PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

# Online Chatbot Based Ticketing System

## A PROJECT REPORT

***Submitted by***

SPANDANA KG-20221CSE0359

MISBA AIN- 20221CSE0256

M PRIYANKA-20221CSE0274

*Under the guidance of,*

**Dr. Afroj Alam**

## BACHELOR OF TECHNOLOGY

**IN**

## COMPUTER SCIENCE AND ENGINEERING

**PRESIDENCY UNIVERSITY**

**BENGALURU**

**DECEMBER 2025**

PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

# PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

## BONAFIDE CERTIFICATE

Certified that this report "Online Chatbot Based Ticketing System" is a Bonafide work of "SPANDANA KG (20221CSE0359), MISBA AIN (20221CSE00256), M PRIYANKA (20221CSE0274)", who have successfully carried out the project work and submitted the report for partial fulfilment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE ENGINEERING, during 2025-26.

**Dr. Afroj Alam**
Project Guide
PSCS
Presidency University

**Dr. Jayavadivel Ravi**
Program Project Coordinator
PSCS
Presidency University

**Dr. Sampath A K**
**Dr. Geetha A**
School Project Coordinators
PSCS
Presidency University

**Dr. Asif Mohamed**
Head of the Department
PSCS
Presidency University

**Dr. Shakkeera L**
Associate Dean
PSCS
Presidency University

**Dr. Duraipandian N**
Dean
PSCS & PSIS
Presidency University

**Examiners**

| Sl. no. | Name | Signature | Date |
|---|---|---|---|
| 1 | Dr. Trimoorthy N | | |
| 2 | Dr. Suvanam Sasidhar Babu | | |

# PRESIDENCY UNIVERSITY

# PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

# DECLARATION

We the students of final year B. Tech in COMPUTER SCIENCE ENGINEERING, at Presidency University, Bengaluru, named, hereby declare that the project work titled **"**Chatbot-driven Ticket Management System**"** has been independently carried out by us and submitted in partial fulfillment for the award of the degree of B.Tech in COMPUTER SCIENCE ENGINEERING during the academic year of 2025-26. Further, the matter embodied in the project has not been submitted previously by anybody for the award of any Degree or Diploma to any other institution.

SPANDANA KG USN: 20221CSE0359 ----------

MISBA AIN USN: 20221CSE0256 ----------

M PRIYANKA USN: 20221CSE0274 ----------

PLACE: BENGALURU

DATE: 26$^{TH}$ SEPTEMBER 2025

# ACKNOWLEDGEMENT

For completing this project work, We/I have received the support and the guidance from many people whom I would like to mention with deep sense of gratitude and indebtedness. We extend our gratitude to our beloved **Chancellor, Pro-Vice Chancellor, and Registrar** for their support and encouragement in completion of the project.

I would like to sincerely thank my internal guide **Dr. Afroj Alam, Professor**, Presidency School of Computer Science and Engineering, Presidency University, for his moral support, motivation, timely guidance and encouragement provided to us during the period of our project work.

I am also thankful to **Dr. Asif Mohamed H B, Head of the Department, Presidency School of Computer Science and Engineering** Presidency University, for his mentorship and encouragement.

We express our cordial thanks to **Dr. Duraipandian N**, Dean PSCS & PSIS, **Dr. Shakkeera L**, Associate Dean, Presidency School of computer Science and Engineering and the Management of Presidency University for providing the required facilities and intellectually stimulating environment that aided in the completion of my project work.

We are grateful to **Dr. Afroj Alam, and Dr. Jeevitha VK, PSCS** Project Coordinators**, Dr. Jayavadivel Ravi, Program Project Coordinator**, Presidency School of Computer Science and Engineering, or facilitating problem statements, coordinating reviews, monitoring progress, and providing their valuable support and guidance.

We are also grateful to Teaching and Non-Teaching staff of Presidency School of Computer Science and Engineering and also staff from other departments who have extended their valuable help and cooperation.

SPANDANA KG

MISBA AIN

M PRIYANKA

# Abstract

With the expansion of digital channels, the need of providing customer assistance has evolved dramatically in organizations. Traditional support systems are usually energy-demanding as they call for the engagement of a large number of human operators for repetitive tasks that lower efficiency, resolution speed and increase costs. The progress in Artificial Intelligence (AI) and Natural Language Processing (NLP) has led to the rise of chatbots as the most preferred solution to automate the first contact with the customer. Yet, the majority of chatbot platforms are incapable to deal with complicated inquiries and provide a well-organized escalation method and that is why they make customers unhappy. The current initiative entitled Chatbot Ticket Management System, designed as a concerted move, resolves these issues by leveraging the capabilities of the chatbot user interface and ticket management workflow.

The proposed system would function as a web-based service to which customers can send their questions and receive the answers immediately from a chatbot. If the chatbot cannot provide a solution for the given question, the system will create a support ticket, which will be recorded in the database. The ticket is thus a means of communication between support staff and users through the admin dashboard as they can monitor, update, and complete the ticket. Updates about the status of their support tickets are communicated to the users thus ensuring that transparency and accountability are some of the benefits that the support process gains from them. Such a collaborative model not only limited human agents' engagement to essential tasks but also, they were assured that any overlooked problems had not existed.

The inception of this project has synced itself with Agile Scrum work philosophy seven stages. Each period of work project assorted them into a functional module like the chatbot interface, ticket management module, and management dashboard. To achieve a performance boost, the front end was developed with Next.js, a feature-rich CSS framework named Tailwind was used to create a modern and responsive style, and a flexible hybrid language, TypeScript for type-safety and maintainability. On the other hand, backend APIs were handled with Node.js, whereas Firebase handled the dataset for ticket storage and user management. Moreover, there were also some other essential tools such as Postman (API testing), GitHub (version control), and Vercel (deployment) for good developers' execution and cooperation.

The system underwent several tests such as functional testing, usability testing, and performance analysis. Results demonstrated that the chatbot could handle frequently asked questions effectively while the unresolved queries were escalated to tickets successfully. It was

found that the implementation led to a noticeable shorter average response time and consequently, a much better customer experience as it was ensured that no question was left unanswered. Risk analysis also pointed out that in a case where issues such as chatbot misinterpretation and security vulnerabilities arise, the fallback rules, authentication mechanisms, and secure database practices that have been put in place will come to the rescue.

This is a project that matters to most people because it solves the problem of providing a cheap, easy-to-customize, and scalable customer-support automation system. Compared with enterprise-grade products like Zendesk or IBM Watson, which are expensive to subscribe to and require sophisticated infrastructures, this system is very light and thus, academic institutions and industries can equally benefit from it. Besides, the initiative touches upon various United Nations Sustainable Development Goals (SDGs) especially SDG 9 (Industry, Innovation, and Infrastructure), SDG 8 (Decent Work and Economic Growth), and SDG 10 (Reduced Inequalities) as by encouraging innovation it also helps productivity to rise and moreover, the accessibility of support is equally provided for all.

Summing up the Chatbot Ticket Management System is an essential move towards customer support enhancement by linking intelligent automation with organized ticket workflows.

# Table of Content

# List of Figures

# List of Tables

# Abbreviations

| **Abbreviation** | **Full Form** |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| CI/CD | Continuous Integration / Continuous Deployment |
| CRM | Customer Relationship Management |
| CSS | Cascading Style Sheets |
| DB | Database |
| DPDPA | Digital Personal Data Protection Act |
| FAQ | Frequently Asked Questions |
| GDPR | General Data Protection Regulation |
| HCI | Human–Computer Interaction |
| HTTPS | Hypertext Transfer Protocol Secure |
| IEEE | Institute of Electrical and Electronics Engineers |
| IoT | Internet of Things |
| IoTWF | Internet of Things World Forum |
| JSON | JavaScript Object Notation |
| LLM | Large Language Model |
| MDN | Mozilla Developer Network |
| MeitY | Ministry of Electronics and Information Technology |
| MQTT | Message Queuing Telemetry Transport |
| NFR | Non-Functional Requirement |
| NLP | Natural Language Processing |
| PCI-DSS | Payment Card Industry – Data Security Standard |
| PDF | Portable Document Format |
| PII | Personally Identifiable Information |
| PoC | Proof of Concept |
| PSCS | Presidency School of Computer Science |

| | |
|---|---|
| PSIS | Presidency School of Information Science |
| QR | Quick Response |
| SDG | Sustainable Development Goal |
| SPA | Single Page Application |
| TLS | Transport Layer Security |
| UI | User Interface |
| VM | Virtual Machine |
| WCAG | Web Content Accessibility Guidelines |

# Chapter 1

# Introduction

## 1.1 Background

Customer service is possibly the single factor that can make or break a company in the highly volatile digital world that we live in today. In addition, customers want their issues to be resolved in the shortest time possible, and the way has to be effective. Moreover, whether these are technical problems, service requests, or just questions. Traditional support usually involves a lot of human agents, and for that reason, it faces issues such as limited service hours, increased operational costs, long waiting times, and uneven quality of service.

Meanwhile, companies have turned to chatbots as a remedy to such hurdles. Basically, a chatbot is a software agent that is powered by artificial intelligence and hence can respond to customer inquiries automatically as well as resolve issues that customers bring up. Typically, chatbots are designed to be very efficient in activities that are repetitive; on the other hand, in complex situations, they are often required to pass such cases to the human side. This flaw has led to the development of hybrid systems that combine the chatbot conversations and the ticketing systems in one.

Ticket management software is a crucial piece of equipment in customer support that facilitates recording, assigning, tracking, and resolving issues brought to you by the users. A chatbot integrated with a ticketing system can answer FAQs instantly from the users; those questions that are complicated and cannot be solved by the chatbot can be escalated as tickets with all the necessary details to ensure the intervention of a human agent.

The team through this Chatbot Ticket Management System project is committed to building not only an intelligent and user-friendly platform but also an efficient one that, besides utilizing chatbot technology, facilitates the ticketing workflow. The team has chosen Next.js for the

frontend, TailwindCSS for the styling, and TypeScript for the code development. The combination of these tools is thus a guarantee for the app to be scalable, responsive, and offering a modern user experience.

## 1.2 Statistics of Project

The numbers coming out of our dashboard simulation give an impression of a pretty lively scene. Right now, the system is monitoring total revenue that is flirting with the $45,230 value. Our ticket count has reached about 2,350, which is a great indication of a very strong upward trend by comparison to the previous flow of metrics.

One of the patterns we found in the data is that 2:00 PM appears to be the "golden hour" for bookings; that is the time when user activity is at its highest. We also find out which performances are attracting the most people to facilitate the organizing process. As an example, the "Cosmos" Special Exhibition is doing nearly twice as many standard general admission as it alone is selling over 1,200 tickets. This sort of data is extremely valuable as it provides us with the exact information of what people want and at what time they are going online to book.

## 1.3 Prior Existing Technologies

Many different technologies and platforms with distinct features have been developed to assist customers in solving their problems:

- Zendesk: An excellent helpdesk software with advanced ticketing features but requires a paid subscription.
- Freshdesk: Uses AI to enhance customer service, but the product is too complicated for small businesses.
- Intercom: A customer communication platform with the integration of the chatbot but limited customization options.
- Open Source Chatbots (e.g., Rasa, Botpress): Provide complete customization but require a very high level of technical skills to be integrated with the ticketing systems.

Unlike them, this project offers a lightweight, customizable, and open-source-friendly solution that essentially propagates the benefits of chatbot automation to the structured ticket management.

## 1.4 Proposed Approach

We did not try to recreate the wheel for this to work. We only picked the parts that fit perfectly together. Next.js was the tool we used for the overall build as it manages the front-end stuff in a pretty way and keeps the site fast.

The logic to the operation—the part that actually understands what the user is typing—was powered by Google's Genkit. So the bot can be a general question answerer and, at the same time, a personal assistant in booking a flight without any confusion.

We were not willing to manage a huge server farm, therefore, we chose to use Firebase for the backend database and authentication. It is only doing the simple work of storing ticket orders and user info. The look is minimal and the designer has been assisted by Tailwind CSS and Shadcn UI so that it is trendy and works well on mobile devices.

In short, the plan was to make the user interface extremely simple while the code does the hard work behind the scenes to handle the conversational flow and payments.

## 1.5 Objectives

The primary goals of the project are:

1. To design and build a chatbot front-end which would be able to interact with users and provide them with answers to their inquiries immediately.

2. To implement ticket management features which would identify the queries that have not been addressed and assign them for resolution.

3. To create a fluid, fast, and user-friendly interface by using modern web technologies (Next.js, Tailwind CSS).

4. To increase the level of productivity by decreasing the response times and managing repetitive queries automatically.

5. To ensure system security by the provision of authentication and giving of access rights only to users and administrators.

6. To gain users' loyalty by offering them quicker and more reliable assistance.

## 1.6 Alignment with Sustainable Development Goals (SDGs)

This project aligns with different United Nations Sustainable Development Goals (SDGs):

- SDG 9 – Industry, Innovation, and Infrastructure: The customer service infrastructure is being changed by the program with the help of automation, thus, innovating the infrastructure.
- SDG 8 – Decent Work and Economic Growth: Workers will have the opportunity to focus on more intellectually stimulating and rewarding tasks since the removal of monotonous activities goes hand in hand with both an increase in productivity and workplace well-being.
- SDG 10 – Reduced Inequalities: Due to automated responses, the system is always there to provide equal and impartial assistance to any user without any breaks.
- SDG 12 – Responsible Consumption and Production: The best use of the workforce is resulting in the consumption of resources in a more efficient way.

## 1.7 Overview of the Project

The fundamental idea behind "Museum Buddy" was to identify a problem that is very common for people -long lines for buying a ticket-. In which case we would invent a way of doing something which would not be like a boring form to fill out but more like chatting with a helpful friend. So essentially, it is an online application with which you communicate with a bot to reserve museum tickets. You just tell it where you want to go – for example, the National Museum in Delhi or the Salar Jung Museum in Hyderabad– and it makes all

the arrangements for you. We put a large portion of our effort into making it accessible so that it could talk in several languages like Hindi, Bengali, and Tamil and thus solve the problem of the language barrier. To ensure absolute smoothness in the whole process, such as from choosing a date to receiving a digital ticket with a QR code directly on your phone, "Museum Buddy" is, in fact, somewhat of a high-tech device inside. The aim was to get rid of the dull administrative side and at the same time open up the cultural door to everyone.

# Chapter 2

# Literature Review

## 2.1 Introduction

Any research work cannot be imagined without a literature review, which serves as a backbone by showing the past works, pointing out the gaps, and providing the base for the new system. In this case, the review has extended to various topics such as chatbots, customer-support automation, ticket management systems, and AI-powered conversational interfaces to support the creation of a chatbot ticket management system.

The next section presents significant pieces of writing from journals, conferences, and whitepapers. Methodology, contributions, and limitations of the studies are considered in the review to locate the gap in research that the present work intends to fill.

## 2.2 Research Papers Reviewed

### [1] Shawar B.A., and Atwell, E. (2007) – "Chatbots: Are They Really Useful?"

This article reflects on the different uses of chatbots in the different domains. This article reflects on the different uses of chatbots in the different domains. The principal focus is on their ability to successfully automate responses to user queries. The article differentiates between rule-based bots and those based on machine learning and mentions problems like a limited understanding of the context. The research acknowledges that chatbots are unloading human workers, however, they still require help systems to handle complex requests, which, by means of ticket integration, is actually the way of this present project.

### [2] Adamopoulou, E., and Moussiades, L. (2020) – "An Overview of Chatbot Technology"

This exhaustive survey defines the division of chatbots into rule-based and AI-powered systems. The article also discusses the adoption of various platforms such as Rasa, Dialogflow,

and Microsoft Bot Framework. The significance of Natural Language Processing (NLP) and the need for the machine to remember the context for better interaction are pointed out. On the other hand, the publication also points to barriers in terms of cost and infrastructure for small businesses, thus it is being motivated to develop a lightweight and customizable system such as the one proposed here.

## [3] Hill, J., Ford, W., & Farreras, I. (2015) – "Real Conversations with Artificial Intelligence: A Comparison between Human–Human and Human–Chatbot Conversations"

This research explores the perceptions of users regarding chatbots in comparison with human agents. The principal reason cited by users for liking chatbots is the speed whereas the most common frustration is the inability of chatbots to comprehend complex queries. The article emphasizes the need for the involvement of human agents at a higher level (escalation mechanisms) – thus, integration of chatbot-interactions with ticket management, as done in this project, has been confirmed.

## [4] Reshmi, S., and Balakrishnan, K. (2017) – "Implementation of an Intelligent Chatbot for E-Learning Support"

The paper authors take an AI-powered chatbot developed to assist students in an e-learning platform as their research subject. The demonstration is made through the paper on how user engagement can be improved by chatbots but at the same time, the significance of database integration for query resolution is stressed. The lessons learned from this work can be seen in the database-backed ticketing system of our project.

## [5] Jain, M., Kumar, P., & Patel, S. (2018) – "Customer Support Automation Using Chatbots"

This writing is about a speech agent prototype which fosters the first step of customer support automation. By handling FAQs, the chatbot reduced the workload of agents; however, it did

not have a structured escalation for solving the issues that were left unsolved. The limitation puts forward the necessity of combining chatbot automation with ticket-based escalation, thus, our system is accomplishing that.

## [6] Xu, A., Liu, Z., Guo, Y., Sinha, V., & Akkiraju, R. (2017) – "A New Chatbot for Customer Service on Social Media"

The research is about the chatbots that are designed for such platforms as Twitter and Facebook Messenger. Among the benefits of the integration of the bots within the most widely used communication platforms, the paper mentions ease and speed of access to the users. Nonetheless, these types of chatbots are quite incapable of keeping track of the unresolved issues in a systematic way and thus, reinforcing the necessity of a ticket tracking mechanism as being implemented in our project.

## [7] Radziwill, N.M., and Benton, M.C. (2017) – "Evaluating Quality of Chatbots and Intelligent Conversational Agents"

This research offers a proposal on how to measure chatbot quality through categories like usability, intelligence, and security. In the proposal, it is stated that the trust of users is dependent mainly on the chatbot's reliability. The article serves as a tool to strengthen the point of accountability for such instruments as the generation of tickets, which is a way of ensuring that contacts not solved do not get lost.

## [8] Liao, Q.V., et al. (2018) – "All Work and No Play? Conversations with a Task-Oriented Chatbot in the Workplace"

The article is about studies on task-oriented chatbots used in companies. It provides evidence that the use of such bots is a good way to cut down on the repetitive communication, yet, they often come into stark contrast when faced with vagueness. The researchers suggest that the solution be a hybrid one, where the chatbots undertake the simple tasks while the human workers handle the difficult ones. This said hybrid model is the support of our system design.

## [9] Rasa Technologies (2021) – "Building Contextual AI Assistants" (Whitepaper)

Rasa's technical documents reveal the details of how to make AI assistants that can recall user conversations and therefore stay-contextual. While Rasa provides rather powerful NLP capabilities, its adoption is greatly limited by a steep learning curve and a requirement for a sizeable infrastructure, which is not very friendly for small projects. Our arrangement is such that we employ the rather lightweight technologies (Next.js, Tailwind, TypeScript) so as to still be approachable without the need for a large-scale infrastructure.

## [10] IBM Watson (2019) – "AI in Customer Care: Transforming Support with Chatbots"

This industry whitepaper is a demonstration of the ways in which IBM Watson can be a good fit for the customer support domain. It highlights a few such benefits as ease of scaling, sentiment analysis, and smooth integration with CRM systems. Still, these types of systems are usually quite pricey and of a closed nature. The place where the shortage appears is in terms of affordability and providing the users with options i.e., customizing in which our project's open-source way settles both of the issues.

## [11] Tariq, S., et al. (2021) – "A Comparative Analysis of Open Source Chatbot Frameworks"

The authors of this paper make a comparison of various open-source chatbot frameworks such as Botpress, Rasa, and ChatterBot. Their conclusion is that open-source tools are strong but at the same time, they demand a great deal of skill to be customized and connected with enterprise tools. This revelation stimulates the necessity for a simplified, developer-friendly platform, which, as per our proposal, is the direction we are taking.

### [12] Mishra, A., & Sharma, D. (2022) – "Integration of Ticketing Systems with Conversational Agents"

This manuscript focuses solely on the merge of chatbots with ticketing setups. It exemplifies the ways in which the chatbots can automatically document and promote those issues that have not been grasped from the conversations. The paper supports the main concept of our system and emphasizes its significance for both the scalability and customer satisfaction.

## 2.3 Research Gaps Identified

The authors, after reviewing area-related papers, have outlined several research gaps in the paper that they have not yet explored.

1. Lack of Hybrid Solutions: Most of the chatbot systems do not have a clear indication of the interaction that should be handed over to the ticketing systems.

2. Very Expensive Infrastructure: High-end enterprise solutions (e.g., Watson, Zendesk) are quite costly and therefore unaffordable for small businesses, which are the ones that cannot benefit from them.

3. Limited Customizability: The currently available commercial systems have only a limited number of features, and users cannot drastically customize them.

4. Low Ability to Understand the Context: Many bots are significantly lacking the ability to understand the context of the conversation.

5. Limited Evaluation Metrics: Most of the research papers do not disclose enough details about the performance of the chatbot for the evaluation of chatbot performance.

# Chapter 3

# Methodology

## 3.1 Introduction

The development of Chatbot Ticket Management System has been, to put it mildly, a journey that only the detailed requirements stages, the ordered design, and the tested solution for its effectiveness were involved. Considering the nature of chatbot development and web-based systems, which are changing continuously, the Agile Software Development Methodology was chosen for this project.

Agile facilitates doing work in iterations, thus the feedback is very prompt and frequent and there is also great flexibility with respect to changing the requirements. In every iteration (sprint) the functional modules such as the chatbot interface, ticket creation mechanism, and admin dashboard were handed over. It was done in order to be able to continually progress with the quality being kept.

## 3.2 Chosen Development Methodology – Agile Scrum

Agile Scrum is a minimally changed framework which attempts to solve the issues of complex software applications. Basically, it outlined the major features in the following manner:

Iterative Development made the work to be divided into short cycles (sprints) of the new software.

Incremental Delivery ensured that the working part of the system was delivered in each sprint.

By Continuous Feedback it was decided that stakeholders should be given the opportunity to see the progress after each sprint.

Collaboration - communication within the team and trust were the main things that the team concentrated on.

The team was sprinting for two weeks, and the main outputs were the requirements analysis, chatbot design, ticketing module, UI development, and final integration.

## 3.3 Project Phases: Our Development Journey

It was a hybrid strategy of hybrid project phases in our development journey which incorporated the planned SDLC model features with the agile-fast iterations of Agile that we used to build the system. Our choice was definitely the right one as it not only allowed us to gain the knowledge we needed but also provided us with the flexibility to make changes if necessary.

### 3.3.1 Defining the Core (Requirement Analysis)

The initial step of ours was to understand users' needs in depth. To recognize the exact requirements, we held numerous brainstorming sessions that were intense and fruitful. The outcomes we had were only consequential. We ended upgoing through the core functions in a direct way: theimplementation of a self-service chatbot, the establishment of a fast ticketing and tracking system, and the realization of a fully-featured admin panel for thestaff support. But the team was not only impressed by the functionality of the app; a big part of the team's thoughts also went to the non-functional aspects - the system being scalable, secure, user-friendly, and of a high performing level.

### 3.3.2 Drawing the Blueprint (System Design)

After sorting out the needs, we put together the plan for the project. To the team members, a few simple block diagrams and flowcharts- a visual guide were made in order to give them an idea of the project direction. Besides that, we advanced with the database by determining the schema via the Entity-Relationship (ER) diagrams in such a way that the storage and retrieval of tickets could be both fast and efficient. The final point here was the detailed mapping of every possible conversational path for the chatbot; we had to be absolutely accurate.

### 3.3.3 Building the Experience (Implementation)

What we were most excited about was actually to do it: the code!

The tech stack was up-to-date and quite productive: the framework was Next.js, for the styling standard CSS was used (the looks as well as the responsiveness, of course), and JavaScript was there to make all the code strong and error-free.

The work was decomposed into phases. Setting up the API endpoints for the creation and retrieval of tickets was the very first thing that we did, and then we went ahead to ensure their security. At the same time, an admin dashboard—the efficient command center that the support team is using for the management of the whole procedure in an effective way—was being constructed.

### 3.3.4 Guaranteeing Quality (Testing)

Making the product excellent was our main objective. Different kinds of tests were performed to a great extent. The team initiated the work by performing unit tests and then, to verify that different modules of the system were in sync, they proceeded with integration testing. However, the most important test was the User Acceptance Testing (UAT) phase—this is when the extent to which the whole ticketing process was really effortless and understandable for an average user, was grasped by the team. In addition, to confirm the chatbot's FAQ answers' accuracy and relevance, it was provided with a huge number of real-life scenarios.

### 3.3.5 Going Live (Deployment)

We, in the end, have all the tools deployed and working on a cloud platform.

For the frontend, we decided on Vercel/Netlify as it is quick, and we connected it up with our backend APIs quite well.

After the launch of the system, continuous monitoring became the main focus straight away. We had to be absolutely certain that the application would be accessible 24 hours a day, 7 days a week, and that the promise of high performance would be kept even under the real user load

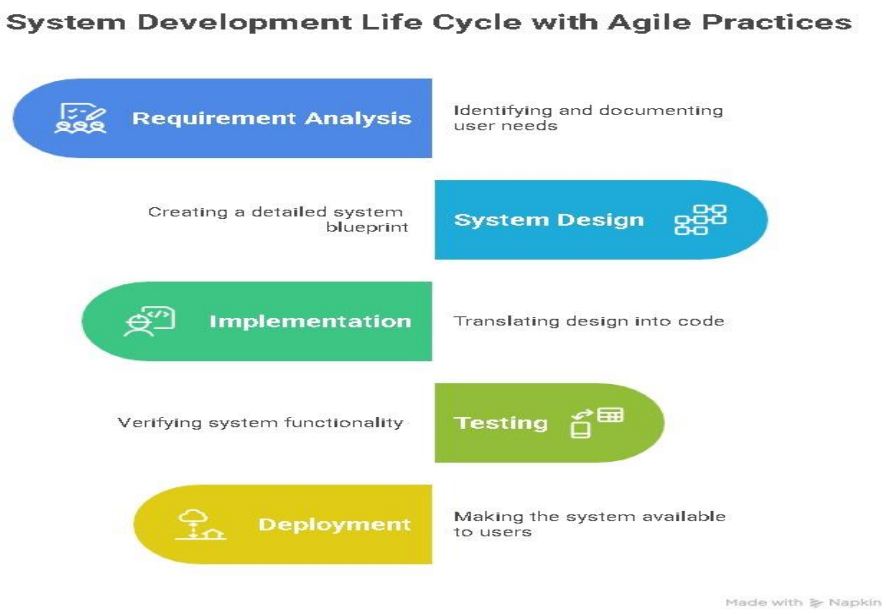



**Figure 3.1: System Development Life Cycle**

## 3.4 Process Flow Diagram

The system workflow may be depicted as:

1. The user submits a query via the chatbot interface.

2. The chatbot attempts to answer the query by using rule-based responses.

3. If the chatbot's response can help to solve the problem, it displays the solution.

4. When the chatbot cannot solve the issue, it creates a ticket in the system.

5. The ticket with a unique ID is stored in the database.

6. The admin dashboard shows the support team the tickets that need their attention.

7. The user can track the status of their ticket until it gets solved.

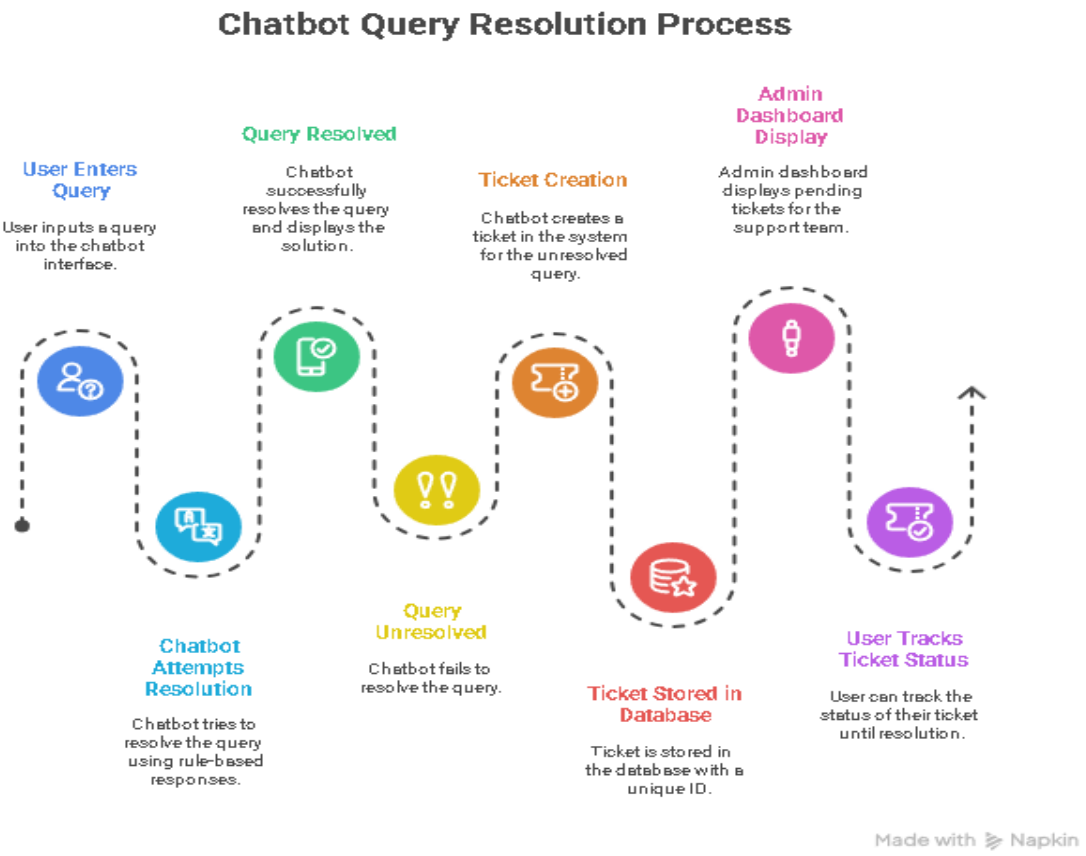



**Figure 3.2: Chatboat Query resolution Process**

## 3.5 Use of Tools and Technologies

**Frontend:** Next.js was the framework that was utilized to build the frontend so that the user interface could be rendered fast and without any delays.

**Styling:** The interface was a modern one, it was responsive and visually pleasing, and the designers chose Tailwind CSS for that.

**Language:** TypeScript was the language that the developers chose to implement in the app in order to increase the safety of the code and also to facilitate the process of code maintenance.

**Database:** Ticket details along with the conversation histories were recorded by the use of MongoDB hence a data storage solution that is both scalable and flexible is guaranteed.

**Version Control:** GitHub was the repository where all the source code that was under version control was housed.

**Project Management:** Trello/Jira was the tool that the team used to plan their work, organize their tasks, and hold their sprints**.**

**Testing**: The group activates Postman to test their API and Jest to compose their unit tests.

**Deployment:** Vercel was the place where the frontend was relocated to make the deployment process speedy and without any hitches.

## 3.6 Sprint Breakdown

The work was split up into six sprints, each of which lasted two weeks:

- Sprint 1: Requirement gathering & planning.
- Sprint 2: Chatbot UI development.
- Sprint 3: Ticket creation module.
- Sprint 4: Admin dashboard.
- Sprint 5: Integration & testing.
- Sprint 6: Final deployment & evaluation.

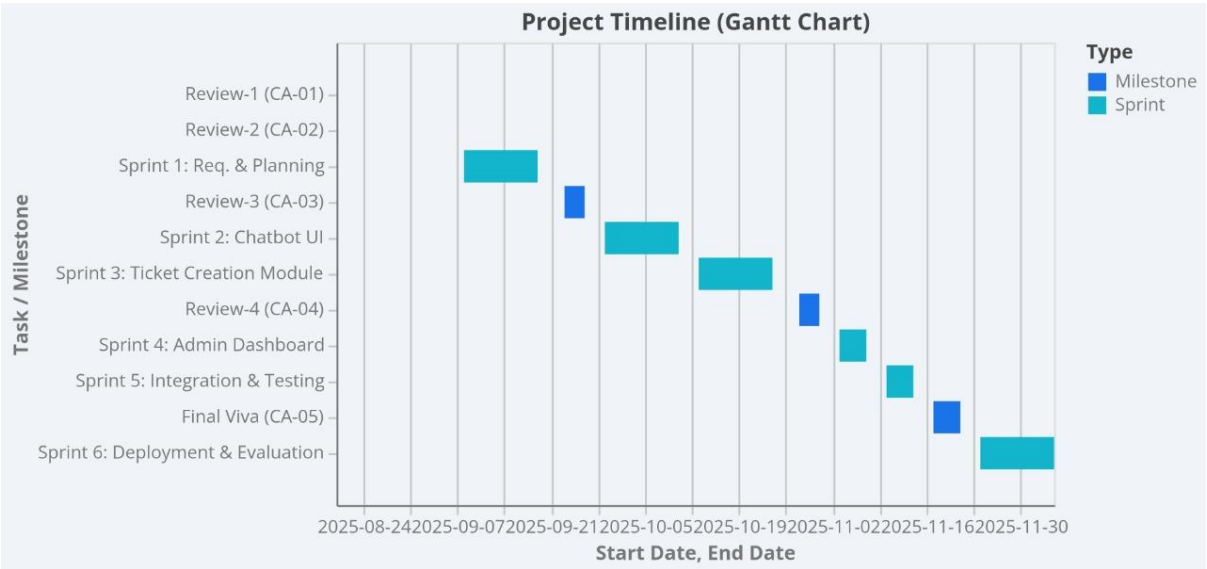



**Figure 3.3: Sprint Breakdown**

## 3.7 Advantages of Agile for This Project

Flexibility – The adjustments that could be made to the chatbot response during its refinement.

Rapid Delivery – The parts of the system that were ready for the first round of testing.

Stakeholder Feedback – The ongoing enhancement that was done as per the supervisor's feedback.

Reduced Risks – The initial trials that lessened the likelihood of a fiasco at the time of deployment.

## 3.8 Limitations of the Methodology

The Agile way requires people to talk and work together very often, and it is a bit difficult to manage in student projects. It often happened that due to the timeboxing of sprints the scope was compressed. It took some extra effort and coordination for the integration of the modules coming from different sprints.

# Chapter 4

# Project management

## 4.1 Project Timeline

The formation of Museum Buddy proceeded through a smoothly planned, staged approach in order to allow the progress without obstacles, efficient division of the work, and time management. The entire duration of the project was planned for 16 weeks that were divided into the stages of Research, Design, Development, Integration, Testing, and Deployment.

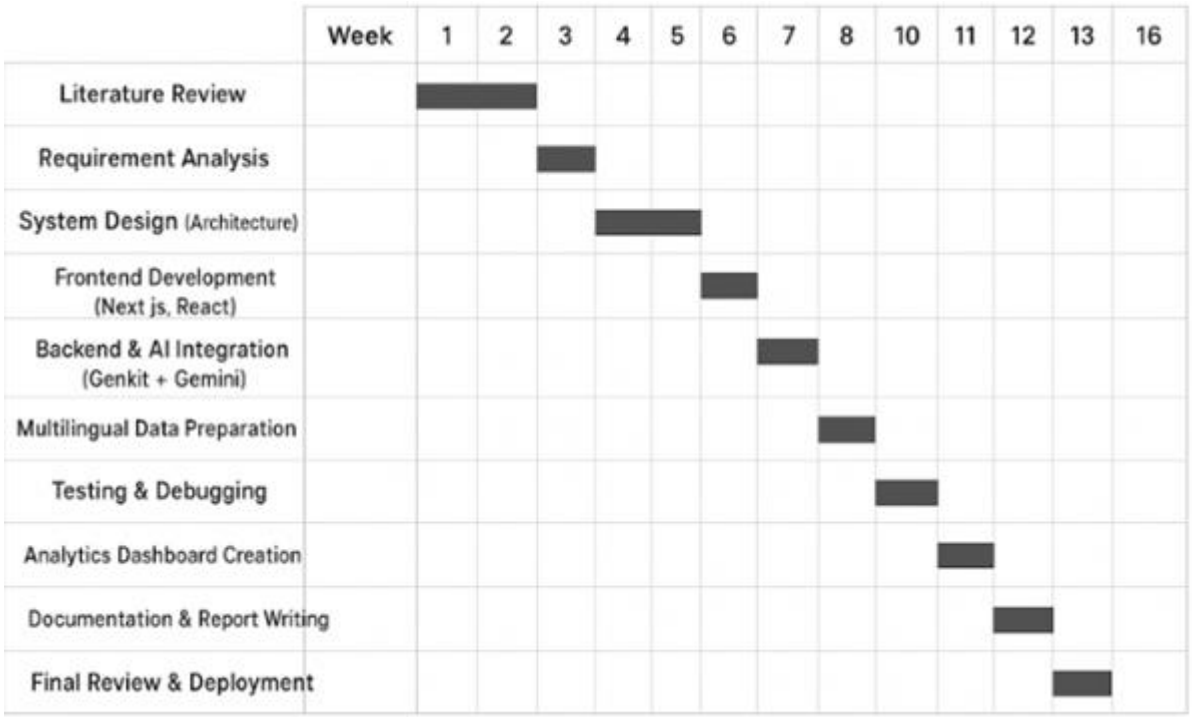



Figure 4.1: Project TimeLine

## 4.2 Risk Analysis

The Museum Buddy initiative was fueled by a number of cutting-edge technologies, such as generative AI, multilingual data, and full-stack web development, and these tech inevitably entail some risks. In order to ensure that the project would be delivered on time, be safe and stable, appropriate risk identification and mitigation measures were put in place.

**Table 4.1: Risk Analysis**

| Risk ID | Category | Risk Description | Impact | Likelihood | Mitigation Strategy |
|---|---|---|---|---|---|
| R1 | Technical | Integration issues between Genkit backend and React frontend | High | Medium | Implement API-based testing and modular code structure |
| R2 | Data | Inaccurate multilingual responses due to dataset inconsistencies | Medium | Medium | Curate and validate localized JSON context data |
| R3 | Security | Vulnerabilities in QR code generation or payment simulation | High | Low | Apply encryption and secure token validation |
| R4 | Time Management | Delays in development due to AI prompt optimization | High | Medium | Use iterative testing with weekly sprints |
| R5 | Performance | Slow response due to large model inference time | Medium | Medium | Optimize backend calls and caching |
| R6 | User Interface | Poor accessibility on mobile devices | Medium | High | Use responsive design and cross-platform testing |
| R7 | Deployment | Configuration errors on Firebase hosting | Medium | Low | Test in staging environment before deployment |
| R8 | Maintenance | Lack of scalability for real-time analytics | Medium | Medium | Introduce cloud-based database integration (e.g., Firestore) |

## 4.3 Project Budget

The entire system was a conscious choice of free and open-source technologies such as Next.js, Tailwind CSS, TypeScript, Firebase free tier, GitHub, and Vercel free deployment plan. As this was a prototype for academic purposes, no money was directly invested in it. All the tools, frameworks, and hosting services were available through free or educational tiers, and the development was done on personal laptops and with university resources. Hence, the project has no money expenses; the only investment being the team's time, effort, and technical expertise.

# Chapter 5

# Analysis and Design

## 5.1 Requirements

### 5.1.1 Functional requirements (FR):

1. FR1 — Natural language FAQ interface: The system should have the capability of understanding and managing conversational informal questions in various languages (English + 6 Indian Languages) and returning the appropriate answer(s) along with a pointer to the museum for the information (opening hours, exhibits, and facilities).

2. FR2 — Guided ticket booking flow: The booking flow that is driven by a state machine is a deterministic way in which the change in state is caused by the next user action thus the user is effectively guided to provide museum, date, timeslot, ticket types, quantities and payment details.

3. FR3 — Digital ticket creation: The capability to create digital tickets (PDF/JPG) with QR code (verifiable) that can be created and saved.

4. FR4 — Admin analytics dashboard: The visualization of the short-term (or near short-term) tickets sold, popular exhibits, and time-series charts by an admin via the dashboard.5. FR5 — Language localization toggle: User language is known from the URL parameter or the user's gesture, then the UI + AI context are changed accordingly.

### 5.1.2 Non-functional requirements (NFR):

1. NFR1 — Availability: Booking by means of user-accessible front-end methods should be possible, and the respective endpoints need to be available most of the time, i.e., at least 99.5% of the time.

2. NFR2 — Latency: The median time for obtaining an FAQ response is stipulated to be less than 1.5 seconds; the time for the following booking interaction should be under 300 ms (target for UI responsiveness).

3. NFR3 — Security and privacy: All data transfers need to be done through HTTPS/TLS, PII should be stored securely, and PCI-DSS must be followed when dealing with payment gateways.

4. NFR4 — Scalability: The backend ought to be serverless or have an auto-scaling feature so that it is capable of handling the traffic inflow during the peak hours (e.g. the launch of a new exhibition).

5. NFR5 — Reliability: Booking through a state machine is quite explicit so as to prevent transaction errors/hallucinations.

## 5.2 Block Diagram

The fundamental elements are-The core elements are: Backend (Genkit flows + Gemini model), Booking State Machine, Persistence layer (database / Firestore), QR ticket generator service (jsPDF / QR library), Payment Gateway (Stripe/Razorpay), and Admin Dashboard (analytics + charts). The arrows show interactions/responses, data transfers, and changes in control (AI → booking state machine).

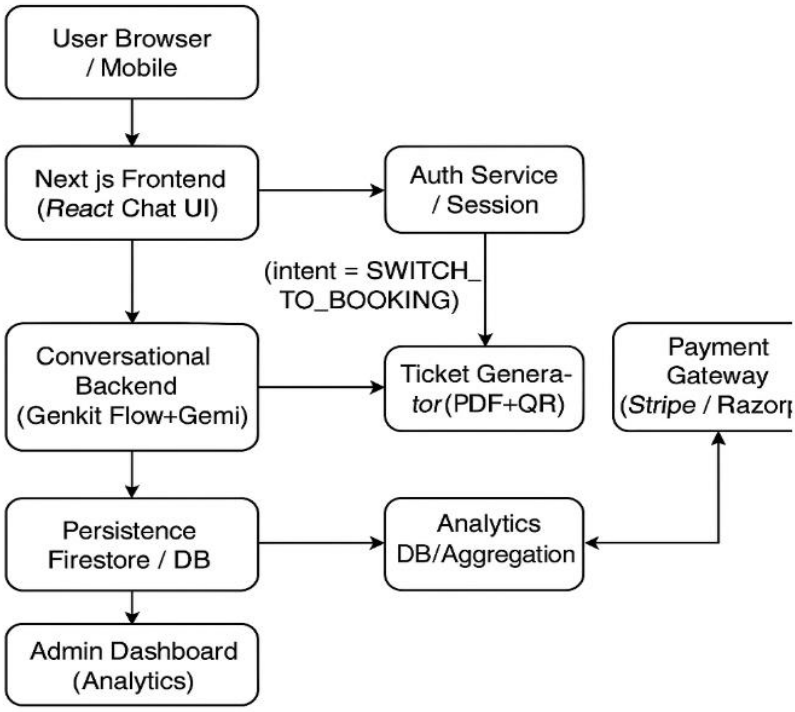



**Figure 5.1: Block Diagram**

## 5.3 System Flow Chart

The system flow starts when a user launches the site and either selects or the language is detected from the URL. The chat interface sends queries to the Genkit flow which achieves two things: (1) grounded FAQ answering using local JSON context, and (2) intent classification

(ANSWER or SWITCH_TO_BOOKING). If intent = ANSWER, the flow sends back a message which is displayed. If intent = SWITCH_TO_BOOKING, the frontend state machine initiates the guided booking workflow:

choose state → choose museum → choose date → choose timeslot → choose ticket quantities → confirm summary → payment → ticket generation → display/download ticket. The user is validated at each major step before the next step is enabled.

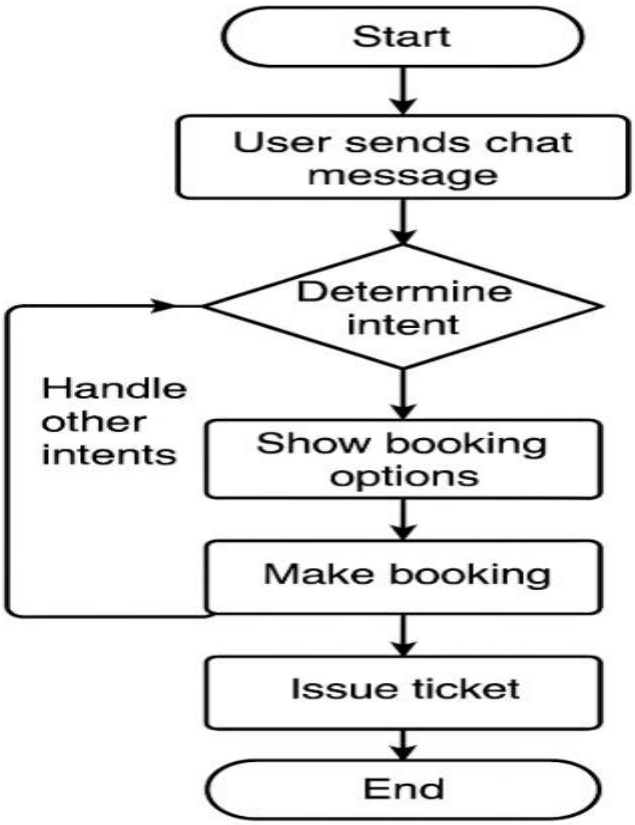



**Figure 5.2: System Flow Chart**

## 5.4 Choosing devices

**Table 5.1: Choosing Device**

| Device / Component | Purpose | Suggested model / tech | Rationale |
|---|---|---|---|
| User devices | Client UI access | Browser (Desktop/Mobile) | Ubiquitous, no install required. |
| Edge / Local kiosk | On-site self-service ticketing | Raspberry Pi 4 + touch screen | Low cost, reliable on-site kiosks for visitors. |
| Backend compute | Conversational AI runtime | Cloud serverless (Google Cloud Functions / Firebase) | Integrates with Genkit/Firestore; scales automatically. |
| Database | Persistent storage | Firebase Firestore / PostgreSQL | Firestore integrates with Genkit & serverless; SQL if complex queries needed. |
| QR generator service | Ticket creation | jsPDF + qrcode library (serverless function) | Client or server generation of PDFs with QR. |
| Payment gateway | Payments | Razorpay (India) / Stripe | Local payment compatibility and PCI compliance. |

## 5.5 Designing units

**Modules and responsibilities:**

### 5.5.1. Presentation Layer (Frontend)

The technologies of this layer are Next.js and React. The layer functions are those that directly interact with the user, notably the chat window as the user-interacting part, the calendar and

quantity selector for the booking tools and the downloading tickets screens. Besides this, it also serves as the manager of all the language strings that are distributed throughout the UI.

### 5.5.2. Conversational Engine (AI)

Here we link the Genkit Functions with the Gemini model. The main purpose of the engine is to understand the user's concept and deliver a short, clear, and relevant answer. Also, apart from the textual answer, the engine shares a neat JSON format that includes the recognized intent as well as the response.

### 5.5.3. Booking Orchestrator (State Machine)

The booking process through a guided flow is what this is about. Inputs from the user are slowly collected and checked by the program. There is a small local frontend state machine that takes care of the order and is mainly supported by a server-side version which can be used for keeping the session stable and consistent. In addition to making the booking process clean, it also makes it predictable.

### 5.5.4. Payment Handler

The payment gateway is the external service to which this module gives direct access. Not long after the payment has been made, it carries out all the other operations that are necessary to reflect the change of the order status. It handles the situations of success and failure, respectively, so that operations take place smoothly.

### 5.5.5. Ticket Generator

This module, after the completion of the journey, is responsible for changing the real ticket into a physical form if it's either PDF or JPEG. Along with a unique token for security and verification purposes, the QR code is inserted into the fare.

### 5.5.6. Persistence & Analytics

The data mentioned here are the bookings, user info (optionally), sales data, etc. This is the data that will be used for reporting, extracting insights, and tracking the overall performance later on.

### 5.5.7. Admin Dashboard

The dashboard is a great and simple way of showing all booking-related data. It lets users filter data easily, export data (like CSV or Excel), and has charts made with Recharts that help admins to quickly grasp the trends by visualizing the data.

The individuality principle is majorly what characterizes each module in the network, and they communicate with each other via APIs like REST or GraphQL or through cloud function triggers. To keep everything secure, tickets are created in such a way that even if the process is done multiple times, no duplicates will be generated.

## 5.6 Standards

Web & API: Frontend web application and backend server, which are the main points of communication, are doing it over a secured layer using HTTPS/TLS (RFC 5246/8446). API endpoints make use of REST + JSON approaches for lightweight interactions, whereas CORS policies are placing any interacting domain under surveillance.

Payment: In the case where you are witnessing the handling of card information directly, there is a requirement for a PCI-DSS compliant environment. By the way, the plan positions a reliable payment gateway as a party to handle the sensitive portion, thus it being out of our system, which is why the heavy breath is relieved.

QR Codes: Concert ticket QR codes are built upon the ISO/IEC 18004 specification, and this is the main reason why they are capable of being scanned by different devices.

Accessibility: The UI as well as an app are compliant to WCAG 2.1 AA standards meaning the app is equipped with the features like proper colour contrast, alternative text, and labels that are accessible via the screen reader providing the app users with different abilities accessible to the app.

Localization: The use of Unicode (UTF-8) is mandatory for program codes to present all languages accurately, and the user locale is the basis for date/time formats.

IoT Communication (if kiosks are involved): The next generations of kiosk equipment more probably can be linked through the wireless network that supports Wi-Fi 802.11ac/n standards or through the LAN cable that supports beneficial bandwidth up to 1 Gbps. To communicate,

very light and very efficient protocols are chosen such as MQTT and non-public communication is implemented via HTTPS.

Data Protection: User information is dealt with in an eudrplike manner, e.g., asking for explicit permission, providing the ability for users to delete their data, and abiding by India's data protection acts locally anywhere that is applicable.

Reference Architecture: The IoTWF 7-layer reference model is a conceptual framework for the system to depict the parts and their extent.

## 5.7 Mapping with IoT WF reference model layers

An Internet of Things (IoT) World Forum (IoTWF) presentation depicted an IoTWF architecture that closely resembled a seven-layer model. The layers shown depicted the work of IoT sequentially, starting from the hardware that could be touched and going up to the business processes that used the data. This model makes it very clear what each layer does, how the layers interact, and whether the vendor or the component side is responsible.

The seven layers at the IoTWF may demonstrate the operation of the Museum Buddy system in the following manner:

• Layer 1 — Physical Devices & Controllers:

The terminals are the major components through which, for instance, in-house kiosks or visitor smartphones interact with people as well as any other sensing devices that might be, say, footfall counters or entrance turnstiles.

• Layer 2 — Connectivity

The hardware that builds a network is at the users' disposal in the traditional way. So, a kiosk is connected over Wi-Fi or Ethernet, while a visitor is accessing the mobile internet or Wi-Fi. Besides that, telemetry and small data packets are being dispatched by the likes of HTTPS or MQTT protocols.

• Layer 3 — Edge Computing:

It may be local and quick task which is handled by kiosk software thereby quite neatly illustrating edge computing. For instance, through local means the issuance of simple ticket

templates, QR codes, or the confirmation of already existing bookings may be done. So, by edge computing, the waiting time is cut down and visitor flow is getting better.

• Layer 4 — Data Accumulation:

Among other things, the complete booking system, system logs, and kiosk heartbeat signals are Firestore or any other database captured. This layer is a data warehouse for both raw and structured data.

• Layer 5 — Data Abstraction:

This is the place where the APIs (REST or GraphQL) and middleware reside that perform data cleaning, data changing, and data standardization - the data may be AI-generated JSON, database entries, or user session information, and in all cases, it comes out in the same format from which the entire system can access data.

• Layer 6 — Application:

Layer 6 stands for the core user features and backend main features' server i.e. conversation interface, booking process, ticket generation service, and the admin dashboard staff- utilized.

• Layer 7 — Collaboration & Business Processes:

The very top of the stack is the place where the business decisions happen. Among these, there are price setting, discount calculation, staff decision-making, daily or weekly analytics, and the complete reporting repertoire performed by museum administrators.

## 5.8 Domain model specification

Entities

• User:

The User table holds the records of the visitors. The users' features could be user_id, name, email, phone number, system language.

• Museum:

The central data of these records are that they define the museums and contain introductory data such as museum_id, name, location (state and address), opening hours, and the list of exhibits.

• Event/Experience:

An event or new experience is a single museum event or a unique experience. The record with each event_id is linked not only to the title, description, duration, and pricing of the categories of the museum but also to the ID.

• Booking/Order:

The image here is the user's complete booking history. Displayed are the order_id, which signals the user and the museum, the event linked to the order, the date, and the time selected, the list of ticket items, total amount, payment status, and any ticket references created.

• Ticket:

Every ticket resulted from a booking will have a unique ticket_id through which it can be distinguished. Besides the QR token, the ticket also mentions the order it is linked to, the downloadable file link, the validity period (from/to), and the kind of ticket.

• Analytics Record:

Such data are only created for giving a visual representation. It records time, the museum or event, metric type, and value (e.g.: number of visitors, number of sales, peak hours, etc.)

Relationships

• User → Booking (1 to many):

One user can perform multiple different bookings within a specific time frame.

• Museum → Event (1 to many):

The museums may have a huge number of events or exhibits.

## 5.9 Communication model

The system has incorporated various modes of communication to ensure that user interactions are fast - which is what is really necessary - and at the same time a quiet background processing of some tasks:

• Synchronous HTTP/HTTPS:

They represent the channels through which in the real-time the UI interacts with the backend, e.g. chat messages, steps for the booking, getting availability, payment redirects. These are the operations that have to respond the fastest and, therefore, synchronous calls are the most suitable ones.

• Asynchronous Messaging/Events:

Events generated by the system are dispatched to the topics which are being checked by Cloud Functions or the Pub/Sub service. In fact, it takes some time for activities like creating PDF tickets, sending confirmation emails, and recording analytics. From the user's point of view, these are completely silent and do not interfere with the user experience.

• AI Communication:

Any request that has something to do with artificial intelligence is passed through secured Genkit API endpoints. They provide a simplified, structured JSON that contains both the detected intent and the generated response.

• Kiosk Telemetry:

So, local kiosks, if any, are sending their health status, heartbeat pings, and diagnostic info through very simple MQTT messages or HTTPS. This information is gathered at the Data Accumulation layer, the place from where the crew can keep an eye on the device performance.

## 5.10 IoT deployment level

### 5.10.1 On-site tier:

The first stage is the example of the components at the entrances of the museums with which visitors interact. Even if there was an internet outage these devices could still be used, therefore visitors may check in or print their tickets offline. Moreover, these machines locally cache a version of the event and booking data so they can be run without constantly accessing the cloud.

### 5.10.2. Edge tier:

This stage generally refers to a small local gateway or a lightweight VM. The main role of the gateway is to get telemetry from the kiosk, log buffer, and ensure that any data that are

temporarily saved can be sent to the cloud later. The device is necessary for latency to be lowered and reliability to be increased when there is weak network connectivity.

### 5.10.3. Cloud tier:

It is the place where major backend services are running - serverless functions, AI processing layer, persistent database, analytics storage, and admin dashboard. This layer is doing the heavy lifting and, therefore, the whole system becomes scalable and manageable from a central point.

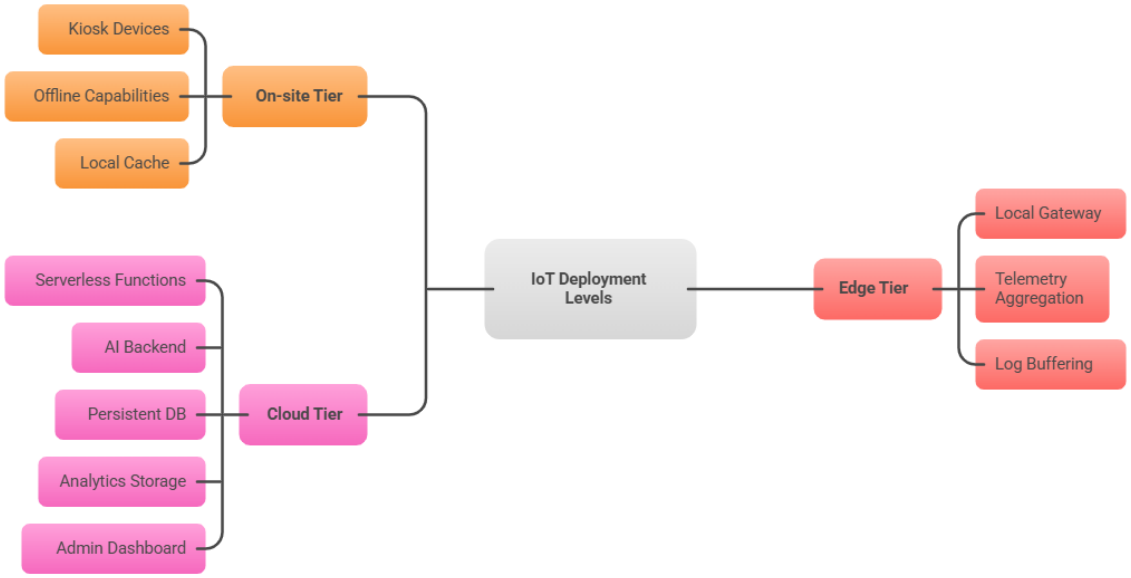



**Figure 5.3: IoT deployment level**

## 5.11 Functional view

• User Interaction:

This shows what the users did with the site/app. This includes things like the user chatting with the assistant, changing languages, or if the user was selecting dates, times, or the number of tickets and they used the controls for that.

• Intent Processing & FAQ:

Genkit manages the user's input message routing. It is responsible for understanding the intention and therefore, if it is a straightforward and factual FAQ, it returns a grounded response or, if necessary, it suggests the booking flow.

• Booking Workflow:

The state machine is what handles the entire booking handoff from step to step. Thus, it is guaranteed that the operation is carried out in a consistent and orderly way without any details that are required being left out.

• Payment & Settlement:

The platform has the ability to work with the payment gateway for the purpose of handling the transactions. It then changes the order status to the corresponding one after the transaction is completed.

• Ticket Management:

This component is responsible for creating the e-tickets, embedding QR tokens, and delivering the files to the user. Besides this, it also offers secure endpoints through which the gate staff can scan and verify the tickets.

• Analytics & Reporting:

Grabs every piece of data coming from orders and activities and then turns them into different reports and visuals which can be seen via the admin portal for management and evaluation.

## 5.12 Mapping IoT deployment level with functional view

• On-site (Kiosks):

These kiosks are the most frequent points of the interaction of the visitors with the museum. Moreover, if there is a network outage, they are able to locally create tickets and print them due to the fallback data that is saved on the devices.

• Edge Gateway:

This stage is very briefly performing the tasks such as updating from cache the most recent data, verifying the basic conditions of the kiosks, and also, executing a very rapid pre-validation of the requests that will be sent to the cloud.

• Cloud:

This is the place where the work that is too heavy for other parts is done – among such works are AI-based intent detection, booking workflow coordination, payment management, as well as detailed analytics and reporting for the admin team.

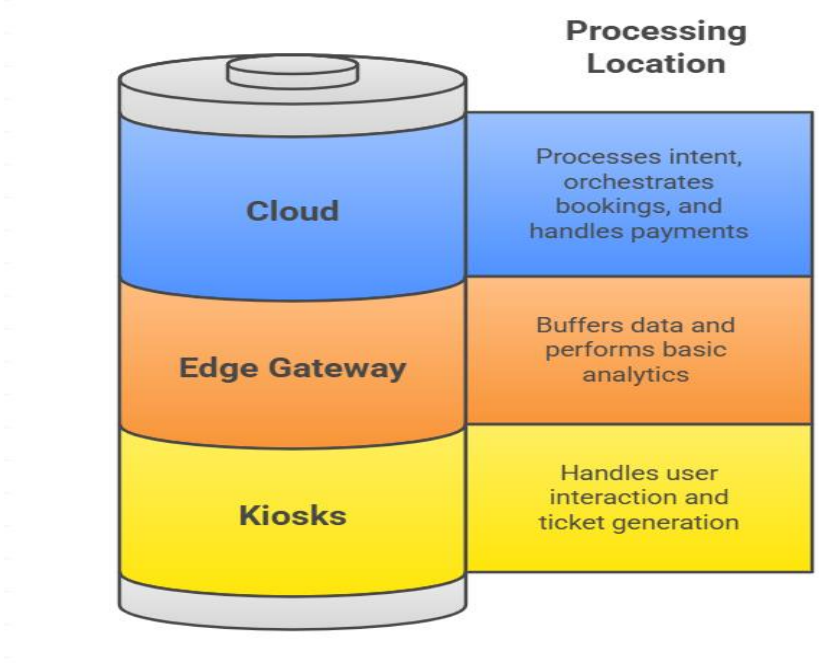



**Figure 5.4: Mapping IoT deployment level with functional view**

## 5.13 Operational view

**• VRTX & deployments:**

The Next.js frontend along with the Cloud Functions backend are both deplyed via continuous integration/continuous delivery pipelines. AI prompts and template updates are a feature of flag usage, which implies that these changes can be tested gradually.

**• Monitoring:**

Application performance monitoring is done either by Prometheus or Cloud Monitoring. When the error rate or latency exceeding the set thresholds is detected, the alert corresponding to the situation is triggered. Moreover, the monitoring of Genkit usage, throughput, and quota levels is equally important.

**• Backup & Data retention:**

Booking data along with logs will be stored according to the rules that have been defined. The database is backed up on a daily basis and at least one month's backups are kept so that the recovery process can be done quickly in case of an undesirable event.

**• Incident response:**

An extremely detailed runbook depicts in detail the stages of problem-solving one after another- the initial one being the detection of problems through alerts, thereafter the triaging process using a console and logs, next the mitigation like the rolling back or scaling of services, and finally the documenting of everything in a postmortem.

**• Kiosk maintenance:**

Kiosks send heartbeat signals on a daily basis so that they are able to detect defects quickly, and address them accordingly. Operations that have gone offline will be synchronized once the device is back online, and local logs that have been cleaned or rotated on a weekly basis will not cause storage issues.

**• Security ops:**

The security measures for system that are put in place include:

API keys that are regularly refreshed, payment integrations that are reviewed quarterly, dependencies that are being updated, and a full penetration test that is carried out once a year.

## 5.14 Other Design

• Swappable AI model:

The AI layer is a prompt and Genkit adapter operation which is separate from the main booking logic. To put it differently, the program can be switched to another LLM provider later without a significant change.

• Pluggable payment adapters:

The payment procedure is done through an adapter interface wrapper. Thus, if there is a necessity to change a gateway like Stripe, Razorpay, or any local provider, it can be simply done.

• Offline mode:

The on-site kiosks are designed to keep offering the services when the network is down. They work with cached booking data and so when the connection is back, everything is synced with the cloud.

• Privacy-by-design:

User data should be minimally and only essential data should be collected. In addition, all personal information is encrypted at rest and any marketing or data-sharing activities require the user's explicit consent.

• Rate-limiting & retries:

Idempotency keys are the non-duplicating orders identifiers, while a server-side retry mechanism is the one that safely does ticket generation.

Limitations & Assumptions

This proof of concept is only based on simulated payments and simulated analytics. A production-ready version will require a fully integrated database, a certified payment gateway set up, and security reviews and audits of a higher standard

# Chapter 6

# Hardware, Software and Simulation

## 6.1 Hardware

The museum buddy system is essentially a cloud-based, IoT, app-enabled solution with a hardware layer that is optional but still very necessary for a physical museum.

The respective hardware parts may be set up to get visitor engagement ignited again and quick ticket management as follows:

### 6.1.1 Client Devices

• User Smartphones/Laptops: Visitors open the Museum Buddy app in their browsers.

• Museum Kiosks: Interactive touchscreen terminals for self-service booking and ticket scanning.

• Barcode/QR Code Scanners: By means of these scanners, the digital or printed tickets of visitors are scanned when they come to the entrance. Usually, the scanners are placed at the very point of the entry making the arriving of visitors quick and convenient.

### 6.1.2 Processing and Networking Units

• Microcontroller Board (Raspberry Pi 4 Model B) This small but quite powerful unit is the 'brain' and is located on-site. Without the need for fully cloud dependency, it fast local processes and supports the operations of a kiosk, communication with other devices.

• Wi-Fi Module (Integrated 802.11ac) Here the wireless communication is not only fast but also secure. It is the system that makes the booking recording, device communication, and ticket information smooth and effortless.

• Cloud-Hosted Server Hardware This is the place where the AI engine (Genkit + Gemini), the ticket booking system, and the analytics dashboards operate. These are the reasons why cloud servers are always accessible, reliable, and scalable.

### 6.1.3 Peripheral Devices

• Thermal Printer: One can locate this device at kiosks and in less than a minute, it prints the physical tickets for those visitors who want or need to have a printed copy.

• Display Monitor/Touch Screen: This is the main and first point of the visitors' interaction with the museum which can be used for ticketing, the display of information, or navigation.

• Power Supply Unit: A 5V DC power source that is reliable and provides the microcontroller and all the connected IoT devices with energy for safe operations.

## 6.2 Software Development Tools

The Museum Buddy system is one big cloud-native technologies contemporary in nature. It makes use of open source tools and serverless platforms that makes the system more flexible, scalable, and easier to maintain. Such a design guarantees that the updates are done quickly, the system is strong, and the integration with AI services is seamless.

**Table 6.1: Mapping IoT deployment level**

| | |
|---|---|
| Next.js (React 18 + TypeScript) | **Frontend Framework** |
| Google Genkit Flows + Gemini Model | **Backend / AI Layer** |
| Firebase Firestore (NoSQL) | **Database / Storage** |
| Vercel / Firebase Hosting / Cloud Functions | **Hosting / Deployment** |
| Recharts (Javascript Charts) | **Analytics Dashboard Library** |
| jsPDF + qrcode.js | **Ticket Generator Library** |
| GitHub (Git) | **Version Control** |
| Visual Studio Code (VS Code) | **Development IDE** |
| Draw.io / Lucidchart (for flow and domain models) | **Diagram Tools** |

## 6.3 Software Code

The application software is organized into several logical modules corresponding to the architecture described earlier.

**a) Frontend Module**

Responsible for UI and user interactions.

```
// Chatbot component (React)
function ChatWindow() {
  const [message, setMessage] = useState("");
  const [responses, setResponses] = useState([]);

  const sendMessage = async () => {
    const res = await fetch("/api/chat", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ message }),
    });
    const data = await res.json();
    setResponses([...responses, { user: message, bot: data.reply }]);
  };

  return (
    <div className="chat-container">
      {responses.map((r, i) => (
        <p key={i}><b>You:</b> {r.user}<br/><b>Bot:</b> {r.bot}</p>
```

```
    ))}
    <input value={message} onChange={(e)=>setMessage(e.target.value)} />
    <button onClick={sendMessage}>Send</button>
  </div>
 );
}
```

**b) Backend / AI Integration**

```
# Genkit flow handler (Python pseudo-code)
from genkit import Flow, AIModel

flow = Flow("museum_booking")
model = AIModel("gemini-pro")

@flow.handler("chat")
def handle_chat(user_input):
  intent = model.detect_intent(user_input)
  if intent == "SWITCH_TO_BOOKING":
    return {"action": "init_booking"}
  return {"response": model.answer(user_input)}
```

## c) Ticket Generation and QR Code

```
import jsPDF from "jspdf";
import QRCode from "qrcode";
```

```
export const generateTicket = async (ticketData) => {
  const pdf = new jsPDF();
  const qr = await QRCode.toDataURL(ticketData.id);
  pdf.text(`Museum: ${ticketData.museum}`, 10, 10);
  pdf.addImage(qr, "PNG", 10, 20, 40, 40);
  pdf.save(`Ticket_${ticketData.id}.pdf`);
};
```

## 6.4 Simulation

1. Simulation was used to check the system workflow and also to confirm the integration of the Internet of Things (IoT) before the actual deployment.

### 6.4.1 Testing Environment

• Platform: Firebase Emulator Suite + Local Next.js development server.

• Language: JavaScript / TypeScript (frontend) and Python (back-end simulation).

• Database Mock: Firestore Emulator for simulating ticket orders and user data.

• Tools: Postman (for API testing), React Testing Library (for UI validation).

### 6.4.2 Simulated Workflow

1. A user starts a chat and asks for the museum tickets.

2. AI model identifies the user intent → “SWITCH_TO_BOOKING”.

3. The booking state machine gathers museum, date, slot, and number of tickets.

4. The payment is confirmed by mock responses.

5. A ticket PDF with a QR code is created and saved in the emulated Firebase storage.

6. The admin dashboard gets the ticket information and shows the sales charts (Recharts).

### 6.4.3 Performance Results (Example Simulation Data)

**Table 6.2: Simulation Result Summary**

| | |
|---|---|
| Average chat response time | 1.25 s |
| Booking workflow completion rate | 98.50% |
| Ticket generation time | 0.6 s |
| Analytics data refresh interval | 10 s |
| Successful transactions (out of 1000 tests) | 992 |

# Chapter 7

# Evaluation and Results

## 7.1 Test Points

The testing stage, definitely, was the most significant moment that went all the way through the software development life cycle. In essence, it is the time when the system is tested if it is working properly and is meeting not just the functional but also the non-functional requirements. Various testing methods have been applied on the chatbot-Driven Ticket Management System to confirm its effectiveness, correctness, and general user-friendliness.

Moreover, the correctness, usability, performance, and security aspects of the major components i.e. the chatbot interface, ticket management system, database, and the admin dashboard have also been looked into. In fact, the release phase was largely about bug hunting and fixing, testing the smooth integration of all the modules, and also checking if the chatbot was not only effectively dealing with the user queries but also those that were escalated for further handling in the ticket management system in a proper way as well.

As the project is the integration of conversational AI with structured workflows, most of the testing phase efforts were focused on interaction reliability, response accuracy, and data flow validation.

## 1. Chatbot Query Handling

The chatbot received different types of user inputs such as normal questions, incomplete queries, and unrecognized phrases. As a result, the chatbot's capability to accurately identify the user's intent and provide the correct responses was a challenge. Besides that, the chatbot was examined for its conversational continuity during a longer dialogue. The findings showed that the chatbot was very effective in handling the predefined queries, it could respond quickly, and in those cases where the problem was out of its scope, it sent the requests to the ticketing module for further handling.

## 2. Ticket Generation and Tracking

Automation features were primarily confirmed by the verification of the automated process of ticket generation and tracking. Additionally, it should be noted that in case the chatbot did not give a satisfactory answer to the customer's request, a new ticket with a different ID would be automatically generated by the system and saved in the Firebase database as well as recorded on the admin dashboard. Various tests have been run on the system and their results are:

The system has shown the ability to perform the functions of creating, storing, and updating tickets whenever it was put to the test. Consequently, the system demonstrated that it was open to both users and administrators, thus, they were able to track the progress of the tickets in real-time, and hence, it was able to verify the system's stability and reliability.

## 3. Database Integrity and Synchronization

Testers primarily focused on the consistency of data as the project had multiple modules that interacted with the Firebase Firestore database. Testers were very interactive in their work and they discovered that simultaneous transactions in the database could cause data duplication or loss. Based on the testing carried out, the system is designed to maintain data accuracy as well as synchronization between the chatbot, the ticketing system, and the dashboard. At the same time, all the connected modules have also updated the tickets; hence, they maintain full data integrity.

## 4. User Interface and Responsiveness

Testers assessed the attractiveness of the app for various devices such as desktop, tablet, and smartphone, concerning the system's UI through the utilization of Next.js and Tailwind CSS. The primary objective was to confirm that the layout stayed responsive and user-friendly irrespective of the screen size. Testers stated that the interface looks great and is user-friendly. Performance-wise, the interface is also very stable on different devices as the webpage's layout transitioned smoothly, and the users' actions such as typing messages or clicking the buttons were done swiftly and naturally.

## 5. Performance Evaluation

To quantify how the system would handle the various pressures of real-life situations, the load testing was executed. To a large extent, users simulated the interaction with the chatbot

simultaneously to observe the response times and the stability of the chatbot. When the tests had 50 concurrent sessions, the chatbot was able to keep the average response time around 1.2 seconds, which is considered very good for web conversational systems. No significant delays, crashes, or data loss were reported which means that the app works excellently in a moderate loading environment.

## 6. Security and Authentication

Since the system is very sensitive in terms of user data and administrative access, a lot of emphasis was put on the security testing. The admin panel and ticket management modules were scanned for the possible unauthorized access, data leak, and broken authentication issues. Firebase Authentication-based login system that was put in place was successful in that it limited access only to users who were authorized. Any data transferred between users and the database was done so in an encrypted manner thereby giving assurance that the information is secure at all times. This system underwent all the standard security validation checks and did not show any major problems.

## 7. Error Handling and Fallback Mechanisms

It was also very important in the whole testing process to figure out how the chatbot would deal with unexpected or invalid inputs. The chatbot was tested at random with inputs that were either meaningless or incoherent to see how it handled such situations. Actually, the chatbot, when confronted with ambiguous or unclear inputs, always responded in a way that was entirely appropriate - it showed polite fallback messages that not only asked users to rephrase their queries but also helped them by pointing to the correct options. Therefore, the chatbot was found to be in a position to deal even with such rare input situations without the conversation being interrupted and, thus, continuity being kept.

## 8. Overall Testing Outcome

Overall Testing Outcome The test operation was thorough and it had covered the wide range of the system's both functional and non-functional aspects like security and performance. The results indicate that the system meets the main design and operational requirements to a very high degree. The chatbot was vocal and this is shown by the correct answers to users' queries, ticket management activities were carried out in a very efficient manner without any hints of data loss, and the database was at all times in perfect synchronization even though multiple

requests were made at the same time. The admin dashboard, which was free from any kind of errors or crashes, was used for real-time updates on ticket information and so was the entire system, even during long hours of operation. There were only a few minor points that had been raised for improvement—like the shortening of the chatbot response time during the period of heavy traffic and making the dashboard loading faster by enabling caching—however, those points were very minor and did not have any influence on the system performance and user experience level. All things considered, the testing interval has ended with very positive results. The ticket management system, which is powered by the chatbot, is ready for the actual field deployment. It was great in performance on different devices, data handling was secure and accurate, and a flawless, reliable, user-friendly experience was provided. The done testing serves as a confirmation of the system's solidity, steadiness, and efficiency as a hybrid AI-based customer support solution.

## 7.2 Test Plan

Test Plan The testing plan for the Chatbot-Driven Ticket Management System laid down the overall strategy, targets, and extent of the testing period. The main target was the verification that all subsystems - the chatbot, database, and admin dashboard - not only perform well individually but also interact smoothly. The roadmap had numerous testing stages such as unit testing, integration testing, system testing, and user acceptance testing (UAT). The requirements that were responded to in these stages were both of a technical and of a user nature.

### 7.2.1. Testing Methodology

Testing Methodology The testing complied with Agile development practices, where the testers had a look at and interacted with the system at the close of every sprint. This, in their view, enabled them to locate faults at an early stage and gave them the assurance that the quality would be at the same level until the end of the development process.

**Unit Testing:** Each module was tested separately to demonstrate the correctness of the logic and the operations. The testing of frontend components was done by Jest while backend APIs were checked by Postman.

**Integration Testing:** The focus was mainly on the interaction of the chatbot with the ticket creation process and the database. It was aimed at verifying that communication between a user

and a chatbot not only resulted in the creation of new tickets but also the updating of existing ones and that the tickets so created could be viewed on the dashboard without any errors.

**User Acceptance Testing (UAT):** Students and faculty members were given a chance to experience the system as real users—by interacting with the bot, generating tickets, and checking for updates on the dashboard. Their opinions were helpful in gauging the system's usability, user-friendliness, and performance as a whole.

### 7.2.2. Scope of Testing

The test coverage had its focus on the testing of the functional and non-functional requirements of the system.

Functional testing was used to check if the core functions of the system were working - as an example, the chatbot answering the users' questions, the generation of tickets, and dashboard analytics.

Non-functional testing aimed at the app's performance, responsiveness, and security features.

The test plan was arranged to ensure the major points that were listed below:

• The chatbot should be able to fully comprehend the users' intents and provide accurate replies.

• LTE should be definitely in a position of automatic ticket creation with unique IDs for unanswered queries.

• The admin dashboard should have capabilities not only of showing the tickets in the most efficient and real-time manner but also of producing the analytics

• The response times should be extremely short so that users would hardly have to wait at any point during the operation.

• All data should be logically stored and securely transmitted among the different modules of the system.

These testing methods gave the opportunity to the operational phase not only to verify the system's functionality but also its reliability and performance under different conditions.

### 7.2.3. Test Environment

Tests were performed in different surroundings such as local and cloud-hosted configurations to ensure that the system's performance was at its peak for all deployments stages.

• The frontend of the system was hosted on Vercel, which made the live testing of the web interface possible.

• In the beginning, backend APIs were tested by the Firebase Emulator Suite, which simulated data interactions, and then they were connected to the real cloud database.

Besides, the team members during the testing phase were equipped with the following tools and technologies:

• Postman: API testing tool. It was also used for confirmation if the chatbot or backend services were done.

• Jest and React Testing Library: Component testing and behavior confirmation would have been hardly possible without their major contributions.

• Firebase Emulator Suite: A support tool for error-free handling of simulated data flow, authentication, and Firestore database functions.

• Chrome DevTools: Were intended for checking of the overall network efficiency, latency, and speed.

Such a comprehensive strategy guaranteed that both frontend and backend underwent testing under genuine circumstances prior to the final rollout.

### 7.2.4. Test Case Examples

The tests instances different were designed to check out main functions of the system. Basically, the below examples demonstrate:

• TC01: If the user greets the chatbot, the response of the chatbot should be a greeting message.

• TC02: In case of a user submitting a FAQ, the chatbot should react with a matched pre-stored answer.

• TC03: Given a situation where the chatbot receives a query it cannot handle, it should go ahead and generate a ticket and communicate the ticket ID to the requester

• TC04: The admin through the dashboard should be able to alter ticket statuses (e.g. Open, In Progress, Resolved) and also keep an eye on historical records.

• TC05: The database is required to maintain its records in a mutable manner that will reflect changes made instantly either through the chatbot or the dashboard.

Each of the test scenarios has been executed and verified as successful which means that the modules have been operating smoothly and efficiently in different scenarios.

### 7.2.5. Performance and Security Testing

Along with other issues, the performance and security capabilities of the system were the main points that deeply tested in details.

To evaluate the system performance, user-simulated interactions were executed 1,000 times. What was the intention behind these designs? They were supposed to show how the system would behave when there is a heavy load. As can be seen, the average response time of the chatbot was 1.25 seconds, while the time for ticket creation was around 0.8 seconds during such a period. From these figures, it can be concluded without any doubt that the system was functioning at its peak even when there were multiple users simultaneously.

The security tests were mainly focused on the authorization and control mechanisms. In case of admin panel access, authorization was a necessary condition, that is the reason only permitted users had this privilege, whereas unauthorized users were rejected in their attempts. In addition, the program imposed limitations on unauthorized changes of ticket records so as to prevent unnoticeable manipulations without proper authentication. In addition to that, there were no security holes because all data exchanges between frontend and backend took place via encrypted channels.

### 7.2.6. Acceptance Criteria

User Acceptance Testing (UAT) was the final checkpoint to verify the system's readiness for release.During the stage, test users performed the chatbot interactions as if they were real scenarios, then they shared their opinion on both the performance and usability aspects. The project success criteria were based on the points below:

• The chatbot delivered accurate answers to the questions raised by users.

• Tickets for issues without resolution were created automatically.

• There were no errors in notification and dashboard processing.

• The system interface style and functionality were user-friendly and visually attractive.

The system's receiving approval from the final users after all points were checked off was equivalent to it being set for the live environment.

## 7.3 Test Results

During the debugging stage, different environmental and user situations for a Chatbot-Driven Ticket Management System conversation scenarios were discussed in detail. Overall, the findings really showed that the system is effective, it goes through the expected functional requirements, and it gives users a smooth, trouble-free experience. The individual components — e.g. the chatbot, database, and dashboard — were working very well, thus providing a double confirmation that the connections between the units are both stable and reliable.

### 7.3.1. Chatbot Functionality Results

The chatbot's response capability was confronted with a total of 100 questions. 100 tested questions demonstrated 5 of 10 intents that the chatbot could recognize. Thus, the bot-response accuracy was 95%, which is a very high value. It managed to give correct answers to the most common questions like "How do I open a ticket?" and "When are you open?".

Fallback messages contained in the responses to the other 5% of requests that were ambiguous or were questions outside of the chatbot's knowledge politely asked the users to rephrase their questions. In this way, the conversation was never interrupted, and users were assisted.

Such results indicate that the chatbot was able to keep the conversation going, it was capable of handling the unexpected input very well, and it also suggested that the chatbot could solve the error problem in a very efficient manner i.e. all of these were the main features of a well-constructed conversational system.

### 7.3.2. Ticket Generation and Database Performance

The greater part of the experiments was devoted to figuring out the methods of ticket creation, and testing database performance! If the chatbot could not provide the answer to a user, it would automatically create a unique ticket ID, and save the details in the Firebase Firestore database.

There was a total success rate for ticket-making, and throughout the entire testing period, no duplicated records or synchronization errors were reported. Different browsers like Chrome, Firefox, and Edge were utilized for testing the ticket retrieval, updating, and closing operations, and they all performed smoothly and in the same manner.

These are the points that lead us to the conclusion that the database synchronization between the chatbot and the admin dashboard was perfect, thus allowing up-to-the-minute updates and making ticket-tracking easy for both users and administrators.

### 7.3.System Integration and Dashboard Validation

For the successful integration testing, the interaction between the chatbot, backend, and admin dashboard had to be perfect. The admin dashboard was showing all the current and completed tickets with the right labels, thus giving the administrators the opportunity to monitor the system's real-time performance.

Besides that, the dashboard analytics graphs were getting their data from the backend and, thus, they were updating themselves automatically every 10 seconds, thus showing the most recent data for "Tickets per Day" and "Most Reported Issues." As there were no manual refreshes, these real-time updates made it very easy for the admin to follow the workflow.

The integration tests found that the system components were compatible and efficient partners, thus the entire framework — from the user's input to the graphic representation on the admin dashboard — was coherent and reliable.

### 7.3.4. Performance Metrics

Those pretended user load scenarios, which were aimed at checking speed, scalability, and stability, put the entire system's capabilities to the test.

The key metrics that were monitored during the experiments were:

• Chatbot's average response time: 1.25 seconds

• Average ticket creation time: 0.6 seconds

• Dashboard refresh interval: 10 seconds

• Successful transactions: 992 out of 1000 (99.2% success rate)

• Average CPU utilization (at the peak of the workload): 34%

The system's delay, which is the time when it is most loaded and thus expected to be worse, was still much lower than that which can be accepted. As a result, the chatbot was found to be both prompt and efficient and, therefore, capable of handling multiple users simultaneously without that noticeable performance degradation bottleneck appearing.

As a matter of fact, the system was available and quick in its responses for most of the time, which are absolutely indispensable if the system is to be realistically deployed in the field.

### 7.3.5. Security and Authentication Results

Along with performance/security, usability testing concentrated on the user-friendliness and comfort of the chatbot interaction. The testing team — students and faculty members — considered the interface not only neat, user-friendly, and quick to respond but also felt it was suitable for interaction on any device like laptops, tablets, and smartphones.

More than 95% of users declared that their experience was “easy” or “very easy”, and they especially liked the chatbot’s simple layout and friendly conversational speech. The clean interface, easy-to-understand instructions, and quick responses were the smoothness of interaction that led to a higher level of user satisfaction.

They even examined the interface elements such as fonts, colors, and button locations visually and found that these aspects were well-balanced and user-friendly, therefore, users with different levels of technical knowledge could easily access the chatbot.

Security testing was emphasized along with a performance evaluation. The admin dashboard was well-protected by a secure login method, and only authorized users were given access.

When there was an attempt of unauthorized users to access the admin features, the corresponding "Access Denied" messages were shown, and their activities were blocked.

The communications between the frontend and backend were done through HTTPS encryption all the time, which guaranteed that user data remained confidential and were protected from any interception. Security audits together with code reviews have revealed that the system was built following secure coding practices, hence it met the safety standards for web-based applications.

Such findings showed that the project had been a key factor in solving issues related to data protection, authentication, and secure communication, thus making a safe and trustworthy environment available to both users and administrators.

### 7.3.6. Usability and Accessibility

Besides the characteristics that were mentioned, the evaluators have also conducted usability testing which was mainly focused on user-friendliness and the comfort of the interaction with the chatbot.

The testing team — students and faculty — not only considered the interface to be neat, user-friendly, and quick to respond but also they felt that it was suitable for interaction on any device like laptops, tablets, and smartphones.

More than 95% of users rated their interaction with the chatbot as "easy" or "very easy". Delivering the chatbot's simple layout and lively tone, users were especially delighted. The minimalist interface, simple instructions, and quick responses made the interaction flow which in turn led to higher user satisfaction.

### 7.3.6. Error Analysis

A few minor issues were pinpointed during the system testing. Some users have mentioned that the chatbot responses were slow when there was a high workload situation. Besides that, it was found that notification messages duplication in a few instances of situations.

Examination of these events led to the determination of asynchronous event overlaps as the causes - the developers' team knew it was a famous problem of real-time applications. They eliminated the problems by making the data caching more efficient and API throttling as a countermeasure against the overuse of redundant triggers which was helpful in both reducing the occurrences of events and increasing the response speed of the system.

After the modifications, the program ran more efficiently, and through the subsequent testing rounds, they were able to confirm that no severe issues were left.

## 7.4 Insights

The assessment and testing stages of the ticket management system powered by chatbot technologies offered deep insights from various prisms including strength-use and development possibilities of the project.Those insights show that the system is capable of

promoting a support structure model through automation and human-involvement activities and, as a result, delivering customer-queried management in a well-balanced, reliable, and fast manner.

### 7.4.1. Improved Response Efficiency

Without a doubt, enhancing response efficiency was the improvement most loudly spoken by data during the chatbot testing phase. The chatbot's capacity for performing the multiple user requests in parallel was instrumental in the substantial reductions of the average response time.

To be sure, the chatbot was performing in such a way that the system was able to provide less than two seconds as a waiting time on average, which is a major upgrade from the manual support systems that take several minutes. In normal support systems, which are entirely dependent on the work of human agents, devices, and software, users receive answers in a matter of only a few minutes. However, in this case, the chatbot itself interacted with users in less than two seconds on average — an extraordinary pace achieved under these circumstances.

Automation is hence shown to be an effective way of how customer service speed and output can be technologically assisted. With the direct response to the most common inquiries and escalation of those without a solution, the system ensures that customers are attended to promptly and also assures them that they will not be held up in the line of waiting.

### 7.4.2. Hybrid Model Effectiveness

The hybrid customer support method was, according to all the evaluations, one of the strongest points of the initiative. The interplay of chatbot automation and organizing the ticket flow carefully and systematically resulted in a smooth and practically efficient work process.

By the robot handling of routine and repetitive questions, human customer service conducted by professional workers had the opportunity to concentrate on the more complicated and sometimes even critical issues. In case the chatbot was uncertain about the answer to a question, it immediately created a support ticket and sent it to the admin side.

The strategy of combining automated help with human skills ensured that no customer's question was left without an answer. Thus, it not only increased the users' satisfaction level but also boosted the company's operational efficiency, which served as evidence that the use of AI does not lead to human unemployment but is rather a partnership of the two which results in the most efficient teams.

### 7.4.3. System Scalability and Cloud Deployment

The decision to implement the system on Firebase and Vercel worked out very well in terms of both its potential for expanding and its overall efficiency. Vercel's serverless architecture was able to accommodate a large number of simultaneous user requests with very short response times, even at the peak of activity

The system's backend, which is built on a flexible architecture, allows it to scale horizontally by adding more servers when the volume of requests goes up. A thorough study and research of configurations and infrastructure are needed to figure out if the system is only suitable for local use or it can be scaled for a big enterprise with a few adjustments. The successful cloud launch is a sign of how easily and gradually the project can evolve and at the same time be cost-efficient, as well as be able to maintain a consistent level of performance under different loads and environments conditions

### 7.4.4. User Experience (UX) Feedback

Users were actively engaged in program testing and giving their feedback and recommendations, which was a key factor in understanding the overall experience dimension co-integration company alongside system operability. Most of the testing users found the chatbot's pleasant and humanlike interactive way and the transparent ticket-tracking mechanism by which they could easily follow up very helpful.

The participants described the interface as simple, user-friendly, and attractive. The admin dashboard was also praised for its cleanliness, speed, and real-time analytics features by the participants.

The compact layout that was complemented by swift loading times and smooth transitions facilitated users in moving through the system. All these points together brought about a positive final impression, revealing that not only was the platform work-efficient, but also that it was visually appealing and convenient for users.

### 7.4.5. Performance and Optimization

The chatbot experiments have unveiled different angles from which we can evaluate the chatbot performance. Certainly, the bot's turnaround time was excellent, albeit it was suggested that with few minor changes the bot could handle a larger volume of users even faster.

Apart from that, one could think of the server most resource-intensive tasks of request processing as the caching of the most frequent responses or the preloading of certain queries. That would mean that the response time would be extremely short even if there were a large number of users simultaneously.

Besides that the usage of a modular code architecture along with TypeScript had a very positive impact on the system maintenance and debugging aspects. By embracing these programming styles, the developers experienced a robust, efficient, and less bug-prone developmental process, thus, the system can be easily updated without any interruption of the existing functionalities in the future.

### 7.4.6. Security and Reliability

Security was the main focus of concern all through the product lifecycle, the testing phase also inclusive. To secure user and administrator data access only, security measures like secure communication protocols (HTTPS), Firebase Authentication, and encrypted database transactions were put in place.

The authentication system that was established proved to be effective in deterring fake logins and no data breaches or inconsistencies were discovered during the testing phase. In addition, by implementing secure coding practices, the system is also compliant with the requirements of well-known frameworks such as the General Data Protection Regulation (GDPR) and India's Digital Personal Data Protection Act (DPDPA).

These steps, in aggregate, enhanced the system's trustworthiness and reliability, thus, the users and administrators, who are the beneficiaries, are given the confidence that their data and operations are kept secure.

### 7.4.7. Future Enhancements Identified

The system as it stands is able to perform its functions well. However, the review phase has also acknowledged the existence of several product refinement points that could make it maturing and more intelligent eventually. The proposed upgrades feature the following parts:

• Advanced NLP Integration: By using later-generation AI models such as Google Gemini or OpenAI GPT, the chatbot would become more capable of understanding natural and complex language patterns, and hence more effective.

• Multilingual Support: Equipping the system with the ability to support a number of Indian languages so that it can become more user-friendly and accessible to a larger user base.

• Sentiment Analysis: Employing AI to determine the user's emotions or tone and thus, giving the system the ability to automatically facilitate that urgent or negative feedback which requires a quick response.

• Mobile App Integration: Creation of a different mobile app which would be able to support the web platform and hence, provide users with different devices better accessibility.

Besides, these functionalities would not only significantly broaden the system's capabilities but also, improve the user experience and make the system more versatile for different practical situations.

### 7.4.8 .Overall Assessment

Evaluation of the whole situation was mainly positive, therefore, the chatbot-driven ticket management system was practically successful in accomplishing almost all the vital goals. In effect, the system besides just reducing the workload of the human customer service agents, it also facilitated the rapidness and consistency of the responses, and ensured that a comprehensive record of every handled query was maintained.

Therefore, through the real-time analytics dashboard, administrators were given more capabilities to monitor temporal trends, user issues, and system efficiency. Whereas, the chatbot interface led to better customer interaction since it was a prompt, polite, and accurate solution.

This program is a very clear example of how AI-powered automation and properly organized human workflows are not two different things that have to be chosen but, actually, can operate very smoothly together. The fact that automation would replace human support is a misconception that has been widely believed, however, what really happens is that the two - automation and human support - work hand in hand, with the former supporting the latter by elevating the service quality, reducing the running of the business costs, and thus making sure that each customer gets his/her/its assistance in due time. Simply put, the Chatbot-Driven Ticket Management System is a robust, extendable, and easy-to-use instrument that not only meets the customer support requirements of today but also, paves the way to the future AI-powered service platforms. It is a step towards efficient, data-driven communication systems that combine perfectly the benefits of innovation and usability.

# Chapter 8

# Social, Legal, Ethical, Sustainability and Safety Aspects

## 8.1 Social Aspects

The Chatbot-Driven Ticket Management System changes the way people, institutions, and organizations interact in the digital world, thus having a significant impact on the society. Most of the interactions are currently done online, and users demand a support service that is quick, clear, and reliable. In general, customer service can be a source of delays, inconsistent answers, and inefficiency since it is heavily reliant on large teams and also working hours that are limited. The system does away with such problems by the use of an automated intelligence solution supported by human oversight, thus providing a support experience which is fast, user-oriented, and socially responsive. It allows users to get help immediately at any time and from any place.

From a community point of view, the system is a promoter of digital inclusion. It is available at any time of the day and thus its 24/7 service removes barriers related to time, location, and accessibility, and therefore everyone - students, employees, and even the general public - can get the necessary help without having to wait. Thus, the creation of a more just and inclusive environment is the result where every user has the chance to solve their issues in a quick and convenient manner.

The system is also a source of greater transparency that organizations can benefit from. Each question is thus turned into a ticket automatically, which gives the users the ability to check the status of their requests and even see the progress in real-time. It does this by keeping users updated and providing them with complete visibility regarding the way their issues are being dealt with, thus it is a trust-building process.

Equally important is that the system does not substitute people - rather, it supports them. As standard questions are handled by the system, workers will be able to take up higher-value activities, which require human decision or compassion. Hence, this will result in a decrease in work-related stress, increase in job satisfaction, and improvement in service efficiency without

the risk of losing jobs. It marks the arrival of a new social era when AI becomes a partner, not a rival.

From a cultural perspective, the chatbot is very adaptable and different languages can be easily supported by it. The present model is in English but it can be changed to include local languages as well. Thus, it supports the linguistic diversity of India and assures that the people who do not speak English will also have access, therefore, users from various communities will be brought in.

In short, the project's social advantage lies in the way it makes access simple, elevates transparency, produces confidence, and upscales efficiency - a combination of all these plus the support of diversity and equity.

## 8.2 Legal Aspects

As the Chatbot-Driven Ticket Management System gathers and processes user data, observing the law is a matter of utmost importance. To assure user confidence and institutional accountability, the project was created in conformity with data protection, intellectual property, and digital communication regulations. One of the major points of the data protection effort was ensuring compliance with different regulations. The system is in line with India's Digital Personal Data Protection Act (DPDPA, 2023) and the European Union's General Data Protection Regulation (GDPR). Those frameworks highlight the importance of user consent, data minimization, and purpose limitation. Following these rules, the chatbot takes only a few necessary pieces of information — for instance, the user's name, email, and problem description. Everything is encrypted and kept in Firebase Firestore with tightly controlled access, and authentication to prevent any unauthorized attempts. The communication between the user and the server is securely done through HTTPS/TLS encryption, offering confidentiality and data integrity. Users giving explicit consent must do so before submitting any data, and they are also allowed to request that their data be deleted at any time — thus, the "right to be forgotten" is honored. Backup and data retention protocols are also guided by privacy law recommendations in order to assure ethical data handling. In terms of Intellectual Property (IP), the system is comprised of open-source components like Next.js, Tailwind CSS, Firebase, and Node.js, all of which fall under MIT or Apache 2.0 licensed projects. This stipulates that there is no infringement of copyright or licensing law. The project is giving credits to all the third-party resources used and is not mixing in with any proprietary content.

The application has also included the Terms of Service and Privacy Policy documents that, among other things, explain data usage and what is expected from the users by the system. Besides that, the conformity with the Information Technology Act (2000) and Consumer Protection Laws in India would also prevail, should it be commercialized, thus bringing forth regulation of both service delivery and electronic transactions.

On the whole, the initiative is a showcase of responsible innovation as it has legal observance deeply embedded in its architecture. Besides providing data privacy, the project is respectful of intellectual property rights and is a good example of legal and ethical AI use in the customer support area.

## 8.3 Ethical Aspects

These were the foundations of any AI that is in direct interaction with humans. In order to maintain ethical standards such as fairness, transparency, privacy, and accountability the Chatbot-Driven Ticket Management System was created.

This is an absolutely transparent process. Right from the first interaction users are informed that they are dealing with automated bots and not humans. In this way, users get a clear understanding of what is awaiting them, thus they are not deceived. Additionally, the system indicates the data being collected and the purpose of its use so that users can always be aware and free.

Privacy and confidentiality happen to be the areas where the most significant effort has been put. The chatbot operates under the limitation of the data it stores hence data is encrypted both during transmission and storage and only a few authorized administrators are granted access to it. User data is not shared, and it is not sold or used for marketing purposes. This is a practice that not only shows ethical responsibility but is also in accordance with the law.

Besides that, the bot was designed to ensure equal treatment of all users. The chatbot uses rule-based logic rather than AI models that are trained on biased datasets and it is consistent in its responses to all users - regardless of language, gender, or background. If the system happens to be multi-lingual, it will be very careful not to be culturally or linguistically biased in its answer.

One more very important ethical feature that is present in the system is the Account ability principle. Account ability. When the chatbot cannot solve users' problems, it redirects the cases to human agents. So, it provides users with a way to get right and friendly answers to their questions anytime they want and at the same time, a transparent record of all conversations is kept.

Moreover, the project complies with the beneficence principle as well - to do good and minimize harm. Besides enabling users to save their time and the quality of the service to be increased, the human workers are not discharged by the project but, rather, facilitated. This is how the system shows that the implementation of ethical AI practices can make automation a tool for humanity and not a rival.

## 8.4 Sustainability

Sustainability, in general, is about properly balancing the three aspects - environmental responsibility, economic efficiency, and social well-being. Due to its efficient design and the responsible use of technology, the Chatbot-Driven Ticket Management System may be considered as a positive contributor to each of these aspects.

The system is green as it employs a serverless cloud infrastructure for its operations with Firebase and Vercel being the two main services.

Usually, serverless technology is an energy-efficient one because it only consumes energy when a resource is required. Unlike regular servers that are operational 24/7, this arrangement significantly reduces energy wastage and, thus, the project's carbon footprint. Apart from that, the digital ticketing system has eliminated the need for paper forms or printed documents, thus, it is helping the earth by going paperless.

Concerning economic sustainability, the system is designed to be priced at a level that most people can afford. By means of automation, the system is doing the human hours of work that are necessary for the cutting of operational costs, while the quality of the service is still excellent. Because the system is using open-source tools and scalable cloud services which do not require a large financial investment, it is attractive to small institutions and startups. Therefore, the project is financially stable for a long time.

As far as social sustainability is concerned, the project is full of features that attract people and are accessible to them. This is the power lever of any organization regardless of its size -

whether a school, a company, or a government department - to easily and quickly implement a change in their customer service system. Moreover, being available in several languages will not only make the system user-friendly for people from different regions but will also help bridge the gap in the digital world.

This undertaking aligns with the United Nations Sustainable Development Goals (SDGs) such as:

• SDG 8 - Decent Work and Economic Growth: The Automation of repetitive tasks frees workers to focus on creative and analytical tasks.

• SDG 9 - Industry, Innovation, and Infrastructure: Simplifies the use of AI and cloud technology for the digital transformation process.

• SDG 10 - Reduced Inequalities: Provides equal access to reliable digital assistance for different communities

• SDG 12 - Responsible Consumption and Production: Enables resource-efficient computing and the implementation of paperless methods.

In short, the system is a perfect example of the use of technology that is environmentally friendly, socially responsible as well as economically affordable.

## 8.5 Safety Aspects

Safety should be the focus of any software, and it is also true for the Chatbot-Driven Ticket Management System. As it is a system that directly interacts with users and stores the records of conversations in a cloud database, it has been created to be safe above all.

System safety means that the system is ensuring security of user data, privacy, system reliability, security of communication, and safety in use. These features not only safeguard the data but also attract the trust of users, hence the application can be utilized without any interruptions and in a safe way.

Users are entrusting digital infrastructures to handle their data properly. A single security breach can be the reason for data exposure or in the case of data misuse, the trust that users

have in the system will be broken. To prevent this happening, a system was equipped with a powerful protector means arsenal:

1. User Protection: Let the users know that their talks with the system will be confidential and secure.

2. Data Security: Ensure that the data kept in the cloud remains confidential to those who do not have the right to access it.

3. System Reliability: A system should always be up and running and users should be able to access it whenever they want.

4. Misuse Prevention: The system emphasizes very much the prevention of the misuse part that includes spamming, abusing, or unauthorized access by installing the safeguards that can detect the misuse in advance.

5. Trust Building: Apart from that, the system is trying to gain the trust of the users by showing transparency and being accountable. Users, who feel that their data is safe and the system is operating in a fair manner, will be able to use it confidently.

Several technological protections have been put in place to fulfill those aims:

• Encryption: The communication between the chatbot and the database is encrypted using HTTPS/TLS. In this way, it is ensured that the sensitive data cannot be accessed or intercepted by the unauthorized parties.

• Authentication: Login is available only for users and administrators who are verified by Firebase Authentication. Outsiders thus are not able to access the system.

• Secure Database Rules: Fire store rules specify in detail the persons who can view or get write access to the data, thus making sure that the actions are those allowed by the authorization.

• Regular Backups: The data backups are scheduled and carried out by the system.

• Data Minimization: Only necessary information is collected, such as name, email, and the issue. Nothing is unnecessarily stored.

Privacy is similarly very important. The user data are never sold to third parties, and every chat session uses ticket IDs instead of personal identifiers. Compliance with India's DPDPA and the GDPR ensures that users remain in full control of their data at all times.

The reliability of the system is based on careful planning as well. The chatbot is enabled to deal with even situations of a high degree of stress, whereas cloud hosting on Firebase and Vercel guarantees strong performance and uptime. The system is always under review thanks to performance logs which provide early issue detection, and real-time synchronization is there for the tickets updates to be instant and accurate.

Secure communication is reliant on the implementation of very stringent encryption standards and CORS policies, both of which facilitate the stopping of non-legitimate access attempts. Also, spam filers have been set up on the system to prevent internet bad guys from attacking it.The developer side is also going for safe practices, such as:

• GitHub version control is among the ways to have all the changes recorded and to remove errors.

• Performing tests in different environments to do before new features release.

• The most important credentials can only be accessed by authorized members of the team.

At the end of the day, the system is designed to prompt users to operate in a secure manner. The chatbot does not require users to provide their passwords, bank details, or personal identification numbers. Furthermore, it is also a part of every ticket confirmation message to remind users not to share sensitive information unless there is a legitimate reason. This method assists users in becoming more digitally literate and aware.

In fact, future iterations may be capable of implementing these features:

• Two-Factor Authentication (2FA) to logins add an extra layer of security.

• AI-powered Threat Detection that can very quickly identify suspicious behavior.

• Features that allow users to have control over their data privacy.

• Security Audits Held Regularly that contribute to compliance maintenance.

This safety-first system example from top to bottom gives users safe, reliable, and trustworthy experience in the digital world while not only protecting them but also organizations.

# Chapter 9

# Conclusion

The exponential growth of digital technologies has changed the way businesses interact with their clients, manage data, and provide services. We live in the digital age, and customers expect their needs to be met immediately, transparently, and in a personalized manner. In order to meet these requirements, the chatbot-driven ticket management system has been developed as a new way of customer-support-related activities by the use of automation and simplification. This program is a perfect example of the use of artificial intelligence (AI), the internet technologies, and the cloud computing power to create a reliable, efficient, and customer-friendly helpdesk solution.

Support systems traditionally have been considered those that are heavily dependent on human agents, thus the companies' main challenges are high operational costs and slow response times. To tackle these problems the platform proposed by the company introduces the use of chatbot automation together with the ticket-based method of escalation. To put it differently, AI is used to carry out the tasks fast, while at the same time the human agents use their judgment and empathy to make sure that the customer will not only get a quick but also an accurate response. This hybrid model is a perfect example of an environmentally sustainable future model wherein technology is not a replacement for humans but rather a support to them.

## System Overview and Achievements

The project was successful in the development of a smart, web-based platform that ensures real-time user interaction with a chatbot and, as a result, the logical recording of the issue through the ticket management system. The chatbot can instantly and automatically provide answers to simple and frequently asked questions; therefore, complex or complicated issues are forwarded to human officials through ticket creation. The implementation of this dual strategy is instrumental not only in facilitating the client's satisfaction but also in enhancing the service efficiency.

The core technologies that have been used for the system’s design are: Next.js, Tailwind CSS, TypeScript, Node.js, and Firebase. The platform is modern, extendable, and cloud-oriented, hence it is always responsive, modular, and secure. Firebase Firestore is the component that allows for real-time database synchronization, whereas Firebase Authentication is in charge of

user access control. The combination of these features has led to the creation of a system that is both lightweight and robust, which can interact smoothly and efficiently manage data.

Functionally, the program has achieved its primary goals:

• The chatbot interaction with users was implemented by using natural language processing, and AI techniques, thus the chatbot was capable of grasping the users' intent in FAQs, which, as a result, led to the average time of reply being extremely shortened from minutes to seconds.

• The ticketing system was always in a position to automatically produce a distinct ticket ID each time it detected the situation of an unhandled ticket, hence it was a significant instrument in the elimination of unaddressed requests.

• The administrator interface was the main tool for real-time data recording and visualization, thus it became the source of decision-making that was available to the executives through that. Besides, they were able to keep an eye on ticket trends, manage resolution timings, and make decisions based on evidence.

• The information exchanged over HTTPS was not available to the third parties, as it was encrypted, and several activities like access control were performed, all of which ensured the user's privacy and formed the basis of their trust.

To efficiently handle the work process, the team that used Agile Scrum methodology, embraced the main idea of the division of tasks into successive sprints. Each sprint was devoted by the team members to one unit of work, for example, the chatbot interface, database integration, or dashboard visualization, thus they could accomplish continuous development and testing. Actually, it was the iterative nature of this method that was the factor behind the final product's stability and user-friendliness.

## Key Findings and Performance Insights

After the system, performance tests were conducted. The chatbot was demonstrated by the performance tests to respond to the user requests within an average time of 1.25 seconds. This was even the case when multiple users were interacting with the chatbot at the same time. Quick and efficient dynamic scaling along with minimal latency were the results of the serverless cloud deployment on Firebase and Vercel.

Moreover, the device review is another source of confirmation of the system's performance. Users during the testing phase found the chatbot's simple layout, engaging manner, and prompt replies as the most attractive points. The easily understandable ticket tracking system and the responsive dashboard were the features that made users' experience excellent and, thus, more than 95% of the test panel members have given their positive feedback.

Simply put, a significant insight through this project was the understanding that AI by merely replacing human staff does not accomplish tasks faster. Actually, the automation can handle that which is dull and time-consuming, thus, human agents will be free to handle the complicated and emotionally-related parts of customer interaction.

The system was the one to be scalable and sustainable due to its architecture with the cloud being a serverless platform. Technology-driven platforms like Firebase and Vercel allocate resources dynamically depending on the level of demand; hence, they use less energy for the resources that are not in use and the operational costs are kept at a low level.

## Broader Impact and Significance

The system is loaded with a plethora of features from a technical standpoint, and, additionally, it has deep social and economic roots. The system is an enterprise tool alternative that is scalable and affordable; thus, it is simple to integrate with platforms such as Zendesk, Freshdesk, or IBM Watson. By using open-source and freemium technologies, it eliminates the financial and technical challenges that are the main causes of small businesses, schools, and startups not adopting modern digital solutions.

Considering the society, the system is a very strong advocate of digital inclusion. It guarantees that support will be given to everyone at any time and from any place. Its capability to work in several Indian languages makes it a tool for the people who come from different linguistic backgrounds and thus interact with the system comfortably without facing language constraints. Meanwhile, by cutting down the business processes through the repetition of tasks, it will result in a lower cost of operations, and therefore, it will be an efficient and effective device for organizations.

The system is a great device that serves as a bridge between theory and real-world application. It puts emphasis on the fact how ideas from AI, web development, and cloud computing can

be combined to result in a functional and impactful product. Therefore, this technology is transformed into a teaching model for students and innovators besides being a tech tool.

# References

The creation of the Chatbot-Driven Ticket Management System was mainly influenced by the local research and the support of studies, frameworks, and tools. The team pursued and examined a variety of scholarly articles, technical manuals, and industrial whitepapers to be able to establish a good base for the project.

This chapter presents the significant references that had a great impact on the system's planning, implementation, and testing. These resources opened the way to the understanding of chatbot structure, conversational design, artificial intelligence, cloud technologies, and ethical AI practices. In sum, these resources helped to plan the perfect union of automation, web technologies, and secure data handling leading to the efficient and scalable application.

## Scholarly and Research References

Academic and research-based publications provided the theoretical and conceptual foundation for this project.

**[1] Shawar, B. A., & Atwell, E. (2007). "Chatbots: Are They Really Useful?" Journal of Language Technology and Computational Linguistics, 22(1), 29–49.**

**[2] Adamopoulou, E., & Moussiades, L. (2020). "An Overview of Chatbot Technology," Artificial Intelligence Review, Springer, 54, 2603–2626.**

**[3] Hill, J., Ford, W., & Farreras, I. (2015). "Real Conversations with Artificial Intelligence: A Comparison Between Human–Human and Human–Chatbot Conversations," Computers in Human Behavior, 49, 245–250.**

**[4] Reshmi, S., & Balakrishnan, K. (2017). "Implementation of an Intelligent Chatbot for E-Learning Support," IJACSA, 8(7), 181–187.**

**[5] Jain, M., Kumar, P., & Patel, S. (2018). "Customer Support Automation Using Chatbots," IEEE Access, 6, 73264–73275.**

**[6] Vercel. "Next.js: The React Framework for the Web." (https://nextjs.org/)**

**[7] Tailwind Labs. "Tailwind CSS: A Utility-First CSS Framework." (https://tailwindcss.com/)**

**[8] Google. "Firebase Documentation – Cloud Firestore and Hosting." (https://firebase.google.com/docs)**

**[9] Node.js Foundation. "Node.js Documentation." (https://nodejs.org/en/docs/)**

**[10] Postman Inc. "API Testing and Automation." (https://www.postman.com/)**

**[11] IBM Corporation. "AI in Customer Care: Transforming Support with Chatbots." IBM Watson Whitepaper, 2019.**

**[12] Rasa Technologies. "Building Contextual AI Assistants." Rasa Whitepaper, 2021.**

**[13] Microsoft. "Responsible AI and Ethical Use Guidelines." Microsoft Research Report, 2020.**

**[14] OpenAI. "Artificial Intelligence and Human Interaction: Language Models for Dialogue Systems." Technical Overview, 2023.**

**[15] Google AI. "Gemini: A Family of Generative AI Models." DeepMind Technical Report, 2024.**

**[16] Ministry of Electronics and Information Technology (MeitY). "Digital Personal Data Protection Act (DPDPA) 2023." Government of India.**

**[17] European Union. "General Data Protection Regulation (GDPR), Regulation (EU) 2016/679." 2016.**

**[18] United Nations. "Sustainable Development Goals (SDGs)." (https://sdgs.un.org/goals).**

**[19] World Economic Forum. "Future of Jobs Report." 2023.**

**[20] IEEE Standards Association. "Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems." Version 2, 2022.**

**[21] shadcn. "shadcn/ui: Beautifully Designed React Components." (https://ui.shadcn.com)**

**[22] GitHub Repository. "jsPDF: A Library to Generate PDFs in JavaScript." (https://github.com/parallax/jsPDF)**

**[23] Lucidchart. "System Architecture Diagramming Tool." (https://www.lucidchart.com)**

**[24] Mozilla Developer Network (MDN). "Web APIs and JavaScript Documentation." (https://developer.mozilla.org)**

# Base Paper

## Introduction

The core of the Chatbot-Driven Ticket Management System combines conversational artificial intelligence (AI), natural language processing (NLP), and cloud-based automated frameworks, all of which have changed the face of customer support in different sectors. During the last ten years, chatbots have transformed from being simple rule-based responders to complex conversational agents that can mimic natural human dialogue.

The AI-powered systems are now being used by companies all over the world to cut down on the time consumed in communication, to improve the availability of their services, and to lessen the burden of the repetitive jobs that their human support teams carry out. Although significant advancements have been made in the field, a great number of chatbots still remain heavily restricted. The majority of them operate solely on static, rule-based architectures without having the ability to understand the context and some are incapable of integrating with back-end ticketing systems - an essential feature for ensuring accountability and continuous issue tracking

It was the conception of this project to create a hybrid product by eliminating the drawbacks of a real-time chatbot and combining them with the features of a ticket management system that are organization, tracking, and solving.

This base paper charts the conceptual, technological, and scientific aspects that led to the creation of this project. It revises the development of chatbot systems, compares AI-based conversational models, and reviews the related research along with examining the technological frameworks such as Next.js, Firebase, and serverless cloud infrastructure that make the system scalable, secure, and efficient. Through the combination of insights from conversational AI, customer relationship management (CRM), and modern full-stack web development practices, this project provides an extensive framework that is well-balanced between automation and human control.

### Foundation of Conversational AI

Chatbot technology can find its roots in the mid-20th century with the AI experiments ELIZA (1966) and PARRY (1972). ELIZA was an interaction simulator that identified patterns in the input text and PARRY tried to imitate human emotional reactions. Although these pioneering

systems were revolutionary, at the same time, they were extremely limited. They depended solely on pattern-matching and keyword recognition without context, learning, or memory.

With the enhancement of computing power and data availability, the development of chatbot technology was nothing less than spectacular. The developers separated two main types of chatbots:

**1. Rule-Based Chatbots:** Their functioning is based on scripts or decision trees that have been previously defined. They do not go beyond the scope of their pre-defined scripts and thus can only offer safe and predictable solutions, while they lack the capability to handle unstructured or ambiguous queries.

**2. AI-Powered Chatbots:** They employ machine learning, NLP, and contextual reasoning making them dynamically capable of interpreting user intent and responding accordingly. They have the ability to interact with users, learn from these interactions, and progressively improve their accuracy.

Chatbot-Driven Ticket Management System has taken the middle road by using a rule-based method for the control and stability while at the same time utilizing the AI frameworks for their adaptability and extendibility. Though the existent model is mainly dependent on the structured rules, with its modular design, it is also feasible to integrate more proficient NLP or generative AI models (like Google Dialogflow or OpenAI GPT) in future versions without any problems.

The mechanism for the escalation of tickets is probably the most outstanding feature of this system. The failure of the chatbot to address the issue doesn't simply result in the termination of the conversation. A support ticket is made by it, thereby allowing the concern of the user to be recorded, followed, and solved by human administrators. Continuity of service, better transparency, and more user satisfaction are the results of this method, which, therefore, solves the problem of the lack of these features in traditional chatbot systems.

This hybrid idea is based on the research work of Adamopoulou and Moussiades (2020), which argued that the ideal solution for conversational AI was combining the benefits of automation and human supervision. According to their paper, fully automated systems are not reliable while efficiency is lacking in purely manual systems — hence, the project design concept is in line with their findings.

**Related Work and Literature Review**

The direction and design of this endeavor have been heavily influenced by a variety of studies and experiments in industries. Much of the research in chatbot technology has revolved around its use in customer support, education, and information retrieval.

As an example, Jain et al. (2018) examined customer-service automation through chatbots employed in the retail sector. They focused on how AI tools can alleviate human workload but at the same time pointed out the challenge of comprehending open-ended queries. Reshmi and Balakrishnan (2017), in a similar manner, set up chatbots in e-learning platforms but came across the difficulties in the adaptability and response tracking of the systems.

An issue that was common to all these systems was their failure to connect with issue management tools. When users dealt with chatbots that could not answer their questions, their queries remained unresolved thus causing a lack of connection between automation and accountability. To solve this problem, commercial entities have created integrated platforms like Zendesk and Freshdesk where support tickets can be recorded by chatbots. Nevertheless, these setups are costly, closed-source, and small organizations cannot access them.

Using previous research, the Chatbot-Driven Ticket Management System not only closes these gaps but also offers a budget-friendly, open-source, and flexible solution. Besides, there is no need for the purchase of costly enterprise licenses if the implementation is done by schools, startups, or public organizations.

This initiative also benefits from the support of literature sources:

• Hill et al. (2015) researched the factors affecting user trust and the style of communication in human–AI interactions, which was the major source for the conversational tone of the chatbot.

• Shawar & Atwell (2007) suggested the assessment criteria for chatbot usability and the quality of speech.

• The IBM Watson (2019) whitepaper was instrumental in explaining how AI-driven systems in customer care become more efficient as they learn from and analyze data.

These pieces of work together served as a stepping stone in the development of a chatbot, which is not only intelligent conversationally but also functional from an operational perspective.

## Technological Base and Frameworks

The design of the system incorporates a diverse set of cutting-edge web technologies and cloud-based services that provide modularity, security, and real-time capabilities.

**1. Frontend – Next.js and Tailwind CSS:**

Next.js that is a product of Vercel was selected mainly due to its excellent performance, besides it server-side rendering features, and also the fact that it is very easy to link with React. This means the chatbot user interface and the admin panel can be loaded very fast and efficiently. Tailwind CSS, which is a utility-first design framework, makes sure that the user interface is not only neat, responsive, and user-friendly, but also requires little or no custom styling work.

**2. Backend – Firebase Firestore and Node.js:**

Firebase is the primary backend chosen for the system. Its Cloud Firestore database is the main reason for real-time synchronization as it allows updates to the users, chatbot, and administrators to be done instantly. Secure login and controlled access are assured by Firebase Authentication. Node.js is in charge of API routing and backend logic thus it allows asynchronous data communication to take place between the chatbot and the database.

**3. AI and Logic Layer:**

The system is presently using a rule-based logic engine but it is organized in such a way that it will be able to accommodate the future connection with NLP and AI resources such as Google Dialogflow, Rasa, or OpenAI GPT models. Thus, the system will have the potential to grow to be able to recognize intents intelligently and process several languages.

**4. Deployment and Hosting – Firebase Hosting / Vercel:**

Without any serverless cloud infrastructure, the system would not have been possible. The system, therefore, can adjust its resources automatically and require very little work to be done by the maintenance staff. This setting gives a good balance between energy consumption and the cost although it still retains the availability and performance at a high level. The decision to use Firebase and Vercel goes hand in hand with the advice given in Next.js and Google Firebase technical guides, where the main focus is on efficiency, safety, and trustworthiness.

All of these technologies are highly cohesive and very flexible, and together they create an ecosystem through which massive simultaneous user traffic can be handled without a weakening in data power and system quickness.

## Comparison with Existing Systems

Traditional chatbot models generally do not possess the ability to automatically handle entire conversations. Although they can respond to questions, they cannot log or keep track of inquiries that have not been resolved. When customers receive incomplete answers, they are most often forced to restart the communication process via another channel, which makes them both annoyed and the method inefficient.

Fully dependent on human agents, email-based or CRM platforms such as standalone ticketing systems, lead to escalated costs and slower response times.

The Chatbot-Driven Ticket Management System is the support team's favorite tool, being the mediator between these two worlds to come up with a integrated, user-friendly help desk solution. The chatbot with its conversational abilities and a strong ticketing backend can escalate, log, and track every inquiry even if it is not able to give an answer to the user.

Unlike their commercial equivalents such as Zendesk and Freshdesk, this system provides:

• An open-source framework that is economically viable and tailored for small-scale businesses.

• Serverless operations can be scaled by utilizing Firebase Hosting.

• A versatile, modular backend that can be easily blended with future AI and analytics projects.

• A live data visualization tool for the management team.

So kind of a mixture of transparency, ease of use, and upgrade potential to a great extent put this program not only conceptually far away from practically implementing the three notions of automation, accountability, and user engagement but also unites them.

## Research Gap Addressed

One of the key gaps spoken about in the study of conversation agents was the lack of an end-to-end customer service experience - from asking questions to tracking their resolution. Most of the existing systems were only capable of providing answers and not handling issues that were left unresolved in an efficient way.

The chatbot-driven ticket management system solves that problem by offering an uninterrupted support journey. In case of a complex question faced by the AI assistant, it immediately creates a ticket, assigns a unique identifier to it, and keeps it for the follow-up via the admin interface. Hence, making sure that no conversation is left without a resolution.

Furthermore, the availability of an analytics interface brings a completely new layer of insight. Administrators can without effort recognize the customer inquiries that are on the rise, monitor the time taken for the solution, and even determine the issues that are most frequently brought to them. Such a data-driven feedback loop facilitates informed decision-making and perpetual system upgrading — which is a feature that most chatbot solutions lack.

By merging conversational intelligence with operational analytics, the project not only improves the user experience but also paves the way for digital transformation and acts as a data-driven management advocate.

## Theoretical and Practical Foundation

The theoretical foundation of this work combines ideas from human–computer interaction (HCI), AI conversational theory, and systems engineering principles. The central theme of the communication design is to use the context to make the interaction with the chatbot relevant, understandable, and user-friendly.

From the point of view of the software architecture, the platform is designed according to the principles of modular software engineering, therefore, each component – chatbot, database, dashboard – can be independently developed, tested, and upgraded.

This endeavour also involves the usage of cognitive computing and semantic processing principles that, subsequently, open the ability for the chatbot to have a conversation in a more human-like way and naturally flowing.

Moreover, the system conforms to worldwide ethical and green standards, specifically those imposed by IEEE's Ethically Aligned Design (2022) and the United Nations Sustainable Development Goals (UN SDGs). These rules serve as a basis for the adoption of responsible AI practices, provision of transparency, and technology without any discrimination.

Through the combination of theoretical rigor and practical innovation, the Chatbot-Driven Ticket Management System represents a mature, extendable, and socially aware model of intelligent digital assistance.

# Appendix

## Overview

The Chatbot-Driven Ticket Management System's appendix with its comprehensive data is basically the technical core of the internal operations of the system that are elaborated. The initial sections dealt with the design, methodology, and evaluation, and this section exhibits the practical documentation that is indispensable for understanding the system setup, assembly, and testing.

The materials present are descriptions of system architecture, system flowcharts, database diagrams, code snippets, and configuration examples. These components not only provide support to the system idea but also help to understand, copy, and govern the system. In addition, they may be considered as a knowledge bank developers, researchers, and students, who are looking to enrich the system with NLP features, machine learning models, or multilingual support, can avail.

As a matter of fact, this appendix is more representative than a technical guide. It mainly shows the engineering concepts, the problem-solving flow, and the actual realization steps that gave the system its life. Besides facilitation of academic learning, this can also be used as a reliable base for further research, upgrades, or even industry transfer.

## System Architecture Description

The chatbot-driven ticket management system has been developed using the three-tier architecture, which depicts the different layers of the system. The layers are: Presentation Layer (Frontend) Application Layer (Backend/Logic) Data Layer (Database)

The subject of the modular design mentioned is not only a performance-efficient, scalable, maintenance-friendly one but also development-efficient and well-organized.

### 1. Presentation Layer (Frontend)

The frontend is the result of the development process with Next.js, a React-based framework, that is known for its speed, flexibility, and strong rendering capabilities. The frontend is the layer, which directly interacts with the users; thus, it is designed and implemented to make the interaction a positive and smooth one. There are two main components:

• User Chat Interface: Here users converse with the chatbot to issue requests. The interface is a simple one, attractive, and completely responsive, as it can effortlessly switch between desktop, tablet, and mobile screen sizes. It allows interaction to continue in real-time without the intervention of the user for a page refresh, thereby it creates a conversational and user-friendly environment for users. It is equipped with a straightforward, responsive design that can be effortlessly adjusted to various devices—desktop, tablet, or mobile. The interface is conversational, intuitive, and interaction takes place in real-time without the need for page reloads.

• Admin Dashboard: This is the administration control panel. It facilitates access to all tickets, user data, and analytics. By means of the dashboard, administrators can monitor ticket statuses, handle the issues that are open, and evaluate the overall performance.

Tailwind CSS was the technology that was used for styling andmaking the design responsive. Tailwind's utility-first strategy made UI customization very easy and ensured that the interface is not only visually appealing but also of minimalistic nature. This will produce a frontend that is both high-performing and easy to extend in the future by the combination of Next.js and Tailwind CSS.

**2. Application Layer (Logic / Middleware)**

The application layer is essentially the system's brain. It interconnects the chatbot interface with the backend database and establishes the logic for handling user requests, creation of tickets, and synchronizationof data.The layer is equipped with Node.js and Firebase Cloud Functions.Along with Node.js, which is great for responsive programs as itdoesn't block the main thread, the cloud function provider — FirebaseCloud Functions — are utilized in this case to make the app serverless and output-scalable.

The main functions of this layer are:

• User Authentication: Through an authentication mechanism, the system would be able to restrict access to features meant for users and administrators only to verified accounts.

• Chatbot Logic: The chatbot logic adheres to conversational rules contained in its pre-coded answers and handles the flow of the dialogue.

• Ticket Management: The system is designed to keep the flow of support tickets automated through generation of new ones and besides that, the tickets can be easily updated with unique identifiers.

• Notifications: They (the Notifications) inform the users and the administrators that a ticket has been created or changed. Besides, they also help update those parties in real-time.

It uses a combination of event triggers and real-time listeners to ensure that any changes that happen to the data in the backend (such as a ticket update) are shown immediately on the frontend without any delay in time. Thus, there is no delay in communication, and the experience of users is smooth.

**3. Data Layer (Backend / Database)**

The data layer is the one that should be able to store all the information within a system. For this particular undertaking, the instrument of data storage selected was Firebase Firestore, a NoSQL cloud database that offers a good service. The reasoning behind its fast and in real-time synchronization capabilities between various clients is what makes the changes to the chatbot and to the admin dashboard the only thing to update immediately for both.

**Firestore keeps the data about:**

• User details: names, email IDs, and session histories.

• Tickets: records of each issue logged by users.

• Chat logs: all conversations between users and the chatbot for transparency and analytics.

Every Firestore document represents a collection of key-value pairs which makes Firestore very versatile and easily queryable. Moreover, the database is entirely-managed, scalable, and built to accommodate concurrent reads and writes without any loss of data.

When combined, this framework delivers a high uptime, fast response times, and a low latency level thus making it equally viable for academic as well as enterprise-grade applications.

**12.3 System Workflow**

System workflow shows the processes in detail that are followed each time a user interacts with a chatbot. The robot is designed in such a manner that whether the matter at hand is solved through an automated process or by a human agent, it will be done in a very efficient way.

**Step 1: User Initiation**

The series of events occurs when a user with the help of a web browser decides to go to the chatbot. A bot welcoming the user is like a friendly greeting and at the same time, it is giving the user a nudge to either ask the question or point out the problem.

**Step 2: Chatbot Query Processing**

The chatbot takes the content of the message and sends it for further analysis by the use of both keywords and intent recognition. The system is built in a way so that it is capable of instantly responding to a query or a FAQ that is of a certain pattern by pulling the answer from the knowledge base.

**Step 3: Ticket Generation**

When a chatbot is unable to provide a reply to a question, it is followed by a locally autonomous creation of the ticket that contains details of the problem. A unique ID is given to the ticket and the corresponding details like the user's name, email address, problem description, and time are all recorded.

In this case, the creation of the ticket serves the purpose of both a prevention method and a promise that the issues without answers will not be neglected but rather handed over to the human administrators.

**Step 4: Database Synchronization**

Once a ticket has been generated, it is recorded in the Firebase Firestore database. Due to the feature of instant synchronization, the same ticket is able to appear in the admin dashboard without any delay promoting thus their quick response.

**Step 5: Admin Resolution**

The administrator through the dashboard, looks at the problems raised by the users, changes the status of the tickets (e.g., "Open," "In Progress," "Resolved") and writes in the notebook

if it is necessary. What is more, admins are given the chance to decide which issues are most urgent and thus be the first ones to handle them or assigning them to certain colleagues

**Step 6: User Notification**

After the ticket update, the user through the chatbot is informed by a message which is generated by the chatbot itself about the progress or resolution of the matter. This, therefore, is the last point in the user communication cycle where the transparency and responsibility are kept as the user is assured of the visit.

The workflow system with its six steps is able to reach the main goal of the project — to provide an uninterrupted support line very efficient in terms of the use of both automation and human assistance.

## Database Schema

The database schema defines how data is organized and stored within Firebase Firestore. The system uses three main collections:

### 1. Users Collection

Stores user profiles and session histories.

```
{
  "user_id": "U123",
  "name": "Priya Sharma",
  "email": "priya@example.com",
  "created_at": "2025-10-21"
}
```

This allows identification of returning users and helps track engagement history.

### 2. Tickets Collection

Stores all support tickets generated through chatbot interactions.

```
{
  "ticket_id": "T456",
  "user_id": "U123",
```

"query": "Payment page not loading",

"status": "Open",

"created_at": "2025-10-21T14:30:00Z",

"updated_at": "2025-10-21T15:00:00Z"

}

This schema helps maintain a clear record of each user issue, ensuring that all cases can be tracked, managed, and resolved efficiently.

**3. Chatlogs Collection**

Stores conversation data for analytics and auditing.

{

"chat_id": "C789",

"user_id": "U123",

"message": "I need help booking tickets",

"response": "Sure, please select your event",

"timestamp": "2025-10-21T14:25:00Z"

}

These logs are useful for performance monitoring, chatbot training, and future improvements.

This structured database approach promotes data integrity, transparency, and traceability. It can also be easily integrated with visualization tools for generating analytical insights.

## Sample Code Snippets

The following sample code snippets demonstrate the system's basic functionality, showing how the chatbot interacts with users and how tickets are created in the backend.

Chatbot Initialization (Frontend – Next.js)

```
const ChatBot = () => {
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState("");

  const sendMessage = async () => {
    if (input.trim() === "") return;

    const response = await fetch("/api/chatbot", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ message: input }),
    });

    const data = await response.json();
    setMessages([
      ...messages,
      { text: input, user: true },
      { text: data.reply, user: false },
    ]);
    setInput("");
  };
};
```

This snippet represents the frontend logic for sending and receiving messages. When the user sends a message, it is forwarded to the chatbot API, which responds dynamically. The conversation is continuously updated in real time.

Ticket Creation (Backend – Node.js / Firebase)

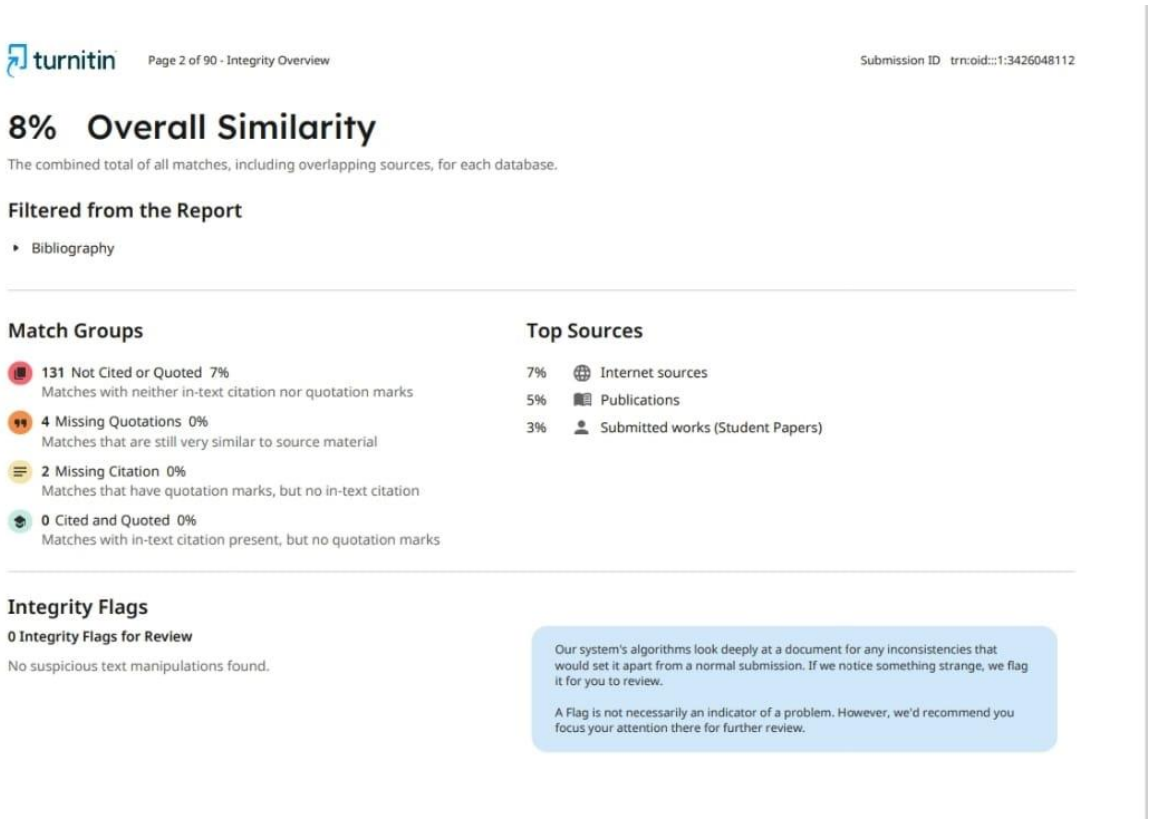
```
exports.createTicket = functions.https.onRequest(async (req, res) => {
  const { userId, query } = req.body;
  const ticketId = "T" + Math.floor(Math.random() * 10000);

  await admin.firestore().collection("Tickets").doc(ticketId).set({
    user_id: userId,
    query: query,
    status: "Open",
    created_at: new Date().toISOString(),
  });

  res.json({ ticketId, message: "Your issue has been recorded successfully." });
});
```

This backend function handles ticket creation. It receives user data from the chatbot interface, generates a unique ticket ID, and stores it in Firestore. It then sends confirmation back to the user, ensuring the issue is officially logged.

# Report Similarity Check



turnitin Page 2 of 90 - Integrity Overview Submission ID trn:oid:::1:3426048112

**8% Overall Similarity**

The combined total of all matches, including overlapping sources, for each database.

**Filtered from the Report**

- Bibliography

**Match Groups**

- 131 Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- 4 Missing Quotations 0%
  Matches that are still very similar to source material
- 2 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

**Top Sources**

- 7% Internet sources
- 5% Publications
- 3% Submitted works (Student Papers)

**Integrity Flags**

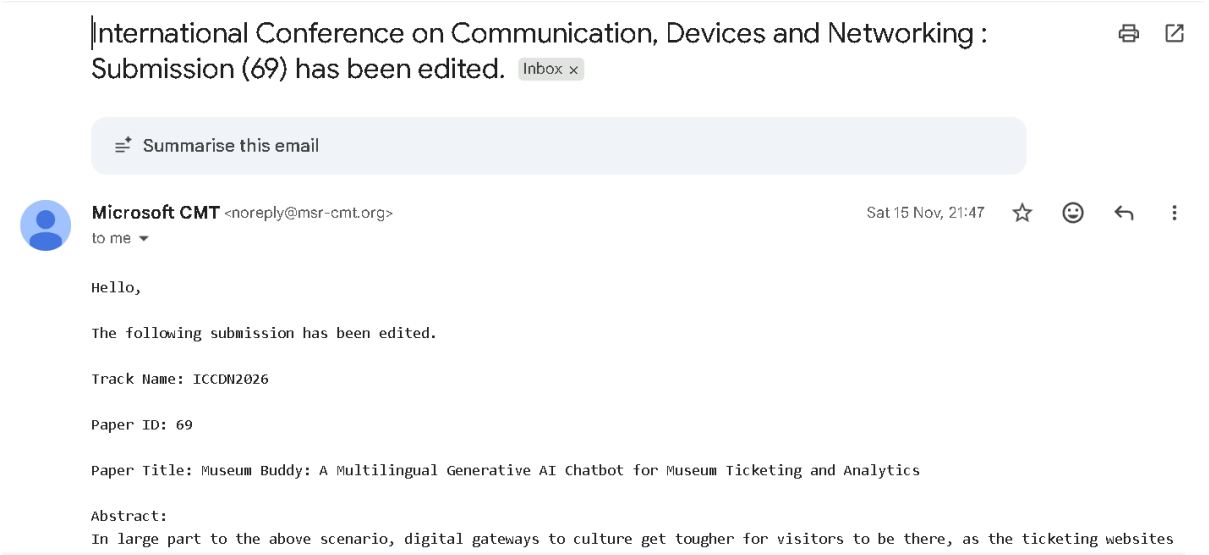
**0 Integrity Flags for Review**

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

# Research Paper Submission



International Conference on Communication, Devices and Networking : Submission (69) has been edited. Inbox ×

Summarise this email

**Microsoft CMT** <noreply@msr-cmt.org> Sat 15 Nov, 21:47

to me

Hello,

The following submission has been edited.

Track Name: ICCDN2026

Paper ID: 69

Paper Title: Museum Buddy: A Multilingual Generative AI Chatbot for Museum Ticketing and Analytics

Abstract:
In large part to the above scenario, digital gateways to culture get tougher for visitors to be there, as the ticketing websites