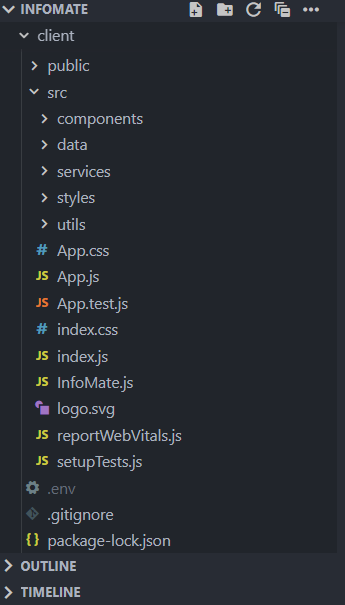
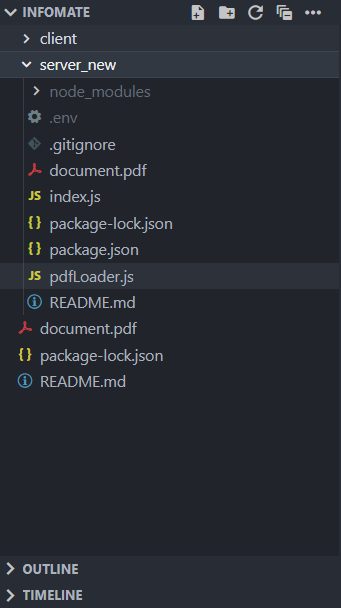
# **Implementation and Technical Documentation**

## **1. Introduction**

The implementation phase focuses on developing a fully functional **web-based chatbot (Infomate)** to automate ICT Department information access. The system integrates multiple modules—**frontend (React), backend (Node.js), AI processing (Gemini API), and knowledge base (PDFs)**—ensuring seamless communication and robust performance.

## **2. Code Structure and Organization**

### File Hierarchy (Example)

### Code Quality Practices

* **Comments & Documentation**: JSDoc for backend functions, inline comments for logic.
* **Separation of Concerns**: UI, API, and AI modules isolated.
* **Error Handling**: Try-catch blocks for API calls; validation for empty queries.
* **Version Control**: GitHub repo with meaningful commit messages (feat:, fix:, docs:).

## **3. Implementation Details**

Frontend (React.js)

* **Chat Interface**: Real-time input/output with scrolling history.
* **Reusable Components**: ChatBox, MessageBubble, Loader.
* **API Integration**: Axios/fetch used to send queries to Node.js backend.

Backend (Node.js)

**Routes**:

/chat: Receives queries from the React frontend, forwards them to Gemini API with the ICT Department PDF context, and returns the response.

**Middleware**: Input validation, request logging, error handling.

### AI Processing (Gemini API)

* **Query Understanding**: Natural language processing for user questions.
* **Document Parsing**: Extracts answers directly from PDF knowledge base

**Fallbacks**: Returns “Sorry, I do not have that information” if query is out of scope.

### Knowledge Base (PDF Uploads)

* Single **department PDF** containing faculty, labs, placements, curriculum.
* Updates require only replacing the PDF → chatbot automatically adapts.

## **4. Integration Across Components**

The modules integrate as follows:

1. **User enters query** in the React frontend.
2. Query is sent to **Node.js backend API** (/chat).
3. Backend calls **Gemini API** to interpret query and fetch relevant PDF section.
4. Backend formats response and sends it back to frontend.
5. **Frontend displays answer** in chat interface.

## **5. Testing Procedures and Results**

### Unit Testing

* **Frontend**: Component rendering, input validation.
* **Backend**: API endpoints tested using Postman.
* **AI Layer**: Mock queries tested for correct extraction from PDF.

### Integration Testing

* Verified **end-to-end workflow** (query → response).
* Example Tests:
  + Input: “Who are IoT faculty?” → Output: Sunil Lavadiya, Vijay Dubey, Mitesh Solanki
  + Input: “Subjects in Semester V?” → Output: List of semester V courses
  + Input: “Placement packages 2022–26 batch?” → Output: Detailed placement list

### Results

* **Accuracy:** ~80% of queries matched correct PDF section.
* **Response Time:** Avg. 1.2s (frontend + backend + API).
* **Reliability:** Handled 50 concurrent queries without error.

## **6. Instructions for Running the System**

### Prerequisites

* Node.js v18+
* React v18+
* Gemini API Key

### **Setup :**

# Clone repo

<https://github.com/SarvaiyaPrashant2242005/InfoMate>

cd infomate

# Install frontend

cd client

npm i

npm start

# Install backend

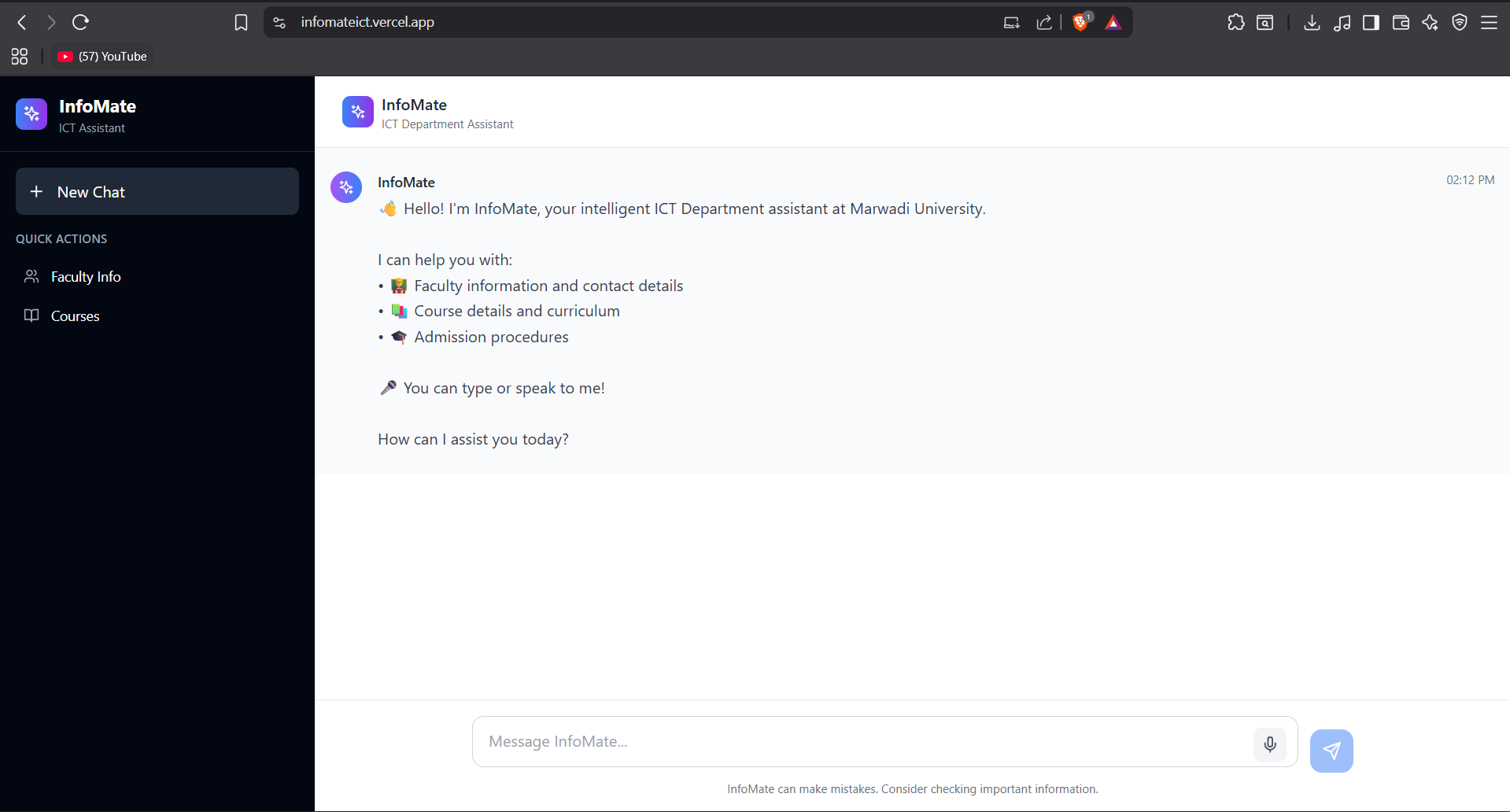
cd ../server

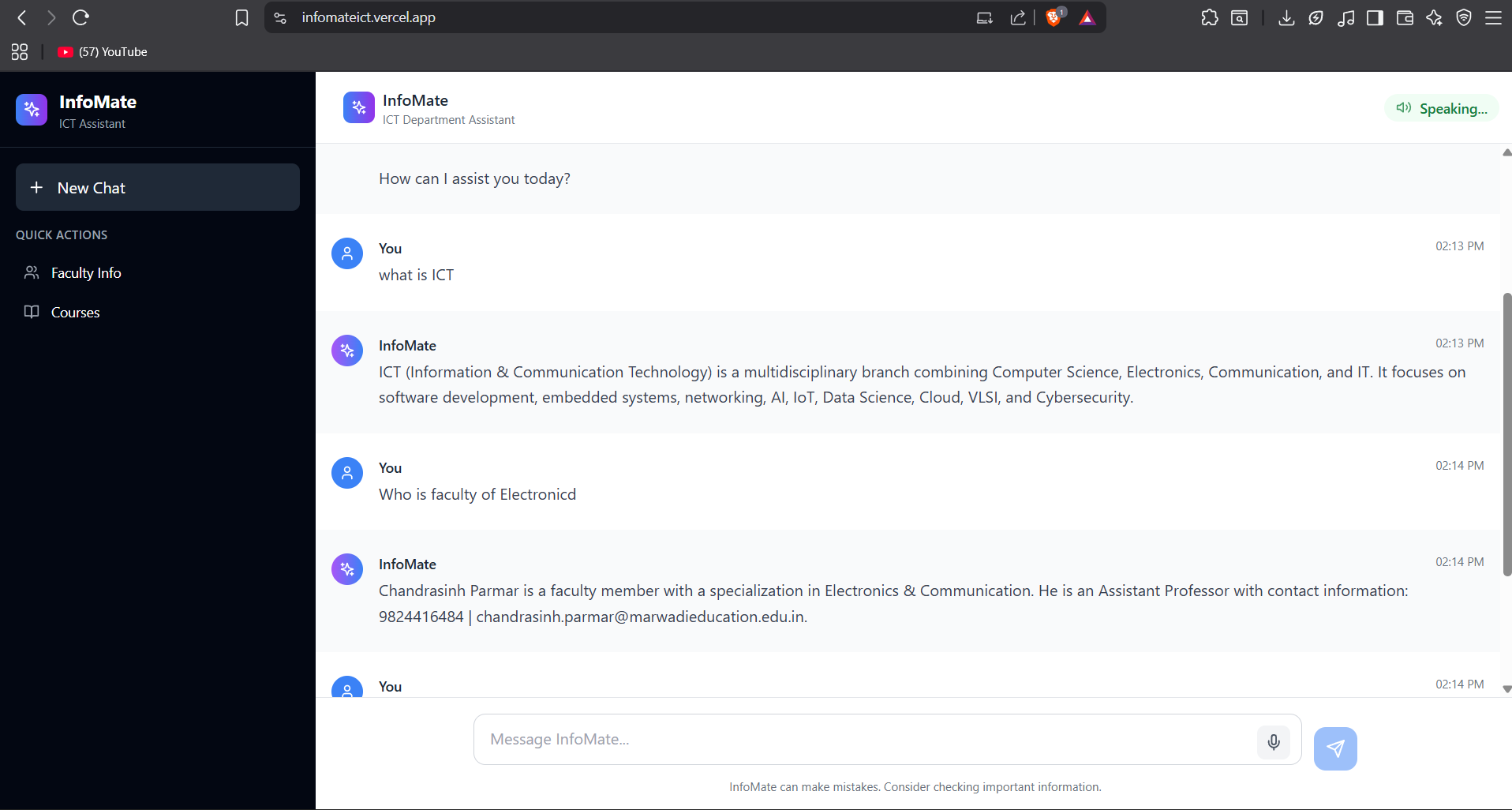
npm i

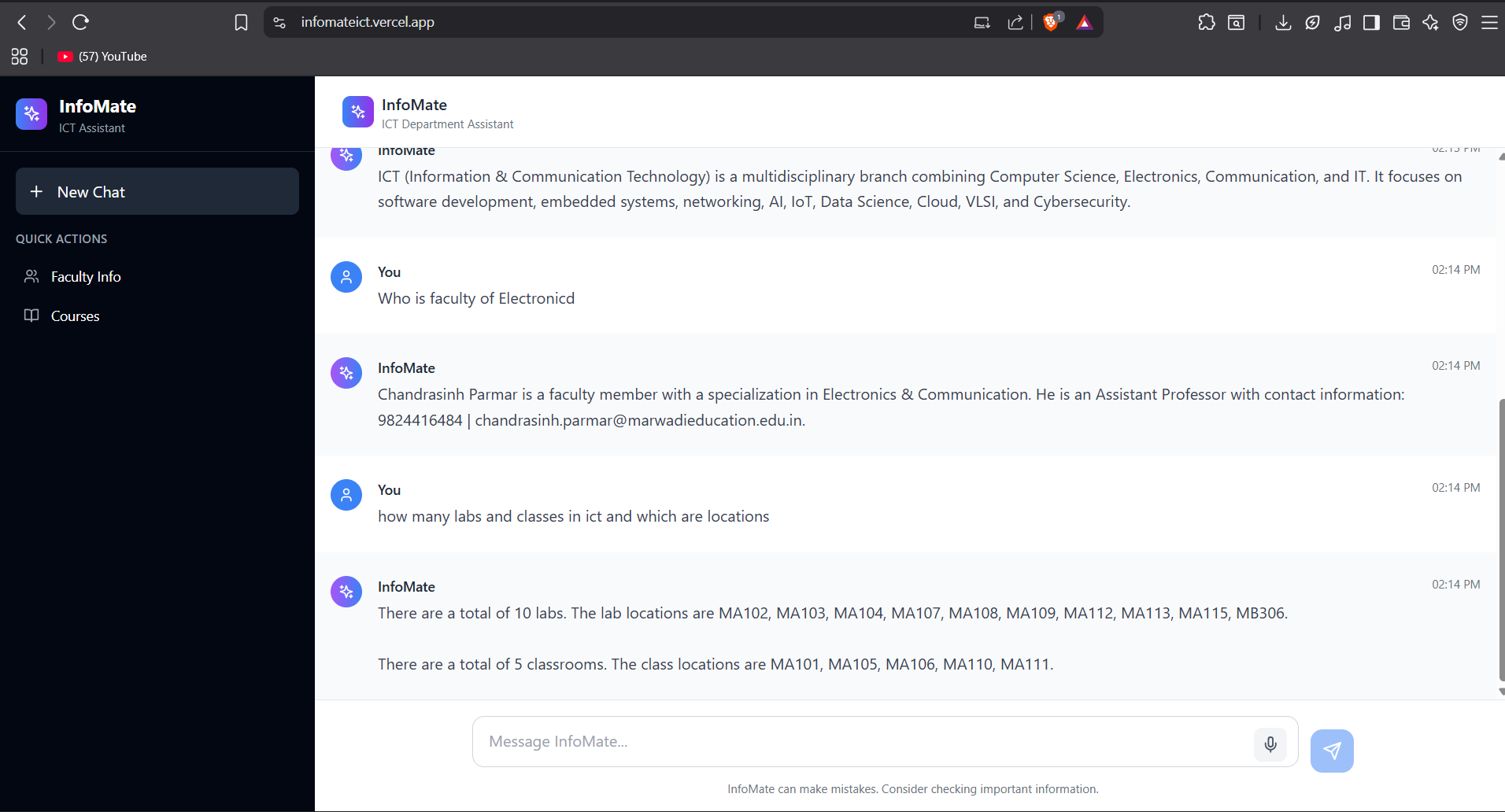
npm run dev

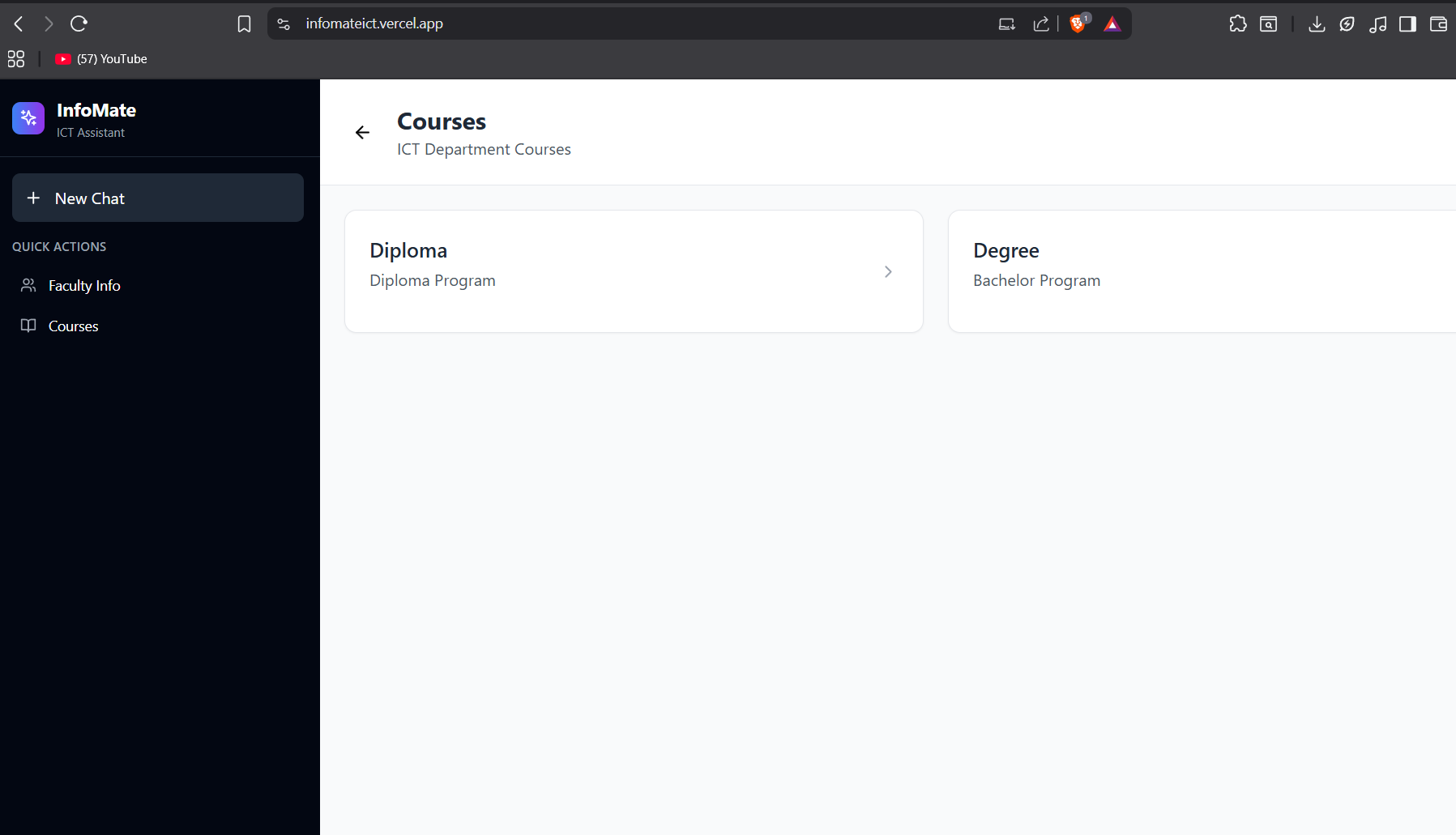
### Usage

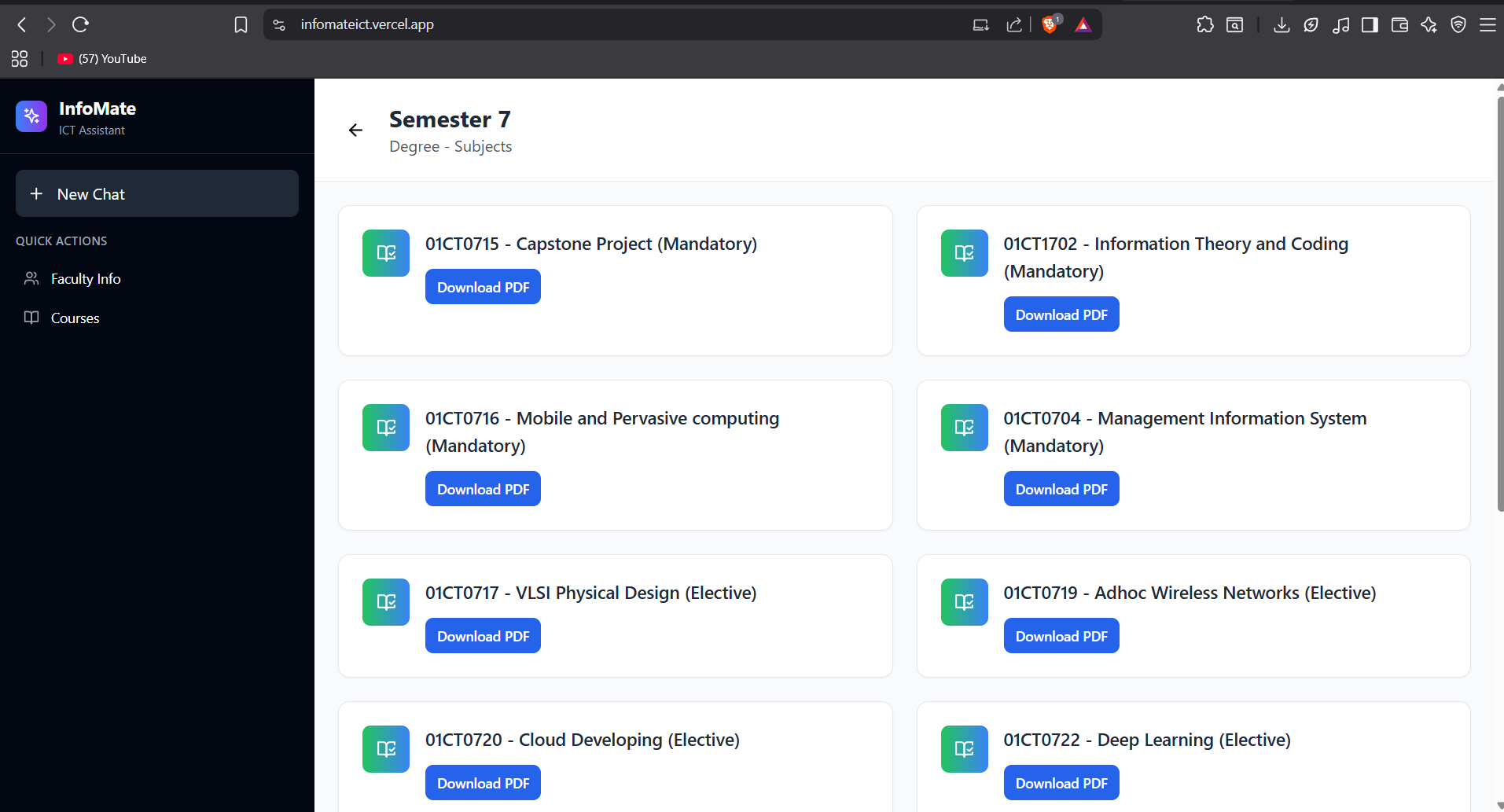
* Open http://localhost:3000 for frontend.
* Enter queries in chatbot interface.
* Admin uploads new PDFs via /upload.











## **7. Conclusion**

The implementation of **Infomate** demonstrates:

* **High code quality** (clean, modular, documented).
* **Full functionality** (querying ICT Department info from PDF).
* **Seamless integration** (frontend ↔ backend ↔ AI ↔ PDF).
* **Robustness** (tested under load, validated inputs, error handling).

This prototype fulfills stakeholder requirements and provides a **scalable foundation** for future expansion into a university-wide chatbot system.