intentional app id.

Step 2

Create a new Firebase project, and add an iOS and/or Android application to the project. Configure your local iOS/Android projects, and be sure you can build your app without issues.

Pro Tip: Create your Firestore database first, with its initial collection and some sample data, prior to

storing objects in Cloud Storage.

In Firebase, create a Firestore database with your first named collection, eg 'posts'. Enable Cloud Storage, and modify the authorization file to not require authentication.

Step 3

Define a simple *main.dart*, an *app.dart*, and appropriate *models*, *screens*, and *widgets* subdirectories. Start with the basic requirements of the List Screen.

Step 4

Implement the UX flow as appropriate screens and navigation in your codebase.

Step 5

Focus on storing the photo in Cloud Storage and retrieving the URL first. Iteratively add new features to your application. See the ***Tips*** below. Pay attention to quality, especially about how your project is organized, whether you have incorporated a simple model, avoided significant nesting, defined and used many of your own custom composed widgets, and so on. The Explorations provide valuable insight on each functional and technical requirement.

Note: If you store an object in Cloud Storage prior to creating a collection in Firestore, the Firebase console may ask if you wish to use the "native" or "real-time" database. Choose the "native" database type. You can't switch once data is stored in the database. One solution is to create a new Firebase project. If you do this, it's no big deal, but don't forget to add your iOS/Android app and update the Firebase config file in your local iOS/Android projects (just overwrite the old *.json* config file with the new one).

You'll know this is complete when you have a program that behaves like the example UX flow displayed above, meets the description of the functional and technical requirements, stores images and post data in Firebase, passes its tests, and your code exhibits a competency of using both Dart language features and Flutter best practices. Be sure to review the rubric.

▶ Resources

There are many resources for this project, including our module reading assignments, the Explorations, the resources linked from within the Explorations, the official Dart documentation, and videos from the Flutter development team. Your goal is to synthesize these learning materials, apply them to a problem domain, and demonstrate that you are comfortable with the fundamentals of mobile software development with an SDK.

If these resources feel overwhelming, start with the readings in the module Overview pages, and then the Explorations.

- Readings listed in the **Module 9** and **Module 10** Overviews

- Readings listed in the Module 6 and Module 10 Overviews
- Exploration: Navigation and Routing (Week 7 Module)
- Exploration: Forms (Week 7 Module)
- Exploration: ListViews (Week 7 Module)
- Exploration: Asynchronous Functions (Week 8 Module)
- Exploration: Platform Hardware Services (Week 9 Module)
- Exploration: Firebase Cloud Firestore & Storage (Week 9 Module)
- Exploration: Testing and Debugging (Week 10 Module)
- Exploration: Accessibility and Internationalization (Week 10 Module)
- Exploration: Analytics and Crash Reporting (Week 10 Module)
- Exploration: Performance (Week 10 Module)
- Flutter Documentation (https://flutter.dev/docs)
- Flutter Cookbooks (https://flutter.dev/docs/cookbook)
- Flutter in Focus: Using Firestore as a Backend to Your Flutter App (https://www.youtube.com/watch?v=DqJ_KjFzL9I)
- Firebase Documentation: Add Firebase to Your Flutter App (Android) (https://firebase.google.com/docs/flutter/setup?platform=android)
- Firebase Documentation: Add Firebase to Your Flutter App (iOS) (https://firebase.google.com/docs/flutter/setup?platform=ios)
- location (https://pub.dev/packages/location)
- image_picker (https://pub.dev/packages/image_picker)
- firebase_storage (https://pub.dev/packages/firebase_storage)
- cloud_firestore (https://pub.dev/packages/cloud_firestore)
- Documentation: An Introduction to Unit Testing (https://flutter.dev/docs/cookbook/testing/unit/introduction)
- Documentation: An Introduction to Widget Testing (https://flutter.dev/docs/cookbook/testing/widget/introduction)
- Documentation: An Introduction to Integration Testing (https://flutter.dev/docs/cookbook/testing/integration/introduction)

Tips

- Consider starting with a "throwaway" app, to ensure that you know how to set up Firebase for your project and can store images in Cloud Storage and data in Cloud Firestore.

- Focus on obtaining an image File first, storing it in Cloud Storage, and obtaining the URL for the image. (See the Note above.)
- Consider passing the File and *imageURL* to the constructor of the widget responsible for the New Post screen.
- For this app, it's probably better to not used named routes. But you can.
- Determine the latitude and longitude just prior to sending the data to Cloud Firestore.
- Use the intl (https://pub.dev/packages/intl) package for formatting dates as String values for display.
- When creating the path for a StorageReference, consider using a String created via

- When creating the path for a StorageReference, consider using a String created via `DateTime.now()` to ensure a unique name.
- Keep your project files organized. Use subdirectories like *model*, *screens*, *widgets*, etc. Keep *main.dart* and perhaps *app.dart* in your project's root directory.
- Use version control. This enables you to create feature branches, experiment, and either back out when you need to try something else or merge your success into *master*.
- Take small, simple steps, and run your program frequently. Compare your current program's behavior with the specification, identify the first minor difference, and then try to solve that single feature.

When you encounter errors, be sure to record the exact command you are running and the exact error output. Then,

1. Spend just a few minutes googling the main part of the error message.
2. Reach out on the course Piazza and Slack channel to seek advice from fellow learners and ask for assistance.

What to turn in

When complete, submit two things: a *.zip* file of your project's *lib* directory with a name formatted like *LNAME_FNAME.zip*, and a video recording of your computer screen that follows this sequence:

1. A Word document, or similar, open full screen with your full name in large letters.
2. A quick (45s) demonstration of the following kinds of interaction:
    1. Starting the app for the first time, displaying the circular progress indicator.
    2. Viewing the List Screen.
    3. Selecting a photo, creating a new post, and seeing the List Screen appear with the new post appearing in the list.
    4. Navigating and viewing the Detail Screen of the new post and navigating back.
    5. Running the test suite and showing that the tests are passing.
3. A quick (45s) overview of your project and code, showing:
    1. The file organization of your project, especially your *lib* directory.
    2. The contents of *pubspec.yaml*, especially the *dependencies* section.
    3. Quickly, the code of your model class.
    4. The code responsible for displaying the List Screen and its StreamBuilder.
    5. The code that stores the image in Cloud Storage.
    6. The code that stores the post data in Cloud Firestore.
    7. The code for the unit tests.
4. For extra credit
    1. Demonstrate
        1. That the total of all items appears in the app bar and is updated when a new post is created, and/or
        2. Running the integration tests and the tests passing, and/or

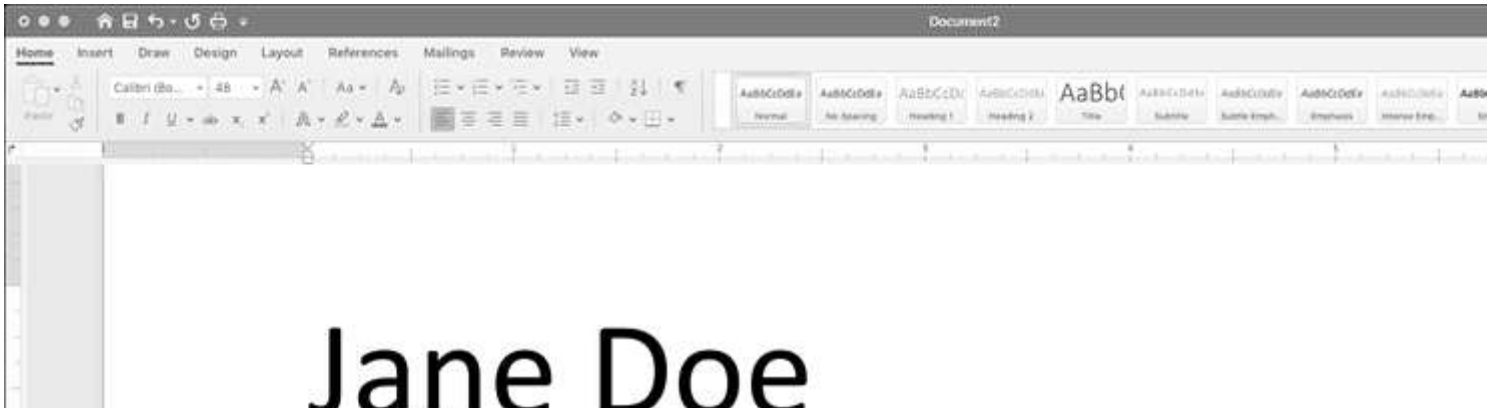
2. Running the integration tests and the tests passing, and/or
3. The application usage data in Firebase Analytics, and/or
4. Triggering some code that causes the app to crash and notify Sentry or Firebase Crashlytics.

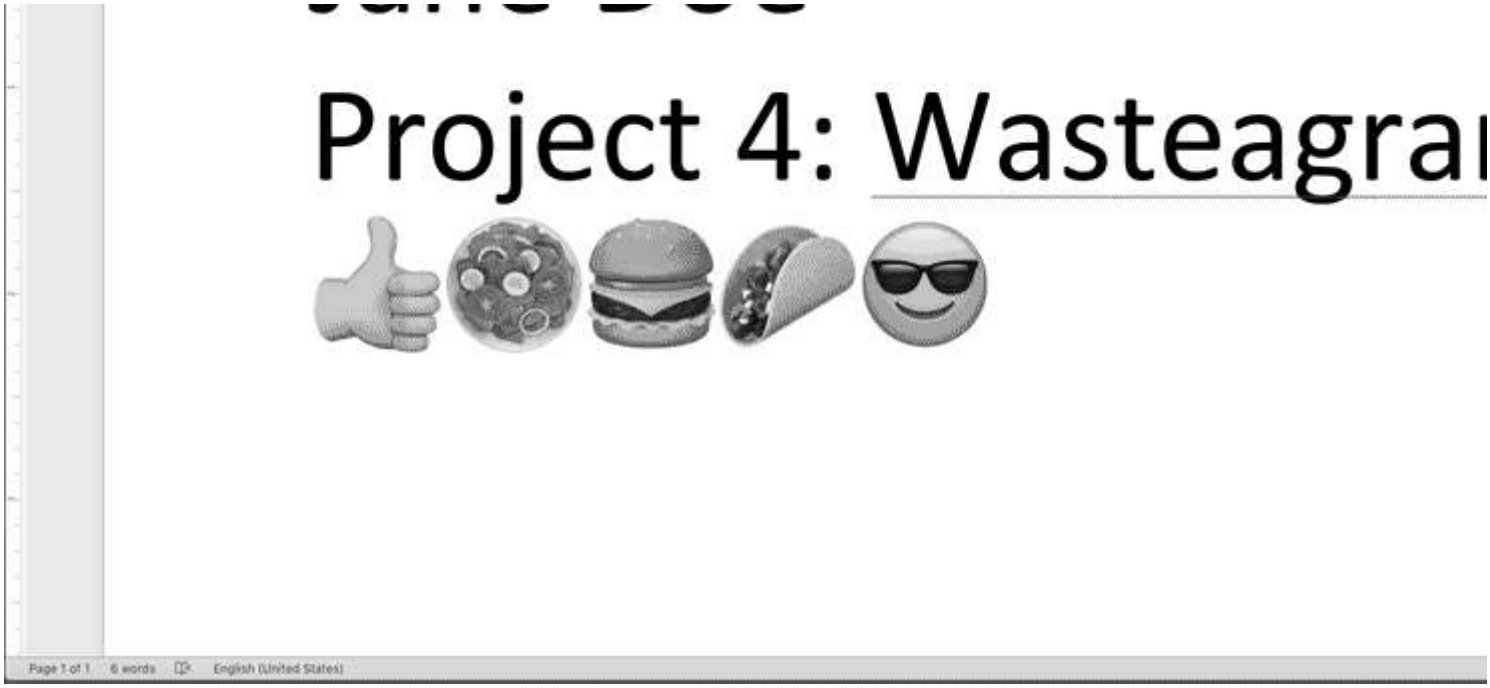
2. You can create a separate video for the extra credit items or demonstrate these in the same video as the required items.
3. If you are submitting any of the extra credit items, add a file EXTRA_CREDIT.txt listing which extra items you have implemented.

This video should be close to 90 seconds in length, and not exceed 120 seconds.

Pro Tip: Have your code in an editor window ready and practice your demonstration before recording your video.

Here is an example:





Grading criteria

This assignment is worth **17%** of your grade.

Be sure to consult the rubric for expectations regarding your submission.

Extensions

Want an extra challenge? Try the following once you have met the project requirements:

1. Obtain the real temperature using a weather API, and include the temperature in the post stored in Firestore. Display each post's temperature in the Detail Screen.
2. Use the **image (https://pub.dev/packages/image)** package to create thumbnails of the images. Store those in Cloud Storage as well, and display the thumbnails as the `leading` component of the ListTiles in the ListView.
3. Incorporate the popular "pull to refresh" feature in the List Screen.
4. Instead of a storing photos, try recording/selecting and storing videos.
5. Investigate how to release an Android app on **the Google Play store (https://flutter.dev/docs/deployment/android)** or **Apple App Store (https://flutter.dev/docs/deployment/ios)**, and deploy your app to either store.

Project 5: Wasteagram

Criteria	Ratings			Pts
Video content Video displays full name, program interaction, project organization and code of implementation. Video displays running the test suite.	**1 pts** **Proficient** Full name displayed large & visible at the start of the video. Interaction is visible, and all required interactions are shown. Code is legible, and quickly shows the project structure, pubspec.yaml, Firebase code, primary widgets, model class, etc. Video displays running the test suite and the tests pass.	**0.5 pts** **Moderate** Missing or illegible visibility on project file structure, pubspec.yaml, Firebase code, model, or widgets. Missing demonstrating the running of the test suite.	**0 pts** **Not Proficient** Name not displayed, or is too small. Program interaction not shown or is not visible. Project file organization is not shown or not legible. Source code is not shown or is not legible	1 pts
Code submission Zip file is named LNAME_FNAME.zip, and contains the contents of the project's lib directory.	**1 pts** **Proficient** Zip file properly named and contains only the contents of the lib directory.	**0.5 pts** **Moderate** Zip file misnamed, but does contain only the contents of the lib directory.	**0 pts** **Not Proficient** Zip file missing or does not contain only the contents of the lib directory.	1 pts
Project Organization pubspec.yaml declares the packages necessary to support technical requirements. Project files organized according to common conventions.	**1 pts** **Proficient** pubspec.yaml properly specified library dependencies. Source files reflect multiple widgets, and are organized in the lib directory tree.	**0.5 pts** **Moderate** Project organization exhibits that pubspec is missing a necessary dependency, OR source code not broken up across multiple, organized dart files.	**0 pts** **Not Proficient** Project organization exhibits that it is missing a library dependency AND source code not broken up across multiple, organized dart files.	1 pts
Implementation: Model Program defines a ▶ el class(es), with cent name and appropriate member variables and methods; and is used by the program (hints: FoodWastePost)	**1 pts** **Proficient** Project includes well-named model classes with member variables, constructor and member functions. Model is instantiated in the scope of the appropriate widgets. Model is mostly a "pure Dart" class and not dependent on widgets.	**0.5 pts** **Moderate** Model is defined, but is either not appropriately used in conjunction with the application, or, it is missing one or two properties, or contains widget-related code.	**0 pts** **Not Proficient** No model class defined or is present but not well-named, missing member variables and or methods. (Present but not useful / ill-defined.)	1 pts

Criteria	Ratings			Pts
Functional Requirement: List Screen Program displays a scrollable list of food waste posts retrieved from Firestore.	**1 pts** **Proficient** Displays a circular progress indicator when there is no data. Displays a list of food waste posts, presenting the date and number of items in each entry.	**0.5 pts** **Moderate** Does not display circular progress indicator when there are no posts. Post dates not formatted as a "Month DD" String, or does not display the number of wasted items in the post.	**0 pts** **Not Proficient** Feature not demonstrated. (No list of food waste posts.)	1 pts
Functional Requirement: Detail Screen Application displays details of each food waste post, when tapped on in the list.	**1 pts** **Proficient** Application displays the details of a food waste post when tapped on in the list. Navigating "back" from the detail screen displays the list screen. Details include photo, date, number of items, and latitude/longitude.	**0.5 pts** **Moderate** Exhibiting two or more of the following missing criteria: Application does not display photo in the detail screen. Application does not display the date and number of items. Application does not display the latitude and longitude. Missing navigating "back" from the detail screen to the list screen.	**0 pts** **Not Proficient** Feature not demonstrated. (No display of the food waste post details.)	1 pts
Functional Requirement: Selecting Photo Tapping the main FAB enables users to take a photo or select an image from the gallery.	**1 pts** **Proficient** Tapping a FAB enables the user to either the camera, to take a photo, or the image gallery, to select a photo.	**0.5 pts** **Moderate** Selecting a photo does not result in displaying the New Entry screen.	**0 pts** **Not Proficient** Feature not demonstrated. (No means of selecting a photo.)	1 pts
Functional Requirement: New Post Screen Application displays the New Post screen after the photo is selected.	**1 pts** **Proficient** Application displays the New Post screen after selecting a photo. New Post screen displays the selected photo, a text field for entering number of items, and a button for uploading the post.	**0.5 pts** **Moderate** Application either does not display the New Post screen after selecting a photo, or the New Post screen does not display the selected photo, does not display a text field, or does not display a button for uploading.	**0 pts** **Not Proficient** Feature not demonstrated. (No New Post screen displayed.)	1 pts
Functional Requirement: Creating a New Post Application enables users to enter number of items and upload the post to Firestore.	**1 pts** **Proficient** New Post screen displays a numeric keypad when the text field has focus. Main upload button triggers validation, displaying an error message when the Number of Items field is blank. Tapping the button causes date, number of items, image URL, and location information	**0.5 pts** **Moderate** App does not display a numeric keypad when the text field has focus, or the form is missing validation, or the main button does not cause the List Screen to appear. Tapping the button does not store date,	**0 pts** **Not Proficient** Feature not demonstrated. (No ability to create a new post.)	1 pts

Criteria	Ratings			Pts
Technical Requirement: Storing Photos in Cloud Storage Application stores photos in Cloud Storage	**1 pts** **Proficient** Application stores photos in Cloud Storage, and generates a unique name for each photo stored. Application retrieves the URL of the stored photo, for creating a food waste post. Viewing the Detail Screen retrieves the image from Cloud Firestore.	**0.5 pts** **Moderate** Application does use a Cloud Storage, but does not meet one of the following criteria. Does not retrieve the URL of the image to incorporate into the food waste post stored in Firestore. OR Does not retrieve the image from Cloud Storage when displaying the Detail Screen.	**0 pts** **Not Proficient** Feature not demonstrated. (No usage of a Cloud Storage.)	1 pts
Technical Requirement: Storing Post Data in Firestore Application stores the food waste post data in Firestore.	**1 pts** **Proficient** Application stores food waste post as a Firestore document with a date, image url, item count and latitude/longitude.	**0.5 pts** **Moderate** Application stores food waste post as in Firestore, but is missing either the date, image url, item count, latitude or longitude.	**0 pts** **Not Proficient** Feature not demonstrated. (No usage of Firestore.)	1 pts
Technical Requirement: Location Information Application obtains location data (lat/lon) and includes it in the uploaded food waste post.	**1 pts** **Proficient** Application requests permission for location services, obtains the latitude and longitude of the device or simulated device, and includes the data in the saved food waste post.	**0.5 pts** **Moderate** Application requests permission for location services, but fails to obtain the latitude and longitude of the device or simulated device.	**0 pts** **Not Proficient** Feature not demonstrated. (No location information obtained.)	1 pts
Technical Requirement: Accessibility Application incorporates the Semantics widget to [illegible] accessibility.	**1 pts** **Proficient** Application wraps interactive widgets, such as buttons, fields and tappable ListTiles in a Semantics widget, and includes meaningful labels and other appropriate properties.	**0.5 pts** **Moderate** Application does not wrap all interactive widgets in a Semantics widget, or is missing appropriate attributes, such as the label property.	**0 pts** **Not Proficient** Feature not demonstrated. (No use of Semantics widget.)	1 pts
Technical Requirement: Test Suite Project includes a passing test suite consisting of unit tests.	**1 pts** **Proficient** Project test suite passes, and includes unit tests that test the model.	**0.5 pts** **Moderate** Project test suite exists, but not all tests pass.	**0 pts** **Not Proficient** Feature not demonstrated. (No test implementations.	1 pts

Criteria	Ratings			Pts
Technical Requirement: Retrieving Data from Firestore Application retrieves food waste post data from Firestore for display in the List Screen.	**1 pts** **Proficient** List Screen uses a StreamBuilder to retrieve an update the list of food waste posts stored in Firestore. The ListView updates the displayed list of food waste posts when new posts are stored in the database.	**0.5 pts** **Moderate** List Screen does not retrieve food waste posts from Firestore, or does not properly integrate a StreamBuilder, or requires a hot restart to see new food waste posts appear in the list.	**0 pts** **Not Proficient** Feature not demonstrated. (List Screen does not display food waste posts retrieved from Firestore.)	1 pts
			Total Points: 15	