Progress Report

IT2080 – Information Technology Project. Year 02 Semester 02

Title : Yakadabadu.lk – Scrap Collection & Recycling Management System

Batch : 4.1

Group Number : 81

Group Details:

IT Number	Name	E-mail	Contact Number
IT 23202122	P P D R Pathirage	IT23202122@my.sliit.lk	077 051 5121
IT 23140752	Gunawardena A A	IT23140752@my.sliit.lk	071 279 2376
IT 23216778	S A R U Amarasinghe	IT23216778@my.sliit.lk	071 799 5000
IT 23320550	K T A Kularathne	IT23320550@my.slit.lk	071 144 3347
IT 23144408	Fernando W A A T	IT23144408@my.slit.lk	076 206 2013

Table of Contents

1.1 Introduction

Yakadabadu.lk - A Scrap Collection and Recycling System - strives to improve productivity in also bilking, scrap collections, recycling, and upscale financial transactions. Manipulation of the system is not complex offering simple interfaces which can be used by customers, workers, and administrators to schedule pickups, view transactions, and streamline logistical activities.

Summary of the objectives of the system include the following:

- Providing secure user login and roll-based access control.
- Developing an automated system for scheduling scrap pickups prims on optimal route.
- Improving financial supervision such as salaries, transaction, and reporting on payments.
- Providing real time feeds to monitor pick up location and status by the driver.
- Developing a system that allows users to perform financial analysis and provide interfaces for Profit & Loss Statements, Brief Financial Reports and Balance Sheets.

To this point, the drivers and routes management, financials and salary allocations has been automated. System also includes real-time monitoring of the drivers and automatically updating of ledgers and journals. All of these improvements will make it possible to achieve better accuracy in GPS tracking, and advanced financial dashboards and enhanced UI/UX.

This progress report outlines the details of the completed and outstanding tasks, individual contributions, system design and the next convergence step in completing the project.

1.2 User Stories

1. **US 001**: As a user, I want to register with my email and phone number.
2. **US 002**: As a user, I want to log in securely using a password.
3. **US 003**: As a user, I want to reset my password via email verification.
4. **US 004**: As a user, I want to update my profile information.
5. **US 005**: As an admin, I want to manage and verify user accounts.
6. **US 006**: As a user, I want to request a scrap pickup by selecting my location.
7. **US 007**: As a user, I want to set a preferred date for pickup.
8. **US 008**: As a user, I want to track my pickup request status.
9. **US 009**: As a driver, I want to accept or reject pickup requests.
10. **US 010**: As an admin, I want to monitor all scheduled pickups.
11. **US 011**: As a driver, I want to receive a list of assigned pickups.
12. **US 012**: As a driver, I want to see an optimized route for multiple pickups.
13. **US 013**: As a driver, I want to update the pickup status as "Completed."
14. **US 014**: As a user, I want to see the estimated arrival time of the pickup driver.
15. **US 015**: As an admin, I want to monitor driver activity and performance.
16. **US 016**: As an admin, I want to generate reports on collected scrap materials.
17. **US 017**: As an admin, I want to track the total weight of recycled materials.
18. **US 018**: As a user, I want to see an overview of my scrap contributions.
19. **US 019**: As an admin, I want to categorize collected materials by type.
20. **US 020**: As a business partner, I want to access reports on available recycled materials.
21. **US 021**: As a user, I want to browse listed items for second-hand materials.
22. **US 022**: As a user, I want to check the current price per kilogram of scrap materials.
23. **US 023**: As a buyer, I want to chat with sellers for price negotiation.
24. US **024**: As a seller, I want to receive notifications for item interest.

25. **US 025**: As a driver, I want to view my salary details.

26. **US 026**: As an owner, I want to calculate staff monthly salaries.

1.3 Completed Features

✓ User Authentication & Management

- User registration (email & phone number) **– US 001**
- Secure login with password **– US 002**
- Password reset via email verification **– US 003**
- Admin-managed user verification **– US 005**

✓ Pickup Management

- Scrap pickup request with location selection **- US 006**
- Set preferred pickup date **- US 007**
- Track pickup request status **- US 008**

✓ Driver & Route Management

- Driver and route management system **- US 009, US010**
- Driver location integration & real-time tracking **- US 014**
- Route display & optimization using Google Maps API **- US 012**
- Pickup confirmation/cancellation functionality **- US 013**
- Admin panel for managing drivers, routes, and pickup requests **- US 015, - US 016**

✓ Financial Management

- Salary calculation system (basic salary, EPF, ETF, no-pay leaves, bonuses) **- US 024**
- Categorized transactions (ledger, bank book, petty cash) **- US 022**
- Automated updates for respective ledgers **- US 022**
- report generation (Profit & Loss Statement, Balance Sheet) **- US 020, - US 026**

1.4 In Progress

✓ Pickup Management

- Schedule pickup form improvements – Amarasinghe (Sprint 3)
- Update pickup details functionality – Amarasinghe (Sprint 3)
- Filterable admin reports – Kulerathne (Sprint 3)

✓ UI/UX

- Home page redesign – Amarasinghe (Sprint 3)
- Location picking system enhancements – Fernando (Sprint 3)
- About Us page development – Amarasinghe (Sprint 3)

✓ Reuse/Recycle

- Form reuse system – Pathirage (Sprint 3)
- CRUD operations standardization – Kularathne (Sprint 3)
- Notification integration – Fernando (Sprint 3)

1.5 To-Do List

✓ Critical

- Driver Management:
    - Real-time location updates (WebSocket/Firebase) – API integration, Algorithm implementation
    - Advanced route optimization (A*/Maps API) – API integration, Algorithm implementation

✓ Major

- Finance Management:
    - HR module integration (Attendance/OT/Leave) – Data pipeline development
    - Chart/Graph visualizations – Chart.js integration

✓ Normal

- Notification System:
    - Reuse/recycle notifications – Firebase Cloud Messaging, Email/SMS integration
    - Pickup confirmation alerts – Firebase Cloud Messaging, Email/SMS integration

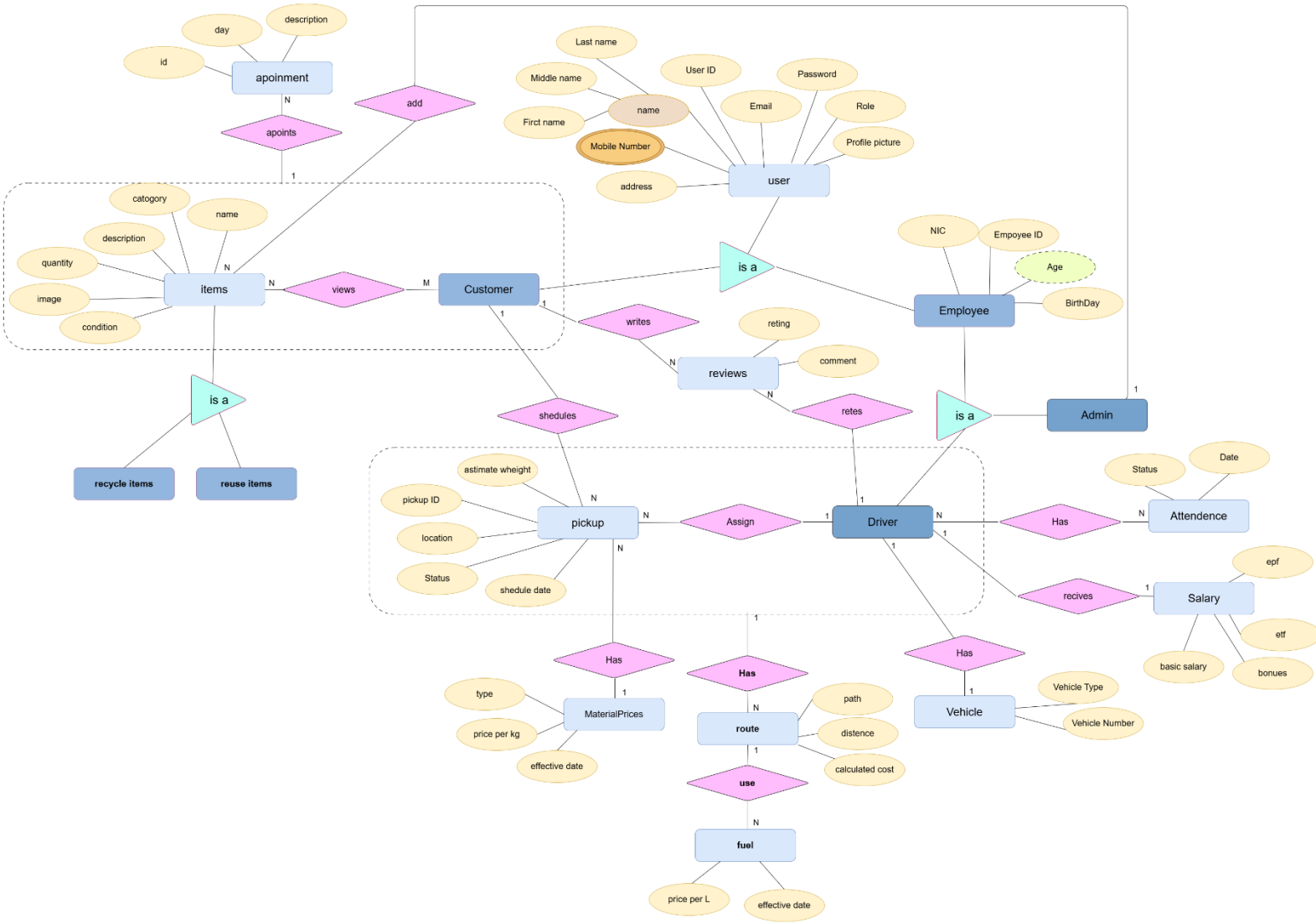
✓ Minor

- UI Enhancements:
  - Map view improvements – React Map Libraries, User testing
  - Dashboard usability upgrades – User testing

1.6 Repository Link

- angeli-sliit/Yakadabadu.lk
- angeli-sliit/driver-route-management

2.1 ER-Diagram



Link

2.2 Normalized schema – (3NF)

2.2.1. Users Schema

```
const userSchema = new mongoose.Schema({
    firstName: { type: String, required: true },
    middleName: { type: String },
    lastName: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
     role: { type: String, enum: ['customer', 'admin', 'employee', 'driver'], required: true },
    profilePicture: { type: String },
    mobileNumber: { type: String, required: true },
    address: { type: String },
    isDeleted: { type: Boolean, default: false },
  },
  { timestamps: true }
);
const User = mongoose.model('User', userSchema);
```

2.2.2. Employees Schema

```
const employeeSchema = new mongoose.Schema({
     userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', unique: true, required: true },
    NIC: { type: String, required: true, unique: true },
    birthday: { type: Date, required: true },
    isDeleted: { type: Boolean, default: false },
  },
  { timestamps: true }
);
const Employee = mongoose.model('Employee', employeeSchema);
```

2.2.3. Admins Schema

```
const adminSchema = new mongoose.Schema({
    employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', unique: true, required: true },
  },
  { timestamps: true }
);
const Admin = mongoose.model('Admin', adminSchema);
```

2.2.4. Customer Schema

```
const customerSchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', unique: true, required: true },
  },
  { timestamps: true }
);
const Customer = mongoose.model('Customer', customerSchema);
```

2.2.5. Reviews Schema

```
const reviewSchema = new mongoose.Schema({
    customerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Customer', required: true },
    employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required: true },
    rating: { type: Number, required: true, min: 1, max: 5 },
    comment: { type: String },
  },
  { timestamps: true }
);
const Review = mongoose.model('Review', reviewSchema);
```

2.2.6. Items Schema

```
const itemSchema = new mongoose.Schema({
    name: { type: String, required: true },
    category: { type: String, required: true },
    description: { type: String },
    quantity: { type: Number, required: true },
    image: { type: String },
    condition: { type: String, required: true },
  },
  { timestamps: true }
);
const Item = mongoose.model('Item', itemSchema);
```

2.2.7. Pickups Schema

```
const pickupSchema = new mongoose.Schema({
    customerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Customer', required:
true },
    estimatedWeight: { type: Number, required: true },
    location: { type: String, required: true },
     status: { type: String, required: true, enum: ['Scheduled', 'Completed',
'Pending'] },
    scheduleDate: { type: Date, required: true },
  },
  { timestamps: true }
);
const Pickup = mongoose.model('Pickup', pickupSchema);
```

2.2 8. Pickup Assignments Schema

```
const pickupAssignmentSchema = new mongoose.Schema({
    pickupId: { type: mongoose.Schema.Types.ObjectId, ref: 'Pickup', required: true },
    driverId: { type: mongoose.Schema.Types.ObjectId, ref: 'Driver', required: true },
  },
  { timestamps: true }
);
const PickupAssignment = mongoose.model('PickupAssignment', pickupAssignmentSchema);
```

2.2.9. Driver Schema

```
const driverSchema = new mongoose.Schema({
    employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', unique:
true, required: true },
  },
  { timestamps: true }
);
const Driver = mongoose.model('Driver', driverSchema);
```

2.2.10. Material Prices Schema

```
const materialPricesSchema = new mongoose.Schema({
    type: { type: String, required: true },
    pricePerKg: { type: Number, required: true },
    effectiveDate: { type: Date, required: true },
  },
  { timestamps: true }
);
const MaterialPrice = mongoose.model('MaterialPrice', materialPricesSchema);
```

2.2.11. Salary Schema

```
const salarySchema = new mongoose.Schema(
 {
    employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required:
true },
    basicSalary: { type: Number, required: true },
    epf: { type: Number, required: true },
    etf: { type: Number, required: true },
    bonus: { type: Number },
  },
  { timestamps: true }
);
const Salary = mongoose.model('Salary', salarySchema);
```

2.2.12. Vehicles Schema

```
const vehicleSchema = new mongoose.Schema({
  vehicleType: { type: String, required: true },
  vehicleNumber: { type: String, required: true, unique: true, index: true },
  capacity: { type: Number, required: true }, // in kg or tons
},
{ timestamps: true }
);
const Vehicle = mongoose.model('Vehicle', vehicleSchema);
```

2.2.13. Routes Schema

```
const routeSchema = new mongoose.Schema({
  path: { type: String, required: true },
  distance: { type: Number, required: true }, // in kilometers
  calculatedCost: { type: Number, required: true },
},
{ timestamps: true }
);
const Route = mongoose.model('Route', routeSchema);
```

2.2.14. Fuel Schema

```
const fuelSchema = new mongoose.Schema({
  pricePerLiter: { type: Number, required: true },
  effectiveDate: { type: Date, required: true },
},
{ timestamps: true }
);
const Fuel = mongoose.model('Fuel', fuelSchema);
```

2.2.15. Attendance Schema

```
const attendanceSchema = new mongoose.Schema({
  employeeId: { type: mongoose.Schema.Types.ObjectId, ref: 'Employee', required:
true },
  date: { type: Date, required: true },
  status: { type: String, required: true, enum: ['Present', 'Absent'] },
},
{ timestamps: true }
);
const Attendance = mongoose.model('Attendance', attendanceSchema);
```

2.2.16. Appointments Schema

```
const appointmentSchema = new mongoose.Schema({
  customerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Customer', required:
true },
  description: { type: String, required: true },
  items: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Item' }],
   status: { type: String, required: true, enum: ['Pending', 'Confirmed',
'Completed', 'Cancelled'] },
  scheduledDate: { type: Date, required: true },
},
{ timestamps: true }
);
const Appointment = mongoose.model('Appointment', appointmentSchema);
```

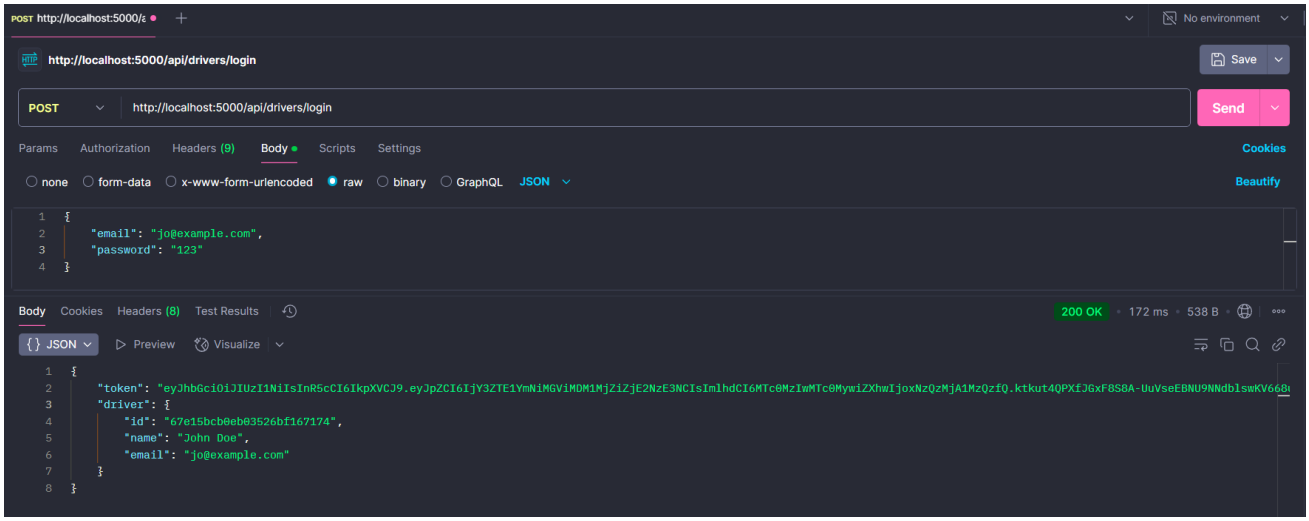
2.2.17. Transactions Schema

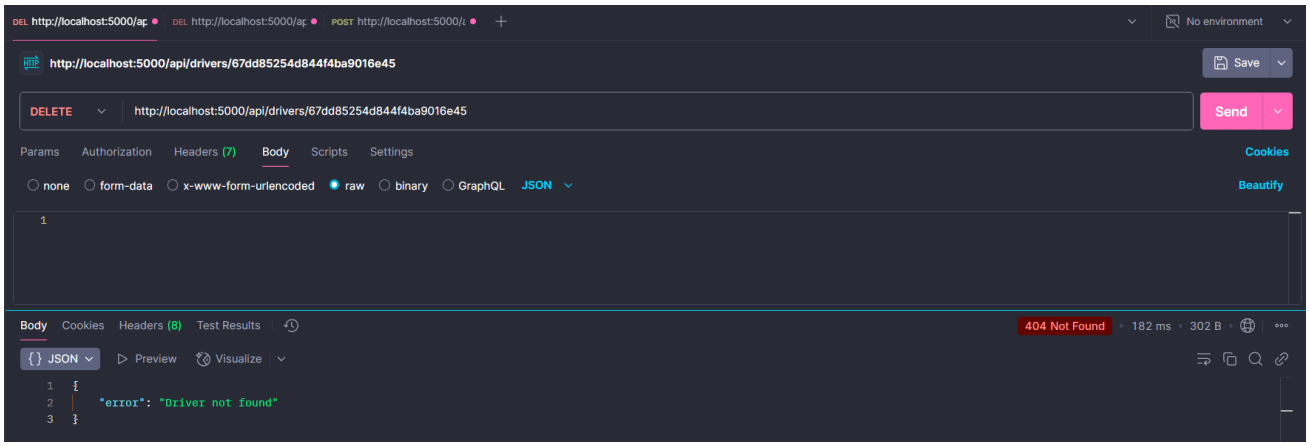
```
const transactionSchema = new mongoose.Schema({
  customerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Customer', required: true },
  amount: { type: Number, required: true },
  transactionType: { type: String, required: true, enum: ['Credit', 'Debit'] },
  paymentMethod: { type: String, required: true, enum: ['Cash', 'Card', 'Online Transfer'] },
  status: { type: String, required: true, enum: ['Pending', 'Completed', 'Failed'] },
},
{ timestamps: true }
);
const Transaction = mongoose.model('Transaction', transactionSchema);
```

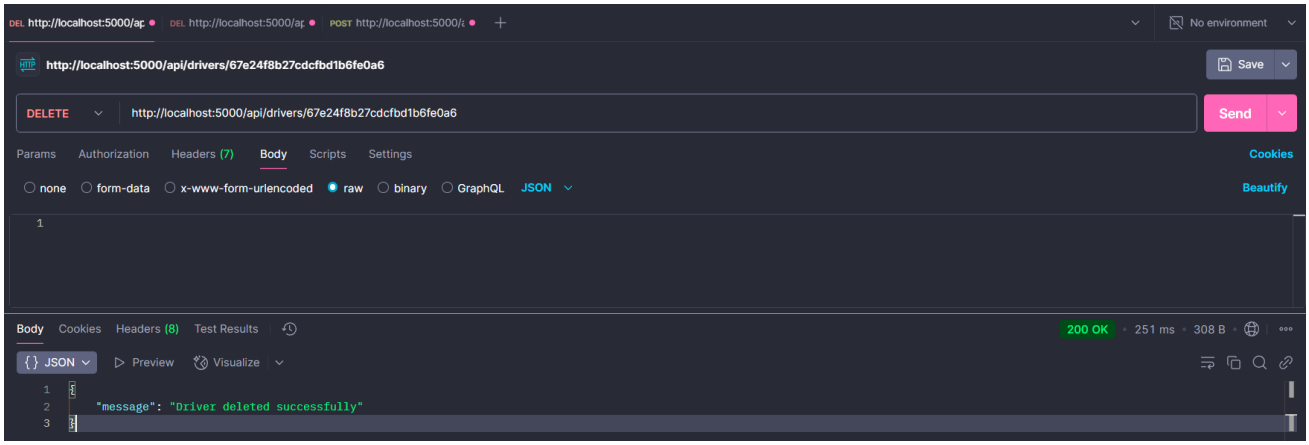
2.2.18. Waste Materials Schema

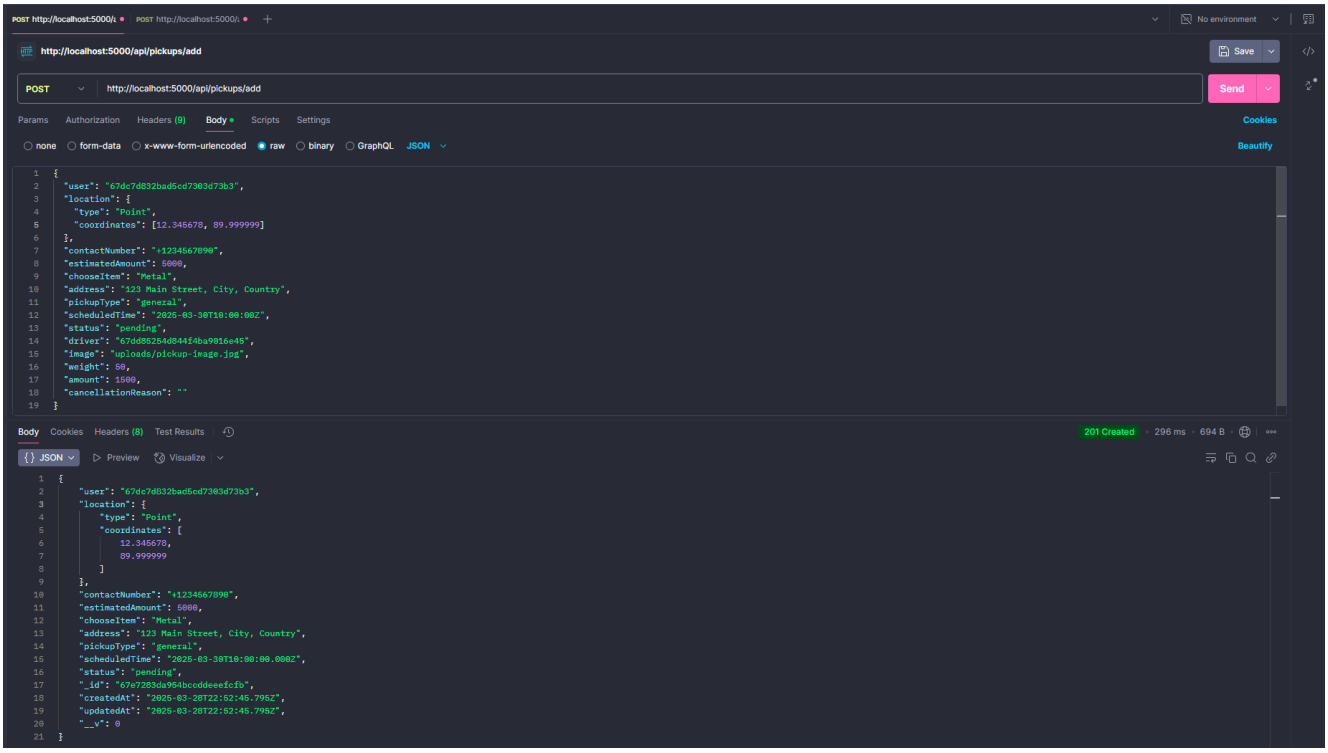
```
const wasteMaterialSchema = new mongoose.Schema({
  type: { type: String, required: true },
  category: { type: String, required: true },
  description: { type: String },
  recyclable: { type: Boolean, required: true },
},
{ timestamps: true }
);
const WasteMaterial = mongoose.model('WasteMaterial', wasteMaterialSchema);
```

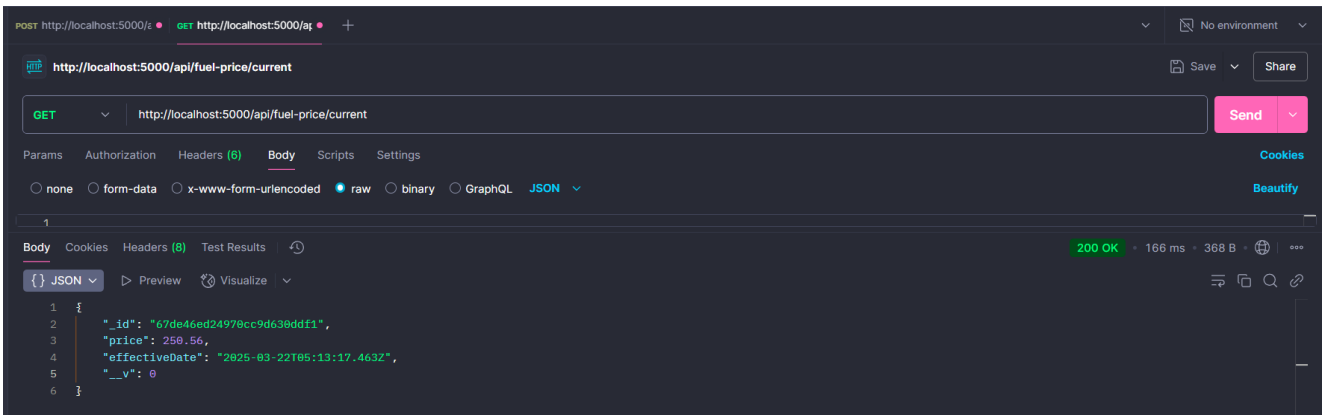
2.3 Test Cases





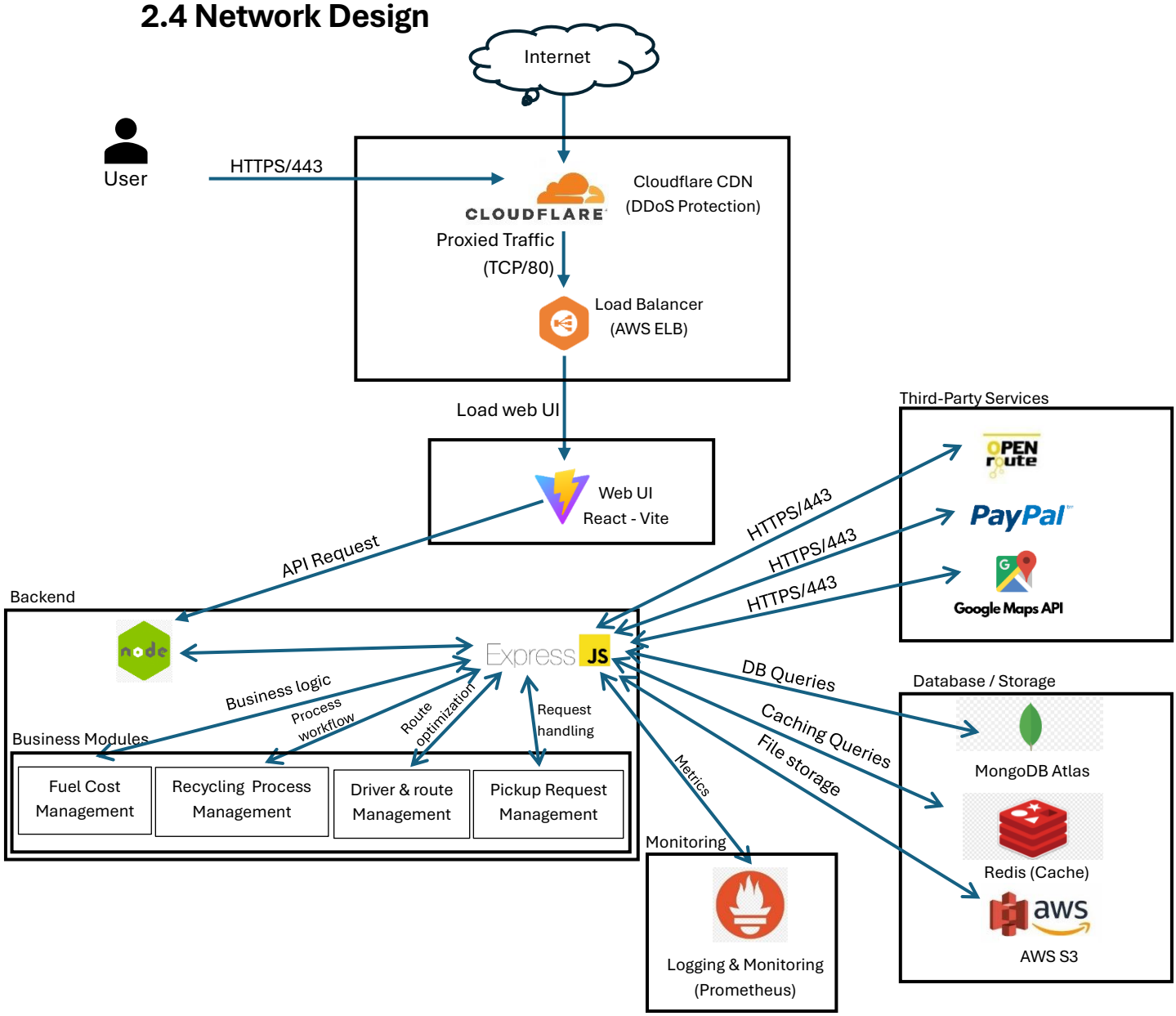



http://localhost:5000/api/pickups/add
POST
http://localhost:5000/api/pickups/add
Send
Save
Params Authorization Headers (9) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"user": "67dc7d832bad5cd7383d73b3",
"location": {
"type": "Point",
"coordinates": [12.345678, 89.999999]
},
"contactNumber": "+1234567890",
"estimatedAmount": 5000,
"chooseItem": "Metal",
"address": "123 Main Street, City, Country",
"pickupType": "general",
"scheduledTime": "2025-03-30T10:00:00Z",
"status": "pending",
"driver": "67dd86254d844f4ba9816e45",
"image": "uploads/pickup-image.jpg",
"weight": 50,
"amount": 1500,
"cancellationReason": ""
}
Body Cookies Headers (8) Test Results
201 Created 296 ms 694 B
JSON Preview Visualize
{
"user": "67dc7d832bad5cd7383d73b3",
"location": {
"type": "Point",
"coordinates": [
12.345678,
89.999999
]
},
"contactNumber": "+1234567890",
"estimatedAmount": 5000,
"chooseItem": "Metal",
"address": "123 Main Street, City, Country",
"pickupType": "general",
"scheduledTime": "2025-03-30T10:00:00.000Z",
"status": "pending",
"_id": "67e7283da954bcoddeeefcfb",
"createdAt": "2025-03-28T22:52:45.795Z",
"updatedAt": "2025-03-28T22:52:45.795Z",
"__v": 0
}

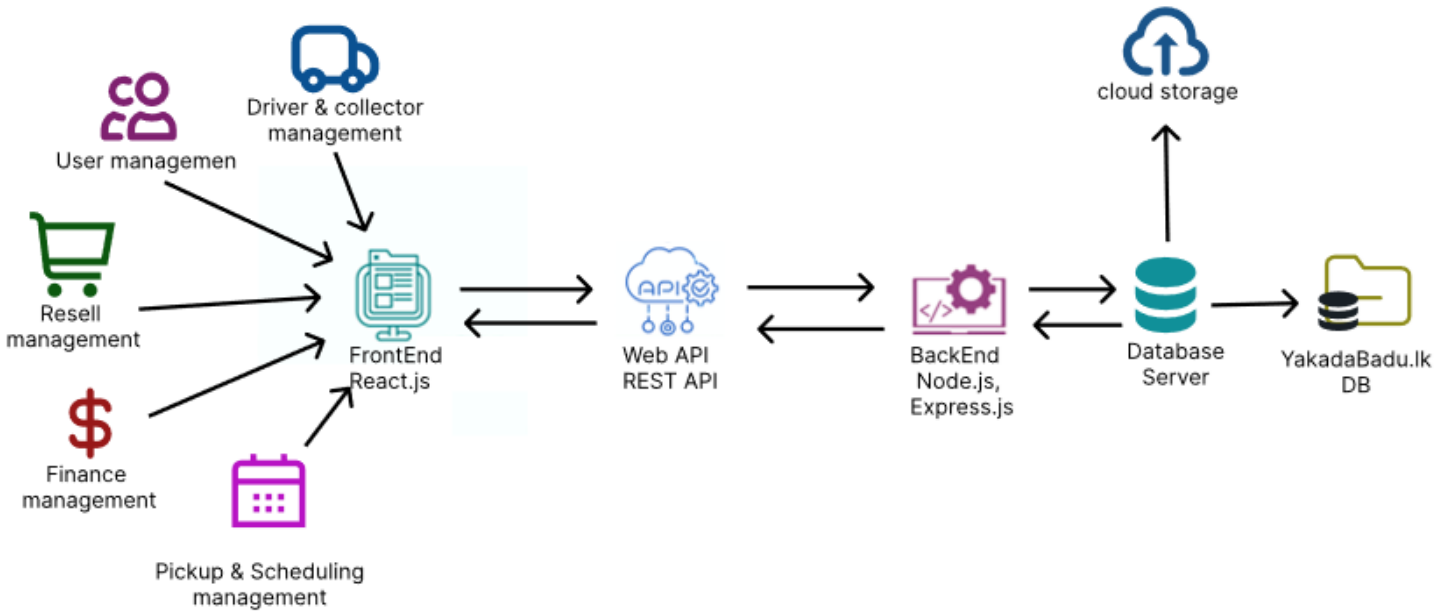

http://localhost:5000/api/fuel-price/current
GET
http://localhost:5000/api/fuel-price/current
Send
Save
Share
Params Authorization Headers (6) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (8) Test Results
200 OK 166 ms 368 B
JSON Preview Visualize
{
"_id": "67de46ed24970cc9d630ddf1",
"price": 250.56,
"effectiveDate": "2025-03-22T05:13:17.463Z",
"__v": 0
}

TC-ID	Description	Input Data	Expected Result	Error Type	Automated
TC-101	Admin approves pending user account	Admin ID, User ID	User status = "Verified"	200 OK	Yes
TC-102	Role-based dashboard access	User Role = "Driver"	Driver-specific UI displayed	200 OK	Yes
TC-103	Validate login functionality	Username, Password	Successful login or error message	200 OK / 401 Unauthorized	Yes
TC-104	Validate password reset	Registered email address	Password reset email sent	200 OK	Yes
TC-105	User login validation (incorrect password)	Username, Incorrect Password	Error message: "Incorrect Password"	401 Unauthorized	Yes
TC-106	User registration	User details (Name, Email, etc.)	New user created	201 Created	Yes
TC-107	Admin deletes user account	User ID	User status = "Deleted"	200 OK	Yes
TC-201	Conflict: Same pickup slot	Duplicate time/location	Error: "Slot unavailable"	409 Conflict	Yes
TC-201	Validate pickup cancellation	Pickup ID, Reason	Pickup status = "Cancelled"	200 OK	Yes
TC-202	Pickup request cancellation by customer	Pickup ID, Reason	Pickup request cancelled	200 OK	Yes
TC-203	Pickup status error handling	Invalid Pickup ID	Error: "Pickup not found"	404 Not Found	Yes
TC-301	Route optimization (5+ pickups)	Colombo city coordinates	Optimal route < 15km	200 OK	Yes
TC-302	Driver availability toggle	Status = "Offline"	Excluded from auto-assign	200 OK	Yes
TC-303	Validate driver route assignment	Driver ID, Route ID	Route assigned to driver successfully	200 OK	Yes
TC-304	Validate fuel consumption rate	Vehicle Type = Van, Distance = 50 km	Fuel consumption = x liters	200 OK	Yes
TC-305	Validate vehicle assignment to driver	Driver ID, Vehicle ID	Vehicle assigned to driver successfully	200 OK	Yes
TC-401	EPF/ETF auto-deduction	Basic salary = Rs. 50,000	EPF = Rs. 8,000, ETF = Rs. 3,000	200 OK	Yes
TC-501	Rate limiting (API protection)	100+ requests/sec	Error 429: Too Many Requests	429 Too Many Requests	Yes
TC-502	User authentication (login/logout)	Username,Password	Login successful or error message	200 OK / 401 Unauthorized	Yes
TC-503	SQL Injection protection	Input: "OR 1=1"	Error: "Invalid Input"	400 Bad Request	Yes
TC-505	Account lock after multiple failed logins	5 failed login attempts	Account locked in X minutes	423 Locked	Yes

2.4 Network Design



2.5 High-level system design diagram



2.6 Innovative Parts of the Project

Dynamic Real-Time Driver Allocation:

The system automatically finds the nearest available driver based on real-time location tracking. It optimizes pickup efficiency and minimizes waiting time, providing a seamless experience for users.

Smart Fuel Price-Based Cost Estimation:

The platform dynamically calculates pickup costs based on real-time fuel prices, ensuring fair and transparent pricing for both users and drivers. This feature enhances financial accuracy and reliability.

Automated Route Optimization:

Integrated with Google Maps API, the system provides the most efficient route for drivers, reducing fuel consumption and optimizing time management. This improves operational efficiency and reduces environmental impact.

Eco-Impact Report Generation:

Users receive an automated report summarizing their contributions to environmental sustainability, such as CO_2 reduction and the amount of recycled material. This feature encourages eco-friendly behaviours.

Digital Invoice & PDF Generation:

Upon pickup completion, the system generates downloadable PDF invoices for users and businesses. This streamlines record-keeping and enhances professionalism for commercial users.

Automated Pickup Scheduling & Notifications:

Users can pre-schedule scrap pickups, and the system automatically sends reminders via email/SMS to ensure timely collection. This feature improves customer convenience and driver efficiency.

2.7 Commercialization

Market Research:

A detailed analysis of the scrap collection and recycling industry identifies the key pain points of individuals, businesses, and recycling centres. Understanding market demand ensures that YakadaBadu.lk effectively addresses user needs and maximizes economic potential.

Competitive Analysis:

Evaluating existing scrap collection platforms and informal collection networks helps identify opportunities for differentiation. Key factors include pricing models, user experience, technology integration, and service efficiency. The goal is to provide a superior, technology-driven alternative.

Customer Support & Feedback System:

- A dedicated support team helps resolve user issues efficiently.
- A feedback mechanism allows users to rate drivers and services, ensuring continuous improvement in platform quality.

AI-powered chatbots can handle common queries, improving customer experience and engagement.

Legal Compliance & Sustainability Standards:

- Ensuring compliance with environmental regulations regarding waste collection and disposal.
- Collaborating with government agencies and eco-certification bodies to validate sustainable operations.
- Implementing strict data protection policies to secure user information and transactions.

3. individual contribution

3.1 – Fernando W A A T (IT23144408)

(Driver & Route management)

3.1.1 - Completion Level

- Current Status: 70% Complete

3.1.2 completed(Done)

- Drivers can sign up and provide details like name, vehicle type, license information, and availability.
- Admins can edit or deactivate driver profiles.
- Drivers can mark themselves as available or unavailable.
- The system updates the list of active drivers.
- Fetches the driver's real-time location when they log in or start a shift.
- Pickup List Generation:
- Based on requests, the system generates a list of pickups for the driver.
- Can be downloaded as a PDF.
- Displays the nearest driver’s location and routes them to the pickup point using Google Maps API.
- Drivers can view and follow optimized routes.

3.1.3 Work not completed (Doing)

Automated Assignment Improvements:

- Currently, nearest driver selection is being implemented.
- Need to refine logic for automatic driver assignment based on pickup priority and distance.

Multi-Stop Route Optimization:

- Currently, it only provides the route to the nearest pickup.
- Need to implement an optimized route for multiple pickups in one trip.

Fuel Cost Estimation:

- Add a system to estimate the fuel cost for a given route.

3.1.4 to be completed (To-Do)

Driver Rating & Performance Tracking:

- A feature to track completed pickups and calculate performance scores.

Dynamic Route Updates:

- Recalculate routes dynamically based on new pickups or cancellations.

Driver Leave Request & Approval System:

- Drivers can request leave through their dashboard, specifying the date and reason.
- The admin can approve or reject the leave request.
- Approved leave status will be reflected on the driver's dashboard.

3.1.5 queries used

Get All Drivers

```
const drivers = await Driver.find().select('-password');
```

Get Current Driver (Authenticated)

```
 const driver = await Driver.findById(req.user.id).select('-password');
```

Get Driver by ID

```
 const driver = await Driver.findById('driverId').select('-password');
```

Delete Driver by ID

```
const deletedDriver = await Driver.findByIdAndDelete(driverId);
```

Mark Driver Attendance

```
const driver = await Driver.findById(driverId);
driver.attendance.push({ date: new Date(), status: 'present' });
await driver.save();
```

Get Available Drivers

```
const availableDrivers = await Driver.find({
    status: 'available',
    'attendance.date': { $gte: new Date().setHours(0, 0, 0, 0) },
  });
```

Update Driver Availability

```
const driver = await Driver.findById(driverId);
const today = new Date();
today.setHours(0, 0, 0, 0);

const attendanceIndex = driver.attendance.findIndex(entry => new
Date(entry.date).setHours(0, 0, 0, 0) === today.getTime());
*



if (attendanceIndex >= 0) {
  driver.attendance[attendanceIndex].status = status;
} else {
  driver.attendance.push({ date: today, status });
}

driver.status = status;
await driver.save();
```

3.1.6 Algorythem

Truck Optimization Code (MERN Stack)

```
export function optimizePickups(pickups, drivers, fuelPrice) {
  const sortedPickups = [...pickups].sort((a, b) => b.estimatedAmount -
a.estimatedAmount);
  const trucks = [];
  const availableTrucks = drivers
    .filter(d => d.status === 'available')
    .map(d => ({
      driver: d,
      capacity: TRUCK_CAPACITIES[d.vehicleType],
      remaining: TRUCK_CAPACITIES[d.vehicleType],
      pickups: [],
```

```
route: []
    }));
  for (const pickup of sortedPickups) {
let assigned = false;
    for (const truck of availableTrucks) {
      if (truck.remaining >= pickup.estimatedAmount) {
        truck.pickups.push(pickup);
        truck.remaining -= pickup.estimatedAmount;
        assigned = true;
        break;
      }
    }
    if (!assigned) {
 console.warn(`Pickup ${pickup._id} (${pickup.estimatedAmount}kg) exceeds all truck
capacities`);
    }
  }
  return availableTrucks
    .filter(truck => truck.pickups.length > 0)
    .map(truck => {
      truck.pickups.sort((a, b) =>
        calculateDistance(COMPANY_LOCATION.latitude, COMPANY_LOCATION.longitude,
            a.location.coordinates[1], a.location.coordinates[0]) -

calculateDistance(COMPANY_LOCATION.latitude,
COMPANY_LOCATION.longitude,
 b.location.coordinates[1], b.location.coordinates[0])
      );
      let totalDistance = 0;
      let previousLocation = COMPANY_LOCATION;

      for (const pickup of truck.pickups) {
        const distance = calculateDistance(
          previousLocation.latitude,
          previousLocation.longitude,
          pickup.location.coordinates[1],
          pickup.location.coordinates[0]
        );
        totalDistance += distance;
```

```
 previousLocation = pickup.location;
      }
      totalDistance += calculateDistance(
  previousLocation.latitude,
        previousLocation.longitude,
        COMPANY_LOCATION.latitude,
        COMPANY_LOCATION.longitude
      );
      const fuelCost = totalDistance * FUEL_CONSUMPTION[truck.driver.vehicleType] *
fuelPrice;
      return {
        driverId: truck.driver._id,
        pickups: truck.pickups.map(p => p._id),
        totalWeight: TRUCK_CAPACITIES[truck.driver.vehicleType] - truck.remaining,
        fuelCost,
        route: truck.pickups.map(p => p.location.coordinates)
      };
   });
}
```

2. Distance Calculation (Haversine Formula)

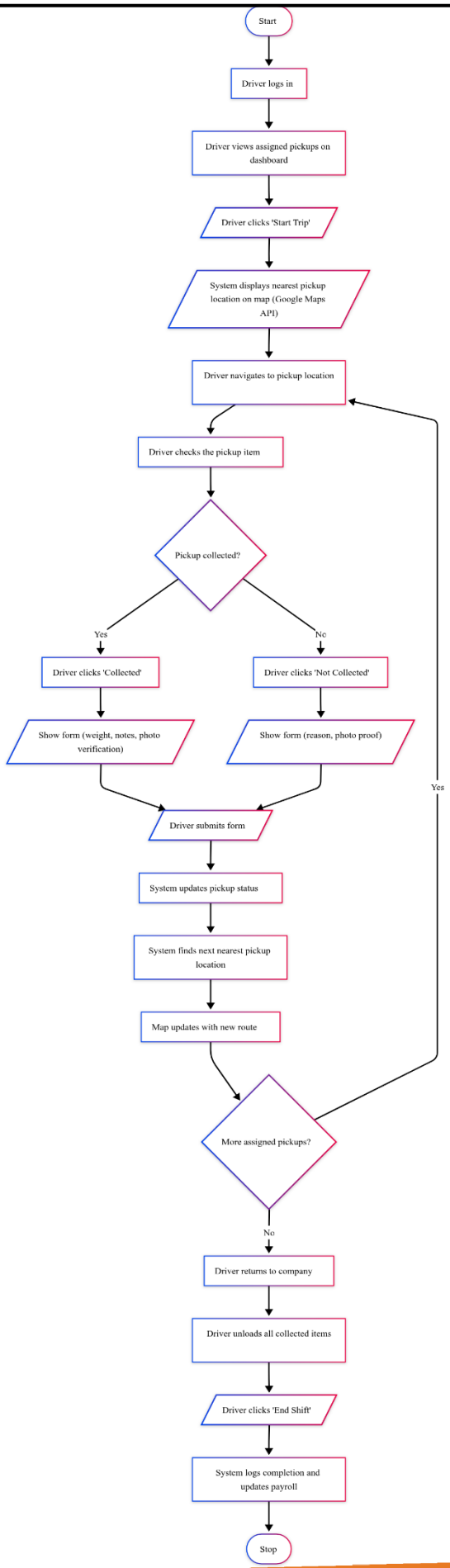
```
export function calculateDistance(lat1, lon1, lat2, lon2) {
    const R = 6371
    const dLat = (lat2 - lat1) * (Math.PI / 180);
    const dLon = (lon2 - lon1) * (Math.PI / 180);
    const a =
      Math.sin(dLat / 2) * Math.sin(dLat / 2) +
      Math.cos((lat1 * Math.PI) / 180) *
        Math.cos((lat2 * Math.PI) / 180) *
        Math.sin(dLon / 2) *
        Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
  }
```

3. Vehicle Route Optimization Using ORS API

```
import axios from 'axios';
const ORS_API_KEY = process.env.OPENROUTE_API_KEY;
const ORS_OPTIMIZATION_URL = 'https://api.openrouteservice.org/optimization';
export const optimizeVehicleRoutes = async (vehicles, jobs) => {
  try {
    const response = await axios.post(ORS_OPTIMIZATION_URL, {
      jobs: jobs.map(job => ({
        id: job.id,
        location: [job.lon, job.lat],
        service: job.service_time || 300,
       amount: [job.weight || 1],
        ...(job.time_window && { time_windows: [job.time_window] })
      })),
vehicles: vehicles.map(vehicle => ({
        id: vehicle.id,
        profile: 'driving-car',
        start: [vehicle.start_lon, vehicle.start_lat],
        end: [vehicle.end_lon, vehicle.end_lat],
        capacity: [vehicle.capacity],
        skills: vehicle.required_skills || [],
        time_window: vehicle.working_hours
      })),
      options: {
        g: true
      } }, {
      headers: {
        'Authorization': ORS_API_KEY,
        'Content-Type': 'application/json'
      } });
    return processOptimizationResponse(response.data);
  } catch (error) {
    throw new Error(`Optimization failed: ${error.response?.data?.error ||
error.message}`);}};-
const processOptimizationResponse = (data) => {
  return data.routes.map(route => ({
```

```
vehicleId: route.vehicle,
    steps: route.steps.map(step => ({
       type: step.type,
      location: step.location,
      arrival: step.arrival,
      duration: step.duration,
      ...(step.job && { jobId: step.job })
    })),
    geometry: route.geometry,
    totalDistance: route.distance,
    totalDuration: route.duration,
    totalWeight: route.amount[0]  }));
};
```

Flow chart


Start
Driver logs in
Driver views assigned pickups on dashboard
Driver clicks 'Start Trip'
System displays nearest pickup location on map (Google Maps API)
Driver navigates to pickup location
Driver checks the pickup item
Pickup collected?
Yes
No
Driver clicks 'Collected'
Driver clicks 'Not Collected'
Show form (weight, notes, photo verification)
Show form (reason, photo proof)
Driver submits form
System updates pickup status
System finds next nearest pickup location
Map updates with new route
More assigned pickups?
Yes
No
Driver returns to company
Driver unloads all collected items
Driver clicks 'End Shift'
System logs completion and updates payroll
Stop

3.2 – Gunawardena A A (IT23140752)

(Finance Management)

3.2.1 – Completion Level

- Current Status: 70% completed

3.2.2 – Completed

- The finance manager can login to finance management dashboard and navigate to finance features.
- Salary calculations implemented with basic salary, EPF, ETF, OT and bonuses.
- Allows manual modifications for deductions and bonuses.
- Implemented separate document to store calculation formulas to modify in the future if necessary.
- Ledger, Bank Book and Petty cash functionalities developed.
- All transactions update respective ledgers and books automatically.
- Ledger and Book balances calculate automatically.
- Search function to find ledgers easily.
- Profit & Loss and Balance Sheet report data calculated.
- Implemented PDF generation for reports.

3.2.3 – To be completed

- HR module integration to fetch employee attendance, OT and leave records for salary calculations.
- Add charts & graphs for income/expense analysis.
- Display salary details in driver view.
- Improve dashboard usability.

3.2.4 – Queries used

- Update salary formula by ID

```
const updatedFormula = await SalaryFormula.findByIdAndUpdate(
    req.params.id,
    { epfPercentage, etfPercentage, otRate, holidayPayRate, updatedAt: Date.now() },
    { new: true }
);
```

- Get salary details by employee ID

```
const salary = await EmployeeSalary.findOne({ employeeId: req.params.employeeId });
```

- Reverse transaction by ID

```
const transaction = await Transaction.findById(transactionId);
```

- Get all ledgers by category

```
const ledgers = await Ledger.find({ category });
```

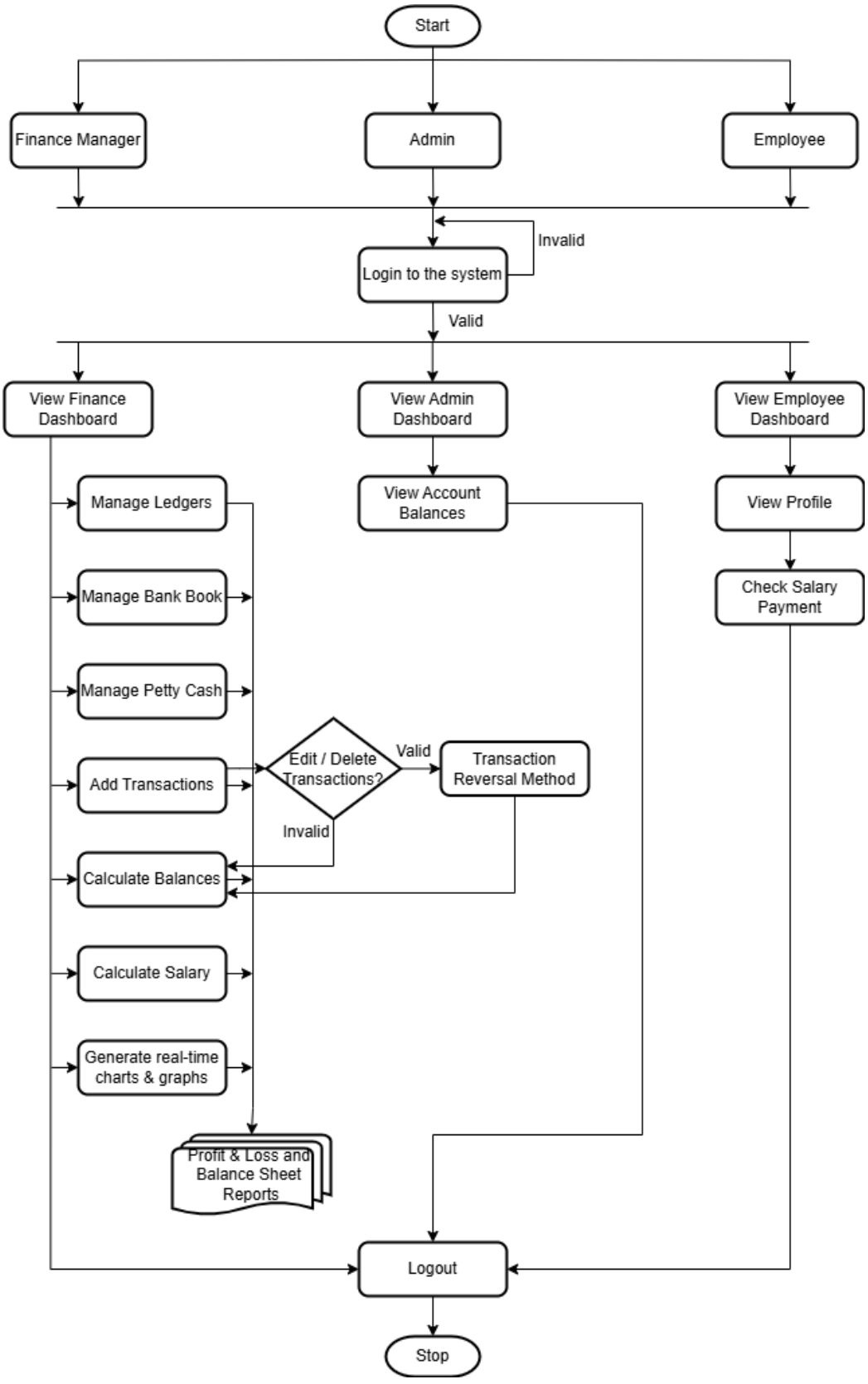
- Generate Profit & Loss Statement for month

```
const report = await ProfitLoss.findOne({ month });
if (!report) {
    return res.status(404).json({ message: "Profit & Loss report not found for this month." });
}

// Fetch all income and expense ledgers
const incomeLedgers = await Ledger.find({ category: "Income" });
const expenseLedgers = await Ledger.find({ category: "Expense" });
```

- Generate Balance Sheet for the month

```
const report = await BalanceSheet.findOne({ month });
if (!report) {
    return res.status(404).json({ message: "Balance Sheet not found for this month." });
}

// Fetch all assets, liabilities, and equity
const ledgerAssets = await Ledger.find({ category: "Asset" });
const bankBookAssets = await BankBook.find({ category: "Asset" });
const liabilities = await Ledger.find({ category: "Liability" });
const equity = await Ledger.find({ category: "Equity" });
```

3.2.5 – Flow Chart



3.3. Pathirage D R – (IT23202122)

(Reuse & Recycle item management)

3.3.1. Completion Level

Current Status: 70% Complete

3.3.2. Completed (Done)

- Recycle Form.
- Reuse Form.
- View Reuse report.
- Update and Deleted Reuse details.
- Update and Deleted Recycle details.
- Search option by item name on the Reuse page.
- Intergrated frontend with backend store data in mongo DB.
- Search option by item name on the Recycle page.
- Reuse Report Generate.

3.3.3. Work not Completed (Doing)

- Improving Reuse and Recycle user interface.
- Auto Generating Recycle Report.

3.3.4. To be Completed (To-Do)

1. Reuse item Admin table.
2. Recycle Filter option.
3. Recycle Items Table.
4. Notification Bar.

5. **3.1.5. Queries Used**
   - Reuse – Find by name

```
    const items = await ReuseItem.find({ itemName: { $regex: name, $options: "i" } });
    res.json(items);
  } catch (error) {
    res.status(500).json({ message: "Error searching items", error });
  }
});
```

- Create new Item

```
const newItem = new ReuseItem({
  itemName,
  category,
  condition,
  quantity,
  description,
  image,
  name,
  email,
  phone,
  reuseOption,
  pickupOption,
  availableUntil,
  communicationMethod,
});
```

- Update Item

```
export const updateItem = async (req, res) => {
  try {
    const { id } = req.params;
    const {
      itemName,
      category,
      condition,
      quantity,
      description,
      name,
      email,
      phone,
      reuseOption,
      pickupOption,
      availableUntil,
      communicationMethod,
    } = req.body;
```

- Delete Item

```
export const deleteItem = async (req, res) => {
  try {
    const { id } = req.params;
    const deletedItem = await ReuseItem.findByIdAndDelete(id);
```

- Search by Item name

```
export const getItems = async (req, res) => {
  try {
    const items = await ReuseItem.find();
    res.json(items);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
```

3.1.6. Algorithms

- Reuse MERN Stack

```
export const createItem = async (req, res) => {
  try {
    const newItem = new ReuseItem(req.body);
    await newItem.save();
    res.status(201).json(newItem);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

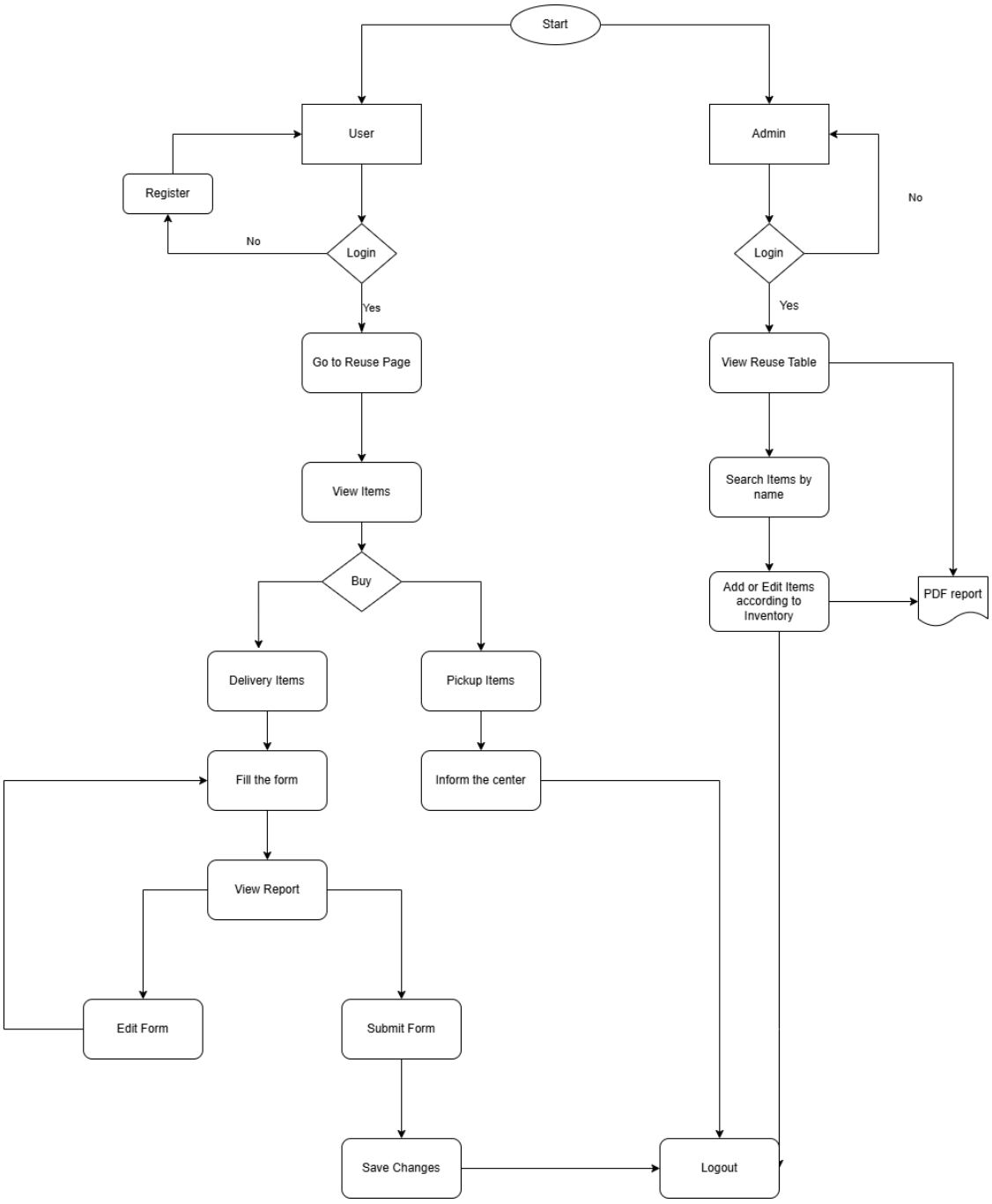
- Delete item List

```
const deleteItem = async (id) => {
  try {
    await fetch(`http://localhost:5000/api/reuse/${id}`, { method: "DELETE" });
    setItems(items.filter((item) => item._id !== id));
  } catch (error) {
    console.error("Error deleting item:", error);
  }
};
```

```
export const getItems = async (req, res) => {
  try {
    const items = await ReuseItem.find();
    res.json(items);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

```
useEffect(() => {
  fetch("http://localhost:5000/api/reuse")
    .then((res) => res.json())
    .then((data) => setItems(data)) // Update state with fetched items
    .catch((err) => console.error("Error fetching items:", err));
}, []);
```

```
const editItem = async (updatedItem) => {
  try {
    const data = new FormData();
    Object.keys(updatedItem).forEach((key) => {
      data.append(key, updatedItem[key]);
    });

    const response = await fetch(`http://localhost:5000/api/reuse/${updatedItem._id}`, {
      method: "PUT",
      body: data,
    });

    const updatedData = await response.json();
    setItems(items.map((item) => (item._id === updatedData._id ? updatedData : item)));
  } catch (error) {
    console.error("Error updating item:", error);
  }
};
```


Start
User
Admin
Register
No
Login
No
Login
Yes
Yes
Go to Reuse Page
View Reuse Table
View Items
Search Items by name
Buy
Add or Edit Items according to Inventory
PDF report
Delivery Items
Pickup Items
Fill the form
Inform the center
View Report
Edit Form
Submit Form
Save Changes
Logout

3.4. Amarasinghe S A R U – (IT23216778)

(Pickup Scheduling System)

3.4.1. Completion Level

Current Status: 70% Complete

3.4.2. Completed (Done)

- Schedule pickup form.
- Location picking system when scheduling pickups.
- View pickup report after scheduling.
- Update scheduled pickup details.
- Deleted/cancel scheduled pickups.
- Filter by date option in scheduled pickups page.
- View scheduled pickups of all users as an admin from admin dashboard. Can be downloaded as a PDF.
- Home page.
- Date validations, contact number validations, notifications, pop-ups, etc.

3.4.3. Work not Completed (Doing)

- Improving navigation features: 60%
- Fixing update pickup location system in update section of scheduled pickups page: 20%
- Make the downloadable PDF report more detailed: 50%

3.4.4. To be Completed (To-Do)

- About us page.
- Collaborate with other members on the complete integration of the project.

3.1.5. Queries Used

Find a pickup by ID

```
const pickup = await Pickup.findById(pickupId);
```

Find All Pickups for a Specific User

```
const pickups = await Pickup.find({ user: userId })
```

Update a Pickup by ID

```
const updatedPickup = await Pickup.findByIdAndUpdate(
```

Find All Pending Pickups After a Given Date

```
        const pickups = await Pickup.find({
  scheduledTime: { $gte: new Date(date) },
  status: 'pending',
});
```

3.1.6. Algorithms

Filter by date: Admin dashboard

```
// If a date is provided, filter pickups for that specific date
const query = date ? {
  scheduledTime: {
    $gte: new Date(new Date(date).setHours(0, 0, 0, 0)), // Start of the day
    $lt: new Date(new Date(date).setHours(23, 59, 59, 999)) // End of the day
  }
} : {};
```

Cancel pickup: Admin dashboard

```
// Check if the pickup is within 24 hours of scheduled time
const timeDiff = pickup.scheduledTime - Date.now();
if (timeDiff < 24 * 60 * 60 * 1000) {
  return res.status(400).json({ error: 'Cannot delete. Pickup time is less than 24 hours away.' });
}
```

Modify pickup assignment: Admin dashboard

```
// Check 24-hour modification window
const timeDiff = pickup.scheduledTime.getTime() - Date.now();
  if (timeDiff < 24 * 60 * 60 * 1000) {
    return res.status(400).json({
      error: "Cannot modify within 24 hours of scheduled time" });}
```

Delete pickup

```
const handleConfirmDelete = async () => {
  if (!pickupToDelete) return;

  const now = new Date();
  const scheduledTime = new Date(pickupToDelete.scheduledTime);
  const timeDifference = scheduledTime - now;

  if (timeDifference < 24 * 60 * 60 * 1000) {
    setError('Cannot delete. Pickup time is less than 24 hours away.');
    setTimeout(() => setError(''), 3000);
    setShowDeleteModal(false);
    return;
  }
```

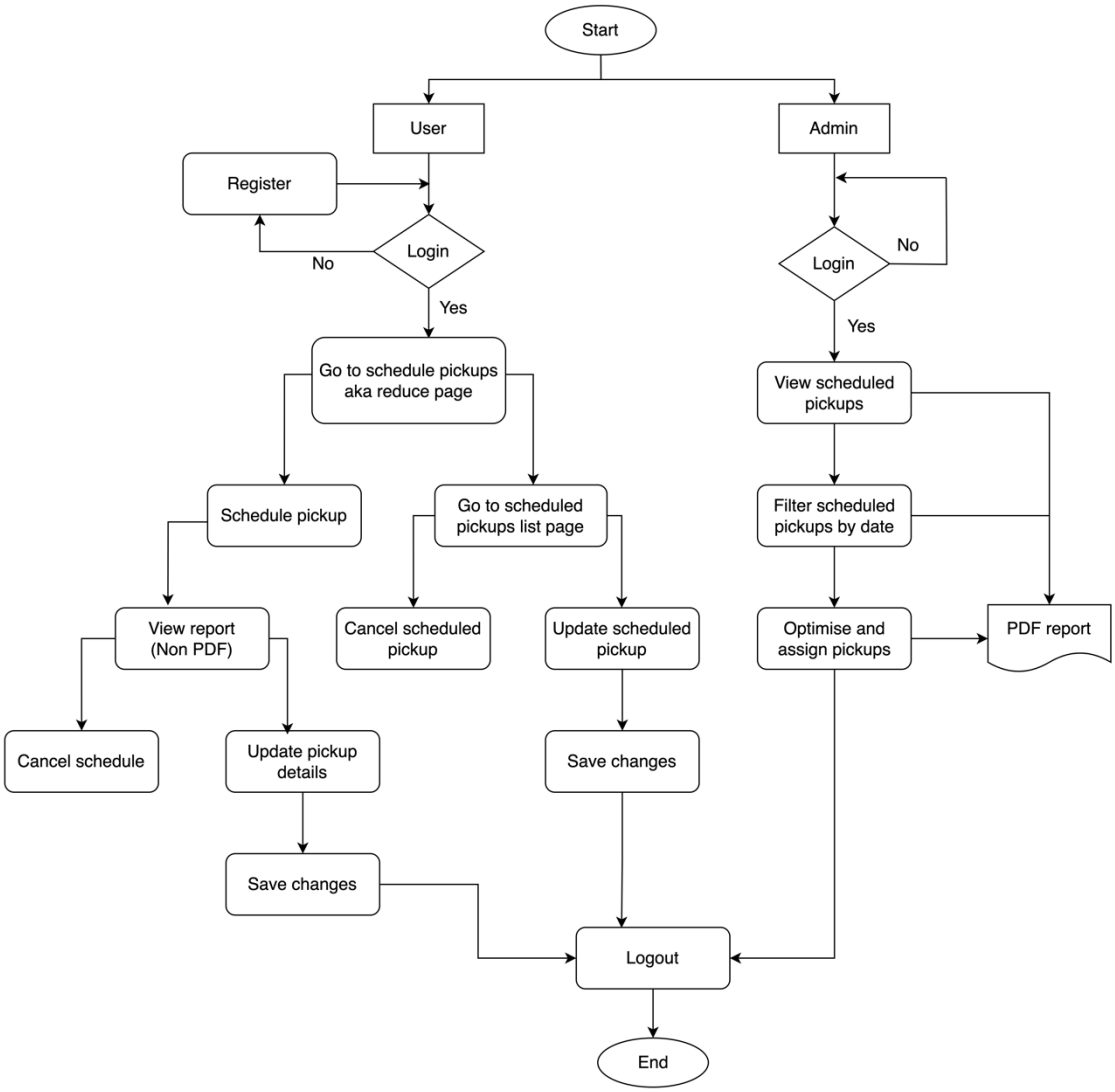
Update pickup details

```
const handleUpdateClick = (pickup) => {
  const now = new Date();
  const scheduledTime = new Date(pickup.scheduledTime);
  const timeDifference = scheduledTime - now;

  if (timeDifference < 24 * 60 * 60 * 1000) {
    setError('Cannot update. Pickup time is less than 24 hours away.');
    setTimeout(() => setError(''), 3000);
    return;
  }
```

Delete/Cancel scheduled pickup

```
const handleDeleteClick = (pickup) => {
  const now = new Date();
  const scheduledTime = new Date(pickup.scheduledTime);
  const timeDifference = scheduledTime - now;
```

```
if (timeDifference < 24 * 60 * 60 * 1000) {
    setError('Cannot delete. Pickup time is less than 24 hours away.');
    setTimeout(() => setError(''), 3000);
    return;
  }
```



3.5 – KULARATHNE K T A (IT23320550)

(User and HR management)

3.5.1 – Completion Level

- Current Status: 70% completed

3.5.2 – Completed

- The user can register and securely log in to the system.
- If user forgot password user can change password according to the given steps.
- Admin can change metal price daily and it show in home page.
- Fetches drivers and show to admin there active status.
- Fetches all users and drivers to admin to see there details.
- Admin can delete users and drivers profile if needed.
- Admin can manually add OT hours and Man hours to each drivers data base.
- Search function to find easily drivers by their name.
- Admin can monthly generate pdf report for each drivers total OT and man hours.
- Show list of drivers leave requests.

3.5.3 – To be completed

- OTP generate system.
- User profile.
- Integrate with driver to show driver ongoing work.
- Implement driver can approve or reject leave requests.

3.5.4 – Queries used

- Get all users

```
const users = await User.find({});
```

- Add OT and Man hours

```
const newEntry = new DriverHours({ driverId, date, manHours, otHours });
await newEntry.save();
```

- Reverse transaction by ID

```
const newUser = new User({ name, email, password, role: role || "user" });
```

- Update metal prices

```
const updatedMetal = await MetalPrice.findOneAndUpdate(
    { metalType },
    { pricePerKg },
    { new: true }
);
```

- Get metal prices

```
const metals = await MetalPrice.find({});
```

- Add new item

```
const newItem = new ReducedItem({ itemName, price, imageName });
await newItem.save();
```

- Generate OT and man hours pdf report

```
const report = await DriverHours.aggregate([
    {
        $group: {
            _id: "$driverId",
            totalManHours: { $sum: "$manHours" },
            totalOtHours: { $sum: "$otHours" }
        }
    },
    {
        $lookup: {
            from: "users",
            localField: "_id",
            foreignField: "_id",
            as: "driverInfo"
        }
    },
    {
        $unwind: "$driverInfo"
    },
    {
        $project: {
            _id: 0,
            driverId: "$_id",
            driverName: "$driverInfo.name",
            totalManHours: 1,
            totalOtHours: 1
        }
    }
```

3.5.5 – Flow Chart