4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang implementasi sistem sesuai dengan analisa dan desain sistem. Implementasi sistem meliputi implementasi login, proses pembuatan konten, *manage* desain *template*, manage kategori, manage sub-kategori, *add photo*, *post now*, dan *scheduled post*. Hubungan antara segmentasi program dengan proses yang dilakukan terdapat pada Tabel 4.1. Selain itu terdapat juga hasil implementasi *User Interface* pada Tabel 4.2.

Tabel 4.1.

Daftar Hubungan *Diagram* dan Segmen Program

Proses	**Segmen Program**	***Activity Diagram & Flowchart***
Request API untuk *upload* gambar	Segmen Program 4.1.	Gambar 3.4.
Color clustering gambar dari *user* menggunakan K-Means Clustering	Segmen Program 4.2.	Gambar 3.4., & Gambar 3.13.
Color clustering desain *template* dan perhitungan jarak warna menggunakan K-Means Clustering dan Euclidean Distance	Segmen Program 4.3.	Gambar 3.4., & Gambar 3.14.
Mendapatkan koordinat letak gambar atau/dan teks dari desain template yang terpilih	Segmen Program 4.4.	Gambar 3.4.
Menampilkan hasil pembuatan konten	Segmen Program 4.5.	Gambar 3.4.
Request API untuk	Segmen Program 4.6.	Gambar 3.6.

mendapatkan seluruh desain *template*		
Request API untuk mendapatkan seluruh kategori desain *template*	Segmen Program 4.7.	Gambar 3.7.
Request API untuk mendapatkan seluruh sub-kategori desain *template*	Segmen Program 4.8.	Gambar 3.8.
Add Photo	Segmen Program 4.9.	-
Post Now	Segmen Program 4.10.	Gambar 3.9.
Scheduled Post	Segmen Program 4.11.	Gambar 3.9.

4.1. Proses Pembuatan Konten

Proses ini dilakukan untuk membuat konten secara berdasarkan *parameter* jenis *output*, kategori, dan sub-kategori yang dipilih *user*. Setiap desain *template* telah ditentukan dari awal jenis *output*, kategori, dan sub-kategorinya. *Parameter* yang di *input* oleh *user* menentukan jenis desain *template* yang akan di-*query* sehingga desain *template* yang dihitung jarak warnanya telah dibatasi.

Berawal dari *user* memilih gambar yang ingin dijadikan konten, aplikasi akan mengunggah gambar tersebut ke dalam *server* dan *server* mengembalikan *filename* kepada *client* yaitu aplikasi.

Segmen Program 4.1. *Request API* untuk *Upload* Gambar dan Mendapatkan *Filename* Desain *Template* yang Tepat

```
val rq = Volley.newRequestQueue(context)
val url = serverUrl + "skripsi/api/upload.php"
val stringRequest: StringRequest =
    object : StringRequest(
        Method.POST,
        url, Response.Listener { response ->
            try {
```

```
                val obj = JSONObject(response)
                val msg = obj.get("msg").toString()
                val objFilename = obj.get("filename").toString()
                userFile = objFilename
                Toast.makeText(
                    context,
                    msg, Toast.LENGTH_SHORT
                ).show()

                val queueColorSegmentation =
Volley.newRequestQueue(context)
                val urlColorSegmentation = serverUrl +
"skripsi/api/colorclustering.php"
                val srColorSegmentation: StringRequest =
                    object : StringRequest(
                        Method.POST, urlColorSegmentation,
                        Response.Listener<String> { response2 ->
                            try {
                                val objColorSegmentation =
JSONObject(response2)
                                colorRed =
objColorSegmentation.get("red").toString()
                                colorGreen =
objColorSegmentation.get("green").toString()
                                colorBlue =
objColorSegmentation.get("blue").toString()
                                val colorRed2 =
objColorSegmentation.get("red2").toString()
                                val colorGreen2 =
objColorSegmentation.get("green2").toString()
                                val colorBlue2 =
objColorSegmentation.get("blue2").toString()

                                val rq2 =
Volley.newRequestQueue(context)
                                val url2 = serverUrl +
"skripsi/api/colorclustering2.php"
                                val stringRequest2: StringRequest =
                                    object : StringRequest(
                                        Method.POST,
                                        url2, Response.Listener
{ response3 ->
                                            try {
```

```
                                                    loading.stopLoading()
                                                    val obj =
JSONObject(response3)
                                                    filename =
obj.get("filename").toString()
                                                    val intent = Intent(
                                                        context,

DesignResultActivity::class.java
                                                    )
                                                    intent.putExtra("text",
etTitle.text.toString())
                                                    val tempArray:
MutableList<String> = arrayListOf()
                                                    for(i in 0..totalText -
1){

tempArray.add(arrDescription[i].text.toString())
                                                    }

intent.putStringArrayListExtra("arr_description", ArrayList(tempArray))

requireContext().startActivity(intent)

                                                } catch (e: JSONException)
{
                                                    loading.stopLoading()
                                                    Log.v("WillyCheck",
e.toString())

                                                }
                                            }, Response.ErrorListener
{ error ->
                                                loading.stopLoading()
                                                Log.v("WillyCheck", "Error
[$error]")
                                                Toast.makeText(
                                                    context,
                                                    "Cannot connect to
server",
                                                    Toast.LENGTH_LONG
                                                )
                                                    .show()
```

```
                                        }) {
                                        override fun getParams():
Map<String, String> {
                                            val params:
MutableMap<String, String> =
                                                HashMap()
                                            params["jenisOutput"] =
DesignFragment.jenisOutput
                                            params["kategori"] =
DesignFragment.kategori
                                            params["subKategori"] =

DesignFragment.subKategori
                                            params["has_children"] =
DesignFragment.hasChildren
                                            params["red"] = colorRed
                                            params["green"] =
colorGreen
                                            params["blue"] = colorBlue
                                            params["red2"] = colorRed2
                                            params["green2"] =
colorGreen2
                                            params["blue2"] =
colorBlue2

if(etTitle.text.toString() != ""){
                                                params["has_text"] =
"1"
                                            }else{
                                                params["has_text"] =
"0"
                                            }
                                            Log.v("WillyCheck",
"params:" + params.toString())
                                            return params
                                        }
                                    }
                                stringRequest2.setRetryPolicy(
                                    DefaultRetryPolicy(
                                        60000,

DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
```

```
DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
                                    )
                                )

                                rq2.add(stringRequest2)

                                // to DesignResultFragment
                            } catch (e: JSONException) {
                                loading.stopLoading()
                                //Log.v("WillyCheck", e.toString())
                                Toast.makeText(
                                    context,
                                    "Error while loading data!",
                                    Toast.LENGTH_LONG
                                ).show()
                            }
                        }, Response.ErrorListener { error ->
                            Log.v("ERROR", "Error [$error]")
                            Toast.makeText(
                                context,
                                "Cannot connect to server",
Toast.LENGTH_LONG
                            )
                                .show()
                        }) {
                        override fun getParams(): Map<String, String> {
                            val params: MutableMap<String, String> =
                                HashMap()
                            params["filename"] = objFilename
                            Log.v("WillyCheck", "params: " + params)
                            return params
                        }
                    }
                srColorSegmentation.setRetryPolicy(
                    DefaultRetryPolicy(
                        60000,
                        DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
                        DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
                    )
                )
                queueColorSegmentation.add(srColorSegmentation)
            } catch (e: JSONException) {
                loading.stopLoading()
```

```
                //Log.v("WillyCheck", e.toString())
                Toast.makeText(
                    context,
                    "Error while loading data!",
                    Toast.LENGTH_LONG
                ).show()
            }
        }, Response.ErrorListener { error ->
            loading.stopLoading()
            Log.v("ERROR", "Error [$error]")
            Toast.makeText(
                context,
                "Cannot connect to server", Toast.LENGTH_LONG
            )
                .show()
        }) {
        override fun getParams(): Map<String, String> {
            val params: MutableMap<String, String> =
                HashMap()
            //Log.v("WillyCheck", myURI)
            params["fileFoto"] = myURI.toString()
            return params
        }
    }
stringRequest.setRetryPolicy(
    DefaultRetryPolicy(
        60000,
        DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
        DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
    )
)
rq.add(stringRequest)
```

Setelah berhasil *upload* gambar dari *user* ke *server*, *filename* didapatkan dan dijadikan parameter pada *script* python yang dijalankan untuk mendapatkan dua pasang warna RGB paling dominan dari gambar tersebut. *Script* python yang dijalankan adalah *script* untuk melakukan *color clustering* yang menghasilkan warna RGB paling dominan dari suatu gambar.

Segmen Program 4.2. *Color Clustering* Gambar dari *User*

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import Kmeans
import sys
import json
from kneed import KneeLocator

#load image
filename='files/'+sys.argv[1]+'.jpeg'
img=cv2.imread(filename)
plt.imshow(img)

img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)

img=cv2.resize(img, None, fx=0.2, fy=0.2)
img=img.reshape((img.shape[1]*img.shape[0],3))

#elbow method
md=[]
for i in range(1,10):
  kmeans=Kmeans(n_clusters=i)
  kmeans.fit(img)
  o=kmeans.inertia_
  md.append(o)

x = range(1, len(md)+1)
kn = KneeLocator(x, md, curve='convex', direction='decreasing')
optimalK=kn.knee

#most dominant colors
kmeans=Kmeans(n_clusters=optimalK)
s=kmeans.fit(img)
labels=kmeans.labels_
labels=list(labels)

centroid=kmeans.cluster_centers_
```

```
#percentage
percent=[]
for i in range(len(centroid)):
  j=labels.count(i)
  j=j/(len(labels))
  percent.append(j)

mx=max(percent[0],percent[1])
secondmax=min(percent[0],percent[1])
firstIndex=0
secondIndex=0
if(mx == percent[1]):
    firstIndex=1
if(secondmax == percent[1]):
    secondIndex=1
n =len(percent)
for i in range(2,n):
    if percent[i]>mx:
        secondmax=mx
        mx=percent[i]
        secondIndex=firstIndex
        firstIndex=i
    elif percent[i]>secondmax and \
        mx != percent[i]:
        secondIndex=i
        secondmax=percent[i]

red=centroid[firstIndex][0]
green=centroid[firstIndex][1]
blue=centroid[firstIndex][2]

red2=centroid[secondIndex][0]
green2=centroid[secondIndex][1]
blue2=centroid[secondIndex][2]

result = {
    "red": red,
    "green": green,
    "blue": blue,
    "red2": red2,
    "green2": green2,
    "blue2": blue2,
    }
```

```
print(json.dumps(result))
```

Warna paling dominan dikembalikan ke aplikasi dalam bentuk JSON. Lalu aplikasi akan melakukan *request API* lagi pada *server* dengan mengirimkan *filename* beserta jenis output, kategori, dan sub-kategori yang dipilih *user* sebelumnya untuk menjalankan *script color clustering* dan perhitungan jarak warna antara warna gambar dari *user* dengan warna gambar desain template yang sesuai dengan jenis *output*, kategori, dan sub-kategori yang ada pada *server.*

Script tersebut membuka dan memanipulasi *file* gambar pada *server* menggunakan *library* OpenCV, matplotlib, numpy, dan pandas. *K-means* digunakan untuk mendapatkan warna paling dominan menggunakan K-Means Clustering. *Script* ini mengembalikan sebuah *response* dengan tipe JSON yang berisi enam data yaitu warna RGB. Lalu aplikasi akan melakukan *request API* lagi untuk memanggil *script* python pada *server* dan memasukkan enam buah data tersebut menjadi *parameter*-nya.

Segmen Program 4.3. *Color Clustering* Desain *Template* dan Perhitungan Jarak Warna

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import Kmeans
import sys
import json
import pymysql
import math
from kneed import KneeLocator

db = pymysql.connect(host="localhost", user="c14170010",
password="TOS2019", db="c14170010",charset='utf8mb4')
cursor = db.cursor()

arrFilename = []

rgb = sys.argv[5].split(",")
rgb2 = sys.argv[6].split(",")
hasChildren = str(sys.argv[8])
```

```
if(hasChildren == "1"):
    # sql = "SELECT * FROM `desain_template` WHERE id_jenis_output = " +
str(sys.argv[2]) + " AND id_kategori = " + str(sys.argv[3]) + " AND
id_sub_kategori = " + str(sys.argv[4]) + " AND has_teks = " +
str(sys.argv[7])+ " AND has_gambar = " + str(sys.argv[9])
    sql = "SELECT * FROM `desain_template` WHERE id_jenis_output = " +
str(sys.argv[2]) + " AND id_kategori = " + str(sys.argv[3]) + " AND
id_sub_kategori = " + str(sys.argv[4]) + " AND has_gambar = " +
str(sys.argv[9])
else:
    sql = "SELECT * FROM `desain_template` WHERE id_jenis_output = " +
str(sys.argv[2]) + " AND id_kategori = " + str(sys.argv[3]) + " AND
has_gambar = " + str(sys.argv[9])
try:
    cursor.execute(sql)
    results = cursor.fetchall()
    for row in results:
        arrFilename.append(row[4])
except:
    print("Error: unable to fetch data")

res = 999
fileCounter = 0

for z in range(int(str(sys.argv[1]))):
    #load image
    filename='templates/'+str(arrFilename[z])

    img=cv2.imread(filename)
    plt.imshow(img)

    img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    plt.imshow(img)

    img=cv2.resize(img, None, fx=0.1, fy=0.1)
    img=img.reshape((img.shape[1]*img.shape[0],3))

    #elbow method
    md=[]
    for i in range(1,10):
      kmeans=Kmeans(n_clusters=i)
      kmeans.fit(img)
      o=kmeans.inertia_
```

```
    md.append(o)

x = range(1, len(md)+1)
kn = KneeLocator(x, md, curve='convex', direction='decreasing')
optimalK=kn.knee

#most dominant colors
kmeans=Kmeans(n_clusters=optimalK)
s=kmeans.fit(img)

labels=kmeans.labels_
labels=list(labels)

centroid=kmeans.cluster_centers_

#percentage of dominant colors
percent=[]
for i in range(len(centroid)):
  j=labels.count(i)
  j=j/(len(labels))
  percent.append(j)

#RGB of most dominant color
mx=max(percent[0],percent[1])
secondmax=min(percent[0],percent[1])
firstIndex=0
secondIndex=0
if(mx == percent[1]):
    firstIndex=1
if(secondmax == percent[1]):
    secondIndex=1
n =len(percent)
for i in range(2,n):
    if percent[i]>mx:
        secondmax=mx
        mx=percent[i]
        secondIndex=firstIndex
        firstIndex=i
    elif percent[i]>secondmax and \
        mx != percent[i]:
        secondIndex=i
        secondmax=percent[i]
```

```
       ### template most dominant RGB and second dominant RGB
    red=centroid[firstIndex][0]
    green=centroid[firstIndex][1]
    blue=centroid[firstIndex][2]

    red2=centroid[secondIndex][0]
    green2=centroid[secondIndex][1]
    blue2=centroid[secondIndex][2]

    userMainColor =
np.array((float(rgb[0]),float(rgb[1]),float(rgb[2])))
    templateMainColor = np.array((red,green,blue))
    diff = np.linalg.norm(userMainColor - templateMainColor)

    if diff < res:
        res = diff
        fileCounter = z

print(arrFilename[fileCounter])
```

Setelah *script* untuk menghitung jarak warna antara gambar dari *user* dengan berbagai desain template, script akan mengembalikan *filename* desain *template* yang jarak warnanya paling dekat. Lalu *filename* tersebut akan digunakan aplikasi untuk memuat gambar desain *template* yang ada di *server*. Kemudian aplikasi melakukan *request API* untuk mendapatkaan koordinat letak gambar atau/dan teks pada desain *template* dan dikembalikan ke aplikasi dalam bentuk JSON.

Segmen Program 4.4. Mendapatkan Koordinat Letak Gambar atau/dan Teks dari Desain *Template* yang Terpilih

```
<?php

include("connect.php");
$hasGambar = 0;
$hasTeks = 0;

$fileName = "'" . $_POST["filenameGambar"] . "'";

$fileName = trim(preg_replace('/\s+/', ' ', $fileName));
```

```
$sql_desain_template = "SELECT * FROM desain_template WHERE
filename_desain_template=" . $fileName;
$res_desain_template = $conn->query($sql_desain_template);
$row_desain_template = $res_desain_template->fetch_assoc();
$id_desain_template = $row_desain_template["id_desain_template"];
$sql = "SELECT * FROM `koordinat` WHERE id_desain_template =
$id_desain_template AND gambar_atau_teks = 'gambar'";

$res = mysqli_query($conn, $sql);
$koordinatGambar = array();
$koordinatTeks = array();

if(mysqli_num_rows($res) > 0){
	while($row = mysqli_fetch_assoc($res)){
		$koordinatGambar[] = array("top_left_x" => (float)
$row["top_left_x"],
		"top_left_y" => (float) $row["top_left_y"],
		"top_right_x" => (float) $row["top_right_x"],
		"top_right_y" => (float) $row["top_right_y"],
		"bottom_left_x" => (float) $row["bottom_left_x"],
		"bottom_left_y" => (float) $row["bottom_left_y"],
		"bottom_right_x" => (float) $row["bottom_right_x"],
		"bottom_right_y" => (float) $row["bottom_right_y"],
	);
	}
}

$sql_desain_template = "SELECT * FROM desain_template WHERE
filename_desain_template=" . $fileName;
$res_desain_template = $conn->query($sql_desain_template);
$row_desain_template = $res_desain_template->fetch_assoc();
$id_desain_template = $row_desain_template["id_desain_template"];
$sql = "SELECT * FROM `koordinat` WHERE id_desain_template =
$id_desain_template AND gambar_atau_teks = 'teks'";

$res = mysqli_query($conn, $sql);
if(mysqli_num_rows($res) > 0){
	while($row = mysqli_fetch_assoc($res)){
		$koordinatTeks[] = array("top_left_x" => (float)
$row["top_left_x"],
		"top_left_y" => (float) $row["top_left_y"],
		"top_right_x" => (float) $row["top_right_x"],
```

```
            "top_right_y" => (float) $row["top_right_y"],
            "bottom_left_x" => (float) $row["bottom_left_x"],
            "bottom_left_y" => (float) $row["bottom_left_y"],
            "bottom_right_x" => (float) $row["bottom_right_x"],
            "bottom_right_y" => (float) $row["bottom_right_y"],
      );
      }
}
$response = array("msg" => "Berhasil",
                  "koordinat_gambar" => $koordinatGambar,
                  "koordinat_teks" => $koordinatTeks
            );
echo json_encode($response);
?>
```

Lalu gambar dari *user* dan gambar desain *template* digabungkan berdasarkan koordinat yang mengatur letak gambar atau/dan teks. Ukuran layar *device* yang digunakan *user* berbeda-beda sehingga perlu untuk menyesuaikan kembali ukuran gambar hasil pembuatan konten sesuai dengan ukuran layar *device*.

Segmen Program 4.5. Menampilkan Hasil Pembuatan Konten

```
val templateUrl = serverUrl + "skripsi/api/templates/" +
Design2Fragment.filename
        val displayMetrics = DisplayMetrics()
        windowManager.defaultDisplay.getMetrics(displayMetrics)
        val screenHeight: Int = displayMetrics.heightPixels
        val screenWidth: Int = displayMetrics.widthPixels
        if(DesignFragment.jenisOutput == "Instagram Story"){
            val widthHeightRatio = 1920f / 1080f
            val tempHeight = screenWidth * widthHeightRatio
            Picasso.get().load(templateUrl).resize(screenWidth,
tempHeight.toInt()).into(ivTemplate)
        }
        else if(DesignFragment.jenisOutput == "Instagram Post"){
            Picasso.get().load(templateUrl).resize(screenWidth,
screenWidth).into(ivTemplate)
        }

        val userImageUrl = serverUrl + "skripsi/api/files/" + userFile
```

```
        val rq = Volley.newRequestQueue(this)
        val url = serverUrl + "skripsi/api/getCoordinate.php"
        val sr: StringRequest =
            @SuppressLint("ResourceType")
            object : StringRequest(
                Method.POST,
                url, Response.Listener { response ->
                    try {
                        val obj = JSONObject(response)
                        val msg = obj.get("msg").toString()
                        val arrKoordinatGambar =
JSONArray(obj.get("koordinat_gambar").toString())
                        val arrKoordinatTeks =
JSONArray(obj.get("koordinat_teks").toString())

                        val displayMetrics: DisplayMetrics =
                            resources.displayMetrics
                        val templateToDp = (1080 - 0.5f) /
displayMetrics.density
                        val dpHeight =
                            displayMetrics.heightPixels /
displayMetrics.density
                        val dpWidth =
                            displayMetrics.widthPixels /
displayMetrics.density
                        val ratio = dpWidth / templateToDp

                        for(i in 0..arrKoordinatGambar.length() - 1){
                            val gambarObj =
JSONObject(arrKoordinatGambar[i].toString())
                            // Passing coordinates
                            val xTopLeft =
gambarObj.get("top_left_x").toString().toFloat()
                            val yTopLeft =
gambarObj.get("top_left_y").toString().toFloat()
                            val xTopRight =
gambarObj.get("top_right_x").toString().toFloat()
                            val yTopRight =
gambarObj.get("top_right_y").toString().toFloat()
                            val yBottomRight =
gambarObj.get("bottom_right_y").toString().toFloat()
```

```
                            val xTopLeftToDp = xTopLeft * ratio
                            val yTopLeftToDp = yTopLeft * ratio
                            val templateLeft = ivTemplate.left
                            val templateTop = ivTemplate.top

                            ivImage.animate().x(templateLeft +
xTopLeftToDp)
                            ivImage.animate().y(templateTop +
yTopLeftToDp)

                            val width = (xTopRight - xTopLeft) * ratio
                            val height = (yBottomRight - yTopRight) *
ratio

Picasso.get().load(userImageUrl).resize(width.toInt(),
height.toInt()).centerCrop().into(ivImage)
                        }
                        val teks = intent.getStringExtra("text")
                        val arrDescription =
intent.getStringArrayListExtra("arr_description")
                        for(i in 0..arrKoordinatTeks.length() - 1){
                            val teksObj =
JSONObject(arrKoordinatTeks[i].toString())
                            // Passing coordinates
                            val tvTeks = TextView(this)
                            val xTopLeft =
teksObj.get("top_left_x").toString().toFloat()
                            val yTopLeft =
teksObj.get("top_left_y").toString().toFloat()
                            val xTopRight =
teksObj.get("top_right_x").toString().toFloat()
                            val yBottomLeft =
teksObj.get("bottom_left_y").toString().toFloat()
                            val scale: Float =
getResources().getDisplayMetrics().density
                            val width = (xTopRight - xTopLeft) * ratio
                            val height = (yBottomLeft - yTopLeft) *
ratio
                            val pixelsWidth = (width).toInt()
                            val pixelsHeight = (height).toInt()
                            val layout =
RelativeLayout.LayoutParams(width.toInt(), height.toInt())
                            tvTeks.layoutParams = layout
```

```
                            tvTeks.textAlignment =
View.TEXT_ALIGNMENT_CENTER
                            tvTeks.gravity = Gravity.CENTER_VERTICAL

                            if(i == 0){

tvTeks.setAutoSizeTextTypeUniformWithConfiguration(
                                    1, 22, 1,
TypedValue.COMPLEX_UNIT_DIP);
                                tvTeks.setTextColor(Color.BLACK)
                                tvTeks.text = teks
                            }else{
                                if(arrDescription!!.size > 0){
                                    Log.v("WillyCheck",
"arrDescription: " + arrDescription!![i - 1])

tvTeks.setAutoSizeTextTypeUniformWithConfiguration(
                                        1, 14, 1,
TypedValue.COMPLEX_UNIT_DIP);
                                    tvTeks.setTextColor(Color.BLACK)
                                    tvTeks.text = arrDescription!![i -
1]
                                }
                            }
                            val xTopLeftToDp = xTopLeft * ratio
                            val yTopLeftToDp = yTopLeft * ratio
                            val templateLeft = ivTemplate.left
                            val templateTop = ivTemplate.top
                            tvTeks.animate().x(templateLeft +
xTopLeftToDp)
                            tvTeks.animate().y(templateTop +
yTopLeftToDp)
                            vResult.addView(tvTeks)
                        }
                    } catch (e: JSONException) {
                        Log.v("WillyCheck", e.toString())
                    }
                }, Response.ErrorListener { error ->
                    Log.v("WillyCheck", "Error [$error]")
                    Toast.makeText(
                        this,
                        "Cannot connect to server", Toast.LENGTH_LONG
```

```
                    ).show()
                }) {
                override fun getParams(): Map<String, String> {
                    val params: MutableMap<String, String> =
                        HashMap()
                    val filename =
Design2Fragment.filename.replace("\n", "")
                    params["filenameGambar"] = filename
                    return params
                }
            }
        sr.setRetryPolicy(
            DefaultRetryPolicy(
                500000,
                DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
                DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
            )
        )
        rq.add(sr)
```

4.2. Proses *Manage* Desain *Template*

Proses ini memberikan *admin* hak akses untuk melakukan *add*, dan *delete* desain *template*. Pada awal tampilan, *admin* diberikan daftar desain *template* yang terdaftar di *database*, lengkap beserta informasi jenis *output*, kategori, sub-kategori, dan jumlah gambar dan teks yang dapat ditampung tiap desain *template*. Tampilan awal juga memiliki sebuah tombol *Add New Template* yang berfungsi untuk menambahkan sebuah desain *template*.

Segmen Program 4.6. *Request API* untuk Mendapatkan Daftar Desain *Template*

```
val rq = Volley.newRequestQueue(context)
val url = serverUrl + "skripsi/api/getTemplate.php"
val sr: StringRequest = object : StringRequest(
                Method.POST,
                url, Response.Listener { response ->
                    try {
                        val arr = JSONArray(response)

                        val arrTemplate : MutableList<Template> =
```

```
arrayListOf()
                        for(i in 0..arr.length() - 1){
                            val temp = JSONObject(arr[i].toString())
                            val filename = temp["filename"].toString()
                            val hasGambar =
temp["has_gambar"].toString()
                            val hasTeks = temp["has_teks"].toString()
                            val namaKategori =
temp["nama_kategori"].toString()
                            val namaSubKategori =
temp["nama_sub_kategori"].toString()
                            val jenisOutput =
temp["jenis_output"].toString()
                            arrTemplate.add(Template(filename,
hasGambar, hasTeks, namaKategori, namaSubKategori, jenisOutput))
                        }
                        rvTemplate.layoutManager =
LinearLayoutManager(context)
                        rvTemplate.adapter =
TemplateAdapter(arrTemplate, requireContext())
                    } catch (e: JSONException) {
                        Log.v("WillyCheck", e.toString())
                        Toast.makeText(context, "Gagal Tampilkan
Template", Toast.LENGTH_SHORT).show()
                    }
                }, Response.ErrorListener { error ->
                    Log.v("WillyCheck", "Error [$error]")
                    Toast.makeText(
                        context,
                        "Cannot connect to server", Toast.LENGTH_LONG
                    )
                        .show()
                }) {
                override fun getParams(): Map<String, String> {
                    val params: MutableMap<String, String> =
                        HashMap()
                    params["is_filter"] = "0"
                    return params
                }
            }
        sr.setRetryPolicy(
            DefaultRetryPolicy(
                30000,
```

```
                DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
                DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
            )
        )
        rq.add(sr)
```

Setelah mendapatkan *response* berupa JSON, aplikasi akan menambahkan data dari JSON tersebut ke dalam *RecyclerView*. Jika salah satu desain *template* ditekan, maka akan muncul pilihan untuk menghapus desain *template* tersebut. Jika tombol *Add New Template* ditekan, *admin* diminta untuk melakukan *input* informasi mengenai desain *template* tersebut lengkap dengan koordinat untuk letak gambar atau/dan teks. Aplikasi akan melakukan *request API* untuk menunggah gambar ke dalam *server* dan memasukkan data ke dalam *database*.

4.3. Proses *Manage* Kategori Desain *Template*

Proses ini memberikan hak akses kepada *admin* untuk melakukan *add*, *edit*, dan *delete* kategori desain *template*. Pada awal tampilan, *admin* diberikan daftar kategori desain *template* yang terdaftar di *database*. Setiap barisnya memiliki dua tombol yaitu untuk *edit*, dan *delete*. Tampilan awal juga memiliki sebuah tombol *Add* Kategori yang berfungsi untuk menambahkan sebuah kategori desain *template*.

Segmen Program 4.7. *Request API* untuk Mendapatkan Daftar Kategori Desain *Template*

```
val rq = Volley.newRequestQueue(requireContext())
val url = serverUrl + "skripsi/api/getKategori.php"
val sr: StringRequest =
    object : StringRequest(
        Method.POST,
        url, Response.Listener { response ->
            try {
                val arr = JSONArray(response)
                arrKategori.clear()
                val arrNamaKategoriString : MutableList<String> =
arrayListOf()
                for(i in 0..arr.length() - 1){
                    val obj = JSONObject(arr[i].toString())
                    val idKategori = obj.get("id_kategori").toString()
                    val namaKategori =
obj.get("nama_kategori").toString()
                    val hasChildren =
```

```
obj.get("has_children").toString()
                    val arrSubKategori =
JSONArray(obj.get("sub_kategori").toString())
                    val arrSk: MutableList<SubKategori> = arrayListOf()
                    arrNamaKategoriString.add(namaKategori)
                    if(hasChildren == "1") {
                        for (j in 0..arrSubKategori.length() - 1) {
                            val subKategoriObj =
JSONObject(arrSubKategori[j].toString())
                            val idSubKategori =

subKategoriObj.get("id_sub_kategori").toString()
                            val idKategori =
subKategoriObj.get("id_kategori").toString()
                            val namaSubKategori =

subKategoriObj.get("nama_sub_kategori").toString()
                            val subKategori =
                                SubKategori(idSubKategori, idKategori,
namaSubKategori)
                            arrSk.add(subKategori)
                        }
                    }
                    val kategori = Kategori(idKategori, namaKategori,
hasChildren, arrSk)
                    arrKategori.add(kategori)


                }
                rvKategori.layoutManager =
LinearLayoutManager(requireContext())
                rvKategori.adapter = KategoriAdapter(arrKategori,
requireContext())
                loading.stopLoading()
            } catch (e: JSONException) {
                loading.stopLoading()
                Log.v("WillyCheck", e.toString())
            }
        }, Response.ErrorListener { error ->
            loading.stopLoading()
            Log.v("WillyCheck", "Error [$error]")
            Toast.makeText(
                requireContext(),
                "Cannot connect to server", Toast.LENGTH_LONG
```

```
            ).show()
        }) {
        override fun getParams(): Map<String, String> {
            val params: MutableMap<String, String> =
                java.util.HashMap()
            return params
        }
    }
sr.setRetryPolicy(
    DefaultRetryPolicy(
        10000,
        DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
        DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
    )
)
rq.add(sr)
```

Setelah mendapatkan *response* berupa JSON, aplikasi akan menambahkan data dari JSON tersebut ke dalam *RecyclerView*. Jika tombol *delete* ditekan, aplikasi akan melakukan *request API* untuk menghapus kategori tersebut dari *database*. Jika tombol *edit* ditekan, aplikasi akan meminta nama kategori baru untuk melakukan *request API update* data kategori. Jika tombol *Add* Kategori ditekan, *admin* diminta untuk melakukan *input* nama kategori baru kemudian melakukan *request API* untuk menambahkan kategori tersebut ke dalam *database*.

4.4. Proses *Manage* Sub-Kategori Desain *Template*

Proses ini memberikan hak akses kepada *admin* untuk melakukan *add*, *edit*, dan *delete* sub-kategori desain *template*. Pada awal tampilan, *admin* diberikan daftar sub-kategori desain *template* dan nama kategorinya yang terdaftar di *database*. Setiap barisnya memiliki dua tombol yaitu untuk *edit*, dan *delete*. Tampilan awal juga memiliki sebuah tombol *Add* Sub Kategori yang berfungsi untuk menambahkan sebuah sub-kategori desain *template*.

Segmen Program 4.8. *Request API* untuk Mendapatkan Daftar Sub-kategori Desain *Template*

```
val rq = Volley.newRequestQueue(context)
            val url = serverUrl + "skripsi/api/addSubKategori.php"
            val sr: StringRequest =
                @SuppressLint("ResourceType")
                object : StringRequest(
                    Method.POST,
                    url, Response.Listener { response ->
                        try {
                            val obj = JSONObject(response)
                            val msg = obj.get("msg").toString()
                            Toast.makeText(context, msg,
Toast.LENGTH_SHORT).show()
                            etNamaSubKategori.setText("")
                            sKategori.setSelection(0)
                            loading.stopLoading()
                        } catch (e: JSONException) {
                            loading.stopLoading()
                        }
                    }, Response.ErrorListener { error ->
                        loading.stopLoading()
                        Toast.makeText(
                            context,
                            "Cannot connect to server",
Toast.LENGTH_LONG
                        ).show()
                    }) {
                    override fun getParams(): Map<String, String> {
                        val params: MutableMap<String, String> =
                            HashMap()
                        params["nama_sub_kategori"] =
etNamaSubKategori.text.toString()
                        params["nama_kategori"] =
sKategori.selectedItem.toString()
                        return params
                    }
                }
            sr.setRetryPolicy(
                DefaultRetryPolicy(
                    500000,
                    DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
```

```
                    DefaultRetryPolicy.DEFAULT_BACKOFF_MULT
                )
            )
            rq.add(sr)
```

Setelah mendapatkan *response* berupa JSON, aplikasi akan menambahkan data dari JSON tersebut ke dalam *RecyclerView*. Jika tombol *delete* ditekan, aplikasi akan melakukan *request API* untuk menghapus sub-kategori tersebut dari *database*. Jika tombol *edit* ditekan, aplikasi akan meminta nama sub-kategori baru, dan kategori apa yang menjadi *parent*-nya untuk melakukan *request API update* data sub-kategori. Jika tombol *Add* Sub-Kategori ditekan, *admin* diminta untuk melakukan *input* nama sub-kategori baru, dan kategori apa yang menjadi *parent*-nya kemudian melakukan *request API* untuk menambahkan sub-kategori tersebut ke dalam *database*.

4.5. Proses Post

Proses ini memiliki dua pilihan yaitu *Post Now*, dan *Scheduled Post. User* dapat melakukan aktivitas *post* dari gambar yang telah diunggah terlebih dahulu melalui fitur "*Add Photo*". Gambar tersebut dapat ditambahkan sebuah *caption* sehingga menjadi *post* untuk diunggah ke Instagram melalui *Instagram Graph API*. *User* dapat memilih untuk melakukan *Post Now* untuk mengunggah gambar tersebut beserta *caption* yang bersifat opsional ke *Instagram* melalui *request Instagram Graph API.*

Segmen Program 4.9. *Add Photo* ke *Server* Sebelum Melakukan *Post*

```
val settings = requireContext().getSharedPreferences("login", 0)
val id: String = settings.getString("instagram_id", "")!!
Val username: String = settings.getString("username", "")!!
Val url = HomeActivity.serverUrl +
"skripsi/api/uploadInstagramUserId.php"
val rq = Volley.newRequestQueue(context)
val stringRequest: StringRequest = object : StringRequest(
    Method.POST, url, Response.Listener { response ->
          try {
               val obj = JSONObject(response)
               val data = JSONObject(obj.get("data").toString())
```

```
                val formId = data.get("form_id").toString()
                val filename = data.get("filename").toString()
                val url = HomeActivity.serverUrl +
"skripsi/api/uploadImage.php"
                val request = object : VolleyFileUploadRequest(
                    Method.POST, url, Response.Listener { response ->
                        Toast.makeText(context, "Berhasil Upload",
                         Toast.LENGTH_SHORT).show()
                        val mediaUrl = HomeActivity.serverUrl +
"skripsi/files/foto/" + filename
                        val preview = Preview("", "", username,
                            "IMAGE", mediaUrl, "", "",
                            "", "", "", 0)
                        PreviewFragment.arrPost.add(0, preview)
                        val manager: FragmentManager = (context as
AppCompatActivity).supportFragmentManager
                        manager.beginTransaction().run {
                            replace(R.id.v_main, PreviewFragment())
                            commit()
            }
      }, Response.ErrorListener { response ->
         Log.v("WillyCheck", "error: " + response.toString())
      }) {override fun getByteData(): MutableMap<String, FileDataPart>{
                val params = HashMap<String, FileDataPart>()
                params["fileFoto"] = FileDataPart(formId, byteArr,
"jpeg")
                return params
        }
}
Volley.newRequestQueue(context).add(request)
} catch (e: JSONException) {
  Log.v("WillyCheck", "exception: " + e.toString())
  Toast.makeText(ontext,"Error while loading
data!",Toast.LENGTH_LONG).show()
}
}, Response.ErrorListener { error ->
    Log.v("WillyCheck","Error [$error], " + error.printStackTrace())
    Toast.makeText(context,"Cannot connect to server",Toast.LENGTH_LONG
    ).show()
}) {
  override fun getParams(): Map<String, String> {
  val params: MutableMap<String, String> = HashMap()
  if(!id.isEmpty()) {
```

```
     params[“instagram_user_id”] = id
  }
   return params
}
}
stringRequest.setRetryPolicy(DefaultRetryPolicy(
               500000,
                DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
                DefaultRetryPolicy.DEFAULT_BACKOFF_MULT))
rq.add(stringRequest)
```

Setelah gambar telah terunggah ke *server*, *user* dapat melihat bahwa *preview Instagram feed* yang menunjukkan bagaimana *feed* akan terlihat jika *user* mengunggah gambar tersebut ke *Instagram*. *User* dapat melakukan dua jenis aktivitas posting, *Post Now* dan *Scheduled Post*. Jika *Post Now* dipilih, aplikasi akan melakukan *request* ke *Instagram Graph API*.

Segmen Program 4.10. *Request* Instagram Graph API dan *Request* API ke S*erver*

```
val rq = Volley.newRequestQueue(this)
val url = “https://graph.facebook.com/” + instagramUserId +
“/media?image_url=” + PreviewFragment.objPreview.mediaUrl + “&caption=”
+ etCaption.text.toString() + “&access_token=” +
settings.getString(“token”, “”)!!

Val sr: StringRequest = object : StringRequest(
                        Method.POST,
                        url, Response.Listener { response →
try {
     val obj = JSONObject(response)
     val creationId = obj.get(“id”).toString()
     val rq2 = Volley.newRequestQueue(this)
     val url2 = “https://graph.facebook.com/” + instagramUserId +
“/media_publish?creation_id=” + creationId + “&access_token=” +
settings.getString(“token”, “”)!!
     Val sr2: StringRequest = object : StringRequest(
     Method.POST,
     url2, Response.Listener { response ->
         try {
             val rq3 = Volley.newRequestQueue(this)
```

```
            val url3 = serverUrl + "skripsi/api/deletePost.php"
            val sr3: StringRequest =
                    bject : StringRequest(
                            Method.POST,
                            url3, Response.Listener { response ->
                            try {
                               Toast.makeText(this, "Berhasil
upload", Toast.LENGTH_SHORT).show()
                               loading.stopLoading()
                            val intent = Intent(this,
MainActivity::class.java)
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
intent.putExtra("open", "preview")
startActivity(intent)
} catch (e: JSONException) {
loading.stopLoading()
Toast.makeText(this, "Gagal Upload", Toast.LENGTH_SHORT).show()
     }
       }, Response.ErrorListener { error →
loading.stopLoading()
Log.v("WillyCheck", "Error [$error]")
Toast.makeText(this,"Cannot connect to server",
Toast.LENGTH_LONG).show()
       }) {
override fun getParams(): Map<String, String> {
val params: MutableMap<String, String> = HashMap()
val filepath = PreviewFragment.objPreview.mediaUrl
val arr = filepath.split("/").toTypedArray()
val filename = arr[arr.size - 1]
params["filename"] = filename
return params
}
}
sr3.setRetryPolicy(
DefaultRetryPolicy(
10000,DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
DefaultRetryPolicy.DEFAULT_BACKOFF_MULT)
)
rq3.add(sr3)
} catch (e: JSONException) {
loading.stopLoading()
Toast.makeText(this, "Gagal Upload", Toast.LENGTH_SHORT).show()
}
```

```
}, Response.ErrorListener { error →
loading.stopLoading()
Toast.makeText(
this,
“Cannot connect to server”, Toast.LENGTH_LONG).show()
}) {
override fun getParams(): Map<String, String> {
val params: MutableMap<String, String> = HashMap()
return params
}
}
sr2.setRetryPolicy(
DefaultRetryPolicy(10000,DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
DefaultRetryPolicy.DEFAULT_BACKOFF_MULT))
rq2.add(sr2)
} catch (e: JSONException) {
loading.stopLoading()
Toast.makeText(this, “Gagal Upload”, Toast.LENGTH_SHORT).show()
}
}, Response.ErrorListener { error →
loading.stopLoading()
Log.v(“WillyCheck”, “Error [$error]”)
Toast.makeText(this,”Cannot connect to server”,
Toast.LENGTH_LONG).show()
}) {
override fun getParams(): Map<String, String> {
     val params: MutableMap<String, String> = HashMap()
     return params
}
}
sr.setRetryPolicy(
    DefaultRetryPolicy(10000,efaultRetryPolicy.DEFAULT_MAX_RETRIES,
     DefaultRetryPolicy.DEFAULT_BACKOFF_MULT)
)
rq.add(sr)
```

Setelah proses *Post Now* berhasil dilakukan, *request API* dilakukan ke arah *server* untuk menghapus data dari data *unposted*. Selain itu *user* juga dapat memilih tombol *Scheduled Post* dimana *user* diminta untuk melakukan input tanggal dan waktu sebagai jadwal *post*. Jadwal *post* ini akan dimasukkan ke dalam *parameter request API* ke *server* untuk melakukan *schedule job AT command*, dan *input* data ke *database*. *Job* ini

didaftarkan ke dalam *server* untuk menjalankan sebuah *script* PHP pada jadwal yang telah ditentukan *user* dimana jadwal yang dapat ditentukan *user* minimal enam menit sebelum jadwal *scheduled post*. *Script* PHP yang dijalankan oleh *AT command* berfungsi untuk melakukan *request* ke *Instagram Graph API*.

Segmen Program 4.11. *Request API* ke *Server* untuk *Schedule Job AT Command*

```
val sdf = SimpleDateFormat("yyyy/MM/dd HH:mm")
val current = sdf.format(Date())
val selectedTime = bPick.text.toString()
val formatter = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm",
Locale.ENGLISH)
var selectedDateTime = LocalDateTime.parse(selectedTime, formatter)
val currentDateTime = LocalDateTime.parse(current, formatter)
var tempDateTime = currentDateTime
val dYear = tempDateTime.until(selectedDateTime, ChronoUnit.YEARS)
tempDateTime = tempDateTime.plusYears(dYear)
val dMonth = tempDateTime.until(selectedDateTime, ChronoUnit.MONTHS)
tempDateTime = tempDateTime.plusMonths(dMonth)
val dDay = tempDateTime.until(selectedDateTime, ChronoUnit.DAYS)
tempDateTime = tempDateTime.plusDays(dDay)
val dHour = tempDateTime.until(selectedDateTime, ChronoUnit.HOURS)
tempDateTime = tempDateTime.plusHours(dHour)
val dMinute = tempDateTime.until(selectedDateTime, ChronoUnit.MINUTES)

val differenceInMinutes = dYear * 525600 + dMonth * 43800 + dDay * 1440
+ dHour * 60 + dMinute
if(differenceInMinutes < 6){
   Toast.makeText(this, "Harap pilih tanggal dan waktu untuk
schedule!", Toast.LENGTH_SHORT).show()
   return@setOnClickListener
}else{
     val settings = getSharedPreferences("login", 0)
     val instagramUserId = settings.getString("instagram_id", "")
     val rq = Volley.newRequestQueue(this)
     val url = serverUrl + "skripsi/api/scheduledPost.php"
     val sr: StringRequest = object : StringRequest(
         Method.POST, url, Response.Listener { response ->
             try {
                    loading.stopLoading()
                    val intent = Intent(this, MainActivity::class.java)
```

```
                                intent.putExtra("open", "preview")
                                startActivity(intent)
                    } catch (e: JSONException) {
                        loading.stopLoading()
                        Toast.makeText(this, "Gagal Upload",
Toast.LENGTH_SHORT).show()
                    }
                }, Response.ErrorListener { error ->
                    loading.stopLoading()
                    Toast.makeText(this, "Cannot connect to server",
Toast.LENGTH_LONG).show()
            }) {
                 override fun getParams(): Map<String, String> {
                    val params: MutableMap<String, String> = HashMap()
                    params["instagram_user_id"] =
instagramUserId.toString()
                    params["caption"] = etCaption.text.toString()
                    params["media_url"] =
PreviewFragment.objPreview.mediaUrl
                    params["access_token"] = settings.getString("token",
"")!!
                    Val time = bPick.text.toString().replace("/", "-")
                    val arrTime = time.split(" ").toTypedArray()
                    params["time"] = arrTime[1] + " " + arrTime[0]
                    val filepath = PreviewFragment.objPreview.mediaUrl
                    val arr = filepath.split("/").toTypedArray()
                    val filename = arr[arr.size - 1]
                    params["filename"] = filename
                    return params
                }
            }
           sr.setRetryPolicy(
               DefaultRetryPolicy(10000,
DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
DefaultRetryPolicy.DEFAULT_BACKOFF_MULT)
)
rq.add(sr)
}
```

Setelah *script* PHP yang dijalankan oleh *AT command* berjalan, *request* ke *Instagram Graph API* pun dilakukan. Setelah request tersebut berhasil, data gambar dan

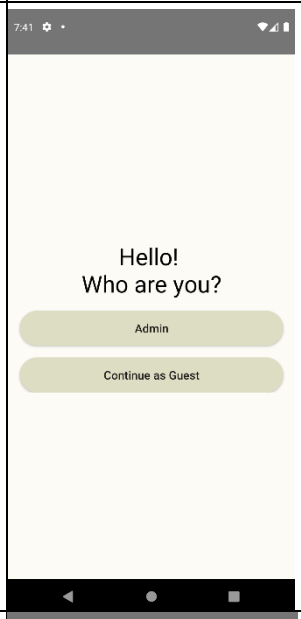
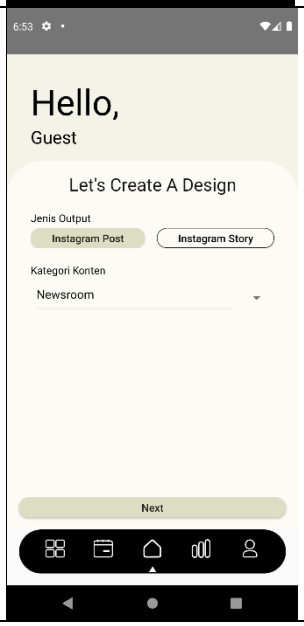
caption yang tersimpan di *database* akan dihapus.

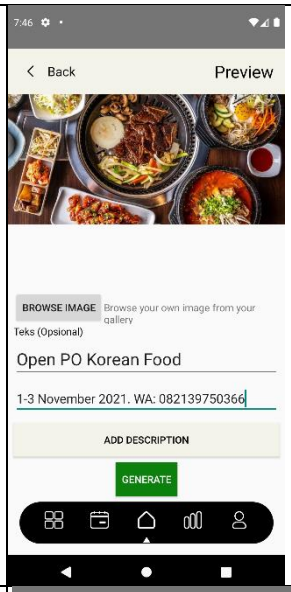
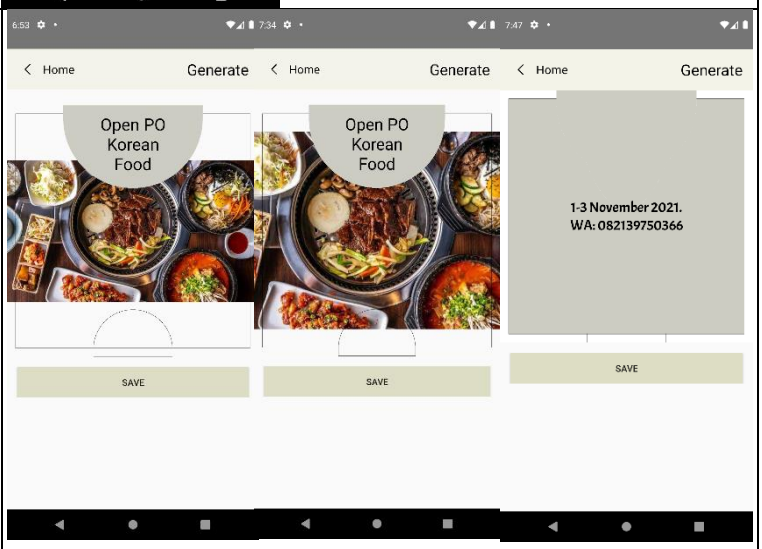
4.6. Implementasi *User Interface* Aplikasi

Pada tahap ini, desain *user interface* aplikasi dibuat sesederhana mungkin dan dengan kebutuhan untuk berpindah halaman yang sedikit. Beberapa *user interface* dapat dilihat pada Tabel 4.2.

Tabel 4.2.

Implementasi *User Interface*

Nama	Implementasi dan Deskripsi	
Halaman Awal		Halaman pertama ini memberikan dua tombol untuk dipilih. Masing-masing tombol akan melakukan *redirect* halaman dimana *admin* akan diminta untuk melakukan *login*, sedangkan *guest* akan langsung diberikan akses ke *main menu.*
Halaman Pertama Pembuatan Desain		Pada halaman ini, *user* diberikan tampilan untuk memilih jenis *output* yang berguna untuk menentukan resolusi hasil desain, dan memilih kategori maupun sub-kategori konten yang akan membedakan desain *template* yang akan terpilih.

Halaman Kedua Pembuatan Desain		Pada halaman ini, *user* diminta untuk memilih satu gambar dari galeri *smartphone*, dan memasukkan satu atau lebih teks.
Halaman Akhir Desain		Pada halaman ini, *user* diberikan hasil penggabungan antara gambar yang diunggah dan desain *template* yang terpilih beserta dengan teks yang telah dimasukkan. Gambar dari *user* dapat di *rescale* yaitu *zoom in* dan *zoom out* untuk penyesuaian. Tersedia juga sebuah tombol *save* untuk menyimpan desain tersebut ke dalam galeri.