IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Recipe Hub - A Full -Stack A.I Powered Recipe Sharing Platform

Siddiqui Mohammed Saif Mujibullah and Shaikh Simra Munaf Zeenat
Student, Master of Computer Applications (MCA)
University of Mumbai, Mumbai, Maharashtra, India

Abstract: *Recipe Hub is a full-stack web application designed to enable collaborative recipe sharing with built-in moderation, AI-driven suggestions, and community engagement. Built on a Node.js/Express backend connected to a MongoDB database and a Next.js 15 frontend, the platform allows users to create, submit, and discover recipes across 17 meal categories. Submitted recipes undergo admin moderation before publication, maintaining content quality. An ingredient-matching engine powered by a custom scoring algorithm suggests existing recipes from the community database, while an integration with Google Gemini 1.5 Flash generates entirely new recipe ideas on demand. Users engage with content through a like/dislike voting system, nested commenting, and the Recipe Arena — a gamified leaderboard that ranks approved recipes by community engagement across configurable timeframes. This paper documents the system architecture, data models, API design, authentication strategy, AI integration, and key engineering decisions behind Recipe Hub.*

Keywords: *Recipe Hub*

I. INTRODUCTION

The proliferation of recipe content on the internet has created a paradox of choice for home cooks: while millions of recipes exist online, finding ones that match available ingredients, dietary needs, and skill level remains a challenge. Existing platforms are largely consumer-facing content repositories rather than community-driven sharing ecosystems with intelligent discovery.

Recipe Hub addresses this gap by combining three pillars: structured community content creation with quality control, ingredient-aware recipe suggestion (both from community recipes and AI-generated), and social engagement mechanics. The project demonstrates how modern web technologies — Next.js App Router, Zustand state management, MongoDB aggregation pipelines, JWT cookie authentication, and large language model APIs — can be composed into a cohesive full-stack application.

This paper is organized as follows: Section 2 surveys the technology stack. Section 3 describes the system architecture. Section 4 details the data models. Section 5 covers the API surface. Section 6 explains authentication and security. Section 7 documents the AI integration. Section 8 describes the frontend architecture. Section 9 analyzes key features. Section 10 discusses limitations and future work. Section 11 concludes.

II. TECHNOLOGY STACK

2.1 Backend Technologies

The server is built on Node.js with the Express 4.x framework, providing a lightweight HTTP layer. MongoDB serves as the primary data store, accessed via the Mongoose 7.x ODM which adds schema validation, middleware hooks, virtuals, and query building. The bcryptjs library handles password hashing with a salt factor of 12 rounds. JSON Web Tokens (JWT) are issued via the json webtoken package and stored exclusively in http Only cookies, providing XSS-resistant session management. The Joi library validates all incoming request payloads at the middleware layer before controllers are invoked. Rate limiting on authentication endpoints is enforced with express-rate-limit.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Package	Version	Role
express	4.18.x	HTTP server and routing
mongoose	7.5.x	MongoDB ODM with schema validation
bcryptjs	2.4.x	Password hashing (12 rounds)
jsonwebtoken	9.0.x	JWT generation and verification
joi	17.9.x	Request payload validation
cookie-parser	1.4.x	Http Only cookie handling
express-rate-limit	6.10.x	Auth endpoint rate limiting
cors	2.8.x	Cross-origin resource sharing
dotenv	16.3.x	Environment variable management

2.2 Frontend Technologies

The client is a Next.js 15 application using the App Router paradigm with React 19. All UI primitives are sourced from the shadcn/ui component library built on Radix UI, providing accessible, unstyled building blocks styled via Tailwind CSS. Client-side state is managed with Zustand 5.x using dedicated stores per feature domain. Data fetching from the backend is handled by Axios. Form validation uses react-hook-form combined with Zod schema resolvers. Framer Motion provides micro-animations. The Google Generative AI JavaScript SDK connects the client directly to the Gemini 1.5 Flash model for AI-powered suggestions.

Package	Version	Role
next	15.1.0	React framework with App Router
react	19.0.0	UI library
zustand	5.0.x	Client-side state management
axios	1.7.x	HTTP client for API calls
@radix-ui/*	various	Accessible UI primitives
tailwindcss	3.4.x	Utility-first CSS framework
framer-motion	11.x	Animation library
react-hook-form	7.53.x	Form state and validation
zod	3.23.x	Schema validation
@google/generative-ai	0.24.x	Gemini AI SDK
@tanstack/react-query	5.59.x	Server state caching

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

III. SYSTEM ARCHITECTURE

3.1 Overall Architecture

Recipe Hub follows a decoupled client-server architecture. The frontend (Next.js) and backend (Express) are separate services communicating exclusively through a RESTful JSON API. During development both services run on localhost — the frontend on port 3000, the backend on port 5000. CORS is configured to allow credentials from the frontend origin, enabling cross-origin cookie transmission.

On startup, the backend connects to MongoDB Atlas (or a local MongoDB instance), then seeds an admin user if none exists. The Express application registers six route namespaces and applies a global error handler that normalizes all unhandled exceptions into a consistent JSON error envelope.

3.2 Route Architecture

The backend API is organized into six route modules, each mapped to a namespace:

Route Namespace	Module	Purpose
/api/auth	auth.js	Registration, login, logout, profile, token refresh
/api/recipes	recipes.js	Recipe CRUD, scaling, per-user stats
/api/admin	admin.js	Moderation, user management, dashboard stats
/api/feed	feed.js	Public feed, search, filter, trending, popular
/api/suggestions	suggestions.js	Ingredient-based matching and AI suggestions
/api/community	community.js	Voting, commenting, arena, voting history

3.3 Middleware Pipeline

Every request flows through the following Express middleware stack before reaching a route handler:

- CORS handler (credentials mode, strict origin allowlist)
- cookie-parser (parses http Only authToken cookie)
- express.json (body parsing, 10 MB limit)
- express.urlencoded (form data parsing, 10 MB limit)
- Route-specific validation middleware (Joi schemas)
- authenticate middleware (JWT verification from cookie)
- authorizeAdmin middleware (role check, admin routes only)
- Controller function (business logic, DB operations)
- Global error handler (catches unhandled errors, returns JSON)

IV. DATA MODELS

4.1 User

The User model enforces strict constraints at the schema level. Usernames must be 3–30 characters, alphanumeric with underscores, stored in lowercase. Emails are validated against an RFC-compliant regex and must be unique. Passwords are never stored in plaintext; a Mongoose pre-save hook hashes them with bcrypt (salt 12) every time the password field is modified. The password field carries select: false, preventing it from being returned in any query unless explicitly included with .select('+password'). A role field supports 'user' and 'admin' values. An isActive boolean allows account deactivation without deletion.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

4.2 Recipe

The Recipe model is the most complex in the system, embedding two sub-schemas: ingredientSchema and nutritionSchema. Ingredients capture name, quantity, and unit (validated against an enum of 34 allowed measurement units). Nutrition information is entirely optional but strongly typed when present. Core recipe fields include title (3–100 chars), description (10–500 chars), servings (1–50), cookingTime (1–1440 minutes), prepTime (1–720 minutes), difficulty (easy/medium/hard), and category (17 allowed values). Images are stored as URL strings with regex validation for supported formats. A status field ('pending'/'approved'/'rejected') drives the moderation workflow, with approval/rejection timestamps and actor references for audit logging. Engagement counters (viewCount, likeCount, commentCount) are denormalized onto the recipe for fast reads. A virtual totalTime getter sums prepTime and cookingTime. A scaleIngredients instance method accepts a new serving count, computes the ratio, and returns a scaled ingredient array rounded to two decimal places.

4.3 Comment

Comments support nested threading through a self-referential parentComment field. A null parentComment indicates a top-level comment; a populated ObjectId indicates a reply. The replyCount counter is maintained on the parent comment. An isEdited flag and editedAt timestamp provide transparency about modified content. Content is limited to 1,000 characters. A virtual 'replies' field is configured for population when needed.

4.4 Vote

The Vote model is intentionally minimal: user, recipe, and type ('like' or 'dislike'). A compound unique index on (user, recipe) enforces the one-vote-per-user-per-recipe constraint at the database level. This model drives both the social voting feature and the Arena leaderboard calculations.

4.5 Comment Like

A separate CommentLike model tracks which users have liked which comments, enabling the comment liking feature without denormalization of individual voter lists onto the Comment document.

V. API DESIGN

5.1 Design Principles

The API follows REST conventions: resource-oriented URLs, appropriate HTTP verbs, and consistent JSON envelopes. All responses share a top-level success boolean and a data or message field. Pagination responses include a pagination object containing currentPage, totalPages, totalRecipes, hasNextPage, and hasPrevPage. Error responses include a message string and, in development mode only, a stack field. All routes except login and signup require authentication. Admin routes additionally require role === 'admin'.

5.2 Key API Endpoints

Method	Path	Auth	Description
POST	/api/auth/signup	Public	Create account, set http Only cookie
POST	/api/auth/login	Public	Verify credentials, set http Only cookie
POST	/api/auth/logout	User	Clear authToken cookie
GET	/api/auth/check	User	Verify session validity
POST	/api/recipes	User	Submit recipe for moderation

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Method	Path	Auth	Description
GET	/api/recipes/my-recipes	User	List own recipes with status filter
POST	/api/recipes/:id/scale	User	Scale ingredient quantities
GET	/api/feed	User	Paginated approved recipe feed
GET	/api/feed/search	User	Full-text search across recipes
GET	/api/feed/trending	User	Trending recipes (recent + engagement)
GET	/api/feed/popular	User	Highest liked/viewed recipes
POST	/api/community/recipes/:id/vote	User	Like or dislike a recipe
POST	/api/community/recipes/:id/comments	User	Post top-level or reply comment
POST	/api/comments/:id/like	User	Toggle comment like
GET	/api/community/arena	User	Ranked recipe arena with filters
POST	/api/suggestions/recipes	User	Ingredient-based recipe matching
POST	/api/suggestions/pantry	User	Pantry-wide suggestion search
GET	/api/admin/stats	Admin	Platform-wide dashboard statistics
PUT	/api/admin/recipes/:id/approve	Admin	Approve a pending recipe
PUT	/api/admin/recipes/:id/reject	Admin	Reject with admin notes
GET	/api/admin/users	Admin	List all users with filters
PUT	/api/admin/users/:id/role	Admin	Promote/demote user role

VI. AUTHENTICATION AND SECURITY

6.1 Cookie-Based JWT Authentication

Recipe Hub uses a deliberate security architecture: JWT tokens are never exposed to client-side JavaScript. On login or registration, the server issues a JWT (signed with a secret from environment variables, configurable expiry) and writes it exclusively to an http Only cookie named authToken. The cookie carries sameSite: 'strict' to defend against CSRF, secure: true in production for HTTPS-only transmission, and a 7-day maxAge.

The authenticate middleware on every protected route reads the cookie via cookie-parser, calls verifyToken, looks up the user by ID from the decoded payload, attaches the user document to req.user, and proceeds. If the cookie is absent or the token is invalid, a 401 response is returned. The frontend never accesses the token directly — the browser transmits it automatically on every same-origin request.

6.2 Password Security

Passwords are hashed with bcrypt at a work factor of 12 (12 rounds of key stretching), making brute-force attacks computationally expensive. The comparePassword instance method on the User model safely compares a candidate password against the stored hash. The password field is excluded from all database query results by default via select: false.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

6.3 Input Validation

All incoming data is validated by Joi schemas in dedicated middleware modules before reaching controllers. Schema violations return 400 responses with structured error arrays. This prevents malformed or malicious data from reaching the database layer. Mongoose schema validation provides a second layer of defense.

6.4 Rate Limiting

Authentication endpoints are protected by an express-rate-limit configuration allowing a maximum of 5 attempts per 15-minute window. Exceeding this threshold returns a 429 response with a descriptive message, defending against credential stuffing attacks.

6.5 Authorization Model

Two middleware functions enforce access control beyond authentication. The authorizeAdmin middleware checks req.user.role === 'admin' and returns 403 otherwise. Business logic in controllers performs ownership checks: only the recipe creator can edit their own recipe, and editing is only permitted while the recipe remains in 'pending' status. Admin users can delete any recipe. View access to unapproved recipes is restricted to the recipe creator and admin users.

VII. AI INTEGRATION

7.1 Dual-Layer Suggestion Architecture

Recipe Hub implements two complementary suggestion mechanisms: a deterministic ingredient-matching engine operating against the community recipe database, and a generative AI layer powered by Google Gemini 1.5 Flash.

7.2 Ingredient Matching Engine

The suggestionService.js module implements a scoring algorithm for matching user-provided ingredients against community recipes. Each ingredient name is normalized by converting to lowercase, stripping trailing plural 's', and removing all non-alphanumeric characters. This reduces 'Tomatoes' and 'tomato' to the same canonical form. For each recipe, the algorithm computes exact matches (normalized user ingredient equals normalized recipe ingredient) and partial matches (one ingredient string contains the other as a substring). Exact matches contribute 100% of their proportional weight; partial matches contribute 50%. The composite matchPercentage is the sum of both weighted components, capped by total ingredient count. Recipes below a configurable minMatchPercentage threshold (default 30%) are filtered out. Missing ingredients — those in the recipe but not matched by the user's pantry — are listed per result to help users plan shopping.

The service also provides a popular ingredients endpoint using a MongoDB aggregation pipeline that unwinds the ingredients array across all approved recipes, groups by name, sorts by occurrence count descending, and returns the top N results. This powers autocomplete suggestions in the UI.

7.3 Gemini AI Integration

The Gemini Service class in gemini.js wraps the @google/generative-ai SDK. The service exposes three methods:

Generate Recipe Suggestions(ingredients, preferences): constructs a structured prompt requesting 3 diverse recipe suggestions as a JSON object with specific fields (title, description, cuisine, difficulty, timing, match Percentage, available/missing ingredients, health score, estimated cost). The Gemini 1.5 Flash model is invoked with temperature 0.7 and top P 0.8 for creative but coherent output.

generateDetailedRecipe(title, ingredients, preferences): given a recipe title selected from suggestions, generates a complete recipe including ingredients with quantities and units, step-by-step instructions with time estimates and tips, nutritional information, storage guidance, and variations.

generateIngredientAlternatives(ingredient): generates a JSON array of five cooking substitutes for any given ingredient.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

All prompts explicitly instruct the model to return ONLY valid JSON with no markdown formatting or prose explanations. The parseRecipeSuggestions and parseDetailedRecipe methods extract JSON from the response by finding the first '{' and last '}' characters, stripping any residual code fences, and calling JSON.parse. Parse failures fall back to a simplified static suggestion. The API key is sourced from the NEXT_PUBLIC_GEMINI_API_KEY environment variable — noting that this exposes the key to the browser bundle, a trade-off that should be addressed in a production deployment via a server-side proxy.

7.4 Ingredient Substitution Database

The backend maintains a static substitution dictionary covering common ingredients such as butter, milk, eggs, sugar, flour, chicken, beef, cream, and cheese, each with a list of alternatives. This provides instant substitutions without an AI call for common cases, reducing latency and API cost. The Gemini integration handles arbitrary ingredients not covered by the dictionary.

VIII. FRONTEND ARCHITECTURE

8.1 Next.js App Router Structure

The frontend uses Next.js 15's App Router with route groups to separate user and admin experiences. The (user) route group contains all authenticated user pages, wrapped by a layout.js that renders the UserHeader and UserSidebar components. The (admin) route group similarly wraps admin pages with AdminHeader and AdminSidebar. The auth directory holds the public login and register pages.

Route	Page	Description
/auth/login	login/page.js	Username/password authentication
/auth/register	register/page.js	Account creation with validation
/dashboard	dashboard/page.js	User home with recipe stats
/feed	feed/page.js	Approved recipe discovery feed
/my-recipes	my-recipes/page.js	User's recipe list with status
/my-recipes/create	create/page.js	Multi-step recipe creation form
/suggestions	suggestions/page.js	Ingredient-based + AI suggestions
/arena	arena/page.js	Community recipe leaderboard
/scale-recipe	scale-recipe/page.js	Recipe serving scaler
/profile	profile/page.js	User profile and activity feed
/admin/dashboard	dashboard/page.js	Admin statistics overview
/admin/recipes	recipes/page.js	Recipe management table
/admin/recipes/pending	pending/page.js	Pending moderation queue
/admin/users	users/page.js	User management and role control

8.2 State Management with Zustand

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Feature-specific Zustand stores replace a single global Redux-style store with composable, isolated slices. Each store encapsulates its own state, async actions, and loading/error states. This structure enables independent testing and reduces coupling between features.

Store	Manages
authStore	Current user, login/logout actions, auth check
adminStore	Pending recipes, all recipes, user list, approval actions
feedStore	Recipe feed, search, filters, pagination
myRecipesStore	User's own recipes, creation, deletion
suggestionsStore	Ingredient input, ingredient-match results, popular ingredients
aiSuggestionsStore	Gemini prompt state, AI suggestion results, detailed recipe
arenaStore	Arena recipes, leaderboard, timeframe/category filters, stats
recipeScalingStore	Recipe browser, selected recipe, scaled ingredient output
profileStore	Profile data, edit modal state, activity feed

8.3 Component Organization

Components are organized by domain alongside the UI primitive library. The /components/ui directory contains 20+ Radix UI-based primitives (Button, Card, Dialog, Select, Tabs, Table, etc.) providing the design system foundation. Domain components include:

admin/: AdminHeader, AdminSidebar, BulkActions, RecipeCard, RecipeDetailModal
arena/: ArenaLeaderboard, ArenaRecipeCard, ArenaStats
profile/: ActivityFeed, ProfileEditModal, ProfileStats
user/: UserHeader, UserSidebar
scaling/: RecipeScaleCard, ScaledIngredientsDisplay, RecipeGridSelector

IX. KEY FEATURES

9.1 Recipe Creation and Moderation Workflow

Recipe creation is a multi-step form in create/page.js supporting dynamic addition of ingredients and numbered instruction steps. Users submit images as external URLs. Upon submission, the recipe enters 'pending' status and is queued for admin review. Admins access the moderation queue at /admin/recipes/pending, where they can view full recipe details in a modal, approve with optional notes, or reject with required notes explaining the decision. Admins can also perform bulk approval or rejection via the BulkActions component. Approved recipes become visible in the public feed; rejected recipes are returned to the creator with admin notes.

9.2 Recipe Scaling

The Scale Recipe page at /scale-recipe allows users to browse all approved recipes and select one for scaling. The user enters a target serving count; the frontend dispatches a POST /api/recipes/:id/scale request with the new serving count. The backend controller calls recipe.scaleIngredients(newServings), which computes a ratio (newServings / originalServings) and multiplies each ingredient's quantity, rounding to two decimal places. The response returns both the original and scaled ingredient lists, the scaling ratio, and the recipe title, giving the user full context.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

9.3 Recipe Arena

The Arena feature ranks community recipes by engagement score within configurable timeframes (this week, this month, all time) and optional category filters. The communityController's getRecipeArena function uses a MongoDB aggregation pipeline that joins the recipes collection with votes, filters by status and date range, computes a composite score (likes, view count, comment count, and time-decay factor), and returns paginated ranked results. The frontend presents three tabs: Leaderboard (ranked cards with rank medals), Stats (aggregate platform metrics), and voting history. The ArenaStats component displays total recipes, total votes, most active categories, and top contributors.

9.4 Community Engagement

The voting system allows one vote per user per recipe. If a user submits the same vote type they previously cast, it is removed (toggle behavior). If they submit the opposite type, their vote is updated. After each vote operation, the system recomputes the current like and dislike counts with two countDocuments queries and returns them in the response, ensuring the UI always reflects the current state. The commenting system supports two-level nesting (top-level comments and direct replies). Users can edit their own comments, which sets the isEdited flag and editedAt timestamp. Comments can be individually liked through the CommentLike model. Admins can delete any comment; regular users can only delete their own.

9.5 User Profiles

The profile page displays aggregated statistics sourced from the backend: total recipes contributed, approval rate, total community engagement (views, likes, comments across all approved recipes), and an activity feed of recent actions. Users can update their email and profile image URL through the ProfileEditModal component. The ProfileStats component renders visual progress indicators for recipe status distribution.

X. LIMITATIONS AND FUTURE WORK

10.1 Current Limitations

API Key Exposure: The Gemini API key is stored in NEXT_PUBLIC_GEMINI_API_KEY, making it accessible in the browser bundle. A production deployment should proxy all Gemini calls through the backend.

No Image Upload: Recipe images are stored as external URLs rather than uploaded files. A production system would integrate cloud storage (e.g., AWS S3 or Cloudinary) for direct uploads.

No Email Verification: Users can register with any email address without verification, leaving the system open to spam accounts.

No Password Reset: The authentication system does not implement a password reset flow, which would require email integration.

Static Substitution Dictionary: The ingredient substitution database is a hardcoded dictionary covering a limited set of ingredients. A dedicated substitution database or LLM-backed endpoint would be more comprehensive.

No Real-Time Features: Voting, comments, and arena rankings require manual refresh. WebSocket integration (e.g., Socket.io) would enable live updates.

No Search Indexing: Text search is implemented via MongoDB's built-in text index on title, description, and tags. A dedicated search engine (e.g., Elasticsearch) would provide superior relevance ranking and faceted filtering.

10.2 Future Work

Server-Side Gemini Proxy: Move all LLM calls to a dedicated /api/ai backend namespace to protect API keys and enable server-side caching of common ingredient combinations.

User Following System: Allow users to follow creators and receive personalized feeds based on followed users' activity.

Recipe Collections: Enable users to save approved recipes to named private collections (e.g., 'Weeknight Dinners').

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 11, June 2026
Impact Factor: 8.2

Nutritional Analysis: Integrate with a nutritional API (e.g., USDA FoodData Central) to compute nutritional information automatically from ingredient lists.
Mobile Application: Port the frontend to React Native using a shared API layer.
A/B Testing for AI Prompts: Instrument prompt variants to measure suggestion quality and click-through rates.

XI. CONCLUSION

Recipe Hub demonstrates a practical, production-oriented full-stack web application combining modern JavaScript frameworks, MongoDB document modeling, JWT cookie authentication, and generative AI. The project achieves its core objectives: a community recipe platform with quality control through admin moderation, intelligent ingredient-based recipe discovery from both the community database and AI generation, and social engagement mechanics including voting, commenting, and a competitive leaderboard.
Key engineering decisions — storing JWTs in http Only cookies, enforcing vote uniqueness at the database index level, using Zustand stores per feature domain, and separating the AI suggestion layer from the community suggestion layer — reflect a thoughtful approach to security, data integrity, and maintainability. The platform provides a solid foundation for the future enhancements described in Section 10, particularly around real-time updates, search quality, and the server-side AI proxy pattern.

REFERENCES

[1]. Mongoose Documentation. (2024). Mongoose v7 API Reference. https://mongoosejs.com/docs/
[2]. Next.js Documentation. (2024). App Router. https://nextjs.org/docs/app
[3]. Zustand. (2024). Zustand State Management. https://zustand.docs.pmnd.rs/
[4]. Google. (2024). Gemini API Documentation. https://ai.google.dev/docs
[5]. jsonwebtoken. (2024). JWT Node.js Implementation. https://github.com/auth0/node-jsonwebtoken
[6]. bcryptjs. (2024). Password Hashing Library. https://github.com/dcodeIO/bcrypt.js
[7]. Radix UI. (2024). Primitive UI Components. https://www.radix-ui.com/
[8]. shadcn/ui. (2024). Component Library. https://ui.shadcn.com/
[9]. Express Rate Limit. (2024). express-rate-limit documentation. https://github.com/express-rate-limit/express-rate-limit