CS 492 Senior Design Project II
Low Level Design Report

Project short-name: Tattoo'd

Ali Onur Geven
Anıl Sert
Beril Başak Tukaç
Nazlı Özge Uçan
Furkan Usta

Supervisor: Ercüment Çiçek

Jury Members: Selim Aksoy, Uğur Güdükbay

tattood.github.io

Table of Contents

1. Introduction

Mobile phones entered our lives a few years ago but it has quickly become the essential part of our daily routines. They offer their users increasing productivity, make budget management, entertainment with the games or utility tools with providing the applications created by software engineers. People are generally using these applications to solve some problems in their lives. The problems and difficulty in deciding which tattoo to make is the problem that we want to solve with our mobile application.

Tattoo'd is a mobile application that is working on smart phones and the purpose of this application is to help people that are in the process of considering to make a tattoo to their body with showing them how their demanded tattoo will look like at their body.

Users need to register to the system with their Google accounts and they can fully experience the application after this registration. People can share tattoos within the application and search specific tags to find tattoos they are looking for or browse other public tattoos shared from the community. After selecting the tattoo they like they can use phone's camera to place tattoo to the body part and look or take a photo of it to share from social media platforms.

1.1. Object design trade-offs

Memory Usage vs Data Usage
This project involves image processing algorithms and we had to choose between either performing these algorithms in the client-side or server-side. We had to choose between high battery consumption or high mobile data usage. As a consequence we chose to perform on the phone and we had to compromise on the data usage since we can come up to solution of that problem more easy.

Complexity vs Usability
Tattoo'd is an entertainment application for people who need to have fun and try the best tattoos before they get the tattoo on their body. Because of that, Tattoo'd has a simple yet user friendly GUI[1] for the favor of usability. A simple GUI[1] enables us to reach a wide range of users because Tattoo'd does not require complex interactions. On the other hand, our application have a very complex background functionality which may cause complexity for the UI. Our priority is to provide a simple yet functional UI.

Compatibility vs Extensibility
Our target is to develop it for wide range of users. Therefore, the system design of the application should be compatible with the different versions of the Android Operating System. Unfortunately, extensibility of system conflicts with compatibility due to that old version of Android operating systems like version 2.2 does not provide flexibility for the programmer unlike Android version 4.0. Other operating systems such as iOS, Windows, also take time to implement and because we have such a limited time we select compatibility over extensibility.

Space vs. Response time
Tattoo'd should be fast for the favor of the users. Since we have complex algorithms in terms of runtime which are executed at the backend, our system favors speed over space such as using servers to execute the algorithms but using internet connection. But we also have space management algorithms to minimize the internet connection.

Space vs. Cost
The program needs external services for storing tattoo and user informations and those services require additional cost. On the other hand considering this project is not funded and it is developed by students, Tattoo'd tries to minimize the cost. We have a limited cost using space management techniques and for the expected number of users in the upcoming times, we favor space over cost.

1.2. Interface documentation guidelines

We used javadoc style for the internal code documentation. Overall structure of the documentation is as follows [2]:

1. The first paragraph is a description of the method documented.
2. Following the description are a varying number of descriptive tags, signifying:
    1. The parameters of the method (@param)
    2. What the method returns (@return)
    3. Any exceptions the method may throw (@throws)
    4. Other less-common tags such as @see (a "see also" tag)

In Section 3 class interfaces are explained as in the following format:

Class Name	Name of the class
Class Description	The description of the class
Attributes	The related attributes of the class
Operations	The operations that can be performed on the objects of the class

For server-side documentation whether the method is POST or GET method is given after the method name within brackets.

1.3. Engineering standards

Engineering standards are the documents that indicate the specifications, characteristics and technical details of the system and also cover the development process of the application. The

purpose is to ensure the application is consistent. We are using different software development environments such as XML[3], Firebase[4], Java and so on.

Our project Tattoo'd aims to reach various people in different personal selections so we need to standardize the artistic and technical challenges within application development. Therefore, we followed Java Language Specification standards[5]. Besides the ethical concerns that we followed, for the technical part of the application, we drew Tattoo'd subsystem decomposition structure according to UML[6] standardizations. By this means, we ensure that classes and objects of Tattoo'd are created based on the Object Oriented Programming[7] specifications.

1.4. Definitions, acronyms, and abbreviations

UML: Unified Modeling Language [6]

SQL: Structured Query Language

GPU: Graphical Processing Unit, a part of computer which is responsible for rendering graphical objects. [8]

Python: Object-oriented extensive programming language (.py)

GUI: Graphical User Interface [1]

XML: Extensible Markup Language [3]

Firebase: Firebase is a mobile and web application platform with tools and infrastructure designed to help developers build high-quality apps. [9]

JSON: JavaScript Object Notation [10]

REST: Representational state transfer. A way of providing interoperability between computer systems on the Internet [11]

2. Packages

2.1 Java Package

Java packages contains necessary model, view and controller files for the android application. Due to Android Frameworks nature most of our view classes are, in fact, XML files.

Controller classes are either Fragment or Activity classes and they extend the corresponding base class from Android SDK. Each controller class is interacts with a view.

Model classes represent the data structures that we have in the server.

2.1.1 Model

User.java
Tatto.java
TatooImage.java

2.1.2 View

CameraPreview.java
activity_login.xml
activity_register.xml
activity_main.xml
fragment_camera.xml
fragment_discovery.xml
fragment_profile.xml

2.1.3 Controller

LoginController.java
CameraFragment.java
DiscoveryFragment.java
LoginActivity.java
ARCamera.java
MainActivity.java
ProfileFragment.java
RegisterActivity.java

SplashScreen.java
Server.java
TatooRecyclerViewAdapter.java
GoogleAuthentication.java

2.2. Server Packages

Server packages maintain the communication between Android-Client, Database and Image Processing packages, they are not allowed to interact with each other without using the server. All the interactions on the server are defined as a REST API, and requires a token.

database.py
server.py
config.py

2.2.1 Image Processing Package

This package is stored in the server-side and executed upon a request from client. It performs tag extraction for uploaded tattoo designs

extract.py

2.2.2 Configuration Package

Configuration files consist of credential information for database connection and API key for the authorization system.

.config

3. Class Interfaces

3.1 Client Side

For brevity all the accessor methods are ignored.

User
This class represents the User class in database and holds related information
-String email -String username -String token -ArrayList<Tattoo> liked -ArrayList<Tattoo> uploaded_private -ArrayList<Tattoo> uploaded_public -ArrayList<Users> followed -ArrayList<Users> followers
+ArrayList<Tattoo> getLiked(Response.Listener<JSONObject>) +ArrayList<Tattoo> getPrivate(Response.Listener<JSONObject>) +ArrayList<Tattoo> getPublic(Response.Listener<JSONObject>) +ArrayList<User> getFollowed(Response.Listener<JSONObject>) +ArrayList<User> getFollowers(Response.Listener<JSONObject>)

Tattoo
This class represents the Tattoo class in database and holds the necessary information
-int owner_id -ArrayList<String> tags
+Tattoo(id) +ArrayList<String> getTags()

TattooImage
This class holds the uploaded image and sends to the server for processing
-int owner_id -Image image
+Tattoo createTattoo(int, Image)

CameraFragment
Controller class for Camera View
-Camera mCamera -CameraPreview mPreview
+void onCreate(Bundle) +void onCreateView(LayoutInflate, ViewGroup, Bundle) +void onPause() +void onResume() +void onAttach(Context) +void onDetach() +void onDestroyView() +CameraFragment newInstance()

DiscoveryFragment
Controller class for Discovery Page
-OnListFragmentInteractionListener listener
+DiscoveryFragment newInstance() +void onCreate(Bundle) +void onCreateView(LayoutInflate, ViewGroup, Bundle) +void onPause() +void onResume() +void onAttach(Context) +void onDetach() +void onClick(View)

LoginActivity
Controller class for Login Action
-String PREFS_NAME -GoogleApiClient mGoogleApiClient -int RC_SIGN_ING -FirebaseAuth mAauth -FirebaseAuth.AuthStateListener mAuthListener
#void onCreate(Bundle) +void onActivityResult(int, int, Intent) +void onConnectionFailed(ConnrectionResult) +void onPause() +void onResume()

MainActivity
Main Controller class that maintains the interaction between User, Discovery and Camera Views
-SectionsPagerAdapter mSectionsPagerAdapter -ViewPager mViewPager -String token
#void onCreate(Bundle) +boolean onCreateOptionsMenu(Menu) +boolean onOptionsItemSelected(MenuItem)

SectionsPagerAdapter
Controller class for Tab Views
+SectionsPagerAdapter(FragmentManaager) +Fragment getItem(int) +int getCount() +CharSequence getPageTitle(int)

RegisterActivity
Controller class for register view
#void onclick(Bundle)

ProfileFragment
Controller class for Profile View
-User user -String token -RecyclerView userLiked -RecyclerView userPublic -RecyclerView userPrivate

+ProfileFragment newInstance(Bundle) +void onCreateView(LayoutInflater, ViewGroup, Bundle) +void onAttach(Context) +void onDetach()

SplashScreen
Controller class for Splash Screen
-String PREFS_NAME
#void onCreate(Bundle)

ARCamera
Controls the AR Camera and makes necessary computations
-Tattoo mTattoo -Camera mCamera -Coordinate referencePoint
+void start() +void stop() +void adjust() +void calibrate()

TattooRecyclerViewAdapter
Controller class for RecyclerView that shows Tattoos
-List<Tattoo> mValues -OnListFragmentInteractionListener mListener -Context mContext -RecyclerView mRecyclerView
+TattooRecyclerViewAdapter(List<Tattoo>, OnListFragmentInteractionListener, Context, RecyclerView) +ViewHolder onCreateViewHolder(ViewGroup, int) +void onBindViewHolder(ViewGroup, int) +int getItemCount() +void onClick(View)

Server
This class provides the necessary interface to the server
-String host +enum UserRequest{Followed, Follower} +enum TattooRequest{Public, Private, Liked}
+boolean isAvailable() +void signIn(context, token, email, callback, error_handler) +void register(context, token, email, callback, error_handler) +void getUserList(context, token, UserRequest, callback, error_handler) +void getTattooList(context, token, TattooRequest, callback, error_handler)

GoogleAuthentication
This class makes the necessary calls for Google Authentication
+void signinGoogle(key) +void signinFirebase(key)

3.2 Server Side

server.py
Implements the REST API for client-side to use
-Database db
+User login(email, token) [POST] +User register(username, email, token) [POST] +int logout(token) [POST] +List<Tattoo> user-likes(token) [GET] +List<Tatoo> user-uploaded(token, boolean private?) [GET] +int like(token, tattoo-id) [POST] +int unlike(token, tattoo-id) [POST] +Tattoo tattoo(tattoo-id) [GET] +[Tattoo|int] tattoo-update(token, tattoo-id, private?) [POST] +[Tattoo|int] tattoo-upload(token, private?, image) [POST] +int tattoo-delete(token, tattoo-id) [POST]

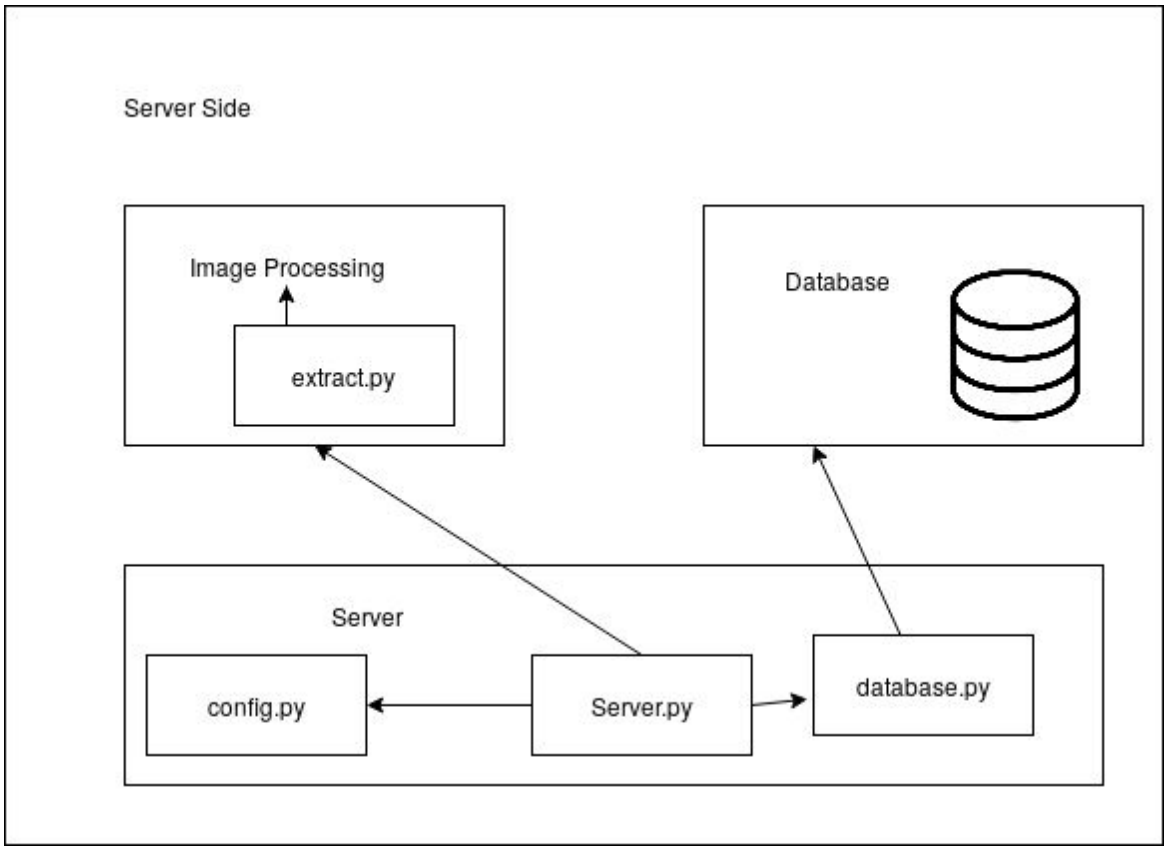
config.py
Holds the configuration parameters and gives upon request
-String WTF_CSRF_ENABLED -String SECRET_KEY -String SQLALCHEMY_DATABASE_URI -Boolean SQLALCHEMY_TRACK_MODIFICATIONS
+List getConfig() +Object getConfig(String)

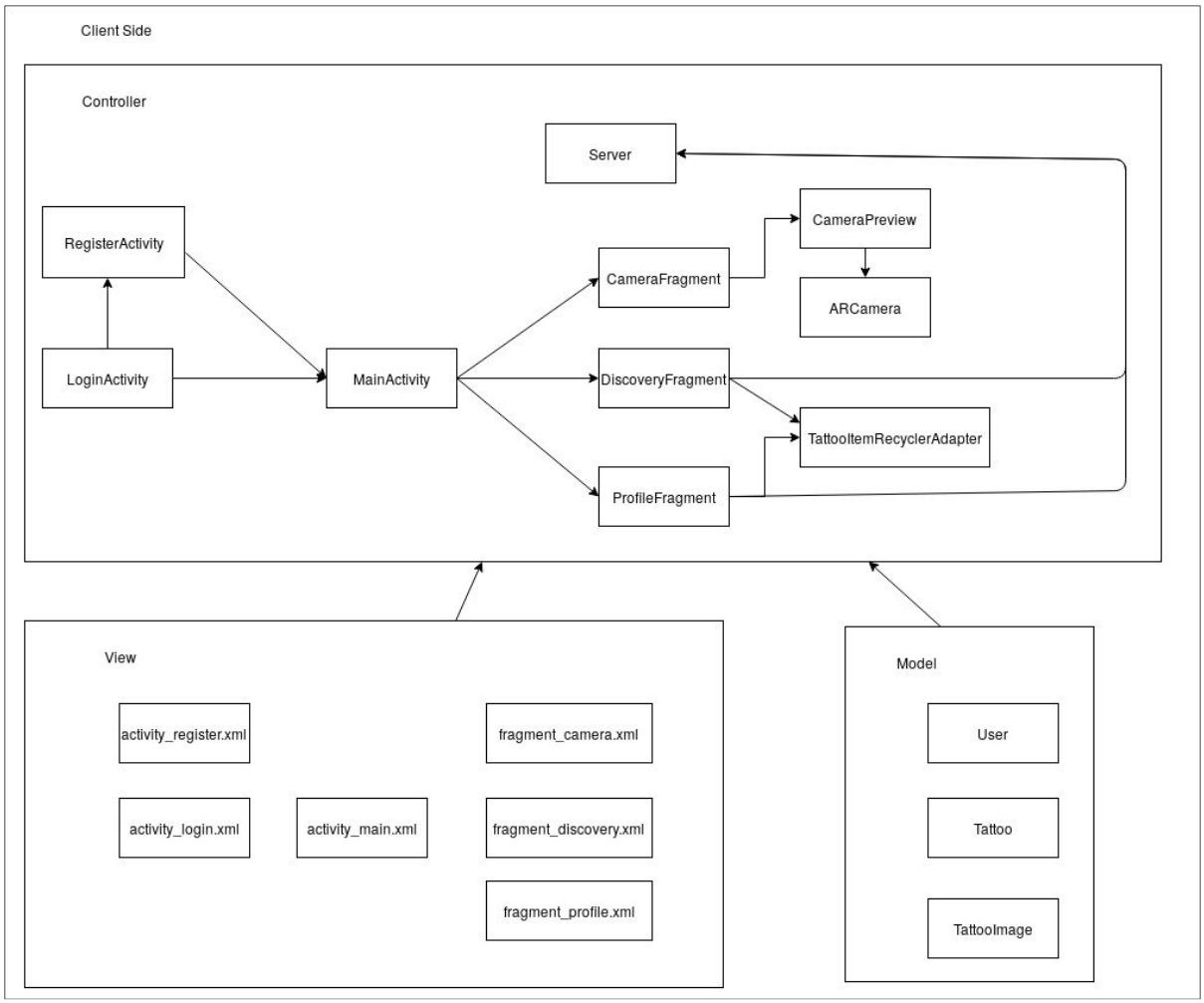
db.py
Represents the models defined in the database as classes in Python
-Database db
+User(username, email) +Tattoo(owner_id, private?, image) +Likes(user_id, tattoo_id) +Tag(description)

extract.py
Extracts the objects from the given image and converts them to MetaTattoo object
-Object img
+List<MetaTattoo> extract(image)

4. Class Relationship Chart

Below are the relationships between packages for server and client sides. Client side performs requests to the server using Server.java and any incoming request will be handled in server.py in the server-side




Client Side
Controller
Server
RegisterActivity
CameraPreview
CameraFragment
ARCamera
LoginActivity
MainActivity
DiscoveryFragment
TattooItemRecyclerAdapter
ProfileFragment
View
activity_register.xml
fragment_camera.xml
activity_login.xml
activity_main.xml
fragment_discovery.xml
fragment_profile.xml
Model
User
Tattoo
TattooImage

5. Glossary

Tattoo: A body modification made by inserting ink to the skin. Application can show virtual tattoos on the body of user using camera and augmented reality.

Discover: Applications browse section that users can search tattoos or look other tattoos that are shared by other people.

Public/Private: Tattoos can be both public and private. Public means everyone can see your uploaded tattoos and private means you are hiding the tattoos that you are uploaded to the application.

Like: Users can like the tattoos in the system.

Tag: People can give tags to the tattoos and other users search these tags to find tattoos with the searched tags.

Search: Users can search and find tattoos they are interested with using tags.

Profile: Every user will have a profile and they can create their profiles with registering to the system. Users can see their uploaded and liked tattoos in their profiles.

Upload: Users can upload images to the system and create tattoo using that image. After uploading they can share it publicly or don't share with the other users of the system.

Share: Users can share the photo which is taken during the tattoo demonstration on the social media accounts.

6. References

[1] GUI - Graphical User Interface

[2] Javadoc - Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Javadoc

[3] XML – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/XML

[4] Firebase [Online]. Available: https://firebase.google.com/

[5] Java Language Specification Standards - Oracle [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html

[6]UML – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Unified_Modeling_Language

[7] OOP – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Object-oriented_programming

[8] GPU – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Graphics_processing_unit

[9] Firebase – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Firebase

[10] JSON – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/JSON

[11] REST -Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer