

**Project 2: Twister**  
Android App Development  
100 points

In teams of twos, you are to create a Twister app that allows users to create and view “twists”, short messages limited to 150 characters. The app uses a web API to access twists that other users have created and saves the twists locally in a SQLite database in case network connectivity is lost.

Here is a working Twister app for becoming familiar with how the app should work:

<http://cs.harding.edu/fmccown/twister-mccown.apk>

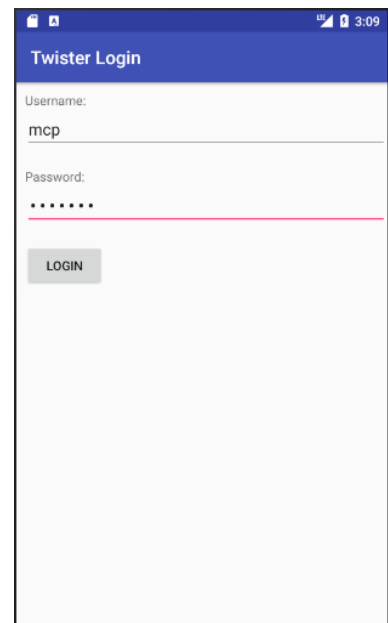
## Screens

The app is composed of four screens:

1. Login Screen – For authenticating
2. List Screen – For browsing twists
3. Add Twist Screen – For posting a twist
4. Details Screen – For viewing details of a user and their twists

## Logic Screen

The Login Screen allows the user to authenticate using a username and password. To simplify the assignment, the username is submitted to the Twister web API to verify the user exists, and the password is ignored. If the web API returns the user’s details, then the user has successfully authenticated, and the List Screen should be displayed. If the API returns back an error, a logic error message should be displayed. The user cannot move to the List Screen until entering a valid username.

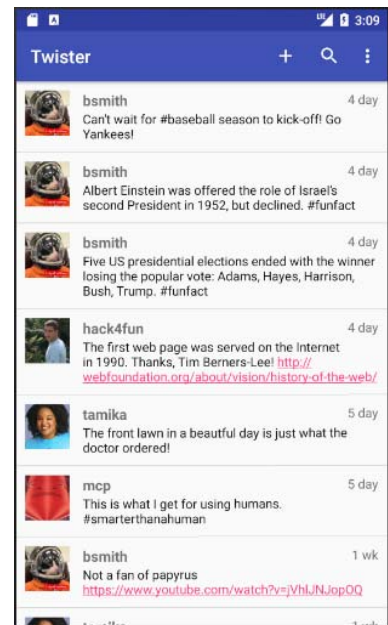
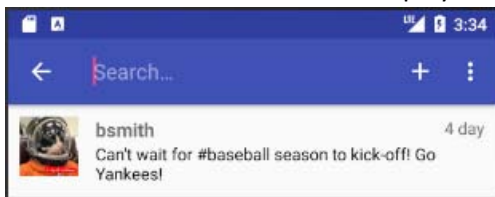


## List Screen

The List Screen displays all the Twists returned by the web API. A RecyclerView should be used to display a scrollable list of twists that display the user's image, the username, a relative time when the twist was posted, and the twist message. If the user presses a twist, the Details Screen is displayed for the selected twist's user.

The action bar supports the following options:

1. Add a twist – Choosing the icon should display the Add Twist Screen.
2. Search for twists – Choosing the search icon should display a SearchView in the action bar that allows the user to enter text and display those twists that match the text. Pressing the Up button cancels the search and displays all twists.

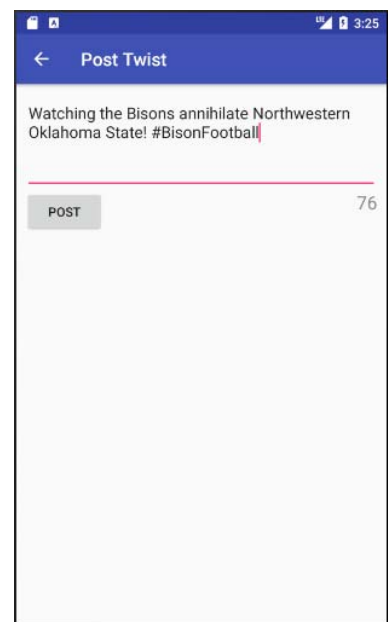


3. Logout - An overflow menu (shish kabob menu) displays a logout option that should display the Login Screen.
4. About – An about overflow menu option should display an about dialog box identifying the two programmers who wrote the app.

## Add Twist Screen

The Add Twist Screen allows the user to enter a twist up to 150 characters in length. The Post button should only be enabled when the Twist contains at least one non-white space character and no more than 150 characters. The number in the right corner should indicate how many characters are left.

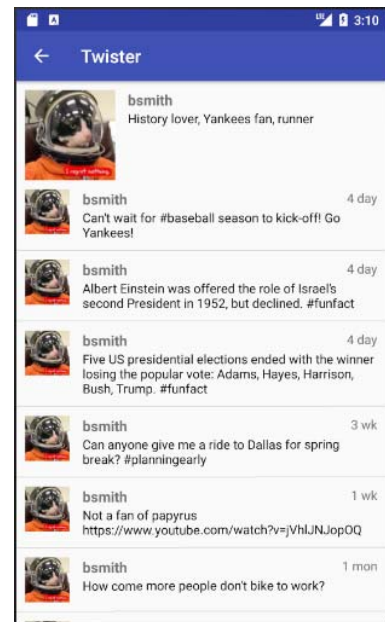
Pressing the Up button shows the List Screen and does not post the twist. Pressing the Post button posts the twist and shows the List Screen. However, the new twist will not display in the list of twists because the twist is not actually posted to the web API. However, if airplane mode is on, the twist will display along with all other twists saved in the SQLite database.



## Details Screen

The Details Screen displays the user's details (image, username, and about) at the top and a scrollable list of the user's past twists. The web API is used to obtain all the information, and a RecyclerView is used to display the twists. Pressing a twist does nothing.

Pressing the Up button shows the List Screen.



## Architecture

The app should be composed using MVC to separate the application logic from the business logic:

Controller:

1. **AddTwistActivity** and **AddTwistFragment** – Adds new twists using TwisterDatabase
2. **DetailActivity** and **DetailFragment** – Displays user details and twists using DataFetcher
3. **ListActivity** and **ListFragment** – Displays twists and search logic using DataFetcher if the network is available or TwisterDatabase if the network is unavailable
4. **LoginActivity** and **LoginFragment** – Authentication logic using DataFetcher

Model:

1. **DataFetcher** – Retrieves info from Twister Web API using Volley. The class should provide interfaces that fragments can implement to retrieve the fetcher's data.
2. **TwisterDatabase** and **TwisterDatabaseHelper** – Classes for accessing and saving twists locally in Sqlite

You may create other classes as needed for implementing the project.

## Twister Web API

An API has been created on jsonstub.com that allows the app to access twists and user information. The API supports GET requests only. The base URL for the API is <https://jsonstub.com/>

1. `/twist/` - Returns a JSON list of all twists in the system.
2. `/user/username` – Returns JSON details about the user. Only usernames found in the list of `/twist/` results are supported.

To make the GET requests, submit the following HTTP headers:

```
Content-Type: application/json
JsonStub-User-Key: edbc267a-f880-4dec-8dec-727cccc27e5d
JsonStub-Project-Key: 40e26003-fc1b-40f3-9ae4-bfab71e6d186
```

## Requirements

Implementing all the requirements below is necessary to receive 100 points:

- The app should implement the previously described screens and architecture. Fragments should be used to implement all screens.
- The app should use the Twister Web API to access user and twister information when the network is available. If the network is unavailable, the SQLite database should be used to access twists.
- New twists should be saved to the SQLite database. Ideally the twists would also be posted to the web API, but this is not a requirement.
- The SQLite database should provide the ability to search twists using a SQL LIKE clause, matching usernames or messages that contain the text. Ideally the text should be matched as the user is typing, but it's acceptable to match when the search button on the keyboard is pressed.
- The app should have a custom icon.

## Bonus

- 10 points: Display the detail fragment and list fragment at the same time if the device is in landscape mode.
- 5 points: Use a regular expression and HTML to make the search text bold where it matches the message text. Example: `<b>search term</b>`
- 5 points: Allow the user to click on links in twists. This requires using a regular expression and HTML to convert URLs into `<a>` tags.

- 10 points: Implement image caching so if the network is unavailable the images are still displayed.

## Teams

Everyone has been assigned to a two-person team:

1. Heather Anderson and Bryan Cuneo
2. Nathan Burner and John David Griffin
3. Samuel Cobb and Caleb Krise
4. Jhoel Zuniga and Benoit Lacoss
5. Bradely Marques and Zack McKee
6. Noah Kinslow and Jerred Shepherd
7. Jared Nesbit and Caleb Spann
8. Trevor Hale and Joshua Werzt
9. Nicholas Terrill and CJ Wilson

Both teammates should aim to contribute equally to the project. Teams may use pair programming if desired. Ideally, the project should be implemented in parallel with each teammate responsible for a part of the application.

Each team should create a private repo on github.com and add me (fmccown) as a contributor. Both teammates should consistently push their work to the repository so a history of the project's creation is accurately captured on GitHub.

The repo's README file should contain the following information:

1. Link to the repo on GitHub
2. Brief summary of what the program does
3. List of any known bugs that still remain (hopefully none)
4. List of any extra credit that was implemented
5. List what each teammate contributed The percentage of work performed by each teammate

Ideally the work performed by each teammate will be 50/50, but if one teammate does more than the other, the percentages should be adjusted accordingly. The percentages will influence each teammate's final grade on the project. Note that the repo history on GitHub should be a fairly accurate indicator of how much work each teammate did. Pair programming is one technique that helps to equalize the amount of effort each teammate puts in, but it does not allow for parallel implementation.

## Grading

These are the things I will be looking for when evaluating your programs:

1. Does your program implement all the requirements and work properly?

2. Is your program well written? Did you use proper Java naming conventions for variables, methods, and classes? Did you use constants and methods when appropriate?
3. Did you clearly cite any code that you may have found online?
4. Did you use git adequately when developing your solution? There should be at **least 20 commits** in your repo, and each person must have committed at least twice. Your .gitignore should keep out any unnecessary files.

The app will be tested on a Nexus 5 emulator.

Your source code on GitHub should not be modified after you have submitted the README to Easel. I will clone your repository on my machine to run and grade it. One teammate should submit the team's README file to Easel before the deadline along with the app's APK file.

This is a challenging project to implement. Starting early and getting help when you need it will benefit you greatly! Like any assignment, you are not to share your code with anyone but your teammate! That's why you are using a private repo.

## Implementation

To implement search, you'll need to dynamically change what is displayed in the RecyclerView. You'll find this link helpful when figuring out how to do it:

<https://stackoverflow.com/questions/31367599/how-to-update-recyclerview-adapter-data>

To convert a twist's timestamp into a more usable "time-ago" string, check out:

<https://stackoverflow.com/questions/3859288/how-to-calculate-time-ago-in-java>