InstaMeme
Design Review

Niklas Bauer
Curtis Schumacher

Application Overview

Meme

Application Overview

1. Take a photo OR load from memory
    - We will provide "stock" photographs
2. Edit the Photo
    - Add text, will automatically resize itself
    - Possible: Crop the photo
    - Photo saved to internal memory
3. Post to social Media
    - Twitter, Facebook
    - If enough time, Pinterest, Instagram

Application Overview

System Metaphor: Photo Booth/Caption Contest

Why this app?

- Memes incredibly popular
- Sharing important/fundamental
- No app exists with a camera

Business Process

- Allow an ad to display after user has posted

Functional Requirements

- User Interface
  - follows UI design principles
  - big, clear buttons with obvious choices, does exactly what the user wants it to
- Camera
  - Writing own camera in case device does not have app
- Gallery Selection
  - Load phone's gallery
  - Select from stock photos
  - Simple UI important here!

Functional Requirements

- Photo Editing/Captioning
  - Uses selected or recently taken photo
  - Two text boxes
  - Font in standard "meme" font, called Impact

- Posting to Social Media
  - Let user decide to post to one or multiple networks
    - Facebook and Twitter first to be implemented
  - Ask for permissions
  - Speed of upload a priority

Non-functional Requirements

- Camera
  - zoom
  - flash
- Saving Photo
  - After taking it with camera
  - After editing
- Other Photo Editing
  - Crop the photo, change font, add "Scumbag Steve" hat, etc.

Non-functional Requirements

- App font, background color
- Notification of progress
  - Meter filling to complete, or showing "Step n of 5"
- Pull pictures from other locations
  - Google Drive, DropBox, Picasa, etc.

Current Status

- The app is in its early alpha state.
  - Able to select and load photos from the gallery.
  - Click through the bulk of the app.
- We are almost done building the camera section of the application.
- Currently we are working on pulling pictures from other folders, such as google drive and dropbox.
- Then on to picture editing!

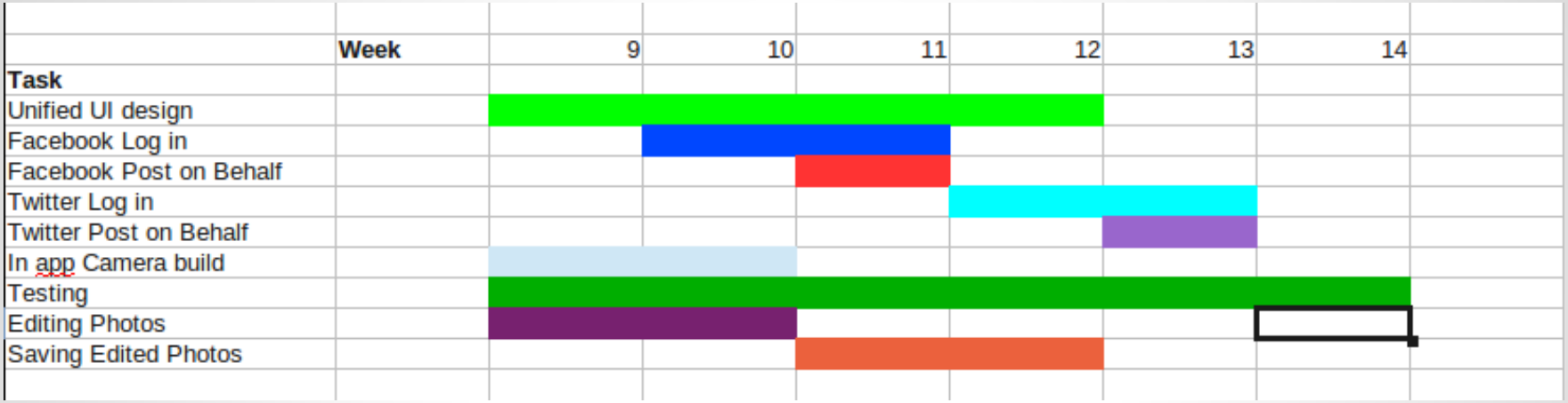
The future of InstaMeme

- Facebook and Twitter interfacing.
  - Login support
  - Post on your behalf support
- Photo editing
  - Meme text
  - Basic photo editing
    - Cropping
    - Extra additions in the meme style
- Unified UI
- Custom launcher Icon

Timeline

- June 25- Select and loading images complete
- July 4- Edit photos, loading stock images
  - captioning, font chooses correct size
- July 11- Camera complete
  - zoom, flash if possible
- July 18 - UI completed
  - finalized font, buttons, color scheme, custom icon
- July 25 - Save photos
  - Add non-functional requirements as well
- August 1- Post to social media, begin test-only
- August 8 - All functionality complete

Timeline



	Week	9	10	11	12	13	14
Task							
Unified UI design							
Facebook Log in							
Facebook Post on Behalf							
Twitter Log in							
Twitter Post on Behalf							
In app Camera build							
Testing							
Editing Photos							
Saving Edited Photos							

Twitter API

Tweet info examples

- Contributors

```
"contributors":
[
    {
        "id":819797,
        "id_str":"819797",
        "screen_name":"episod"
    }
]
```

- Coordinates

```
"coordinates":
{
    "coordinates":
    [
        -75.14310264,
        40.05701649
    ],
    "type":"Point"
}
```

Twitter API

Tweet info

- created_at - timestamp
- current_user_retweet -id of the retweet
- favorite_count
- favorited
- id_str -for javascript, because of ints>53bits
- in_reply_to_screen_name
- in_reply_to_status_id
- retweeted
- truncated

Twitter API

Users info follows similar style

- default_profile
- default_profile_image
- description - "Official InstaMeme Page"
- following - bool
- followers

Also extends to "entities" such as hashtags, urls, hyperlinks, user mentions, multimedia

Twitter API - REST API v1.1

GET commands

- Return data from Twitter

- GET statuses/user_timeline
- GET statuses/retweets_of_me
- GET statuses/show:id
  - returns status with given id number
- GET direct_messages

Twitter API- REST API v1.1

POST commands

- Post new statuses, direct messages, update profile, etc.

- POST direct_messages/new
- POST direct_message/destroy
- POST friendships/create
  - now following user. ID of use to follow entered as param

Twitter API - examples and how to

- Register app on the Twitter website
- Download appropriate .jar files and libraries
  - add to application
- Need to connect to Twitter first
  - log in
  - verify app allowed to post on user's behals

Twitter- examples and how to

Log in and verify

- Twitter object

```
mHttpOauthprovider = new DefaultOAuthProvider("http://twitter.com/oauth/request_token",
                                              "http://twitter.com/oauth/access_token",
                                              "http://twitter.com/oauth/authorize");
```

- getAccessToken() method
- twitterobj.setOAuthAccessToken()
- verifyCredentials()

Twitter- examples and how to

- Editor object
- Editor.putString("auth_key", getAccessToken())
- twitterObj.updateStatus();
  - catch TwitterException();

- Always need to check for for authorization, get user credentials, open/close connection, and error check all calls!

Chain-of-responsibility Pattern

Object based behavioral design pattern.

Motivation:

- Avoids the need to directly link the object creator to the object handler.
- Make object handling clean and avoids the necessity to make all objects and methods public to handle objects.
- Allows private methods to handle objects of different classes.

Chain-of-responsibility Cont'd

Intent:

- Remove the necessity to attach the object creator to the object handler.
- Gives other objects the ability to handle a request.
- Allows for processing of different types of requests e.g. mouse events or keyboard events

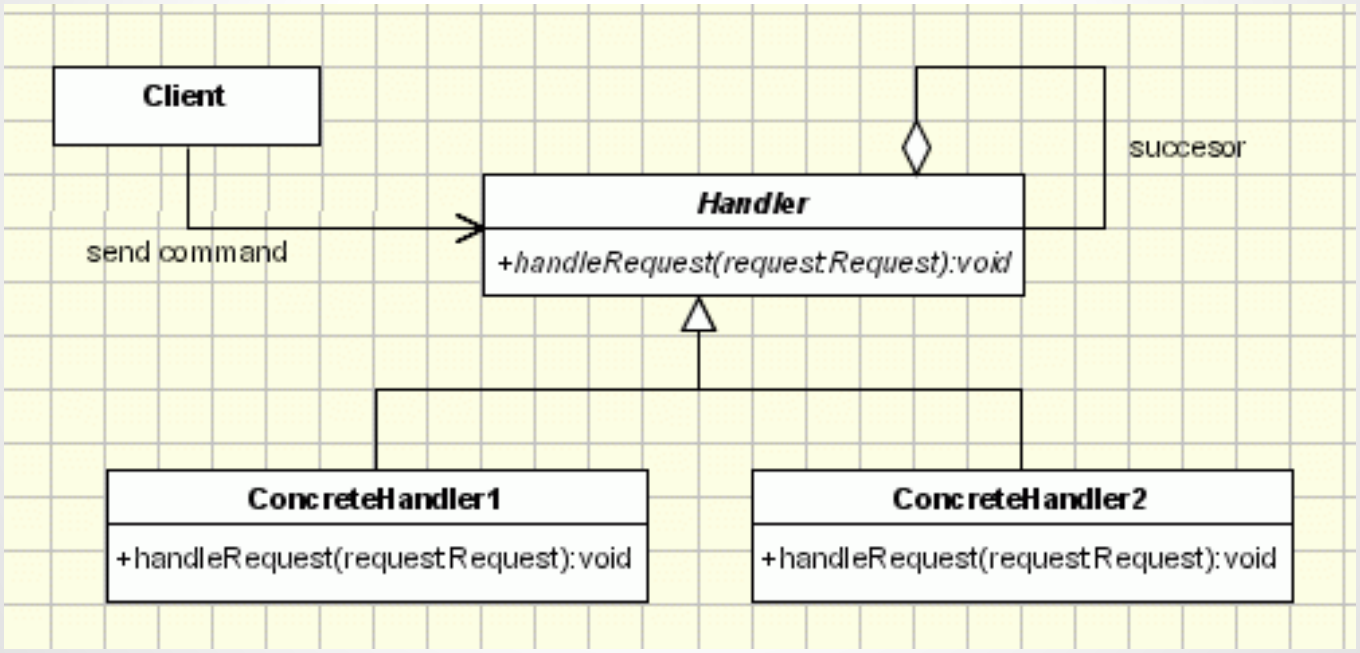
Chain-of-responsibility Cont'd

Participants:

- *Handler* - defines an interface for handling requests
- *RequestHandler* - handles the requests it is responsible for. If it cannot handle the request, it is sent on to the successor.
- *Client* - Sends commands to the first object in the chain that may handle the command.

Chain-of-responsibility Cont'd

Implementation:



Chain-of-responsibility Cont'd

Possible applications:

- When more than one object may handle a request and the actual handler is not known in advance.
- The handler needs to be selected automatically based on the object type, size, or other characteristics.

Chain-of-responsibility Cont'd

Consequences:

- Unhandled requests.
  - The chain does not guarantee that a request will be handled.
- Breaking the chain
  - Programmer error; being that they don't implement a handler for a specific case and the chain is broken.

Benefits:

- Dynamically choose the object handler
- Allows for greater obfuscation
  - No hard coded class names and such in the handler.