



International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Vaidyam: AI-Integrated Intelligent Voice-Enabled Healthcare Service and Management Platform

Mallikarjun C V¹*, Darshan C N¹, Manju H Venukanth¹, D G Rangaswamy¹, Kavyashree P N²

¹ Undergraduate Student, Department of Computer Science and Engineering, G. M. Institute of Technology, Davangere, Karnataka, India (affiliated to Visvesvaraya Technological University, Belagavi, Karnataka, India)

² Assistant Professor, Department of Computer Science and Engineering, G. M. Institute of Technology, Davangere, Karnataka, India

* e-mail: mallikarjuncv12@gmail.com

ABSTRACT:

Vaidyam is a web-based healthcare appointment and management platform that coordinates patients, doctors and administrators through three separate React/Vite portals backed by a shared service. It combines doctor discovery and filtering, appointment lifecycle management, consultation reports, ratings and reports, appointment-scoped chat, administrative moderation, and payment-flow endpoints. A bounded voice assistant supports English and ten specified Indian-language options. English audio is sent to Gemini for structured intent interpretation; for the ten Indian languages, audio is transcribed using Sarvam AI, interpreted into schema-constrained intent data by Gemini, and non-English replies can be spoken using Sarvam text-to-speech. Booking and cancellation actions require explicit confirmation in the voice workflow and are subsequently checked by authorized backend operations. The booking controller includes date/time and availability validation, conditional slot reservation, request-ID idempotency, partial unique indexes, transaction attempts with a fallback path, and real-time slot events. Evaluation of the supplied source snapshot included an executable in-memory booking simulation, which reported 29 passing assertions across ten scenarios. Because this harness uses a mock database and simulates reservation behavior, it does not demonstrate database-level concurrency safety or production reliability. A separate voice-safety test could not run because Mongoose was absent from the test environment. The paper describes the implementation and its current evidence limits; live service testing, speech accuracy, usability, and end-to-end payment validation remain future work.

Keywords: digital healthcare platform, healthcare appointment scheduling, voice user interface, Indian languages, conversational AI, role-based authorization, software verification

Introduction:

Digital health systems increasingly use connected software to support access, coordination and delivery of health-related services. The World Health Organization's global strategy positions digital health within broader health-system planning and governance rather than treating technology as an isolated intervention [1]. In appointment workflows, patients need to discover suitable practitioners, understand availability, select a time and follow the status of their booking. Providers need dependable availability management, and administrators need bounded oversight of accounts and activity.

Online appointment systems have been developed to move practitioner discovery and appointment management into web or mobile workflows [2], [3]. Scheduling research also examines optimization and simulation to make better use of limited clinical resources [4]. These goals depend on a consistent source of appointment state: a slot displayed as available in a browser can become unavailable before a booking request reaches the server, particularly when users submit duplicate or concurrent requests.

Conversational interfaces offer another access path for health-related services, but their scope and evaluation differ across applications. Reviews of healthcare conversational agents describe a varied body of work and identify limitations in evidence quality, usability assessment and reporting [5], [6]. Design-focused research emphasizes transparency in the interaction between people and automated systems [7], while patient-perspective research on digital healthcare platforms highlights that the value of remote-care technology depends on how patients experience the service [8]. Systems such as C-PATH focus on conversational patient assistance and triage [9]; Vaidyam instead concentrates on application navigation and transactional appointment tasks, and it is not a diagnostic system.

This paper presents Vaidyam as an integrated appointment and management platform with separate Patient, Doctor and Administrator portals, a task-oriented multilingual voice layer, and server-side appointment safeguards. The contribution is an implementation-level account of how these components are connected and bounded, together with a carefully scoped review of the available test evidence. The paper does not claim clinical effectiveness, language accuracy or production reliability without the corresponding experiments.

Problem Statement and Objectives

A healthcare appointment platform must coordinate at least three parties with different permissions. Patients need discoverability and transparent booking; doctors need control over availability and consultation records; administrators need oversight without mixing their privileges with patient or doctor sessions. Concurrent requests can target the same slot, stale browser state can misrepresent availability, and conversational intent may be ambiguous. The system must address these conditions with shared server-side validation while keeping common tasks understandable.

The objectives of Vaidyam are:

1. To provide separate, role-specific web portals for patients, doctors and administrators while reusing a common backend.
2. To let patients discover doctors, view availability, book or reschedule appointments, manage appointment records, and access consultation follow-up features.
3. To offer a contextual voice interface with English and ten supported Indian-language options for common navigation and appointment tasks.
4. To reduce double booking through input validation, atomic slot reservation, idempotent retries and database uniqueness constraints.
5. To support communication, reporting, moderation and audit operations with server-side authorization.

Methodology:

The study follows an implementation-oriented software review. The supplied Vaidyam Version 24 archive was inspected across the Patient, Doctor, Administrator and Backend directories. Relevant materials included package manifests, React/Vite application components, Express routes and controllers, Mongoose schemas, voice-processing providers, authorization middleware, Socket.IO handlers, and the included test scripts. The version label identifies the source snapshot examined in this evaluation; no Version 25 source archive was included among the files available for this review.

The booking simulation was executed from the backend directory using ``node tests/concurrentBookingTest.js``. Its output reported 29 passed assertions and zero failures across ten scenario groups. Inspection of the script shows that it runs against an in-memory mock state and contains local helper implementations that mirror selected booking checks; it does not import and exercise the production booking controller against a live MongoDB deployment. Consequently, the result is evidence about the behavior of the supplied simulation, not a production database concurrency benchmark.

The separate ``node tests/voiceSafetyTest.js`` command was also attempted. It stopped during module loading because the ``mongoose`` package was unavailable in the execution environment, before the test assertions ran. No live Gemini or Sarvam calls, live database integration, complete browser-level usability study, load test, or production payment transaction was executed as part of this review. Findings below distinguish source-code inspection from the mock-based test result.

System Architecture

The Patient, Doctor and Administrator interfaces are independent React applications built with Vite. They communicate with a shared Node.js and Express backend through REST endpoints. MongoDB, accessed through Mongoose models, persists user and doctor accounts, appointment records, booked and blocked slots, chat messages, reports, moderation actions, service-control settings and audit events. JWT-based middleware identifies the active role before protected operations are processed. Socket.IO provides a real-time channel for authorized appointment chat and doctor slot-availability events. External services are called from the backend so provider credentials need not be embedded in the browser source.

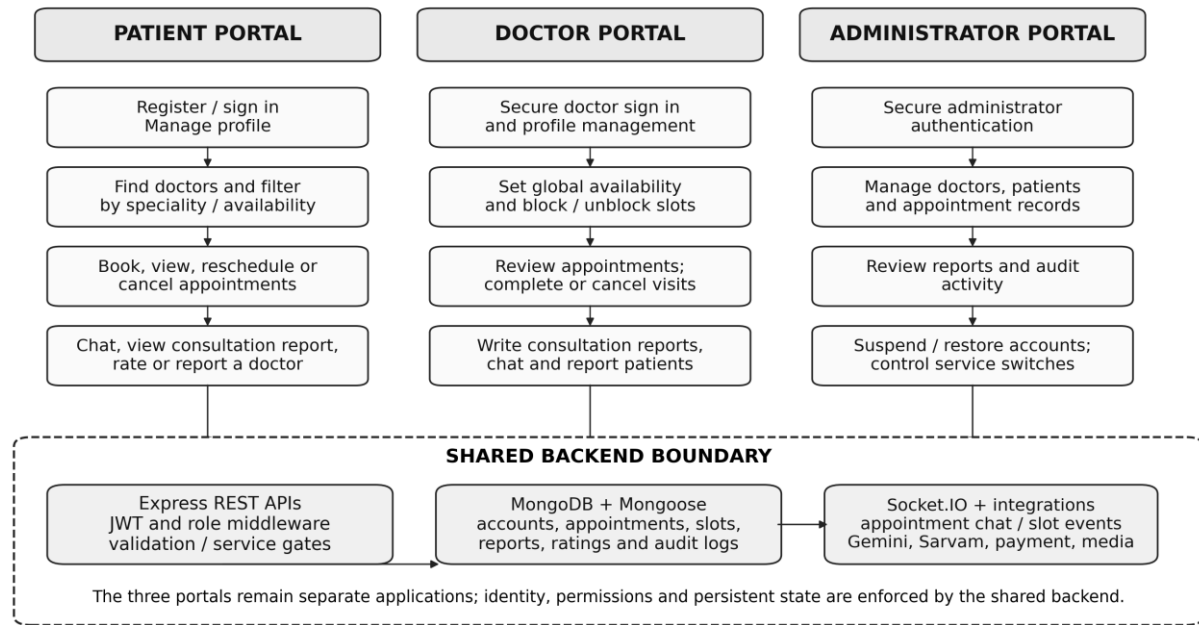


Fig. 1. Role-based workflows and shared backend boundary in Vaidyam.

The separation into three front ends preserves distinct workflows and session state. Patient, Doctor and Administrator authentication tokens are kept independent in the client applications; protected endpoints validate the signed identity and corresponding account or role on the server. Administrative operations include doctor and patient management, appointment inspection, report moderation, temporary suspension or permanent deactivation, restoration, forced logout, audit viewing and service-control toggles. This division allows operational responsibilities to be separated while retaining a common source of appointment state.

Patient, Doctor and Administrator Workflows

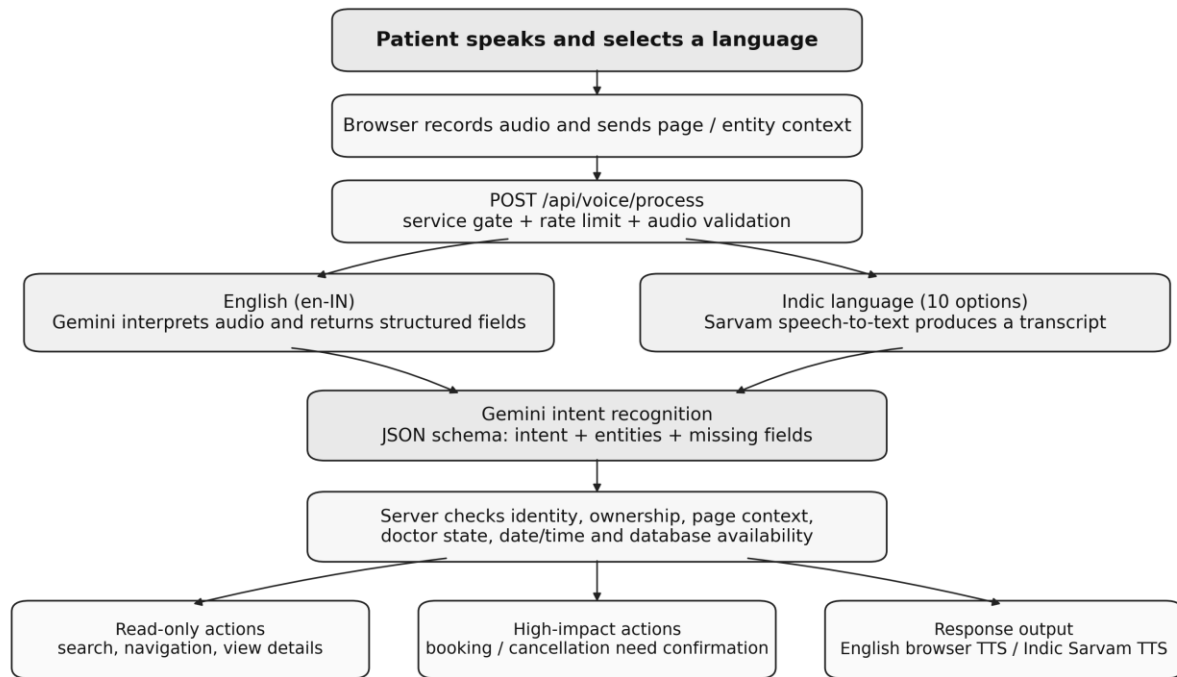
In the Patient portal, a person can register or sign in, maintain profile details, search and filter doctors, open a doctor profile, select an appointment slot, view appointment history, manage an eligible appointment, and access chat and consultation follow-up features. A patient may submit a rating for a completed appointment or report a doctor when the appointment state permits it. Appointment details include status, payment-related fields and a consultation report when supplied by the doctor.

The Doctor portal supports profile management, global availability, slot blocking and unblocking, appointment review, consultation-report entry, completion or cancellation and communication with patients associated with authorized appointments. Historical chat is grouped by appointment session; completed, cancelled or expired sessions are treated as read-only and the backend rejects unauthorized or closed-session socket access. Doctors can report patients, but account suspension and permanent deactivation are Administrator-controlled operations.

The Administrator portal provides cross-system oversight. It supports doctor and patient lists and details, appointment search and detail views, credential-reset operations, report review, account suspension and restoration, permanent deactivation, audit-log review and selected service controls such as rate limiting and AI/voice service availability. Sensitive administrator identifiers are intentionally omitted from client-facing audit responses. These features are management controls, not a replacement for institutional policy or a formal clinical-record system.

Multilingual Voice Interaction

Voice interaction is implemented as a bounded command layer rather than a free-form system with unrestricted database access. The interface offers English (en-IN), Kannada, Hindi, Tamil, Telugu, Malayalam, Marathi, Bengali, Gujarati, Punjabi and Odia. The browser records an audio clip and sends it to the backend voice endpoint with a sanitized conversation history and page/entity context. The endpoint applies service gating and rate limiting, validates the audio upload and normalizes the selected language.



Confirmation tickets are short-lived and bound to the authenticated patient; the model proposes intent but does not authorize database changes.

Fig. 2. Vaidyam voice pipeline, language branches and confirmation boundary.

For English, the backend passes the audio to Gemini and requests a schema-constrained JSON response containing the interpreted intent, entities, missing fields, confirmation requirement and a short reply. Google documents structured outputs as a way to constrain model responses to a specified JSON schema, while warning that schema-valid output still requires application-level validation [10]. For the ten supported Indian languages, Sarvam speech-to-text produces a transcript that Gemini interprets into the same intent representation [11]. Non-English replies can be synthesized by Sarvam text-to-speech [12], while English responses use browser speech synthesis.

The voice prompt supplies the current India time and the visible application context so that expressions such as “tomorrow”, “this doctor” and “the second appointment” can be resolved against available data. The model is instructed not to invent doctor IDs, appointment IDs, availability, prices or medical records. The backend remains authoritative: it checks authenticated patient ownership and current database state before a protected action. Searching, viewing or navigating can proceed without a confirmation ticket, while booking and cancellation require an explicit confirm-or-cancel step. Confirmation tickets are short-lived and bound to the authenticated patient. The voice assistant provides informational support and does not diagnose medical conditions.

Appointment Scheduling and Conflict Prevention

The appointment controller normalizes the requested date and time, validates the requested doctor, checks the global availability flag, rejects blocked slots and tests for an existing active booking. The supplied implementation accepts appointment dates within the next seven days in the Asia/Kolkata time zone and uses 30-minute time slots from 10:00 AM through 8:30 PM. A booking-request identifier is used for retry idempotency so that a repeated request can return the previously created result rather than create another appointment.

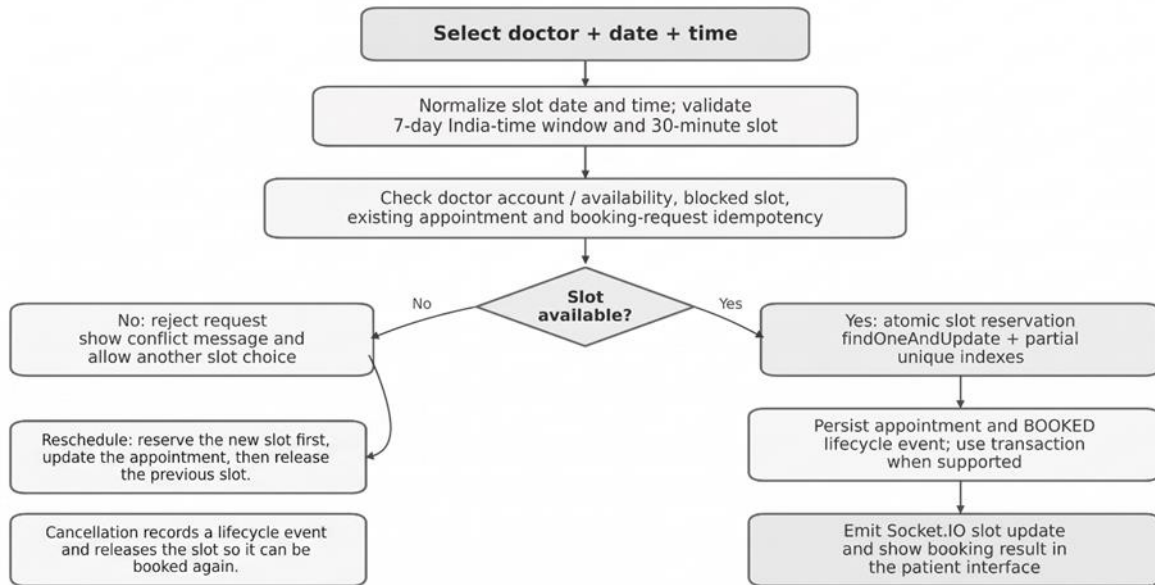


Fig. 3. Conflict-aware appointment scheduling and slot-state update workflow.

Slot reservation uses a conditional atomic update to add a selected time to the doctor's booked slots. The appointment schema also declares partial unique indexes for active doctor-slot combinations and active patient-slot combinations. Where the configured MongoDB deployment supports transactions, booking attempts to use a transaction; a guarded atomic path is retained as a fallback for deployments that do not support transactions. MongoDB documentation describes single-document atomicity and the use of unique indexes to prevent conflicting writes [13]. After a successful booking, the backend stores a BOOKED lifecycle event and emits a Socket.IO slot update. Rescheduling reserves the destination slot before updating the appointment and releasing the old slot; cancellation records the lifecycle event and releases the slot.

This approach aims to keep browser availability indicators synchronized with backend state and to prevent duplicate active bookings. The UI is not treated as the authority for whether a slot remains available: requests are checked again by the backend at the point of mutation. When a conflicting request arrives, the operation is rejected and the patient is asked to choose a different slot rather than silently moving the appointment.

Results and Discussion

Source inspection identified the three role-specific portals, shared REST backend, MongoDB/Mongoose schemas, JWT-based middleware, Socket.IO handlers, voice-service modules, appointment lifecycle fields, report/rating structures and administrator moderation controls in the supplied archive. The following test results come only from the included standalone in-memory booking simulation. Table 1 summarizes its ten scenario groups; each outcome is a pass reported by that harness, not an end-to-end pass against a live database.

Table 1. Source-based functional verification of appointment safeguards

No.	Scenario	Check / observed behaviour	Outcome
1	Normal booking	A valid request creates one mock appointment and marks the mock doctor slot as booked.	Pass
2	Duplicate click	A second active request by the same patient for the same doctor slot is rejected.	Pass
3	Idempotent retry	A repeated request with the same bookingRequestId returns an already-booked response without a second record.	Pass
4	Sequential contention	A second patient requesting a slot already reserved by another patient is rejected.	Pass
5	Two concurrent requests	Two simulated requests for one slot produce exactly one success and one failure.	Pass
6	Cancel and rebook	After the mock cancellation releases the slot, another patient can reserve it.	Pass

No.	Scenario	Check / observed behaviour	Outcome
7	Five concurrent requests	Five simulated patients competing for one slot produce one successful reservation.	Pass
8	Invalid time values	Eight malformed or out-of-range time values are rejected and no mock appointment is created.	Pass
9	Past date	A booking request for a past date is rejected.	Pass
10	Unavailable doctor	A booking request is rejected when the simulated doctor is unavailable.	Pass

The harness reported 29 passed assertions and zero failures. The test source makes the scope of that result important: its concurrency cases use an in-memory mock database and a simulated lock to model a reservation race; helper functions are copied from the production controller rather than invoking the controller or MongoDB directly. The result is therefore a useful deterministic check of intended input and state-transition behavior, but not independent evidence of real database concurrency guarantees or production load handling.

The separate voice-safety test did not reach its assertions because module loading failed with ``ERR_MODULE_NOT_FOUND`` for ``mongoose``. Voice intent validation, confirmation expiry, external API errors and audio output therefore remain source-inspection topics rather than independently executed test results in this environment. Browser testing, language-specific word error rate, intent/entity accuracy, latency, user experience, live payment processing and security penetration testing were not measured.

Security, Limitations and Future Work

The supplied code includes signed JWT checks, role-specific route middleware, account-status and session-version checks, appointment ownership checks, upload validation, voice service gating, request-rate limits, and authorization before private appointment chat is joined or used. The JSON Web Token specification describes the token format and claims representation [14], while OWASP authorization guidance emphasizes that authentication and authorization are distinct and that access decisions must be enforced for the requested resource [15]. These observations are implementation-level, not evidence of a formal security audit or compliance certification. A deployment should additionally review transport security, secret management, database privileges, monitoring, backups, retention, incident response and privacy obligations.

Payment handling requires particular qualification. The inspected code includes a server request to create a payment order through SugamPay and an endpoint that updates the appointment payment flag when a transaction identifier is supplied. The reviewed verification handler does not itself demonstrate a provider-side transaction-status or signature check. Accordingly, the available source does not justify a claim of independently verified end-to-end payment processing; the payment path should be treated as an integration requiring validation and security review before any production use.

The primary evaluation limitation is that the available evidence consists of source inspection and a deterministic mock harness. Future work should add automated tests that invoke the actual backend handlers against a configured MongoDB test deployment, verify the unique indexes and transaction fallback, and exercise simultaneous requests under realistic failure conditions. The voice pipeline should be assessed separately for task-completion rate, word error rate, intent and entity accuracy, response latency, ambiguous-command recovery and performance in each supported language. Browser-level accessibility and usability testing, end-to-end payment verification, load testing and an independent security review are also needed before making claims about deployment readiness or patient outcomes.

Conclusion

Vaidyam is an integrated healthcare appointment and management platform that separates Patient, Doctor and Administrator workflows while maintaining shared server-side appointment state. Its implementation combines doctor search, appointment booking and lifecycle management, payment flow hooks, consultation reports, ratings, reporting, administrative moderation and appointment-linked real-time chat. The multilingual voice assistant uses a hybrid pipeline that combines Gemini-based intent interpretation with Sarvam speech services for Indian-language audio and makes confirmation a boundary for booking and cancellation actions. Conditional atomic reservation, unique active-slot indexes, idempotency and real-time slot events are used to reduce double-booking risk. A bundled booking simulation passed 29 of 29 assertions, but live integration, speech accuracy, usability and production performance remain to be evaluated. The next development stage should focus on empirical evaluation, operational reliability and formal privacy and security review.

References:

[1] World Health Organization, *Global Strategy on Digital Health 2020-2027*. Geneva, Switzerland: WHO, 2025. ISBN: 978-92-4-011687-0. [Online]. Available: <https://www.who.int/publications/i/item/9789240116870>. Accessed: Oct. 17, 2026.

-
- [2] A. H. Odeh, R. Abdelhadi, and H. Odeh, "Medical patient appointments management using smart software system in UAE," in *2019 International Arab Conference on Information Technology (ACIT)*, 2019, pp. 97-101, doi: 10.1109/ACIT47987.2019.8991064.
- [3] T. Arthanari, D. N. S. Siddharth, G. S. Srivatsan, and V. Shreenath, "Optimization and enhancement of doctor appointment booking system using Next.js, Strapi, and REST API," in *2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN)*, 2024, pp. 79-83, doi: 10.1109/ICPCSN62568.2024.00021.
- [4] C. Valenzuela-Nunez, G. Latorre-Nunez, and F. Troncoso-Espinosa, "Smart medical appointment scheduling: Optimization, machine learning, and overbooking to enhance resource utilization," *IEEE Access*, vol. 12, pp. 7551-7562, 2024, doi: 10.1109/ACCESS.2024.3349953.
- [5] L. Laranjo et al., "Conversational agents in healthcare: A systematic review," *Journal of the American Medical Informatics Association*, vol. 25, no. 9, pp. 1248-1258, 2018, doi: 10.1093/jamia/ocy072.
- [6] M. Milne-Ives et al., "The effectiveness of artificial intelligence conversational agents in health care: Systematic review," *Journal of Medical Internet Research*, vol. 22, no. 10, Art. no. e20346, 2020, doi: 10.2196/20346.
- [7] M. Jovanovic, M. Baez, and F. Casati, "Chatbots as conversational healthcare services," *IEEE Internet Computing*, vol. 25, no. 3, pp. 44-51, 2021, doi: 10.1109/MIC.2020.3037151.
- [8] N. Helander, A. Pundziene, R. Sermonyte-Baniule, J. Carrabina, C. Mathieu, S. Boethius, and I. Georgescu, "Digital healthcare platforms enabling remote care value creation: The patient perspective," in *2023 IEEE International Conference on Technology and Entrepreneurship (ICTE)*, 2023, pp. 38-43, doi: 10.1109/ICTE58739.2023.10488518.
- [9] Q. Shi, Q. Han, and C. Soares, "C-PATH: Conversational patient assistance and triage in healthcare system," arXiv preprint arXiv:2506.06737, 2025. [Online]. Available: <https://arxiv.org/abs/2506.06737>.
- [10] Google, "Structured outputs," *Gemini API Documentation*. [Online]. Available: <https://ai.google.dev/gemini-api/docs/structured-output>. Accessed: Oct. 17, 2026.
- [11] Sarvam AI, "Speech-to-Text API Reference," *Sarvam AI Model APIs*. [Online]. Available: <https://docs.sarvam.ai/api-reference/speech-to-text/transcribe>. Accessed: Oct. 17, 2026.
- [12] Sarvam AI, "Text-to-Speech API Reference," *Sarvam AI Model APIs*. [Online]. Available: <https://docs.sarvam.ai/api-reference/text-to-speech/convert>. Accessed: Oct. 17, 2026.
- [13] MongoDB, Inc., "Atomicity and Transactions," *MongoDB Manual*. [Online]. Available: <https://www.mongodb.com/docs/manual/core/write-operations-atomicity/>. Accessed: Oct. 17, 2026.
- [14] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, May 2015, doi: 10.17487/RFC7519. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc7519.html>.
- [15] OWASP Foundation, "Authorization Cheat Sheet," *OWASP Cheat Sheet Series*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html. Accessed: Oct. 17, 2026.