

# Sarthi — Software Design Document (SDD)

**Project Title:** Sarthi — IIITG Campus Connect

**Author(s):** M Shweta Singha - Software Architect (2521520)

Jalaj Gupta - Team Lead (2521513)

Meghna Hazari - Software Analyst (2521522)

Arunabha Dutta - Software Developer (2521506)

Hillol Pratim Kalita - Software Tester (2521511)

**Submitted to:** Dr. Nilotpal Chakraborty

**Date:** 24 November 2025



*Indian Institute of Information Technology Guwahati*  
Department of Computer Science and Engineering

# 1 Introduction

## 1.1 Purpose

This Software Design Document (SDD) provides a complete architectural and component-level design for the Sarathi web application. It ensures that the design aligns with the Software Requirements Specification (SRS) and guides developers, testers, and stakeholders through implementation details.

## 1.2 Scope

**Sarathi** is a campus-exclusive platform for IIITG students, faculty, and admins staffs, providing modules for Marketplace, Ride-Sharing, Lost/Found, and planned future enhancements like Vendor Marketplace and Rental Listings. System access is restricted to IIITG domain emails for safety and authenticity.

## 1.3 Audience

This document is intended for:

- **Developers** — for architecture and module implementation.
- **Testers** — to derive high-quality test cases.
- **Project Supervisors & Stakeholders** — to verify SRS compliance.

## 1.4 References

- Software Requirements Specification (Group 4, Version 2.0)

# 2 System Overview / High-Level Description

The **Sarathi** system serves as a multi-module platform for students to buy/sell items and share rides securely within the IIITG community.

## Main Features

- Google OAuth 2.0 authentication restricted to IIITG domain.
- Marketplace module.
- Ride-sharing module
- Lost/Found module
- Real-time messaging (Socket.IO).

## Users

- Students
- Faculty
- Staff

## Environment

- **Frontend:** React.js
- **Backend:** Node.js + Express.js
- **Databases:**
  - **MarketplaceDB** (stores user profile)
  - **RideShareDB** (stores user profile)
- **Messaging:** Socket.IO
- **Auth:** Google OAuth 2.0 → JWT Token
- **Hosting:** Render + Docker

## High-Level Architecture

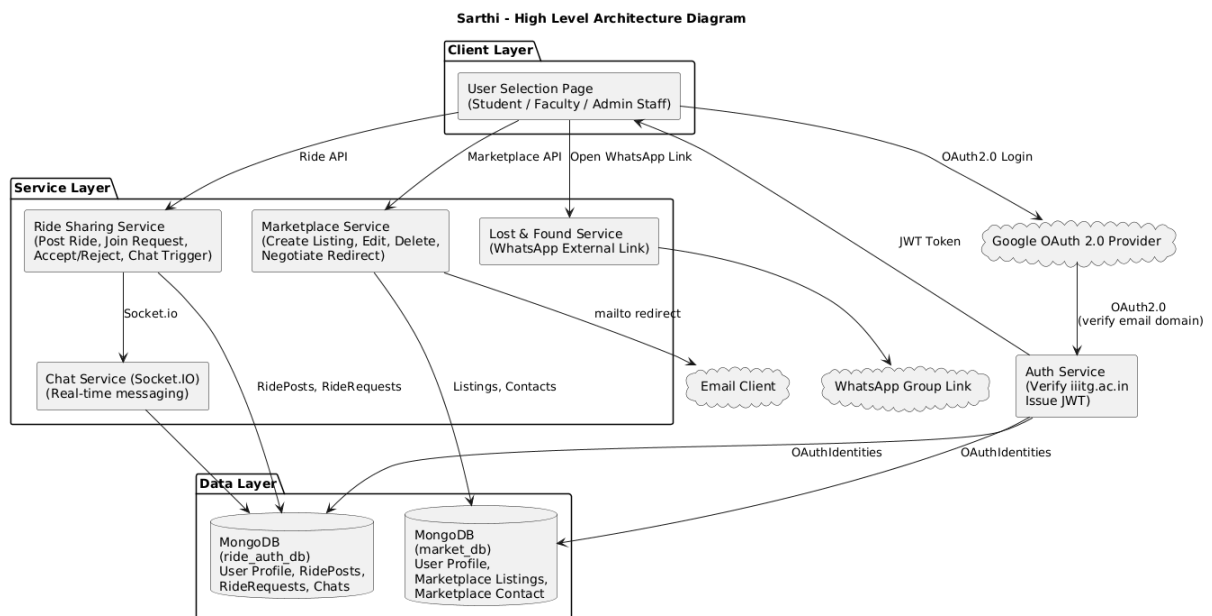


Figure 1: High-Level Architecture

## 3 Architecture Design

### 3.1 Architecture Pattern: Microservices

Sarathi follows a **Microservices Architecture**, where each module runs as an independent service communicating through REST APIs.

**Justification:**

- Independent scaling for Marketplace, Ride Sharing, Messaging, etc.
- Fault isolation—failure of one service does not affect others.
- Fast, parallel development aligned with Agile methodology.
- Simplifies continuous deployment.

### 3.2 System Components and Interactions

- **Frontend (Client):** Handles user interactions and UI rendering. Communicates via REST APIs.
- **Backend (Server):** Processes business logic, authentication, and data access.
- **Database:** Stores persistent entities (Users, Listings, Trips, Messages).

### 3.3 Development Methodology (Agile Model)

Sarathi uses the **Agile SDLC Model** to support iterative and flexible development.

### 3.4 UML Diagrams

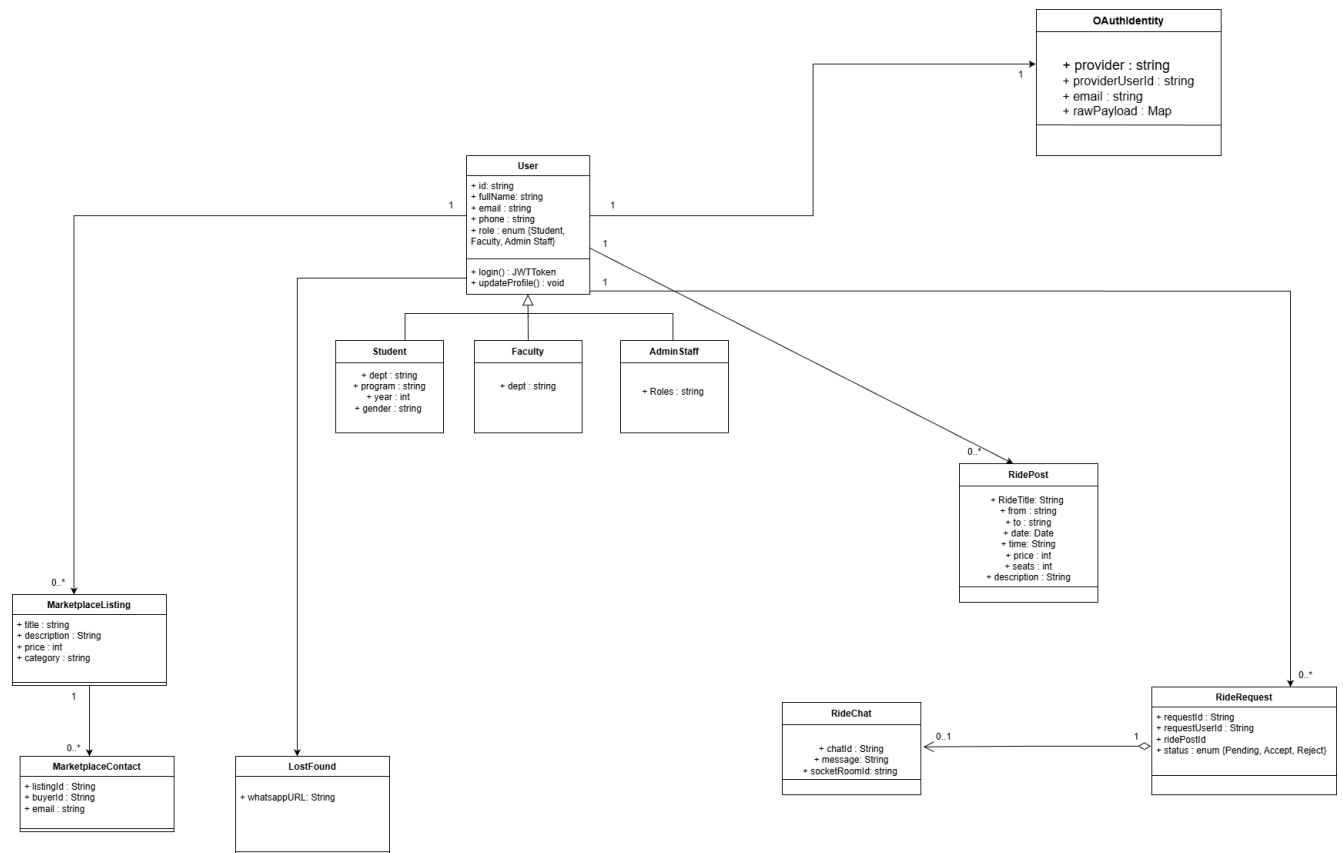


Figure 2: Class Diagram

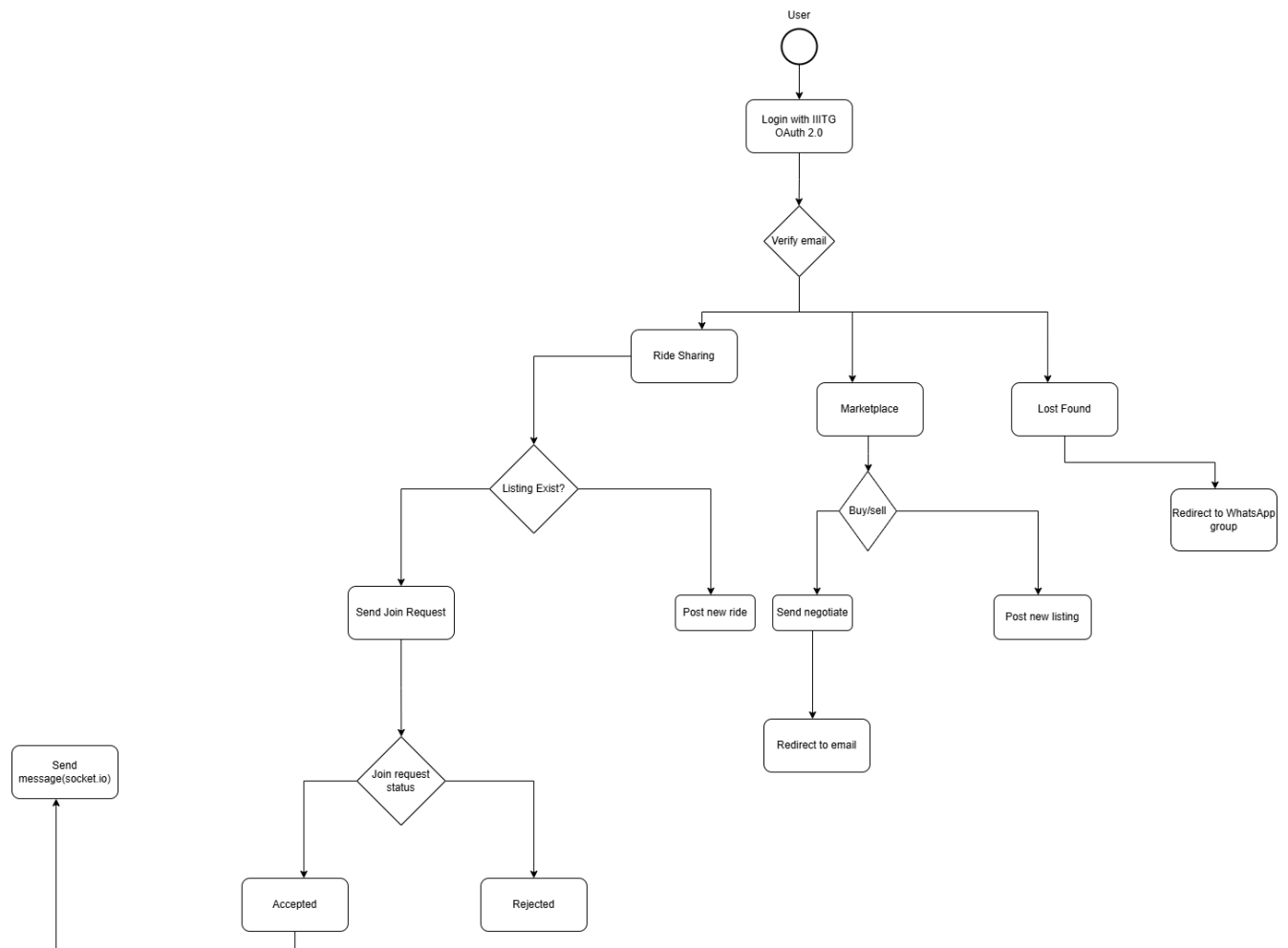


Figure 3: Activity Diagram

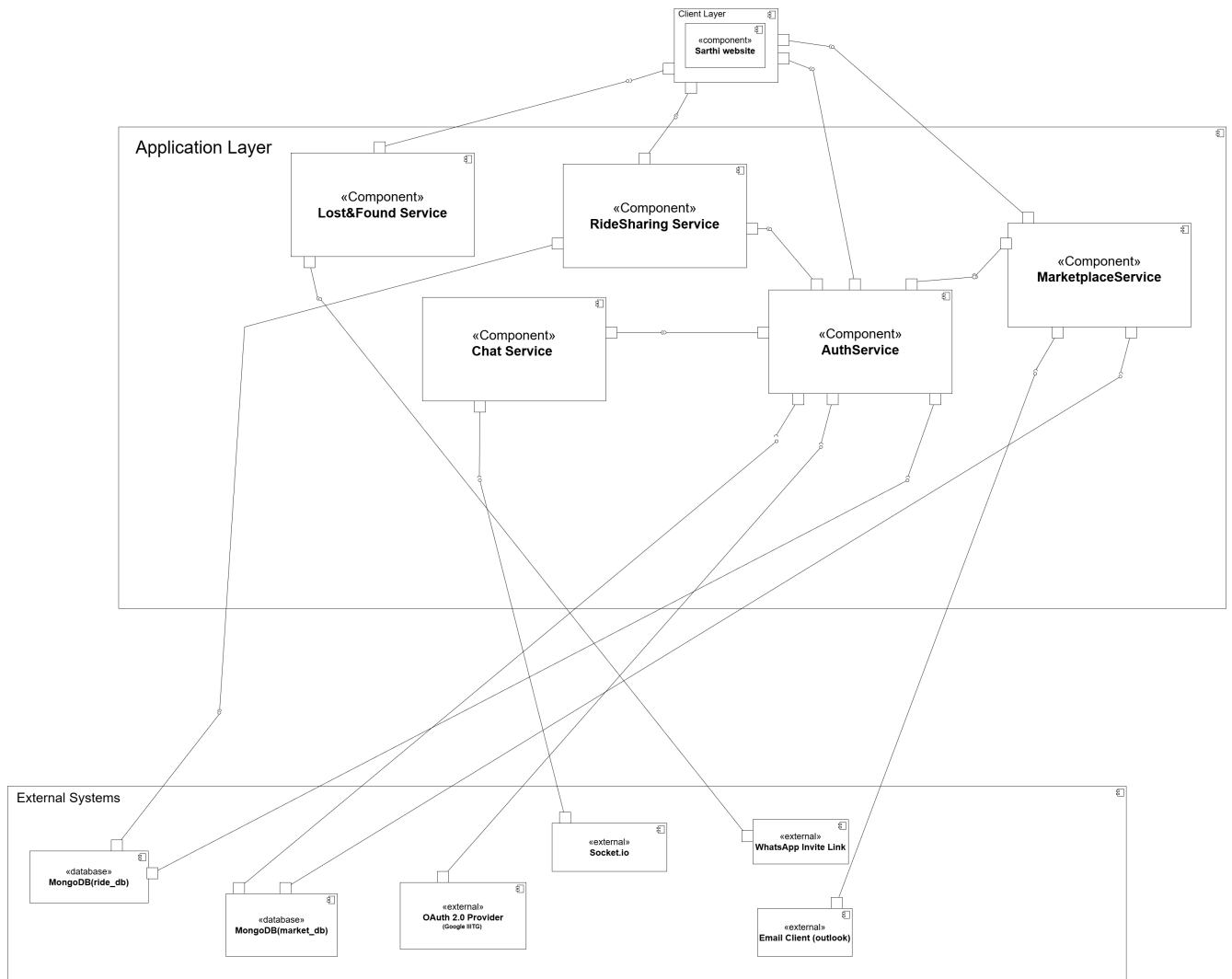


Figure 4: Component Diagram

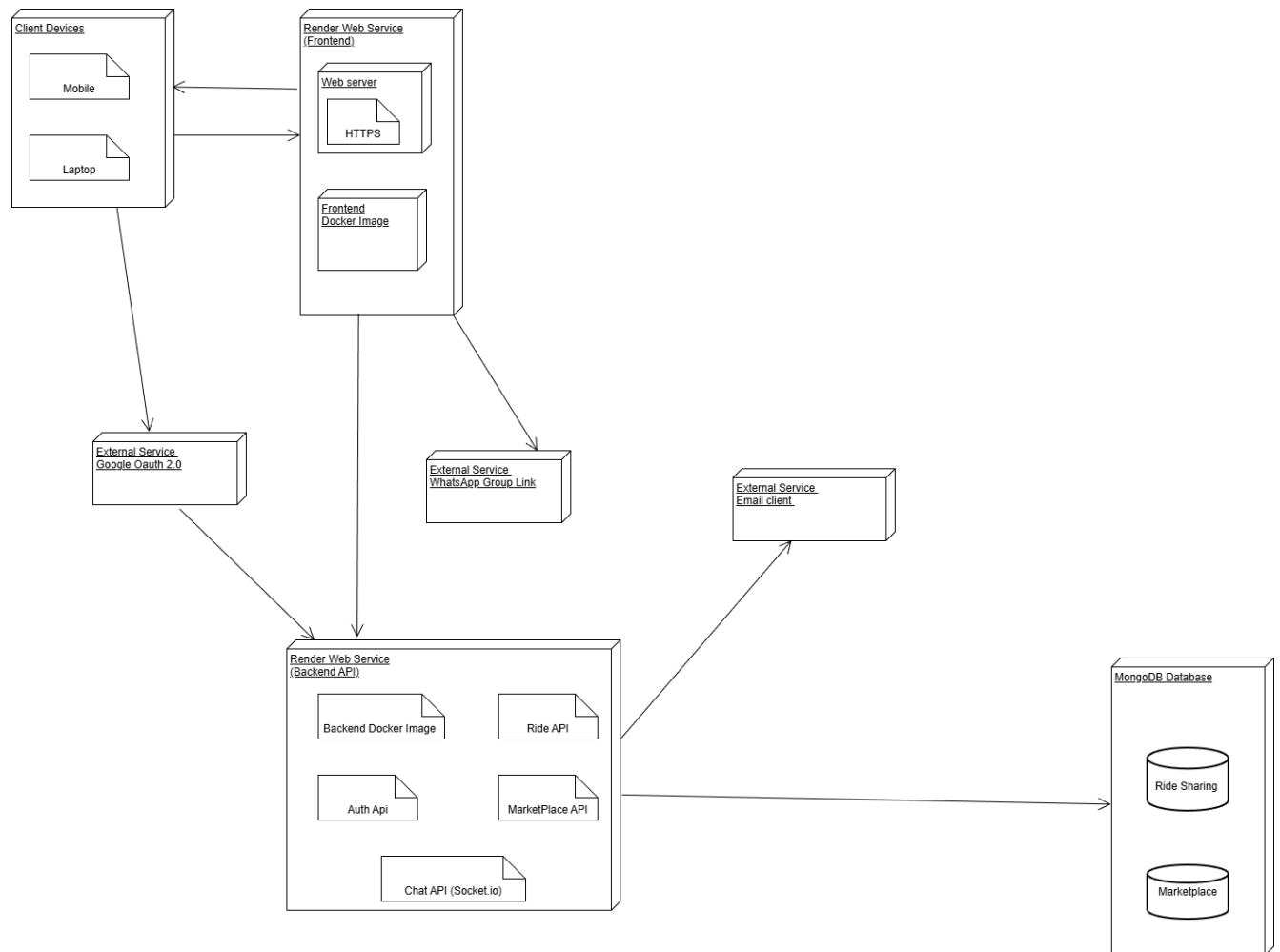


Figure 5: Deployment Diagram

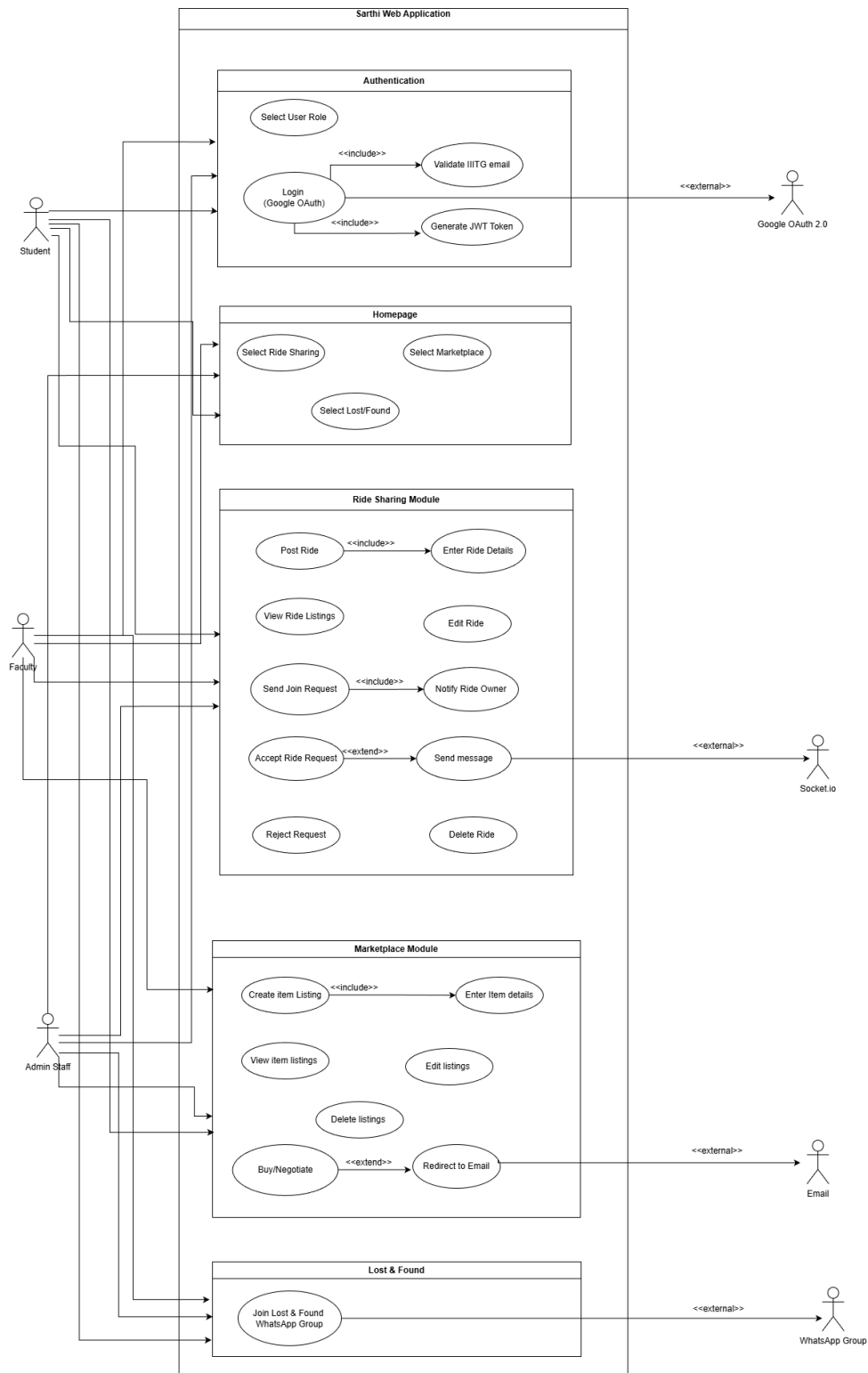


Figure 6: Use Case Diagram

## 4 Detailed Component Design

### 4.1 Authentication Module

Google OAuth 2.0 → JWT Process:

- User logs in using Google OAuth (restricted to @iiitg.ac.in).
- Backend verifies ID token.
- Backend issues a signed JWT token.
- All services validate JWT for authorization.

## 4.2 Marketplace Module

Handles product listings, updates, and deletions.

## 4.3 Ride-Sharing Module

Stores and manages trip details, join requests, and approval workflows.

## 4.4 Messaging Module (Socket.IO)

- Live chat between users.
- WebSockets ensure real-time communication.
- Messages encrypted in transit.

# 5 Security and Privacy Considerations

- **Authentication:** OAuth2.0 with Google domain verification and JWT Authorization
- **Authorization:** Role-based access control (RBAC) for student, faculty, admin staff.
- **Data Protection:** Input sanitization, HTTPS-only communication.
- **Privacy:** Hide personal contact info unless shared voluntarily.
- **CIA:** Only authorized users access messages (Confidentiality), request flow can't be tampered (Integrity), and the chat becomes available only when valid (Availability).

# 6 Testing Strategy

## 6.1 Unit Testing

**Framework:** Jest / Mocha **Scope:** Controllers, Services, and Utils

## 6.2 Integration Testing

Test end-to-end API flows (Auth → Listing → Trip) using mock DBs (MongoMemory-Server).

## 6.3 System Testing

Full-stack testing of web interactions using Cypress or Playwright.

## 6.4 Test Data & Environment

- Pre-seeded mock data for users, listings, and trips.
- Separate `.env.test` configuration.

# 7 Deployment

## 7.1 Render Deployment

- Frontend deployed as Render Static Site.
- Backend deployed as Render Web Service (Docker).
- MongoDB Atlas.

## 7.2 Docker Deployment

- Entire system containerized using Docker images.
- Services run using:

```
docker-compose up --build
```

## 7.3 Localhost Setup

### Prerequisites

- Node.js, npm
- MongoDB installed locally or via Atlas

### Steps

1. Clone repository: `git clone <repo>`
2. Install dependencies: `npm install`
3. Set environment variables in `.env`
4. Start backend: `npm run dev`
5. Start frontend: `npm start`
6. Visit `http://localhost:3000`

## 7.4 Containerized Setup (Optional)

Use `docker-compose up` to run backend, frontend, and MongoDB.

— End of Document —