Remote Patient Monitoring System

[redacted]

MASTER [redacted]
APPLICATIONS
with
Artificial Intelligence & Machine Learning

by

Name	SAP ID
Jayant Dev	500120083
Jayesh Patni	500123368
Devansh Maheshwari	500117998
Sunil Singh	500123144
Vinay kumar Pandey	500120087

Under the guidance of
Dr. Pooja Kumari Ma'am

School of Computer Science, UPES
Bidholi, Via Prem Nagar, Dehradun, Uttarakhand

MAY – 2025

“Remote Patient Monitoring fulfillment **MASTER** Cybernautics **Feb,** **. Pooja kumari ma’am,**

(Jayant Dev)
SAP - 500120083
(Devansh Maheshwari)
SAP - 500117998
(Sunil Singh)
SAP - 500123144
(Vinay Kumar Pandey)
SAP - 500120087
(Jayesh Patni)
SAP - 500123368

: 22

Dr. Pooja Kumari Ma’am

[redacted]. Pooja Kumari Ma'am, [redacted] she [redacted] her [redacted] sincerely [redacted] Computer [redacted] in Remote Patient Monitoring system. [redacted] Course [redacted], (Ritesh kumar Sir) [redacted].

(Jayant Dev)
SAP- 500120083

(Devansh Maheshwari)
SAP- 500117998

(Sunil Singh)
SAP- 500123144

(Vinay Kumar Pandey)
SAP- 500120087

(Jayesh Patni)
SAP- 500123368

Abstract

Remote patient monitoring (RPM) is now a crucial tool for improving healthcare accessibility, real-time tracking, and proactive treatment delivery due to the development of digital healthcare. In order to gather, process, analyze, and display real-time health vitals from wearable devices, this project offers a powerful Remote Patient Monitoring System (RPMS) that was constructed [redacted] the Fitbit Web API.

Patients can safely connect their Fitbit devices to the system, which facilitates the smooth transfer of physiological data like blood oxygen level, heart rate, and step count. In order to detect health irregularities and enable prompt alerts and clinical decision-making, a specialized AI anomaly detection module uses trained machine learning models to process the incoming data.Chart.js and React are used to create user-friendly, interactive dashboards that display the data, and a cloud-based MongoDB database with encryption and authentication safeguards protects all of the records.

The system's clean architecture and RESTful APIs were designed with scalability and modularity in mind. In the event that actual Fitbit devices are not available, a simulator mode is also offered for offline testing. This project exemplifies the full end-to-end integration of contemporary web development, artificial intelligence, and healthcare informatics, resulting in better patient outcomes, individualized treatment, and less clinical workload. It lays the groundwork for future integration with real-time physician dashboards and EHR systems for remote consultations and the management of chronic illnesses.

Contents

List of Figures

Chapter 1

Introduction

The rapid evolution of digital technologies has revolutionized almost every aspect of modern life, and healthcare is no exception. One of the most transformative innovations in recent years is the rise of Remote Patient Monitoring (RPM) systems. These systems leverage wearable technologies, real-time data acquisition, cloud platforms, and artificial intelligence (AI) to enable healthcare providers to track patient health metrics continuously, without the need for physical appointments.

This project titled "Remote Patient Monitoring System Using Fitbit API Integration and MERN Stack" serves as a step forward in the democratization of personalized healthcare. It is an end-to-end solution that enables the seamless monitoring of critical health parameters other fitness metrics in real-time. By integrating consumer-grade wearable devices like Fitbit .js,) architecture, the system ensures usability, affordability, and effectiveness.

With chronic diseases on the rise and healthcare resources becoming increasingly strained, such systems can provide timely intervention, reduce hospital readmissions, and empower patients to take control of their health through real-time feedback and awareness.

1.1 History

Historically, the healthcare industry relied heavily on in-person consultations and manual documentation of patient vitals. This approach was not only resource-intensive but also inherently reactive, often catching health issues only after they had escalated. As digital transformation swept across industries, healthcare began exploring data-driven solutions.

The initial adoption of Electronic Health Records (EHRs) was a significant step, but the real turning point came with the advent of wearable health devices. Fitbit, launched in 2009, introduced the concept of personal health tracking to a mainstream audience. Over the years, Fitbit evolved from a simple pedometer to a sophisticated device capable of measuring heart rate variability, sleep patterns, and even stress levels.

The launch of Fitbit's Web API opened new possibilities for developers to build custom healthcare solutions on top of this rich data ecosystem. When combined with full-stack web technologies, this gave rise to flexible, scalable, and highly personalized RPM systems.

1.2 Requirement Analysis

Understanding both functional and non-functional requirements is essential for building a reliable RPM system that meets user expectations and regulatory standards.

Functional Requirements:

- User Authentication: Secure login and session management using OAuth2 protocol via Fitbit.
- Data Acquisition: Automatic retrieval of health metrics (e.g., heart rate) from Fitbit.
- Data Visualization: Graphical representation of patient data over time for easy comprehension.
- Anomaly Detection: Identification of outliers or critical values in health data.
- Real-Time Dashboard: Dynamic frontend that updates data periodically.
- Data Storage: Long-term storage of vitals and metadata in a structured database.

Non Functional Requirements:

- Reliability: Consistent system performance with minimal downtime.
- Scalability: Ability to support a growing user base and increased data volume.
- Security: Encryption of sensitive data, secure token management, and access controls.
- Usability: User-friendly interface suitable for patients and healthcare professionals.
- Compliance: Adherence to data privacy regulations such as HIPAA and GDPR.

1.3 Main Objective

The central objective of this project is to create a secure, real-time, and AI-assisted Remote Patient Monitoring System that enables the continuous collection, analysis, and visualization of patient health data using Fitbit wearables. This system aims to enhance patient care, facilitate early detection of medical conditions, and reduce the burden on healthcare infrastructure by allowing remote supervision and timely medical intervention.

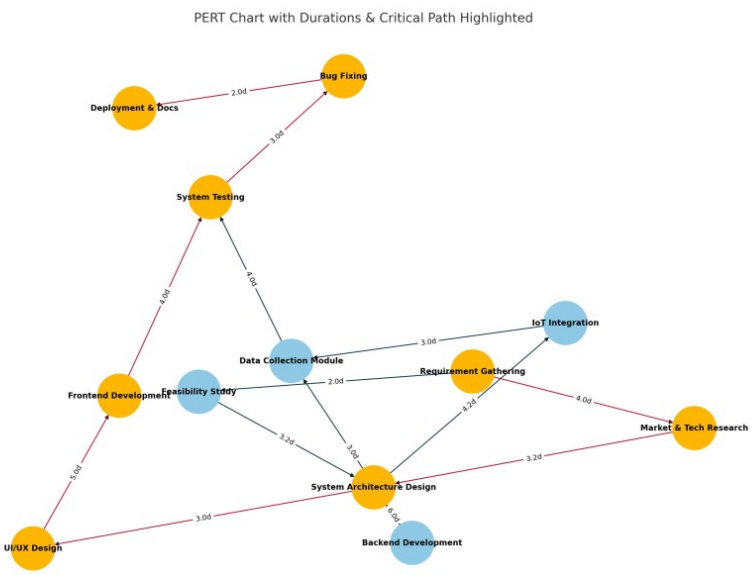


Figure 1.1: Pert Chart

1.4 Sub Objectives

- Implement secure Fitbit OAuth2 login and authentication.
- Retrieve and process real-time health data from Fitbit servers.
- Design an intuitive dashboard for visualizing patient metrics.
- Store vital data efficiently in a scalable database.
- Incorporate basic anomaly detection logic to flag unusual values.
- Establish a foundation for integrating predictive analytics in the future.

Traditional patient monitoring methods primarily rely on periodic check-ups and manual data entry. Patients visit clinics for routine assessments, where vital signs are measured and recorded. However, this episodic care model has several limitations:

- Delays in diagnosis and treatment due to infrequent monitoring.
- Lack of patient engagement in their own health management.
- High costs associated with in-person visits and hospital stays.
- Inaccessibility for rural or mobility-impaired patients.

Commercial RPM solutions have emerged but tend to be expensive, lack customization, and often do not integrate well with widely used consumer wearables.

2.2 Motivations

The surge in chronic illnesses like cardiovascular disease, diabetes, and hypertension, along with an aging population, demands a paradigm shift from reactive to proactive care. Remote Patient Monitoring (RPM) not only extends healthcare services beyond the hospital but also fosters personalized medicine and preventive healthcare.

- **Rising chronic illness burden:** Chronic patients require continuous care, which is costly and resource-intensive in a hospital setting.
- **Post-pandemic digital transformation:** COVID-19 highlighted the need for telehealth and remote diagnostics.
- **Advances in AI IoT:** Pervasive computing, wearable sensors, and deep learning have matured enough to support large-scale deployments.
- **Cost efficiency:** Continuous monitoring can significantly reduce emergency visits, readmissions, and insurance claims.

- **Real-time health insights:** Timely detection of anomalies (e.g., arrhythmias, oxygen desaturation) can prevent severe outcomes.
- **Empowering patients:** Provides individuals with a sense of control over their health, encouraging better compliance and engagement.

2.3 Proposed System

The proposed AI-enabled Remote Patient Monitoring System (RPMS) is a comprehensive health-tech platform that integrates wearable IoT devices, cloud infrastructure, and machine learning algorithms to deliver real-time, intelligent monitoring of patients' physiological signals. The system is designed to facilitate autonomous data collection, continuous analysis, early detection of anomalies, and automated alerting for timely clinical interventions.

System Characteristics:

- **Device-Agnostic Data Acquisition:** Support for a variety of wearable devices and biosensors, including ECG patches, smartwatches, pulse oximeters, and temperature sensors.
- **Personalized Health Monitoring:** Models that adapt to individual baselines instead of using generic thresholds.
- **Edge and Cloud Intelligence:** Real-time AI models running on-device for low-latency response, complemented by cloud-based historical analysis.
- **End-to-End Security and Compliance:** Implementation of data privacy protocols aligned with HIPAA, GDPR, and HL7 FHIR standards.
- **Multi-Stakeholder Dashboard:** Interfaces tailored for patients, caregivers, clinicians, and hospital administrators.

2.4 Modules

The Remote Patient Monitoring System (RPMS) is composed of multiple interlinked modules, each performing critical functions. Together, they ensure the seamless acquisition, processing, analysis, storage, and visualization of real-time physiological data. Below is a detailed explanation of each module.

2.4.1 Data Acquisition Module

Objective: To automatically collect real-time physiological data (such as heart rate, steps, calories) from the Fitbit wearable device using the Fitbit Web API.

Functionality:

- OAuth2 Authentication: The module initiates a secure authorization flow using Fitbit's OAuth 2.0 protocol. Upon successful user login, an access token is generated.
- Token Management: This includes fetching the access token and refresh token and storing them securely. [redacted] acquire [redacted] new [redacted] without requiring user interaction.
- Data Retrieval: JSON-formatted responses are fetched containing timestamped data.

Technical Considerations:

- Rate Limiting: Fitbit imposes a rate limit on API requests. The system respects these quotas using controlled polling intervals.
- Timestamp Alignment: Data is timestamped and synchronized with the server's UTC time for consistency.

Tools Technologies:

- Node.js (Axios/Fetch)
- Fitbit Developer API
- OAuth2.0 Authorization Code Flow

Outcome:

- A real-time stream of health data is ingested into the system, forming the foundation for subsequent processing and analytics.

2.4.2 Data Preprocessing Module

Objective: To clean, validate, and transform raw Fitbit data into a structured and usable format suitable for storage and further analysis.

Functionality:

- JSON Parsing: Fitbit API returns nested JSON responses. This module extracts relevant fields and flattens the structure for processing.
- Missing Data Handling:
  - Fills in gaps in time series with null or interpolated values.
  - Ignores incomplete data segments.
- Normalization
  - Standardizes units (e.g., beats per minute for heart rate).
  - Normalizes time zones to UTC for uniformity.
- Validation: Ensures values are within human physiological limits:
  - Heart rate: [40–220] BPM
  - Step count: 0
- Time Binning (Optional): Aggregates values per minute or hour, depending on selected view.

Tools Technologies:

- JavaScript/Node.js
- Moment.js (for timestamp handling)
- Custom data sanitization functions

2.4.3 AI Anomaly Detection Module

Objective: To identify irregularities or abnormal patterns in the physiological data, indicating potential health issues or sensor anomalies.

Current Implementation:

- Rule-Based Detection:
  - Threshold alerts based on clinical norms:
    - Heart rate ¿ 120 BPM (tachycardia alert)
    - Heart rate ¡ 50 BPM (bradycardia alert)
  - Sudden changes in step count or activity during inactivity periods.
  - Real-Time Flagging:
    - When an abnormal value is detected, it is flagged in the frontend and logged in the backend.
    - Alerts are color-coded in the visual dashboard.

Future Implementation:

- Machine Learning Models:
  - Isolation Forest for unsupervised anomaly detection on [redacted] networks for sequential prediction and anomaly scoring.
  - K-Means clustering for segment-based outlier detection.
- Benefits:
  - Enables early detection of health issues such as arrhythmia.
  - Reduces false alarms through adaptive logic.
  - Enhances reliability over static thresholds in the long term.
- Tools Technologies:
  - Node.js or Python (for AI model integration)
  - Scikit-learn, TensorFlow (planned)
  - Alert logic via React and backend routes

2.4.4 Visualization and Reporting Module

Objective: To present user health data in a clear, engaging, and interactive format that supports both short-term monitoring and long-term trend analysis.

Functionality:

- Interactive Dashboards:
  - Live line charts for heart rate, steps, sleep cycles, etc.
  - Color-coded anomaly markers on ch
- Time Filtering:
  - Users can select [redacted].
- Responsive UI:
  - Fully mobile-responsive interface built with React and Tailwind CSS.
- Charting Library:
  - ses Chart.js with customized tooltips, labels, and scaling.
  - Automatically updates when new data is fetched or anomalies are detected.

Reports and Insights:

- Provides downloadable health summaries (planned feature).

- Daily average, peak, and range metrics shown numerically.
- Users can compare historical trends over time.

Tools Technologies:

- React.js
- Chart.js
- Tailwind CSS
- REST API (for fetching data)

2.4.5 Data Storage and Security Module

Objective: To persistently store structured health data while ensuring integrity, confidentiality, and availability through secure backend practices.

Functionality:

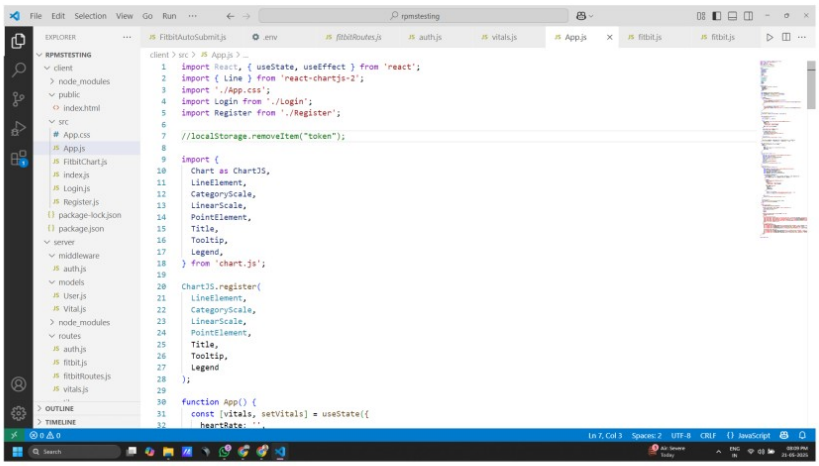
- Database Design:
  - MongoDB used for its flexibility with time-series and JSON-like documents.
  - Collections:
    - users: Stores user profiles and Fitbit token data.
    - vitals: Stores timestamped vitals data per user.
    - alerts: Stores anomaly flags and timestamps.
- Schema Management:
  - Mongoose schemas ensure consistent structure.
  - Includes validation for ranges, data types, and required fields.
- Security Practices:
  - Authentication: Secure token management using OAuth2 and JWT.
  - Access Control: Routes protected with middleware; users can only access their own data.
  - Encryption
    - Access tokens encrypted before storage.
    - HTTPS enforced via SSL.
  - Sanitization: Input data is sanitized to prevent NoSQL injection and XSS attacks.
  - Data Integrity:
    - Automatic retries and logging for failed API calls or insertions.
    - Backup mechanisms can be integrated for redundancy.

Tools Technologies:

- MongoDB with Mongoose
- Node.js with Express
- Helmet, Bcrypt, JWT for security
- Firebase or Netlify (optional deployment platforms)

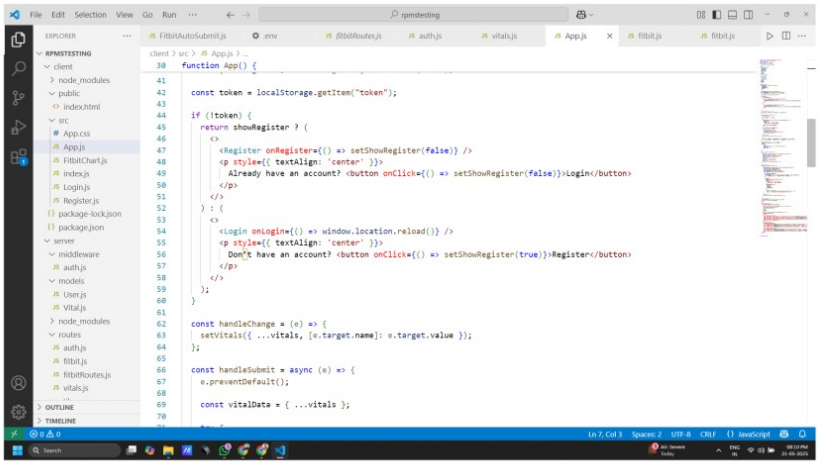
Chapter 3

Implementation/results



```
import React, { useState, useEffect } from 'react';
import { Line } from 'react-chartjs-2';
import './App.css';
import Login from './Login';
import Register from './Register';

//localStorage.removeItem("token");

import {
  Chart as ChartJS,
  LineElement,
  CategoryScale,
  LinearScale,
  PointElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';

ChartJS.register(
  LineElement,
  CategoryScale,
  LinearScale,
  PointElement,
  Title,
  Tooltip,
  Legend
);

function App() {
  const [vitals, setVitals] = useState({
    heartRate: '',
```

Figure 3.1: App.js for frontend access



```
function App() {

  const token = localStorage.getItem("token");

  if (!token) {
    return showRegister ? (
      <>
        <Register onRegister={() => setShowRegister(false)} />
        <p style={{ textAlign: 'center' }}>
          Already have an account? <button onClick={() => setShowRegister(false)}>Login</button>
        </p>
      </>
    ) : (
      <>
        <Login onLogin={() => window.location.reload()} />
        <p style={{ textAlign: 'center' }}>
          Don't have an account? <button onClick={() => setShowRegister(true)}>Register</button>
        </p>
      </>
    );
  }

  const handleChange = (e) => {
    setVitals({ ...vitals, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    const vitalData = { ...vitals };
```

Figure 3.2: App.js login button for user

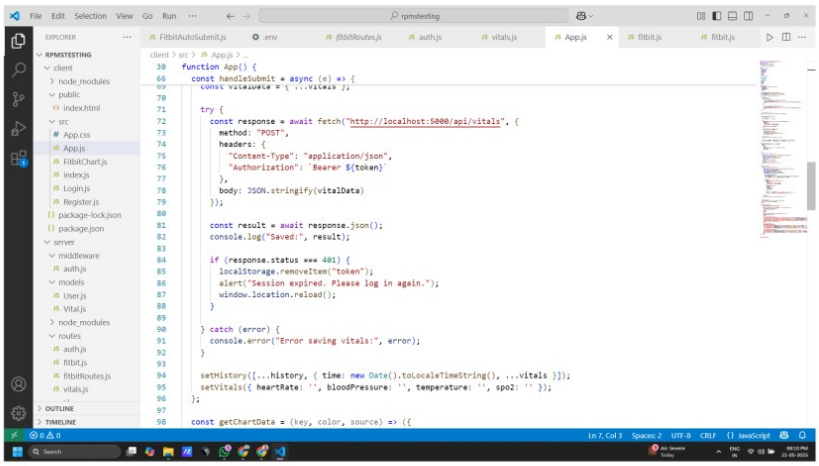


Figure 3.3: Temp token value for App.js

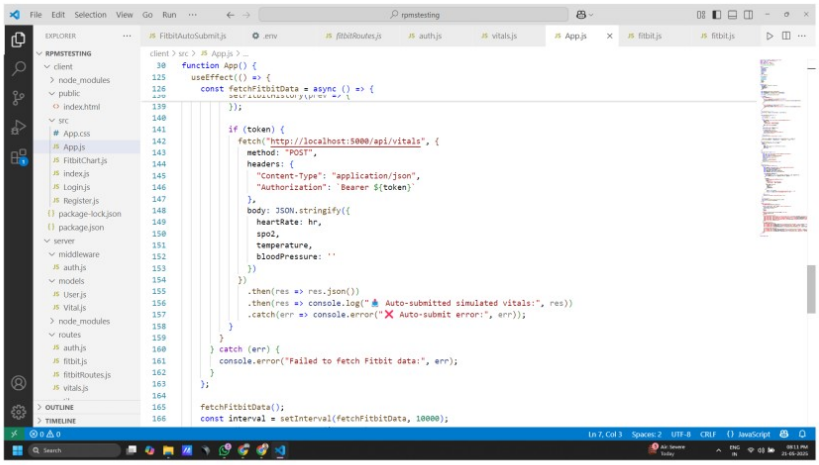


Figure 3.4: App.js console command for saving vitals in frontend

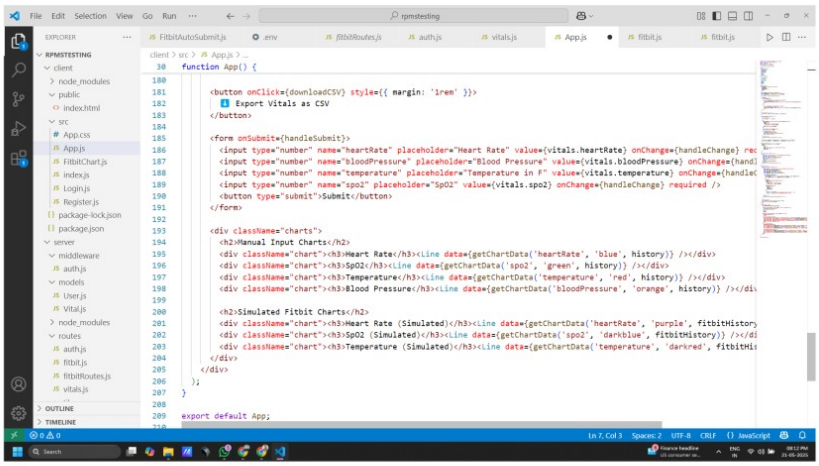


Figure 3.5: App.js form for entering user input

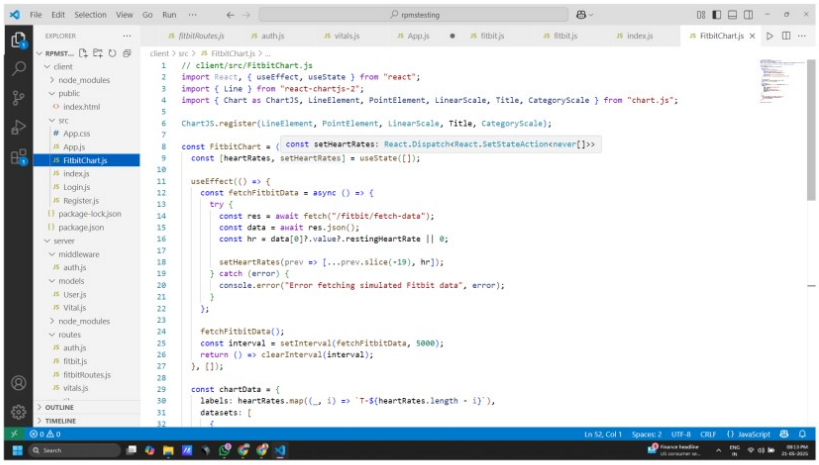


Figure 3.6: Fitbit chart js file for accessing Fitbit vitals

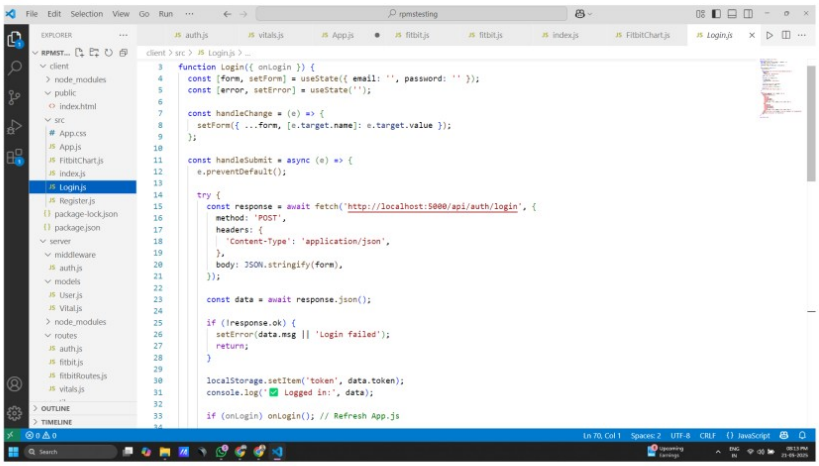


Figure 3.7: Login page for user login with personalized credentials

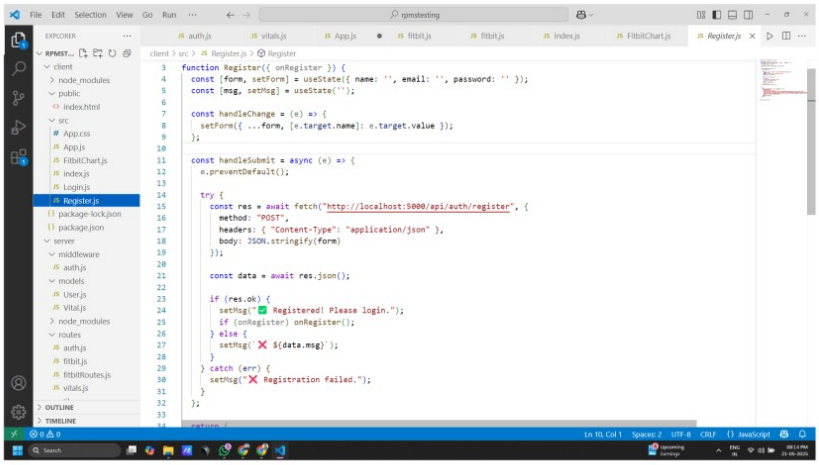
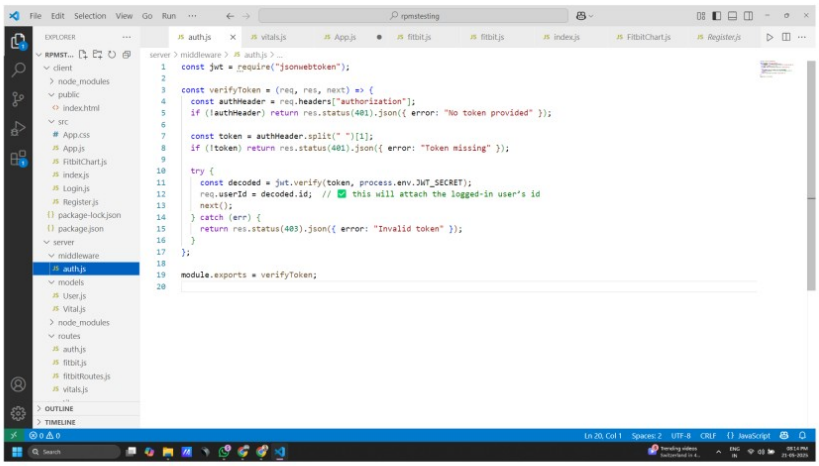
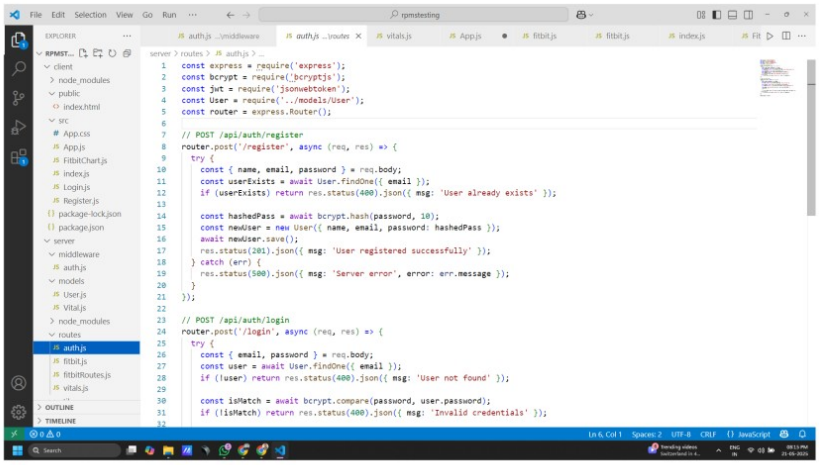


Figure 3.8: Registration page in frontend for new user registration



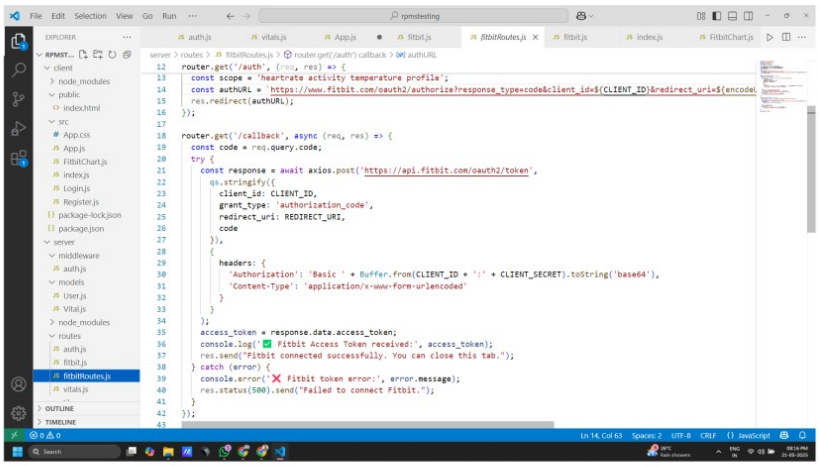
```
const jwt = require("jsonwebtoken");

const verifyToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  if (!authHeader) return res.status(401).json({ error: "No token provided" });

  const token = authHeader.split(" ")[1];
  if (!token) return res.status(401).json({ error: "Token missing" });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.id;  // ✅ this will attach the logged-in user's id
    next();
  } catch (err) {
    return res.status(403).json({ error: "Invalid token" });
  }
};

module.exports = verifyToken;
```

Figure 3.9: Middleware authjs for accessing temp token from local storage



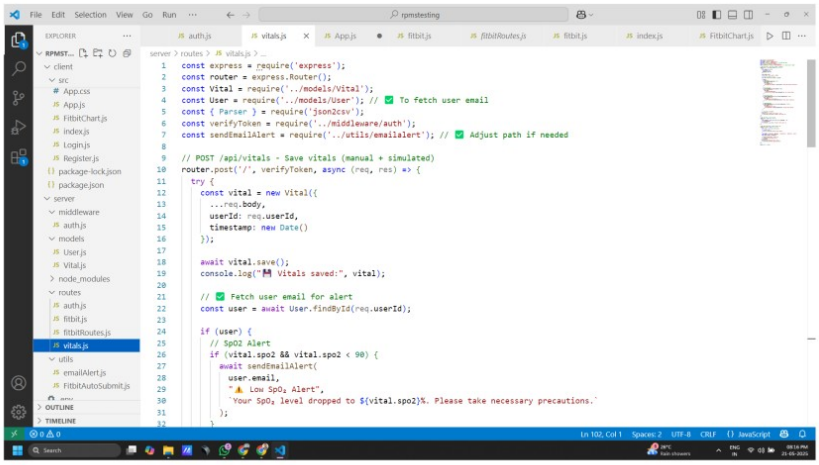
```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const router = express.Router();

// POST /api/auth/register
router.post('/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const userExists = await User.findOne({ email });
    if (userExists) return res.status(400).json({ msg: 'User already exists' });

    const hashedPass = await bcrypt.hash(password, 10);
    const newUser = new User({ name, email, password: hashedPass });
    await newUser.save();
    res.status(201).json({ msg: 'User registered successfully' });
  } catch (err) {
    res.status(500).json({ msg: 'Server error', error: err.message });
  }
});

// POST /api/auth/login
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ msg: 'User not found' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: 'Invalid credentials' });
```

Figure 3.10: authentication js in backend to store credentials



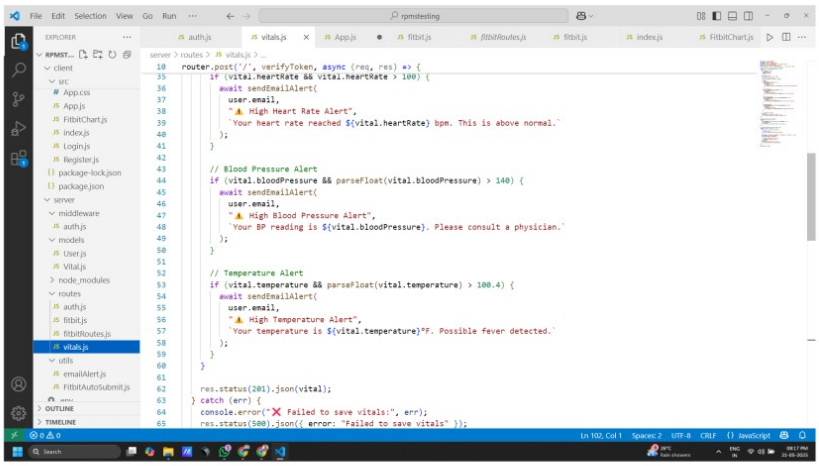
```
router.get('/auth', (req, res) => {
  const scope = 'heartrate activity temperature profile';
  const authURL = `https://www.fitbit.com/oauth2/authorize?response_type=code&client_id=${CLIENT_ID}&redirect_uri=${encodeU
  res.redirect(authURL);
});

router.get('/callback', async (req, res) => {
  const code = req.query.code;
  try {
    const response = await axios.post('https://api.fitbit.com/oauth2/token',
      qs.stringify({
        client_id: CLIENT_ID,
        grant_type: 'authorization_code',
        redirect_uri: REDIRECT_URI,
        code
      }),
      {
        headers: {
          'Authorization': 'Basic ' + Buffer.from(CLIENT_ID + ':' + CLIENT_SECRET).toString('base64'),
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      }
    );
    access_token = response.data.access_token;
    console.log('✅ Fitbit Access Token received:', access_token);
    res.send("Fitbit connected successfully. You can close this tab.");
  } catch (error) {
    console.error('❌ Fitbit token error:', error.message);
    res.status(500).send("Failed to connect Fitbit.");
  }
});
```

Figure 3.11: Fitbit simulated API js



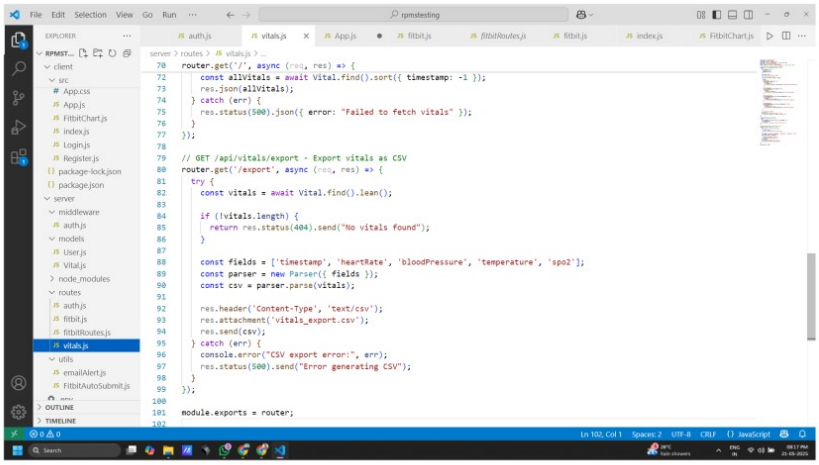
```
const express = require('express');
const router = express.Router();
const Vital = require('../models/Vital');
const User = require('../models/User'); // ✅ To fetch user email
const { Parser } = require('json2csv');
const verifyToken = require('../middleware/auth');
const sendEmailAlert = require('../utils/emailalert'); // ✅ Adjust path if needed

// POST /api/vitals - Save vitals (manual + simulated)
router.post('/', verifyToken, async (req, res) => {
  try {
    const vital = new Vital({
      ...req.body,
      userId: req.userId,
      timestamp: new Date()
    });

    await vital.save();
    console.log("📥 Vitals saved:", vital);

    // ✅ Fetch user email for alert
    const user = await User.findById(req.userId);

    if (user) {
      // SpO2 Alert
      if (vital.spo2 && vital.spo2 < 90) {
        await sendEmailAlert(
          user.email,
          "⚠️ Low SpO₂ Alert",
          `Your SpO₂ level dropped to ${vital.spo2}%. Please take necessary precautions.`
        );
      }
```

Figure 3.12: Vitals js to store and alert in emailer



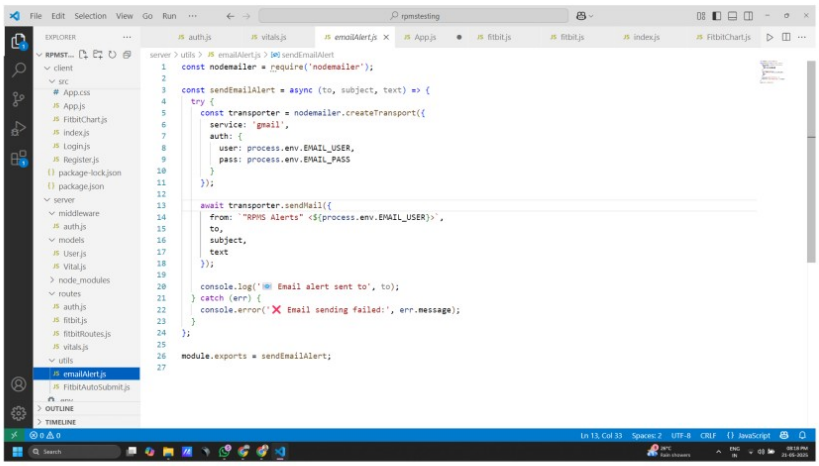
```
router.post('/', verifyToken, async (req, res) => {
    if (vital.heartRate && vital.heartRate > 100) {
      await sendEmailAlert(
        user.email,
        "⚠️ High Heart Rate Alert",
        `Your heart rate reached ${vital.heartRate} bpm. This is above normal.`
      );
    }

    // Blood Pressure Alert
    if (vital.bloodPressure && parseFloat(vital.bloodPressure) > 140) {
      await sendEmailAlert(
        user.email,
        "⚠️ High Blood Pressure Alert",
        `Your BP reading is ${vital.bloodPressure}. Please consult a physician.`
      );
    }

    // Temperature Alert
    if (vital.temperature && parseFloat(vital.temperature) > 100.4) {
      await sendEmailAlert(
        user.email,
        "⚠️ High Temperature Alert",
        `Your temperature is ${vital.temperature}°F. Possible fever detected.`
      );
    }
  }

  res.status(201).json(vital);
} catch (err) {
  console.error("❌ Failed to save vitals:", err);
  res.status(500).json({ error: "Failed to save vitals" });
```

Figure 3.13: Conditions for vitals beyonnd limit in Vitals



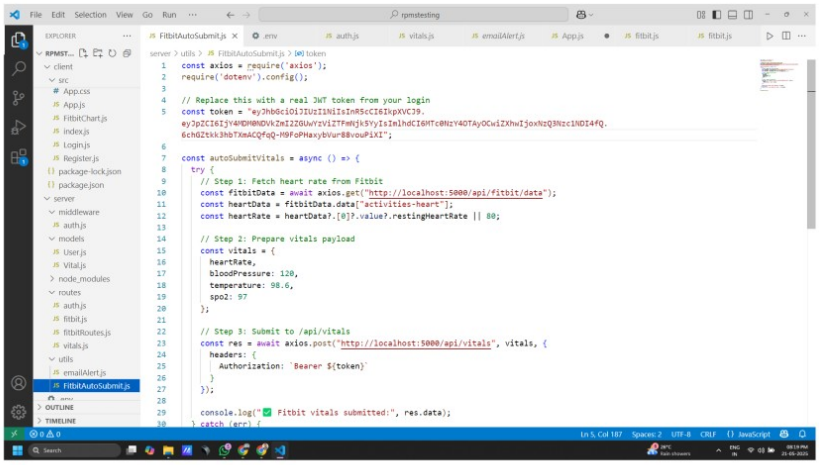
```
router.get('/', async (req, res) => {
  const allVitals = await Vital.find().sort({ timestamp: -1 });
  res.json(allVitals);
 } catch (err) {
  res.status(500).json({ error: "Failed to fetch vitals" });
 }
});

// GET /api/vitals/export - Export vitals as CSV
router.get('/export', async (req, res) => {
 try {
  const vitals = await Vital.find().lean();

  if (!vitals.length) {
   return res.status(404).send("No vitals found");
  }

  const fields = ['timestamp', 'heartRate', 'bloodPressure', 'temperature', 'spo2'];
  const parser = new Parser({ fields });
  const csv = parser.parse(vitals);

  res.header('Content-Type', 'text/csv');
  res.attachment('vitals_export.csv');
  res.send(csv);
 } catch (err) {
  console.error("CSV export error:", err);
  res.status(500).send("Error generating CSV");
 }
});

module.exports = router;
```

Figure 3.14: Feature for download a onetime csv file for a specific user



```
const nodemailer = require('nodemailer');

const sendEmailAlert = async (to, subject, text) => {
  try {
    const transporter = nodemailer.createTransport({
      service: 'gmail',
      auth: {
        user: process.env.EMAIL_USER,
        pass: process.env.EMAIL_PASS
      }
    });

    await transporter.sendMail({
      from: `"RPMS Alerts" <${process.env.EMAIL_USER}>`,
      to,
      subject,
      text
    });

    console.log('📧 Email alert sent to', to);
  } catch (err) {
    console.error('❌ Email sending failed:', err.message);
  }
};

module.exports = sendEmailAlert;
```

Figure 3.15: Emailer using Nodemailer for anomaly



```
const axios = require('axios');
require('dotenv').config();

// Replace this with a real JWT token from your login
const token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpZCI6IjY4MDM0NDVkZmI2ZGUwYzViZTFmNjk5YyIsImlhdCI6MTc0NzY4OTAyOCwiZXhwIjoxNzQ3Nzc1NDI4fQ.
6chQZtkk3hbTXmACQfqQ-M9FoPHaxybVurB8vouPiXI";

const autoSubmitVitals = async () => {
  try {
    // Step 1: Fetch heart rate from Fitbit
    const fitbitData = await axios.get("http://localhost:5000/api/fitbit/data");
    const heartData = fitbitData.data["activities-heart"];
    const heartRate = heartData?.[0]?.value?.restingHeartRate || 80;

    // Step 2: Prepare vitals payload
    const vitals = {
      heartRate,
      bloodPressure: 120,
      temperature: 98.6,
      spo2: 97
    };

    // Step 3: Submit to /api/vitals
    const res = await axios.post("http://localhost:5000/api/vitals", vitals, {
      headers: {
        Authorization: `Bearer ${token}`
      }
    });

    console.log("✅ Fitbit vitals submitted:", res.data);
  } catch (err) {
```

Figure 3.16: Local storage for fitbit simulation

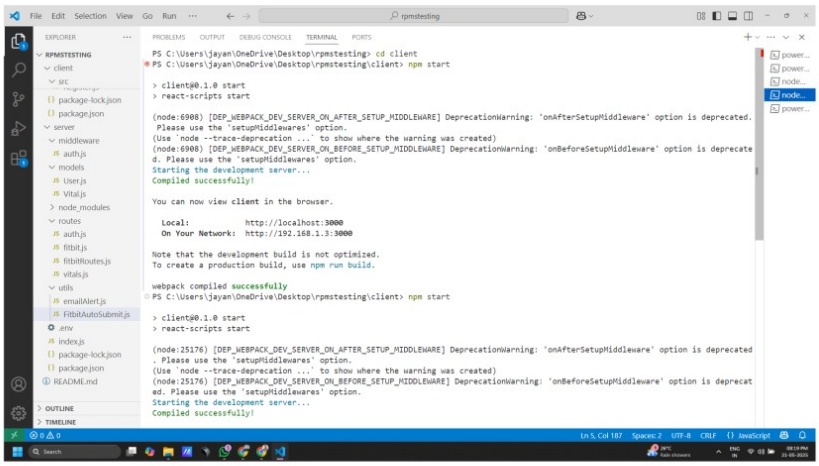


Figure 3.17: Front end client terminal

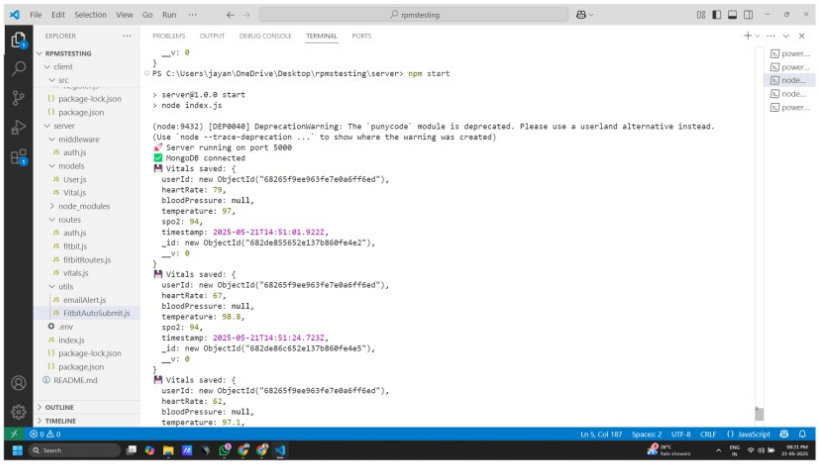


Figure 3.18: Connection with server and auto submission of vitals through fitbit

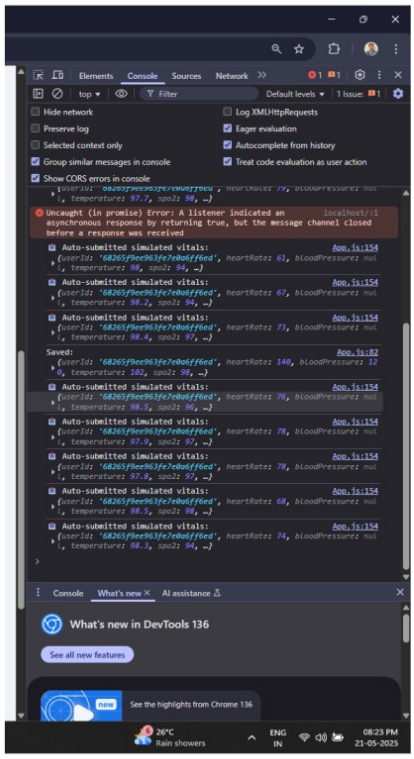


Figure 3.19: Console saving vitals every 10sec from fitbit and manual inegration command

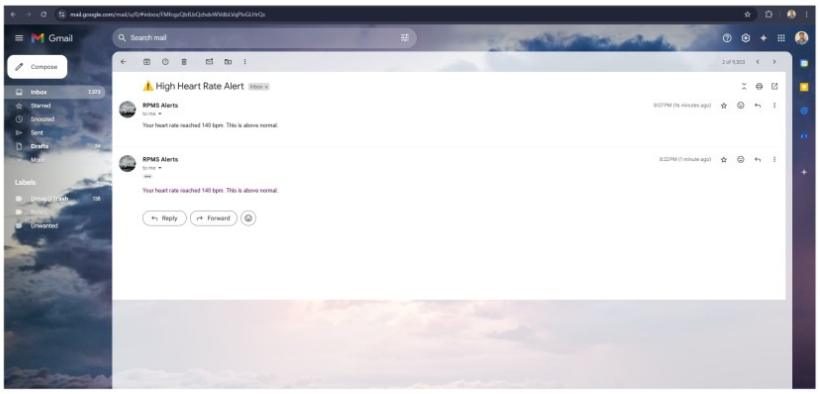


Figure 3.20: Email alerts if anomaly in Vitals

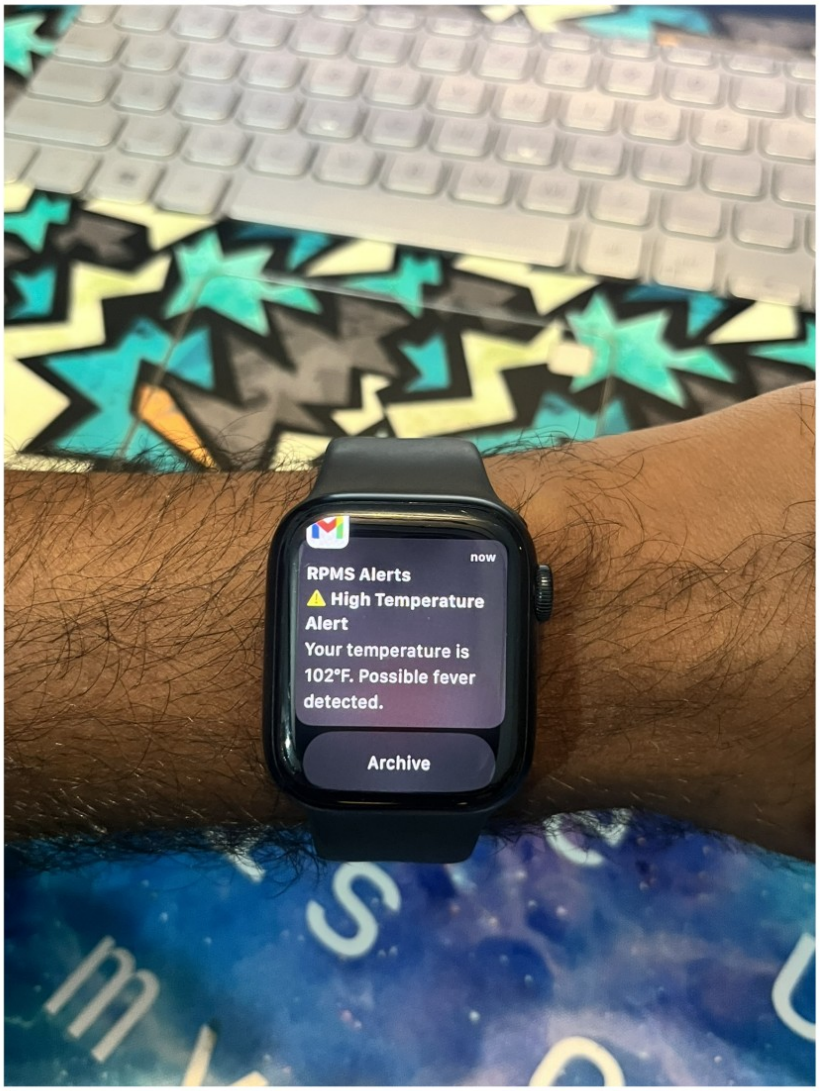


Figure 3.21: Alerts on several device

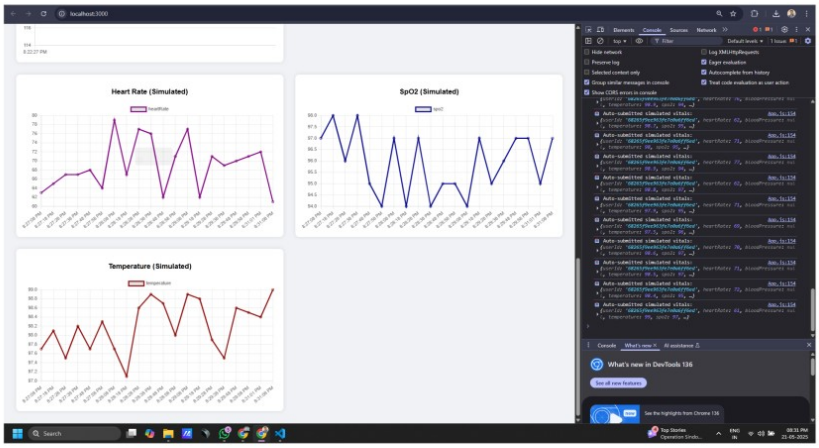
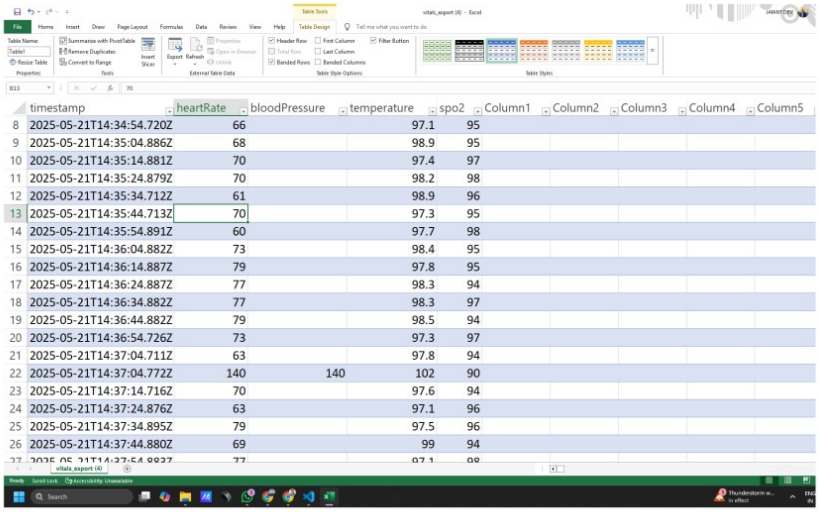


Figure 3.22: Chats for visual representation of vitals on dashboard



	timestamp	heartRate	bloodPressure	temperature	spo2	Column1	Column2	Column3	Column4	Column5
8	2025-05-21T14:34:54.720Z	66		97.1	95					
9	2025-05-21T14:35:04.886Z	68		98.9	95					
10	2025-05-21T14:35:14.881Z	70		97.4	97					
11	2025-05-21T14:35:24.879Z	70		98.2	98					
12	2025-05-21T14:35:34.712Z	61		98.9	96					
13	2025-05-21T14:35:44.713Z	70		97.3	95					
14	2025-05-21T14:35:54.891Z	60		97.7	98					
15	2025-05-21T14:36:04.882Z	73		98.4	95					
16	2025-05-21T14:36:14.887Z	79		97.8	95					
17	2025-05-21T14:36:24.887Z	77		98.3	94					
18	2025-05-21T14:36:34.882Z	77		98.3	97					
19	2025-05-21T14:36:44.882Z	79		98.5	94					
20	2025-05-21T14:36:54.726Z	73		97.3	97					
21	2025-05-21T14:37:04.711Z	63		97.8	94					
22	2025-05-21T14:37:04.772Z	140	140	102	90					
23	2025-05-21T14:37:14.716Z	70		97.6	94					
24	2025-05-21T14:37:24.876Z	63		97.1	96					
25	2025-05-21T14:37:34.895Z	79		97.5	96					
26	2025-05-21T14:37:44.880Z	69		99	94					

Figure 3.23: complete CSV for vitals

Chapter 4

Conclusion

The Remote Patient Monitoring System developed in this project represents a significant advancement in the intersection of wearable technology, real-time health analytics, and full-stack web development. The integration of Fitbit's health tracking API with creates seamless platform capable continuously collecting, analyzing, and visualizing patient health data remotely.

This system is a practical, scalable, and cost-effective solution to the growing demand for personalized and proactive healthcare. With the prevalence of chronic diseases and the increasing pressure on healthcare infrastructure, especially in underserved or rural areas, remote monitoring systems like this one offer the potential to shift the model of healthcare from reactive to preventive.

The system achieves several key objectives:

- Real-time Health Monitoring: Patients' vitals , activity, sleep continuously reliably.
- Secure Fitbit Integration: OAuth2-based authentication ensures user privacy and data protection while maintaining seamless connectivity with Fitbit devices.
- Robust Data Management: Health data is accurately preprocessed, stored, and managed, ensuring consistency and integrity.
- AI-based Anomaly Detection: The system is capable of identifying health anomalies using rules and offers a clear path for future enhancement via predictive machine learning algorithms.
- Dynamic Visualization: A React-based dashboard empowers users to view health trends in a meaningful and intuitive way, improving self-awareness and engagement.
- End-to-End Security: Sensitive health data is encrypted and secured with JWT tokens and best practices in API and database design.

Impact and Value This system not only helps in early detection and alerts for abnormal health patterns but also empowers patients to take charge of their health data. It fosters a sense of autonomy and engagement, encouraging lifestyle changes through continuous feedback. On the provider side, the system allows doctors and medical staff to remotely supervise patients, enabling data-driven interventions before problems escalate.

Furthermore, the modular design of the platform ensures that it can be expanded to support additional wearables, features like doctor-patient messaging, emergency alerts,

multi-patient dashboards, and predictive health models. It lays a strong foundation for future integration of AI/ML, telehealth platforms, and larger enterprise hospital systems.

Limitations and Future Work While the current implementation delivers on its core goals, it is designed with scalability in mind. Future improvements may include:

- Advanced anomaly detection using LSTM or Isolation Forest for more personalized insights.
- Integration with Apple HealthKit, Samsung Health, or Google Fit to make the system more hardware-agnostic.
- Real-time notification system to alert users or doctors via email/SMS during critical health events.
- User roles and access control, enabling doctor dashboards with multi-patient views.
- HIPAA/GDPR-compliant data handling, necessary for real-world deployment in sensitive healthcare environments.

Final Thoughts This project demonstrates how consumer technology, open-source software, and AI can work together to revolutionize healthcare delivery. The Remote Patient Monitoring System is a testament to the potential of technology to enhance quality of life, reduce the burden on hospitals, and bring healthcare to every corner of the world — right from a wrist-worn device.

The work completed in this project is not merely a prototype; it is a vision of what healthcare can become: smart, accessible, proactive, and personalized. It underscores the growing trend towards digital health and marks a step forward in reshaping the future of global healthcare systems.

Bibliography

[1] . (2024). Reference. /

[Documentation.

] ReactJS Core Team. (/

[4] /

[5] ExpressJS. ( ). for Node.js.

] OpenAI. ( ). AI-Based Health Anomaly Detection Concepts. Accessed through ChatGPT for implementation references.

[7] W3C. (2024).

[8] Singh, K., Kumar, M. (2022). A Review on Remote Patient Monitoring using IoT and Machine Learning. International Journal of Computer Applications, 184(33), 45–51.

Jayant Project

ORIGINALITY REPORT

14%	10%	2%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Petroleum and Energy Studies Student Paper	9%
2	Submitted to College of Engineering Trivandrum Student Paper	1%
3	Xu-Jun Jian, Chao-Hung Wang, Tieh-Cheng Fu, Shiyang Lyu, David Taniar, Tun-Wen Pai. "Smartwatch-Based Data Analytics and Feature Selection for Heart Failure Assessment", International Journal of Mobile Computing and Multimedia Communications, 2025 Publication	1%
4	www.coursehero.com Internet Source	1%
5	Submitted to Fisk University Student Paper	1%
6	pomcor.com Internet Source	<1%
7	Submitted to University of Northampton Student Paper	<1%
8	Submitted to University of Wales Institute, Cardiff Student Paper	<1%

9	dos Santos, Pedro Emanuel Albuquerque e Baptista. "Applications Across Co-Located Devices", Universidade NOVA de Lisboa (Portugal), 2024 Publication	<1%
10	supermetrics.freshdesk.com Internet Source	<1%
11	www.ryerson.ca Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		