BAB III
METODE PENELITIAN

3.1. Metode Pengumpulan Data

Metode pengumpulan data adalah cara yang digunakan untuk memperoleh informasi yang dibutuhkan dalam proses penelitian. Dalam penelitian ini, beberapa metode digunakan untuk mengumpulkan data, sebagai berikut :

a. Wawancara, wawancara dilakukan kepada salah satu petani di kecamatan Iringmulyo kota Metro dengan memberikan beberapa pertanyaan terkait tanaman jagung.

Tabel 3.1 Hasil Wawancara dengan petani di Kecamatan Iringmulyo

Narasumber		Bapak Suroto
No	**Pertanyaan**	**Jawaban**
1.	Berapa usia tanaman jagung yang terdapat pada lahan ?	Usia jagung yang ada pada lahan berusia 80 hari
2.	Berapa ukuran lahan untuk menanam tanaman jagung tersebut ?	Ukuran lahan yang digunakan yaitu sekitar 28m x 95m yang menghasilkan panen sekitar 1 ton
3.	Apakah tanaman yang bapak tanamn pernah terjangkit suatu penyakit ?	Tanaman sangat jarang terkena penyakit, karena jika terkena salah satu penyakit yaitu bulai, dapat mengakibatkan gagal panen dan harus menanam ulang
4.	Langkah apa yang dilakukan untuk memeriksa kondisi kesehatan tanaman jagung ?	Untuk melihat kondisi biasanya dilakukan pemeriksaan dengen mengecek secara visual atau langsung per 10 hari masa tanam
5.	Jenis penyakit apa yang sering tanaman jagung tersebut alami ?	Jenis penyakit yang pernah dialami yaitu bernama bercak putih atau bulai jamur
6.	Bagaimana cara untuk mengatasi penyakit tanaman tersebut ?	Untuk mengatasi penyakit tanaman tersebut yaitu dengan memberikan anti jamur pada bibit, selain itu juga dilakukan penyemprotan pada tanaman
7.	Apakah penyakit yang menyerang tanaman jagung berdampak signifikan pada hasil panen ? Jika iya, seberapa besar kerugiannya	Jika tanaman jagung terkena penyakit tersebut dapat berdampak sangat besar yaitu dapat menyebabkan gagal panen yang mengharuskan tanam ulang
8.	Apakah teknologi sudah diterapkan dalam mendeteksi tanaman jagung yang terjangkit penyakit ?	Belum, belum pernah menggunakan teknologi dalam memerika suatu penyakit tanaman jagung

b. Observasi, pada proses observasi peneliti melakukan pengamatan secara langsung ke lahan pemilik tanaman jagung agar mendapatkan data yang diingikan.

Gambar 3.1 Lokasi observasi

Gambar 3.2 Obervasi ke lahan pertanian

Gambar 3.1 merupakan lahan atau lokasi yang digunakan dalam pengambilan gambar atau *sampling*, dan gambar 3.2 merupakan gambaran lahan yang digunakan untuk observasi.

c. Sampling, pada tahap sampling dilakukan pengambilan foto secara acak pada tanaman jagung baik yang memiliki penyakit ataupun tidak memiliki penyakit.

Data tersebut digunakan untuk menguji aplikasi guna memastikan bahwa model yang sudah dilatih menggunakan dataset *Kaggle* yaitu penyakit tanaman jagung mampu mengenali kondisi tanaman jagung di lapangan. Data gambar diambil menggunakan 3 jenis hp yaitu Xiaomi Redmi 5A, Xiaomi Redmi 12, dan Samsung A21s.

Gambar 3.3 *Sample* hp Xiaomi Redmi 5A

Gambar 3.4 *Sample* hp Xiaomi Redmi 12

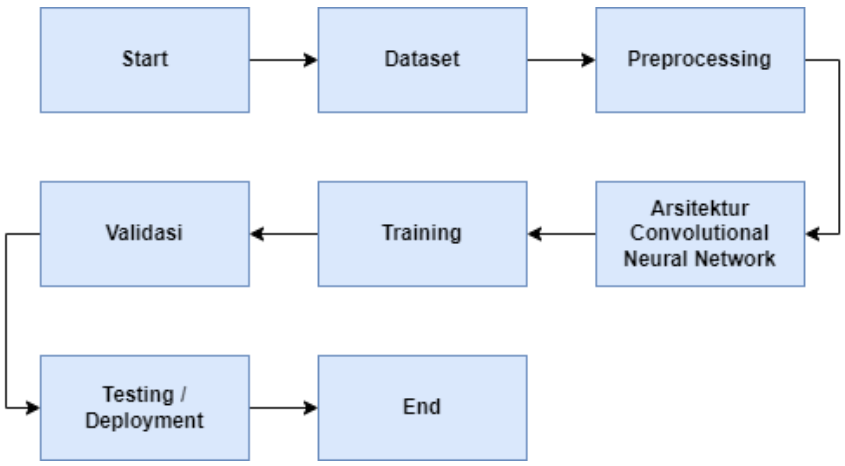
Gambar 3.5 *Sample* hp Samsung A21s

d. Studi Pustaka, pada tahap studi pustaka penulis mencari sumber-sumber untuk mendukung penelitian seperti jurnal ilmiah, buku, dan skripsi-skripsi sebelumnya.

3.2. Metode Convolutional Neural Network (CNN)

Dalam penelitian ini tahapan penelitian *dataset* memanfaatkan metode *Convolutional Neural Network*. Perancangan *training* data pada penelitian ini menggunakan Google Colaboratory dengan menggunakan bahasa pemrograman python. Perancangan arsitektur metode *Convolutional Neural Network*,

menggunakan beberapa tahap meliputi *dataset*, *preprocessing*, arsitektur *Convolutional Neural Network*, *training*, validasi, *testing* [38] seperti pada gambar 3.6, diantaranya :



Gambar 3.6 Alur Metode *Convolutional Neural Network* (CNN)

a. *Dataset*

Pada penelitian ini pada tahap CNN pertama dibutuhkannya data yang akan digunakan dalam pendeteksian penyakit tanaman jagung. *Dataset* yang digunakan merupakan *open dataset* (dataset terbuka) yang diperoleh dari situs *Kaggle*. *Dataset* tersebut terdiri dari 4 kelas, yaitu 1 kelas sehat dan 3 kelas tanaman jagung yang memiliki penyakit, *Dataset* terdiri atas 4000 gambar dengan rincian 1000 gambar tanaman jagung sehat (*healthy*), 1000 gambar tanaman jagung terkena hawar (*blight*), 1000 gambar tanaman jagung terkena karat daun jagung (*common rust*), 1000 gambar tanaman jagung terkena bercak daun abu-abu (*gray leaf spot*). *Dataset* tersebut kemudian disimpan ke Google Drive yang berfungsi menjadi media simpan pada Google Collaboratory.

Gambar 3.7 Daun jagung sehat (*Healthy*)

Gambar 3.8 Daun jagung hawar (*blight*)

Gambar 3.9 Karat daun jagung (*common rust*)

Gambar 3.10 Daun jagung bercak daun abu-abu (*gray leaf spot*)

b. *Preprocessing*

Pada tahap *preprocessing* ini meyiapkan *dataset* yang terdiri atas 4 kelas yang sudah diunggah ke *google drive*. Setelah itu *dataset* akan dipisahkan menjadi 2 bagian yaitu *training* dan *test*, dengan 2 rasio 70:30 dan 80:20.

```
#Split data
x_train, x_test, y_train, y_test = train_test_split(df['image'], df['label'], test_size=0.3, random_state=42, stratify=df['label']) # Mempertahankan proporsi label

df_train = pd.DataFrame({'image': x_train, 'label': y_train}) # Data pelatihan
df_test = pd.DataFrame({'image': x_test, 'label': y_test}) # Data test/pengujian

encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)
```

Gambar 3.11 Rancangan *Split* data python 70:30

```
#Split data
x_train, x_test, y_train, y_test = train_test_split(df['image'], df['label'], test_size=0.2, random_state=42, stratify=df['label']) # Mempertahankan proporsi label

df_train = pd.DataFrame({'image': x_train, 'label': y_train}) # Data pelatihan
df_test = pd.DataFrame({'image': x_test, 'label': y_test}) # Data test/pengujian

encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)
```

Gambar 3.12 Rancangan *Split* data python 80:20

Kemudian *dataset* gambar yang sudah dimiliki pada rentang 0-255 yang kemudian akan dinormalisasikan menjadi rentang 0-1 untuk mempermudah dalam proses pembelajar, dan mengubah ukuran gambar menjadi 256 x 256 agar model gambar dilatih dengan ukuran gambar yang tidak berubah-ubah.

```
image_size = (256, 256)
batch_size = 32

train_datagen = ImageDataGenerator(
    rescale=1./255,           # Mengubah skala piksel dari [0, 255] menjadi [0, 1].
    rotation_range=20,        # Memutar gambar acak hingga 20 derajat.
    width_shift_range=0.2,    # Menggeser gambar secara horizontal
    height_shift_range=0.2,   # Menggeser gambar secara vertical
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'       # Mengisi piksel kosong dengan nilai piksel terdekat.
)

train_generator = train_datagen.flow_from_dataframe(
    df_train,                 # DataFrame yang berisi data pelatihan.
    x_col='image',
    y_col='label',
    target_size=image_size,   # Mengubah ukuran gambar menjadi 256x256 piksel.
    batch_size=batch_size,    # Ukuran batch yang digunakan adalah 64 gambar per batch.
    class_mode='categorical',
    shuffle=True              # Mengacak urutan gambar pada setiap epoch.
)


test_generator = train_datagen.flow_from_dataframe(
    df_test,                  # DataFrame yang berisi data test/pengujian.
    x_col='image',
    y_col='label',
    target_size=image_size,   # Mengubah ukuran gambar menjadi 256x256 piksel.
    batch_size=batch_size,    # Ukuran batch yang digunakan adalah 64 gambar per batch.
    class_mode='categorical',
    shuffle=False             # Tidak diacak
)
```

Gambar 3.13 Rancangan Mengubah ukuran gambar

c. Arsitektur *Convolutional Neural Network* (CNN)

Dalam arsitektur *Convolutional Neural Network* terdiri atas kumpulan layer-layer yang akan dilewati gambar pada proses training data. Pada gambar 3.8 rancangan yang peneliti buat menggunakan 256 x 256 x 3 (RGB), *kernel* yang digunakan yaitu 3 x 3 dan filter 32 layer pada konvolusi pertama.

Pada konvolusi kedua menggunakan 32 filter dan *kernel* 3 x 3, pada konvolusi 3 dan 4 menggunakan 64 filter dengan ukuran *kernel* 3 x 3, Output layer (softmax) dibagi menajdi 4 kelas.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(256, 256, 3))) # Input shape 256x256x3 # Kernel 3x3 # Filter layer 32
model.add(MaxPooling2D(2,2))
model.add(BatchNormalization())
model.add(Conv2D(32, (3,3), activation='relu', padding='same')) #Konvolusi kedua 32 filter ukuran kernel 3x3
model.add(MaxPooling2D(2,2))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu', padding='same')) # Konvolusi ketiga 64 filter ukuran kernel 3x3
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64,(3,3),activation='relu', padding='same')) # Konvolusi keempat 64 filter ukuran kernel 3x3
model.add(MaxPooling2D(2,2))
model.add(Flatten())
model.add(Dense(126, activation='relu')) #Lapisan fully connected
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax')) #Klasifikasi multi-kelas (4 kelas)
```

Gambar 3.14 Rancangan *Convolutional Neural Network* (CNN)

d. *Training*

Training dilaksanakan dengan tujuan agar model CNN yang sudah dirancang dapat mengenali dan membedakan gambar pada daun tanaman jagung baik daun sehat maupun tidak sehat. Training dilakukan kepada 4 kelas yang sebelumnya sudah dilakukan *split* data.

```
history = model.fit(
    train_generator,
    batch_size = 32,
    validation_data = test_generator,
    epochs = 50,
)
```

Gambar 3.15 Rancangan Proses training *dataset*

e. Validasi

Validasi digunakan untuk menguji keakuratan hasil training, dari hasil validasi memungkinkan untuk meminimalisir hasil *overfitting* pada data training, validasi dilihat dari data hasil training awal hingga akhir proses training, jika ada lonjakan pada bagian *loss* dapat berakibat data yang diolah menjadi *overfitting* dan sulit untuk digunakan dalam mendeteksi penyakit tanaman, selain itu hasil lainnya yang

digunakan yaitu F1 Score, Recall dan Precision. *Overfitting* merupakan keadaan model yang terlalu fokus, fokus yang terlalu berlebihan pada model dapat membuat model menangkap gambar yang kurang jelas atau disebut *noise* yang seharausnya data tersebut tidak ditangkap. Dengan model yang terkena *overfitting* dapat mengakibatkan akurasi dari model menjadi berkurang.

f. *Testing* / *Deployment*

Setelah semua proses telah selesai dijalankan maka proses terakhir pada metode *convolutional neural network* (CNN) adalah dengan melakukan *testing*, hal ini merupakan kewajiban untuk membuktikan bahwa model yang telah dibuat adalah model yang akurat sebelum model tersebut diproses untuk dijadikan aplikasi Android.

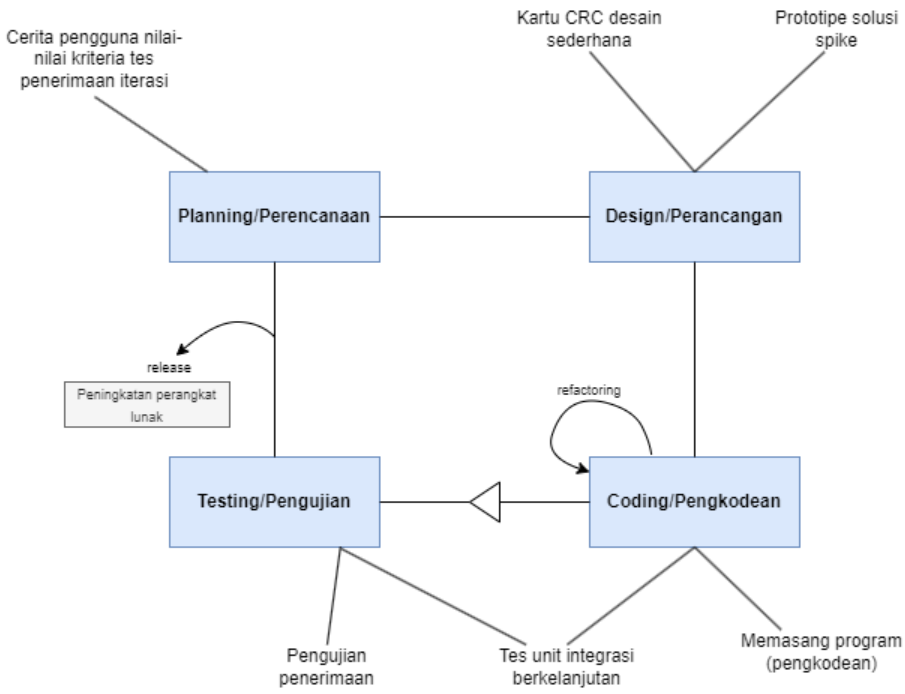
Dari hasil *testing* tersebut dihitung nilai akurasi dengan rumus sebagai berikut :

$$akurasi = \frac{Jumlah\ data\ testing\ yang\ benar}{Jumlah\ keseluruhan\ data\ testing}\ x\ 100\%$$

3.3. Metode Pengembangan Sistem

Pada penelitian ini, pengembangan sistem yang digunakan untuk mengembangkan aplikasi deteksi penyakit tanaman jagung adalah dengan memanfaatkan *Extreme Programming*.

Extreme Programming (XP) merupakan tenik pengembangan perangkat lunak yang populer karena mencakup beberapa langkah penting, yaitu perencanaan (*planning*), perancangan (*design*), pengkodean (*coding*), dan pengujian (*testing*) [39]. Langkah-langkah ini dapat dilihat pada gambar 3.16.



Gambar 3.16 Alur metode *Extreme Programming*

a. Perencanaan (*Perencanaan*)

Pada tahap pertama ini peneliti memulai memahami konteks dari aplikasi dan mendapatkan gambaran mengenai keluaran fitur-fitur dan fungsinya.

b. Perancangan (*Design*)

Dalam tahap ini peneliti memulai tahap perencanaan yang dilakukan dengan pembuatan sistem berdasarkan yang dibutuhkan. Pada tahap ini dibuatkan permodelan sistem yaitu UML (*Unified Modelling Language*) yang terdiri atas *Use Case Diagram*, *Activity Diagram*, *Class Diagram*, dan *Sequence Diagram*.

c. Pengkodean (*Coding*)

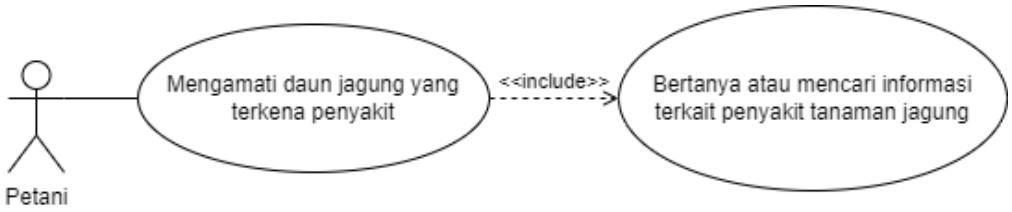
Dalam tahapan ini dilakukan implementasi dari perancangan model yang telah dirancang kedalam kode-kode program android. Dalam penelitian ini, pengkodean aplikasi dikembangkan menggunakan bahasa pemrograman Kotlin dan *platform* Android Studio.

d. Pengujian (*Testing*)

Tahap ini adalah tahap uji coba pada aplikasi yang telah dikembangkan, pada tahap ini, pengujian difokuskan pada fitur-fitur yang telah dirancang sehingga fitur-fitur tersebut tidak ada kesalahan dan telah sesuai.

3.4. UML (*Unified Modelling Language*)

3.4.1. *Use Case Diagram* Berjalan



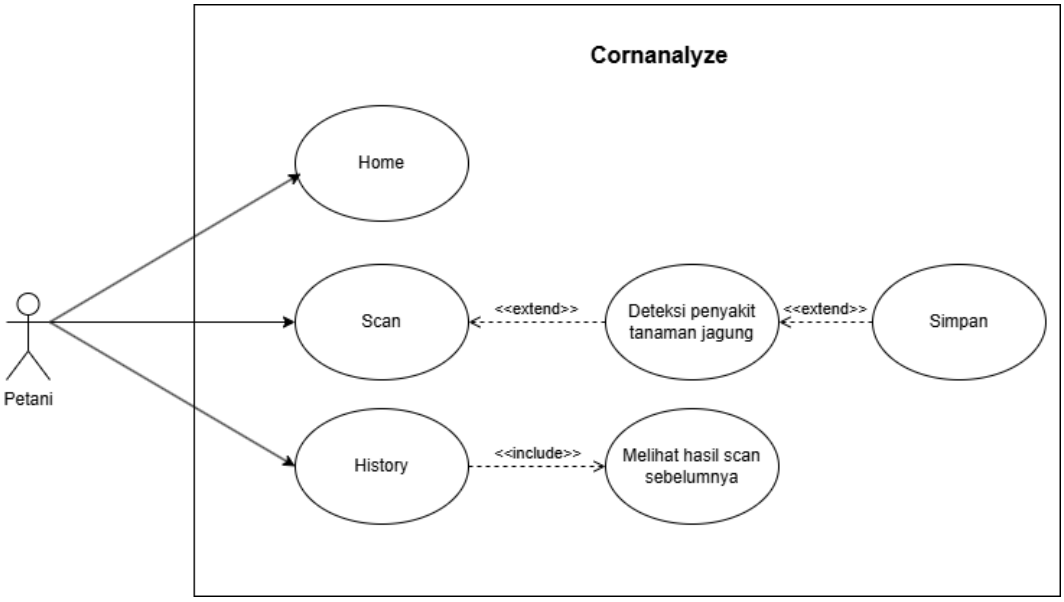
Gambar 3.17 *Use case diagram* berjalan

Berdasarkan gambar 3.17 mengenai *use case diagram* berjalan terdapat satu aktor yaitu petani dan dua *use case*. Untuk deskripsi dari fungsionalitas *use case* dapat dilihat pada tabel 3.2 Deskripsi *use case* berjalan.

Tabel 3.2 Deskripsi *use case diagram* berjalan

No	Aktor	*Use case*	Deskripsi
1	Petani	Mengamati daun jagung yang terkena penyakit	Petani secara langsung memantau kondisi daun jagung yang memperlihatkan tanda-tanda terkena penyakit
2	Petani	Bertanya atau mencari informasi terkait penyakit tanaman jagung	Setelah memantau kondisi daun jagung, petani berusaha mencari informasi lebih lanjut dari tanda-tanda penyakit yang dialami oleh daun jagung

3.4.2. *Use Case Diagram* Yang Diusulkan



Gambar 3.18 *Use case diagram* yang diusulkan

Berdasarakan gambar 3.18 *use case diagram* yang diusulkan terdiri dari aktor dan *use case*, aktor tersebut digambarkan sebagai seorang petani yang dapat mengakses *home*, *scan*, *history*, *setting*. Untuk keterangan dari fungsionalitas *use case* yang diusulkan, dapat dilihat pada tabel 3.3 Deskripsi *use case diagram* yang diusulkan.

Tabel 3.3 Deskripsi *use case diagram* yang diusulkan

No	Aktor	*Use case*	Deskripsi
1	Petani	*Home*	*Home* merupakan proses dimana aktor berhasil membuka aplikasi
2	Petani	*Scan*	Aktor memilih menu *scan* dan menu *scan* tampil
3	Petani	*History*	Aktor memilih menu *history* dan menu *history* tampil
4	Petani	Deteksi penyakit tanaman jagung	Deteksi penyakit tanaman jagung akan dijalankan apabila aktor ingin melakukan deteksi dengan opsi *opsional*
5	Petani	Melihat hasil *scan* sebelumnya	Hasil scan sebelumnya akan tampil apabila aktor menekan menu *history*
6	Petani	Simpan	Simpan dapat dijalankan apabila aktor sudah melakukan deteksi penyakit pada tanaman jagung dan ingin melakukan simpan pada hasil deteksi yang bersifat *opsional*

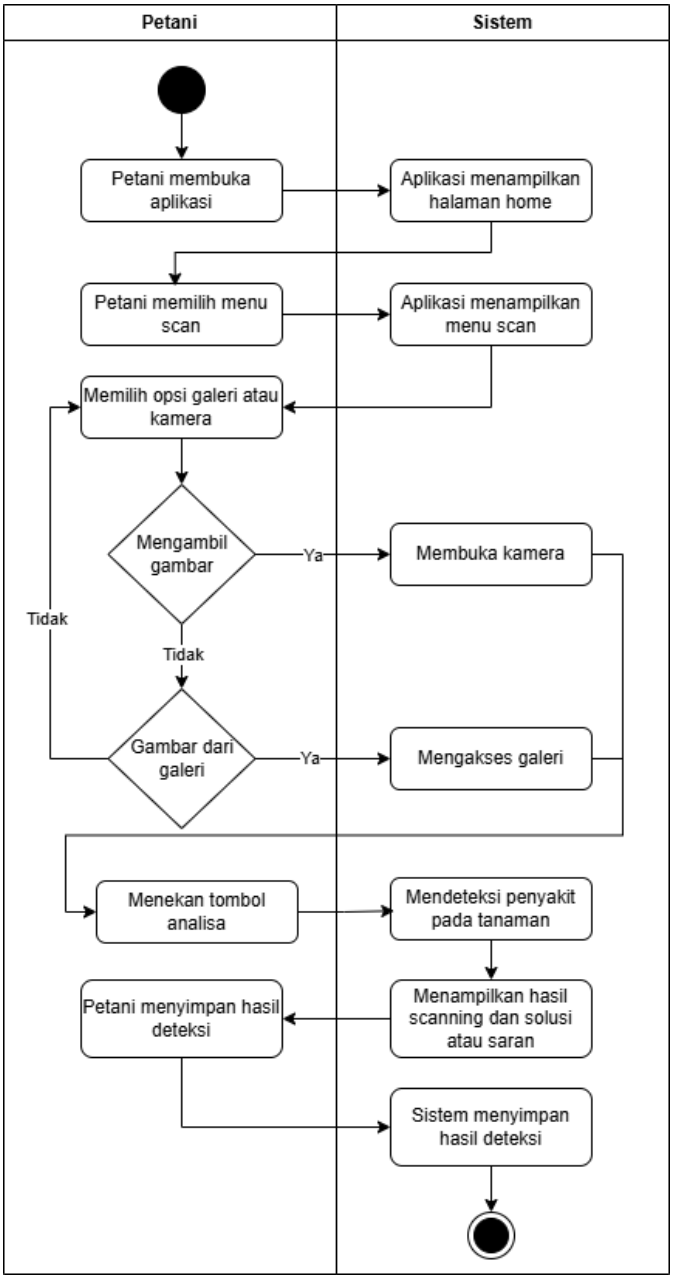
Tabel 3.4 *Use case scenario scan* tanaman jagung

Petani	**Sistem**
1. Petani membuka aplikasi CornAnalyze	
2. Petani memilih menu *scan*	
	3. Sistem menampilkan menu *scan*
4. Petani melakukan deteksi pada tanaman jagung	
	5. Sistem memproses gambar
	6. Sistem menampilkan hasil deteksi penyakit tanaman
6. Petani menyimpan hasil deteksi	
	7. Sistem menyimpan hasil deteksi

Tabel 3.5 *Use case scenario* melihat hasil *scan*

Petani	**Sistem**
1. Petani membuka aplikasi CornAnalyze	
2. Petani memilih menu *history*	
	3. Sistem menampilkan hasil *scan* yang disimpan sebelumnya
4. Petani memilih salah satu hasil scan	
	5. Sistem menampilkan informasi dari hasil scan tersebut

3.4.3. *Activity Diagram*

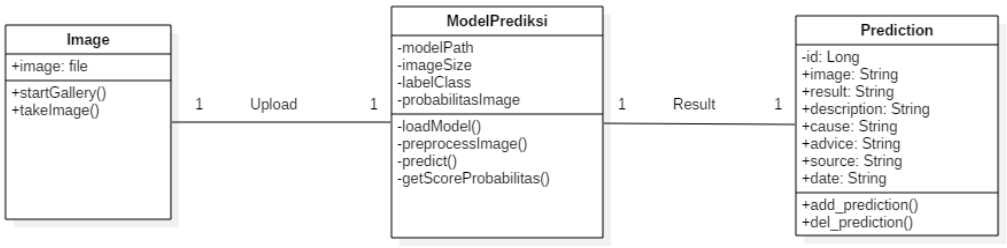


Gambar 3.19 *Activity diagram*

Berdasarkan gambar 3.19 proses dimulai dengan petani membuka aplikasi kemudian sistem merespon dengan menampilkan halaman *home*, lalu petani membuka menu *scan* selanjutnya aplikasi menampilkan halaman *scan* kepada petani, setelah itu petani memilih opsi untuk mengambil gambar menggunakan kamera atau menggunakan gambar yang ada pada galeri *handphone*, dari masing-

masing opsi tersebut aplikasi merespon dengan membuka kamera atau memilih untuk mengakses galeri. Selanjutnya petani menekan tombol analisa yang terdapat pada menu, kemudian *system* mulai mendeteksi dari gambar yang dimasukkan oleh petani, setelah itu sistem aplikasi menampilkan hasil *scanning* dan solusi atau saran, terakhir petani menekan tombol simpan untuk menyimpan hasil deteksi, kemudian sistem menyimpan hasil tersebut.

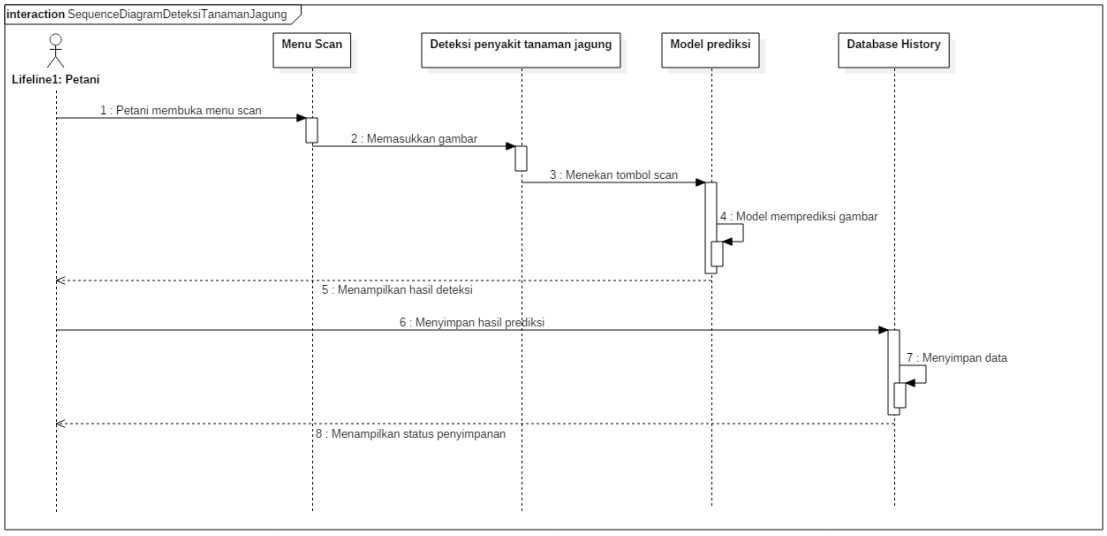
3.4.4. *Class Diagram*



Gambar 3.20 *Class diagram*

Pada gambar 3.20 *Class diagram*, *class image* memiliki metode membuka galeri ataupun menggambil gambar, gambar tersebut kemudian di upload ke ModelPrediksi melalui *modelPath* untuk mendapatkan *probabilitasImage* pada ModelPrediksi memiliki metode *loadModel* (memuat model), *preprocessImage* (memproses gambar), *prediksi* (mendapatkan hasil prediksi), *getScoreProbabilitas* (mendapatkan skor probabilitas). Dari ModelPrediksi akan didapatkan hasilnya dan kemudian menuju *class Prediction*, data yang dihasilkan berupa id, *image*, *result*, *description*, *cause*, *advice*, *source*, dan *date*, metode yang dapat digunakan pada *class Predicition* adalah *add_prediction* (menambahkan prediksi), dan *del_prediction* (menghapus prediksi).

3.4.5. *Sequence Diagram*



Gambar 3.21 *Sequence diagram*

Pada gambar 3.21 *Sequence diagram* alur deteksi tanaman jagung yang dilakukan oleh petani dengan memasukkan gambar yang kemudian menekan tombol *scan*, selanjutnya sistem akan memproses pada model prediksi. Jika sudah dilakukan proses prediksi maka akan ditampilkan hasil deteksi ke petani, petani selanjutnya menyimpan hasil prediksi pada database, dan database menyimpan data tersebut kemudian akan menampilkan status penyimpanan.

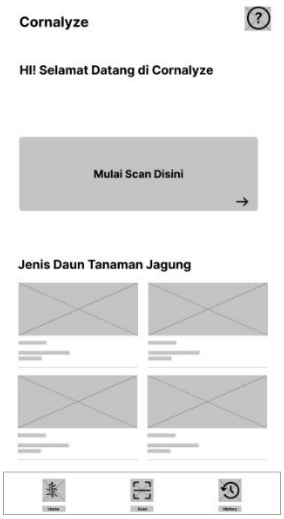
3.5. Rancangan Antar Muka

3.5.1. Halaman *Splash Screen*

Gambar 3.22 *Wireframe splash screen*

Rancangan *splash screen* ini merupakan halaman *loading* awal setelah aplikasi ditekan, halaman ini akan muncul selama beberapa detik yang kemudian akan berpindah otomatis ke halaman *home*.

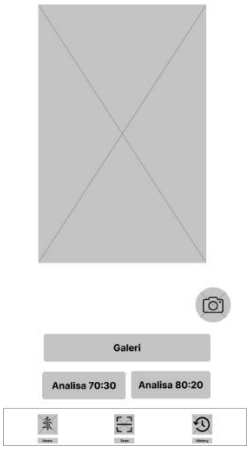
3.5.2. Halaman *Home*



Gambar 3.23 *Wireframe home*

Halaman *home* ialah halaman awal daripada aplikasi CornAnalyze, halaman ini nantinya akan berisikan informasi cara penggunaan aplikasi, langsung *scan*, dan informasi mengenai jenis-jenis penyakit jagung.

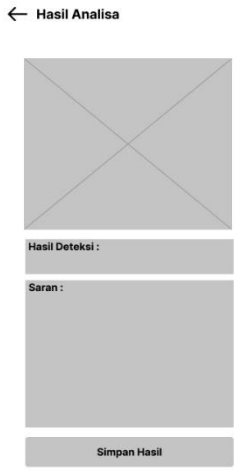
3.5.3. Halaman *Scan*



Gambar 3.24 *Wireframe scan*

Halaman *scan* merupakan halaman yang digunakan untuk melakukan *scan* pada tanaman jagung dapat mengambil melalui galeri yang ada di *handphone* ataupun menggunakan kamera yang disediakan pada aplikasi.

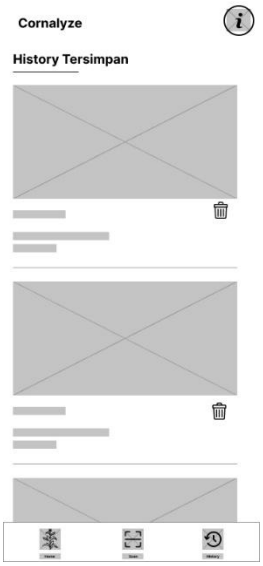
3.5.4. Halaman Hasil



Gambar 3.25 *Wireframe* hasil

Rancangan antar muka hasil merupakan halaman yang berisi hasil dari *scan* pada halaman *scan*, nantinya hasil scan akan muncul di halaman hasil yang akan berisi hasil deteksi dan juga saran.

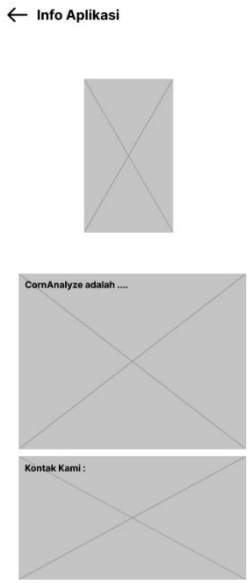
3.5.5. Halaman *History* Tersimpan



Gambar 3.26 *Wireframe history* tersimpan

Rancangan antar muka *history* merupakan halaman yang berisi dari hasil-hasil *scan* yang sebelumnya *user* simpan, halaman *history* nantinya dapat dilihat lagi oleh para user untuk mengetahui hasil-hasil *scan* sebelumnya yang sudah *user* simpan.

3.5.6. Halaman Info Aplikasi



Gambar 3.27 *Wireframe* info aplikasi

Halaman setting merupakan halaman yang dapat digunakan oleh para *user* untuk melihat tentang aplikasi CornAnalyze dan kontak.