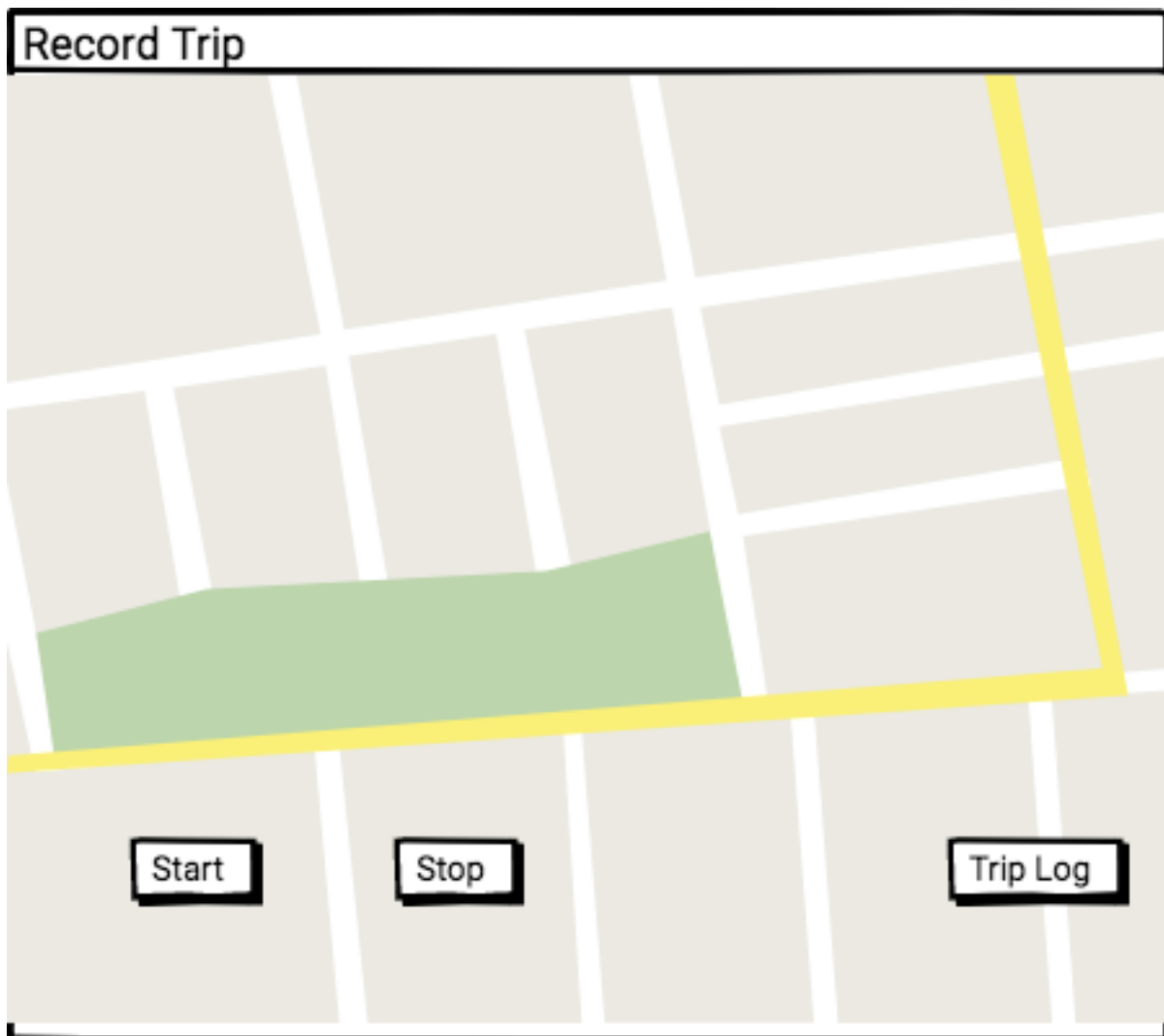


Capstone Project Proposal

Resubmit Updates

Application Widget Wireframe



The Application Widget for the home screen will present the view that displays the current trip being recorded or enable the user to begin recording a trip if a trip is not underway. The Widget on the home screen enables the user to see their current trip plot without opening the app and have easy access to the Start and Stop buttons for starting or stopping the trip.

The Trip Log button will open the app to the main Trip Log list view.

Share Button

(Reviewer suggested)

Share buttons, perhaps a FAB will be added to the map views and the list view to share the content via email or Facebook.

The Share buttons will be implemented as part of Task 8 in the project plan.

Required Functionality

Per the review:

Must implement at least one of the three

If it regularly pulls or sends data to/from a web service or API, app updates data in its cache at regular intervals using a SyncAdapter or JobDispatcher.

OR

If it needs to pull or send data to/from a web service or API only once, or on a per request basis (such as a search application), app uses an IntentService to do so.

OR

If it performs short duration, on-demand requests(such as search), app uses an AsyncTask.

Kindly add any one of these to qualify the rubric. If you are not able to find a use case where to use them you can find a small use case like downloading some Image to satisfy the requirement and keep the app implementation as it is.

Update

TrackMe receives location updates in the background when the user has initiated tracking a new trip. As described in the Application Architecture section, a **LocationProcessorService (an Intent Service)** handles

GitHub Username: chipk215

App Name: TrackMe

Proof of concept work: <https://github.com/chipk215/TrackMe>

Note to reviewer

Initially my plan was to build a fleet tracking SaaS solution which enabled a group or enterprise to track the location of a fleet of vehicles, like a pizza delivery franchise or a landscaping business with vehicles in the field. An android device would serve as both the sensor for obtaining the location of a vehicle and to provide views to see the tracks and positions of the vehicles in the fleet to which the user belonged.

During the proof of concept work it became clear that the scope of the fleet tracking project was both too large for the class and the work to build a multi-tenant solution required a significant amount of back end server work (joining into groups, group administration, authentication, etc.) that detracted from the android functionality that is the focus of the project. So I pivoted to a single user case.

Description

TrackMe makes it easy to create trip logs to augment expense records for mileage or fuel reimbursements. Start recording a trip with the touch of a button and your location will be displayed and updated on a map. Stop recording your movement and a trip log entry is created and saved for future reference.

Trip data is recorded in the background so the user can continue using their phone or tablet without interruption.

Trip logs can be filtered and sorted by date and/or distance to easily use when completing expense reports.

Trip data is securely stored only on your phone or tablet and is not accessible to anyone other than you.

processing location samples that have been received by the LocationUpdatesBroadcastReceiver. The prototype code for this intent service is listed below.

Additionally, at this point ***at least one AsyncTask*** has been defined and planned for the application. When a user starts a new trip, a trip segment that will represent the collection of location samples needs to be created. A StartSegmentTask which extends AsyncTask creates a segment id corresponding to the trip segment and writes this segment to SharedPreferences for reference when the first location samples from the FusedLocationProvider arrive. This is required because there appears to be a defect with Google LocationServices which prevents sending extras in the intents that start location sampling, see <https://tinyurl.com/y9gpvcoe>.

Anyway having both an IntentService (one of many services) and an AsyncTask should fulfill the project requirements.

Here is the initial implementation of the IntentService:

```
public class LocationProcessorService extends IntentService {

    private final static String NAME="LocationProcessorService";
    public final static String LOCATIONS_EXTRA_KEY =
"locationsKey";
    public final static String SEGMENT_ID_EXTRA_KEY =
"segmentIdKey";

    public final static String LOCATION_BROADCAST_PLOT_SAMPLE =

"com.keyeswest.trackme.location.samples.intent.action.PLOT_SAMPL
E";

    public final static String PLOT_SAMPLES_EXTRA_KEY =
"sampleData";

    public LocationProcessorService() {
        super(NAME);
    }
}
```

```

    }

    /**
     * A new location sample has arrived.
     *
     * Generally, we want to save the location in the db and
    update the segment to which the
     * location belongs.
     *
     * Updating the segment includes updating the segment
    distance by computing the distance
     * the new location is from the last location sample (unless
    this is the first location of
     * the segment, and then updating the bounding lat/lon box
    associated with the segment.
     * @param intent
     */
    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        Double segmentDistance=0d;

        Timber.d("Entering LocationProcessorService
    onHandleIntent");
        if (intent != null) {
            LocationResult result =
    intent.getParcelableExtra(LOCATIONS_EXTRA_KEY);

            //get the location samples
            List<Location> locations = result.getLocations();

            if ((locations != null) && (locations.size() > 0)) {

                String segmentId =
    intent.getStringExtra(SEGMENT_ID_EXTRA_KEY);
                Timber.d("SegmentId= " + segmentId);

                com.keyeswest.fleettracker.models.Location
    previousLocation = null;

                // retrieve the last location belonging to this
    segment from the db
                LocationCursor previousLocationCursor =
    Queries.getLatestLocationBySegmentId(
                    this, segmentId);
            }
        }
    }

```

```

        LatLonBounds bounds =
saveLocationSamples(locations, segmentId);

        // broadcast the location samples for plotting
        broadcastLocationSamples(locations);

        if (previousLocationCursor.getCount() == 1){
            // a previous location exists for the
segment
            previousLocationCursor.moveToFirst();
            previousLocation =
previousLocationCursor.getLocation();

            Date d = new
Date(previousLocation.getTimeStamp());
            Timber.d("Previous timestamp: " +
d.toString() );

            // we are going to need to read the segment
record
            Segment segment =
Queries.getSegmentFromSegmentId(this,segmentId);

            // get the last location sample
            Location lastSample =
locations.get(locations.size()-1);
            // need the distance between lastSample and
previousLocation
            float[] results = new float[1];

Location.distanceBetween(previousLocation.getLatitude(),
                        previousLocation.getLongitude(),
                        lastSample.getLatitude(),
                        lastSample.getLongitude(),
                        results);
            double incrementDistance = results[0];
            Timber.d("Distance measured between: %s %s
%s %s ",
Double.toString(previousLocation.getLatitude()),
Double.toString(previousLocation.getLongitude()),
Double.toString(lastSample.getLatitude()),

```

```

Double.toString(lastSample.getLongitude()));

        Timber.d("Segment increment distance = %s",
            Double.toString(incrementDistance));

        segmentDistance = segment.getDistance() +
incrementDistance;

    }

    //update segment with distance and bounding box
data
    Queries.updateSegment(this, segmentId,
        bounds.getMinLat(), bounds.getMaxLat(),
        bounds.getMinLon(), bounds.getMaxLon(),
        segmentDistance);

    }
}

    }
    private void broadcastLocationSamples(List<Location>
locations){
        Intent intent = new Intent();
        intent.setAction(LOCATION_BROADCAST_PLOT_SAMPLE);

        ArrayList<Location> arrayList = new
ArrayList<>(locations);

        intent.putParcelableArrayListExtra(PLOT_SAMPLES_EXTRA_KEY,
arrayList);

        Timber.d("Broadcasting locations samples for plotting");

        LocalBroadcastManager.getInstance(this).sendBroadcast(intent);

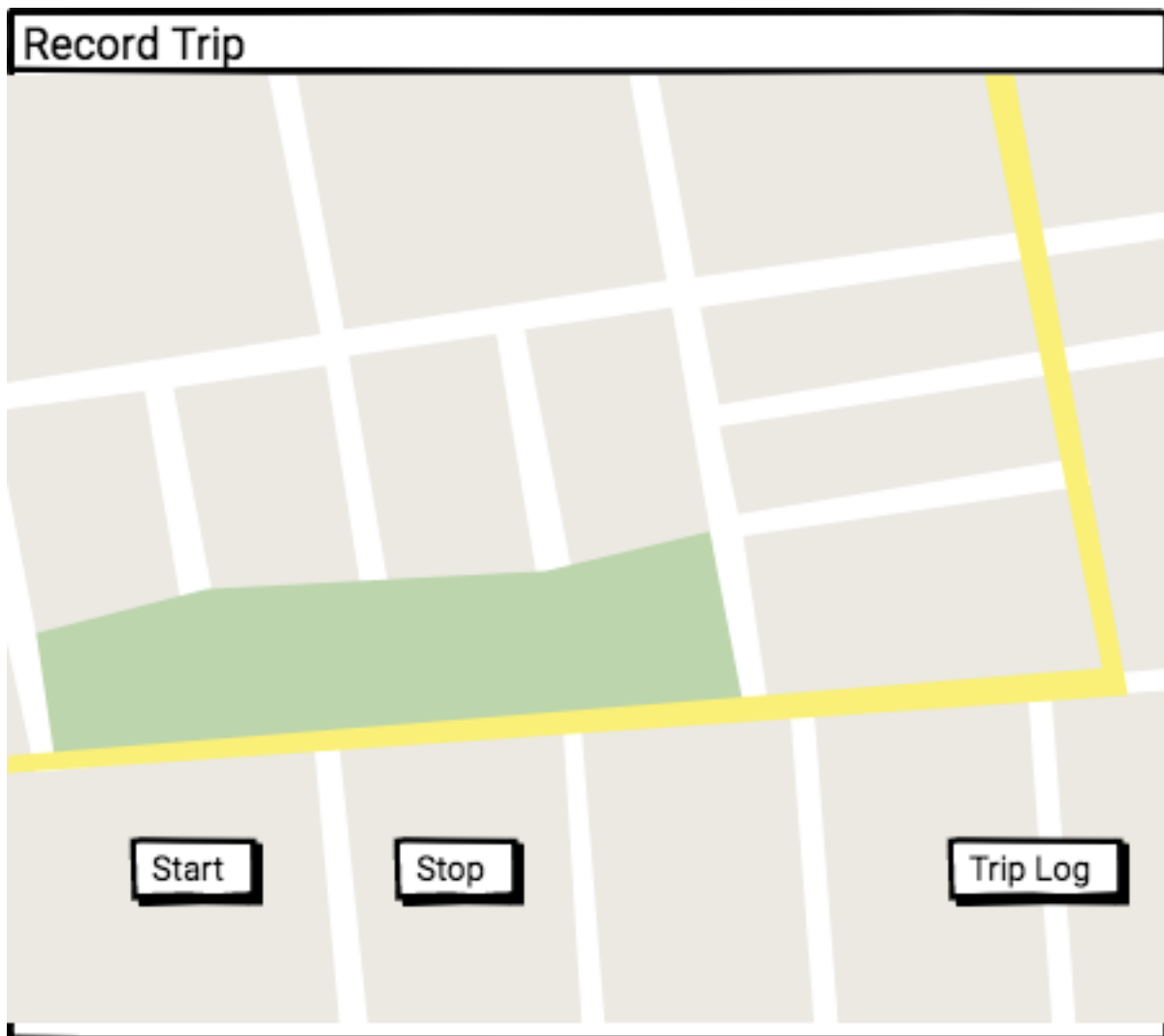
    }
}

```

Capstone Project Proposal

Resubmit Updates

Application Widget Wireframe



The Application Widget for the home screen will present the view that displays the current trip being recorded or enable the user to begin recording a trip if a trip is not underway. The Widget on the home screen enables the user to see their current trip plot without opening the app and have easy access to the Start and Stop buttons for starting or stopping the trip.

The Trip Log button will open the app to the main Trip Log list view.

Share Button

(Reviewer suggested)

Share buttons, perhaps a FAB will be added to the map views and the list view to share the content via email or Facebook.

The Share buttons will be implemented as part of Task 8 in the project plan.

Required Functionality

Per the review:

Must implement at least one of the three

If it regularly pulls or sends data to/from a web service or API, app updates data in its cache at regular intervals using a SyncAdapter or JobDispatcher.

OR

If it needs to pull or send data to/from a web service or API only once, or on a per request basis (such as a search application), app uses an IntentService to do so.

OR

If it performs short duration, on-demand requests(such as search), app uses an AsyncTask.

Kindly add any one of these to qualify the rubric. If you are not able to find a use case where to use them you can find a small use case like downloading some Image to satisfy the requirement and keep the app implementation as it is.

Update

TrackMe receives location updates in the background when the user has initiated tracking a new trip. As described in the Application Architecture section, a **LocationProcessorService (an Intent Service)** handles

processing location samples that have been received by the LocationUpdatesBroadcastReceiver. The prototype code for this intent service is listed below.

Additionally, at this point ***at least one AsyncTask*** has been defined and planned for the application. When a user starts a new trip, a trip segment that will represent the collection of location samples needs to be created. A StartSegmentTask which extends AsyncTask creates a segment id corresponding to the trip segment and writes this segment to SharedPreferences for reference when the first location samples from the FusedLocationProvider arrive. This is required because there appears to be a defect with Google LocationServices which prevents sending extras in the intents that start location sampling, see <https://tinyurl.com/y9gpvcoe>.

Anyway having both an IntentService (one of many services) and an AsyncTask should fulfill the project requirements.

Here is the initial implementation of the IntentService:

```
public class LocationProcessorService extends IntentService {

    private final static String NAME="LocationProcessorService";
    public final static String LOCATIONS_EXTRA_KEY =
"locationsKey";
    public final static String SEGMENT_ID_EXTRA_KEY =
"segmentIdKey";

    public final static String LOCATION_BROADCAST_PLOT_SAMPLE =

"com.keyeswest.trackme.location.samples.intent.action.PLOT_SAMPL
E";

    public final static String PLOT_SAMPLES_EXTRA_KEY =
"sampleData";

    public LocationProcessorService() {
        super(NAME);
    }
}
```

```

    }

    /**
     * A new location sample has arrived.
     *
     * Generally, we want to save the location in the db and
    update the segment to which the
     * location belongs.
     *
     * Updating the segment includes updating the segment
    distance by computing the distance
     * the new location is from the last location sample (unless
    this is the first location of
     * the segment, and then updating the bounding lat/lon box
    associated with the segment.
     * @param intent
     */
    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        Double segmentDistance=0d;

        Timber.d("Entering LocationProcessorService
    onHandleIntent");
        if (intent != null) {
            LocationResult result =
    intent.getParcelableExtra(LOCATIONS_EXTRA_KEY);

            //get the location samples
            List<Location> locations = result.getLocations();

            if ((locations != null) && (locations.size() > 0)) {

                String segmentId =
    intent.getStringExtra(SEGMENT_ID_EXTRA_KEY);
                Timber.d("SegmentId= " + segmentId);

                com.keyeswest.fleettracker.models.Location
    previousLocation = null;

                // retrieve the last location belonging to this
    segment from the db
                LocationCursor previousLocationCursor =
    Queries.getLatestLocationBySegmentId(
                    this, segmentId);
            }
        }
    }

```

```

        LatLonBounds bounds =
saveLocationSamples(locations, segmentId);

        // broadcast the location samples for plotting
        broadcastLocationSamples(locations);

        if (previousLocationCursor.getCount() == 1){
            // a previous location exists for the
segment
            previousLocationCursor.moveToFirst();
            previousLocation =
previousLocationCursor.getLocation();

            Date d = new
Date(previousLocation.getTimeStamp());
            Timber.d("Previous timestamp: " +
d.toString() );

            // we are going to need to read the segment
record
            Segment segment =
Queries.getSegmentFromSegmentId(this,segmentId);

            // get the last location sample
            Location lastSample =
locations.get(locations.size()-1);
            // need the distance between lastSample and
previousLocation
            float[] results = new float[1];

Location.distanceBetween(previousLocation.getLatitude(),
                        previousLocation.getLongitude(),
                        lastSample.getLatitude(),
                        lastSample.getLongitude(),
                        results);
            double incrementDistance = results[0];
            Timber.d("Distance measured between: %s %s
%s %s ",
Double.toString(previousLocation.getLatitude()),
Double.toString(previousLocation.getLongitude()),
Double.toString(lastSample.getLatitude()),

```

```

Double.toString(lastSample.getLongitude()));

        Timber.d("Segment increment distance = %s",
            Double.toString(incrementDistance));

        segmentDistance = segment.getDistance() +
incrementDistance;

    }

    //update segment with distance and bounding box
data
    Queries.updateSegment(this, segmentId,
        bounds.getMinLat(), bounds.getMaxLat(),
        bounds.getMinLon(), bounds.getMaxLon(),
        segmentDistance);

    }
}

    private void broadcastLocationSamples(List<Location>
locations){
        Intent intent = new Intent();
        intent.setAction(LOCATION_BROADCAST_PLOT_SAMPLE);

        ArrayList<Location> arrayList = new
ArrayList<>(locations);

        intent.putParcelableArrayListExtra(PLOT_SAMPLES_EXTRA_KEY,
arrayList);

        Timber.d("Broadcasting locations samples for plotting");

        LocalBroadcastManager.getInstance(this).sendBroadcast(intent);

    }
}

```

Intelligent location tracking algorithms are used which minimize power consumption by the app by combining GPS and cellular network information to obtain location data.

Intended User

Anyone that needs to account for trip mileage and travel expenses will find this app useful. On business travel, rental car travel can be recorded to correlate fuel expense with miles traveled. A small business owner can record travel and create a trip log to support travel deductions and trip expenses.

Users who routinely follow the same routes can compare similar trip data over different days to visually uncover disparities in trip time or trip mileage.

Features

Committed

- Start and Stop trip recording with a single button click.
- Animate track data as it is displayed to the user.
- View a trip in progress as it is recorded on a map.
- Create a trip log for recorded trips.
- Trip data is captured in the background and is saved in a local database.
- TrackMe does not interfere with other running Android applications.
- Search and sort trip log entries by date and/or trip distance.
- Mark trips as Favorites for easy retrieval.

- Compare trip tracks on different days.
- Delete trips from the trip log.
- Provides an application widget which shows the track of a trip currently being recorded.

Potential Features

(Not currently part of minimum viable product)

1) Manually starting and stopping trip recording can be eliminated by using a Bluetooth proximity beacon like:

<https://accent-systems.com/product/ibks-105/>

The beacon can be placed in a vehicle and when detected by the app will automatically start recording a trip. When the user exits the vehicle the app will stop trip recording when the proximity beacon is no longer detected.

This requires purchasing or obtaining the proximity beacons and providing application settings for configuring the beacon.

2) Exporting and sharing log data. Export a CSV formatted file of selected trip log data. Sharing map images of trip data

User Interface Mocks

Mock views are available for review at these publicly accessible links:

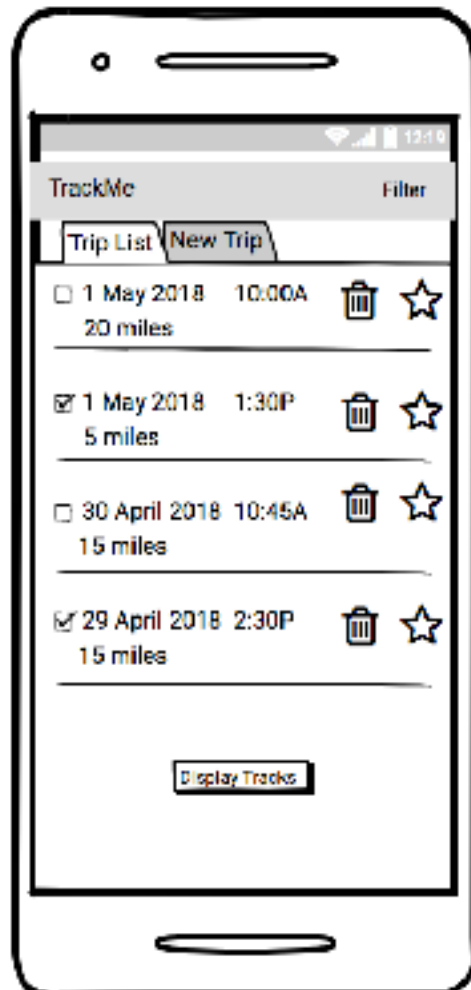
Phone: <https://balsamiq.cloud/sd6q9of/pmdn4qw>

Tablet: <https://balsamiq.cloud/sd6q9of/ps2kbpz>

Please visit these links where notes are provided in the information area of each screen and navigation is supported with simulated hyper links.

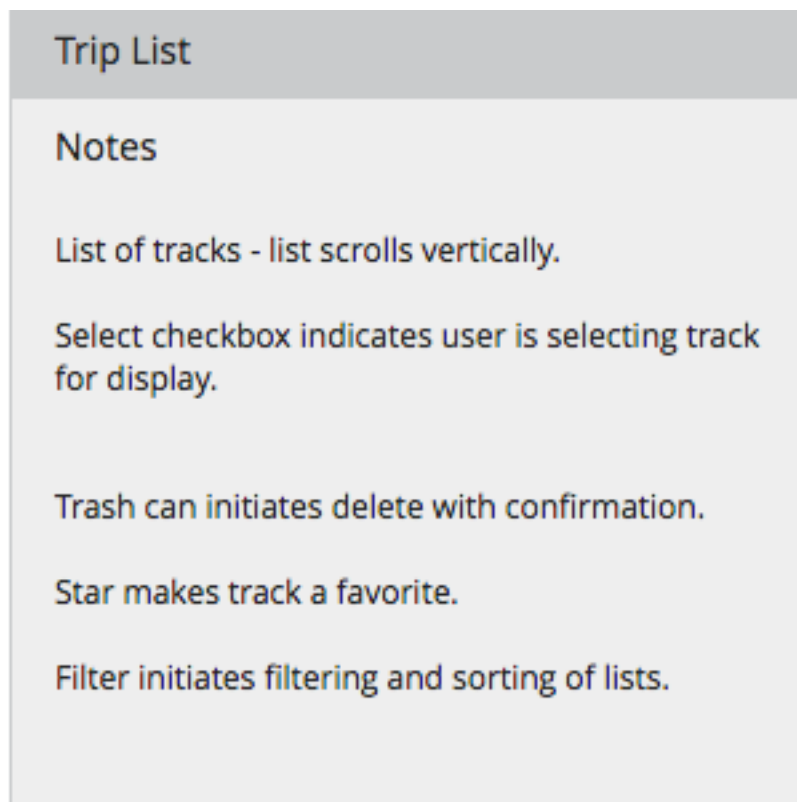
In case it is required all of the wireframes have been attached at the end of this document.

Example Screen from the phone layout:



In the Balsamic view there are red markers on each button or icon that are clickable and function as navigational tools for the mocked screens.

The notes section for this screen appears in Balsamiq as:



Key Considerations

How will your app handle data persistence?

Consideration was given to using a NoSQL Firebase Realtime database in the cloud but the use cases don't warrant the off device database. First, since the database is only required to store data for the single device, there is no requirement to scale the database in the future. The user can manage the amount of required persistent storage on the device by deleting unwanted track files from the local database.

If a multi-tenant cloud based SaaS solution were to be developed to track a fleet of vehicles for multiple companies or groups an off device database would be appropriate and to achieve flexible scaling a NoSQL database should be considered.

TrackMe uses a simple two entity relational database. Tracks or segments are stored entities which contain descriptive attributes about a trip including starting dates and times, distances, and bounding latitude and longitude coordinates that encompass the collection of location fixes that make up the segment.

A second locations entity contains location fixes (latitude, longitude, speed, etc.) with a foreign key referencing the trip to which the location belongs.

A content provider is implemented which allows the trip data to be accessed by another application if the user grants access to the data. The content provider might be useful to an expense reporting application to query the trip log data.

Describe any edge or corner cases in the UX.

Providing intuitive filtering and sorting capabilities of the trip data is challenging. I've reviewed several articles:

<https://www.raywenderlich.com/155041/6-best-practices-for-mobile-app-search-filtering>

<https://medium.com/@thierrymeier/filtering-and-sorting-best-practices-on-mobile-61626449cec>

<https://www.smashingmagazine.com/2012/04/ui-patterns-for-mobile-apps-search-sort-filter/>

and looked at apps that have received positive reviews for implementations of filtering and sorting functionality including Amazon, Etsy, Target, and Walmart. The wireframes capture some of these learnings.

I also would like to animate track data overlaid on a map rather than simply displaying the trip data in a single snapshot. I do not have the technique for implementing the animation of the track data figured out yet but have seen examples.

Another interesting challenge is dealing with an asynchronous race condition occurring on a UI thread. When displaying a trip segment on a map a callback is used to indicate that a map is ready for display. At the same time, a loader is initialized which pulls the trip data from the database and a callback is used to indicate that the data is ready. Since either the map callback or the loader ready callback can complete in any sequence, the activity or fragment must be able to proceed with drawing the map and trip data when the last of these two callbacks complete without blocking the UI thread. A solution has been implemented in the proof of concept app.

Finally, it would be nice to have a thumbnail image of the track file to display in the trackless that could be used in a transition animation to the map view displays.

Describe any libraries you'll be using and share your reasoning for including them.

Custom View Range Bar Widget (<https://tinyurl.com/yb8aaewe>)

- enables the user to use a widget to specify minimum and maximum trip distances when filtering trip log list data.

Timber - well known application logging library which facilitates logging during development and error logging in Release versions.

ButterKnife - well known library for matching layout resources to code definitions.

Stetho - development tool (no release) to aid in database debugging.

Gson - JSON library used in development only (no release) for assisting in mocking location data.

Android libraries:

General compatibility support for fragments, activities, themes, etc:

`com.android.support:appcompat-v7`

`com.android.support:design`

`com.android.support.constraint:constraint-layout`

Google maps:

`com.google.android.gms:play-services-maps` - track files are overlayed on maps.

Google location services:

`com.google.android.gms:play-services-locations` - using

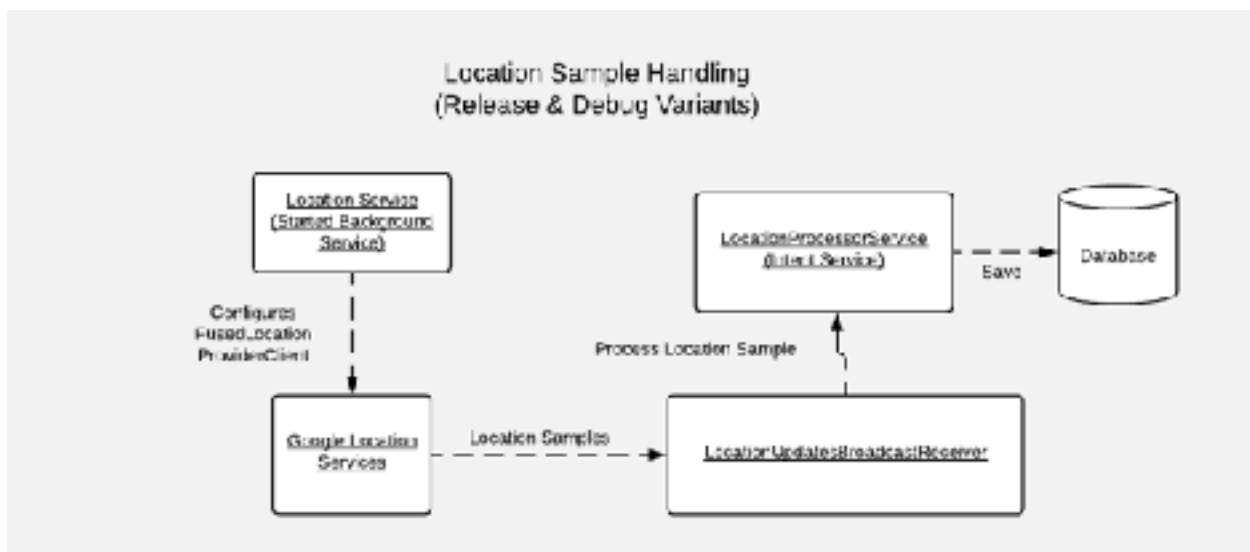
`FusedLocationProviderClient` to provide location samples (fixes). Location samples will be provided in the background.

Application Architecture

A principal requirement of the TrackMe app is to receive and process location updates in the background. A proof of concept application:

<https://github.com/chipk215/TrackMe>

has been used to arrive at the background processing services depicted below. From a TrackerActivity, the background LocationService is started which receives messages to either start the generation of location updates or stop the updating of location updates. The LocationService, in the Debug and Release variants configures the Google Location Services FusedLocationProvider to generate location updates for the device. A broadcast receiver “LocationUpdatesBroadcastReceiver” receives the location updates and sends them to the LocationProcessor intent service where the location samples are processed and saved to database.



During development, it was discovered that being able to provide mocked location updates was very helpful to verify the workflow processing of the sampled data. Reading samples from a file enables

using both captured data and manufactured data to use in plotting trips and viewing past trips.

A new “mock” build variant was created where the implementation for the Location Service is replaced with a service that reads a location data file and broadcast the sample locations to the LocationUpdatesBroadcast receiver. This functionality augments testing and of course is not present in the release variant of the app.

See the provided GitHub link for the proof of concept app for for more detail.

Task 0: Proof of Concept

Prior to submitting the project proposal a proof of concept application

<https://github.com/chipk215/TrackMe>

was developed which explored Google Map Services, Location Services, methodologies for mocking location services, and the data model and database needed to support the application. This code will be highly leveraged in the project.

Task 1: Project Setup

Gradle is used to configure and manage the Android Studio project.

The project gradle file uses build script dependencies:

`com.android.tools.build.gradle:3.1.2`

`com.google.gms:google-services:3.2.0`

The module gradle file uses the most recent release versions for the:

`androidSupportVersion (27.1.1)`

`constraintLayoutVersion (1.1.0)`

`Timber version (4.7.0)`

`Butterknife version (8.8.1)`

As noted in the previous section an additional build variant called “mock” is used which is configured by:

```

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
        'proguard-rules.pro'
    }
    debug {
        debuggable true
    }
    mock {
        initWith debug
    }
}

```

The application uses a TrackMe class which extends from the Application class to configure Timber for debug and release variants.

The final configuration requirements are handled in the manifest files for the different build variants where permissions, activities, providers, services and broadcast receivers are declared.

Task 2: Initial app installation processing (MainActivity)

Location services, specifically accessing fine location values that the user will be prompted to grant user permission for accessing location data the first time the app runs after installation. Explanatory text will be provided if the user fails to grant permission communicating that the permission is required for the app to function.

Task 3: Database, Content Provider, and Loader Implementations and Tests

Import the database, data models, cursor wrappers, content provider, data loaders and supporting tests that were developed during proof of concept. The database and data models will facilitate development of the UI flow.

Task 4: Tabbed Layout UI and Tests

As depicted in the wireframes a tabbed layout is provided to switch between the list view of trip log entries and the view for recording a new trip.

UI Subtasks include:

- Implementation of recycler list view for log entries
- Selection box implementation for selecting log entries to view on map
- Implementation of trash can icon for delete operation
 - Use cascaded deletes when deleting trip entries to remove corresponding location data entries
- Implementation of the favorite icon
- Implementation of the Display Tracks button which initiates navigation to the map view of the selected log entries
- Implementation of the plotted track view corresponding to a trip entry. Use animation when drawing the track.
- Espresso testing of UI views
- Fragments for the list and map views.
- List detail views for the tablet.

Task 5: New Trip View UI

The new trip map view is where the user initiates the recording of a new trip. The initial map view without a track will correspond to the user's current location.

Start and Stop tracking buttons will be implemented.

Updates to the map will be implemented by creating a polyline overlay on the map.

Task 6 : Filter and Sort Functionality

Implement UI for filtering and sorting corresponding to wireframes

<https://balsamiq.cloud/sd6q9of/pmdn4qw>

Top Level Filter Screen

Sort By

Fiter Options

Date Filter

Filter Selections

Sort Selection

Distance Rage Filter

Task 7 : Application Widget

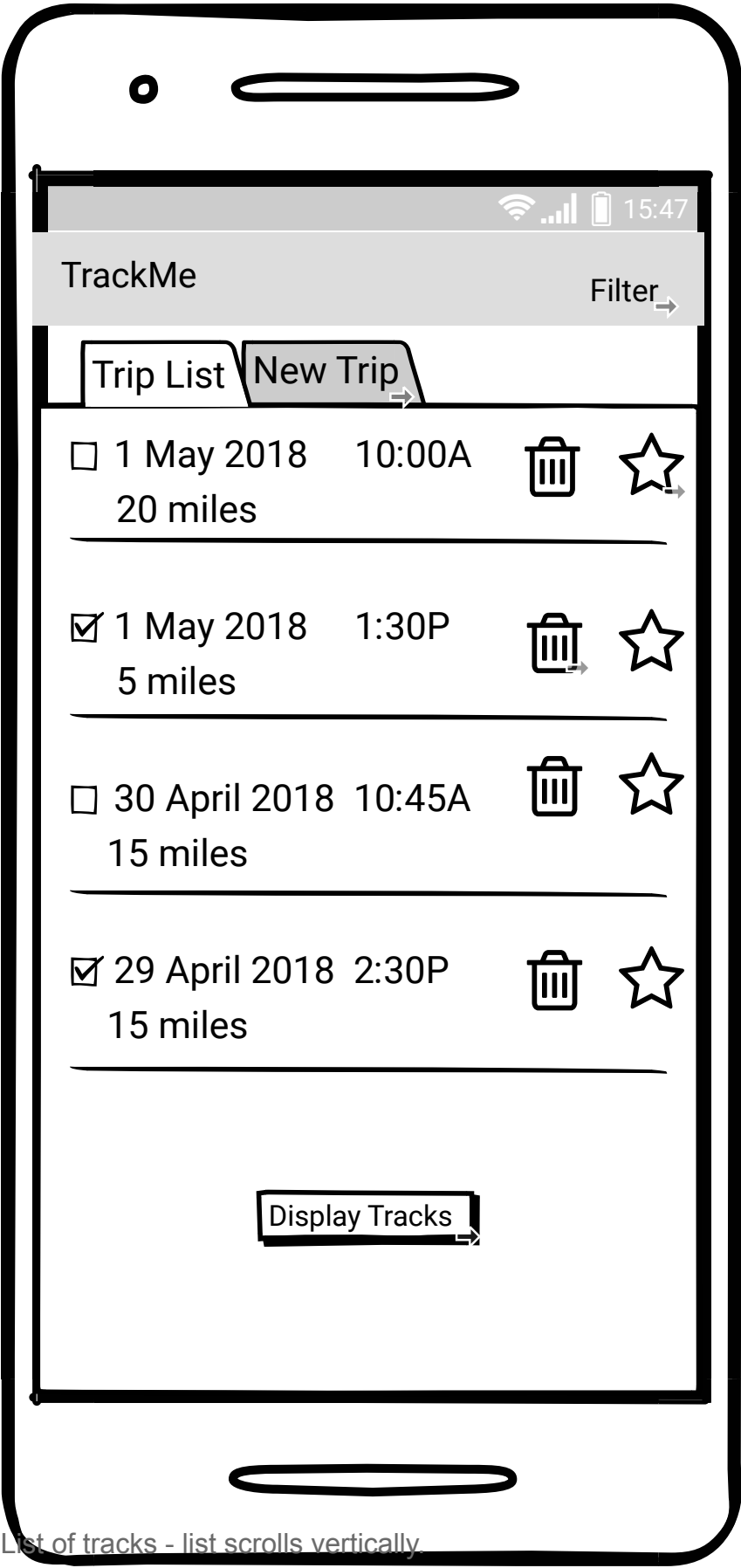
The application widget will be a desktop map view of either the current trip being recorded with corresponding track updates or the last trip saved to the database.

Task 8 : Material Design and App Theme

Meet Material Design guidelines, pick color palette, investigate animating transitions.

Task 9: Field Tests

Travel with the app and verify trip log data is logged and displayed correctly.



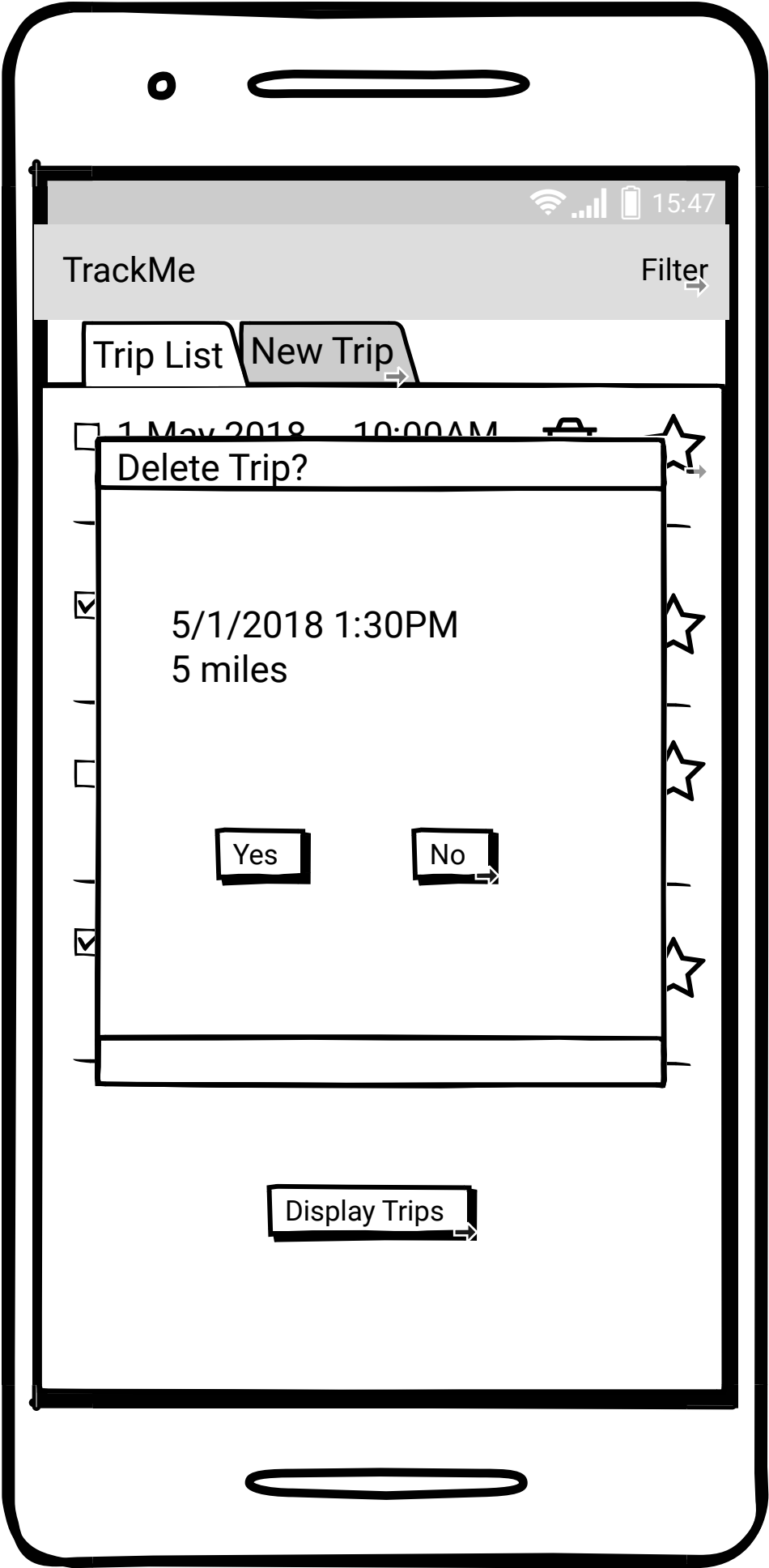
List of tracks - list scrolls vertically.

Select checkbox indicates user is selecting track for display.

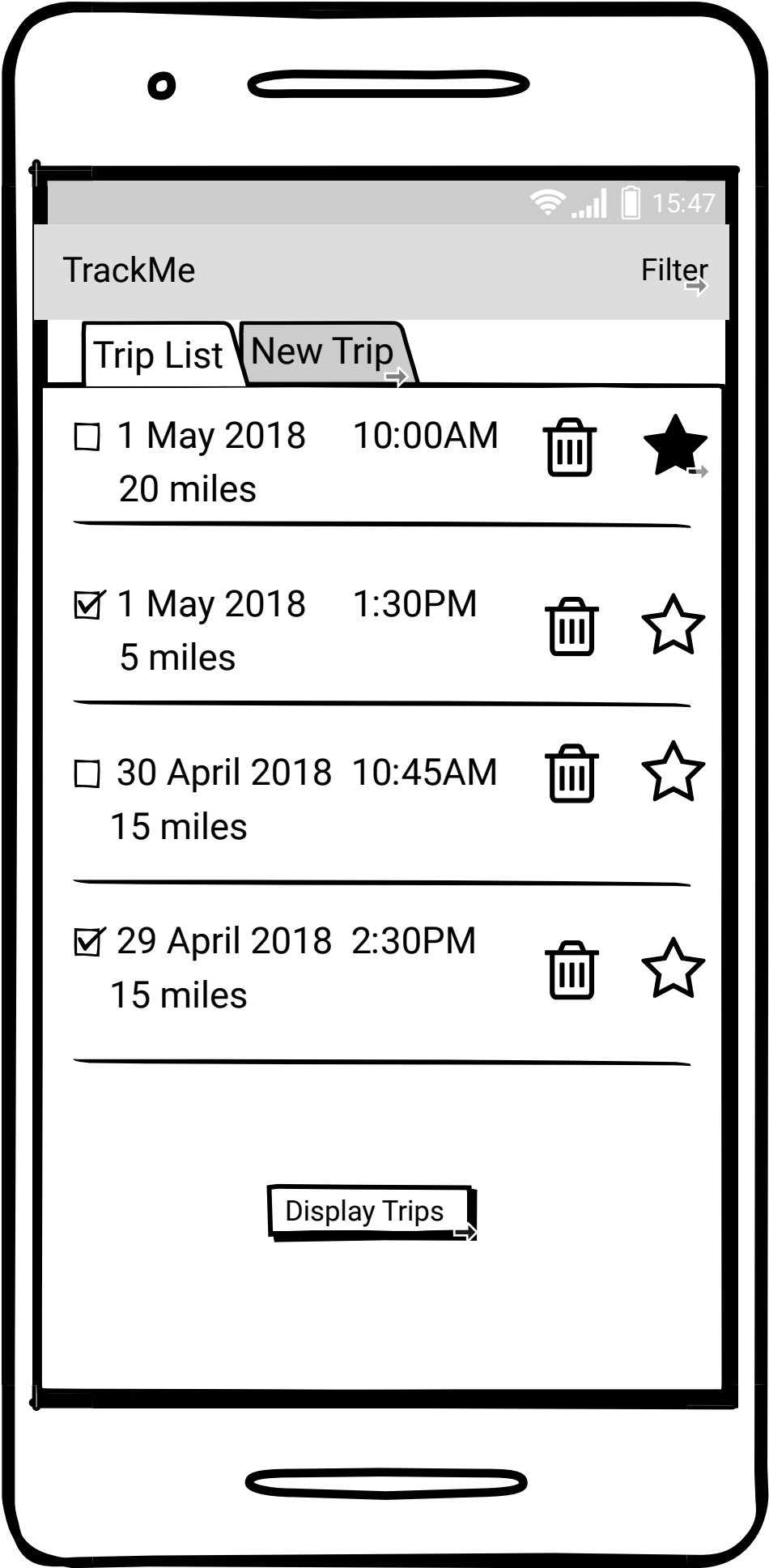
Trash can initiates delete with confirmation.

Star makes track a favorite.

Filter initiates filtering and sorting of lists.



Delete is destructive so requires a confirmation on the operation.

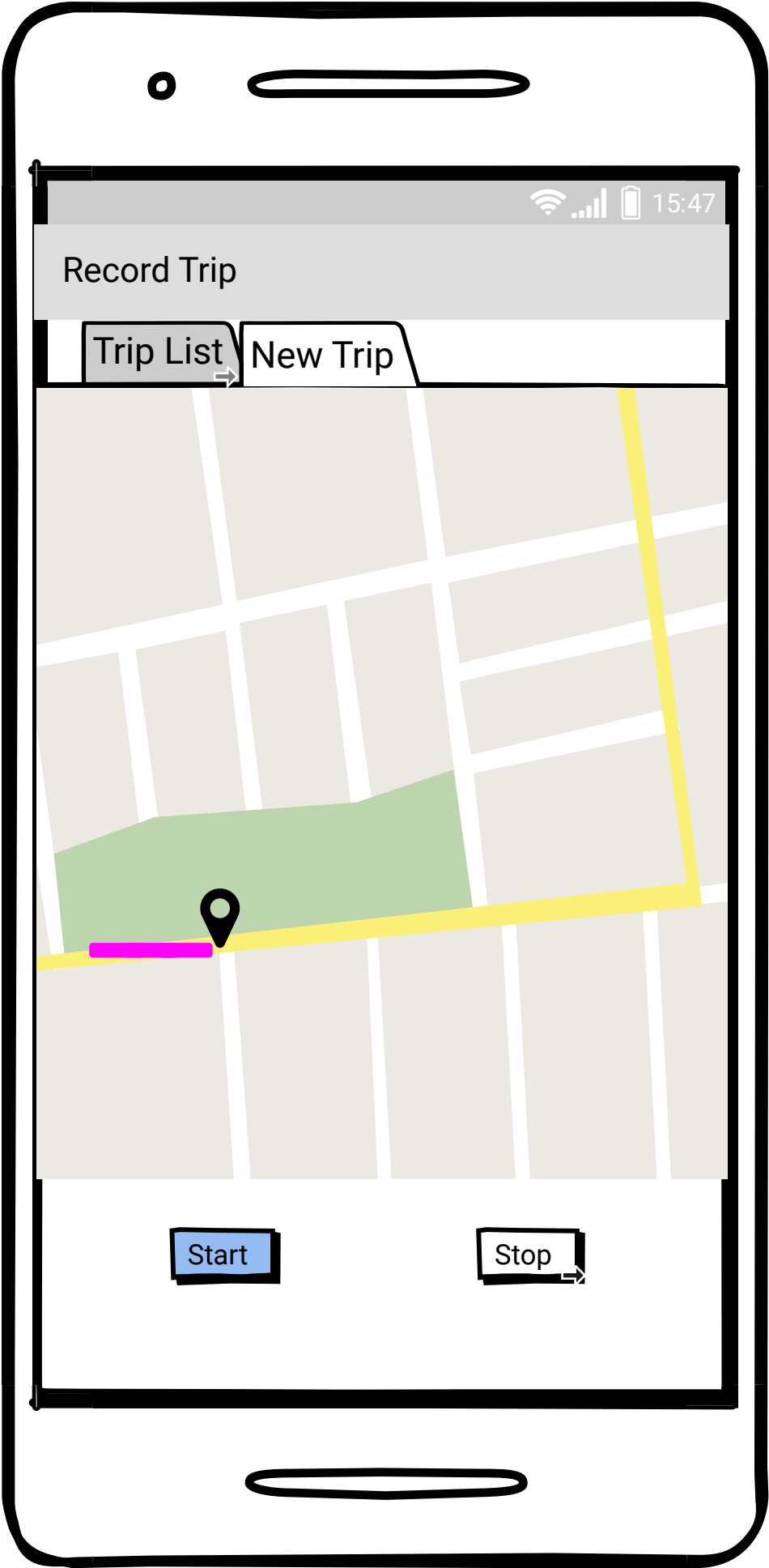


The Star button toggles on and off indicating that the trip is marked as a Favorite. Trip log entries can be filtered by the favorite attribute.

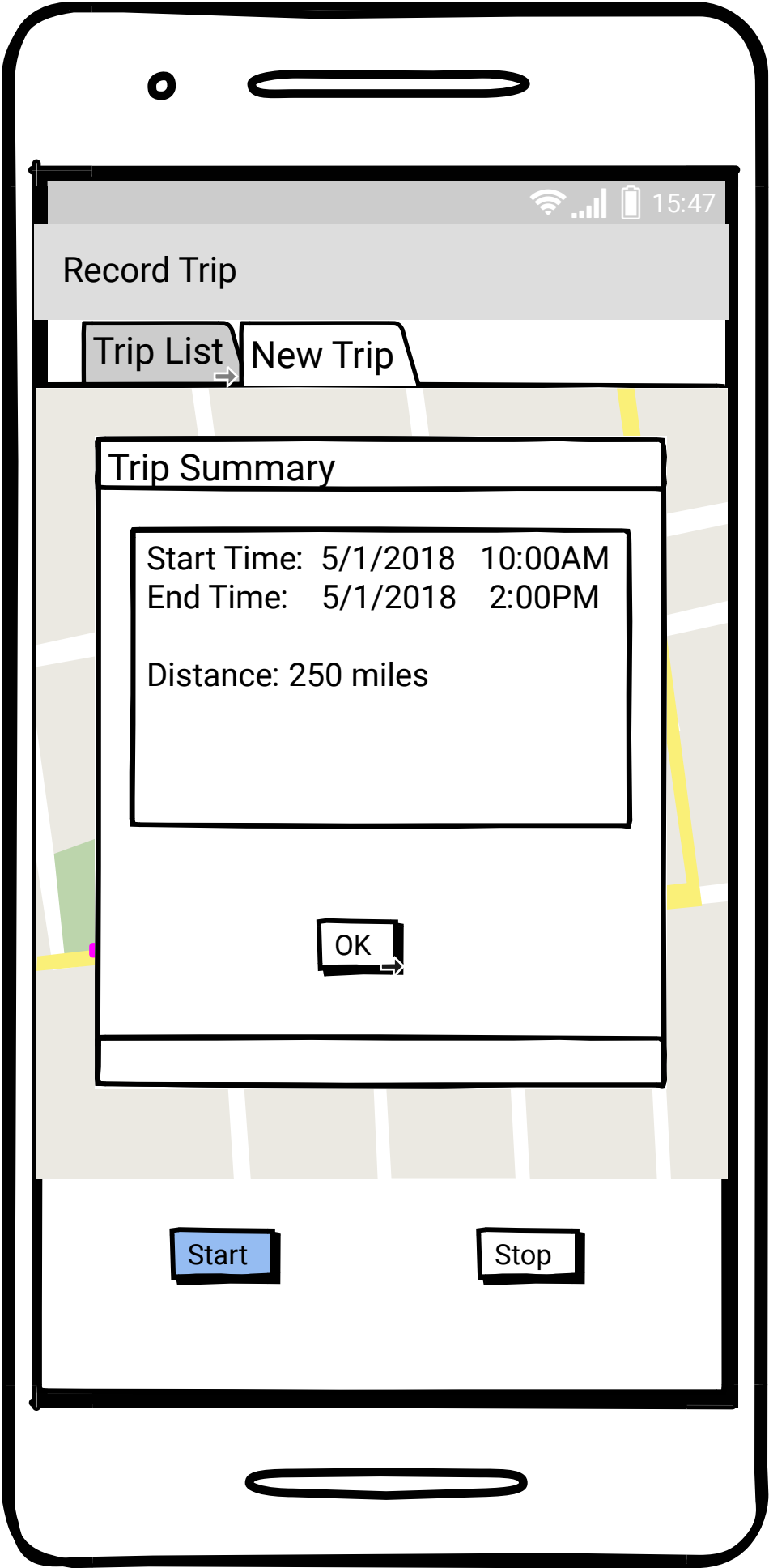


Initial screen view when New Trip tab button is tapped.

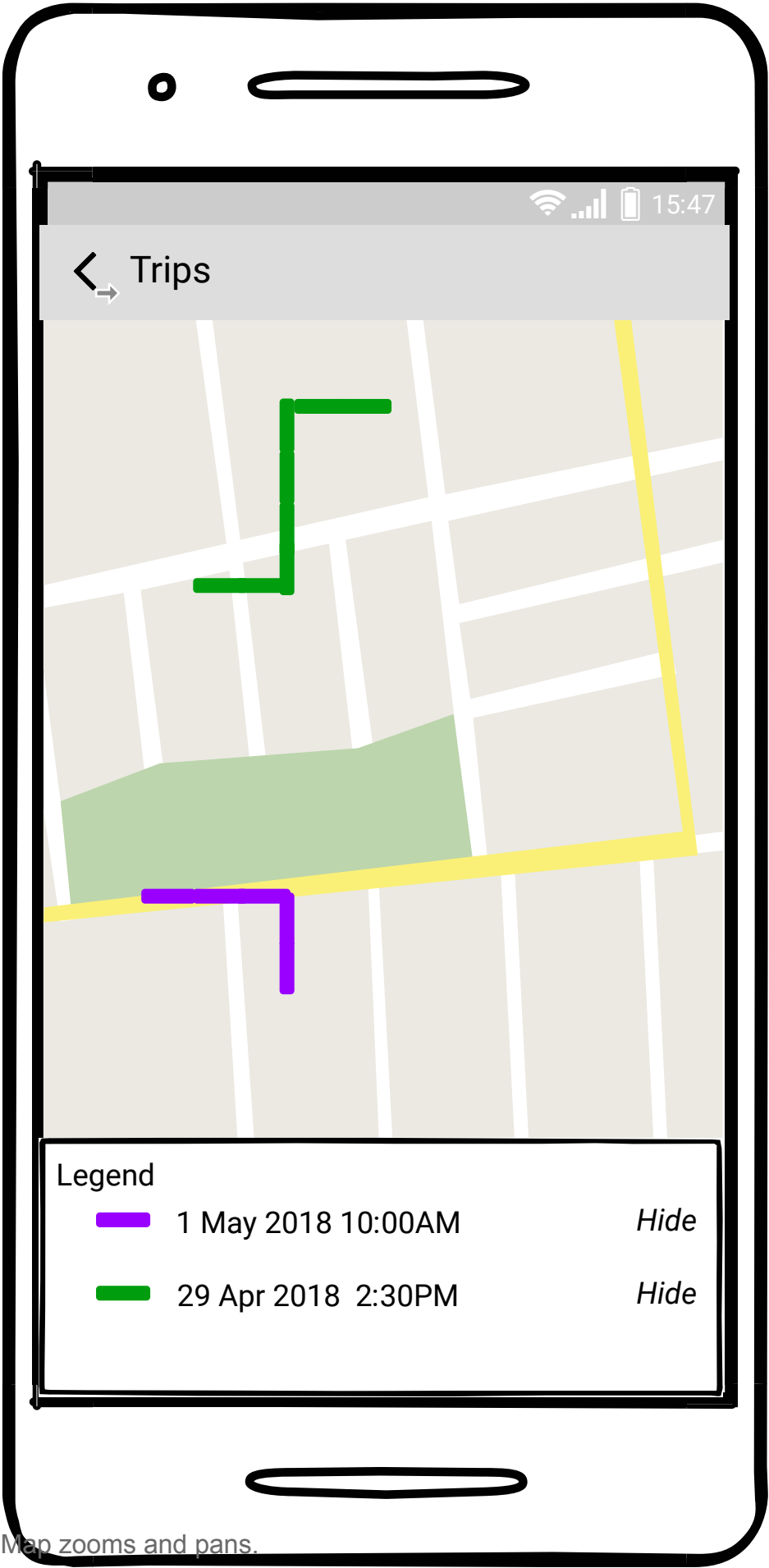
User must tap on the Start button to begin recoding the trip.



A blinking Place marker corresponds to last location fix and updates with each new location fix.



After clicking on Stop when recording a trip, the summary details of the trip are displayed.

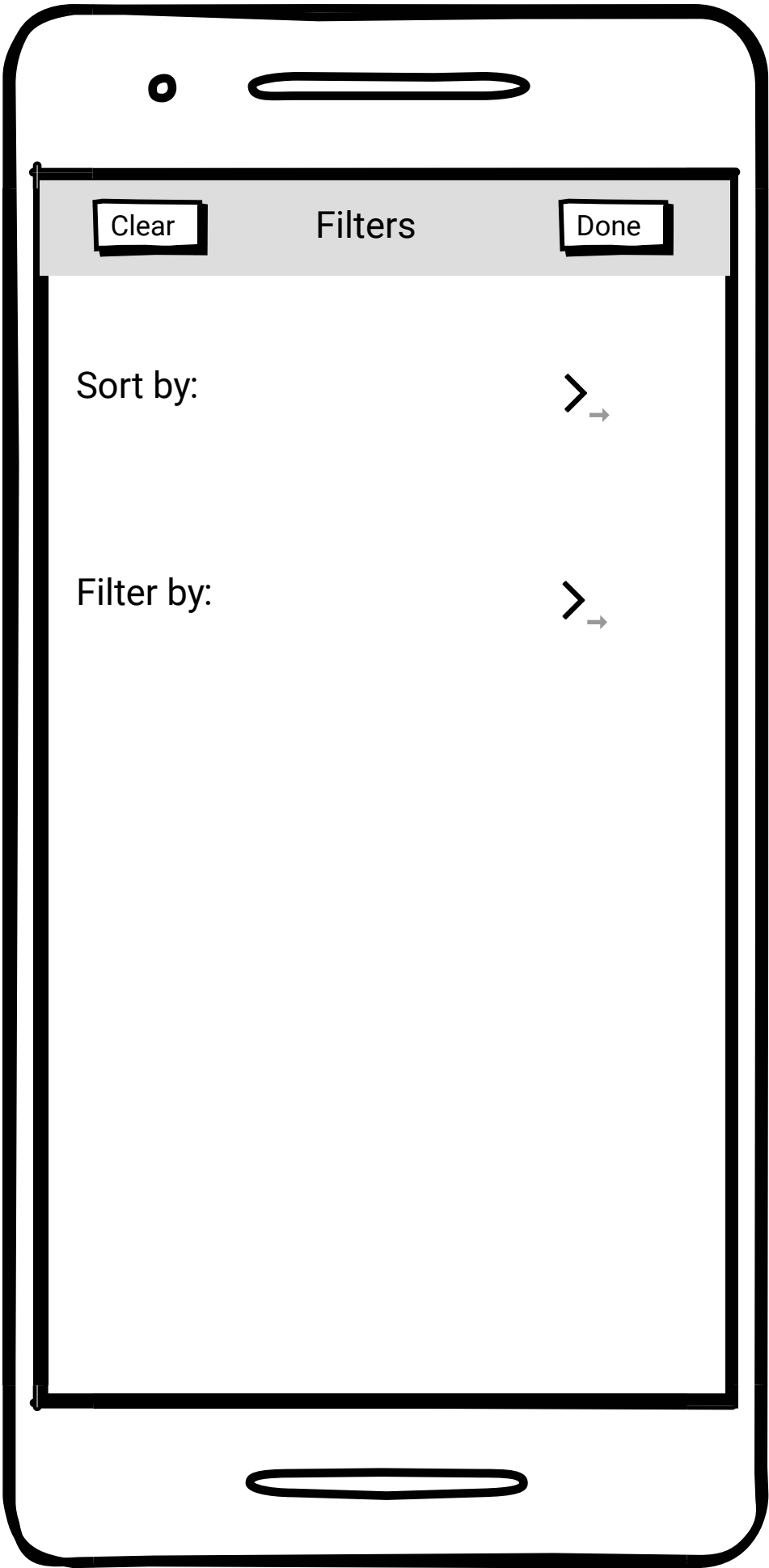


Map zooms and pans.

Tracks can be toggled to hide and show.

Tapping a visible track will display details including start time, end time, and distance in a dismissable Toast.

Tracks should draw with animation.



Top level view displayed when the Filter item in the menu bar is tapped.

Sub menus are presented for sorting and filtering when the corresponding right facing chevron is tapped.

The image shows a mobile application interface for sorting tracks. At the top, there is a status bar with a grey background containing icons for signal strength, battery level, and the time 15:47. Below the status bar, the text 'Sort by:' is displayed. The interface contains two main sections, each enclosed in a black rectangular border. The first section is titled 'Date:' and contains three radio button options: 'Most Recent' (which is selected), 'Oldest', and 'No Date Sort'. The second section is titled 'Distance:' and contains three radio button options: 'Shortest', 'Longest' (which is selected), and 'No Distance Sort'. At the bottom of the interface, there is a 'Continue' button with a right-pointing arrow. The entire app interface is framed by a thick black border representing the phone's screen.

Two sets of radio buttons for sorting (ordering) by data and/or distance.

Sort by scenarios:

1. Date only

User chooses either most recent or Oldest and track listings are sorted accordingly.

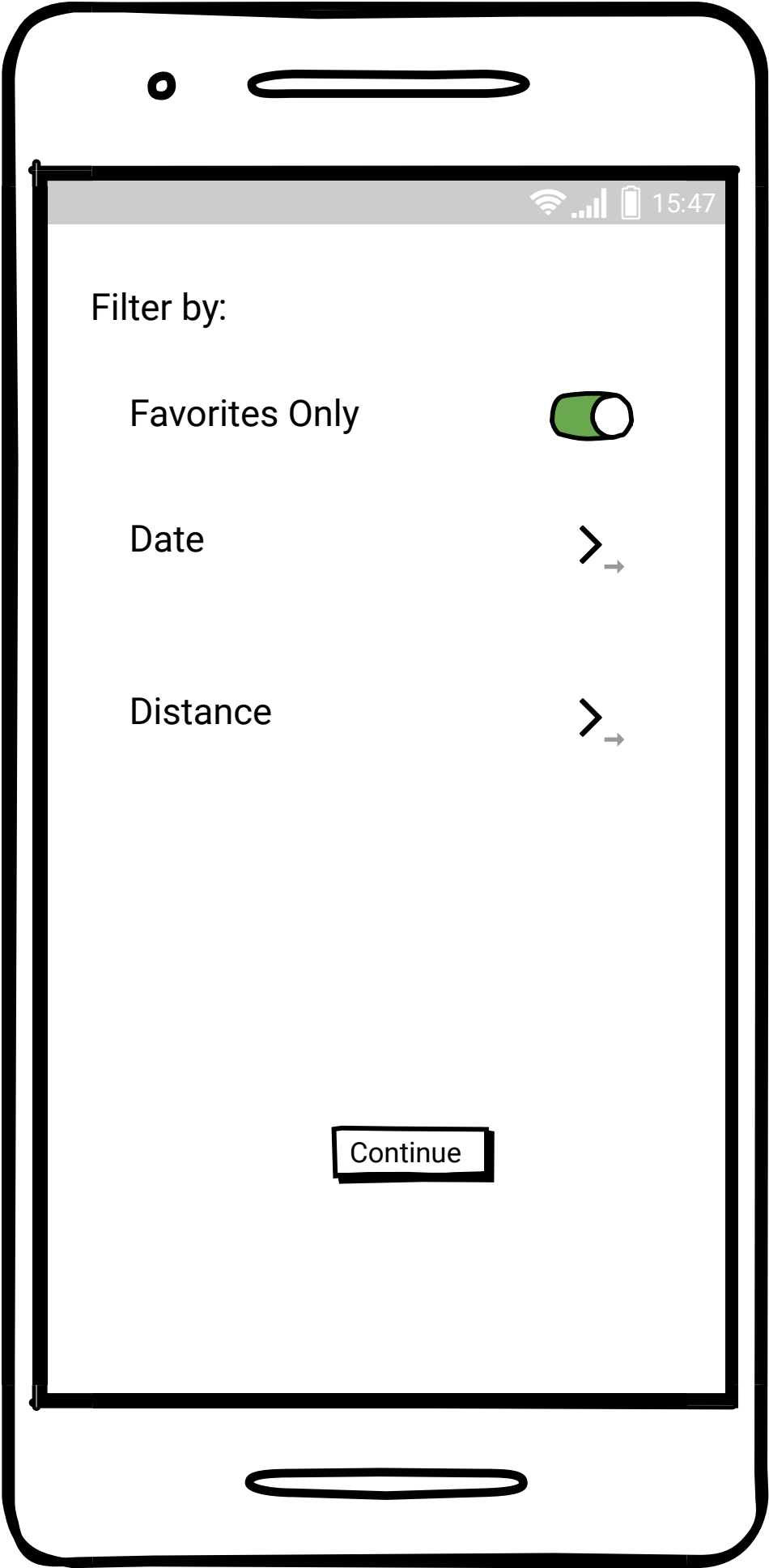
2. Distance only

User chooses shortest or longest and track listings are sorted by track length irrespective of date.

3. Date and Distance

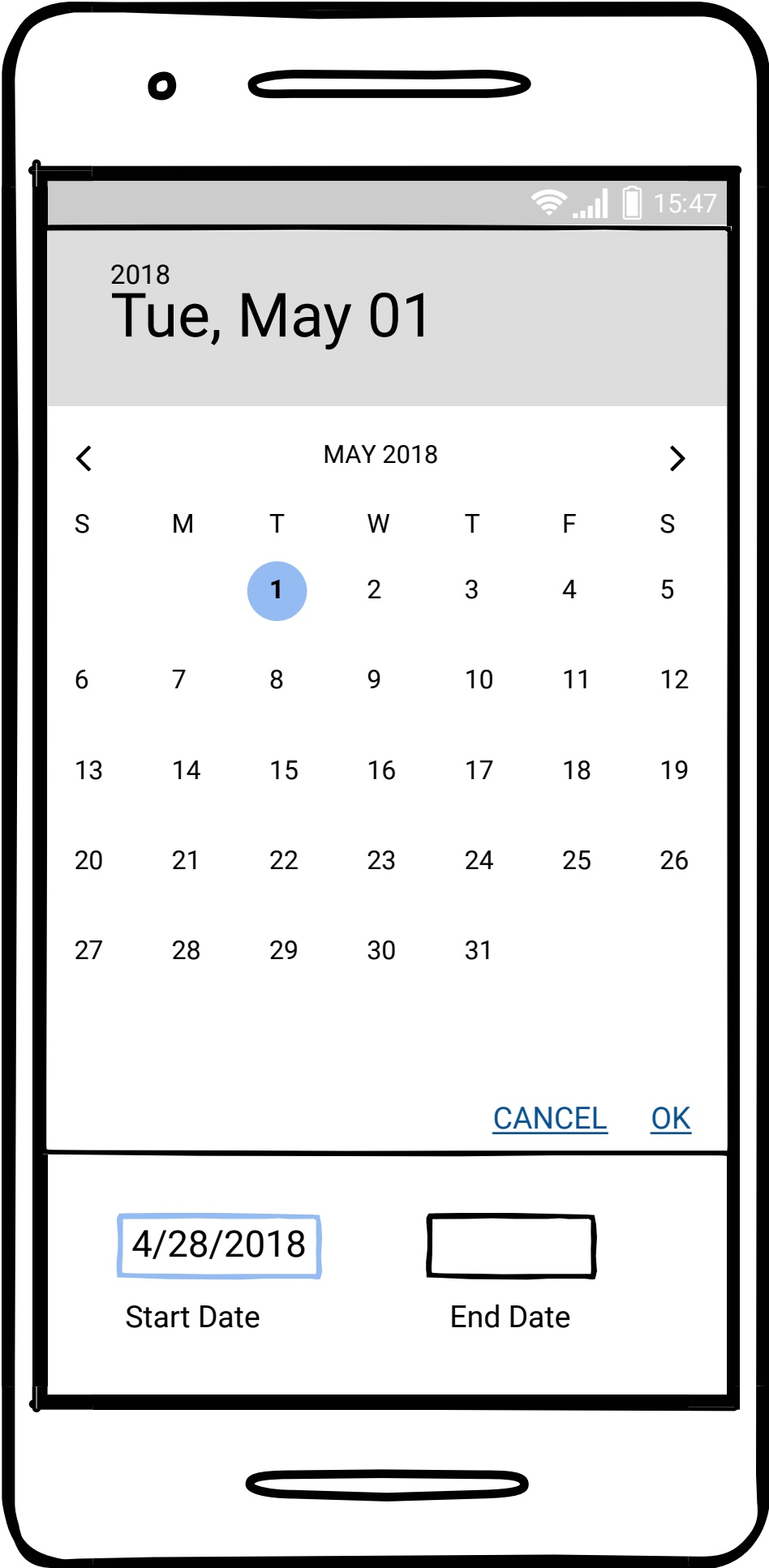
Track listings are sorted first by Date and then within dates by distance. Start time and end times are not considered when sorting by date.

E.g. list tracks by date with the longest tracks appearing before shorter tracks for each day.



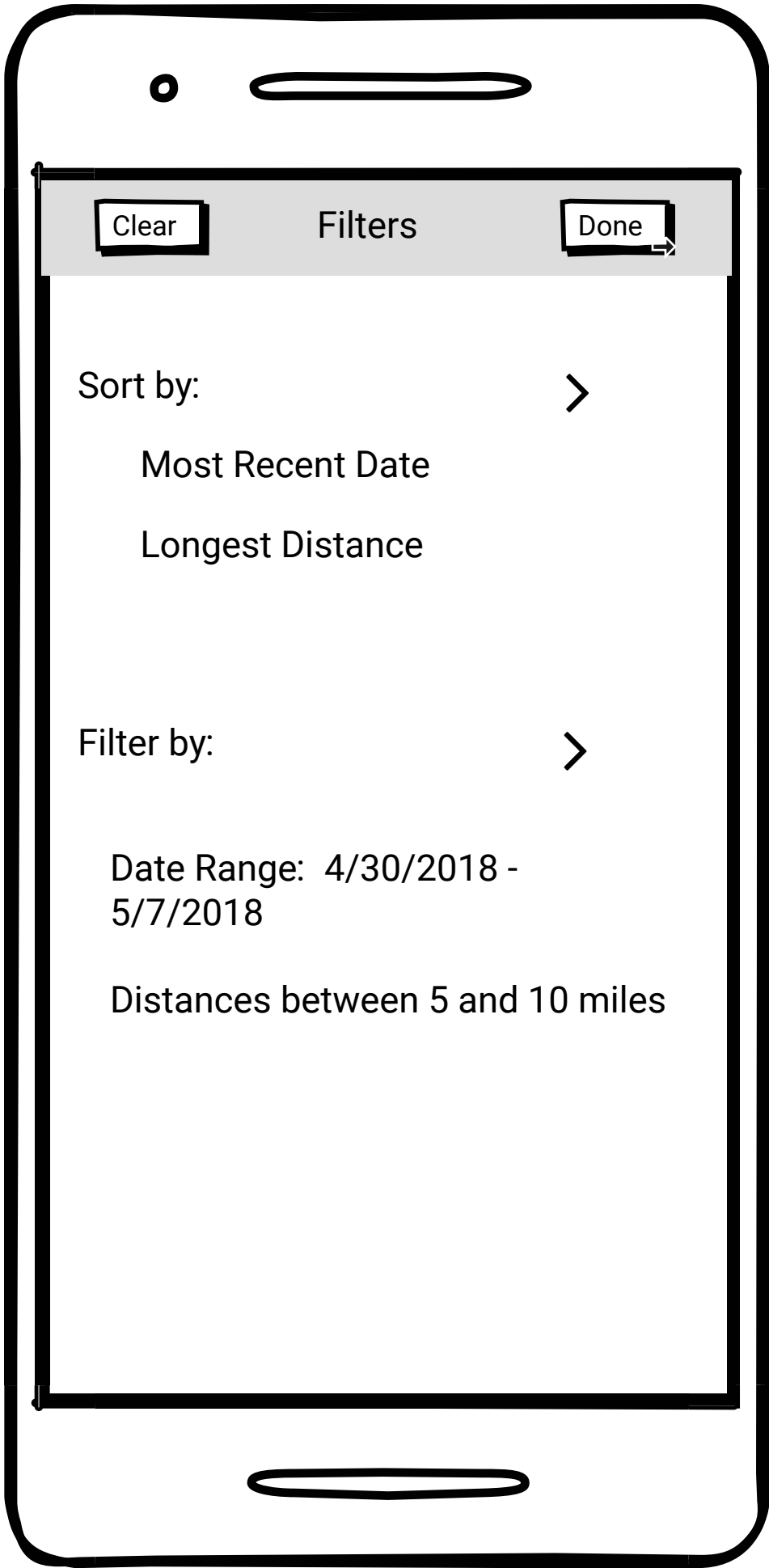
Filter sub-menu.

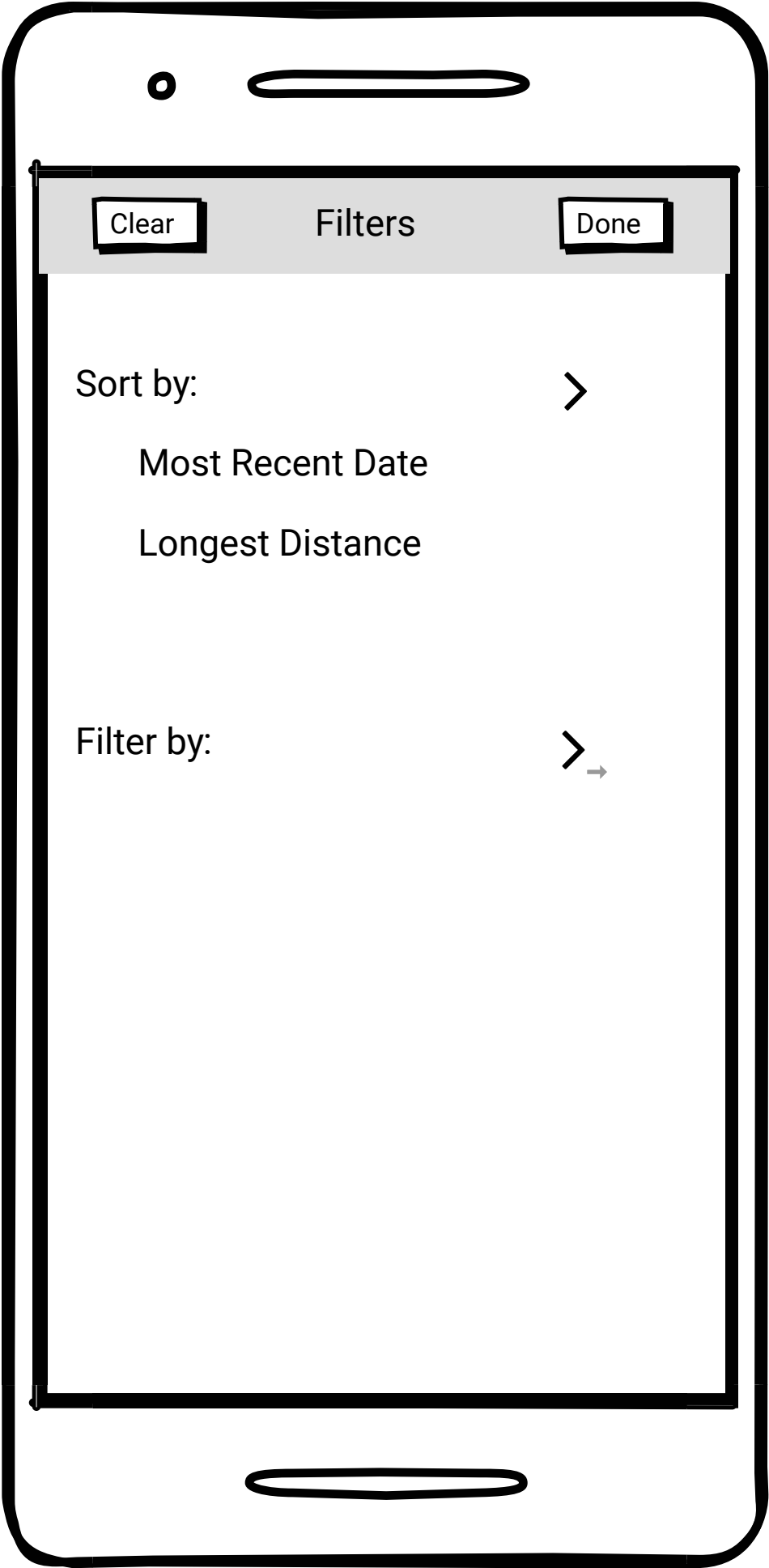
Date filters and Distance filter menus are accessed via the right facing chevron.

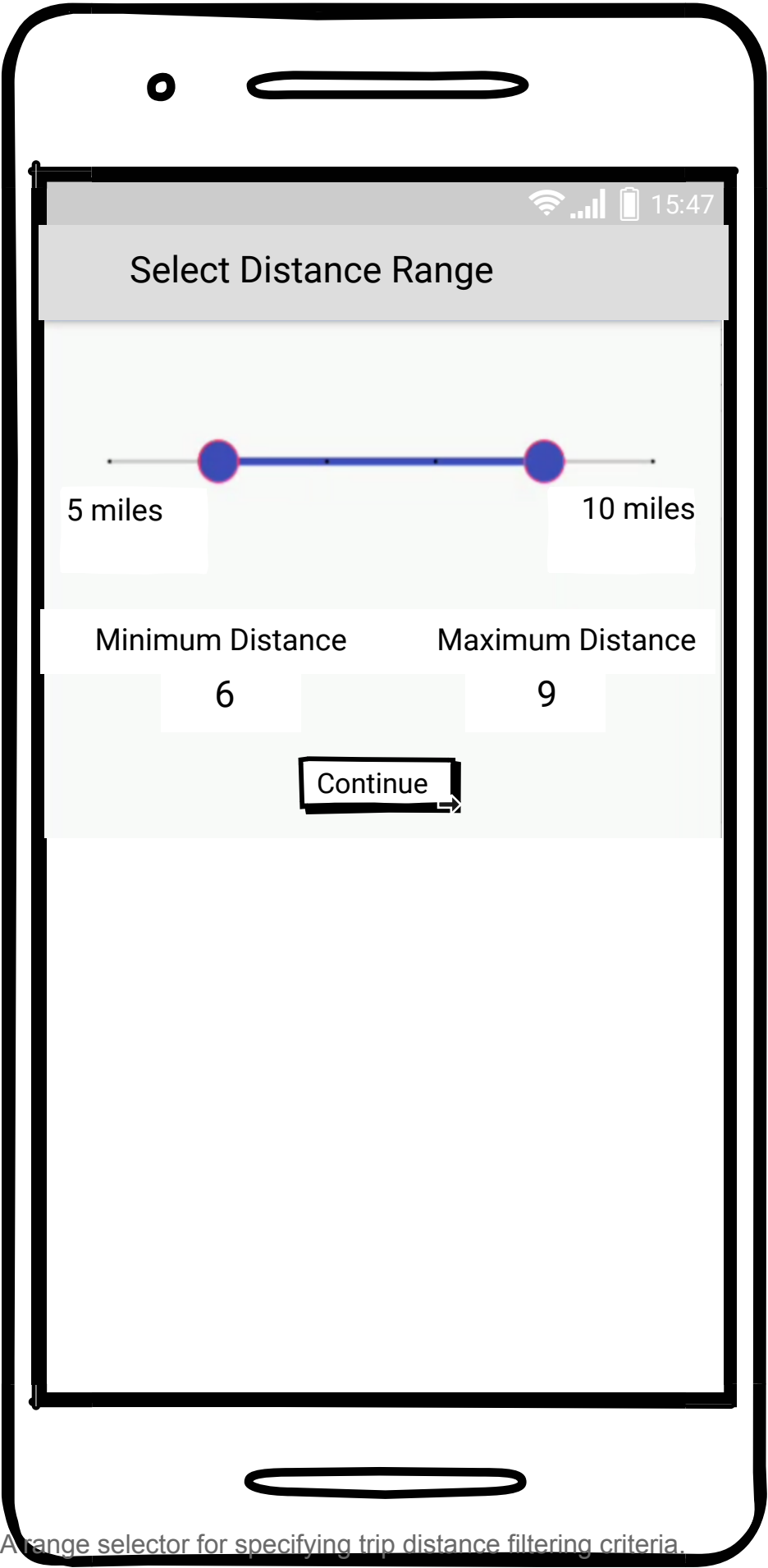


Will find/use a Material Design compatible date picker library.

User can specify single date by setting start and end dated to same value.







A range selector for specifying trip distance filtering criteria.

I've used a custom view from









<https://tinyurl.com/yb8aaewe>

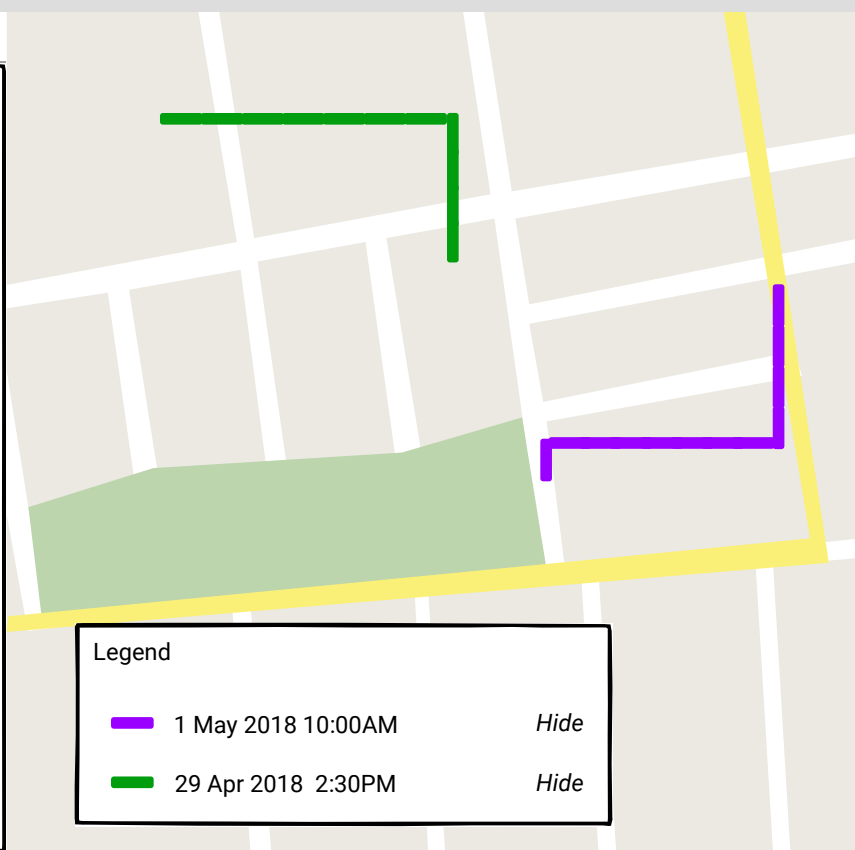
as the basis for the range widget.

TrackMe

Trip List

New Trip

- ☐ 1 May 2018 10:00AM
20 miles  
- ☒ 1 May 2018 1:30PM
5 miles  
- ☐ 30 April 2018 10:45AM
15 miles  
- ☒ 29 April 2018 2:30PM
15 miles  



TrackMe

Trip List

New Trip

Start

Stop

Start time: 10:00AM

Trip time: 35 minutes

Trip distance: 15 miles

