COLLEGE CODE:8206

COLLEGE NAME:ARASU ENGINEERING COLLEGE

DEPARTMENT:ARTIFICIAL INTELLIGENCE&DATA SCIENCE

STUDENT NM ID: 27F25B9977FB4A9A79F4AA0B2CF412C7

ROLL NUMBER :820623243019

NAME:S.HARITHA

DATE:24/10/2025

Completed the project named as:

HACKATHON PHASE_1 REAL-TIME NOTIFICATION SERVICE

SUBMITTED BY,

S.HARITHA,

820623243019.

PHASE_1 REAL TIME NOTIFICATION SERVICE FOR (PHASE_1. HACKATHON)

PROJECT&OVERVIEW

Problem Statement:

In many web or mobile systems, users need instant updates — like chat messages, delivery alerts, or system status.
Traditional systems refresh slowly or require manual reloads. Our project solves this by building a real-time notification platform that instantly pushes messages from server to users.

Objectives:

- To develop a real-time notification service capable of delivering instant updates using WebSocket-based communication.
- To enable multi-channel notifications (web, email, SMS, push).
- To create a scalable, fault-tolerant backend for handling high volumes of events.
- To provide a clean, responsive front-end interface where users can manage, view, and filter notifications.
- To ensure message reliability, persistence, and delivery tracking.

Key Features:

- Send and receive notifications instantly using WebSockets (Socket.IO)
- Separate admin and user views (sender / receiver)
- Store messages and timestamps in a database
- Show “latest notification” dynamically in UI
- Scalable backend with Node.js + Express

Expected Outcome:

At the end of the project, a functional **Real-Time Notification Service** will be developed that ensures instant communication between systems and users. The service will enhance user experience, increase responsiveness, and reduce latency in message delivery, making it suitable for applications like e-commerce alerts, chat systems, and IoT updates.

2. Technology Stack & Environment Setup

Backend:

- Node.js – JavaScript runtime environment
- Express.js – Backend web framework
- Socket.IO – For real-time communication (WebSocket protocol)

Frontend:

- React.js – To build interactive UI
- Axios / Socket.IO-Client – For API calls and real-time connection

Database:

- **MongoDB** for storing notification data, user preferences, and delivery logs.
- **Redis** for message queueing and fast pub/sub operations.

Tools & Environment:

- **Postman** for API testing.
- **Git & GitHub** for version control.
- **VS Code** for development.
- **Docker** for containerized deployment.
- **IBM Cloud / AWS / Heroku** for hosting and scalability.

3. API Design & Data Model

Planned REST Endpoints:

- `POST /notifications` → Create and send new notification.
- `GET /notifications/:userId` → Retrieve notifications for a user.
- `PUT /notifications/:id/read` → Mark notification as read.
- `GET /status/:id` → Check delivery status of a notification.

Request/Response Format:

```
//Example request
{
  "userId": "12345",
  "title": "New Message",
  "message": "You have a new message from Admin",
  "type": "alert"
}

// Example response
{
  "notificationId": "abcd1234",
  "status": "sent",
  "timestamp": "2025-10-24T10:00:00Z"
}
```

Database Schema (MongoDB Example):

```
 {
  "notificationId": String,
  "userId": String,
  "title": String,
  "message": String,
  "type": String,
  "status": String,
  "timestamp": Date
}
```

4. Front-End UI/UX Plan

Wireframes:

- Dashboard displaying live notifications.
- Pop-up alert panel and notification center.
- User settings for enabling/disabling notification types.

Navigation Flow:

1. User logs in →
2. Real-time connection established via Socket.io →
3. New notification appears instantly in UI →
4. User can view details or dismiss →
5. Notification status updates in backend.

State Management Approach:

1. Global state handled via **Redux** (notifications list, unread count).
2. Component-based subscription to updates using WebSocket listeners

5. Development & Deployment Plan

Team Roles:

- Backend Developer – Node.js + Socket.IO
- Frontend Developer – React UI + Socket.IO Client
- Tester – Testing real-time flow and edge cases
- Deployment Manager – GitHub + Render / Vercel / Netlify deployment

Git Workflow:

`main` branch for production, `dev` for integration.

- Feature branches for each module (`feature/api`, `feature/ui`, etc.).
- Pull requests and code reviews before merging.

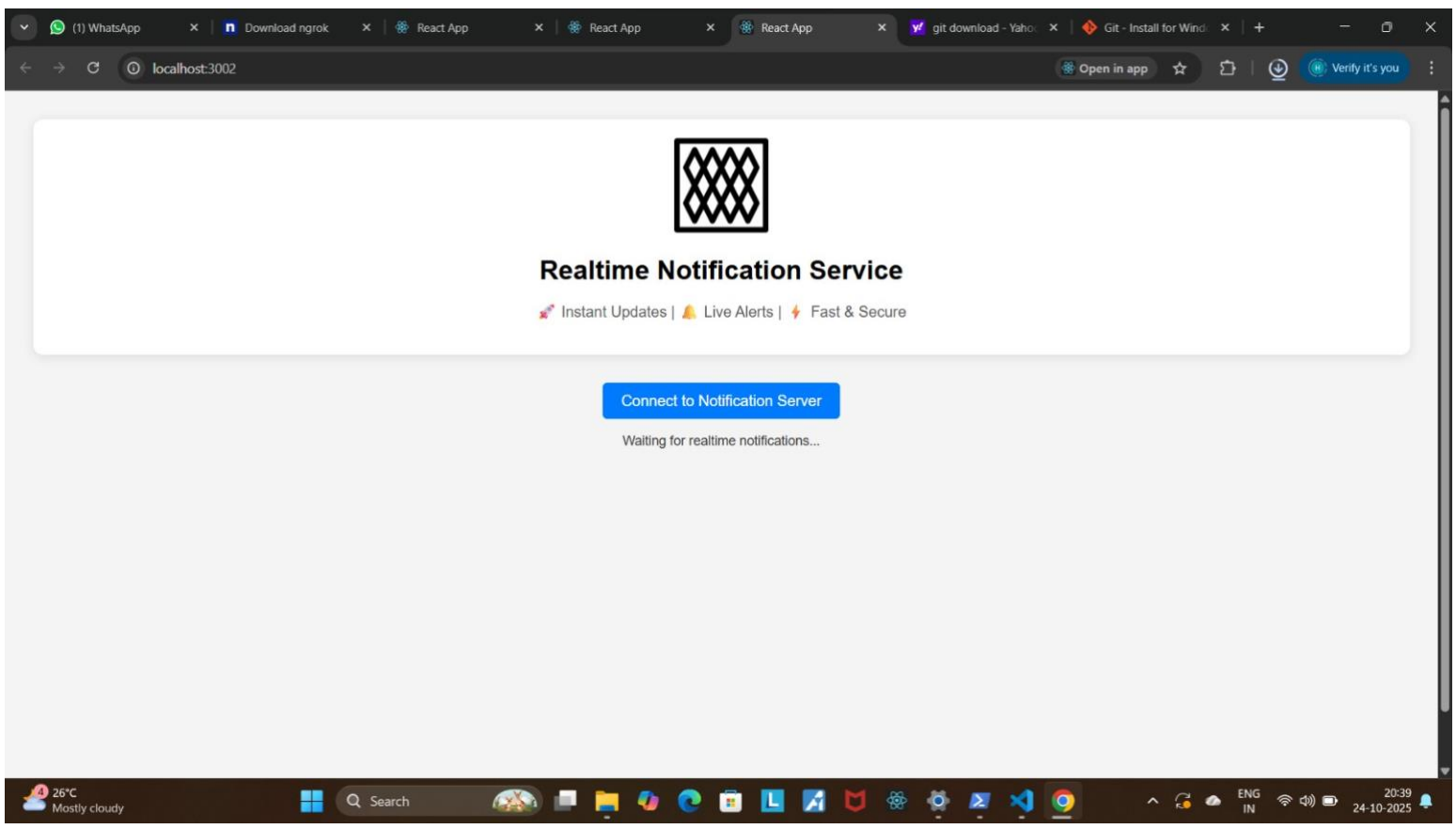
Testing Approach:

- Unit testing for API responses
- Functional testing for message flow
- UI testing for real-time updates

Hosting / Deployment Strategy:

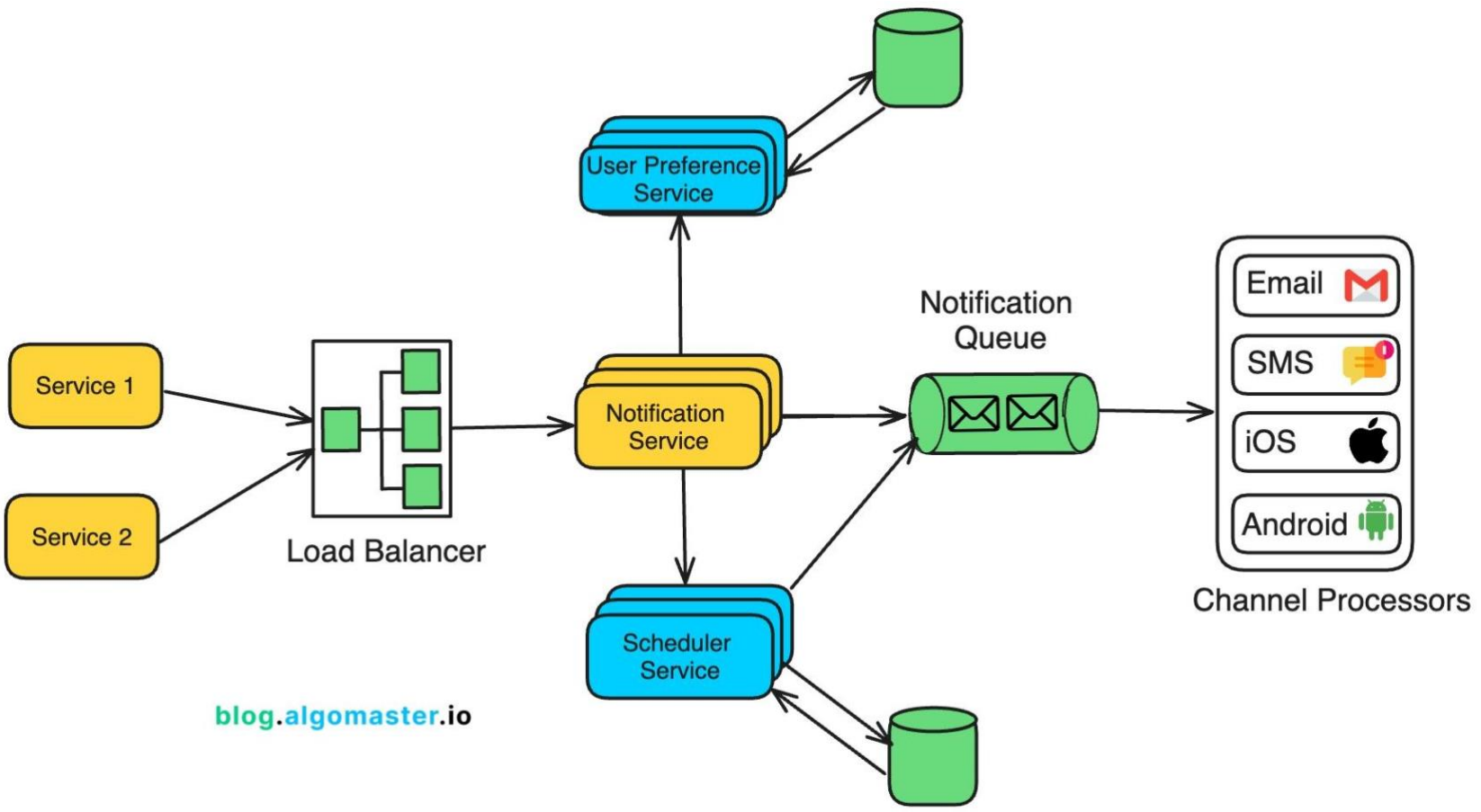
1. Use **Docker** containers for both backend and frontend.
2. Deploy via **IBM Cloud Kubernetes Service** or AWS ECS.
3. Set up **CI/CD pipeline** with GitHub Actions.
4. Enable monitoring using **Prometheus/Grafana** or **IBM Observability**.

DEPLOYMENT IMAGE:



DEPLOYMENT LINK:https://github.com/harithasenthil70/react

WOKFLOW IMAGE:



Conclusion:

“The Real-Time Notification Service project addresses a critical need in modern applications for instantaneous communication between systems and end-users”. By implementing WebSocket-based real-time communication, the system eliminates the delays and inefficiencies of traditional polling mechanisms, ensuring that users receive notifications immediately as events occur. This capability significantly enhances user engagement, satisfaction, and responsiveness, which are essential in applications such as e-commerce platforms, collaborative tools, messaging apps, and IoT-based monitoring systems.

From a technical perspective, the project demonstrates the seamless integration of multiple technologies including **“Node.js, Socket.io, MongoDB, and Redis, forming a robust, scalable, and fault-tolerant backend”.** On the front-end, the use of React.js combined with state management tools like Redux allows dynamic updates of the user interface without page reloads, creating a smooth and intuitive user experience. The inclusion of multi-channel support — in-app notifications, email, SMS — ensures that users remain informed regardless of their platform, further increasing the system’s versatility and value.

Moreover, by incorporating security measures such as **JWT-based authentication, encrypted communications, and role-based access control, the system not only delivers performance but also ensures that sensitive information remains protected.** Scalability has been considered throughout the architecture, with message queues, caching strategies, and cloud deployment allowing the service to handle high volumes of simultaneous users and notifications efficiently.

In essence, this project is not merely a technical exercise but a practical solution with real-world applicability. It lays a strong foundation for future enhancements such as AI-driven notification prioritization, analytics-based insights on user engagement, and cross-platform integration. The Real-Time Notification Service stands as an example of how modern software engineering principles can converge to produce systems that are **fast, reliable, secure, and user-centric, fulfilling the expectations of today's connected digital landscape.**