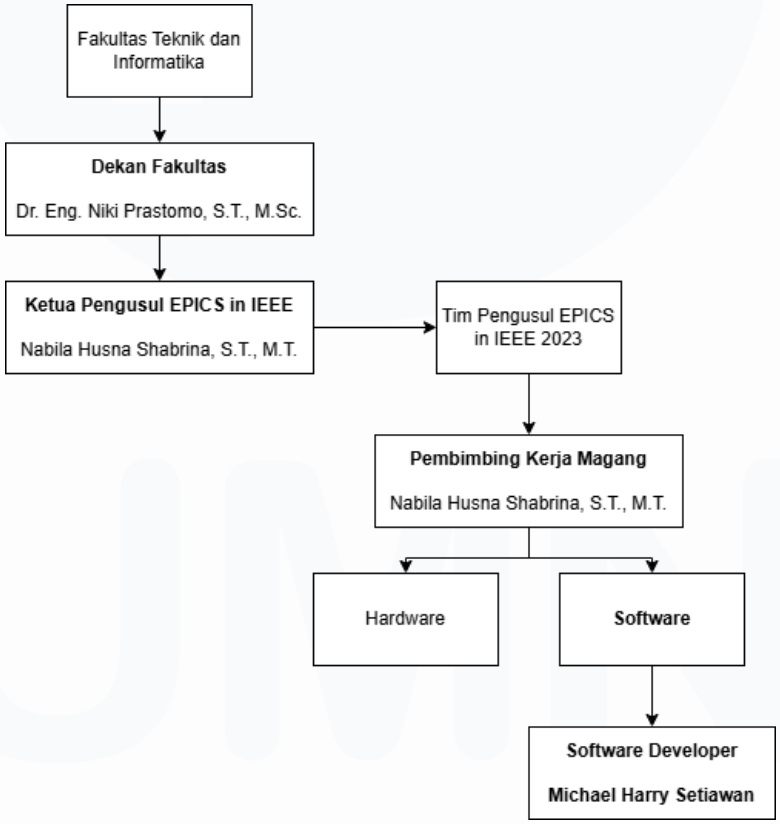
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

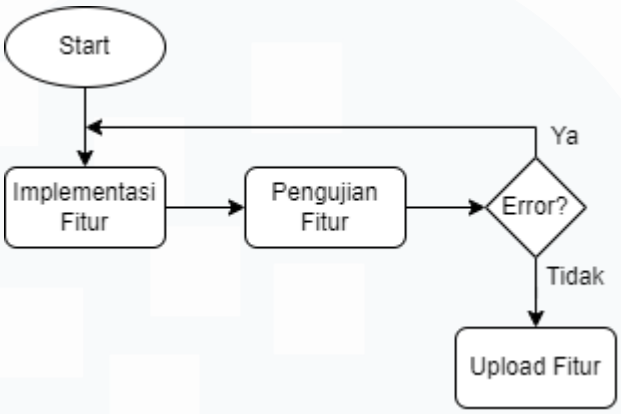
Dalam proyek ini ada dua divisi utama yang tersedia, yaitu divisi *hardware* dan divisi *software*. Hirarki dapat dilihat lebih jelas dari Gambar 3.1, penulis berada pada divisi *software* tepatnya sebagai *software developer* dengan fokus utama yaitu mengembangkan sebuah aplikasi untuk digunakan oleh para petani. *Software developer* disini mencakup *frontend* sebagai tampilan dari aplikasi serta *backend* yang melakukan proses pada aplikasi.



Gambar 3.1 Hirarki kedudukan pemagang

Proses pengembangan sebuah aplikasi tentunya banyak melalui tahap *trial and error*, yaitu diawali dengan membuat sebuah fitur baru, lalu menguji fitur tersebut untuk mencari tahu apakah masih terdapat error atau tidak, jika masih terdapat error maka fitur akan diubah dan diujikan kembali. Proses tersebut akan berlangsung

sampai fitur terimplementasikan dengan baik dan memiliki error sekecil mungkin, proses dapat dilihat lebih jelas pada Gambar 3.2.



Gambar 3.2 Flowchart alur kerja

3.2 Tugas dan Uraian Kerja Magang

3.2.1 Tugas dan Uraian Kerja Magang

Tabel 3.1 menjelaskan kegiatan yang dilakukan selama periode magang, dimulai dari November 2024 hingga pembuatan laporan pada Februari 2025.

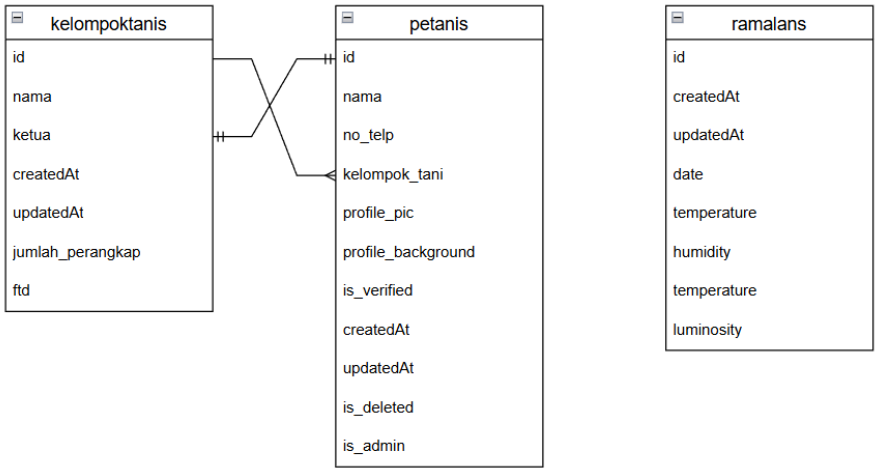
Tabel 3.1 Timeline kerja magang

Bulan	Minggu	Kegiatan
November	1	Rapat bersama tim untuk mendiskusikan fitur-fitur yang penting untuk diimplementasikan.
	2	Pembuatan page verifikasi petani. Ini merupakan menu yang bisa diakses oleh ketua kelompok tani untuk memverifikasi petani yang mendaftar dalam kelompok tani tersebut.
	3	Pembuatan fitur tambah kurang jumlah perangkap. Ini merupakan fitur ketua tani untuk menambah atau mengurangi jumlah perangkap pada kebun.

	4	Implementasi model kecerdasan buatan LightGBM. Implementasi pada Flask, Express.js dan tampilan pada React.js
Desember	1	Lanjutan pengimplementasian model kecerdasan buatan.
	2	Implementasi perhitungan FTD. Untuk menampilkan FTD pada peta persebaran serta manajemen hama.
	3	Implementasi cron *scheduler* untuk mengatur jadwal aplikasi melakukan ramalan cuaca dan *socket* agar beberapa fitur dapat berinteraksi secara *real-time*.
	4	Memperbaiki antarmuka pengguna, memperbaiki bug.
Januari	1-3	Finalisasi aplikasi, bug fixing, dan launching aplikasi di Sleman.
	4	Memperbaiki bug yang ditemukan setelah aplikasi di gunakan oleh beberapa petani.
Februari	1-4	Periode pembuatan laporan magang.

Penulis memiliki beberapa tanggung jawab dalam pengembangan aplikasi MySalak. Yang pertama adalah pembuatan verifikasi petani yang bisa dilakukan oleh ketua tani untuk memverifikasi petani yang mendaftar, fitur tambah kurang jumlah perangkap untuk ketua tani mengubah jumlah perangkap yang dipasang pada kebun guna melakukan perhitungan FTD, implementasi model kecerdasan buatan untuk melakukan ramalan cuaca mulai dari *backend* sampai *frontend*, implementasi perhitungan FTD dan cron *scheduler* untuk menjadwalkan peramalan cuaca. Gambar 3.3 merupakan skema *Entity Relation Diagram* (ERD) atau skema dari tabel yang relevan, dalam tabel petanis dapat dilihat terdapat kolom *is_verified*

yang akan diubah ketika petani di verifikasi. Pada tabel kelompoktanis dapat dilihat jumlah perangkap yang bisa diubah jumlahnya dan nanti akan digunakan dalam perhitungan FTD pada kelompok tani, dan pada tabel ramalans akan berisikan data seperti *date*, temperature, *humidity*, *precipitation* dan *luminosity*. Dapat dilihat juga relasi antar tabel, 1 kelompok tani bisa memiliki 1 ketua dan 1 kelompok tani dapat memiliki beberapa petani.

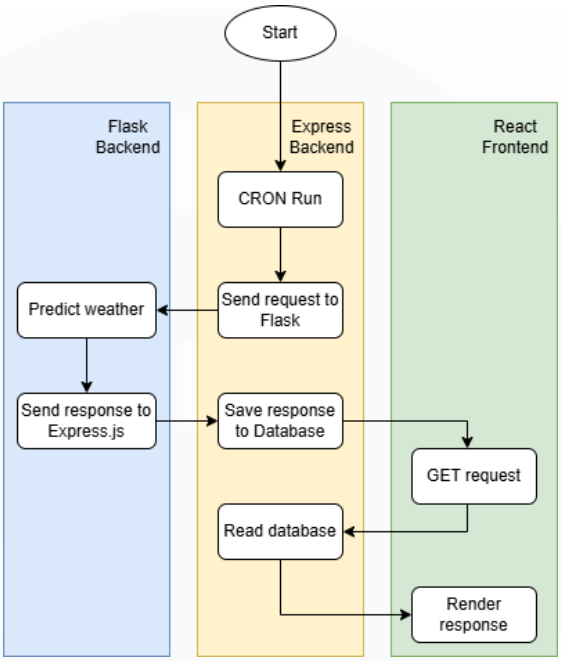


Gambar 3.3 Skema relasi tabel petani, kelompok tani dan hasil ramalan

3.2.2 Bahasa Pemrograman

Bahasa pemrograman yang digunakan adalah Javascript dan Python, Javascript digunakan untuk mengembangkan *frontend* menggunakan *framework* React.js [2] dan *backend* server menggunakan *framework* Express.js [3] serta Python digunakan sebagai *backend* untuk memuat model kecerdasan buatan menggunakan *framework* Flask [4]. Javascript dipilih karena kompatibilitasnya untuk digunakan sebagai *Progressive Web Applications* (PWA), PWA dipilih karena keterbatasan waktu pengembangan yang tersedia dan dengan pertimbangan bahwa *handphone* yang digunakan petani beragam dan sudah cukup tua. Python digunakan untuk memuat model kecerdasan buatan karena pengembangan model kecerdasan buatan juga menggunakan Python.

3.2.3 Integrasi Prediksi Cuaca



Gambar 3.4 *Flow diagram* integrasi prediksi cuaca

Gambar 3.4 merupakan alur program melakukan prediksi cuaca, pertama dimulai dengan cron yang berjalan setiap jam 12 malam di Express *backend*, cron dijalankan pada Express karena Flask digunakan hanya untuk memuat model dan semua *logic* dalam aplikasi diatur melalui Express, lalu akan mengirimkan *request* ke Flask untuk melakukan prediksi sesuai tanggal dan hasilnya akan dikirimkan kembali ke Express untuk disimpan ke dalam database. React akan mengirimkan *request* ke Express untuk mengambil data yang diperlukan untuk di *render* ke dalam tampilan. Cron merupakan sebuah alat pada Linux untuk membuat penjadwalan otomatis untuk melakukan sebuah tugas [5].

Model kecerdasan buatan yang sudah dikembangkan sebelumnya akan dimuat dalam server Flask, dapat dilihat pada Gambar 3.5 ada 4 model kecerdasan buatan yang dimuat untuk melakukan prediksi terhadap masing-masing variabel cuaca. Model kecerdasan buatan dikembangkan oleh penulis, menerima *input* berupa fitur dari tanggal dan akan mengeluarkan *output* berupa variabel sesuai dengan model yang digunakan.

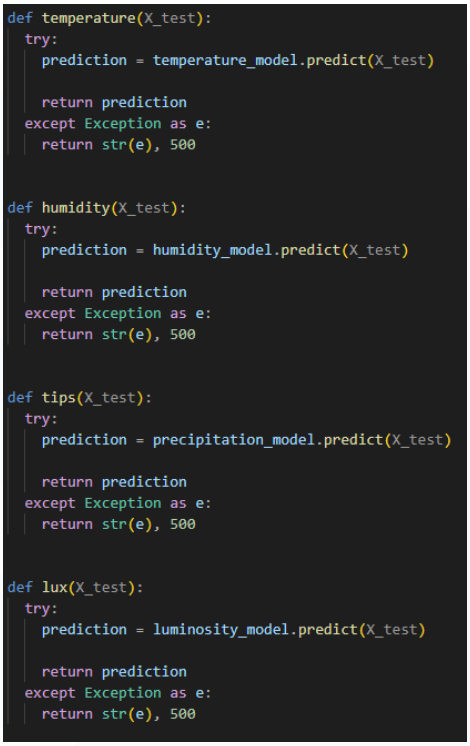
```
base_path = os.path.join(os.path.dirname(__file__), 'Model', 'LightGBM')

temperature_model = lgb.Booster(model_file=os.path.join(base_path, 'temp_lgbm.txt'))
humidity_model = lgb.Booster(model_file=os.path.join(base_path, 'humidity_lgbm.txt'))
precipitation_model = lgb.Booster(model_file=os.path.join(base_path, 'precip_lgbm.txt'))
luminosity_model = lgb.Booster(model_file=os.path.join(base_path, 'lux_lgbm.txt'))
```

Gambar 3.5 Memuat model kecerdasan buatan

```
def pred_result_lightgbm():
  new_val = request.json

  new_df = pd.DataFrame({
    'Tanggal': [pd.to_datetime(new_val['createdAt'])+pd.Timedelta(hours=7)]
  })

  new_df.set_index('Tanggal', inplace=True)
  new_df = new_df.resample('h').nearest()
  one_week = pd.DataFrame({'Tanggal':pd.date_range(start=new_df.index[0], periods=168, freq='h')})
  one_week.set_index('Tanggal', inplace=True)

  one_week["day_of_week"] = one_week.index.dayofweek
  one_week["day_of_year"] = one_week.index.dayofyear
  one_week["hour"] = one_week.index.hour
  one_week["minutes"] = one_week.index.minute
  one_week["day"] = one_week.index.day
  one_week["month"] = one_week.index.month
  one_week["quarter"] = one_week.index.quarter
  one_week["year"] = one_week.index.year

  cols = [col for col in one_week.columns if col not in ['humidity', 'temperature', 'tips', 'lux', 'Tanggal']]
  X_test = one_week[cols]

  res = {
      'date': pd.date_range(start=new_df.index[0], periods=168, freq='h').strftime('%Y-%m-%d %H:%M:%S'),
      'temperature': temperature(X_test),
      'humidity': humidity(X_test),
      'precipitation': tips(X_test),
      'luminosity': lux(X_test)
    }
  data_serializable = {key: value.tolist() for key, value in res.items()}
  return json.dumps(data_serializable)
```

Gambar 3.6 Sintaks lengkap prediksi LightGBM

Gambar 3.7 menunjukkan cara kerja server Flask untuk melakukan ramalan cuaca, untuk melakukan prediksi menggunakan model LightGBM, variabel yang diperlukan hanya tanggal yang nantinya akan diekstraksi fiturnya seperti pada Gambar 3.6. Model lalu akan melakukan prediksi berdasarkan fitur-fitur yang sudah di ekstraksi seperti pada Gambar 3.7. Model akan melakukan prediksi sebanyak 168 periode atau ekuivalen dengan 7 hari.



```python
def temperature(X_test):
  try:
    prediction = temperature_model.predict(X_test)

    return prediction
  except Exception as e:
    return str(e), 500


def humidity(X_test):
  try:
    prediction = humidity_model.predict(X_test)

    return prediction
  except Exception as e:
    return str(e), 500


def tips(X_test):
  try:
    prediction = precipitation_model.predict(X_test)

    return prediction
  except Exception as e:
    return str(e), 500


def lux(X_test):
  try:
    prediction = luminosity_model.predict(X_test)

    return prediction
  except Exception as e:
    return str(e), 500
```

Gambar 3.7 Fungsi prediksi masing-masing variabel cuaca

Gambar 3.8 merupakan fungsi yang berada di server Express.js yang digunakan untuk mengirimkan *payload* ke server Flask untuk melakukan prediksi. *Truncate* diatas merupakan sebuah fungsi *database* untuk menghapus data dalam sebuah tabel, data dalam tabel akan selalu diperbaharui setiap fungsi prediksi ini dipanggil, maka dari untuk dilakukan *truncate* agar database tidak penuh. Setelah menerima respon dari server Flask, server Express akan menyimpan hasil prediksi ke *database*.

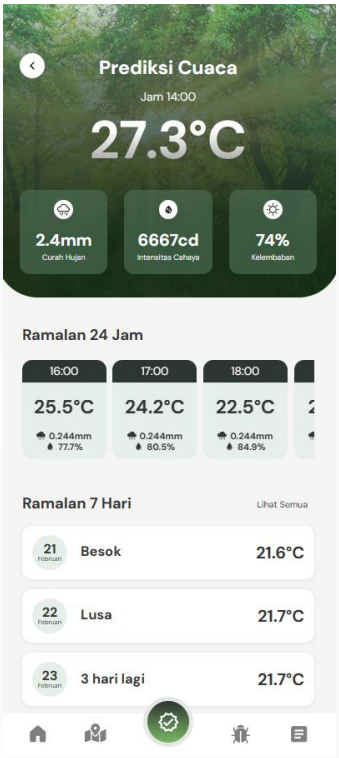
```
const makeForecast = async () => {
  const truncateStat = await Ramalan.findAll();
  if (truncateStat) await Ramalan.truncate();

  const payload = {
    createdAt: new Date(),
  };

  const response = await axios.post(lgbm_url, payload);
  const predArr = response.data;

  for (let i = 0; i < predArr.date.length; i++) {
    await Ramalan.create({
      id: i + 1,
      date: predArr.date[i],
      temperature: predArr.temperature[i],
      humidity: predArr.humidity[i],
      precipitation: predArr.precipitation[i],
      luminosity: predArr.luminosity[i],
    });
  }
}
```

Gambar 3.8 Fungsi memanggil fungsi prediksi LightGBM

```
cron.schedule('0 0 * * *', async () => {
  console.log('forecast cron scheduler active')
  await makeForecast();
})
```

Gambar 3.9 Cron *scheduler* untuk melakukan ramalan cuaca

Awalnya aplikasi akan melakukan prediksi setiap diterima data dari node hardware, namun karena node rentan tidak mengirimkan data cara tersebut menjadi kurang optimal. Maka dari itu, penulis mengusulkan penggunaan cron untuk melakukan prediksi. Cron merupakan fungsi penjadwalan, yaitu untuk menjalankan fungsi sesuai dengan jadwal yang sudah ditentukan. Gambar 3.9 menunjukkan cron yang digunakan dalam aplikasi MySalak, schedule ‘0 0 * * *’ berarti fungsi cron ini akan aktif setiap jam 12 malam setiap harinya. Karena model kecerdasan buatan tidak memerlukan variabel cuaca untuk melakukan prediksi, cron ini menjadi sebuah solusi terhadap kondisi node yang tidak mengirimkan data.

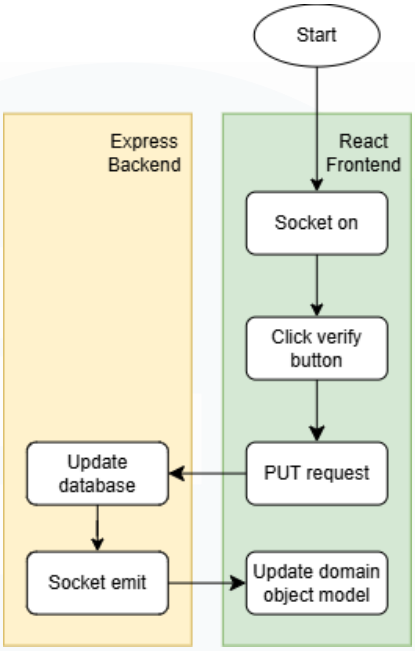


Gambar 3.10 Tampilan frontend ramalan cuaca

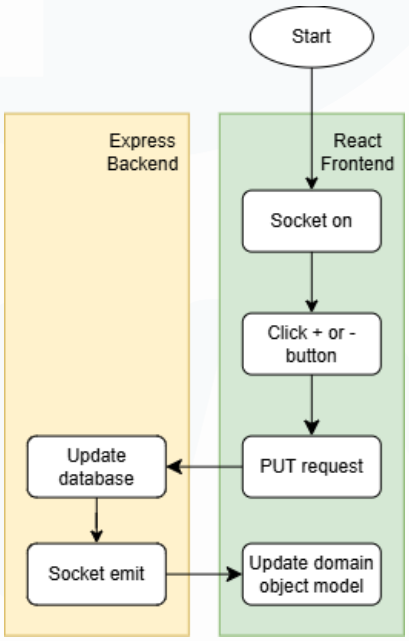
Gambar 3.10 merupakan contoh tampilan dari ramalan cuaca aplikasi MySalak, dapat dilihat terdapat ramalan 24 jam dan ramalan 7 hari dalam aplikasi.

3.2.4 Fitur Verifikasi dan Update Jumlah Perangkap Ketua Tani

Menu *update* jumlah perangkap dan verifikasi berisi fitur yang hanya dapat diakses oleh ketua kelompok tani, menu ketua tani memiliki 2 fitur, yang pertama adalah fitur menambah atau mengurangi jumlah perangkap pada kebun, jumlah perangkap harus di tentukan karena akan digunakan dalam penghitungan FTD. Yang kedua adalah fitur verifikasi petani, petani yang mendaftar ke kebun tertentu perlu diverifikasi terlebih dahulu oleh ketua tani agar bisa mengakses fiturnya secara lengkap, hal ini dibuat untuk memitigasi orang asing masuk dan melakukan perubahan dalam kelompok tani tertentu.



Gambar 3.11 *Flow diagram* verifikasi petani



Gambar 3.12 *Flow diagram* update jumlah perangkap

Gambar 3.11 merupakan alur kerja dari fitur verifikasi petani, dimulai dengan *socket* yang di *instantiate* atau dimulai ketika membuka laman, ketua tani yang mengetuk tombol untuk memverifikasi petani sehingga React akan mengirimkan *request* ke Express untuk melakukan *update*

terhadap petani yang diverifikasi. *Socket* digunakan supaya interaksi antara Express dan React terjadi secara *real-time,* Socket.IO merupakan sebuah library *socket* yang memungkinkan komunikasi *real-time* dua arah berbasis *event* [6]. Koneksi client ditempatkan pada App.js atau pada aplikasi ketika memuat aplikasi, *socket* akan menjalankan fungsi *emit* pada Express, untuk mengirimkan data dan akan diterima oleh React menggunakan fungsi *on* melalui useEffect dengan dependensi *socket* untuk membatasi perubahan terjadi hanya ketika terjadi perubahan pada socket. Untuk membedakan data dari *socket* yang berbeda, diberikan nama event yang berbeda misalnya kelompok-updated dan verify updated, *real-time update* diperlukan supaya petani dapat melihat langsung perubahan yang dibuat pada aplikasi tanpa perlu melakukan refresh page. Untuk pengembangan kedepannya, *socket* bisa digunakan jika terjadi perubahan ketua tani atau ada penambahan penanggung jawab kedua pada kebun supaya ketika salah seorang melakukan perubahan bersamaan, perubahan dapat dilihat secara langsung.

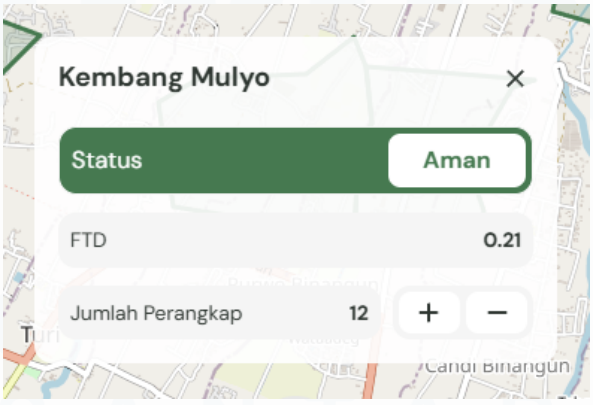
Gambar 3.12 merupakan alur kerja dari fitur *update* jumlah perangkap, secara umum alurnya serupa dengan fitur verifikasi petani, yang membedakan hanyalah fungsi dan *Application Programming Interface* (API) *endpoint* yang digunakan pada Express. Dimulai dengan inisialisasi *socket* ketika laman dibuka ketua tani yang mengetuk tombol untuk tambah atau kurang, sehingga React akan mengirimkan *request* ke Express untuk melakukan *update* terhadap jumlah perangkap pada kebun. Lalu supaya langsung terlihat perubahan secara *real-time*, digunakan *socket* untuk menjadi media interaksi antara Express dan React, *socket* akan menjalankan fungsi *emit* pada Express, yaitu untuk mengirimkan data dan data akan diterima oleh React melalui fungsi *on*.

Gambar 3.14 merupakan fitur tambah perangkap yang bisa diakses oleh ketua kelompok tani, fitur ini juga bersinggungan dengan perhitungan FTD, karena jumlah perangkap merupakan salah satu bagian dari formula FTD. Fitur ini juga dibatasi per kelompok tani, artinya ketua kelompok tani

Kembang Mulyo seperti pada gambar diatas tidak bisa mengganti jumlah perangkap dari kelompok tani lainnya. Supaya interaksi bisa terjadi secara *real-time*, fitur ini menggunakan *socket* yang bisa mendukung interaksi agar terjadi secara langsung, implementasinya dapat dilihat lebih lanjut dari Gambar 3.15.

```
{nama === detail.KelompokTani.ketua ? (
  <Flex
    bg={"#F5F5F5"}
    justify={"space-between"}
    justifyItems={"center"}
    p={1}
    h={"inherit"}
    borderRadius={"12px"}
  >
    {isProcessing ? (
      <Spinner justifyItems={"center"} alignContent={"center"} />
    ) : (
      <Flex>
        <Box
          w={"10vw"}
          h={"100%"}
          justifyItems={"center"}
          alignContent={"center"}
          mr={1}
          borderRadius={"8px"}
          bg={"white"}
          onClick={() => incrementJumlahPerangkap()}
          disabled={isProcessing}
        >
          <FaPlus />
        </Box>
        <Box
          w={"10vw"}
          h={"100%"}
          justifyItems={"center"}
          alignContent={"center"}
          borderRadius={"8px"}
          bg={"white"}
          onClick={() => decrementJumlahPerangkap()}
          disabled={isProcessing}
        >
          <FaMinus />
        </Box>
      </Flex>
    )}
  </Flex>
) : (
  <></>
)}
```

Gambar 3.13 Sintaks kode tambah perangkap

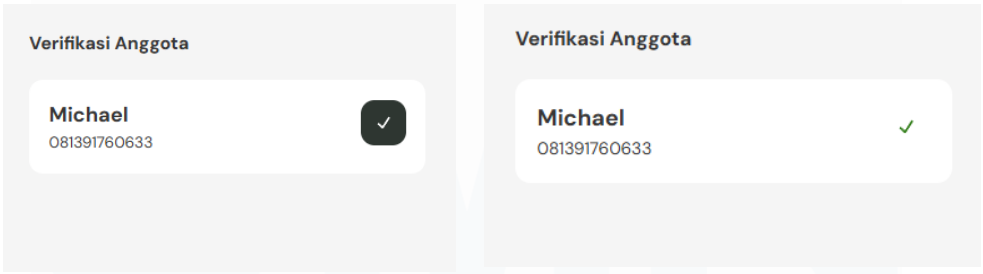


Gambar 3.14 Tampilan fitur tambah perangkap

```
useEffect(() => {
  // console.log("socket triggered");
  socket.on("kelompok-updated", (updatedData) => {
    // console.log(updatedData);
    setDetail((prevState) => ({
      ...prevState,
      KelompokTani: {
        ...prevState.KelompokTani,
        jumlah_perangkap: updatedData.jumlah_perangkap,
        ftd: updatedData.ftd,
      },
    }));
  });

  return () => {
    socket.off("kelompok-updated");
    //   socket.close();
  };
}, [socket]);
```

Gambar 3.15 Implementasi socket untuk fitur jumlah perangkap

Selanjutnya adalah menu verifikasi anggota, disini ketua tani dapat melihat nama dan nomor telepon petani yang mendaftarkan dirinya dalam kelompok tani itu seperti pada Gambar 3.16, ketua tani bisa mengetuk tombol centang untuk memverifikasi petani tersebut. Verifikasi petani dibuat sangat sederhana karena merupakan hasil diskusi dengan kelompok tani, untuk sementara tidak ada cara untuk memastikan apakah orang yang melakukan registrasi adalah benar atau tidak.



(a) (b)

Gambar 3.16 Tampilan verifikasi anggota (a) sebelum verifikasi, (b) sesudah verifikasi

```
<Flex
  direction={"column"}
  w={"85%"}
  mx={"auto"}
  pos={"relative"}
  minH={"100vh"}
  py={"20%"}
>
  <Text mb={5} fontWeight={"bold"}>Verifikasi Anggota</Text>
  {petani.filter(item => item.kelompok_tani === kelompok_tani)
    .map((item) => (
      <Stack
        h={"100%"}
        p={"5%"}
        bgColor={"#FFFFFF"}
        mb={"1.5vh"}
        borderRadius={"xl"}
      >
        <Flex>
          <PetaniCard props={item} />
          <Spacer />
          <Button
            variant={"unstyled"}
            borderRadius={"xl"}
            justifyItems={"center"}
            w={1}
            onClick={!item.is_verified ? () => verify(item.id) : null}
            background={!item.is_verified ? "#2c3631" : "transparent"}
            disabled={item.is_verified}
          >
            <RxCheck
              style={{ color: item.is_verified ? "green" : "white" }} //
              size={20}
            />
          </Button>
        </Flex>
      </Stack>
    ))}
```

Gambar 3.17 Sintaks kode verifikasi petani

```
useEffect(() => {
  socket.on('verify-updated', (updatedData) => {
    console.log(updatedData.resp.id)
    setPetani((prevState) =>
      prevState.map((item) =>
        item.id === updatedData.resp.id ? { ...item, is_verified: true } : item
      )
    );
  });

  return () => {
    socket.off('verify-updated');
    //   socket.close();
  };
}, [socket]);
```

Gambar 3.18 Implementasi socket dalam verifikasi anggota

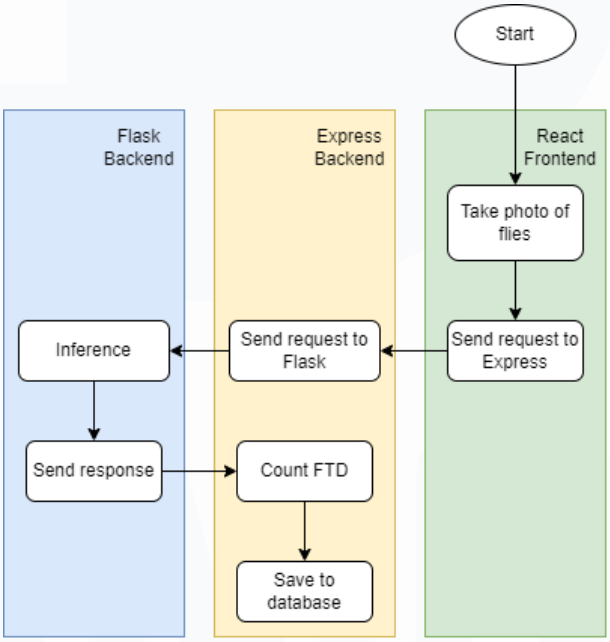
Gambar 3.17 dan Gambar 3.18 merupakan sintaks dari *page* verifikasi petani, urutan petani yang akan ditampilkan dalam menu ini sebatas petani dalam kelompok tani, untuk interaksi juga dibuat supaya terjadi secara *real-time* menggunakan socket, ketika petani mengetuk tombol akan mendapatkan respon secara langsung sehingga tampilannya langsung berubah seperti pada Gambar 3.16.

3.2.5 Integrasi FTD

Flies per Trap per Day (FTD) merupakan sebuah indeks populasi rata-rata lalat [7] dan merupakan pengukuran utama dalam melakukan ekspor impor salak, dalam perhitungan parameter yang penting adalah jumlah lalat tertangkap dan jumlah perangkap dapat dilihat pada rumus dibawah:

$$FTD = \frac{F}{T \times D} \quad (1)$$

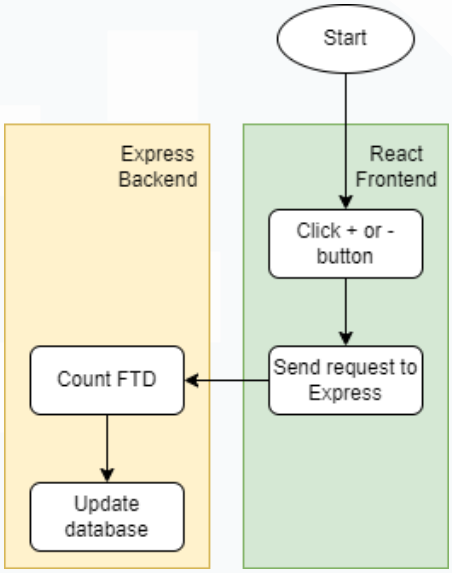
Flies (F) menandakan jumlah lalat buah yang tertangkap, *Trap* (T) menandakan jumlah perangkap yang terpasang pada kebun, dan *Day* (D) jumlah hari perangkap dibiarkan sebelum dihitung jumlah lalat yang masuk.



Gambar 3.19 Flow diagram perhitungan FTD dari foto lalat

Gambar 3.19 merupakan alur ketika melakukan perhitungan FTD melalui foto lalat, pertama pengguna akan mengambil foto lalat dan mengirimkan request ke Express dan Flask untuk melakukan inferensi foto, dan responsnya akan digunakan untuk menghitung FTD dan disimpan ke dalam database. Gambar 3.20 merupakan alur perhitungan FTD ketika ketua tani melakukan update jumlah perangkap, setelah mengubah jumlah perangkap React akan mengirimkan request ke Express, sesuai dengan

jumlah perangkap yang baru akan melakukan perhitungan FTD dan mengupdate database.



Gambar 3.20 Flow diagram perhitungan FTD dari update jumlah perangkap

Gambar 3.21 menunjukkan implementasi rumus yang digunakan untuk melakukan perhitungan FTD dalam aplikasi, yaitu jumlah lalat buah yang tertangkap dibagi jumlah perangkap kali 7 hari.

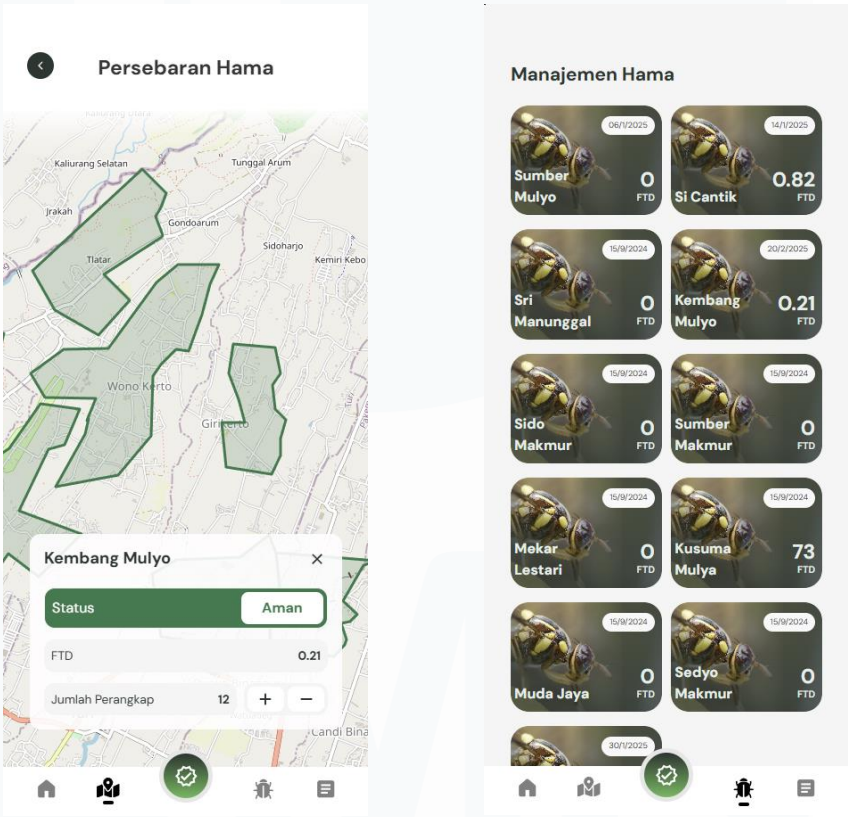
```
const calculateFtd = (jumlah, perangkap) => {
  const ftd = parseFloat(jumlah / (perangkap * 7)).toFixed(2);

  return ftd;
};
```

Gambar 3.21 Formula perhitungan FTD

Karena terdapat beberapa perangkap yang dipasang dalam satu kebun, maka dalam perhitungan FTD perlu mengambil jumlah lalat dari semua perangkap. Dapat dilihat dari Gambar 3.22, data diambil dari *StartOfWeek* dan *EndOfWeek*, yaitu akan mengambil data dalam rentan 1 minggu dan akan dijumlahkan semua pada variabel total sebelum akhirnya masuk ke dalam rumus perhitungan FTD.

```
let date = moment.tz(data.KelompokTani.updatedAt, "Asia/Jakarta")
let startOfWeek = date.startOf('week').tz("Asia/Jakarta").format()
let endOfWeek = date.endOf('week').tz("Asia/Jakarta").format()

const r1 = await TangkapanHama.findAll({
  where: {
    createdAt: {
      [Op.between]: [new Date(startOfWeek), new Date(endOfWeek)]
    },
    id_kelompok_tani: data.KelompokTani.id
  }
})

const total = r1.reduce((sum, record) => sum + record.jumlah, 0)

const ftd = ftdCalc(total, newJumlah)
```

Gambar 3.22 Alur perhitungan FTD



(a) (b)

Gambar 3.23 Contoh tampilan dari perhitungan FTD, (a) peta sebaran, (b) manajemen hama

Gambar 3.23 menunjukkan contoh penggunaan FTD dalam aplikasi, yang pertama ada pada peta sebaran hama dan yang kedua pada manajemen hama.

3.2.6 Pengembangan Lebih Lanjut

Berhubungan magang ini merupakan proyek yang masih berlanjut, berikut adalah beberapa saran pengembangan lebih lanjut:

1. Dalam pengembangan *software*, terutama bagian bagian *machine learning* bisa mencari data berupa jumlah tangkapan hama untuk digunakan dalam prediksi, sehingga tidak hanya memprediksi cuaca saja, tetapi terdapat juga model untuk memprediksi jenis dan jumlah hamanya.
2. Untuk pengembangan *hardware*, ditegaskan untuk pelaksanaan pengujian yang lebih ketat, contohnya dicoba dinyalakan sekitar 1 bulan di sekitar area kampus terlebih dahulu, hal ini bertujuan untuk mengantisipasi jika error terjadi sehingga perbaikan dapat lebih mudah dilakukan.

3.3 Kendala yang Ditemukan

Beberapa fitur diatas dikembangkan karena ditemukan kendala dalam aplikasi, yaitu:

1. Tidak terdapat mentor dan terbatasnya pengetahuan.
2. Jika node mati aplikasi tidak akan melakukan prediksi kondisi cuaca.
3. Tambah perangkap dan verifikasi petani tidak *real-time*, sehingga setelah mengetuk tombol perlu *refresh* terlebih dahulu agar hasilnya terlihat.

3.4 Solusi atas Kendala yang Ditemukan

Berdasarkan kendala yang sudah dijabarkan, berikut adalah solusi yang ditemukan:

1. Banyak mencari contoh dan melihat dokumentasi penggunaan secara *detail*.

2. Ramalan cuaca diubah yang awalnya menunggu data dari node, menjadi menggunakan tanggal hari ini. Ramalan ini dijadwalkan menggunakan cron *scheduler* setiap jam 12 malam.
3. Mengimplementasikan socket agar interaksi API bisa menjadi *real-time* tanpa perlu *refresh page* untuk melihat perubahan yang sudah terjadi.