BAB III
IMPLEMENTASI

3.1. Tempat Kegiatan Implementasi

Proyek ini dikerjakan dari rumah oleh penulis, dengan seluruh proses pengembangan aplikasi, mulai dari *backend*, *frontend*, hingga persiapan untuk distribusi, dilakukan menggunakan perangkat dan infrastruktur pribadi. Pemilihan tempat ini didasari oleh fleksibilitas yang dibutuhkan untuk menyelesaikan pengembangan dengan efisien.

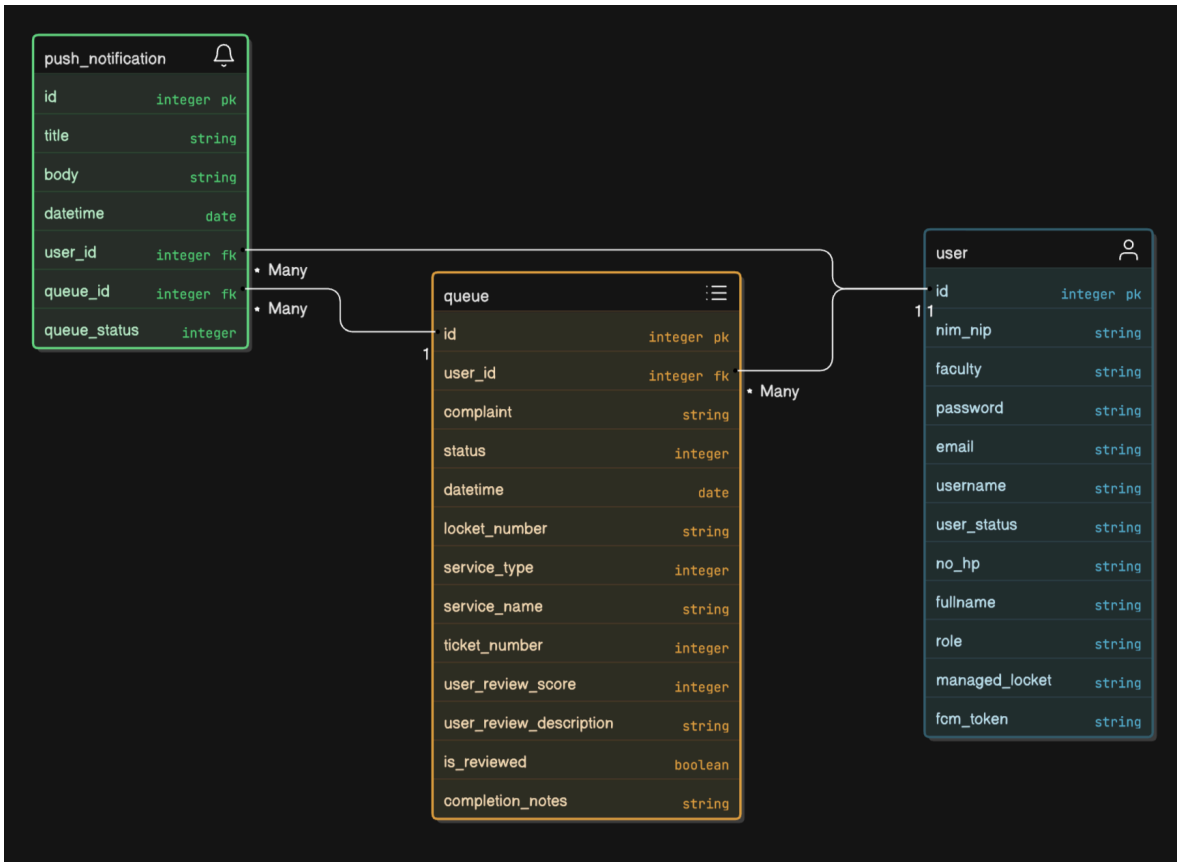
Setelah pengembangan selesai, proses penyerahan dan perilisan dilakukan di UPA TIK UPN "Veteran" Jakarta, yang berperan dalam penyediaan infrastruktur hosting untuk aplikasi *web* dan memastikan aplikasi *mobile* dapat diakses melalui Google Play Store untuk perangkat Android.

Untuk pengujian aplikasi, tempat yang dipilih adalah Unit Layanan Terpadu (ULT) UPNVJ. ULT dipilih karena aplikasi yang dikembangkan adalah aplikasi manajemen antrian yang dirancang khusus untuk meningkatkan efisiensi dan kenyamanan dalam mengelola antrian layanan di ULT. Oleh karena itu, ULT merupakan lokasi yang ideal untuk menguji fungsionalitas aplikasi secara langsung dalam lingkungan penggunaannya. Kolaborasi dilakukan dengan pihak Humas UPNVJ untuk memastikan pengujian aplikasi berjalan sesuai dengan kebutuhan operasional di lapangan.

3.2. Metadata

Metadata adalah informasi yang memberikan deskripsi mengenai data lainnya, termasuk identifikasi, deskripsi, dan lokasi konten data. Peran metadata sangat penting dalam menjaga integritas, pengendalian, serta manajemen risiko yang berkaitan dengan konten digital. Pengelolaan metadata diperlukan untuk menjamin perlindungan, identifikasi, dan keterjangkauan konten digital.

Dalam pengembangan sistem pendaftaran dan pemantauan antrian di ULT UPNVJ, metadata memiliki beberapa peran penting. Metadata membantu memastikan bahwa informasi dari berbagai sumber, seperti data pengunjung, notifikasi, dan antrian, dapat diakses dan digunakan secara terintegrasi oleh berbagai pihak, seperti pengunjung ULT, dan staf admin ULT. Berikut adalah metadata yang digunakan dalam pengembangan sistem pendaftaran dan pemantauan antrian di ULT UPNVJ.



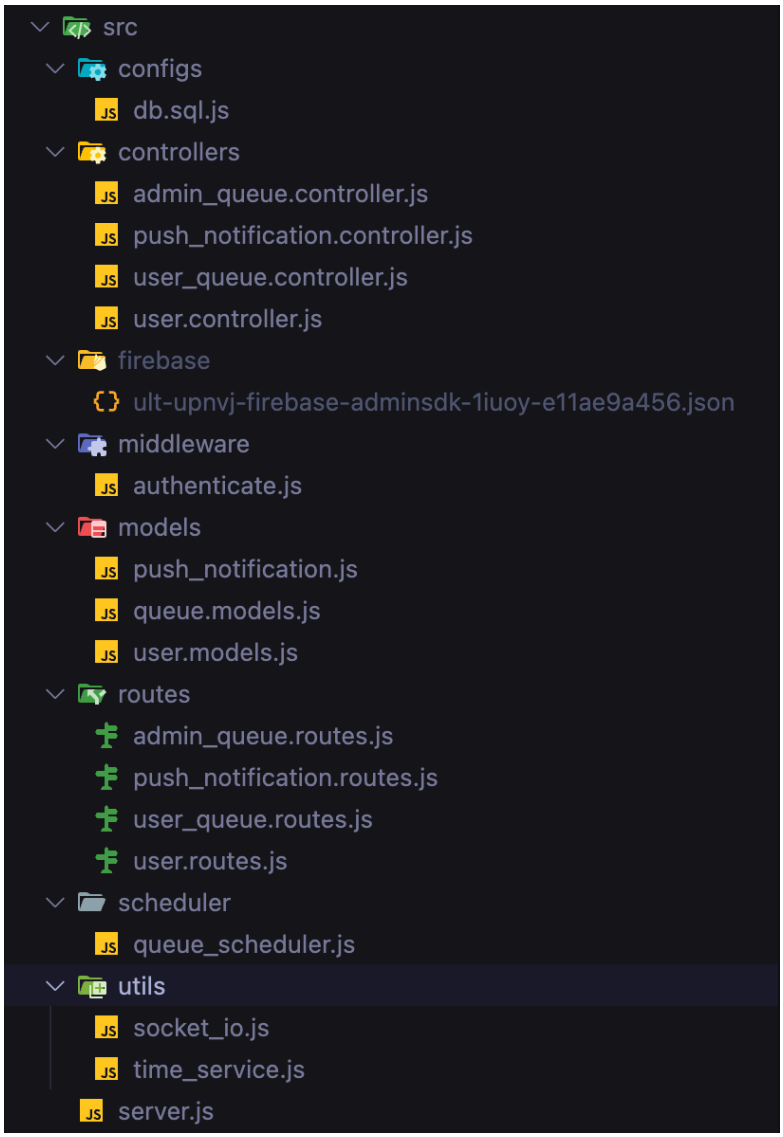
Gambar 3. 1 *Metadata* sistem pendaftaran dan monitoring antrian ULT UPNVJ

3.3. Teknik Implementasi

Pada tahap ini dilakukannya proses implementasi aplikasi yang terdiri dari implementasi pengkodean Backend, implementasi pengkodean Frontend Web, dan implementasi pengkodean Frontend Mobile.

3.3.1. Implementasi Pengkodean *Backend*

Pada tahap ini dilakukannya proses pembuatan *backend* aplikasi dengan menggunakan teknologi yang diperlukan diantaranya yaitu *framework Express.Js* dengan menggunakan *library Socket.io* dan *library Firebase* dari bahasa pemrograman *Javascript* untuk memastikan bahwa semua data dan komunikasi yang diperlukan oleh aplikasi sisi *Frontend* dapat ditangani dengan cepat dan efisien. Sistem real-time yang dikembangkan dengan Socket.io dan Firebase memastikan bahwa perubahan yang terjadi pada antrian dapat langsung direspons oleh server dan diperbarui secara langsung pada antarmuka pengguna. berikut adalah tahap-tahap pengkodean secara umum.



Gambar 3. 2 Struktur *Folder* Aplikasi *Backend*

Pada aplikasi ini, peneliti memisahkan setiap komponen fungsional aplikasi backend ke dalam beberapa folder untuk menjaga keteraturan kode, memudahkan pengujian, serta mempermudah pemeliharaan di masa mendatang. Setiap folder pada struktur ini memiliki peran penting dalam membagi tanggung jawab sesuai dengan fungsinya. Berikut uraian dari setiap folder yang ada pada aplikasi *backend*:

A. Folder configs

berfungsi sebagai tempat untuk file konfigurasi database. File **db.sql.js** di dalamnya mengatur koneksi ke database SQL serta menangani query dasar yang mungkin dibutuhkan dalam aplikasi. Dengan memisahkan konfigurasi ini, aplikasi dapat dengan mudah dikonfigurasi ulang tanpa mengubah file kode utama.

B. Folder controllers

berisikan logika bisnis atau proses dari aplikasi dimana *Controller* bertugas untuk menerima permintaan dari klien (*request*), memprosesnya, dan kemudian mengirimkan tanggapan (*response*).

C. Folder firebase

berisikan kredensial dari Firebase Admin SDK yang memungkinkan aplikasi untuk mengirim *push notifications* dan mengakses layanan Firebase. Ini digunakan untuk komunikasi real-time dan layanan pengiriman pesan kepada pengguna.

D. Folder middleware

sebagai tempat untuk logika perantara yang berfungsi memverifikasi dan memproses permintaan sebelum diteruskan ke *controller*.

E. Folder models

berisikan representasi struktur data pada *table database* yang digunakan di aplikasi.

F. Folder routes

bertanggung jawab untuk mendefinisikan endpoint API yang digunakan oleh aplikasi. Setiap rute bertugas menghubungkan request dari klien ke controller yang sesuai.

G. Folder scheduler

digunakan untuk mengelola tugas-tugas terjadwal yang akan selalu dijalankan pada waktu tertentu pada aplikasi.

```
import { Model, DataTypes } from "sequelize";
import { sequelize } from "../configs/db.sql.js";
import User from "./user.models.js";

export class Queue extends Model {}
Queue.init(
  {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
    complaint: {
      type: DataTypes.STRING,
    },
    status: {
      type: DataTypes.INTEGER,
    },
    user_id: {
      type: DataTypes.INTEGER,
      references: {
        model: "user",
        key: "id",
      },
    },
    datetime: {
      type: DataTypes.DATE,
    },
    locket_number: {
      type: DataTypes.STRING,
    },
    service_type: {
      type: DataTypes.INTEGER,
    },
    service_name: {
      type: DataTypes.STRING,
    },
    ticket_number: {
      type: DataTypes.INTEGER,
    },
    user_review_score: {
      type: DataTypes.INTEGER,
    },
    user_review_description: {
      type: DataTypes.STRING,
    },
    is_reviewed: {
      type: DataTypes.BOOLEAN,
    },
    completion_notes: {
      type: DataTypes.STRING,
    },
  },
  {
    timestamps: true,
    sequelize,
    modelName: "ticket_antrian",
    tableName: "ticket_antrian",
    schema: "antrian_loket",
  }
);

Queue.belongsTo(User, { as: "user", foreignKey: "user_id" });
User.hasMany(Queue, { as: "queue", foreignKey: "user_id" });

export default Queue;
```

Gambar 3. 3 Kode Model Queue

Gambar 3.2 ini mendefinisikan model Sequelize yang disebut `Queue` yang merepresentasikan sebuah tabel dalam database. Model ini memiliki berbagai field seperti `id`, `complaint`, `status`, `user_id`, `datetime`, `locket_number`, `service_type`, `service_name`, `ticket_number`, `user_review_score`, `is_reviewed`, dan `completion_notes`. Metode `init` digunakan untuk

mendefinisikan struktur tabel dalam database. Bidang-bidang didefinisikan menggunakan `DataTypes` yang disediakan oleh Sequelize. Properti `references` digunakan untuk membuat hubungan kunci asing antara bidang `user_id` di tabel `Queue` dan bidang `id` di tabel `User`. Metode `belongsTo` dan `hasMany` digunakan untuk mendefinisikan hubungan antara model `Queue` dan model `User`. Metode `belongsTo` menentukan bahwa sebuah `Queue` adalah milik `User`, dan metode `hasMany` menentukan bahwa sebuah `User` memiliki banyak `Queue`. Properti `as` digunakan untuk menentukan alias untuk asosiasi. Objek yang diteruskan ke metode `init` mengkonfigurasi perilaku model. Ini menetapkan opsi `timestamps` ke `true`, yang berarti bahwa Sequelize akan secara otomatis mengelola field `createdAt` dan `updatedAt`. Opsi `sequelize` diatur ke instance Sequelize berdasarkan file db.sql.js yang ada pada folder configs, yang diperlukan agar model dapat bekerja dengan database. Opsi `modelName`, `tableName`, dan `schema` digunakan untuk menyesuaikan nama model, nama tabel, dan nama skema.

```
async function getNextTicketNumber(locketNumber, currentDate) {
  try {
    const startOfDay = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth(),
      currentDate.getDate(),
      0,
      0,
      0
    );
    const endOfDay = new Date(
      currentDate.getFullYear(),
      currentDate.getMonth(),
      currentDate.getDate(),
      23,
      59,
      59
    );

    const lastQueue = await Queue.findAll({
      where: {
        locket_number: locketNumber,
        datetime: {
          [Op.between]: [startOfDay.toISOString(), endOfDay.toISOString()],
        },
      },
      limit: 1,
      order: [["ticket_number", "DESC"]],
      attributes: ["ticket_number"],
    });

    const nextNumber = lastQueue[0]
      ? parseInt(lastQueue[0].ticket_number.slice(1), 10) + 1
      : 1;
    const prefixTicketNumber =
      locketNumber === "CS1" ? "A" : locketNumber === "CS2" ? "B" : "C";
    return `${prefixTicketNumber}${nextNumber.toString().padStart(3, "0")}`;
  } catch (error) {
    console.error("Error fetching last queue:", error);
    throw error; // Handle or throw the error as appropriate
  }
}
```

Gambar 3. 4 Kode *function GetNextTicketNumber*

Fungsi bernama `getNextTicketNumber` pada **Gambar 3.3** bertugas untuk menghasilkan nomor tiket antrian berikutnya berdasarkan nomor loket tertentu (`locketNumber`) dan tanggal saat ini (`currentDate`). Fungsi ini dimulai dengan membuat dua variabel, `startOfDay` dan `endOfDay`, yang digunakan untuk mendefinisikan rentang waktu dalam satu hari. `startOfDay` mewakili awal hari pada pukul 00:00:00, sementara `endOfDay` mewakili akhir hari pada pukul 23:59:59.

Tanggal ini diperoleh dengan menggunakan method bawaan JavaScript `getFullYear()`, `getMonth()`, dan `getDate()` dari objek `currentDate`.

Selanjutnya, fungsi melakukan query ke database menggunakan model `Queue` dengan memanfaatkan metode `findAll()` dari Sequelize, yang mengembalikan hasil berupa antrian terakhir yang ada di loket tersebut pada hari yang sama. Pada query ini, kondisi `where` mencakup dua kriteria: pertama, nomor loket (`locket_number`) harus sesuai dengan loket yang ditentukan (`locketNumber`), dan kedua, waktu (`datetime`) harus berada di antara `startOfDay` dan `endOfDay`. Query ini juga dibatasi hanya mengembalikan satu baris data menggunakan `limit: 1`, dan diurutkan secara menurun berdasarkan kolom `ticket_number` menggunakan `order: [["ticket_number", "DESC"]]`. Pada akhirnya, hanya kolom `ticket_number` yang diambil dengan menggunakan opsi `attributes: ["ticket_number"]`.

Setelah query selesai, fungsi memeriksa apakah ada antrian sebelumnya pada hari tersebut di `lastQueue[0]`. Jika ada, fungsi mengambil nomor tiket terakhir tersebut, memotong karakter prefix-nya, mengonversi sisanya menjadi integer, lalu menambahnya dengan 1 untuk menghasilkan nomor antrian berikutnya. Jika tidak ada antrian pada hari itu, nomor tiket berikutnya akan dimulai dari 1. Prefix tiket ditentukan berdasarkan `locketNumber`, dengan "A" untuk "CS1", "B" untuk "CS2", dan "C" untuk "CS3". Nomor tiket kemudian dikonversi menjadi string, diikuti dengan padding angka nol di depannya hingga tiga digit menggunakan method `padStart()`. Hasil akhir adalah nomor tiket yang berupa kombinasi antara prefix dan nomor tiket berikutnya.

Jika terjadi kesalahan selama proses ini, error akan ditangkap di dalam blok `catch`, dicatat ke dalam log dengan `console.error()`, dan kemudian dilempar kembali agar bisa ditangani lebih lanjut di tempat lain dalam kode.

```
export const broadcastUpdateQueueLocketA = async (io) => {
  await io.emit("update-locket-a");
};

export const broadcastUpdateQueueLocketB = async (io) => {
  await io.emit("update-locket-b");
};

export const broadcastUpdateQueueLocketC = async (io) => {
  await io.emit("update-locket-c");
};

export const broadcastUpdateLatestCalledQueue = async (io) => {
  await io.emit("update-latest-called-queue");
};
```

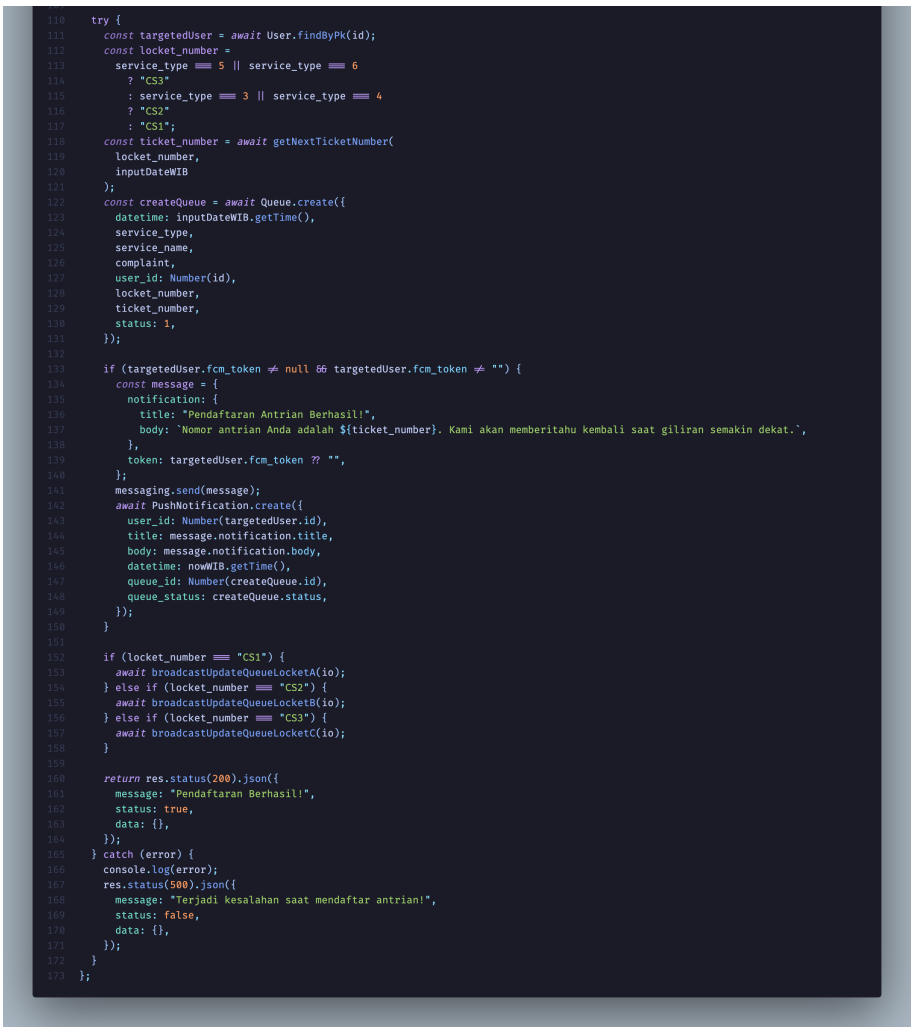
Gambar 3. 5 Kode *broadcast update event* pada *WebSocket*

Pada **Gambar 3.4** terdapat beberapa kode fungsi yang bertanggung jawab untuk mengirimkan pembaruan secara *real-time* melalui *Socket.io*. Fungsi-fungsi ini digunakan untuk menyiarkan informasi terbaru mengenai status antrian dan pemanggilan nomor di beberapa loket. Setiap fungsi didefinisikan sebagai fungsi asinkron, yang artinya eksekusi fungsi ini akan menunggu sampai semua operasi asinkron di dalamnya selesai. Fungsi pertama, `broadcastUpdateQueueLocketA`, menggunakan objek `io` untuk memancarkan event bernama "update-locket-a". Ketika event ini dikirim, semua klien yang terhubung dengan Socket.io dan mendengarkan event tersebut akan menerima pembaruan terkait dengan antrian di Loket A. Fungsi kedua dan ketiga (`broadcastUpdateQueueLocketB` dan `broadcastUpdateQueueLocketC`) bekerja dengan cara yang sama, tetapi masing-masing memancarkan event untuk loket B dan loket C, dengan event "update-locket-b" dan "update-locket-c". Ini memungkinkan aplikasi untuk menginformasikan klien secara real-time bahwa ada pembaruan di antrian loket B dan C, misalnya ketika ada perubahan nomor antrian yang baru dipanggil. Fungsi terakhir, `broadcastUpdateLatestCalledQueue`, memancarkan event "update-latest-called-queue", yang digunakan untuk memberi tahu semua klien tentang perubahan pada nomor antrian terakhir yang telah dipanggil. Event ini penting untuk memberikan pembaruan kepada pengguna yang mungkin sedang menunggu antrian mereka dipanggil. Secara keseluruhan, fungsi-fungsi ini digunakan untuk menyinkronkan data antrian antara server dan klien secara real-time, sehingga setiap perubahan atau

pembaruan pada antrian loket tertentu atau pada pemanggilan antrian terbaru dapat langsung diketahui oleh semua pengguna yang terhubung.

```
export const addQueue = async (req, res, io, messaging) => {
  const { service_type, service_name, complaint, date } = req.body;
  const { id } = req.user;

  if (!service_type || !service_name || !date) {
    return res.status(400).json({
      message: "Semua kolom data harus diisi!",
      status: false,
      data: {},
    });
  }

  const nowWIB = new TimeService().getCurrentTime();
  const inputDateWIB = new TimeService().getCurrentTime(date);

  // Validate if the provided date is within the allowed range (today to 4 days from today) and is a weekday
  const maxDate = new Date(nowWIB);
  maxDate.setDate(maxDate.getDate() + 4);

  const dayOfWeek = inputDateWIB.getDay();
  const isWeekday = dayOfWeek >= 1 && dayOfWeek <= 5;

  if (
    inputDateWIB.getDate() < nowWIB.getDate() ||
    inputDateWIB > maxDate ||
    !isWeekday
  ) {
    return res.status(400).json({
      message:
        "Tanggal yang diinput harus berada dalam rentang hari ini hingga 4 hari ke depan dan hanya di hari kerja (Senin-Jumat)!",
      status: false,
      data: {},
    });
  }

  const currentHour = nowWIB.getUTCHours();
  const currentMinutes = nowWIB.getUTCMinutes();

  if (
    inputDateWIB.getDate() === nowWIB.getDate() &&
    currentHour >= 18 &&
    currentMinutes >= 0
  ) {
    return res.status(400).json({
      message:
        "Kamu tidak bisa mendaftar antrian untuk layanan hari ini di luar jam kerja!",
      status: false,
      data: {},
    });
  }
```

```
  // Check if the time of today is past 15:30 and there are 5 or more remaining queues
  if (
    inputDateWIB.getDate() === nowWIB.getDate() &&
    currentHour > 15 &&
    currentMinutes >= 30
  ) {
    const locket_number =
      service_type === 5 || service_type === 6
        ? "CS3"
        : service_type === 3 || service_type === 4
        ? "CS2"
        : "CS1";

    const remainingQueues = await Queue.count({
      where: {
        locket_number,
        datetime: {
          [Op.between]: [
            new Date(
              nowWIB.getFullYear(),
              nowWIB.getMonth(),
              nowWIB.getDate(),
              0,
              0,
              0
            ),
            new Date(
              nowWIB.getFullYear(),
              nowWIB.getMonth(),
              nowWIB.getDate(),
              23,
              59,
              59
            ),
          ],
        },
        status: { [Op.in]: [1, 2, 3] },
      },
    });

    if (remainingQueues >= 5) {
      return res.status(400).json({
        message:
          "Tidak bisa mendaftar antrian untuk layanan hari ini setelah jam 15:30 jika masih ada 5 antrian tersisa!",
        status: false,
        data: {},
      });
    }
  }

  if (!service_type || !service_name) {
    return res.status(400).json({
      message: "Lengkapi terlebih dahulu data yang diperlukan!",
      status: false,
      data: {},
    });
  }
```



```
  try {
    const targetedUser = await User.findByPk(id);
    const locket_number =
      service_type == 5 || service_type == 6
        ? "CS3"
        : service_type == 3 || service_type == 4
        ? "CS2"
        : "CS1";
    const ticket_number = await getNextTicketNumber(
      locket_number,
      inputDateWIB
    );
    const createQueue = await Queue.create({
      datetime: inputDateWIB.getTime(),
      service_type,
      service_name,
      complaint,
      user_id: Number(id),
      locket_number,
      ticket_number,
      status: 1,
    });

    if (targetedUser.fcm_token != null && targetedUser.fcm_token != "") {
      const message = {
        notification: {
          title: "Pendaftaran Antrian Berhasil!",
          body: `Nomor antrian Anda adalah ${ticket_number}. Kami akan memberitahu kembali saat giliran semakin dekat.`,
        },
        token: targetedUser.fcm_token ?? "",
      };
      messaging.send(message);
      await PushNotification.create({
        user_id: Number(targetedUser.id),
        title: message.notification.title,
        body: message.notification.body,
        datetime: nowWIB.getTime(),
        queue_id: Number(createQueue.id),
        queue_status: createQueue.status,
      });
    }

    if (locket_number == "CS1") {
      await broadcastUpdateQueueLocketA(io);
    } else if (locket_number == "CS2") {
      await broadcastUpdateQueueLocketB(io);
    } else if (locket_number == "CS3") {
      await broadcastUpdateQueueLocketC(io);
    }

    return res.status(200).json({
      message: "Pendaftaran Berhasil!",
      status: true,
      data: {},
    });
  } catch (error) {
    console.log(error);
    res.status(500).json({
      message: "Terjadi kesalahan saat mendaftar antrian!",
      status: false,
      data: {},
    });
  }
};
```

Gambar 3. 6 Kode *Function addQueue*

Gambar 3.5 adalah *function* addQueue untuk *endpoint API "/create-queue"* yang menangani penambahan entri tiket antrian baru. *function* ini memvalidasi data permintaan, memeriksa berbagai kondisi seperti batasan tanggal dan waktu, lalu membuat entri antrean baru jika semua pemeriksaan lolos. Jika pembuatan berhasil, maka akan mengirimkan push notifikasi ke pengguna dan menyiarkan *event websocket* agar dilakukan pembaruan data pada *display list* loket antrian yang dituju pada sisi *frontend mobile*. Berikut proses yang dilakukan pada function addQueue:

1. Memvalidasi data permintaan: `service_type`, `service_name`, `complaint`, dan `date`
2. Melakukan pemeriksaan batasan tanggal dan waktu terhadap tanggal yang dipilih:
    a. Tanggal yang dipilih harus dalam 4 hari ke depan dan dalam hari kerja
    b. Tidak dapat mendaftar untuk antrian hari ini setelah pukul 16:00 WIB

c. Tidak dapat mendaftar untuk antrian hari ini setelah pukul 15:30 WIB jika ada 5 atau lebih antrian yang tersisa

3. Membuat entri tiket antrian baru dengan menghasilkan output nomor tiket dan nomor loket dengan data yang telah divalidasi
4. Apabila entri tiket antrian baru sukses dibuat, mengirimkan push notifikasi kepada pengguna menggunakan *function* dari *library firebase-admin* jika token *FCM* mereka tersedia
5. Apabila entri antrian baru sukses dibuat, menyiarkan pembaruan event *websocket* ke loket antrian berdasarkan nomor loket dari tiket antrian yang dibuat

```
import express from "express";
const router = express.Router();

import {
  addQueue,
  getDisplayListLocket,
  getUserQueueHistoryList,
  getLatestCalledQueue,
  reviewUserQueue,
} from "../controllers/user_queue.controller.js";

import { authenticate } from "../middleware/authenticate.js";

const userQueueRoutes = (io, messaging) => {
  router.post("/create-queue", authenticate, (req, res) =>
    addQueue(req, res, io, messaging)
  );
  router.post("/review", authenticate, (req, res) => reviewUserQueue(req, res));

  router.get("/locket-a", authenticate, (req, res) =>
    getDisplayListLocket(req, res, "A")
  );
  router.get("/locket-b", authenticate, (req, res) =>
    getDisplayListLocket(req, res, "B")
  );
  router.get("/locket-c", authenticate, (req, res) =>
    getDisplayListLocket(req, res, "C")
  );

  router.get("/latest-called", authenticate, getLatestCalledQueue);

  router.get("/user-history", authenticate, getUserQueueHistoryList);

  return router;
};

export default userQueueRoutes;
```

Gambar 3. 7 Kode *Router userQueueRoutes*

Gambar 3.6 adalah bagian dari implementasi *routes* di aplikasi *backend* menggunakan *Express.js*, yang mendefinisikan rute *API* untuk manajemen antrian pengguna. Di dalamnya, beberapa fungsi dari user_queue.controller.js diimpor, seperti `addQueue`, `getDisplayListLocket`, `getUserQueueHistoryList`, `getLatestCalledQueue`, dan `reviewUserQueue`. Fungsi-fungsi ini mengelola logika bisnis terkait fitur antrian. Selain itu, *middleware* `authenticate` juga digunakan untuk memastikan setiap permintaan telah melalui proses autentikasi sebelum mencapai fungsi yang diinginkan.

Instance router dibuat menggunakan `express.Router()` untuk mendefinisikan rute-rute terkait antrian pengguna. Beberapa route seperti `POST /create-queue` bertanggung jawab untuk menambahkan pengguna ke antrian dengan memanfaatkan fungsi `addQueue`. Pada route ini, middleware `authenticate` memastikan bahwa pengguna telah terautentikasi sebelum antrian dibuat, dan fungsi `addQueue` dipanggil dengan tambahan parameter `io` untuk *Websocket* dengan *library Socket.io* dan `messaging` untuk *Firebase Cloud Messaging* dengan *library firebase-admin*. Rute lain seperti `POST /review` memungkinkan pengguna memberikan ulasan terhadap layanan antrian, sementara `GET /locket-a`, `GET /locket-b`, dan `GET /locket-c` masing-masing digunakan untuk mendapatkan daftar nomor antrian paling pertama saat ini dan nomor antrian selanjutnya berdasarkan loket tertentu, yakni A, B, atau C. Selain itu, route `GET /latest-called` menyediakan informasi tentang antrian terakhir yang dipanggil, dan `GET /user-history` memungkinkan pengguna untuk melihat riwayat antrian mereka. Semua route ini dilindungi oleh middleware `authenticate`, sehingga hanya pengguna yang sudah terautentikasi yang dapat mengaksesnya.

Fungsi `userQueueRoutes` menerima parameter `io` dan `messaging`, yang masing-masing merupakan instance dari Socket.io untuk komunikasi real-time dan Firebase Cloud Messaging dari firebase-admin untuk mengirim push notifikasi. Pada route `/create-queue`, kedua teknologi ini digunakan untuk memastikan bahwa pembaruan real-time dan notifikasi push diterima oleh pengguna dengan cepat dan efisien. Terakhir, router dikembalikan setelah semua route didefinisikan agar bisa digunakan di aplikasi utama, dan akan di-mount pada URL path tertentu dengan prefix `/api/queue` yang nantinya akan digunakan seperti `/api/queue/create-queue`.

```
import express from "express";
import morgan from "morgan";
import bodyParser from "body-parser";
import cors from "cors";
import dotenv from "dotenv";
import UserRoute from "./routes/user.routes.js";
import UserQueueRoutes from "./routes/user_queue.routes.js";
import PushNotificationRoute from "./routes/push_notification.routes.js";
import http from "http";
import { Server } from "socket.io";
import admin from "firebase-admin";
import AdminQueueRoutes from "./routes/admin_queue.routes.js";
import queueScheduler from "./scheduler/queue_scheduler.js";

dotenv.config();

const app = express();
const server = http.createServer(app);

app.use(
  cors({
    origin: "*", // Your React app's URL
    methods: ["GET", "POST"], // Allowed HTTP methods
    allowedHeaders: [
      "Content-Type",
      "Accept",
      "Authorization",
      "Access-Control-Allow-Origin",
      "Access-Control-Allow-Headers",
      "User-Agent",
      "Origin",
    ],
    credentials: true, // Allow credentials (cookies, authorization headers, etc.)
  })
);
app.use(morgan("dev"));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
const io = new Server(server, {
  connectionStateRecovery: {},
  cors: {
    origin: "*", // Your React app's URL
    allowedHeaders: [
      "Content-Type",
      "Accept",
      "Authorization",
      "Access-Control-Allow-Origin",
      "Access-Control-Allow-Headers",
      "User-Agent",
      "Origin",
    ],
    methods: ["GET", "POST"], // Allowed HTTP methods
    credentials: true, // Allow credentials (cookies, authorization headers, etc.)
  },
});

const serviceAccount = JSON.parse(process.env.GOOGLE_APPLICATION_CREDENTIALS);

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
});

const messaging = admin.messaging();

const PORT = process.env.PORT || 3000;

// Handle Socket.IO connections
io.on("connection", (socket) => {
  // Handle Socket.IO disconnections
  socket.on("disconnect", () => {});
});

server.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
  queueScheduler();
});

app.use("/api/user", UserRoute);
app.use("/api/queue", UserQueueRoutes(io, messaging));
app.use("/api/push-notif", PushNotificationRoute);
app.use("/api/admin", AdminQueueRoutes(io, messaging));
```

Gambar 3. 8 Kode Inisiasi *Server Backend*

Gambar 3.7 adalah bagian dari inisiasi *server backend* yang dikembangkan menggunakan *Express.js*. Kode dimulai dengan mengimpor berbagai modul yang dibutuhkan, seperti *express*, morgan untuk *logging*, *body-parser* untuk menangani

body request, *cors* untuk mengizinkan *cross-origin requests*, serta *dotenv* untuk memuat variabel lingkungan dari file `.env`. Selain itu, ada beberapa *route* yang diimpor dari file terpisah untuk menangani berbagai fitur, seperti UserRoute, UserQueueRoutes, PushNotificationRoute, dan AdminQueueRoutes. Kode ini juga mengimpor http untuk membuat *server HTTP* dan *Socket.io* untuk menangani koneksi *real-time websocket*. *Firebase Admin SDK* juga diimpor menggunakan modul admin untuk mengelola *push notifications*.

Setelah memuat variabel lingkungan melalui `dotenv.config()`, *instance Express* dibuat dan *server HTTP* didefinisikan menggunakan `http.createServer()`. *Middleware CORS* kemudian diterapkan untuk memungkinkan akses dari semua asal (`*`), mengizinkan metode *HTTP* seperti `GET` dan `POST`, dan memperbolehkan penggunaan kredensial seperti *cookies* atau *header* otorisasi. *Morgan* digunakan untuk mencatat setiap *request* yang masuk, dan *body-parser* memungkinkan *server* untuk mengurai *body request* dalam format *URL-encoded* dan *JSON*.

Socket.io diinisialisasi dengan dukungan untuk pemulihan status koneksi dan pengaturan *CORS* serupa dengan konfigurasi pada *Express* yang mengizinkan semua asal untuk melakukan koneksi *WebSocket*. *Firebase Admin SDK* kemudian diinisialisasi menggunakan kredensial yang diambil dari variabel lingkungan yang disimpan dalam file `.env`, dan instance *messaging* dari *Firebase Messaging* dibuat untuk memungkinkan pengiriman push notifikasi.

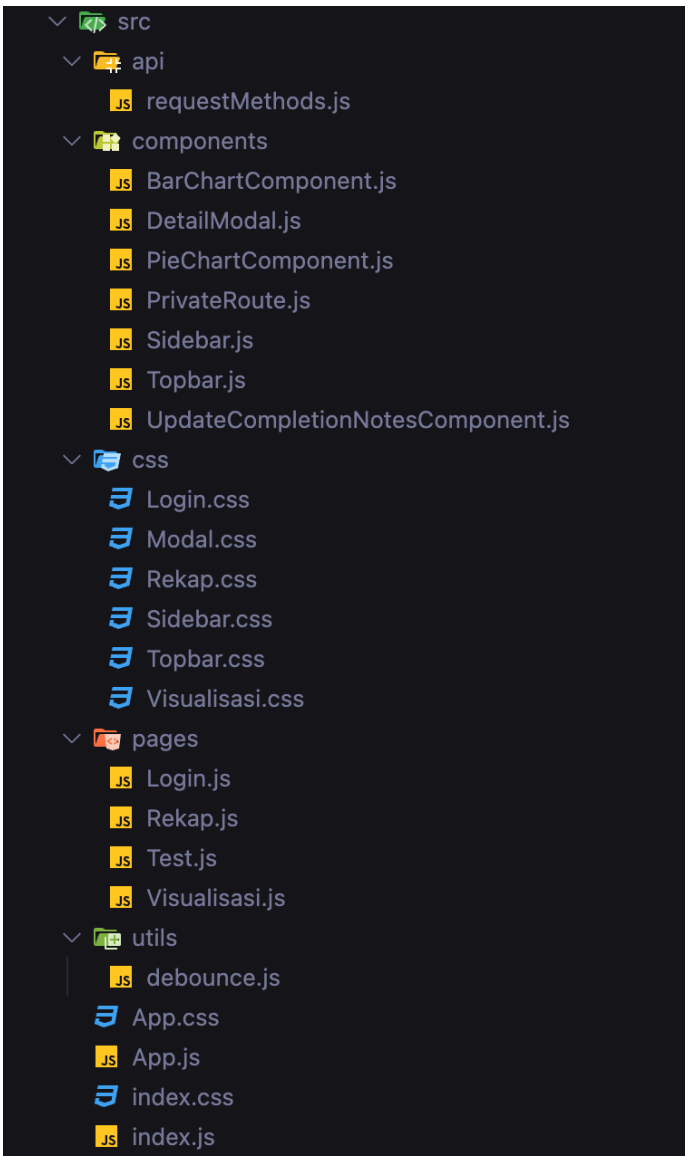
Server kemudian diatur untuk mendengarkan pada *port* yang ditentukan oleh variabel lingkungan `PORT` atau *default* ke *port* 3000. Ketika *server* berhasil dijalankan, pesan akan ditampilkan pada console yang menunjukkan *port* tempat *server* berjalan, dan fungsi `queueScheduler` dipanggil untuk memulai *function* yang ada pada *scheduler* tersebut.

Berbagai route diatur untuk menangani request API, seperti route untuk UserRoute pada path `/api/user` yang berisikan routes yang mengatur seluruh operasi terkait pengguna, seperti login, pendaftaran, dan pengelolaan akun pengguna, UserQueueRoutes pada path `/api/queue` yang menerima instance Socket.io dan messaging untuk operasi terkait pengelolaan antrian pengguna, seperti menambahkan antrian, memeriksa antrian terkini dan memeriksa riwayat antrian pengguna, route PushNotificationRoute pada `/api/push-notif untuk operasi terkait pemeriksaan daftar push notifikasi pengguna, serta AdminQueueRoutes yang juga

menerima Socket.io dan messaging pada `/api/admin` untuk operasi terkait manajemen antrian yang dikelola oleh admin, seperti mengubah status antrian atau melihat daftar antrian secara keseluruhan.

3.3.2. Implementasi Pengkodean *Frontend Web*

Pada tahap ini dilakukannya proses pembuatan *frontend* aplikasi *web* dengan menggunakan teknologi yang diperlukan diantaranya yaitu *framework React.Js* dengan menggunakan *library Socket.io* dari bahasa pemrograman *Javascript*. Sistem real-time yang dikembangkan dengan Socket.io memastikan bahwa perubahan yang terjadi pada data antrian dapat langsung direspon dan diperbarui secara langsung pada *frontend* aplikasi tanpa diperlukannya interaksi manual oleh pengguna terlebih dahulu. Untuk mengimplementasikan integrasi *API* antara aplikasi *React.js* dan *backend Express*, digunakan *library* Axios dan *API* yang didokumentasikan di *Postman*. Berikut adalah tahap-tahap pengkodean secara umum.



Gambar 3. 9 Struktur *Folder* Aplikasi *Frontend Web*

Pada struktur *folder* aplikasi *frontend web* yang ditampilkan pada **Gambar 3.8**, proyek ini dibagi menjadi beberapa direktori utama seperti `api`, `components`, `css`, `pages`, dan `utils`. Setiap direktori ini memiliki peran dan fungsi spesifik yang membantu dalam pengembangan aplikasi secara modular dan terorganisir.

Direktori `api` berisi *file* `requestMethods.js`, yang kemungkinan besar bertanggung jawab untuk mengelola permintaan *API*, baik untuk *GET*, *POST*, atau metode *HTTP* lainnya, yang digunakan untuk berinteraksi dengan *backend* atau *server* eksternal. File ini berfungsi untuk mengabstraksi logika yang berhubungan dengan pengambilan data dari *server* dan memudahkan pengelolaan *request* di seluruh aplikasi.

Direktori `components` berisi berbagai komponen React.js yang digunakan untuk membangun elemen-elemen *UI* di aplikasi ini. Di antaranya terdapat `BarChartComponent.js`, `PieChartComponent.js`, dan `DetailModal.js`, yang masing-masing mungkin bertanggung jawab untuk menampilkan grafik batang, grafik pie, dan detail data dalam bentuk modal. Komponen-komponen ini memisahkan logika tampilan visual dan memungkinkan pengembang menggunakan kembali komponen yang sama di berbagai tempat. *File* `PrivateRoute.js` digunakan untuk melindungi rute yang hanya bisa diakses oleh pengguna yang sudah terautentikasi, sedangkan `Sidebar.js` dan `Topbar.js` merupakan komponen yang bertanggung jawab untuk menampilkan navigasi sisi dan bagian atas dari *UI*. `UpdateCompletionNotesComponent.js`digunakan untuk menampilkan atau memperbarui catatan penyelesaian layanan antrian dalam aplikasi.

Direktori `css` berisi berbagai *file CSS* yang bertugas mengatur gaya visual dari halaman dan komponen dalam aplikasi. Misalnya, file `Login.css` dan `Rekap.css` digunakan untuk mengatur gaya halaman *login* dan halaman rekapitulasi data. File *CSS* lainnya seperti `Sidebar.css`, `Topbar.css`, dan `Modal.css` bertugas untuk menentukan tampilan dari komponen terkait seperti *sidebar*, *topbar*, dan *modal*.

Direktori `pages` berisi halaman-halaman utama dari aplikasi, di mana *file* `Login.js`, `Rekap.js`, `Test.js`, dan `Visualisasi.js` mewakili halaman-halaman berbeda yang akan diakses oleh pengguna. Setiap file ini mewakili *route* atau halaman yang dapat dirender berdasarkan navigasi pengguna. *File* `Login.js` berfungsi untuk menangani logika login pengguna, sedangkan `Rekap.js` dan `Visualisasi.js` bertanggung jawab untuk menampilkan data dalam bentuk rekapitulasi dan visualisasi grafik.

Terakhir, direktori `utils` berisi file `debounce.js`, yang biasanya digunakan untuk mengoptimalkan performa fungsi dengan menunda eksekusinya sampai tindakan tertentu berhenti dilakukan oleh pengguna. Selain itu, ada juga file `App.js`, `App.css`, `index.js`, dan `index.css`, yang merupakan bagian inti dari aplikasi *React.js*. `App.js` adalah komponen utama yang me-*render* seluruh aplikasi, sementara `index.js` adalah *file entry point* yang memuat aplikasi ke dalam elemen *DOM* di *browser*. *File-file CSS* yang terkait (`App.css`, `index.css`) mengatur gaya keseluruhan aplikasi.

Dengan struktur ini, aplikasi *frontend web* dibangun secara modular, di mana setiap elemen *UI* dan logika dipisahkan ke dalam komponen dan halaman tertentu, yang memudahkan pemeliharaan, pembaruan, dan pengujian aplikasi di masa mendatang.

```
import axios from "axios";

const BASE_URL = process.env.REACT_APP_BASE_API_URL;

// Create the axios instance without the token in headers (we'll add it dynamically)
export const requestWithHeaders = axios.create({
  baseURL: BASE_URL,
  headers: {
    "Content-Type": "application/json",
    Accept: "*/*",
  },
});

// Add an interceptor to dynamically set the Authorization header before each request
requestWithHeaders.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token") || "";
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

```

Gambar 3. 10 Kode Konfigurasi *Request API* Menggunakan *Axios*

Gambar 3.9 merupakan implementasi metode *request API* menggunakan *axios*, sebuah pustaka *HTTP client* berbasis *Promise* di *JavaScript*, yang digunakan untuk membuat permintaan *HTTP* ke *API*. Pada awalnya, kode ini mengimpor axios dan menginisialisasi konstanta BASE_URL yang diambil dari variabel lingkungan bernama REACT_APP_BASE_API_URL. Variabel ini digunakan sebagai dasar URL untuk semua permintaan API yang akan dibuat dalam aplikasi React tersebut, memastikan bahwa URL dasar dapat dikonfigurasi secara dinamis tergantung pada lingkungan (misalnya pengembangan atau produksi).

Selanjutnya, objek *axios* baru diinisialisasi melalui requestWithHeaders, di mana axios.create() digunakan untuk membuat instans baru dengan beberapa pengaturan khusus. Pada objek yang dihasilkan ini, properti baseURL diset ke nilai BASE_URL yang telah ditentukan sebelumnya. Selain itu, ada beberapa headers

yang disetel secara global, termasuk `Content-Type` yang diatur ke `application/json` dan `Accept` yang diatur untuk menerima semua tipe konten. Ini bertujuan agar permintaan yang dilakukan dari aplikasi ini selalu menyertakan tipe konten *JSON*, yang biasanya digunakan dalam komunikasi *API* modern.

Bagian penting dari kode ini adalah penggunaan interceptor pada axios. Interceptors adalah mekanisme di axios yang memungkinkan penanganan tambahan dilakukan sebelum atau sesudah setiap permintaan dan respons. Pada kasus ini, interceptor dipasang pada permintaan (request) untuk secara dinamis menambahkan token otorisasi di header setiap kali permintaan *API* dilakukan. Token ini diambil dari localStorage, dan jika token ditemukan, ia akan ditambahkan ke dalam header Authorization dalam format Bearer. Ini berguna untuk melakukan autentikasi berbasis token pada server, yang sering digunakan dalam aplikasi berbasis *JWT* (*JSON Web Token*). Jika tidak ada token yang ditemukan, permintaan tetap akan diteruskan tetapi tanpa otorisasi.

Terakhir, interceptor juga menangani kesalahan jika terjadi masalah saat menambahkan token atau konfigurasi lainnya. Jika terdapat kesalahan, Promise.reject(error) dipanggil, yang akan menolak permintaan dan mengirimkan error ke tempat di mana permintaan dipanggil, sehingga aplikasi dapat menangani kesalahan tersebut dengan tepat di lapisan logika bisnisnya.

```jsx
import "./App.css";
import Sidebar from "./components/Sidebar";
import Topbar from "./components/Topbar";
import { useEffect, useState } from "react";
import { requestWithHeaders } from "./api/requestMethods";
import { io } from "socket.io-client";
import debounce from "./utils/debounce";
import UpdateCompletionNotesComponent from "./components/UpdateCompletionNotesComponent";

function App() {
  const socket = io(process.env.REACT_APP_WEBSOCKET_URL, {
    withCredentials: true, // Allows sending credentials such as cookies
    transports: ["websocket"],
    autoConnect: true,
  });
  const [modal, setModal] = useState(false);
  const [paginationData, setPaginationData] = useState(null);
  const [modalData, setModalData] = useState({});
  const [page, setPage] = useState(1);
  const [loading, setLoading] = useState(false);
  const limit = 10;
  const [managedLocket, setManagedLocket] = useState("");
  const [utteranceLoading, setUtteranceLoading] = useState(false);
  const [lockets, setLockets] = useState([]);

  const service_types = [
    { service_type: 1, name: "Mahasiswa" },
    { service_type: 2, name: "Calon Mahasiswa" },
    { service_type: 3, name: "Dosen" },
    { service_type: 4, name: "Tenaga Pendidik" },
    { service_type: 5, name: "Masyarakat" },
    { service_type: 6, name: "Alumni" },
  ];

  const fetchData = async ({ resetPage = true }) => {
    if (managedLocket === "") return;
    if (resetPage && page !== 1) {
      setPage(1);
    }
    try {
      const res = await requestWithHeaders.get(
        `/api/admin/locket-${managedLocket}?status=active&page=${page}&limit=${limit}`
      );
      if (res?.data?.status) {
        setLockets(() => res?.data?.data);
        setPaginationData(() => res?.data?.pagination);
      }
    } catch (err) {
      console.log(err);
    }
  };

  const debouncedFetchData = debounce(fetchData({ resetPage: true }), 300);

  const toggleModal = (locket) => {
    setModal(!modal);
    setModalData(locket);
  };

  const changeStatus = async (id, status) => {
    setLoading(true);
    try {
      await requestWithHeaders.post(`/api/admin/change-status`, {
        id,
        status,
      });
      setLoading(false);
    } catch (err) {
      setLoading(false);
      if (page === 1) {
        debouncedFetchData();
      }
    }
  };

```

```
const speak = (ticketNumber = "") => {
  // Mengecek apakah browser mendukung SpeechSynthesis

  if ("speechSynthesis" in window) {
    const utterance = new SpeechSynthesisUtterance(
      `nomor antrian ${ticketNumber} silahkan menuju loket.`
    );
    utterance.onstart = () => {
      setUtteranceLoading(true); // Start loading when speech begins
    };
    utterance.onend = () => {
      setUtteranceLoading(false); // Start loading when speech begins
    };
    utterance.lang = "id-ID"; // Set bahasa ke Indonesia
    utterance.rate = 0.6; // Set tingkat kecerdasan suara
    window.speechSynthesis.speak(utterance);
    setTimeout(() => {
      window.speechSynthesis.speak(utterance);
    }, 1000);
  } else {
    console.log("Browser Anda tidak mendukung Text-to-Speech.");
    alert("Browser Anda tidak mendukung Text-to-Speech.");
  }
};

const reconnectSocket = () => {
  setTimeout(socket.connect(), 200);
};

const initManagedLocket = async () => {
  const userLocket = await requestWithHeaders.get("api/user/profile");
  setManagedLocket(() => userLocket?.data?.data?.managed_locket);
};

const initSocket = async () => {
  socket.on("connect", () => {
    console.log("Connected to websocket");
    debouncedFetchData();
  });

  if (managedLocket === "a") {
    socket.on("update-locket-a", () => {
      console.log("update-locket-a");
      debouncedFetchData();
    });
  }

  if (managedLocket === "b") {
    socket.on("update-locket-b", () => {
      console.log("update-locket-b");
      debouncedFetchData();
    });
  }

  if (managedLocket === "c") {
    socket.on("update-locket-c", () => {
      console.log("update-locket-c");
      debouncedFetchData();
    });
  }

  socket.on("disconnect", (event) => {
    socket.close();
    if (event !== "io client disconnect") {
      reconnectSocket();
    }
  });
  socket.on("error", (error) => {
    socket.disconnect();
    reconnectSocket();
  });
  socket.connect();
};

useEffect(() => {
  initManagedLocket();
}, []);

useEffect(() => {
  if (managedLocket === "") return;
  initSocket();
}, [managedLocket]);

useEffect(() => {
  fetchData({ resetPage: false });
}, [page]);
```

Gambar 3. 11 Cuplikan Kode Logika Bisnis pada App.js

Gambar 3.10 merupakan bagian dari aplikasi pada main dashboard page di file App.js yang berfungsi untuk mengelola dan menampilkan data antrian di beberapa loket menggunakan koneksi *WebSocket* dan *API*. Pertama, kode ini menginisialisasi koneksi *Socket.IO* dengan menggunakan *URL WebSocket* yang diambil dari variabel lingkungan, memungkinkan pengiriman kredensial seperti cookies dan mengonfigurasi transportasi yang digunakan untuk koneksi, di mana koneksi akan otomatis terhubung saat aplikasi dijalankan. Berbagai state dikelola menggunakan *hook useState*, termasuk modal untuk menampilkan informasi lebih lanjut, data untuk *pagination*, data yang ditampilkan di modal, dan status loading untuk interaksi pengguna. Hal ini menciptakan struktur yang fleksibel untuk menangani data antrian dan status aplikasi secara keseluruhan.

Fungsi fetchData() digunakan untuk mengambil data antrian berdasarkan loket yang dikelola saat ini. Jika tidak ada loket yang dikelola, fungsi ini tidak akan melakukan permintaan. Jika pengaturan resetPage diaktifkan dan halaman saat ini bukan halaman pertama, maka halaman akan di-reset ke satu. Data yang diperoleh dari *API* disimpan dalam state yang relevan, yang memungkinkan aplikasi untuk menampilkan antrian dengan benar. Fungsi ini juga dibungkus dalam mekanisme *debounce*, sehingga pemanggilan fungsi dapat ditunda untuk menghindari permintaan yang terlalu sering ke server. Kode ini juga mencakup fungsi toggleModal() untuk membuka atau menutup modal yang menampilkan informasi tentang loket tertentu, dan fungsi changeStatus() untuk memperbarui status antrian, dengan feedback loading yang memberikan pengguna informasi tentang proses yang sedang berlangsung.

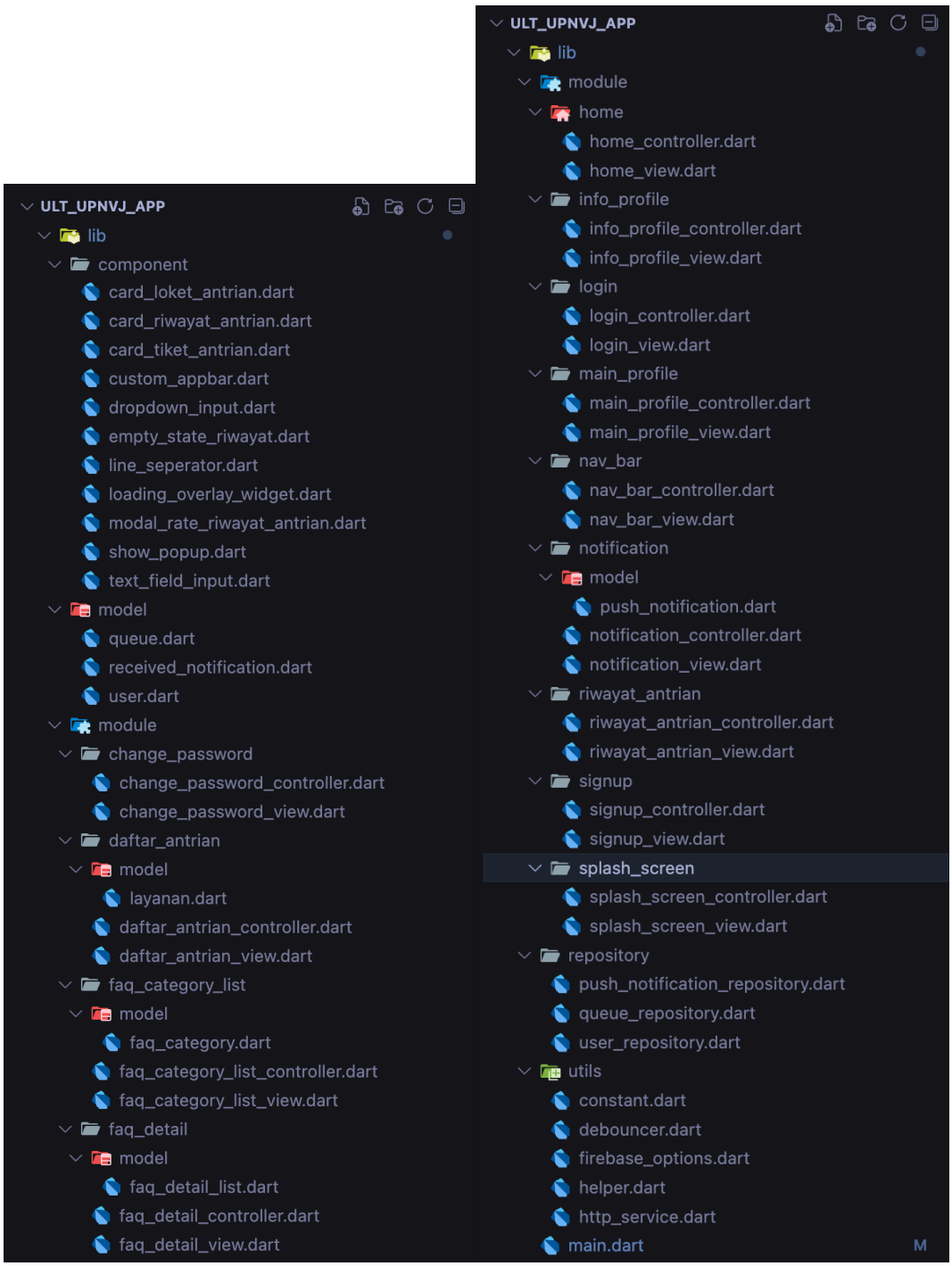
Di sisi lain, fungsi speak() memanfaatkan *Web Speech API* untuk memberikan umpan balik suara kepada pengguna. Fungsi ini memeriksa apakah *browser* mendukung *text-to-speech*, dan jika mendukung, akan membuat objek SpeechSynthesisUtterance untuk mengonversi teks menjadi suara. Hal ini ditujukan untuk memberikan instruksi kepada pengguna tentang nomor antrian mereka dengan pengaturan bahasa dan tingkat suara tertentu. Fungsi ini juga menangani state loading yang menunjukkan apakah suara sedang diputar.

Sebagai bagian dari manajemen koneksi *WebSocket*, fungsi initSocket() mengatur berbagai *listener* untuk menangani *event* dari *server*, termasuk pembaruan status loket. Fungsi ini akan terhubung ke *socket* dan mendengarkan *event* seperti

update-locket-a, update-locket-b, dan update-locket-c, yang akan memicu pembaruan data antrian ketika pembaruan terjadi. Koneksi *socket* juga menangani *event disconnect* dan *error*, yang akan mencoba untuk menyambung kembali secara otomatis jika terjadi kesalahan. Terakhir, *useEffect* digunakan untuk menginisialisasi loket yang dikelola ketika komponen pertama kali dimuat, serta untuk memulai koneksi *WebSocket* berdasarkan *state* loket yang dikelola. Dengan cara ini, aplikasi mampu mempertahankan interaksi yang dinamis dan responsif, memberikan pengalaman pengguna yang lebih baik saat berinteraksi dengan antrian.

3.3.3. Implementasi Pengkodean *Frontend Mobile*

Pada tahap ini dilakukannya proses pembuatan *frontend* aplikasi *mobile* dengan menggunakan teknologi yang diperlukan diantaranya yaitu *framework Flutter* dengan menggunakan *state management Provider* dan *library Socket.io* dari bahasa pemrograman *Dart*. Sistem real-time yang dikembangkan dengan *Socket.io* memastikan bahwa perubahan yang terjadi pada data antrian dapat langsung direspon dan diperbarui secara langsung pada *frontend* aplikasi tanpa diperlukannya interaksi manual oleh pengguna terlebih dahulu. Untuk mengimplementasikan integrasi *API* antara aplikasi *Flutter* dan *backend* Express, digunakan *library* http dari bahasa pemrograman *dart* dan *API* yang didokumentasikan di *Postman*. Berikut adalah tahap-tahap pengkodean secara umum.



Gambar 3. 12 Struktur *Folder* Aplikasi *Frontend Mobile*

Pada struktur *folder* aplikasi *frontend mobile* yang ditampilkan pada **Gambar 3.11**, proyek ini dibagi menjadi beberapa direktori utama seperti `components`, `model`, `module`, `repository`, dan `utils`. Setiap direktori ini memiliki peran dan fungsi spesifik yang membantu dalam pengembangan aplikasi secara modular dan terorganisir. Penulis juga menggunakan arsitektur *MVC (Model – View – Controller)* yaitu pemisahan *file* atau baris kode antara *model* dengan *view* dan *controller*. Berikut uraian dari setiap *folder* yang ada pada aplikasi *frontend mobile*:

A. *Folder component*

Folder ini berisi berbagai komponen *UI* yang dapat digunakan kembali dalam berbagai bagian aplikasi. *File* seperti card_loket_antrian.dart, card_riwayat_antrian.dart, dan custom_appbar.dart, yang menunjukkan bahwa aplikasi memiliki komponen UI *card* khusus untuk menampilkan informasi antrian atau riwayat antrian serta *appbar* yang dikustomisasi. Selain itu, komponen-komponen lain seperti dropdown_input.dart dan text_field_input.dart membantu dalam membangun antarmuka input data, sedangkan loading_overlay_widget.dart dan empty_state_riwayat.dart digunakan untuk menampilkan status *loading* dan kondisi data kosong. Dengan adanya komponen-komponen ini, penulis bisa merancang antarmuka pengguna yang konsisten di seluruh halaman aplikasi.

B. *Folder model*

Folder ini menyimpan definisi *model* data yang digunakan di aplikasi secara global. File seperti queue.dart, received_notification.dart, dan user.dart mendefinisikan struktur data untuk antrian, notifikasi yang diterima, serta informasi pengguna. *Model-model* ini berfungsi sebagai representasi objek data yang diperoleh dari *API* atau disimpan di dalam aplikasi, sehingga bisa digunakan dengan mudah di komponen *controller* aplikasi.

C. *Folder module*

Folder ini menyimpan *sub-folder* berdasarkan fitur atau modul yang ada dalam aplikasi. Setiap modul memiliki *file controller* dan *view* untuk mengelola logika bisnis serta tampilan antarmuka pengguna pada aplikasi. Apabila terdapat *model* data yang digunakan di modul tersebut saja, maka file *model*-nya akan disertakan di *sub-folder* untuk modul tersebut.

D. *Folder repository*

Folder repository digunakan untuk menangani akses data dan abstraksi terhadap *API*. Di dalamnya terdapat *file* seperti push_notification_repository.dart, user_repository.dart dan queue_repository.dart yang bertanggung jawab untuk pengambilan dan pengelolaan data notifikasi, data user dan data antrian. *Repository* ini memisahkan logika pemanggilan data *API* dari *controller*, menjadikan kode lebih terstruktur dan mudah di-*maintain*.

E. *Folder utils*

Folder ini berisi utilitas yang digunakan di seluruh aplikasi. Beberapa *file* seperti constant.dart, debouncer.dart, dan firebase_options.dart membantu dalam pengaturan umum, seperti konfigurasi *Firebase*, penggunaan *debounce* dalam *event handler*, dan mendefinisikan nilai-nilai konstan. *File* http_service.dart digunakan sebagai *wrapper* untuk melakukan pemanggilan *HTTP* dengan *API*.

F. Main.dart

File ini adalah *entry point* dari aplikasi *Flutter*. Di dalamnya, biasanya penulis menginisialisasi berbagai bagian aplikasi, seperti mengatur *routing*, *state management*, dan konfigurasi *firebase*, konfigurasi *push notification* dan berbagai konfigurasi aplikasi lainnya saat aplikasi pertama kali dijalankan.

```dart
class Queue {
  final int id;
  final int serviceType;
  final String serviceName;
  final int userId;
  final String complaint;
  final int status;
  final DateTime datetime;
  final String locketNumber;
  final String ticketNumber;
  final bool isReviewed;
  final int userReviewScore;

  Queue({
    required this.id,
    required this.serviceType,
    required this.serviceName,
    required this.userId,
    required this.complaint,
    required this.status,
    required this.datetime,
    required this.locketNumber,
    required this.ticketNumber,
    required this.isReviewed,
    required this.userReviewScore,
  });

  factory Queue.fromJson(Map<String, dynamic> json) {
    return Queue(
      id: int.tryParse(json['id'] ?? '-1') ?? -1,
      serviceType: json['service_type'],
      serviceName: json['service_name'],
      userId: int.tryParse(json['user_id'] ?? '-1') ?? -1,
      complaint: json['complaint'],
      status: json['status'],
      datetime: DateTime.parse(json['datetime']),
      locketNumber: json['locket_number'],
      ticketNumber: json['ticket_number'],
      isReviewed: json['is_reviewed'],
      userReviewScore: json['user_review_score'],
    );
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = <String, dynamic>{};
    data['id'] = id;
    data['service_type'] = serviceType;
    data['service_name'] = serviceName;
    data['user_id'] = userId;
    data['complaint'] = complaint;
    data['status'] = status;
    data['date_time'] = datetime.toIso8601String();
    data['locket_number'] = locketNumber;
    data['ticket_number'] = ticketNumber;
    data['is_reviewed'] = isReviewed;
    data['user_review_score'] = userReviewScore;
    return data;
  }

  Queue copyWith({
    int? id,
    int? serviceType,
    String? serviceName,
    int? userId,
    String? complaint,
    int? status,
    DateTime? datetime,
    String? locketNumber,
    String? ticketNumber,
    bool? isReviewed,
    int? userReviewScore,
  }) {
    return Queue(
      id: id ?? this.id,
      serviceType: serviceType ?? this.serviceType,
      serviceName: serviceName ?? this.serviceName,
      userId: userId ?? this.userId,
      complaint: complaint ?? this.complaint,
      status: status ?? this.status,
      datetime: datetime ?? this.datetime,
      locketNumber: locketNumber ?? this.locketNumber,
      ticketNumber: ticketNumber ?? this.ticketNumber,
      isReviewed: isReviewed ?? this.isReviewed,
      userReviewScore: userReviewScore ?? this.userReviewScore,
    );
  }
}

```

Gambar 3. 13 Kode *Model Queue*

Gambar 3.12 mendefinisikan *model* kelas `Queue` yang merepresentasikan objek antrian dengan beberapa atribut seperti `id`, `serviceType`, `serviceName`, `userId`, `complaint`, `status`, `datetime`, `locketNumber`, `ticketNumber`, `isReviewed`, dan `userReviewScore`. Kelas ini dilengkapi dengan sebuah konstruktor yang memerlukan semua atribut sebagai parameter dan memastikan bahwa semua atribut bersifat wajib (`*required*`).

Selain itu, terdapat juga `*factory constructor*` bernama `fromJson`, yang bertugas untuk membuat *instance* `Queue` dari data berbentuk `Map<String, dynamic>` atau *JSON*. Metode ini berusaha mengonversi nilai-nilai dari *JSON* menjadi tipe data yang sesuai, seperti `int` atau `DateTime`. Jika data *JSON* tidak sesuai, maka akan disediakan nilai default untuk menghindari kesalahan *parsing*.

Metode `toJson` dalam kelas ini mengonversi *instance* `Queue` kembali ke bentuk `Map<String, dynamic>`, yang biasanya digunakan saat ingin mengirim data ini ke dalam format *JSON*. Atribut-atribut seperti `datetime` juga diubah ke format yang kompatibel dengan *JSON*, yaitu *string* dalam bentuk ISO8601.

Selain itu, ada metode `copyWith` yang memungkinkan pembuatan salinan objek `Queue` yang ada dengan beberapa perubahan pada nilai atribut tertentu. Metode ini berguna ketika hanya beberapa atribut yang ingin diubah sementara sisanya tetap sama. Jika suatu atribut tidak diberikan dalam metode ini, maka nilai aslinya dari *instance* `Queue` tersebut akan dipertahankan.

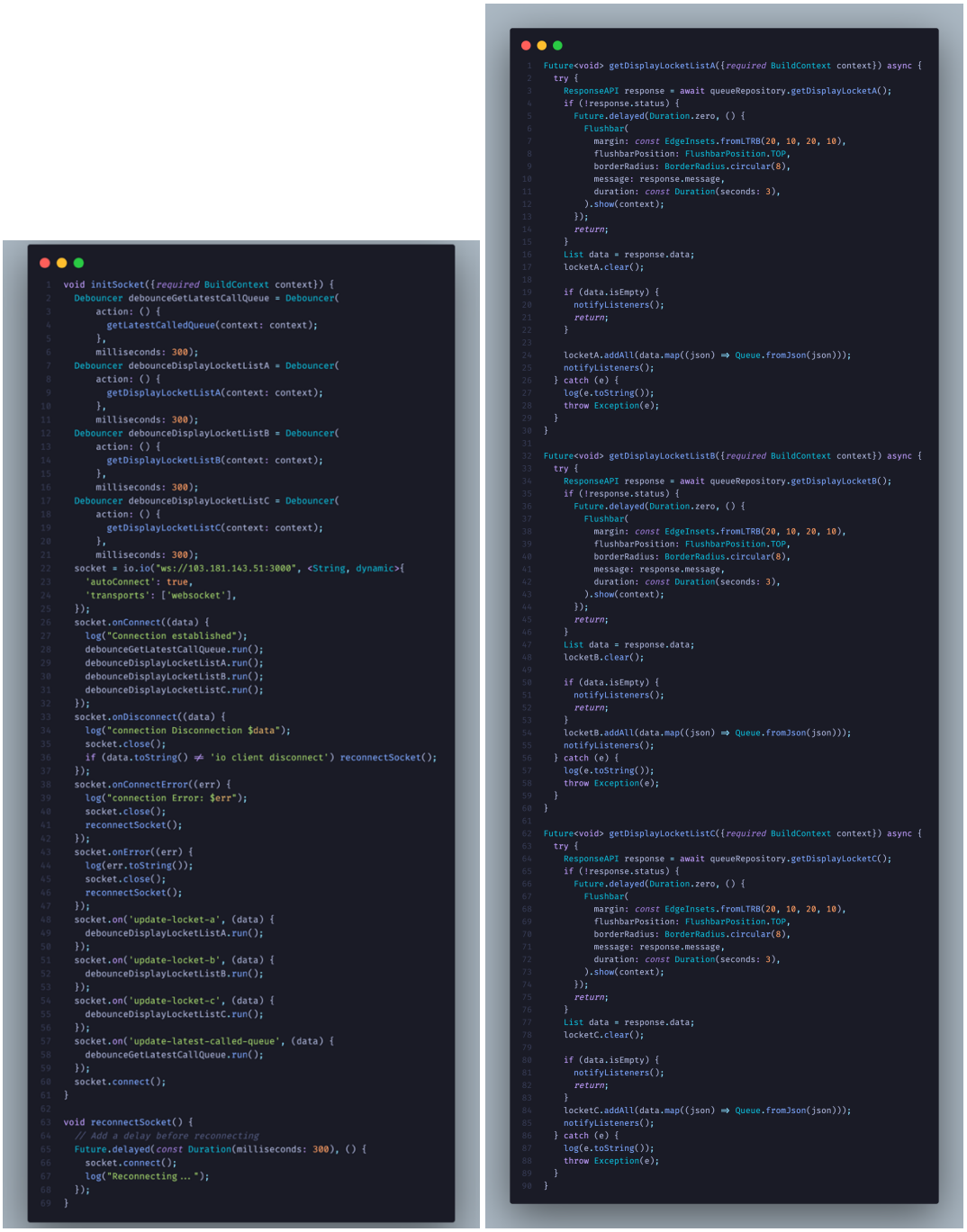
```dart
import 'dart:developer';

import 'package:ult_upnvj_app/utils/http_service.dart';

class QueueRepository {
  final HttpService httpService = HttpService();
  String baseModulePrefix = "queue/";

  Future<ResponseAPI> addQueue({
    required int serviceType,
    required String serviceName,
    required String complaint,
    required String date,
  }) async {
    Map<String, dynamic> body = {
      "service_type": serviceType,
      "service_name": serviceName,
      "complaint": complaint,
      "date": date,
    };
    log(body.toString());
    try {
      ResponseAPI response =
          await httpService.post('${baseModulePrefix}create-queue', body: body);
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> getDisplayLocketA() async {
    try {
      ResponseAPI response =
          await httpService.get('${baseModulePrefix}locket-a');
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> getDisplayLocketB() async {
    try {
      ResponseAPI response =
          await httpService.get('${baseModulePrefix}locket-b');
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> getDisplayLocketC() async {
    try {
      ResponseAPI response =
          await httpService.get('${baseModulePrefix}locket-c');
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> getHistoryQueueList({
    required String status,
    required int offset,
    required int limit,
  }) async {
    try {
      ResponseAPI response = await httpService.get(
          '${baseModulePrefix}user-history?status=$status&offset=$offset&limit=$limit');
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> getLatestCalledQueue() async {
    try {
      ResponseAPI response =
          await httpService.get('${baseModulePrefix}latest-called');
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  Future<ResponseAPI> reviewUserQueue({
    required int id,
    required int userReviewScore,
  }) async {
    Map<String, dynamic> body = {
      "id": id,
      "user_review_score": userReviewScore,
    };
    try {
      ResponseAPI response =
          await httpService.post('${baseModulePrefix}review', body: body);
      return response;
    } catch (e) {
      log(e.toString());
      throw Exception(e);
    }
  }

  void dispose() {
    httpService.dispose();
  }
}
```

Gambar 3. 14 Kode *Repository Queue*

Gambar 3.13 mendefinisikan *repository* kelas `QueueRepository` yang berfungsi sebagai abstraksi untuk menangani interaksi dengan *API* terkait antrian (*queue*) di dalam aplikasi. Kelas ini menggunakan `HttpService` untuk melakukan operasi *HTTP* seperti `*POST*` dan `*GET*` ke berbagai *endpoint*.

Kelas ini memiliki beberapa fungsi yang berhubungan dengan antrian, seperti `addQueue` yang digunakan untuk menambah data antrian baru dengan parameter yang diperlukan seperti `serviceType`, `serviceName`, `complaint`, dan `date`. Ada juga fungsi untuk mengambil data tampilan loket (A, B, dan C) melalui `getDisplayLocketA`, `getDisplayLocketB`, dan `getDisplayLocketC` yang masing-masing melakukan permintaan `*GET*` ke *endpoint* terkait untuk menampilkan antrian pada loket tersebut. Selain itu, terdapat fungsi `getHistoryQueueList` yang digunakan untuk mengambil daftar riwayat antrian berdasarkan `status`, `offset`, dan `limit` yang ditentukan, serta fungsi `getLatestCalledQueue` untuk mendapatkan antrian terbaru yang dipanggil. Fungsi `reviewUserQueue` digunakan untuk mengirim ulasan pengguna terhadap antrian tertentu dengan memberikan skor ulasan.

Setiap fungsi dibungkus dalam blok `try-catch` untuk menangani kesalahan, dan kesalahan yang tertangkap akan dicatat menggunakan fungsi `log` serta dilempar sebagai pengecualian baru. Fungsi `dispose` digunakan untuk membersihkan sumber daya `HttpService` ketika kelas ini tidak lagi diperlukan.



```
void initSocket({required BuildContext context}) {
  Debouncer debounceGetLatestCallQueue = Debouncer(
      action: () {
        getLatestCalledQueue(context: context);
      },
      milliseconds: 300);
  Debouncer debounceDisplayLocketListA = Debouncer(
      action: () {
        getDisplayLocketListA(context: context);
      },
      milliseconds: 300);
  Debouncer debounceDisplayLocketListB = Debouncer(
      action: () {
        getDisplayLocketListB(context: context);
      },
      milliseconds: 300);
  Debouncer debounceDisplayLocketListC = Debouncer(
      action: () {
        getDisplayLocketListC(context: context);
      },
      milliseconds: 300);
  socket = io.io("ws://103.181.143.51:3000", <String, dynamic>{
    'autoConnect': true,
    'transports': ['websocket'],
  });
  socket.onConnect((data) {
    log("Connection established");
    debounceGetLatestCallQueue.run();
    debounceDisplayLocketListA.run();
    debounceDisplayLocketListB.run();
    debounceDisplayLocketListC.run();
  });
  socket.onDisconnect((data) {
    log("connection Disconnection $data");
    socket.close();
    if (data.toString() != 'io client disconnect') reconnectSocket();
  });
  socket.onConnectError((err) {
    log("connection Error: $err");
    socket.close();
    reconnectSocket();
  });
  socket.onError((err) {
    log(err.toString());
    socket.close();
    reconnectSocket();
  });
  socket.on('update-locket-a', (data) {
    debounceDisplayLocketListA.run();
  });
  socket.on('update-locket-b', (data) {
    debounceDisplayLocketListB.run();
  });
  socket.on('update-locket-c', (data) {
    debounceDisplayLocketListC.run();
  });
  socket.on('update-latest-called-queue', (data) {
    debounceGetLatestCallQueue.run();
  });
  socket.connect();
}

void reconnectSocket() {
  // Add a delay before reconnecting
  Future.delayed(const Duration(milliseconds: 300), () {
    socket.connect();
    log("Reconnecting ...");
  });
}
```

```
Future<void> getDisplayLocketListA({required BuildContext context}) async {
  try {
    ResponseAPI response = await queueRepository.getDisplayLocketA();
    if (!response.status) {
      Future.delayed(Duration.zero, () {
        Flushbar(
          margin: const EdgeInsets.fromLTRB(20, 10, 20, 10),
          flushbarPosition: FlushbarPosition.TOP,
          borderRadius: BorderRadius.circular(8),
          message: response.message,
          duration: const Duration(seconds: 3),
        ).show(context);
      });
      return;
    }
    List data = response.data;
    locketA.clear();

    if (data.isEmpty) {
      notifyListeners();
      return;
    }

    locketA.addAll(data.map((json) => Queue.fromJson(json)));
    notifyListeners();
  } catch (e) {
    log(e.toString());
    throw Exception(e);
  }
}

Future<void> getDisplayLocketListB({required BuildContext context}) async {
  try {
    ResponseAPI response = await queueRepository.getDisplayLocketB();
    if (!response.status) {
      Future.delayed(Duration.zero, () {
        Flushbar(
          margin: const EdgeInsets.fromLTRB(20, 10, 20, 10),
          flushbarPosition: FlushbarPosition.TOP,
          borderRadius: BorderRadius.circular(8),
          message: response.message,
          duration: const Duration(seconds: 3),
        ).show(context);
      });
      return;
    }
    List data = response.data;
    locketB.clear();

    if (data.isEmpty) {
      notifyListeners();
      return;
    }
    locketB.addAll(data.map((json) => Queue.fromJson(json)));
    notifyListeners();
  } catch (e) {
    log(e.toString());
    throw Exception(e);
  }
}

Future<void> getDisplayLocketListC({required BuildContext context}) async {
  try {
    ResponseAPI response = await queueRepository.getDisplayLocketC();
    if (!response.status) {
      Future.delayed(Duration.zero, () {
        Flushbar(
          margin: const EdgeInsets.fromLTRB(20, 10, 20, 10),
          flushbarPosition: FlushbarPosition.TOP,
          borderRadius: BorderRadius.circular(8),
          message: response.message,
          duration: const Duration(seconds: 3),
        ).show(context);
      });
      return;
    }
    List data = response.data;
    locketC.clear();

    if (data.isEmpty) {
      notifyListeners();
      return;
    }
    locketC.addAll(data.map((json) => Queue.fromJson(json)));
    notifyListeners();
  } catch (e) {
    log(e.toString());
    throw Exception(e);
  }
}
```

Gambar 3. 15 Cuplikan Kode *Controller Home*

Fungsi `initSocket` pada **Gambar 3.15** digunakan untuk menginisialisasi koneksi *WebSocket* ke *server* menggunakan library `socket_io_client`. Fungsi ini menghubungkan aplikasi dengan *server* pada alamat `ws://103.181.143.51:3000` dengan protokol *websocket* yang memastikan bahwa komunikasi data yang dilakukan adalah *real-time*. Saat koneksi berhasil dibuat, beberapa tindakan yang ditunda menggunakan `Debouncer` akan dieksekusi, seperti memanggil data antrian terbaru dan daftar tampilan locket A, B, dan C. Fungsi ini juga menangani berbagai kejadian seperti ketika koneksi terputus, terjadi kesalahan koneksi, atau terdapat

kesalahan umum dengan menutup koneksi dan mencoba untuk menghubungkan kembali melalui fungsi `reconnectSocket`.

Fungsi ini juga mendengarkan pembaruan dari *server* terkait antrian loket tertentu, dan ketika *server* mengirimkan notifikasi pembaruan (misalnya pada event `''update-locket-a''`, `''update-locket-b''`, `''update-locket-c''`, atau `''update-latest-called-queue''`), fungsi yang bersesuaian akan dipanggil melalui `Debouncer` untuk memperbarui data tampilan secara efisien. Akhirnya, koneksi *WebSocket* ini diaktifkan melalui pemanggilan fungsi `socket.connect()` pada akhir dari fungsi `initSocket`.

Fungsi `getLatestCalledQueue`, `getDisplayLocketListA`, `getDisplayLocketListB`, dan `getDisplayLocketListC` masing-masing bertanggung jawab untuk mengambil data antrian terbaru dan daftar tampilan locket dari *server* melalui permintaan *API*. Pada setiap fungsi, permintaan *HTTP* dibuat menggunakan `queueRepository`, yang berkomunikasi dengan *server* untuk mendapatkan data yang sesuai. Misalnya, `getLatestCalledQueue` mengambil data tiket antrian terkini yang sedang dipanggil oleh sistem, sementara fungsi `getDisplayLocketListA`, `getDisplayLocketListB`, dan `getDisplayLocketListC` mengambil daftar tiket antrian paling terkini dan 1 tiket antrian selanjutnya yang tampil di loket A, B, dan C.

Jika respons dari *server* menunjukkan bahwa permintaan berhasil, data yang diterima kemudian diproses dan diperbarui dalam *model Queue*, seperti daftar `latestCalledQueue` untuk antrian terbaru dan daftar `locketA`, `locketB`, atau `locketC` untuk masing-masing loket. Apabila terdapat kesalahan atau respons menunjukkan kegagalan, pesan kesalahan akan ditampilkan kepada pengguna menggunakan `Flushbar`, dan data yang sudah ada sebelumnya tidak akan diperbarui. Setiap fungsi juga menggunakan metode `notifyListeners()` untuk memberi tahu sistem bahwa data telah diperbarui, yang memungkinkan antarmuka pengguna untuk memperbarui tampilannya secara otomatis.

```dart
@override
Widget build(BuildContext context) {
  return SafeArea(
      child: Scaffold(
    body: SingleChildScrollView(
      child: Column(
        children: [
          const HeaderHome(),
          const SizedBox(
            height: 12,
          ),
          Consumer<HomeController>(
            builder: (context, value, child) {
              return CardLoketAntrian(
                listLocket: value.locketA,
                locketNumber: "CS1",
              );
            },
          ),
          Consumer<HomeController>(
            builder: (context, value, child) {
              return CardLoketAntrian(
                listLocket: value.locketB,
                locketNumber: "CS2",
              );
            },
          ),
          Consumer<HomeController>(
            builder: (context, value, child) {
              return CardLoketAntrian(
                listLocket: value.locketC,
                locketNumber: "CS3",
              );
            },
          ),
        ],
      ),
    ),
  ));
}
```

Gambar 3. 16 Cuplikan Kode *View Home*

Fungsi `build` pada **Gambar 3.15** tersebut merupakan cuplikan kode *view home* yang bertanggung jawab untuk membangun antarmuka pengguna (*UI*) untuk halaman *home* aplikasi. *Widget* ini membungkus komponen *UI* yang lebih kecil, yaitu `HomeView`. Pada `HomeView`, komponen utama terdiri dari sebuah `Scaffold` dengan tampilan `AppBar` yang diisi dengan berbagai bagian seperti gambar *header*, informasi antrian terbaru, serta kartu antrian dari beberapa locket (CS1, CS2, CS3). Masing-masing bagian ini ditampilkan dalam kolom yang bisa di-scroll, menggunakan widget `SingleChildScrollView`.

Bagian *header* ditampilkan menggunakan widget `HeaderHome`, yang mencakup nama pengguna yang diambil dari `controller.user.fullName`, dan tombol notifikasi. Selain itu, bagian utama dari *UI* menampilkan nomor antrian terbaru beserta locket yang dituju menggunakan *widget* `Stack` untuk mengatur tata letaknya. Dalam halaman *home*, *widget* `CardLoketAntrian` digunakan tiga kali

untuk menampilkan antrian pada masing-masing loket (A, B, dan C), yang datanya diperoleh dari `HomeController` melalui `Consumer` untuk memperbarui *UI* secara otomatis sesuai dengan perubahan data masing-masing loket.

Gambar 3. 17 Kode *Component* CardLocketAntrian

Kode pada **Gambar 3.16** mendefinisikan sebuah *component widget* `CardLoketAntrian` yang digunakan untuk menampilkan informasi mengenai antrian di setiap loket dalam bentuk kartu. *Widget* ini menerima dua parameter, yaitu `listLocket`, yang merupakan daftar antrian dari *model* `Queue`, dan `locketNumber`, yang merupakan nomor loket yang terkait.

Dalam metode `build`, kartu antrian ditampilkan menggunakan *widget* `Container` yang diberi *padding* dan *margin*. Di dalamnya terdapat dua bagian utama yang diatur dengan `Row` untuk diposisikan secara horizontal: bagian informasi antrian dan bagian untuk menampilkan antrian selanjutnya. Bagian informasi antrian ditampilkan dalam sebuah `Column`, yang berisi tiga elemen: status antrian (ditampilkan dengan gradient warna dan ikon status), nomor tiket dari antrian terbaru, dan nomor loket. Warna, teks, dan ikon dalam elemen-elemen ini akan berubah tergantung pada status antrian yang ada di indeks pertama dari `listLocket`.

Di sebelah kanan, kolom kedua menampilkan antrian selanjutnya jika ada lebih dari satu antrian di `listLocket`, dan jika tidak ada, akan menampilkan teks '---'. Dengan demikian, *widget* ini berfungsi untuk menampilkan informasi terkini dari antrian di setiap loket serta memberikan informasi mengenai antrian yang akan datang.

3.4. Pengujian Produk

Penulis menggunakan metode *Black Box Testing* untuk pengujian produk. Pada tahap ini, dilakukan beberapa skenario pengujian untuk mengetahui apakah skenario tersebut berhasil dijalankan atau tidak. Berikut adalah beberapa skenario pengujian yang dilakukan baik untuk skenario pengujian aplikasi *web* manajemen sistem antrian untuk admin dan aplikasi *mobile* pendaftaran antrian untuk pengunjung.

Tabel 3. 1 Tabel Pengujian Aplikasi *Web* Manajemen Sistem Antrian

Item Uji	Masukkan	Hasil yang diharapkan	Hasil yang didapat	Kesimpulan
Login	- Memasukkan *email* dan password - Klik tombol *login*	Sistem melakukan verifikasi apakah *email* dan password yang dimasukkan sesuai dengan *database*. Apabila data berhasil diverifikasi, maka akan otomatis diarahkan ke halaman dashboard	Sistem melakukan verifikasi apakah *email* dan password yang dimasukkan sesuai dengan *database*. Data berhasil diverifikasi dan otomatis diarahkan ke halaman utama	Berhasil
Melihat daftar antrian aktif pengunjung	Klik halaman dashboard pada sidebar	Menampilkan daftar antrian aktif pengunjung hari ini yang terdapat di database	Ditampilkan daftar antrian aktif pengunjung hari ini yang terdapat di database	Berhasil
Memanggil nomor antrian teratas	Klik ikon mic pada kolom `panggil` di halaman daftar antrian aktif pengunjung	Memanggil nomor antrian teratas yang terdapat di database melalui speaker	Nomor antrian teratas yang terdapat di database dipanggil melalui speaker	Berhasil
Melayani nomor antrian teratas	Klik tombol `layani` pada kolom `action` di halaman daftar antrian aktif pengunjung	Mengubah status nomor antrian teratas yang terdapat di database menjadi `dilayani`	Status nomor antrian teratas yang terdapat di database terubah menjadi `dilayani`	Berhasil
Melihat detail infomasi nomor	Klik tombol `lihat` pada kolom `detail` di halaman daftar	Menampilkan *popup modal* berisikan informasi antrian dan informasi pengunjung	*Popup modal* berisikan informasi antrian dan informasi pengunjung ditampikan	Berhasil

antrian teratas	antrian aktif pengunjung			
Memasukkan catatan akhir layanan pada nomor antrian teratas	- Klik tombol `lihat` pada kolom `detail` di halaman daftar antrian aktif pengunjung - Lakukan input data pada modal detail di kolom `Catatan Akhir Pelayanan` - Klik tombol `update`	Memperbarui data catatan akhir layanan yang terdapat di database berdasarkan data yang diinput	Data catatan akhir layanan yang terdapat di database berhasil diperbarui berdasarkan data yang diinput	Berhasil
Menyelesaikan pelayanan nomor antrian teratas	Klik tombol `selesai` pada kolom `action` di halaman daftar antrian aktif pengunjung	Mengubah status nomor antrian teratas yang terdapat di database menjadi `selesai` dan nomor antrian berikutnya menjadi nomor antrian teratas	Status nomor antrian teratas yang terdapat di database berhasil terubah menjadi `selesai` dan nomor antrian berikutnya menjadi nomor antrian teratas	Berhasil
Melewati pelayanan nomor antrian teratas	Klik tombol `lewati` pada kolom `action` di halaman daftar antrian aktif pengunjung	Mengubah status nomor antrian teratas yang terdapat di database menjadi `dilewati` dan nomor antrian berikutnya menjadi nomor antrian teratas	Status nomor antrian teratas yang terdapat di database berhasil terubah menjadi `dilewati` dan nomor antrian berikutnya menjadi nomor antrian teratas	Berhasil

Menampilkan rekapitulasi antrian	- Klik halaman rekap antrian pada sidebar - Tentukan periode antrian - Tentukan loket - Klik tombol submit	Menampilkan daftar antrian pengunjung yang terdapat di database berdasarkan periode dan loket yang ditentukan	Daftar antrian pengunjung yang terdapat di database berdasarkan periode dan loket yang ditentukan berhasil ditampilkan	Berhasil
Mengunduh rekapitulasi antrian	- Klik halaman rekap antrian pada sidebar - Tentukan periode antrian - Tentukan loket - Klik tombol download	Mengunduh daftar antrian pengunjung yang terdapat di database berdasarkan periode dan loket yang ditentukan	Daftar antrian pengunjung yang terdapat di database berdasarkan periode dan loket yang ditentukan berhasil diunduh	Berhasil
Menampilkan grafik visualisasi data antrian	- Klik halaman visualisasi data pada sidebar - Tentukan periode antrian - Klik tombol submit	Menampilkan grafik visualisasi data antrian pengunjung yang terdapat di database berdasarkan periode yang ditentukan	Grafik visualisasi data antrian pengunjung yang terdapat di database berdasarkan periode yang ditentukan berhasil ditampilkan	Berhasil
Logout	Klik nama akun pada *topbar* kanan halaman *website* lalu klik tombol *logout*	Berhasil keluar dari halaman utama *website* dan menuju ke halaman *login*	Berhasil keluar dari halaman utama *website* dan menuju ke halaman *login*	Berhasil

Tabel 3. 2 Tabel Pengujian Aplikasi *Mobile* Pendaftaran Antrian

Item Uji	Masukkan	Hasil yang diharapkan	Hasil yang didapat	Kesimpulan
Register	- Memasukkan data nama lengkap, NIM/NIP, nomor telepon, fakultas/satker, status pengunjung, *username*, *email*, *password*, dan konfirmasi *password*. - Klik tombol daftar	Sistem memverifikasi data yang telah diinput apakah sudah terisi semua, dan data yang diberikan sesuai dengan yang diharapkan database. Setelah terverifikasi, otomatis diarahkan ke halaman utama.	Sistem berhasil memverifikasi data yang telah diinput apakah sudah terisi semua, dan data yang diberikan sesuai dengan yang diharapkan database. Setelah terverifikasi, otomatis diarahkan ke halaman utama.	Berhasil
Login	- Memasukkan *email* dan password - Klik tombol *login*	Sistem melakukan verifikasi apakah *email* dan password yang dimasukkan sesuai dengan *database*. Apabila data berhasil diverifikasi, maka akan otomatis diarahkan ke halaman dashboard	Sistem melakukan verifikasi apakah *email* dan password yang dimasukkan sesuai dengan *database*. Data berhasil diverifikasi dan otomatis diarahkan ke halaman utama	Berhasil
Melihat halaman beranda	Klik tombol *icon* `Beranda` pada	Menampilkan halaman beranda yang berisikan nama lengkap akun	Ditampilkan halaman beranda yang berisikan nama lengkap akun	Berhasil

	bottom navigation bar	pengunjung, tombol *icon* notifikasi, *card* informasi nomor antrian terkini yang sedang dipanggil, *card* informasi antrian untuk loket CS1, CS2, dan CS3	pengunjung, tombol *icon* notifikasi, *card* informasi nomor antrian terkini yang sedang dipanggil, *card* informasi antrian untuk loket CS1, CS2, dan CS3	
Melihat daftar notifikasi	Klik tombol *icon* notifikasi pada ujung atas kanan halaman beranda	Menampilkan daftar notifikasi antrian milik akun pengunjung yang terdapat di database	Ditampilkan daftar notifikasi antrian milik akun pengunjung yang terdapat di database	Berhasil
Mendaftar Antrian	- Klik tombol *icon* `Daftar` pada *bottom navigation bar* - Tentukan tanggal antrian - Tentukan jenis layanan - Pilih layanan yang dibutuhkan - Isi kolom heluhan (opsional) - Klik tombol `daftar` pada halaman tersebut	Menampilkan *popup* notifikasi bahwa tiket antrian berhasil dibuat	Ditampilkan *popup* notifikasi bahwa tiket antrian berhasil dibuat	Berhasil
Melihat halaman riwayat	Klik tombol *icon* `Riwayat` pada *bottom navigation bar*	Menampilkan daftar riwayat akun pengunjung yang disajikan dengan 2 *tab,*	Ditampilkan daftar riwayat akun pengunjung yang disajikan dengan 2 *tab,*	Berhasil

		yaitu riwayat sedang berjalan dan riwayat sudah selesai	yaitu riwayat sedang berjalan dan riwayat sudah selesai	
Melakukan review pelayanan riwayat antrian	- Klik tombol *icon* `Riwayat` pada *bottom navigation bar* - Klik *tab* riwayat sudah selesai - Tentukan dan klik *card* antrian yang belum di-*review* - Tentukan dan klik berapa banyak nilai bintang yang akan diberikan untuk layanan tersebut - Klik tombol `kirim`	Nilai antrian di-*update* ke *database* dan *card* riwayat antrian menampilkan nilai *review*-nya	Nilai antrian berhasil di-*update* ke *database* dan *card* riwayat antrian ditampilkan nilai *review*-nya	Berhasil
Melihat Data Informasi pengguna	- Klik tombol *icon* `Akun` pada *bottom navigation bar* - Klik tombol `info profil` pada halaman akun	Menampilkan data Informasi terkait profil akun pengunjung	Ditampilkan data Informasi terkait profil akun pengunjung	Berhasil
Mengubah data Informasi pengguna	- Klik tombol *icon* `Akun` pada *bottom navigation bar*	Berhasil menyimpan hasil pengubahan data profil akun pengunjung ke dalam *database*	Berhasil menyimpan hasil pengubahan data profil akun pengunjung ke dalam *database*	Berhasil

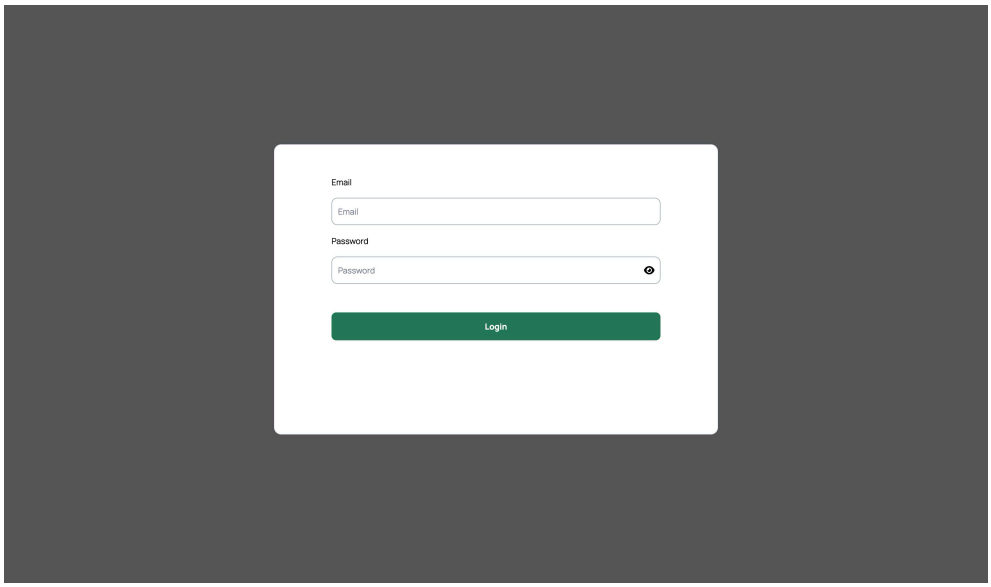
	- Klik tombol `info profil` pada halaman akun - Ubah kolom data profil yang ingin dirubah - Klik tombol `simpan perubahan`			
Mengubah *password* pengguna	- Klik tombol *icon* `Akun` pada *bottom navigation bar* - Klik tombol `Ganti *Password*` pada halaman akun - Masukkan *password* saat ini - Masukkan *password* baru - Masukkan konfirmasi *password* baru - Klik tombol `simpan perubahan`	Berhasil menyimpan hasil pengubahan data *password* akun pengunjung ke dalam *database*	Berhasil menyimpan hasil pengubahan data *password* akun pengunjung ke dalam *database*	Berhasil
Melihat *FAQ*	- Klik tombol *icon* `Akun` pada *bottom navigation bar*	Menampilkan daftar *FAQ* berdasarkan jenis *FAQ* dan pertanyaan	Ditampilkan daftar *FAQ* berdasarkan jenis *FAQ* dan pertanyaan	Berhasil

	- Klik tombol `*FAQ*` pada halaman akun - Klik jenis *FAQ* yang diinginkan - Klik pertanyaan *FAQ* yang diinginkan	yang diinginkan jawabannya	yang diinginkan jawabannya	
Logout	- Klik tombol *icon* `Akun` pada *bottom navigation bar* - Klik tombol `Keluar` pada halaman akun	Berhasil *logout* dan diarahkan kembali ke halaman *login*	Berhasil *logout* dan diarahkan kembali ke halaman *login*	Berhasil

3.5. Laporan Hasil Implementasi

3.5.1. Hasil Implementasi *Website* Manajemen Sistem

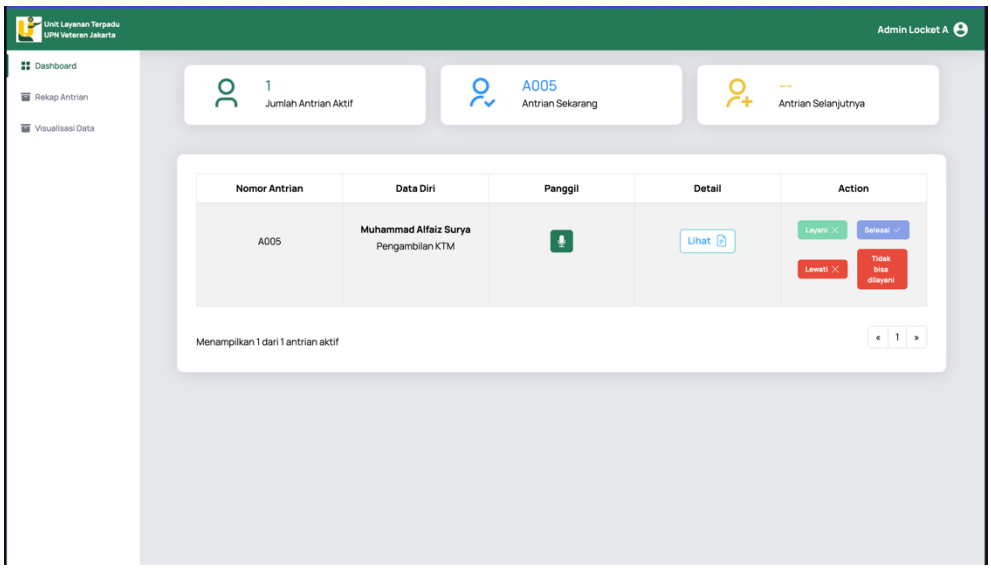
a. Halaman *Login*



Gambar 3. 18 Halaman *Login Website*

Gambar 3.17 adalah tampilan halaman *login* dimana admin ULT dapat menginput data kredensial mereka yaitu *email* dan *password*. Setelah dilakukan penginputan data, maka admin dapat mengklik tombol *login* agar selanjutnya akan diarahkan ke halaman *dashboard* apabila data yang diinput terverifikasi.

b. Halaman *Dasboard*

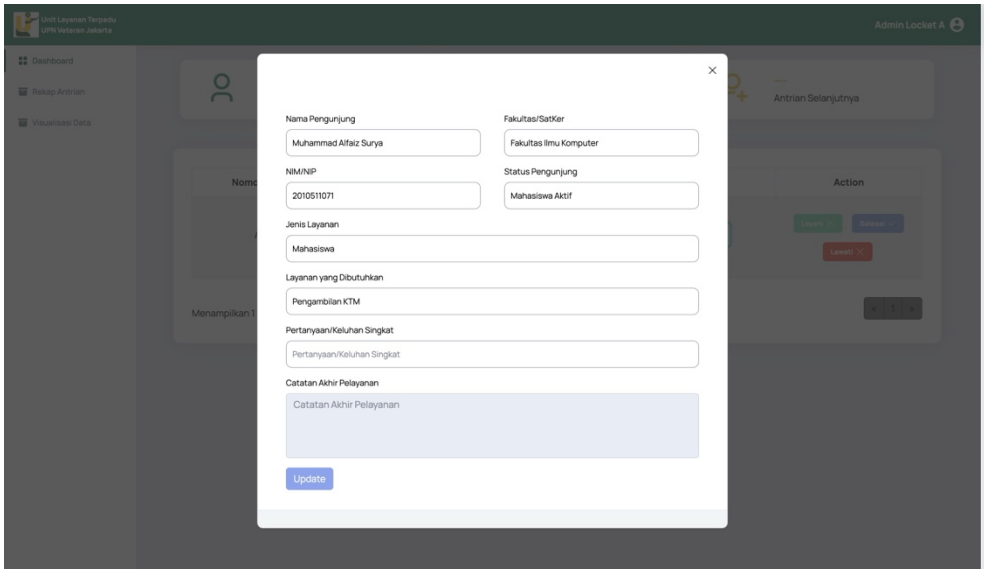


Gambar 3. 19 Halaman *Dasboard*

Gambar 3.18 adalah tampilan halaman *dasboard* dimana admin ULT dapat melihat berbagai informasi tentang antrian hari ini di loket yang

dikelola. Pada halaman ini, ditampilkan loket apa yang dikelola oleh akun admin pada *top bar* halaman. Admin juga dapat melihat informasi jumlah antrian aktif hari ini, informasi nomor antrian yang sedang atau akan dilayani sekarang, dan informasi nomor antrian selanjutnya. Selanjutnya ditampilkan pula *list* antrian berupa *table* yang menampilkan data antrian aktif hari ini. Pada *table tersebut* terdapat kolom nomor antrian, kolom data diri antrian yang berisi nama pengunjung dan layanan yang dibutuhkan, kolom panggil yang menampilkan tombol untuk memanggil nomor antrian tersebut, kolom *detail* yang berisikan tombol untuk menampilkan *popup modal detail* antrian, dan kolom action yang berisikan tombol layani untuk melayani antrian, tombol selesai untuk menyelesaikan layanan antrian, tombol lewati untuk melewati antrian, dan tombol tidak dapat dilayani untuk melewati antrian karena admin sudah tidak dapat melayani lagi untuk hari ini. Seluruh data pada halaman ini di-*update* secara *real-time*.

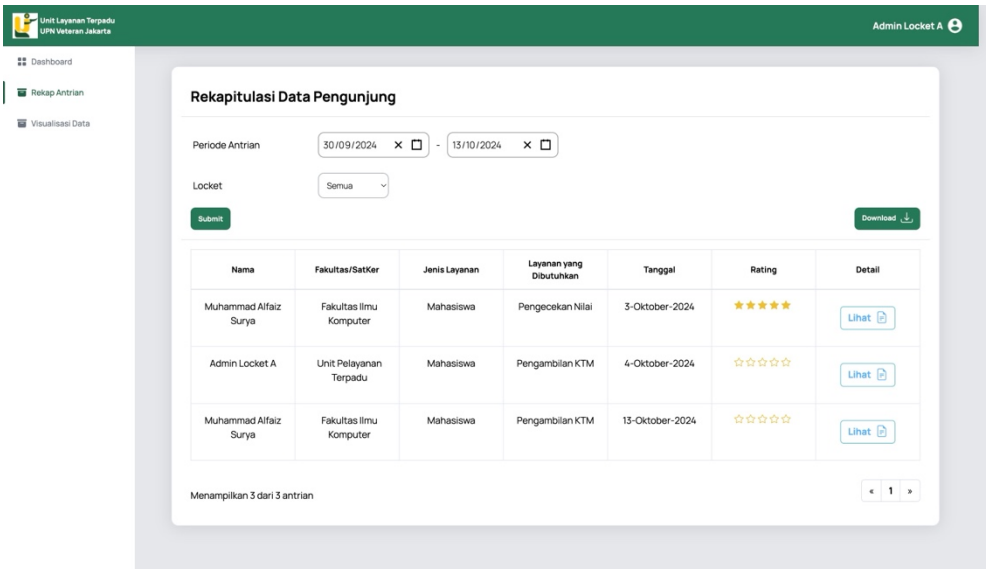
c. *Popup Modal Detail* Antrian



Gambar 3. 20 Popup Modal Detail Antrian

Gambar 3.19 adalah tampilan *popup modal detail* antrian dimana admin ULT dapat melihat berbagai informasi tentang data antrian yang dipilih pada halaman *dashboard*. Pada *modal* ini, ditampilkan data nama pengunjung, fakultas/satket pengunjung, NIM/NIP pengunjung, status pengunjung, jenis layanan, layanan yang dibutuhkan, pertanyaan/keluhan singkat yang diberikan pengujung. Selanjutnya ditampilkan pula kolom catatan akhir pelayanan yang dapat diinput dan disubmit oleh admin ketika antrian tersebut sedang dalam status dilayani.

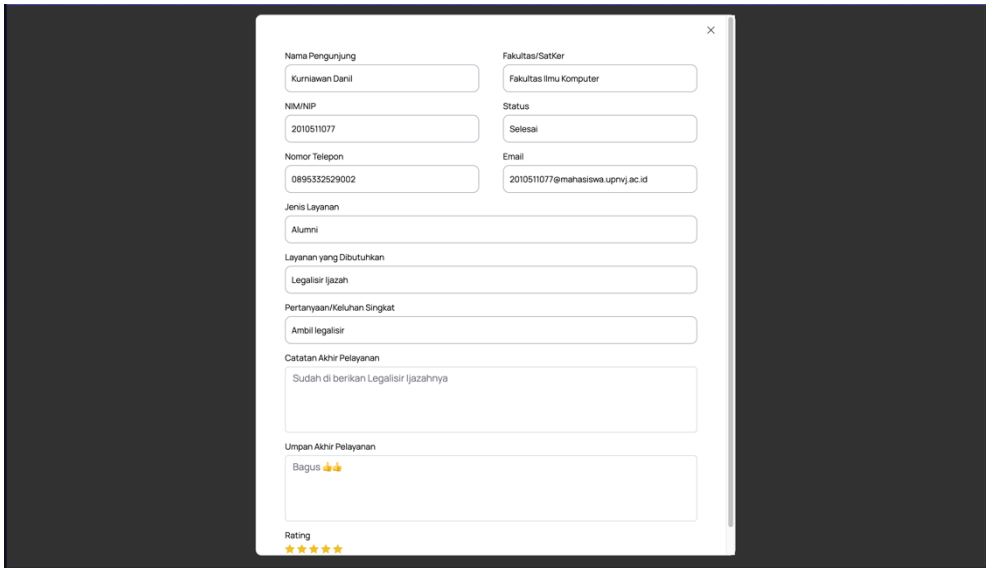
d. Halaman Rekapitulasi



Gambar 3. 21 Halaman Rekapitulasi

Gambar 3.20 adalah tampilan halaman rekapitulasi dimana admin ULT dapat melihat rekapitulasi data antrian berdasarkan periode waktu dan berdasarkan loket yang ditentukan. Pada halaman ini, ditampilkan loket apa yang dikelola oleh akun admin pada *top bar* halaman. Admin diperlukan untuk menentukan periode antrian terlebih dahulu pada *datepicker* yang ditampikan di halaman ini. Selanjutnya dapat Admin diperlukan untuk menentukan loket apa saja yang ingin direkapitulasi pada loket *selector* yang ditampikan di halaman ini. Setelah mengklik tombol submit, akan ditampilkan *list* data rekapitulasi antrian berupa *table* yang dapat di-*download* dalam format excel ketika mengklik tombol *download*. Pada *table tersebut* terdapat kolom nama, kolom fakultas/satker, kolom jenis layanan, kolom layanan yang dibutuhkan, kolom tanggal antrian, kolom rating pelayanan pada antrian, dan kolom *detail* yang berisikan tombol untuk menampilkan *popup modal detail* rekapitulasi antrian.

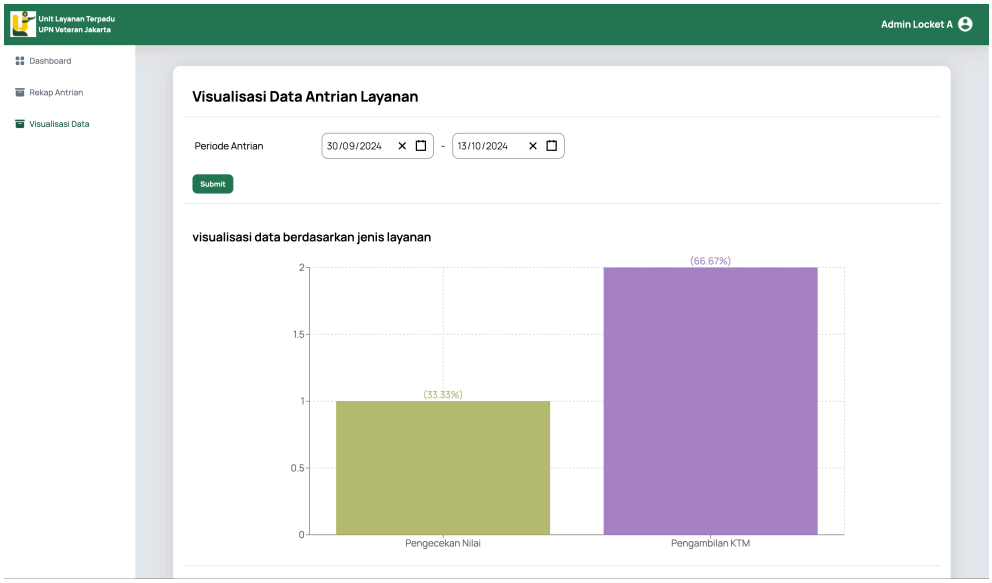
e. *Popup Modal* Detail Rekapitulasi Antrian

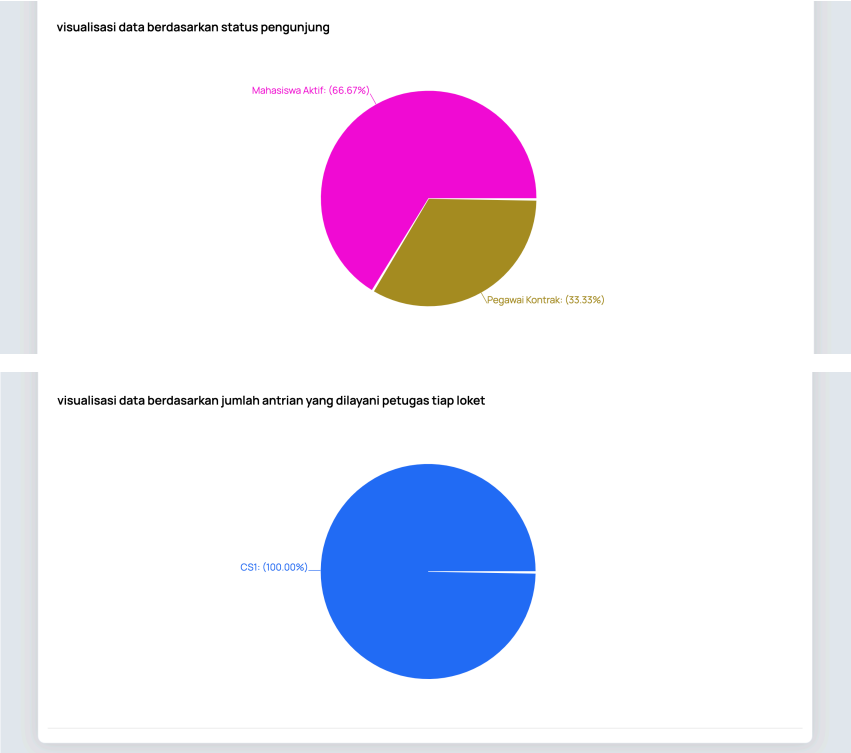


Gambar 3. 22 *Popup Modal Detail* Rekapitulasi Antrian

Gambar 3.21 adalah tampilan *popup modal detail* Rekapitulasi antrian dimana admin ULT dapat melihat berbagai informasi tentang data rekapitulasi antrian yang dipilih pada halaman Rekapitulasi. Pada *modal* ini, ditampilkan data nama pengunjung, fakultas/satker pengunjung, NIM/NIP pengunjung, status pengunjung, jenis layanan, layanan yang dibutuhkan, pertanyaan/keluhan singkat yang diberikan pengujung, catatan akhir pelayanan, dan *rating* pelayanan antrian tersebut.

f. Halaman Visualisasi Data



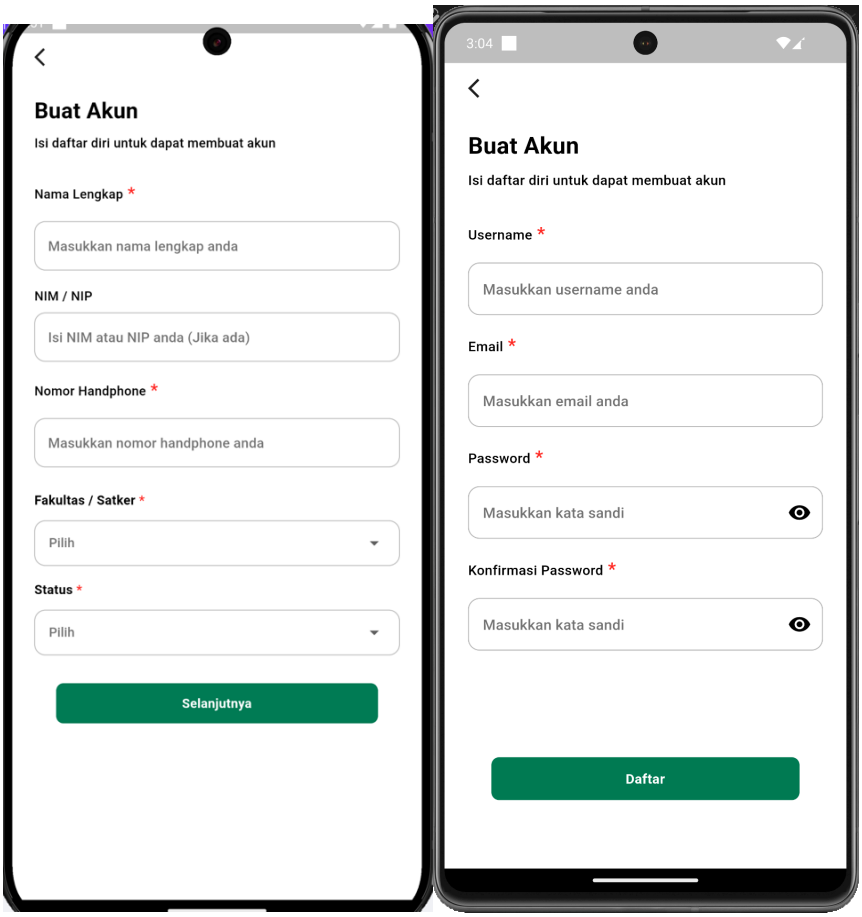


Gambar 3. 23 Halaman Visualisasi Data

Gambar 3.22 adalah tampilan halaman visualisasi data dimana admin ULT dapat melihat grafik visualisasi data antrian berdasarkan periode waktu ditentukan. Pada halaman ini, ditampilkan loket apa yang dikelola oleh akun admin pada *top bar* halaman. Admin diperlukan untuk menentukan periode antrian terlebih dahulu pada *datepicker* yang ditampikan di halaman ini. Setelah mengklik tombol submit, akan ditampilkan grafik visualisasi data berdasarkan jenis layanan yang berbentuk *bar chart,* grafik visualisasi data berdasarkan status pengunjung yang berbentuk *pie chart,* dan grafik visualisasi data berdasarkan jumlah antrian yang dilayani petugas setiap loketnya yang berbentuk *pie chart.*

3.5.2. Hasil Implementasi Aplikasi *Mobile*

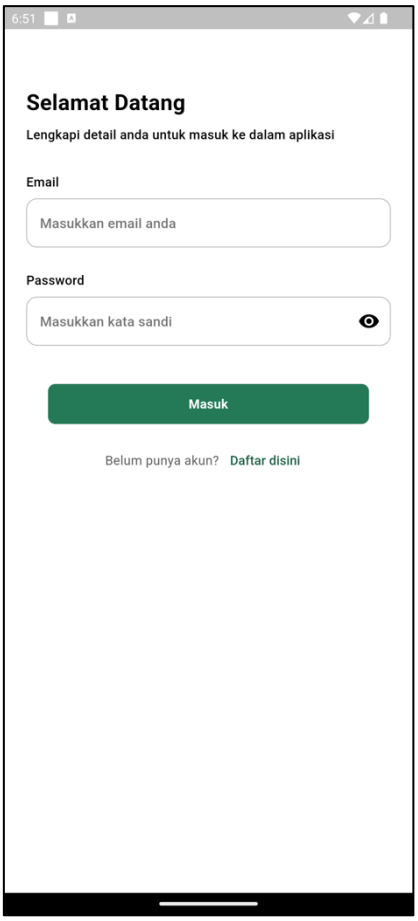
a. Halaman *Register*



Gambar 3. 24 Halaman *Register*

Gambar 3.23 menampilkan dua tampilan layar dari proses registrasi pada halaman register. Halaman pertama meminta pengguna untuk mengisi nama lengkap, NIM/NIP, nomor handphone, fakultas/satker, dan status. *Field* fakultas/satker dan status menggunakan *dropdown menu* untuk memudahkan pengguna memilih opsi yang tersedia. Setelah mengisi data tersebut, pengguna dapat melanjutkan ke tampilan kedua dengan menekan tombol "Selanjutnya" untuk menuju ke halaman kedua. Lalu, pada tampilan halaman kedua pengguna diminta untuk mengisi username, email, password, dan konfirmasi *password*. Setelah semua data terisi dengan benar, pengguna dapat menyelesaikan proses registrasi dengan menekan tombol "Daftar". Apabila data berhasil diverifikasi maka pengguna akan diarahkan ke halaman beranda.

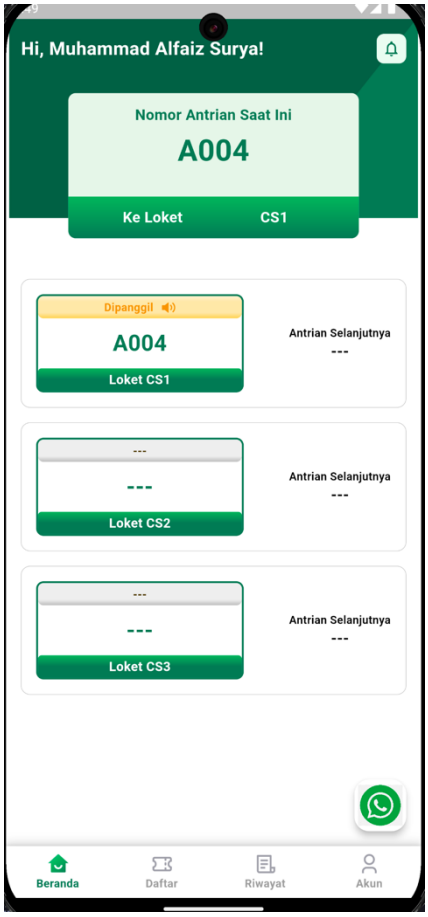
b. Halaman *Login*



Gambar 3. 25 Halaman *Login*

Gambar 3.24 menampilkan halaman *login* aplikasi. Pada halaman ini, pengguna diminta untuk memasukkan email atau nama pengguna serta kata sandi yang telah didaftarkan sebelumnya. Setelah kedua informasi tersebut dimasukkan dengan benar, pengguna dapat menekan tombol "Masuk" untuk mengakses aplikasi dan apabila data yang dimasukkan berhasil diverifikasi, maka pengguna akan diarahkan ke halaman beranda. Bagi pengguna yang belum memiliki akun, terdapat opsi "Daftar disini" yang akan mengarahkan mereka ke halaman registrasi untuk membuat akun baru.

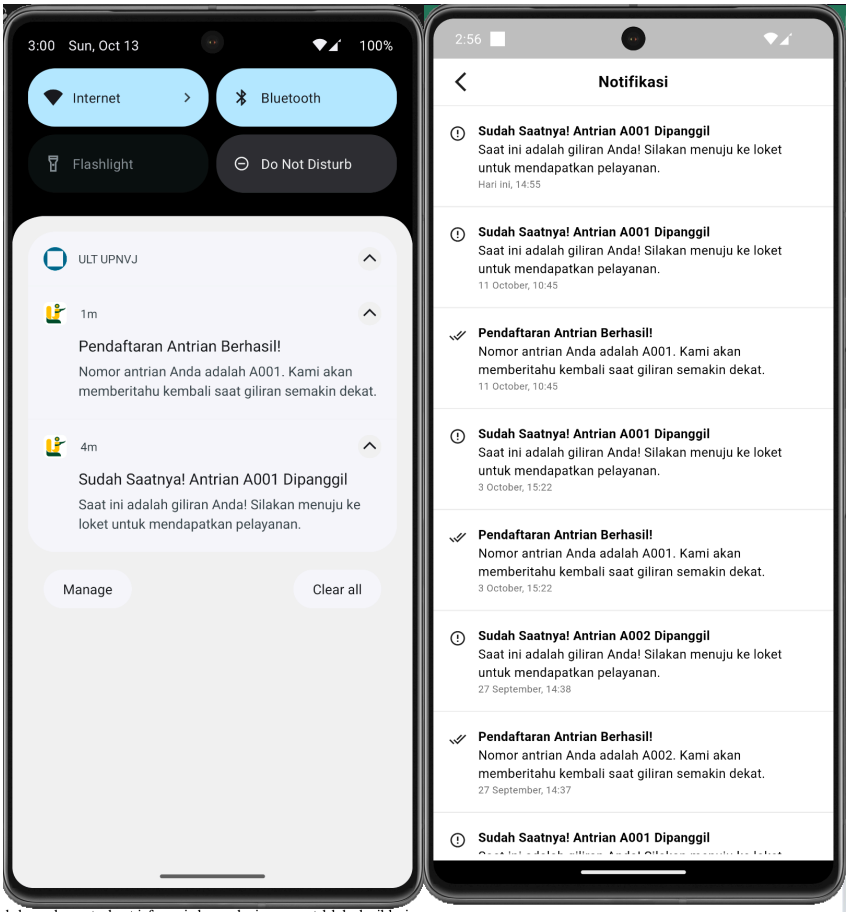
c. Halaman Beranda



Gambar 3. 26 Halaman Beranda

Gambar 3.25 menampilkan halaman beranda aplikasi yang digunakan untuk menampilkan *monitoring* antrian secara *real-time* kepada pengguna. Pada halaman ini, ditampilkan *header* yang berisikan informasi nama pengguna, tombol *icon* notifikasi untuk menuju halaman notifikasi, dan informasi nomor antrian saat ini yang sedang dipanggil. Selain itu, halaman ini juga menampilkan informasi mengenai antrian terkini pada tiap loket yang ada, seperti loket CS1, CS2 dan CS3, beserta nomor antrian yang sedang menunggu di tiap loket tersebut. Di bagian bawah layar, terdapat beberapa menu navigasi seperti Beranda, Daftar, Riwayat, dan Akun yang memungkinkan pengguna untuk mengakses fitur-fitur lain dalam aplikasi ini.

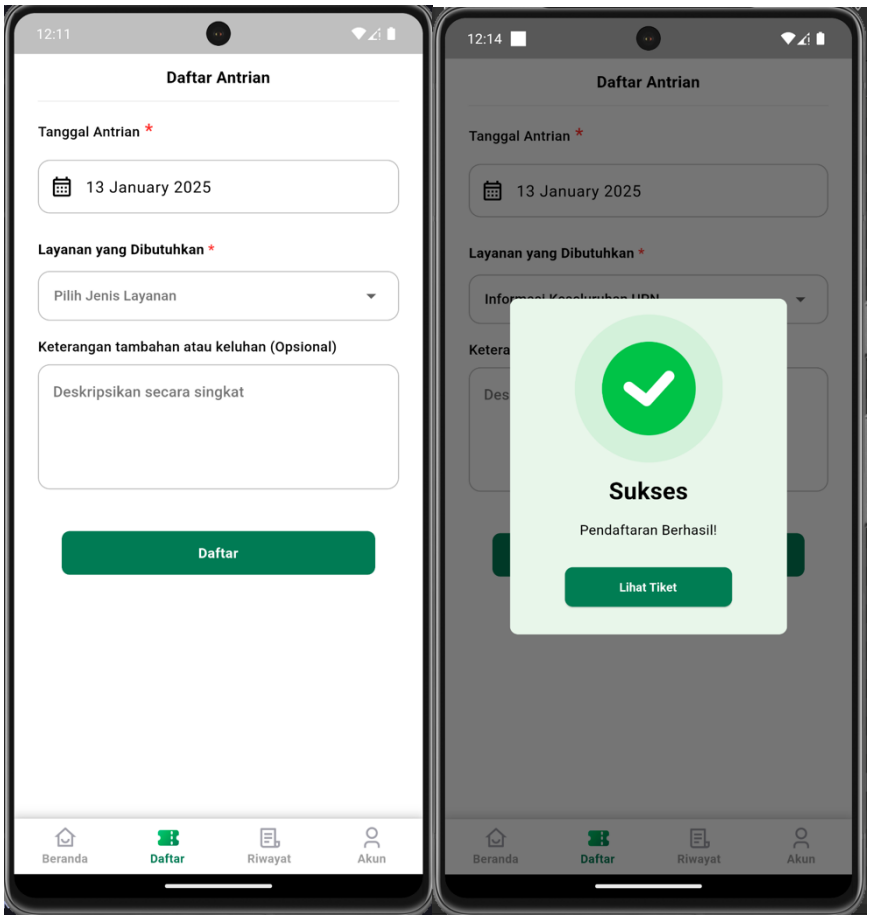
d. Halaman Notifikasi



Gambar 3. 27 Halaman Notifikasi

Gambar 3.26 menampilkan halaman notifikasi aplikasi yang digunakan untuk menampilkan daftar notifikasi antrian kepada pengguna. Pada halaman ini, ditampilkan daftar notifikasi yang tiap itemnya terdapat *icon*, judul, dan deskripsi notifikasinya. Halaman ini juga dapat diakses ketika pengguna mengklik salah satu *push* notifikasi aplikasi ULT UPNVJ dari daftar *push* notifikasi perangkat pengguna.

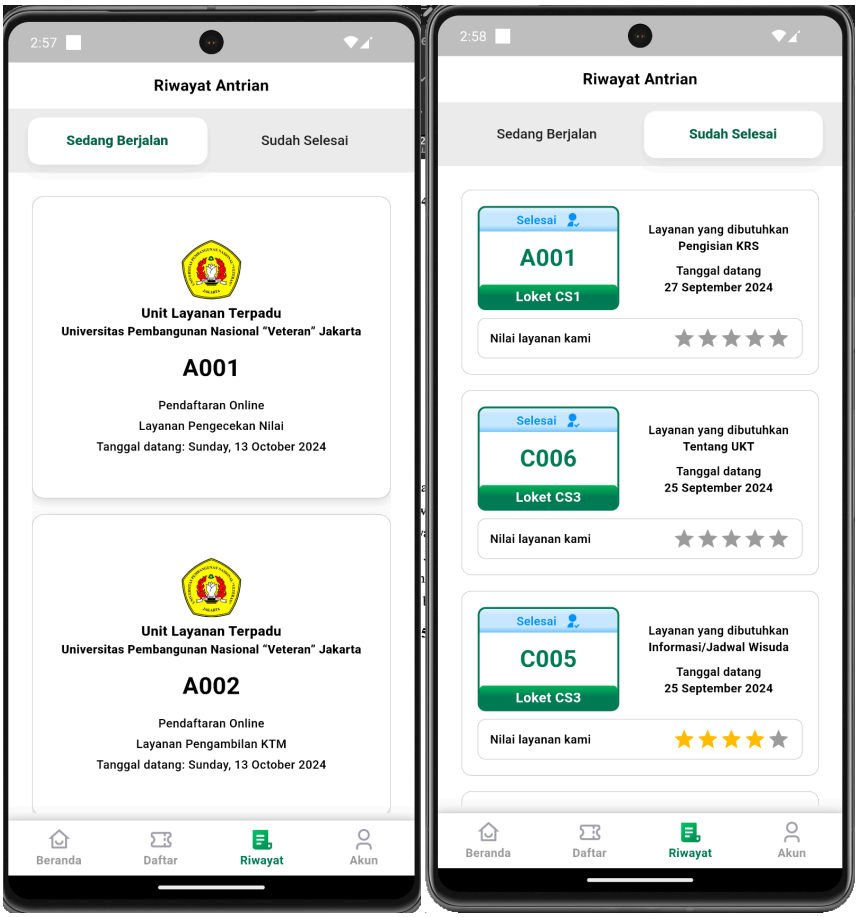
e. Halaman Daftar



Gambar 3. 28 Halaman Daftar

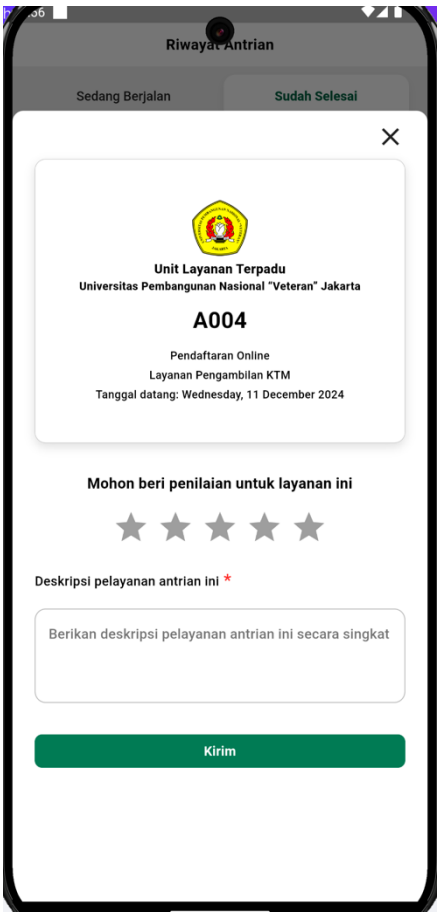
Gambar 3.27 menampilkan halaman pendaftaran antrian aplikasi yang digunakan untuk mendaftar pelayanan antrian di ULT UPNVJ. Pada halaman ini, ditampilkan *datepicker* untuk memilih tanggal antrian, *dropdown menu* untuk memilih layanan yang dibutuhkan dan *text field* untuk menginput keterangan tambahan atau keluhan (opsional). Jika semua data telah diinput, maka pengguna dapat mengklik tombol "Daftar". Selanjutnya, apabila data tersebut terverifikasi, akan dimunculkan *popup* notifikasi bahwa pendaftaran berhasil dan pengguna dapat mengklik tombol "Lihat Tiket" untuk diarahkan ke halaman riwayat tiket yang sedang berjalan.

f. Halaman Riwayat



Gambar 3. 29 Halaman Riwayat

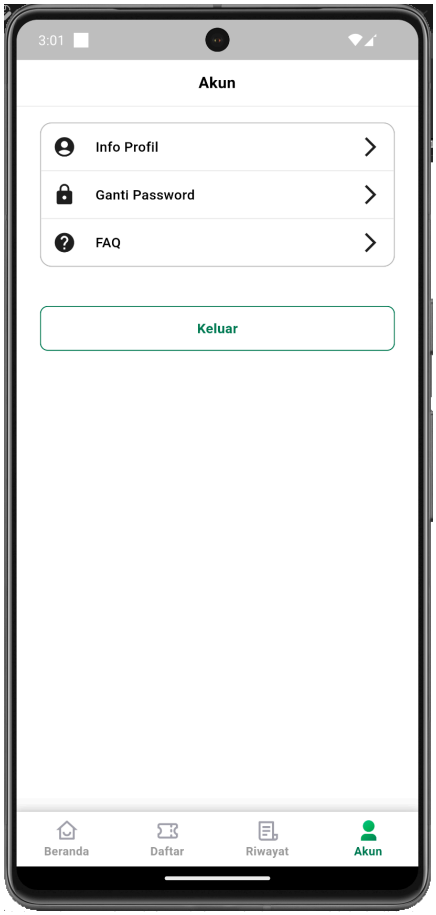
Gambar 3.28 menampilkan halaman riwayat antrian aplikasi yang berisikan daftar riwayat pelayanan antrian di ULT UPNVJ. Pada halaman ini, ditampilkan *Tab Bar* berisikan riwayat antrian "Sedang Berjalan" dan "Sudah Selesai" dimana pengguna dapat memilih salah satunya untuk menampilkan jenis daftar riwayat antrian yang diinginkan. Riwayat antrian "Sedang berjalan" berisikan daftar tiket antrian yang aktif atau sedang berjalan. Riwayat antrian "Sudah Selesai" berisikan daftar tiket antrian yang memiliki status sudah selesai atau dilewati oleh admin.



Gambar 3. 30 *Modal Review* Antrian yang Telah Selesai

Pada tiket antrian yang memiliki status selesai, pengguna dapat melakukan *review* pelayanan dengan mengklik tombol yang memiliki teks "Nilai Layanan Kami" dan *icon* bintang penilaiannya. Selanjutnya, dimunculkan *popup modal* sesuai pada **Gambar 3.29** untuk me-*review* pelayanannya dengan mengklik jumlah bintang dan mengisi deskripsi pelayanan yang diinginkan oleh pengguna. Ketika sudah yakin dengan nilai dan deskripsi *review*-nya, pengguna dapat mengklik tombol "Kirim" untuk meng-*update* nilai *review* pelayanan antrian tersebut dan akan dikembalikan ke halaman daftar riwayat antrian yang sudah selesai.

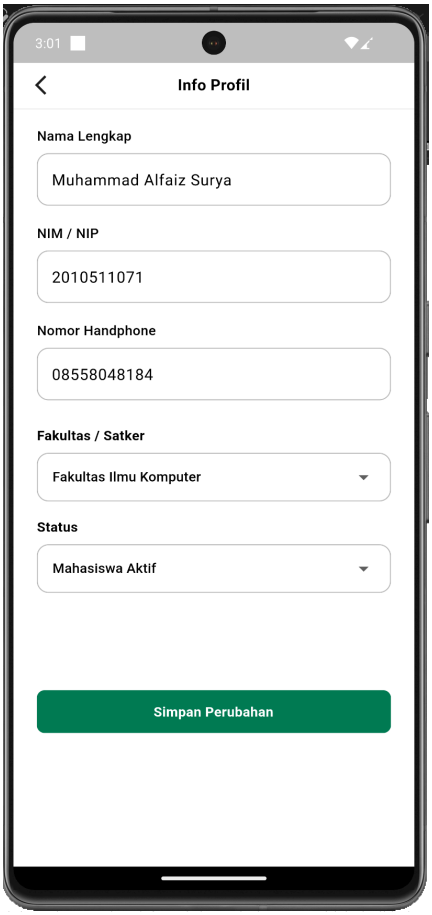
g. Halaman Akun



Gambar 3. 31 Halaman Akun

Gambar 3.30 menampilkan halaman akun yang berisikan fitur yang berhubungan dengan akun dan lainnya di aplikasi *mobile* ULT UPNVJ yang terdiri dari fitur "Info Profil", "Ganti *Password*", dan "FAQ".

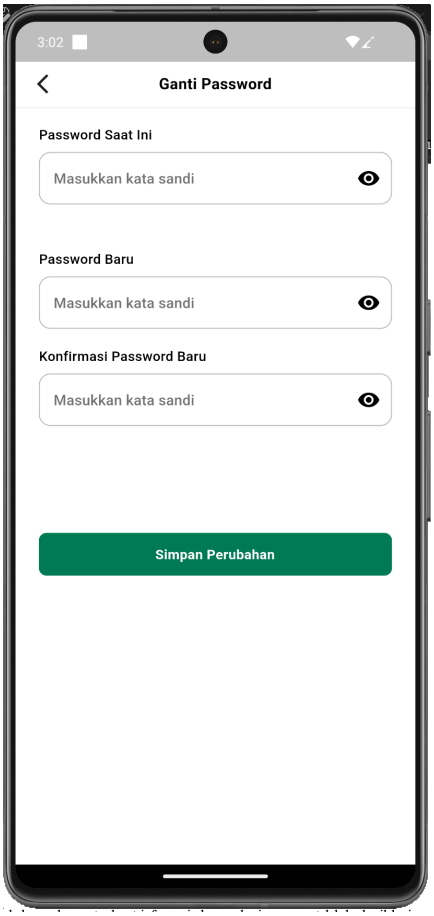
h. Halaman Info Profil



Gambar 3. 32 Halaman Info Profil

Gambar 3.31 menampilkan halaman info profil yang berisikan data informasi profil pengguna yang terdiri dari nama lengkap, NIM/NIP, Nomor *Handphone*, Fakultas/Satker, dan Status. Data dari tiap kolom teks yang ada dapat diubah dan disimpan perubahannya dengan mengklik tombol "Simpan Perubahan".

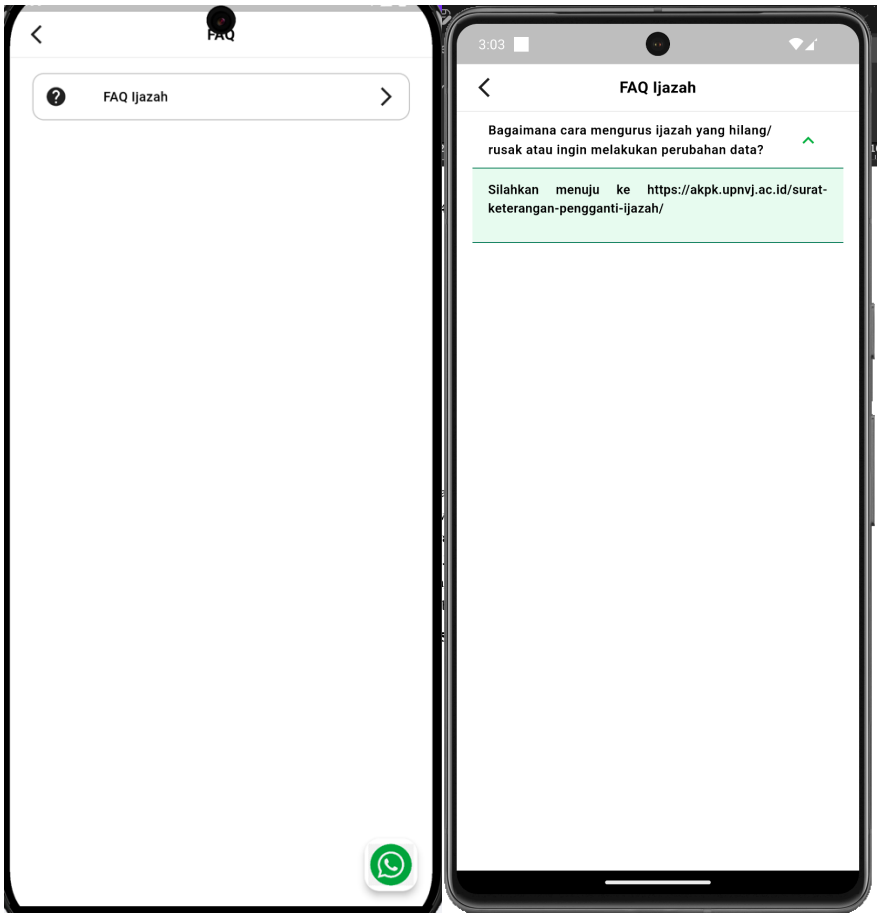
i. Halaman Ubah *Password*



Gambar 3. 33 Halaman Ubah *Password*

Gambar 3.32 menampilkan halaman ubah password yang menampilkan kolom teks untuk mengubah *password* akun pengguna. Kolom data yang perlu diisi yaitu password saat ini, password baru, dan konfirmasi password. Ketika seluruh kolomnya telah diisi, maka perubahannya dapat disimpan dengan mengklik tombol "Simpan Perubahan".

j. Halaman FAQ

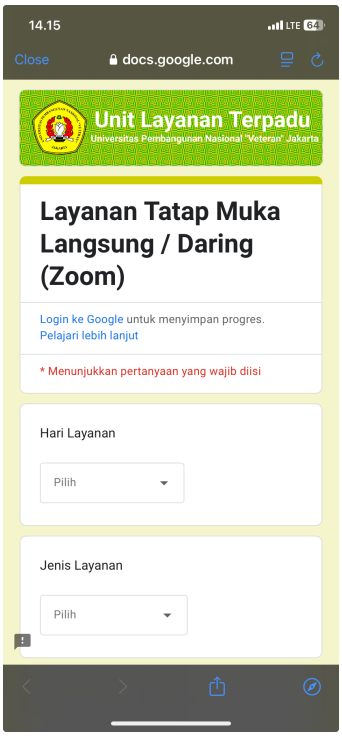
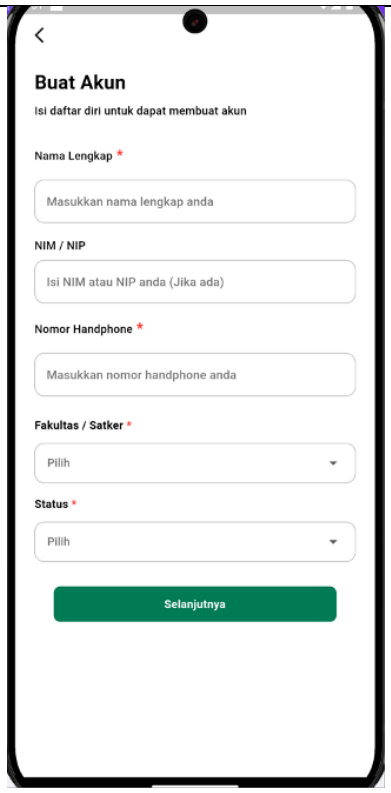
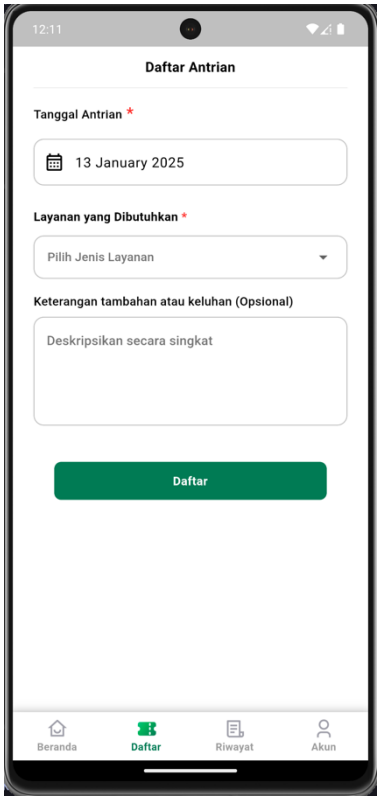


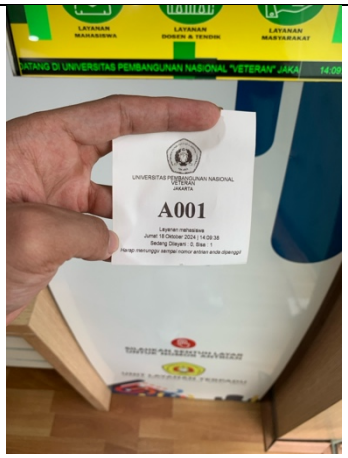
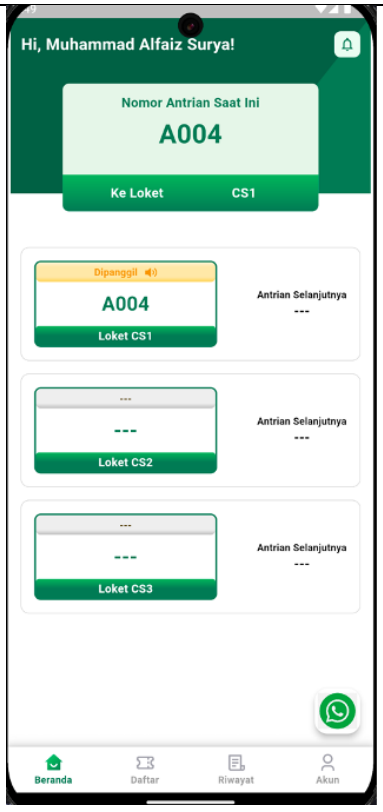
Gambar 3. 34 Halaman *FAQ*

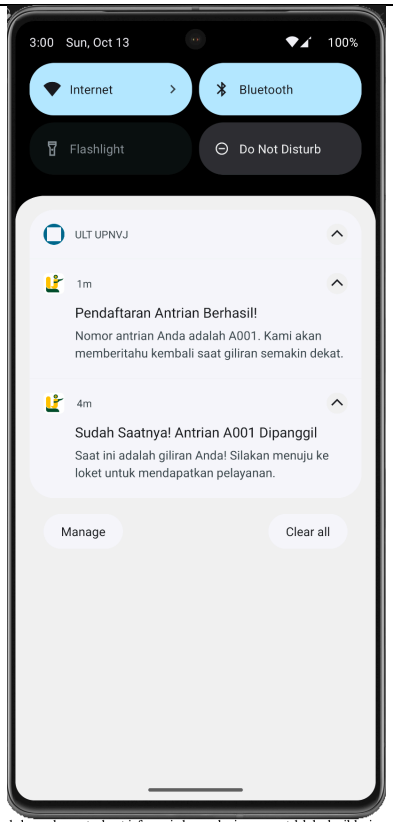
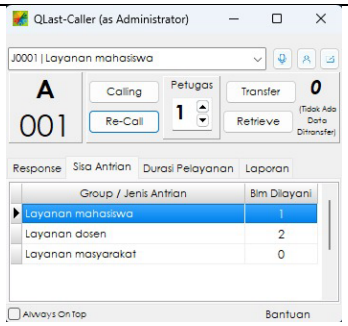
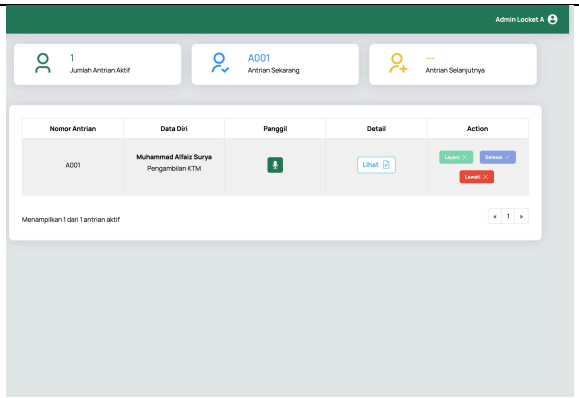
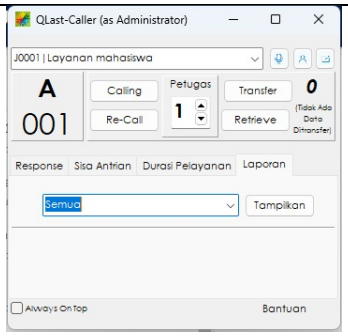
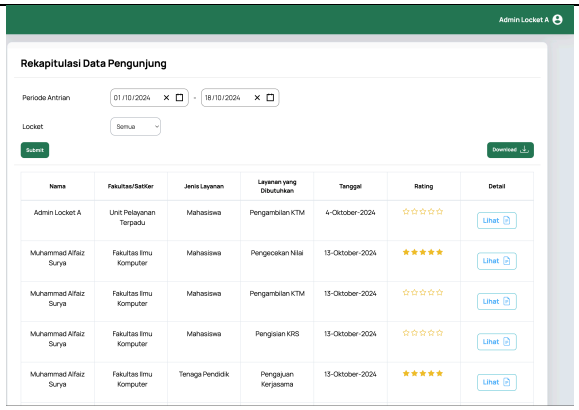
Gambar 3.33 Menampilkan halaman *FAQ* yang dimulai dari tampilan daftar jenis *FAQ* yang dapat dipilih oleh pengguna. Apabila pengguna mengklik salah satu jenisnya, maka akan ditampilkan halaman yang berisikan daftar *FAQ* yang ada pada jenis tersebut. Selanjutnya, pengguna dapat menampilkan detail jawaban dari pertanyaan yang ingin diketahui jawabannya dengan mengklik *item* pertanyaan tersebut.

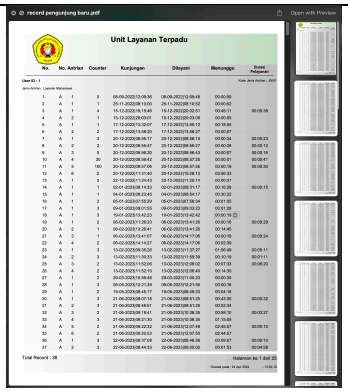
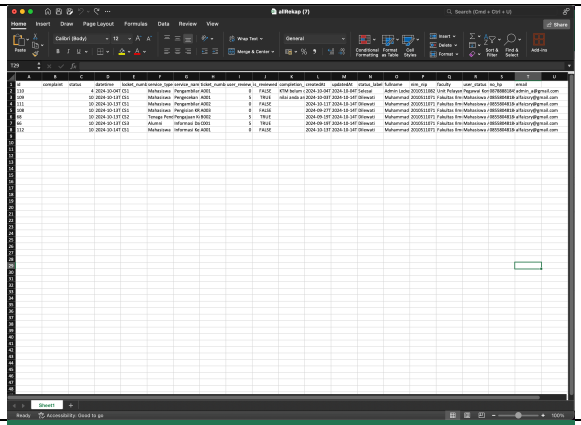
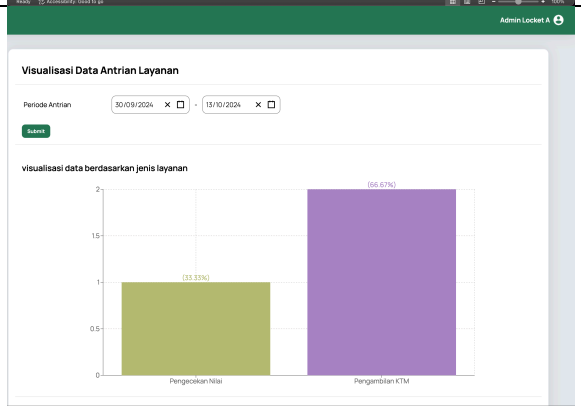
3.5.3. Perbandingan Sistem Lama dengan Hasil Implementasi Sistem Baru

Tabel 3. 3 Perbandingan Gambar Sistem Lama dengan Sistem Baru

Item Fungsi	Sistem Lama	Sistem Baru
Pendaftaran Antrian	  	 

Tiket Antrian		
Monitoring Antrian		

		
Pengelolaan Antrian		
Rekapitulasi Antrian		

		
Visualisasi Data Antrian	---	

Tabel 3. 4 Hasil Perbandingan Sistem Lama dengan Sistem Baru

***Item* Fungsi**	**Hasil Perbandingan**
Pendaftaran Antrian	1. Pada sistem lama, proses pendaftaran antrian menggunakan 2 sistem dengan platform yang berbeda yaitu dengan mesin pendaftaran antrian untuk memilih jenis layanan, dan *google form* untuk mengisi data diri pengunjung dan layanan yang dibutuhkan. Sedangkan pada sistem baru, pendaftaran antrian menggunakan 1 sistem saja pada *platform* aplikasi *mobile.* 2. Pada sistem lama, pengunjung perlu mengisi data diri setiap kali mendaftar antrian pelayanan. Sedangkan pada sistem baru, pengunjung hanya perlu mengisi data diri sekali pada saat proses pendaftaran akun aplikasi *mobile*
Tiket Antrian	Pada sistem lama, tiket antrian berbentuk kertas dan dicetak pada mesin antrian ULT UPNVJ. Sedangkan pada sistem baru, tiket antrian berbentuk digital yang tersimpan dan dapat dilihat pada aplikasi *mobile.*
Monitoring Antrian	1. Pada sistem lama, informasi antrian terkini dapat dilihat pada layar *Monitor* di ruangan ULT UPNVJ. Sedangkan pada sistem baru,

	informasi terkini antrian dapat dilihat pada layar perangkat masing masing pengguna aplikasi *mobile.* 2. Pada sistem lama, nomor antrian terkini dipanggil oleh admin menggunakan speaker di ruangan ULT UPNVJ. Sedangkan pada sistem baru, nomor antrian terkini dipanggil oleh admin menggunakan speaker di ruangan ULT UPNVJ dan dikirimkan pemberitahuan melalui *push* notifikasi ke akun pengguna dengan nomor antrian tersebut.
Pengelolaan Antrian	Pada sistem lama, pengelolaan antrian . Sedangkan pada sistem baru, pengelolaan antrian .
Rekapitulasi Antrian	1. Pada sistem lama, rekapitulasi antrian hanya berisikan data informasi antrian. Sedangkan pada sistem baru, rekapitulasi antrian berisikan data informasi antrian beserta data diri pengunjung, layanan yang dipilih, keluhan yang ditanyakan, nilai pelayanan dan catatan akhir pelayanan. 2. Pada sistem lama, data rekapitulasi antrian dapat diunduh dalam bentuk file *PDF* yang tidak dapat diolah datanya. Sedangkan pada sistem baru, data rekapitulasi dapat diunduh dalam bentuk file *Excel* yang dapat diolah kembali datanya.
Visualisasi Data Antrian	Pada sistem lama, belum terdapat fitur visualisasi data antrian. Sedangkan pada sistem baru, terdapat fitur visualisasi data antrian.

3.5.5. Kesimpulan

Kesimpulan yang penulis dapatkan dari hasil pengembangan proyek ini adalah sebagai berikut:

a. Aplikasi *mobile* untuk pendaftaran dan *monitoring* antrian di ULT UPN "Veteran" Jakarta berhasil dibangun menggunakan teknologi *Flutter* dan *WebSocket*. Dengan adanya aplikasi ini, pengunjung dapat mendaftar dan memantau status antrian langsung dari perangkat *smartphone* mereka, sehingga meningkatkan kenyamanan dan pengalaman pengguna dari segi fleksibilitas lokasi pendaftaran, kepraktisan proses pendaftaran, serta kemudahan akses informasi status antrian. Implementasi *WebSocket* dalam aplikasi ini memberikan manfaat berupa komunikasi data yang lebih efisien

dan dengan *bandwith internet* yang lebih minim karena data dapat dikirim dan diterima secara langsung (*real-time*) tanpa diperlukannya *refresh* data secara manual, sehingga pengunjung selalu mendapatkan informasi terbaru tentang status antrian mereka.

b. Aplikasi *website* untuk manajemen sistem antrian untuk staf admin ULT juga berhasil dikembangkan untuk dapat mengelola dan merekapitulasi data antrian pengunjung. Dengan adanya aplikasi ini, staf admin ULT dengan mudah dapat menganalisis dan mengevaluasi performa pelayanan ULT secara keseluruhan dengan rekapitulasi data antrian yang lebih lengkap pendataannya serta dengan adanya visualisasi grafik data antrian pengunjung. Manfaat implementasi *WebSocket* dalam aplikasi ini adalah memfasilitasi pembaruan data secara langsung tanpa perlu *reload* halaman, sehingga staf admin dapat melihat perubahan status antrian secara *real-time*, mempermudah pengelolaan dan pengawasan antrian dengan mengurangi waktu yang dibutuhkan untuk memantau dan memperbarui data antrian secara manual.

3.5.6. Saran

Agar sistem yang dibangun dapat lebih optimal dan berjalan dengan lancar, maka terdapat beberapa saran yang dapat dijadikan bahan pertimbangan untuk pengembangan sistem selanjutnya yang diuraikan sebagai berikut:

a. Diharapkan pengembang selanjutnya dapat mengembangkan *website* untuk pendaftaran antrian ULT yang diperuntukkan kepada pengunjung yang berhalangan untuk mengunduh aplikasi *mobile.*
b. Diharapkan pengembang selanjutnya dapat mengembangkan fitur chatbot pada aplikasi untuk meningkatkan interaksi dan memberikan layanan informasi secara otomatis kepada pengunjung. Fitur chatbot ini dapat membantu menjawab pertanyaan umum seputar prosedur pendaftaran, jam operasional, persyaratan dokumen, serta informasi lainnya yang sering ditanyakan oleh pengunjung.