ECE 563 Final Project - Omnimoji DB

Django File Server Final Project Report

Binwu Peng, Ting Chen

20190501

1 Demo and tests

File service from web browser: censored
Repository containing the source code for file service: https://gitlab.oit.duke.edu/tc233/ece563
Repository containing the source code for iOS app: https://gitlab.oit.duke.edu/ECE564/Omnimoji/Project The *Omnimoji* app beta version (netid login): https://appstore.colab.duke.edu/apps/78

Steps to use the service

- For server frontend side
  - Create user password account
  - Upload your file or image to this server with form *Upload file by form* on the left
  - Specify an expiration time in iso-UTC format
  - Specify how many times it can be downloaded
  - Download the file and the view count will decrease
- For mobile app side
  - Download the app to your phone
  - Set the username in settings page according to your registration
  - Create a sticker from any of the features in the app
  - Scroll to right and tap the *share* button
  - Tap *Share to Django* button
  - It will put a link in the clipboard as well as a QRCode on screen
  - Tap *Share* if you want to share the link to other apps

Also, if you'd like to have a look at our iOS app source code, please send Ting an email and he'll invite you to the group.

2 Purpose of this project

Initially we want to build an extension to our last semester iOS project *Omnimoji* app, which is to create a temporal image server so that our users can share the self-created stickers to others. Most importantly, with this service, we can share the sticker through WeChat and bypass its limitation for GIF format images. Later we found that we can generalize it to a file server and share file with each other, among different devices and platforms, and utilize this file server as a minimum temporal storage.

2.1 Background and usage

Last autumn, we created a cool emoticon generator called *Omnimoji* as our course project for iOS programming. One restriction for the sharing functions for the app is that it cannot share animated GIF file to WeChat due to its restrictions and security issues.

The follow-up updates for it is to create (or find) an easy-to-use API to share the emoticons and stickers with those who cannot view animated image formats, or from mobile device to desktop. The main method is to save the sticker on a remote server and give users a link in plain text format to share with others, so that the receiver can view the image in any web browser. In addition, since most uncompressed GIF files are large in size, and there is no elegant way to compress them on a resource-limited mobile device, we'd like to process those GIFs with a backend server with Python scripts, which allow us to use Python libraries like *Gifsicle* and *ImageMagick* to process images. Dealing with raw data is not handy with *Objective-C*, but could be much facilitated with Python libs. Last but not the least, Ting is learning system & socket programming this semester, and would like to have some hand on experiences with HTTP requests as well as parsing request data, indexing files with customized entries management "database" and so on.

2.2 Expected features and results

Below are the expected functions of our service at the start of this semester

- Create an easy-to-use iOS toolkit with Swift to integrate temporary file sharing service into any app
- On the device side: create a modified version of *UIActivityViewController* to share/upload the file to the server
- On the server side: the file can be stored, categorized with user's id and tracked with an indexing system
- The user can decide the lifespan/expiry of his shared image before sharing
- The user can decide either the image can only be download once or not
- File size compression
- Provide some processing ability: allow designated Python script to run after the file is uploaded and before it is saved on the server
- Basic responsive UI with image embedded
- User management: signup, login and grouping
- File management from web browser
- File service with customized API

3 Enabling Technologies

Both of the group members are developing web service for the first time, and following tools and frameworks are chosen to develop this project.

3.1 Django framework

Django is an open source web framework providing python API that encourages rapid web application design. With Django, it is much easier to develop a server backend for a variety types of services, rather than using the traditional LAMP (Linux/Apache/MySQL/PHP) architecture. Django has a design pattern similar to MVC, which is Model/View/Form and Template pattern. Data models are defined in models, request handling happens in views and results are displayed at frontend within rendered templates.

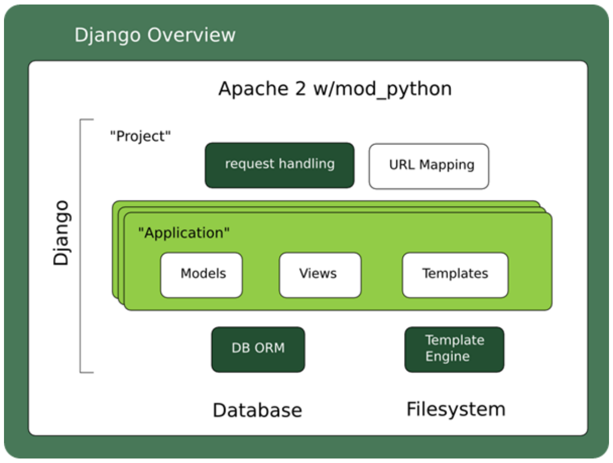

FIGURE DJANGO FRAMEWORK ARCHITECTURE. SOURCE: PIXELS1ST.COM

Django can directly leverage the power of *Python* programming language. Native Python scripts can be directly embedded into Django backend, which brings a large variety of open source modules that can be imported to the service, and allow developers to handle most of the processes efficiently. Especially for image processing, with Python we can avoid rewriting a huge portion of existing algorithms and use image processing methods simply.

3.2 HTML & CSS

Binwu primarily worked on the frontend design for the service. Major HTML tags used are a table and a file upload form. The table is used to list all the files uploaded.

```
<table class='table' style="width:100%">
    <thaed>
        <tr>
            <th> Des</th>
            <th> upload time</th>
            <th> download cnt</th>
            <th> link </th>
        </tr>
        </thead>
        <tbody>
            {% for file in files %}
            <tr>
                <th> {{ file.Des}}</th>
                <th> {{ file.TimeStamp}}</th>
                <th> {{ file.Num}}</th>
                <th> <a href="{{file.Doc.url}}" class="btn btn-primary btn-
sm">download</a></th}}>
            </tr>
            {% endfor %}
        </tbody>
</table>
```

Binwu also created the following CSS code to provide a consistent look for frames.

```
<style> * {
    box-sizing: border-box
}

body {
    font-family: Arial, Helvetica, sans-serif;
}


/* Style the header */

header {
    background-color: #245;
    padding: 1px;
    text-align: center;
    font-size: 15px;
    color: blue;
}


/* Create two columns/boxes that floats next to each other */

nav {
    float: left;
    width: 20%;
    background: #AAA;
    padding: 20px;
}


/* Style the list inside the menu */

nav ul {
    list-style-type: none;
    padding: 0;
}
```

```css

article {
    float: left;
    padding: 20px;
    width: 80%;
    background-color: #f1f1f1;
}

section:after {
    content: "";
    display: table;
    clear: both;
}


/* Style the footer */

footer {
    background-color: #777;
    padding: 10px;
    text-align: center;
    color: white;
}

table,
th,
td {
    border: 1px solid black;
}


/* Responsive layout - makes the two columns/boxes stack on top of each other instead of next to each other, on small screens */

@media (max-width: 600px) {
    nav,
    article {
        width: 100%;
        height: auto;
    }
}
```

```
</style>
```

3.3 HTTP request with Swift

The mobile end uses the *URLSession.shared.dataTask* to create a post request with the file to our file server. Basically, Ting constructed the POST data according to the form structure defined in our data model in Django, and then POST it with Swift HTTP related APIs. Below is a skeleton of the Swift *public func shareAnimatedToDjango(filename: String, VC: UIViewController) -> Void* function. Sorry for the highlighting, Swift seems not to be supported by this tool.

```swift
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    if let error = error {
        // error handing
        print("Error: session : \(error)")
        return
    }
    guard let httpResponse = response as? HTTPURLResponse,
        (200...299).contains(httpResponse.statusCode) else {
            // http response handling
            print("Error: cannot receive response with post request")
            return
    }
    if let mimeType = httpResponse.mimeType,
        mimeType == "text/html",  // text/plain text/html application/json
        let data = data {
        let outputStr  = String(data: data, encoding: String.Encoding.utf8) ?? "unrecognized response txt"
        if outputStr.contains("share") {
            UIPasteboard.general.string = rootURL+outputStr  // copy to Clipboard
            DispatchQueue.main.async {  // Make sure you're on the main thread here
                // ...
                // present views to the parent view controller here on the main queue
            }
        }
        else {
            DispatchQueue.main.async {  // Make sure you're on the main thread here
                // handle failed response here
            }
        }
    }
}
task.resume()
```

3.4 Other iOS Frameworks

In addition to *Foundation* framework, Ting also used *CommonCrypto*, *UIKit* and *CoreImage* to compute the MD5 value for uploaded file and draw the QRCode with transparent background for sharing. Since those code are related more with user experience improvement than the HTTP request process, we will not paste them here. Check out my commentary in the video.

Also, Ting created more sections in the settings page with a customized UITableView and provided a bunch of potential fields that could be embedded into settings. Check our app for a better understanding.

3.5 Webserver

The backend part has many features, including file integrity check with MD5, file format check and animated GIF compression. All of these features are developed based on either native or 3rd-party Python libraries. You can see these functions under */Django_base_directory/ftp/utils.py*. For example, the MD5 function is implemented with Python *hashlib* as below, and calculate the hash of a file in an iterative fashion, which avoids memory overflow when the file size is too big.

```python
def get_file_md5(path):
    hash_md5 = hashlib.md5()
    blocksize = 1048576  # 1 MBytes
    with open(path, 'rb') as f:
        for chunk in iter(lambda: f.read(blocksize), b''):
            hash_md5.update(chunk)
    md5 = hash_md5.hexdigest()
    return md5
```

3.6 Database Table design

We use the lightweight SQLite as our database, and the following is the primary table for uploaded file entries.

Field Name	Type	Description
file_id	primary_key	Auto-incremented primary key
file_size	int	Size of stored file
file_format	String	File format: image or binary
view_count	int	Allowed view count
file_entity	FileField	The file storage handler
file_description	String	File description
path	String	Absolute path to the file

time_stamp	DateTime	Upload time
expiry	DateTime	Expiration time
user_id	String	User-unique id
md5	String	MD5 value for file integrity checking
share_url	String	Link as the response for post data
file_name	String	File name as a reference

CHART DOCS TABLE IN OUR DJANGO DATA MODEL

Also, other tables including Users and Groups are included in the database. Another temporary table is used for storing the temporary file for processes like compression and image filtering.

4 System Design

Our service provides a cross-platform short-term file sharing solution with customizable settings and access controls. Below are the system designs for the major components of our project.

4.1 Architecture

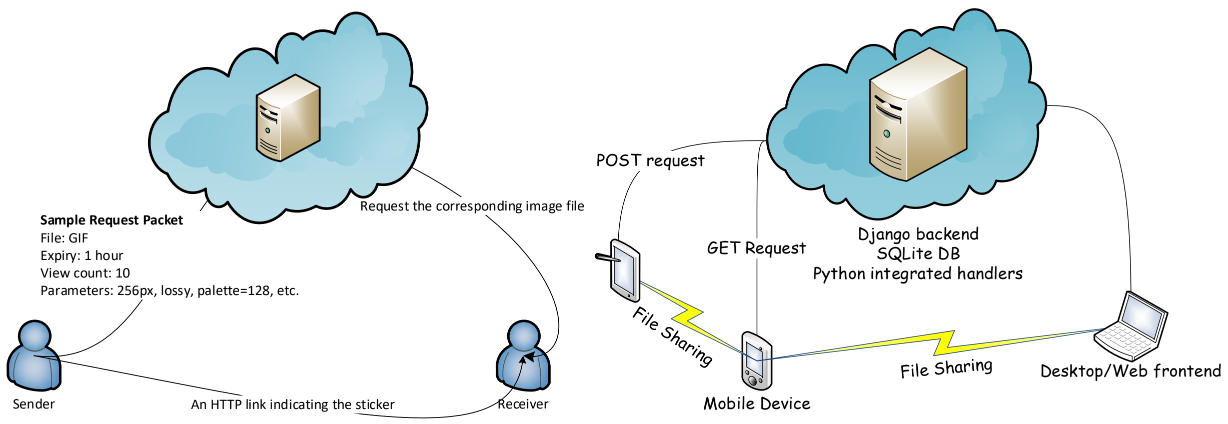

FIGURE LEFT - HTTP REQUEST CONTENTS FOR THE SERVICE. RIGHT - HTTP REQUEST LOGICS AMONG DEVICES

Backend Django server serves as an HTTP file server with additional capabilities, and all the devices of the users communicate with the server with HTTP requests. File sharing is achieved with file-entity-unique URLs, and can be shared among not only mobile devices but desktop computers as well.

The creator of a sticker, or the file uploader first upload the file to the server with a POST request. On mobile devices, this request is automatically constructed according to settings in the app; the web page frontend upload function need the user to manually fill in the required form columns. After that, the server will either store the file or process it before putting into storage, regarding the additional parameters from iOS app settings.

When the recipient access a shared link, a GET request is made and server will update the *view count* field of the certain file accessed. Also, a scheduled task running in the background will check the expired files and delete them accordingly.

4.2 Server design

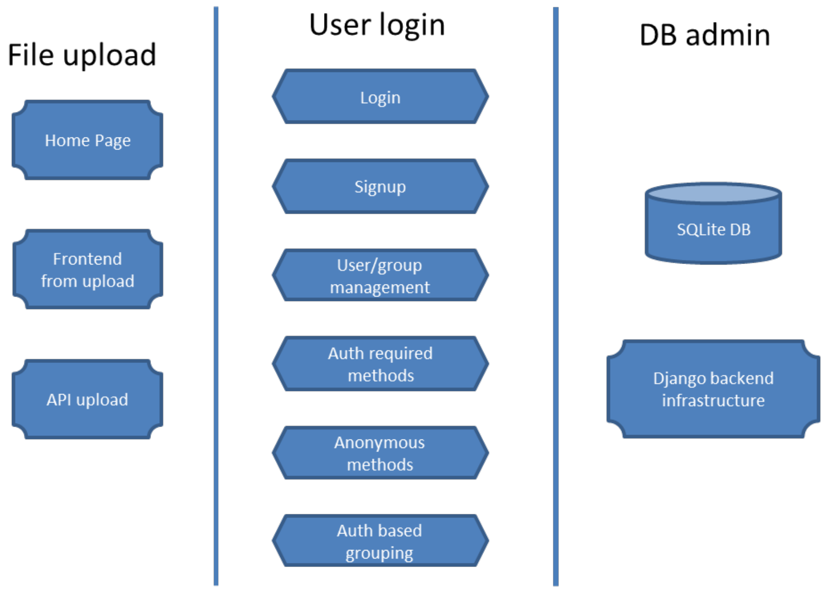

FIGURE SERVER DESIGN WITH MODEL-VIEW-TEMPLATE/FORM PATTERN

The server can be mainly divided into two part: frontend and backend. On the frontend, Binwu created multiple static HTML/CSS templates, and the content at frontend are re-rendered when certain methods in views are called. Also, with the default Django Admin application and middleware, user management can be achieved easily with a user-friendly graphical interface on the web page.

As for the backend, we provided multiple methods to update the content for web pages with specified authentication and criteria. Basic users can view their own collection of files on the server, while administrators can see all the files entries in the table. Also, the upload API as well as the form upload page is rendered by functions in views.

Finally, when a transaction is completed, the functions in views will hence commit the updates or creation of entries to the database, which completes a full lifecycle of an entry.

4.3 Mobile app design

The app inherited most of the design patterns of our previous development. Improvements are embedded into the sharing related functions with minimal changes to the original code.

When the user tap the share button in a feature, the app pops up a menu. If the user select Share to Django, then the sticker is rendered asynchronously in the background, and when the sticker is ready to share, it calls a function to pop up a modified *UIAlertView* to let user decide sharing methods. For both choices, the URL for shared image will be paste into users' clipboard, so that they can share the sticker with others by pasting the URL to any of the message text boxes.

Before uploading, the app will check MD5 value of the image file, and will append a time stamp to the POST data, so that the server side can check file integrity in case unstable connection damaged the file. After the file is successfully uploaded, the server will return a success message along with the generated URL, and the app can write the URL to clipboard.

5 Results

5.1 Screenshots

Instead of providing some screenshots here in the document, we've attached a short video to describe the whole project. Please check the video in the attachment or at: http://duke.is/jAqSgt. Please note: after iOS 12.x Apple enforced some audio encoding format, which means the audio track of screen recordings can only be decoded with QuickTime or iPhone native video player.

5.2 Target accomplish list

- ✔ Create an easy-to-use iOS toolkit with Swift to integrate temporary file sharing service into any app
- ✔ On the device side: create a modified *UIActivityViewController* to share/upload the file to the server
- ✔ On the server side: the file can be stored, categorized with user's id and tracked with an indexing system
- ✔ The user can decide the lifespan/expiry of his shared image before sharing
- ✔ The user can decide either the image can only be download once or not
- ✔ File size compression
- ✔ Provide some processing ability
- ✔ Basic responsive UI with image embedded
- ✔ User management: signup, login and grouping
- ✔ File management from web browser
- ✔ File service with customized API
- ✘ The user can specify the key in the link, e.g. https://myserver.edu/[specific_key]
- ✘ File size limit

5.3 Highlights

Backend

5.3.1 Exception handling

The overall exception handling is taken into consideration, and the service can deal with most valid requests that might lead to exception like FileNotFound, AccessDenied, etc.

5.3.2 Integrity checking

MD5 file integrity checking is implemented on both server-end and iOS client.

5.3.3 View count

Provided a field to limit the view count of a file, so that the uploader can limit the popularity of a sticker into a smaller range.

5.3.4 Expiration handling

Uploader can specify the expiration period so that the shared sticker could be erased from the server after a certain amount of time, which further improved sharing privacy.

Frontend

5.3.5 Image listings based on user identity

Users logged in can view a list of files that belongs to them, which give them a better notion of their activities.

5.3.6 User management

User can sign up, login and view his file list within web page.

Mobile

5.3.7 Sharing process

The designed sharing process successfully overcome the restrictions of WeChat GIF sharing policies. With the shared link, users can now easily share their newly created sticker to anyone and add it to their WeChat sticker collection.

5.4 Skills learned

This is the first time we create a file sharing service. Below are some skills that we acquired from the development.

5.4.1 Django framework

Below is the summary from Binwu

Django provided a super lightweight and handy framework to quickly develop a web application. In a normal web application development process, we have to setup the database, create the data table use SQL language, setup up the web server, create the dynamical web page and process the application logic use designated language and finally deploy the web package to the web server.

However, in Django, we create the database table by create a python class with all the required field. The Django will help mapping those objects to a SQL table. Actually, we can modify the object anytime, and the Django will help do the migration and there is no need for us to worry about that. As for the web page part, all we need to do is to create a view method and map the web URL to this view by add a path item into the URL object. It also provides a setting configuration file, which will glue all part together. For example, we can change our database from SQLite to MySQL just change one line in the setting file.

Below is the summary from Ting

This is the third time that I use Django as a lightweight backend server, and is the first time to investigate its file service capability. Django is super suitable for smaller personal projects, despite that a project as huge as *Instagram* can also run with Django. I will use Django more often in the following days for developing personal web/cloud services.

Also, when I was developing this service, I found some logic problems that I used to neglect in the past, such as how to ensure a clean transaction, how to design a distributed storage system, etc. I'd love to learn more about distributed and multi-user architectures in the future.

5.4.2 Web application design flow

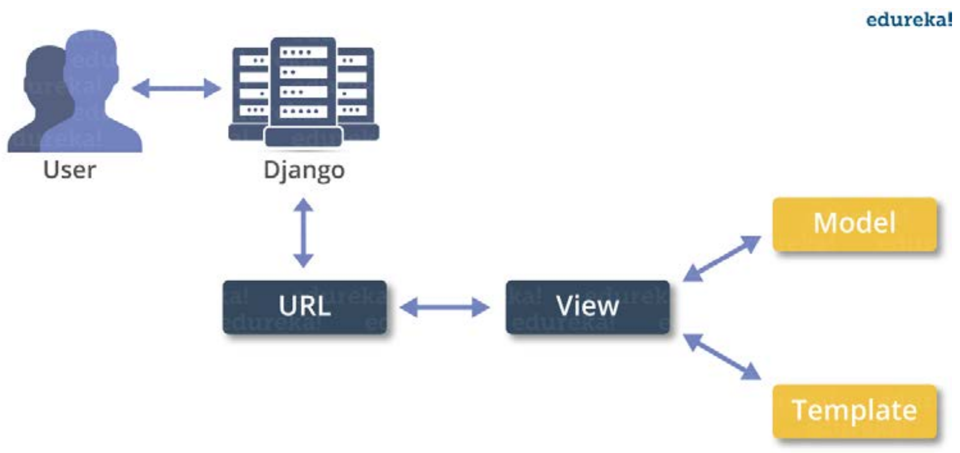

FIGURE DJANGO DESIGN FLOW. Source: www.edureka.co

Below is the summary from Binwu

Although the Django frame work is specific, the general web application design process is the general to all regardless of the framework. We should always start with the database design, because that's where the permeant data stored, wither it is use account data, or our business data like a piece of merchandise record. Then it is about the business logic handler, such as a payment transaction or a database record update. Finally, we should usually separate the static html page content and CSS layout style from the dynamically generated content.

5.4.3 HTTP protocol

Below is the summary from Binwu

The four major http request is Get, Post, Put and Delete. Those four request methods also form the model rest API basis. As long we understand the request protocol and their specific field, we can exchange information with a web server from our own python, java or other code. In fact, our iOS application is uploading file to our Django file server by swift code following the HTTP post protocol.

Also, understand the status code of the HTTP response is very helpful for us to debug our code.

- 2xx stands for success, such as 200
- 3xx means redirection, you are redirect to another page or other website.
- 4xx indicates client Error, such as 401 means you are not Unauthorized to visit current page.
- 5xx represents server Error, such as 503 is to tell you current service is unavailable

6 Contribution of team members

The workload between...

Binwu focuses on the web page and web server logic

- HTML Page layout
- Page link flow
- Consistent page CSS style
- Web server template and framework
- Initial database table design
- File upload and download from the browser
- User registration and authentication
- Password reset

Ting focuses on the iOS app part and web server enhancement:

- iOS file upload and sharing
- Database design
- View count
- File type handling
- Scheduled database cleansing task
- QR code generating
- Exception handling

7 Future improvements

7.1 Distribute file system and Load Balance

Currently, we only use the default Django web and file server. The CPU computation power is limited, so as the disk space. If we our user base increased to a big size such that we need to provide file service as google file or Dropbox, a distributed file system will be established and clients will be evenly distribute to the nearest server.

7.2 High accessibility

Now, we only use one host to run the service, if something wrong with this host, the service is down. To provide better availability service, we would like to expand to multiple servers. Also, those servers keep heart beat with each other, and in case the master server is down, the backup server will run spontaneously.

7.3 User total file space accounting

Now we don’t limit the total space a user can use, to expand this prove of concept demo to a real business model. We would like to account each user and give each user different privilege and charge different feed per month.

7.4 Anti-leech

Currently our service is mainly a static service, where malicious users could potentially leech the file links without making GET requests to Django services. In order to avoid leeching, a few method could apply:

- Dynamic link with authentication parameters encrypted. Just like Getty Image and other commercial image providers, we could add encrypted identification data to the link so that anonymous users cannot access the file entity.

- File unique fetch code. For each file uploaded, we generate a specific fetch code for files. When making request of a file, the fetch code serve as an additional parameter and factor for authentication.

7.5 SSL encryption

To make our service more applicable, an SSL certificate should be added to our VCM domain.

7.6 Additional suggestions for VCM providers

A few things that come to Ting's mind during the development

- Maybe Duke VCM manager could provide containers or droplets for lightweight services such as Flask or Django, so that students can just run really small services without wasting the electricity of maintaining a full virtual machine.
- SSL certificate is automatically set when a VM is assigned.
- Increase the default reservation limitation for VM. I don't know if it's unique to me, but I can only reserve 1 VM from my account, which is a bit annoying when developing a secure web service and a rootkit simultaneously.

8 Conclusion

From the emergence of cloud computing concept, and with more than a half century's evolvement, cloud computing is now ubiquitous in our daily life. In fact, many companies run their service on Amazon AWS, Microsoft Azure or other cloud service providers. The cloud service helps them reduce their IT cost and accelerate their development process.

In this semester, Binwu gave an introduction about the virtualization, which is the foundation of cloud computing and how the cloud service providers implement their service. Ting elaborated how mobile application and cloud service could combine to provide better service. We also learned many other new topics about cloud computing such as NoSQL, content distribution network and google file system, etc.

This Django web file service application is a SaaS (software as a service) by leveraging Duke private-cloud service. Through the development process of this project, we not only enhanced our understanding of cloud computing concepts, but also learned the practical skills to develop a web application by leverage Django framework.