Kshitiz Sharma 2026, 14:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

DentalSaaS: Design and Implementation of A Branch-Aware Dental Clinic Management Prototype

Kshitiz Sharma[1] Vansh Suryan[2] Mohit Dahiya[3] Ms. Asha[4] Dr.Pradeep[5]
Department Of Computer Science Engineering Hmr Institute Of Technology & Management, Delhi, India

Abstract- Dental practices operating multiple branches need to make relevant patient history available across locations without treatin g each branch as an independent data silo. This paper presents DentalSaaS, branded in the user interface as SmileCraft Dental Clinic, a web-based prototype for one clinic organization with multiple branches. The system combines a React/Vite client, a Node.js/Express API and MongoDB models managed with Mongoose. JWT-authenticated clinic-wide doctors, branch-assigned doctors and patients receive different operational scopes. Patient profiles, visits, treatment records and invoices are shared within the organization, while appointments and inventory are associated with a branch. The patient portal supports self-registration, staff-issued expiring activation links for existing records, branch selection for appointment requests, and access to a patient's own records and invoices. Inventory operations are constrained by branch assignment, and active appointment slots are unique within a branch. Clinic-metric questions are answered locally; optional Gemini use is limited to general questions, and the patient assistant is local. The work is an implementation case study based on source and configuration review, not a clinical trial or performance evaluation. No patient data, usability measurements, load-test results or clinical outcomes are claimed. Seeded records are illustrative examples only.

Index Terms—Dental clinic management, multi-branch systems, shared patient records, branch-scoped access, patient portal, MongoDB, JWT.

I. INTRODUCTION

Digital health systems can support continuity of care when they are designed around organizational, human and technical requirements rather than technology alone [1]. In a dental group with multiple locations, patient history may otherwise remain attached to the branch where it was created. A second branch may then lack the earlier visit notes or prescription context needed for an informed follow-up. This project addresses that coordination problem within a deliberately bounded setting: one clinic organization operating multiple branches.

DentalSaaS is a browser-based clinic-management prototype, presented to users as SmileCraft Dental Clinic. Its design makes patient profiles, visits, treatment records and invoices available across branches of the same organization, while keeping appointment schedules and inventory local to each

branch. This is not a multi-tenant platform for unrelated clinics. The distinction between organization-wide clinical records and branch-local operations is the principal design decision examined in this paper.

The contribution is an implementation-oriented case study: it describes the system boundary, role and branch scoping, data model, selected workflows, and limitations that follow from a prototype deployment. It does not claim clinical effectiveness, measurable time savings, improved patient outcomes, or suitability for production health-data processing. All seeded records are example data and are not real patient information.

The guiding engineering question is how a small clinic-management application can represent records that must remain visible across locations while enforcing location-specific scheduling and stock operations. The paper answers this question by tracing three connected concerns: where each resource is stored, how a request is authorized, and which constraints are enforced by the API or database. It also distinguishes implemented behavior from work that would be necessary before any real-world deployment.

II. BACKGROUND AND DESIGN CONTEXT

© 2026 Kshitiz Sharma Vansh Suryan Mohit Dahiya, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

1. Digital Records And Organizational Scope

The World Health Organization's global digital-health strategy emphasizes that initiatives require coordinated organizational, human, financial and technical resources [1]. In the present system, the organizational boundary is one clinic group, not a network of independent providers. That choice enables shared records among its branches but also means that branch membership alone is not a privacy boundary for patient history. The design therefore makes this assumption explicit and identifies consent management, audit logging and tenant isolation as unresolved production concerns.

2. Access Control And Signed Sessions

Authentication establishes who is making a request; authorization determines whether that identity may perform the requested operation. OWASP guidance recommends mapping users, resources and allowed operations and enforcing least privilege at the server [2]. DentalSaaS applies this distinction through signed JSON Web Tokens (JWTs) and server-side checks for doctor, patient, clinic-wide and branch-assigned workflows. A token is not itself a complete authorization policy: the server must still scope database queries to the authenticated user's role, patient link or branch assignment.

JWT is a compact claims format described by RFC 7519 [3]; RFC 8725 provides best-current-practice guidance for secure JWT use [4]. The project issues eight-hour signed tokens containing identity and role data, with patient or branch identifiers where applicable. This describes the current implementation, not a security certification. Deployment still requires a strong secret, secure transport, careful token handling and independent security review.

3. Document Data Modeling

MongoDB supports both embedded documents and references, and recommends shaping a schema around access patterns [5]. DentalSaaS uses references for entities with independent lifecycles: users, branches, patients, visits, appointments, treatments, invoices and inventory items. Patient-linked records preserve a common patient identity, while branch references on appointments and inventory encode local ownership. Mongoose schemas add application-level field definitions and validation, but they do not remove the need for careful authorization and migration planning.

III. SYSTEM REQUIREMENTS AND ARCHITECTURE

1. Functional Scope

The doctor console supports patient management, visit documentation, appointment operations, treatment tracking,billing, inventory, dashboard summaries and reports. The patient portal supports account registration or activation of an existing patient record, appointment requests, access to the patient's own medical and billing records, and general navigation and care guidance. The clinic assistant answers selected metrics locally and may use an external language-model API for general questions when configured. The patient assistant remains local. Neither assistant is intended to diagnose or prescribe.

2. Three-Tier Architecture

The client is a React single-page application built with Vite [8]. It exchanges JSON with an Express API over HTTP [9]; the API is the component that accesses MongoDB through Mongoose schemas [5], [10]. This layered arrangement follows a common client/server separation, while its endpoints use HTTP methods and resource-oriented routes [6]. The application is stateless between API requests at the server layer and authenticates protected requests with bearer tokens. The development Vite proxy is not a production routing configuration.

At startup, the server connects to MongoDB, synchronizes patient account email links, ensures an active branch exists, migrates legacy branchless appointments and inventory according to documented rules, and creates the branch/date/time uniqueness index for active appointment states. The migration copies the previous shared inventory quantity to each active branch; it does not divide stock. Real deployment therefore needs a reviewed inventory reconciliation rather than an assumption that this demonstration migration reflects physical stock.

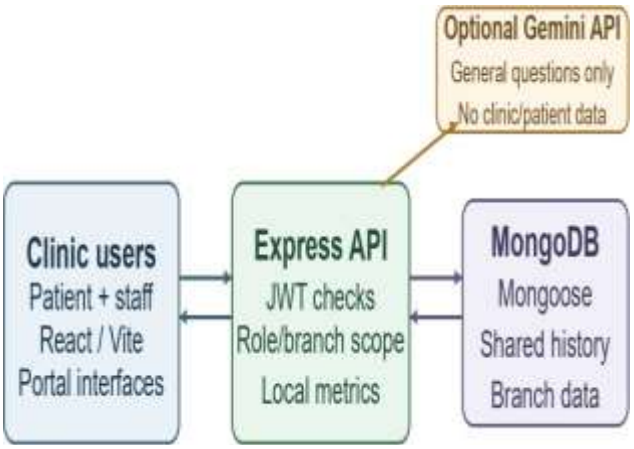

Fig. 1. Current application architecture. Optional Gemini requests contain general questions only; clinic metrics are answered locally.

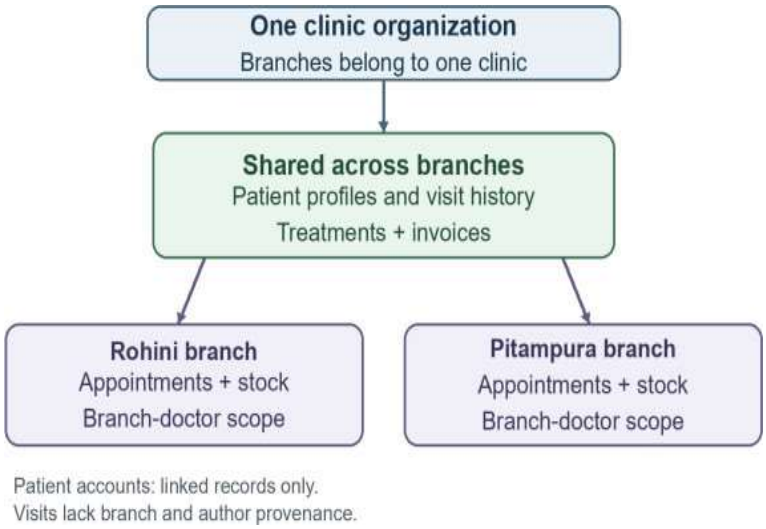

Fig. 2. Organization-wide patient history and branch-specific appointment and inventory resources.

3. Domain And Resource Scopes

Resource	Scope and access rule
Patient, visit, treatment, invoice	Shared across branches within this one clinic organization; patients see only their linked records.
Appointment	Assigned to a branch; branch doctors see their own branch; clinic-wide doctors can manage across selected branches.
Inventory item	Required branch reference; branch doctors are limited to their assigned branch.
Branch	Managed by clinic-wide doctor accounts; patients are offered active branches for booking.
Patient invitation	Single-use activation token stored as a hash; expires after 24 hours; staff share it manually.

The current model contains nine Mongoose models: User, Branch, PatientInvitation, Patient, Visit, Appointment, Treatment, Invoice and Item. Users may carry a patientId or branchId. Visit documents reference Patient and do not carry branch or author provenance. Appointments and items require Branch references. This supports organization-wide patient history and branch-local scheduling and stock, but it does not identify which clinician or branch authored a visit.

4. Resource Ownership And Data Lifecycle

Resource ownership is expressed through references rather than separate databases for each branch. A Patient document holds demographic and health-history fields, while Visit documents reference the patient and hold a date, reason, prescription text, optional prescription file data and notes. Treatment, Appointment and Invoice documents can retain a patientId reference, while some older or display-oriented fields also store the patient's name. This mixture supports portal links and existing records, but it creates a need for consistent identity linkage when data is created, migrated or queried. Inventory is different: its required branchId is part of the resource identity used in each read and update.

The startup migration reflects this model transition. It links email addresses from patient accounts to their patient records, ensures that at least one active branch exists, assigns legacy appointments without a branch to the earliest-created active branch, and copies each legacy unassigned inventory item's quantity and reorder threshold to every active branch before removing the unassigned item. The copy is not a stock split or a count of physical supplies. This makes the migration convenient for a demonstration but potentially misleading for real inventory, so operators must review and reconcile branch quantities before treating migrated values as operational facts.

The appointment index is another data-lifecycle boundary. It replaces the older date/time uniqueness rule with a partial unique index on branch, date and time for Booked, Confirmed and Scheduled records. The API also checks for a conflict before creating or moving an appointment and translates duplicate-key races into a conflict response. The database index remains important because two simultaneous requests can both pass an application-level pre-check. Cancelled or completed appointments are outside the partial index and therefore do not permanently reserve a slot.

IV. IMPLEMENTATION

1. Branch And Role Enforcement

The API distinguishes clinic-wide doctor accounts from branch doctors through the presence of a branchId claim. Clinic-wide branch-management endpoints reject branch-assigned accounts. Appointment listing filters rows by the doctor's branch when present. Inventory handlers derive the branch from the doctor's token, or accept a selected branch for clinic-wide accounts, then query or update items using both item identity and branch identity. This server-side scoping is essential; a selector in the client alone would not enforce access.

2. Appointment Lifecycle And Integrity

Patients choose an active branch and submit a future date, 24-hour time and reason. Requests are created as Booked pending staff confirmation. A patient can reschedule or cancel only their own future Booked or Confirmed appointment. A unique partial index on branchId, date and time applies to active appointment statuses, allowing different branches to use the same time while preventing duplicate active slots at one location. Cancelled and completed appointments no longer reserve a slot.

3. Existing-Record Patient Activation

When a patient already has a clinic record, staff can add an email and generate an activation link. The server creates a random token, stores its SHA-256 hash with the patient identifier and email, and sets a 24-hour expiry. Activation verifies the hash and expiry, confirms the email still belongs to the patient record, creates the linked patient account, then removes outstanding invitations for that patient. Links are manually shared; automatic email delivery is not configured. This avoids creating a disconnected second profile when a patient joins the portal.

4. Inventory And Dashboard

Inventory records require branchId, non-negative quantity and a reorder threshold. The API returns low-stock items for the selected or assigned branch. Clinic-wide summaries identify the branch associated with each low-stock item; branch doctors see their branch's data. In the current dashboard endpoint, patient counts, treatment totals and invoice-derived revenue remain organization-wide even for branch-assigned doctors, while appointment and inventory queries are branch-filtered. These summary boundaries should be made clear to users and revisited if a clinic expects financial or operational metrics to be restricted by location.

5. Assistant Privacy Boundary

Supported clinic metrics and follow-up information are answered locally from application records. General questions may be sent to Gemini if the key is configured; patient records and clinic statistics are excluded from those calls. Browser speech recognition is available where supported, and the browser's speech service may process voice input. The system warns users not to include identifying health information in general prompts. The patient-facing assistant handles navigation and general care questions locally.

6. End-To-End Workflow And Data Boundaries

A typical clinic setup begins with a clinic-wide doctor account creating active branches and, where needed, branch-assigned doctor accounts. The branch identifier is included in a branch doctor's signed token at login. Subsequent appointment and inventory requests are scoped using that identity on the server. Clinic-wide staff can select a location for inventory work and review appointments across locations. This arrangement supports one organization with centralized oversight, but it does not create branch-level clinical-record isolation.

Patient onboarding has two routes with different data consequences. A new patient can register using name, email, ten-digit phone number and a password; the

server creates a patient record and a linked portal account. For an existing clinic record, staff instead enter a valid email and issue a one-time activation token. The patient chooses a password through the activation flow, which links the account to the existing patientId. Email uniqueness checks prevent a second account or duplicate patient email from silently taking ownership of a record. Because delivery is manual, staff must transmit the link through a suitable channel and protect it from disclosure.

After login, a patient can request a future appointment at an active branch, then view the branch label and appointment state in the portal. A reschedule can choose another active branch and is checked against the target branch's active slots. Staff can record a visit against the shared patient profile and add treatment and invoice records. Those records can be viewed through the patient's linked account. Inventory counts, by contrast, remain separate by branch. This path illustrates the intended separation between clinical continuity and location-dependent operations; it does not show that the workflow has been evaluated with clinic staff.

The API also exposes a reminder preference on the patient record. Saving an enabled preference changes the stored boolean only; no scheduler or message provider is implemented. Dashboard summaries are assembled from clinic records at request time. Clinic data questions and follow-up lists are handled through local code, while optional Gemini requests are limited to general questions. These boundaries matter because a user-visible setting, dashboard card or assistant answer should not be mistaken for an integrated messaging or clinical decision-support service.

V. EVALUATION METHOD AND OBSERVATIONS

1. Method

This paper uses a descriptive implementation review. The README, client routes and state handling, Express middleware and route handlers, Mongoose models, seed script and project repository [11] were inspected to map implemented workflows to their data scope. This is a source-based characterization, not an experiment with participants. No patient records, operational clinic data, production telemetry, quantitative workload measurements or performance benchmarks were used.

2. Implementation Observations

Source inspection confirms that the implementation represents two demo branches (Rohini and Pitampura), clinic-wide and branch-assigned doctor accounts, patient portal accounts, branch-owned appointment and inventory records, shared patient-linked records, activation invitations, and a patient reminder preference. The seed dataset is explicitly illustrative. These are implementation facts; they do not establish clinical utility, usability, privacy compliance or resistance to attack.

3. Verification Matrix (Proposed; Not Executed)

Concern	Verification case and evidence to record
Branch authorization	Branch doctor can read and change assigned-branch inventory, while a cross-branch identifier is rejected. Record token role and branch, request, response and resulting database row.
Appointment uniqueness	The same date/time is allowed at different branches, but a duplicate active slot at one branch is rejected. Record both requests, response codes and final rows.
Patient ownership	A patient account reads only records linked to its patientId; another account cannot retrieve them. Record account identifiers, endpoint responses and record identifiers.
Activation lifecycle	A valid link activates once; expired, reused and email-mismatched links fail. Record expiry, activation outcome and account linkage.

Concern	Verification case and evidence to record
Workflow transitions	A patient can change an eligible future booking; past or terminal records are rejected. Record initial state, request, resulting state and branch.
Dashboard scope	Branch appointment and inventory values are local; patient and invoice summaries are identified as organization-wide. Record role, response fields and query scope.
Reminder boundary	Preference can be changed and read back without claiming delivery. Record the saved preference and absence of a send event.
Migration review	Legacy data is migrated in an isolated database copy. Record before/after counts, branch mapping and reconciled quantities.

The following matrix turns the design claims into reproducible checks. It is a test plan, not a report of successful test runs. Every case below remains unexecuted for this paper and must be run against a named source revision and disposable database before its result can be reported.

The strongest authorization checks should call the API directly with a valid token and a substituted resource identifier; a hidden button in the client is not evidence that the server denies the request. Slot-integrity checks should include concurrent submissions to exercise the unique index, not only sequential pre-checks. Activation tests should inspect invitation and account state without storing raw tokens in logs. The branch-specific cases in the accompanying project report are marked as not run or requiring re-test until executed against the current build.

4. Evidence And Reproducibility

The demo seed script clears and recreates its collections and must be run only against a disposable database. Branch-doctor passwords are generated at seed time and printed once. A reproducible evaluation should record the source revision, Node.js and browser versions, database configuration, seed run, user role, request, observed response and evidence for each test. Test status should be assigned only after observing the current build. This paper therefore reports no pass rate, latency, throughput, usability score or clinical outcome.

VI. DISCUSSION

The central design trade-off is between continuity and compartmentalization. Sharing patient history across branches avoids duplicated patient identities and fragmented visits, while branch-local appointments and inventory reflect location-specific operations. However, sharing every patient's clinical history with all branch doctors is a strong organizational assumption. Some organizations may require patient consent, treatment-specific access, or restrictions between subsidiaries. Those policies would require an explicit consent and authorization model rather than merely adding more branches.

The present distinction is not a complete branch-scoped data policy. Branch doctors receive branch-filtered appointment and inventory operations, but the dashboard computes patient counts, treatment aggregates and invoice totals from organization-wide collections. Aggregate values reveal less than row-level records, yet their visibility may still conflict with a clinic's intended financial or operational boundaries. A future policy should declare scope per resource and per metric, then enforce it in API queries and test it explicitly; the interface should label organization-wide values rather than imply that all cards are location-local.

Branch assignment is currently a scope attribute on a doctor account, not a fine-grained role system. It limits appointment and inventory operations but does not restrict access to patient records. The same role value ('doctor') is used for clinic-wide and branch-assigned

accounts. This keeps the prototype small but makes administrative privilege distinctions less expressive than separate clinic administrator, branch manager, receptionist and clinician permissions. OWASP guidance supports explicitly mapping actions to principals and resources and testing the resulting authorization matrix [2].

The database model also makes a deliberate trade: visits are shared but lack branch and clinician attribution. The project can demonstrate shared history but cannot provide complete authorship provenance for every entry. Adding branchId, authorId, immutable event history and audit review would improve traceability, but these changes require careful migration and privacy design. Similarly, storing prescription PDFs within

visit documents is convenient for a prototype but is not a complete document lifecycle or secure file-storage strategy.

The data model also retains display names alongside patient references in some appointment, treatment and invoice records. References support stable linkage, while names make simple lists readable and can accommodate legacy records. The coexistence creates consistency risks if a patient's name changes or a reference is absent. A production migration should define a canonical patient identifier, reconcile legacy rows, validate references at write boundaries and decide whether denormalized display fields are snapshots or live values. The existing startup migration does not solve all historical identity ambiguities.

Finally, the assistant boundary is narrower than a clinical decision-support system. Local metric answers use application data; optional external model use is limited to general questions. This reduces data transfer but does not by itself establish privacy compliance or safe clinical performance. The prototype explicitly avoids diagnosis and prescribing, and reminder preference persistence must not be confused with a functioning notification service.

There is also a practical scalability consideration in how summaries are calculated. The current stats handler loads collections and derives several counts and totals in application memory rather than relying exclusively on database aggregation. This is straightforward for a small demonstration dataset, but it does not establish performance at larger clinic volumes. Any optimization should preserve the stated branch/organization scope, be checked against known test fixtures, and be measured under a documented workload before a performance claim is made.

VII. LIMITATIONS AND ETHICAL CONSIDERATIONS

The system is a starter/demo application and is not production-ready. It has no implemented patient-consent workflow for organization-wide sharing, no audit trail of record access or changes, no tenant isolation for unrelated clinic organizations, and no separate receptionist or assistant role. Generic management routes need broader validation; rate limiting, account lockout, password recovery and multi-factor authentication are not implemented. Production HTTPS, origin restrictions, backups, restore procedures, monitoring and independent security testing have not been established.

The server also supplies a development fallback for the JWT signing secret when the environment variable is absent. A deployed instance must configure a strong, private secret and must not rely on that fallback. Authentication and authorization should be reviewed together: token expiry alone does not provide account recovery, session revocation, step-up verification or protection from every form of credential compromise. Cross-origin policy, request limits, log redaction and safe handling of invitation links also need explicit deployment decisions.

Printed bills are not statutory-grade invoices; tax treatment and mandatory business details require professional review. Uploaded prescription PDFs lack a dedicated storage and retention system. Appointment reminder delivery is in progress. No automated unit or integration test suite, load test, usability study, clinical validation or deployment evaluation is reported here. These limitations are material because health information requires strong governance and technical safeguards. India's Digital Personal Data Protection Act, 2023 is a relevant legal context [7], but this paper makes no claim that the prototype complies with that Act or any healthcare regulation.

All seeded patient names, health notes, appointments, prescriptions, invoice values and inventory quantities are illustrative only. They are not real patient data and must not be presented as clinical evidence. Before any real-data pilot, the organization would need documented lawful purpose and notice, consent where applicable, data minimization, access governance,

secure deployment, retention and deletion policies, breach response, backup protection, and an independent legal and security review.

1. Future Work

Further development should begin with a written authorization matrix rather than additional screens. For each role, resource and operation, the matrix should state whether access is clinic-wide, branch-scoped or patient-owned, and whether access is read, create, update or delete. Dashboard aggregates need explicit decisions alongside record-level routes. Automated API tests can then exercise permitted and denied cases using different branch and patient identities, including direct requests that bypass client controls.

A second priority is clinical provenance and governance. Visit records should capture the authenticated author and relevant branch, while an audit mechanism records access and changes without exposing sensitive details in ordinary logs. Patient-facing notices and consent controls should explain organization-wide sharing and support withdrawal or correction according to the clinic's legal and operational policy. These changes require a reviewed migration strategy because existing visits do not contain author or branch attribution.

A third priority is production data handling. Prescription files should move from large values embedded in database documents to a managed storage design with authorization checks, size and type validation, retention rules and protected backups. Deployment should require explicit security configuration, including a strong JWT secret, HTTPS, restrictive origin policy, monitoring and recovery procedures. Data retention, export and deletion workflows should be documented before any real patient information is introduced.

Finally, the workflows need staged validation. First, run focused model and API tests for authorization, appointment concurrency, identity linkage, migration behavior and dashboard scope. Next, conduct usability reviews with representative clinic staff and patients using synthetic data. Only after legal, privacy and security review should any controlled pilot be considered. Actual reminder delivery would require an explicit provider integration, opt-in and opt-out behavior, delivery status and failure handling. Any later report should distinguish test results, user feedback and clinical outcomes, and should include only measurements that were genuinely collected.

VIII. CONCLUSION

DentalSaaS demonstrates a branch-aware architecture for a single dental clinic organization. It shares patient-linked history across branches while keeping appointments and inventory branch-specific, and it provides clinic-wide doctor, branch-doctor and patient workflows. The implementation uses a React/Vite client, Express API, MongoDB/Mongoose models and JWT-based sessions. The paper's contribution is a documented engineering design and an explicit account of its data scope, workflow constraints and limitations—not an empirical claim that outcomes or efficiency improved. Future work should prioritize authorization testing, patient consent and auditability, provenance for clinical entries, production data safeguards, reminder delivery and independent usability/security evaluation.

REFERENCES

1. World Health Organization, Global Strategy on Digital Health 2020–2025. Geneva, Switzerland: WHO, 2021, ISBN 978-92-4-002092-4.

2. OWASP Foundation, “Authorization Cheat Sheet,” OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet. html. Accessed: Nov. 20, 2026.
3. M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015. doi: 10.17487/RFC7519.
4. Y. Sheffer, D. Hardt, and M. Jones, “JSON Web Token Best Current Practices,” RFC 8725, Internet

Engineering Task Force, Feb. 2020. doi: 10.17487/RFC8725.
5. MongoDB, Inc., "Data Modeling in MongoDB," MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/data-modeling/. Accessed: Nov. 20, 2026.
6. R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation, University of California, Irvine, 2000.
7. Government of India, The Digital Personal Data Protection Act, 2023, Act No. 22 of 2023, Gazette of India, Aug. 11, 2023.
8. Meta Platforms, Inc. and contributors, "Quick Start," React Documentation. [Online]. Available: https://react.dev/learn. Accessed: Nov. 20, 2026.
9. OpenJS Foundation, "Express routing," Express Documentation. [Online]. Available: https://expressjs.com/en/guide/routing/. Accessed: Nov. 20, 2026.
10. Automattic, Inc., "Mongoose Schemas," Mongoose Documentation. [Online]. Available: https://mongoosejs.com/docs/guide.html. Accessed: Nov. 20, 2026.
11. Project source code, "DentalSaaS / SmileCraft Dental Clinic," GitHub repository. [Online]. Available: https://github.com/mohItD4hiya/DentalSaas. Accessed: Nov. 20, 2026.