GitHub Username: sdunstan

Two Minute Plays

Description

Two Minute Plays is a fun way to create a short skit with your friends. Unleash your inner theater nerd! Pick from one of our clever and funny skits, record a part, and share with your friends. Once finished, Two Minute Plays will combine the parts into a single awesome movie!

Intended User

Two Minute Plays is for everyone! It is an entertainment app that everyone can enjoy.

Features

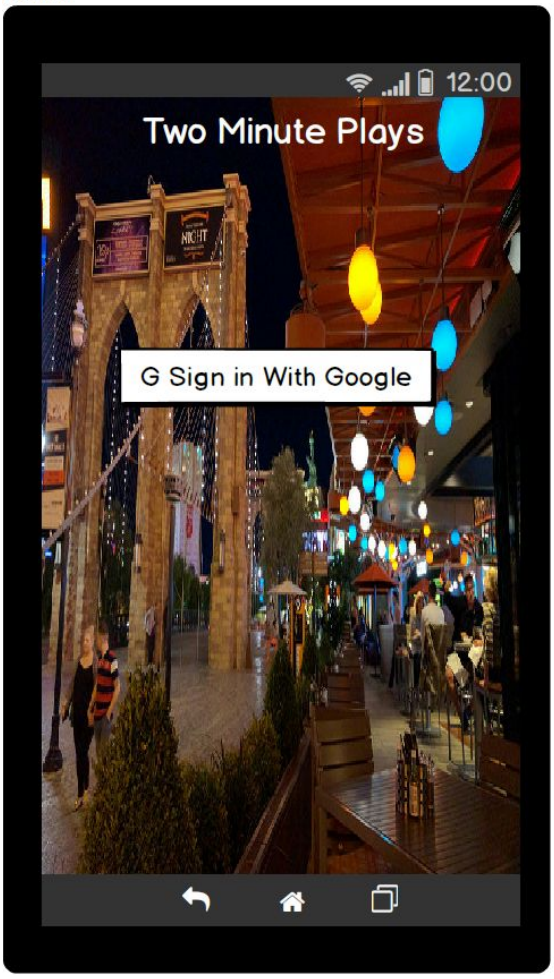
- Takes pictures and video
- Saves movie data to a local database
- Saves and reads movie data to a remote database (Firebase)
- Saves movie clips to a CDN
- Allows users to sign in with their Google account (OAuth)
- Deep linking and sharing

User Interface Mocks

Login View

When the user first opens the app, they are prompted to log in via their Google account. A blurred image is in the background to create depth.

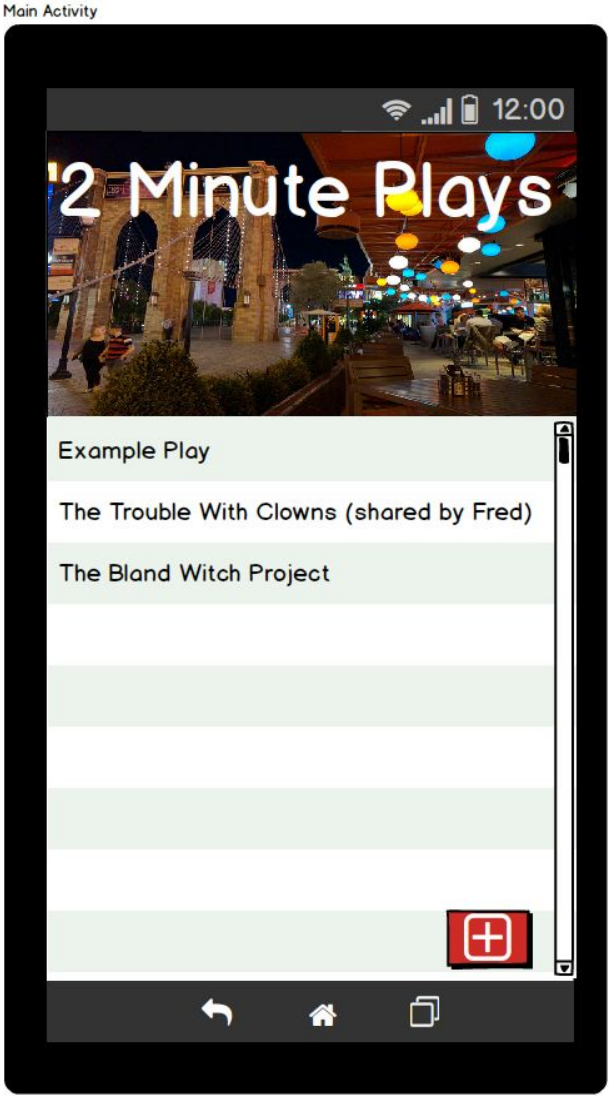
Login View



Home View -- Main Activity

The Home Screen displays a Card View collection of movies in the user's library. It displays an Example Play card in the list. This allows the user to get a feel for what the app is like without providing instructions.

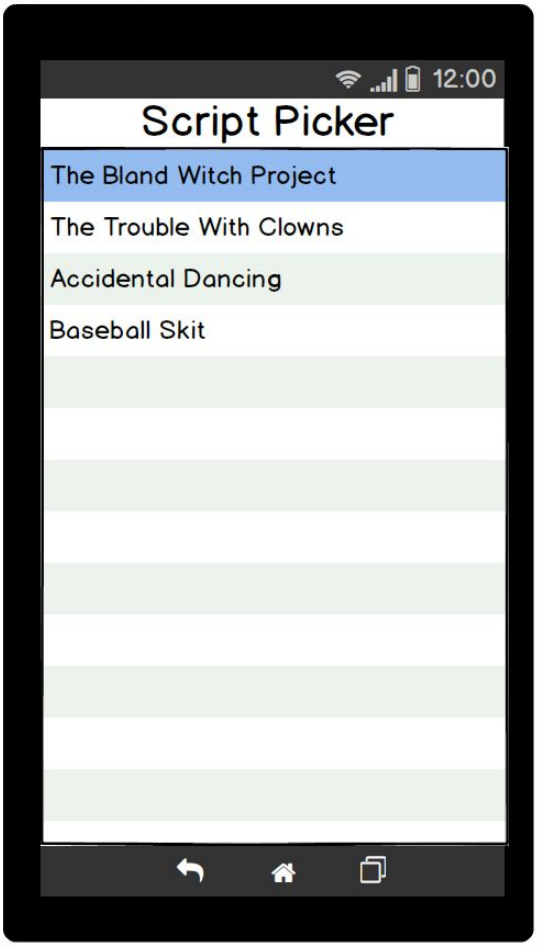
Main Activity



Script Picker Activity

The Script Picker is used to select a movie template. It uses the same card view as the main activity. It displays all available scripts.

Script Picker Activity



Play View Activity

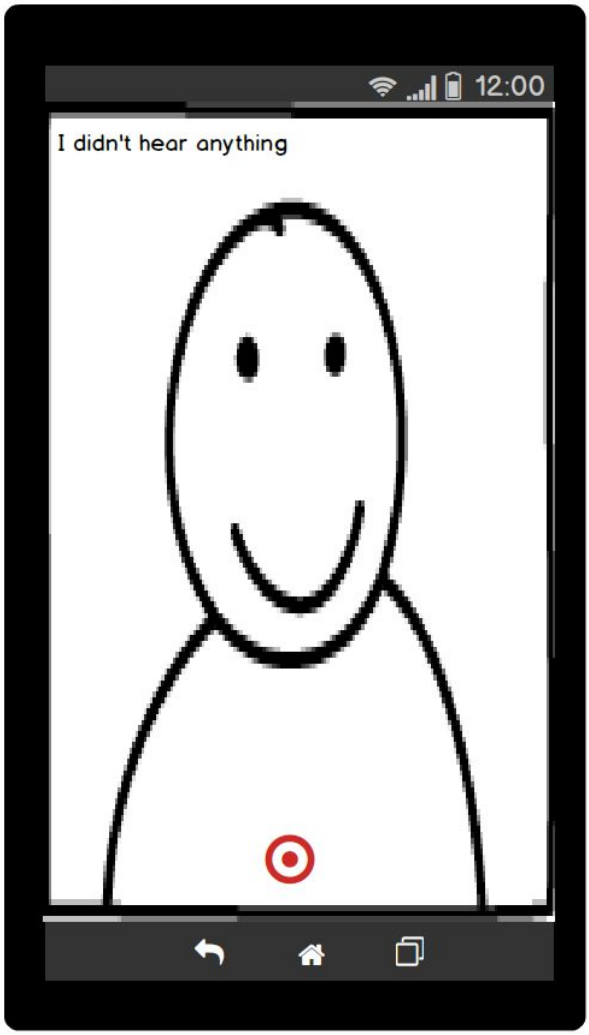
The Play View can operate in several modes. Before you select a part, it will display buttons for each available part. Selecting a part navigates you to the recorder view. Once the play has been shared and merged, a play movie FAB appears in the Play View that navigates to the Movie View.

Play View Activity

Recorder View

The Recorder View is used to control the camera and display the lines. Lines are displayed one at a time. Pressing the record FAB starts recording and the button icon becomes stop. When stop is pressed, the next line is displayed, and a snackbar animates from the bottom, pushing the FAB up. The snackbar says “LINE RECORDED” and provides an undo button. When all lines have been recorded the view navigates back to the Play View.

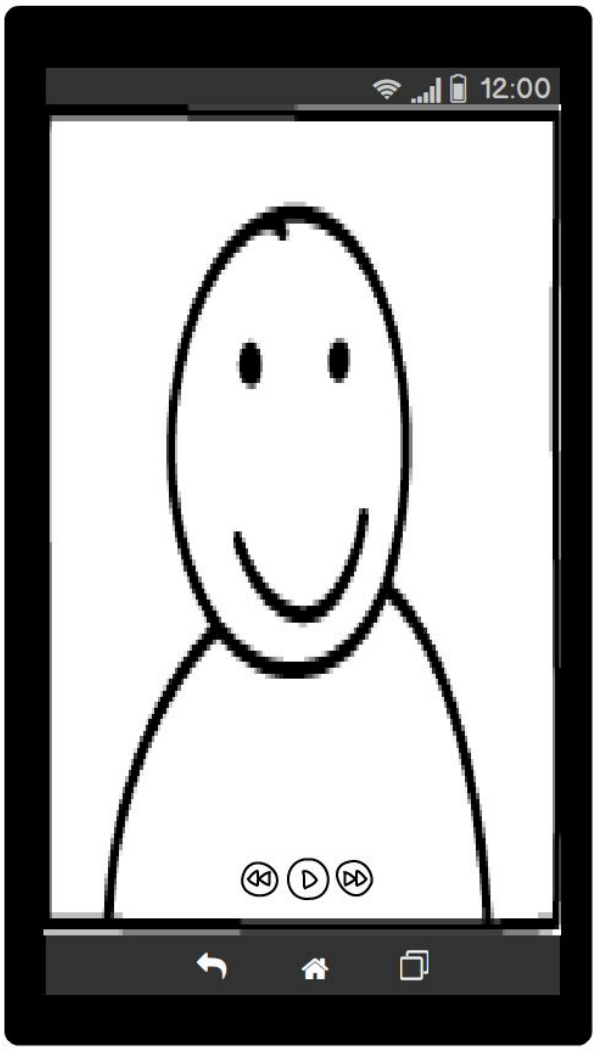
Recorder Activity



Movie View

Used to display the finished movie. Has playback controls and a canvas/TextureView.

Movie View



Key User Interface Flows

Watch A Movie Flow

The Main Activity has a Card View collection of movies the user has completed in their library. To watch a movie, tap the movie which navigates to the Play View Activity, then, the user can tap the Play FAB which takes the user to the Movie Activity. The Movie Activity has media controls to play the movie.

Alternatively, the user can navigate directly to the Movie Activity by tapping the play button on the card view item.

Record A Movie Flow

Starting on the Main Activity, the user can press the New Movie (+) FAB which navigates to the Script Picker. The users selects a Movie Template and navigates to the Play View. On the Play View, the user selects a part to play which navigates to the Recorder View. The lines are recorded one at a time in the Recorder View until all they are done. When the last line is read, the user is popped back to the Play View.

Shared With Me Flow

When a play is shared with me via e-mail, SMS, or some other manner via a deep link, the user is navigated to the Play View. The part the owner has already recorded is not available to select. When the user selects the other part to play, lines are recorded in the Recorder View.

Key Considerations

How will your app handle data persistence?

The app will use a Content Provider for local data access. For sharing, the app will use Firebase. And Google Cloud Data Services for an inexpensive large data store.

Describe any corner cases in the UX.

- The basic movie creation flow is as follows: user logs in, selects a movie template, chooses a part, and reads the lines for the part. If the user is interrupted before reading all lines for a part, and then returns later to complete reading their lines, they last line they didn't record is displayed.
- Choosing another role in the movie will delete any lines you have already read.
- Once you have shared a movie, you cannot make changes.
- If the device does not support the required video format or does not have a front facing camera, a message is displayed to that effect and the app terminates itself.

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

Firebase Client (com.firebase:firebase-client-android) For remote database
Java MP4 Parser (com.googlecode.mp4parser:iosparser) For concatenating video clips

Android support and design libraries for Material Design, e.g., Card View, Recycler View

Sharing Philosophy

Anyone with the sharing link can record lines for your movie. This way, you can share your movie with one person via a text message or multiple people via Twitter. Anyone can audition to be in your play.

Sharing Algorithm:

1. Movie share hash is generated from the movie ID (MID) and private salt (MD5)
2. Information about the movie is saved to /movies/MID. Including the owner, the movie template, and which part was recorded by the owner. All of this information is read-only by any logged in user.
3. A deep link to the app is shared with movie ID and the hash
4. Friend receives deep link in any way (SMS, e-mail, whatever)
5. Friend opens deep link on their device. Hash is verified.
6. If ok:
    a. Reads movie info from /movies/MID/sharedWith/HASH/
    b. After recording lines, writes info to /movies/MID/sharedWith/HASH/friends/UID. Data written will be the part recorded and links to the individual line clips.
    c. The line video clips are saved to Google Cloud Storage.

Any valid user with the movie ID and hash can contribute to your movie.

Next Steps: Required Tasks

Task 1: Project Setup

- Create a new Android Studio project with a main activity.
- Package is com.twominuteplays.app

Permissions required:

For Camera:
android.permission.CAMERA
For OAuth:
android.permission.GET_ACCOUNTS
android.permission.USE_CREDENTIALS
For Firebase:
android.permission.INTERNET

Task 2: Implement UI for the MainActivity

The MainActivity is the entry point for Two Minute Plays. It displays a list of all the plays you have created with the plays that have been shared with you. It also displays a special Example Play at the top.

Creating a play is the main idea of this app. Pressing the Add Play FAB navigates the user to the Play List View so they can begin creating a new play.

- Build UI for MainActivity
    - Create a transparent Toolbar app bar with the title of the app: “Two Minute Plays”.
    - Create a PlayCardView that has the play artwork, play name, play description, author, and space for the play actions.
        - Available play actions depend on the Movie State.
        - Possible actions are SHARE (RECORDED), EDIT (anything but SHARED or MERGED), and VIEW (RECORDED, SHARED, or MERGED.)
        - Movies shared with me should display a circular badge with the friend’s contact photo or initial.
    - Create a domain model including a Movie POJO.
    - Mock data for the RecyclerView as a basic array or list as in the documentation.
    - Sort cards so the example movie is first, then my completed or in progress movies ordered in reverse create date order (newest first).
    - Display a FAB for creating a new play that navigates to the ScriptPickerActivity.

Task 3: Implement UI for ScriptPickerActivity

This is the view that shows a list of plays available to create.

- Initially there will only be a few (maybe 5) plays/scripts to pick from. If/when the library starts to grow, a search feature can be implemented.
- Reuse the Recycler View and Card View from the MainActivity. The data model should be populated from the list of all available plays. All plays in this view will be in the TEMPLATE state. For now, just mock the data.
- When a card is tapped, navigate to the PlayViewActivity.

Task 4: Implement UI for PlayViewActivity

This view displays the play’s script and available parts.

- Display the play name in the Toolbar.
- A part is available when someone else has not chosen it.
    - If a part is available, the character’s name is displayed as a button.
    - If a part has been recorded, the person who recorded it is displayed next to the character’s name (see mockups).
- The script is markup, display it in a TextView below the parts.

Task 5: Implement UI for MovieActivity

The movie activity is used to show a completed movie. It has a canvas and playback controls.

- Lock to portrait mode.
- Add a canvas to display the movie.
- Add playback controls.
- Stub out activity methods for play and pause.

Task 6: Implement UI for RecorderActivity

The recorder activity is used to record lines for a movie.

- Locked to portrait mode
- Add a canvas to show the user what they are recording
- Add a recordOrStop button. Button should be raised with text that says RECORD or STOP.
- Add a text label to display the line.
- Add an activity method for recordOrStopPressed. It should toggle the button label between RECORD and STOP. When STOP is pressed, the undo snackbar should animate into place, moving up the recordOrStop button.
- Stub a method for the undo functionality.

Task 7: Implement Core Video Capture

In the RecorderActivity use the camera2 API to capture video when the record button is tapped. When the stop button is pressed, the view displays the next line to record and a snackbar is displayed showing the message “Line recorded” and an UNDO button.

Now, the user can either press UNDO to go back to delete the just recorded video and re-record the previous line or press the record button to record the current line.

If there is no next line to go to, the RecorderActivity is dismissed and the user is navigated back to the PlayViewActivity.

See this example to get started.

Create a stub method for when the video has been captured. Eventually, this method will be used to save the location to the local database and upload it to Google cloud storage.

Attempt to use H264, 720X486, 15fps. If the front facing camera does not support this format, display a message at app startup and quit. “We’re sorry, we do not support your device.”

Task 8: Implement Local Database

Database use-cases:

- Find my movies
- Find movie templates
- Create movie
- Find a movie by ID
- Update movie
- Delete movie

To implement this in code create the following classes:

- Create Contract with content URIs for each use-case
- Create ContentProvider with
- Create DB Helper subclass of SQLiteOpenHelper
    - Create a file called baselineData.json which has movie template data.
    - onCreate insert the baseline data from the baselineData.json file.

Task 9: Wire Up Content Provider to Views

- Replace the mocked data provider with the database content provider.
- In the RecorderActivity, update the Movie with the path to the recorded video file for the each line recorded.

Task 10: Write Video Handler Service

Create an IntentService that will do the video handling. That is, combining all the lines into a single movie.

- Integrate Java MP4 Parser, adding the dependency to the project gradle file.
- Create the VideoHandler IntentService. This service expects to be passed a Movie object. In the onHandleIntent method, grab the Movie object from the Intent and concatenate all the files in the movie.
- When done, the completed video is uploaded to the Google Cloud Storage service and the local file location is saved to the local database.
- Individual movie files are cleaned up.

Task 11: Add Distributed Firebase Database

- Create Firebase account and app
- Create Firebase app
- Create Firebase security rules:
    - Only the owner can read or write to /movies/MID/OWNER
    - Any valid user can write to /movies/MID/sharedWith/HASH/friends/UID
- Enable Firebase Google OAuth
    - Create credentials
    - Add Google client ID and secret to Firebase OAuth console

- Add firebase client to gradle. Fix any overlapping libraries.
    - ```
      dependencies {
      ```
    - ```
        compile 'com.firebase:firebase-client-android:2.3.1+'
      ```
    - ```
        packagingOptions {
            exclude 'META-INF/LICENSE'
            exclude 'META-INF/LICENSE-FIREBASE.txt'
            exclude 'META-INF/NOTICE'
        }
      ```
- Add Firebase.setAndroidContext(this) to Application class to init firebase.

Task 12: Add Auth UI

- Create the LoginActivity
- When user presses Login With Google buton:
    - Get OAuth Token just like this example
    - Call authWithOAuthToken()

Task 13: Add Sharing

Sharing user's perspective:

- The Movie POJO is saved to the Firebase path /movies/MID/OWNER
- Integrate the MobileDeepLinking library and create a scheme for the app. This will provide cross-platform deep linking in the case that we port the app to iOS.
- When the share icon is tapped on the card, generate the deep link and use Intent.ACTION_SEND to share the link.

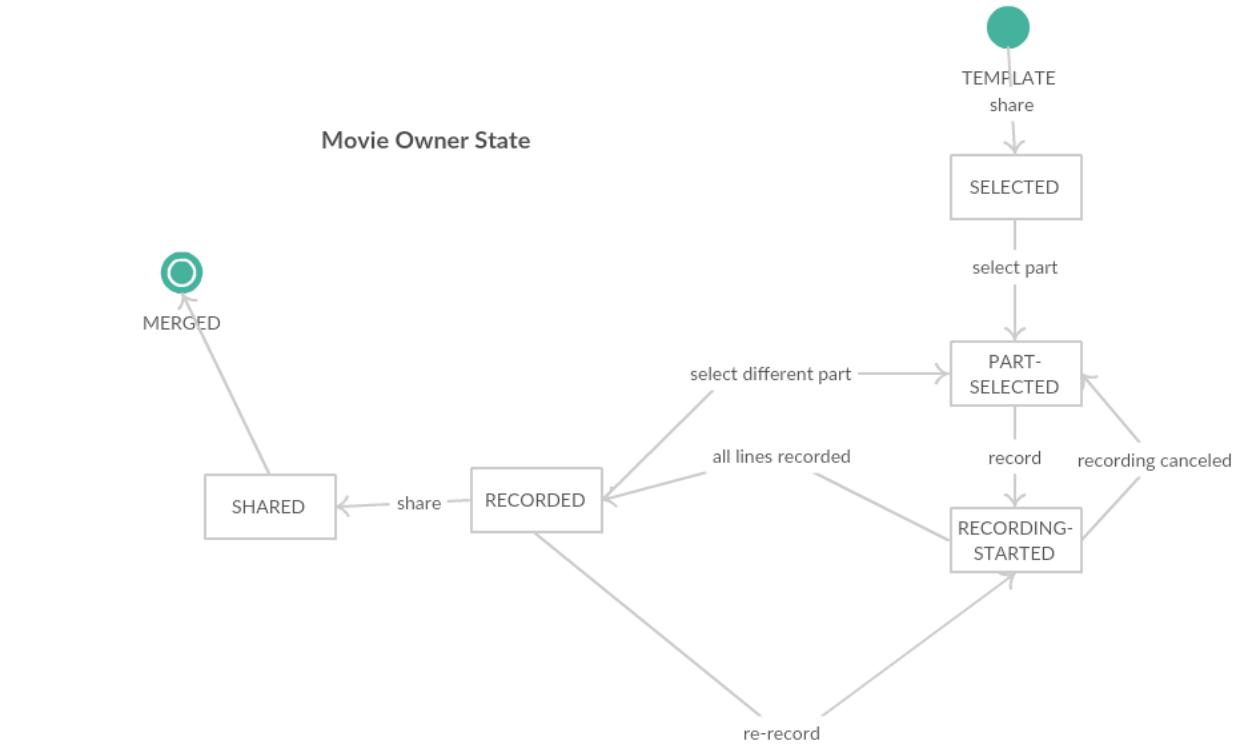
Friend's perspective (movie shared with me):

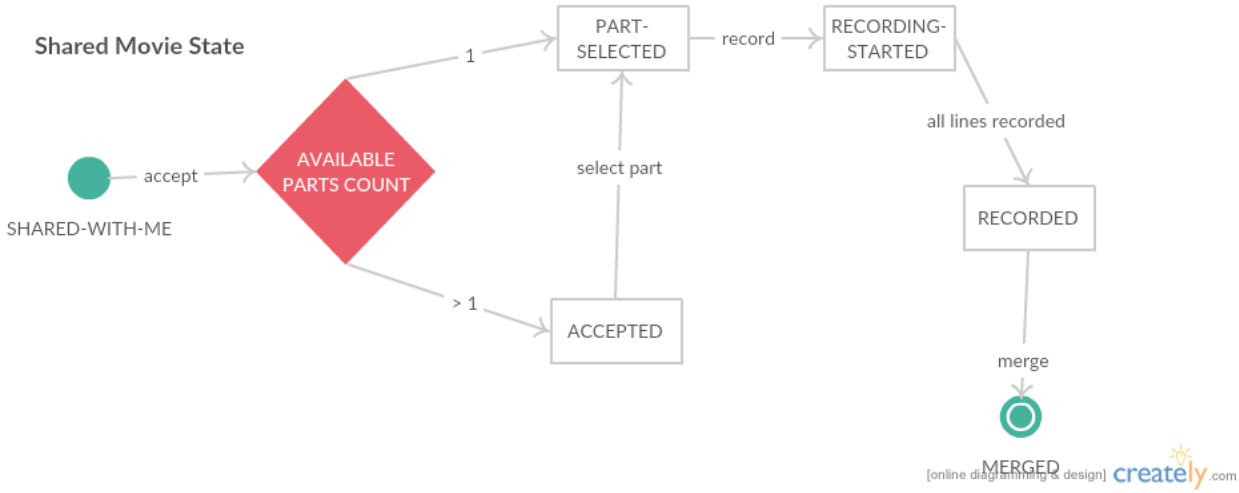
- The deep link points to the MovieActivity. Write code to detect the deep link, read the movie data from Firebase, setup the given movie template, and disable choosing the part the owner has already recorded.
- When the user records the lines, they are saved to Google Cloud Storage
- The link to the line clips are saved to /movies/MID/sharedWith/HASH/friends/UID/lines/LINE-ID

Sharing users' perspective; event listener

- Create a value event listener. Every time /movies/MID/sharedWith/HASH/friends changes:
    - Insert the data into the local database
    - Merge the video using the Video Handler's merge method
      ```

Appendix 1: Movie State Transition Diagrams





Appendix 2: Local Database Design