International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Design and Development of a Full-Stack AI Chat and Image Generation Platform (QuickGPT)

Mr. Ashish Ranjan Sahoo Student, Dept. of CSE, GIFT Autonomous Bhubaneswar

Mr. Pradyumna Baral Student, Dept. of CSE,GIFT Autonomous, Bhubaneswar

Dr. Satya Ranjan Pattnaik Assistant Professor, Dept. of CSE, GIFT Autonomous, Bhubaneswar

A*bstract*—This report documents the design and development of QuickGPT, a full-stack AI-powered chat and image generation platform built during an internship at the National Informatics Centre (NIC), Kendrapara. The platform integrates Google Gemini Flash as its core large language model, accessed via the OpenAI-compatible Node.js SDK, and uses ImageKit for AI image generation. The frontend is built with React 19, Vite, and Tailwind CSS, while the backend uses Express.js with MongoDB Atlas for data persistence and JWT-based authentication. A key innovation of the project is a custom real-time data pipeline that classifies user queries for live data requirements and fetches current information from external APIs—including Open-Meteo (weather), CoinGecko (cryptocurrency), Alpha Vantage (stocks), Wikipedia (factual and news), and CricketAPI (sports)—before injecting it into the Gemini prompt context. This effectively implements Retrieval-Augmented Generation (RAG) without a vector database. A Stripe-based credit and subscription system was designed and tested, though temporarily disabled in the live deployment. Both the React frontend and the Express backend are deployed independently on Vercel's serverless platform. The report covers the full system architecture, database design, backend and frontend implementation, real-time data pipeline, deployment strategy, testing, results, and future work.

***Index Terms*—Large Language Models, Generative AI, Retrieval-Augmented Generation, Full-Stack Development, React, Express.js, MongoDB, Gemini Flash, Real-Time Data, Serverless Deployment, AI Chatbot**

I. Introduction

A. Background

The proliferation of large language models (LLMs) has created new opportunities for building AI-powered web applica- tions accessible to general users. However, most commercially available AI chat interfaces impose usage limits, restrict concurrent queries, or require paid subscriptions for full functionality. Additionally, LLMs suffer from a fundamental limitation: their knowledge is bounded by a training data cutoff, making them unable to respond accurately to queries about current events, live market prices, or real-time conditions without external augmentation.

B. Motivation and Objectives

This project was developed as part of the academic curricu- lum at GIFT Autonomous, Bhubaneswar under the supervision of Er. Jagannath Ray, Assistant Professor, Dept. of CSE. The primary objective was to develop a full-stack AI chat applica- tion that provides users with fast, unlimited access to a capable LLM, supplemented by real-time data retrieval to overcome the training cutoff limitation. Secondary objectives included image generation capability, user authentication, a community gallery for shared content, and a scalable subscription model.

C. Scope of This Report

This report documents the complete development lifecycle of QuickGPT, covering:

- Background study of the technologies used.
- System architecture and full technology stack.
- Backend and frontend implementation details.
- Design and implementation of the real-time data pipeline.
- Deployment architecture on Vercel.
- Testing methodology and results.
- Observed challenges, known issues, and future develop- ment plans.

The live application is accessible at https://quickgpt-2.vercel. app.

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

II. Background and Technology Study

A. Generative AI on the Web

Large language models have made it feasible to integrate sophisticated natural language generation into web applications through API access. A key challenge encountered during development was rendering LLM responses that include code blocks in a structured, professional format. This was resolved using Prism.js for syntax highlighting, applied dynamically after each AI response is received.

B. Google Gemini Flash

Google Gemini Flash was selected as the core LLM for QuickGPT due to its low latency (typically one to three seconds for a response), its support for large context windows (approximately one million tokens), and its accessibility via an OpenAI-compatible API endpoint. Integration was accom- plished using the OpenAI Node.js SDK, which eliminated the need for custom HTTP call management.

C. trieval-Augmented Generation (RAG)

Retrieval-Augmented Generation is a technique that en- hances LLM responses by retrieving relevant information from external sources and injecting it as context before querying the model [**?**]. In QuickGPT, rather than using a traditional vector database, a custom keyword-based query classifier was implemented to detect real-time data requirements and fetch live information from domain-specific APIs. This constitutes a practical, lightweight RAG implementation suited to the project's deployment constraints.

D. Serverless Deployment

Both the frontend and backend are deployed on Vercel using serverless functions. In this model, the cloud provider manages infrastructure scaling dynamically. A known trade-off is the cold start delay: the first request after an inactive period incurs an additional 800–1500 ms of latency while the Node.js runtime initialises.

E. Stripe Payments

A subscription-based credit system was designed using Stripe Checkout. When a user initiates a purchase, a Stripe checkout session is created and the user is redirected to the Stripe payment page. On successful payment, Stripe sends a signed webhook to the backend server, which verifies the signature and credits the user's account. A critical technical challenge was that Stripe webhook verification requires the raw, unparsed request body, which is consumed by Express's `express.json()` middleware if registered first. This was resolved by registering the Stripe webhook route with `express.raw()` before the global JSON parser.

III. System Architecture

A. Development Environment

All development was performed on an ASUS laptop running Windows 11 (64-bit) with an AMD Ryzen 7 4000-series CPU and 16 GB RAM. The Node.js version used was v20 LTS.

B. Full Technology Stack

Table I presents the complete technology stack used in the project.

TABLE I
Full Technology Stack

Layer	Technology	Purpose
Frontend	React 19 Vite 7.x Tailwind CSS 4.x React Router 7.x Axios 1.13	UI component rendering Build tooling and HMR Styling Client-side navigation HTTP calls to backend
Backend	Express.js 5.x MongoDB + Mongoose 9.x JWT + bcryptjs	REST API framework Database and ODM Authentication
AI / APIs	Gemini Flash ImageKit Stripe 20.x	Core LLM AI image generation Payment processing
Hosting	Vercel MongoDB Atlas	Deployment platform Cloud database

C. High-Level Architecture

The system consists of three primary layers. The frontend is a React single-page application (SPA) that communicates exclusively with the backend via REST API calls. The backend is an Express.js server that handles authentication, chat management, AI inference calls, and payment processing. A real-time data pipeline layer sits between the backend message controller and the Gemini API call, checking each user prompt for live data requirements and enriching the prompt context if needed.

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

D. Data Flow

For a typical text message, the end-to-end flow proceeds as follows:

1) The user submits a prompt from the ChatBox component.
2) The frontend sends the prompt along with the chat ID and JWT to the backend.
3) The backend validates the JWT via the `protect` middleware.
4) The `contextMiddleware` injects the current date and time into the request.
5) The `isCurrentDataQuery()` function scans the prompt for live data keywords.
6) If live data is required, the appropriate real-time fetcher retrieves it from external APIs.
7) The full prompt—including date context, real-time data, and the original query—is assembled by `geminiContextFormatter`.
8) Gemini is called via the OpenAI SDK; the response is returned to the client immediately.
9) The chat document is persisted to MongoDB asyn- chronously after the response is sent.

E. Database Design

1) User Collection: Stores account data. Passwords are never stored in plaintext; bcrypt automatically hashes them before persistence. Each user is initialised with 20,000 credits (currently unlimited during the promotional period).

2) Chat Collection: Each document represents one conver- sation, with all messages embedded as a subdocument array. Each message stores: `role` (user or assistant), `content` (text or image URL), `isImage` (Boolean), `isPublished` (Boolean, for the community gallery), `timestamp`, and a `dateContext` object.

3) Transaction Collection: Tracks payment attempts. A transaction record is created when a user initiates a purchase and its `isPaid` flag is set to `true` only after Stripe's webhook confirms payment.

IV. Backend Implementation

A. Server Entry Point and Middleware

The server entry point is `server.js`. Middle- ware is registered in the following strict order: (1) `connectDB()`, (2) Stripe webhook handler with `express.raw()`, (3) `cors()`, (4) `express.json()`, and (5) `contextMiddleware`. The ordering of the Stripe route before the global JSON parser is critical for webhook signature verification.

B. Authentication System

On successful login, the server generates a JWT signed with `JWT_SECRET` that expires after 30 days. The `protect` middleware validates the JWT on every protected route and attaches the user object to the request for downstream use. Passwords are compared using `bcrypt.compare()`. Error messages are deliberately generic (e.g., "Invalid email or password") to prevent enumeration attacks.

C. API Routes

Table II summarises the complete API route reference.

TABLE II
API Route Reference

Method	Endpoint	Auth	Function
POST	/api/user/register	No	Create account, return JWT
POST	/api/user/login	No	Verify credentials, retur JWT
GET	/api/user/data	Yes	Fetch user profile and cred its
GET	/api/user/published-images	No	Fetch community images
GET	/api/chat/create	Yes	Create new chat session
GET	/api/chat/get	Yes	Fetch all user chats
POST	/api/chat/delete	Yes	Delete a specific chat
POST	/api/message/text	Yes	Send prompt, return AI reply
POST	/api/message/image	Yes	Generate AI imag
GET	/api/credit/plan	No	Return subscription plan
POST	/api/credit/purchase	Yes	Create Stripe checkout se sion
POST	/api/stripe	No	Handle Stripe webhook

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

D. Message Controller

The `textMessageController` is the most complex component in the backend. It orchestrates the full pipeline: loading the chat from MongoDB, classifying the prompt, fetching real-time data if needed, assembling the enriched prompt, calling Gemini, returning the response to the client, and persisting the updated chat asynchronously. Responding before saving reduces perceived latency for the user.

The `imageMessageController` constructs an Im- ageKit image generation URL from the user's prompt, uploads the result to ImageKit for hosting, and returns the hosted URL to the client.

TABLE III SUBSCRIPTION PLANS

Plan ID	Name	Price (USD)	Credits
basic	Basic	$10	100
pro	Pro	$20	500
premium	Premium	$30	1,000

V. Frontend Implementation

A. Application Entry and Routing

The SPA is bootstrapped in `main.jsx`. The `App` com- ponent applies two routing branches based on authentication state: unauthenticated users see the Login component, while authenticated users see the Sidebar with React Router rendering ChatBox, Credits, and Community pages.

B. Global State Management

Application state is managed with React's built-in Con- text API in `AppContext.jsx`, avoiding the overhead of an external state management library. The context stores: `user`, `token`, `chats`, `selected` (current chat), `theme`, and `loadingUser`. Three `useEffect` hooks manage the application lifecycle for authentication, chat loading, and dark mode.

C. ChatBox Component

The ChatBox component occupies the main content area. On message submission, the user's message is optimistically added to the local state and the input is cleared before the API call completes. On API failure, the original prompt is restored. A `useEffect` on the messages array ensures auto-scrolling to the latest message. A mode toggle switches between text and image generation endpoints.

D. Message Rendering

User messages are right-aligned with a white background; AI responses are left-aligned with a purple background. AI response text is rendered through `react-markdown` to pro- duce proper HTML formatting for bold text, code blocks, and lists. PrismJS syntax highlighting is applied via `useEffect` whenever message content changes. Image responses are rendered using a standard `<img>` tag with the ImageKit-hosted URL.

E. Sidebar

The Sidebar is always visible on desktop and slides in from the left on mobile. It contains the application logo, a new chat button, a chat search box, the chat session list, the community page link, a credits display, a dark mode toggle, and a user name with a logout button on hover.

VI. Real-Time Data Pipeline

E. Subscription Plans

Three subscription plans are defined, as shown in Table III. The credit deduction system is currently suspended for the promotional unlimited-access period.

A. Motivation

Gemini Flash, like all LLMs, has a training data cutoff that prevents accurate responses to queries about current events, live prices, or real-time conditions. To address this, a custom

real-time data pipeline was designed and implemented as a lightweight RAG layer that enriches prompts with live data before they are sent to Gemini.

B. Query Classification

The function `isCurrentDataQuery()` performs

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

an ini- tial keyword scan to determine whether the prompt requires live data. If affirmative, `detectQueryType()` classifies the query into one of several categories. Table IV shows the categories, example keywords, and corresponding data sources.

TABLE IV
QUERY CATEGORIES AND REAL-TIME DATA SOURCES

Category	Example Keywords	Data Source
Weather	weather, tempera- ture, forecast, rain	Open-Meteo
Crypto	bitcoin, ethereum, btc, eth	CoinGecko
Stock	stock, nasdaq, tesla, trading	Alpha Vantage
News	news, breaking, lat- est, today's	Wikipedia Recent Changes
Sports	ipl, cricket, match, live score, nba	CricketAPI / CricAPI
Factual	who is, president, prime minister	Wikipedia REST API
Trending	trending, viral, pop- ular	Wikipedia Random/Recent

A single query may match multiple categories simultaneously, in which case the relevant fetchers execute in parallel and their results are concatenated into a single context block.

C. External API Integrations

1) Weather (Open-Meteo): The location name is extracted from the prompt (defaulting to "New Delhi" if absent). Open- Meteo's geocoding API converts the name to coordinates, which are then used to fetch current temperature, humidity, wind speed, and weather condition. Open-Meteo requires no API key.

2) Cryptocurrency (CoinGecko): CoinGecko's free API returns current USD price, 24-hour change, volume, and market cap for identified coins. API calls are wrapped in a 3-second timeout to guard against rate-limit delays.

3) Stocks (Alpha Vantage): Alpha Vantage provides stock market data. The integration currently uses a demo key, which only returns data for IBM. Production deployment requires a registered personal key.

4) News and Factual Data (Wikipedia): For factual queries, the Wikipedia Search API identifies the most relevant article title, and the REST summary endpoint returns a 2–4 sentence in- troduction injected as context. For news queries, the Wikipedia Recent Changes API surfaces recently created articles as a proxy for current events.

5) Sports (CricketAPI and CricAPI): Sports integration presented the greatest challenge due to inconsistent API uptime and non-uniform date formats across providers (Unix seconds, Unix milliseconds, and numeric strings). A helper function `isTodaysMatch()` was written to normalise all date formats before comparison. The system first attempts CricketAPI; on failure, it falls back to CricAPI. If both fail, a graceful message is returned to the user suggesting ESPNcricinfo.

D. Caching Strategy

An in-memory cache with per-entry TTL timers was imple- mented to reduce redundant API calls. Queries are normalised into shared cache keys (e.g., all cricket-related prompts map to `sports_YYYY-MM-DD`). Table V shows the TTL values and their rationale.

TABLE V
CACHE TTL VALUES BY DATA TYPE

Data Type	TTL (s)	Reasoning
Crypto	180	Prices change rapidly
Weather	600	Conditions change slowly
Sports (match found)	900	Mid-game data is stable
Sports (no match)	60	Retry quickly for upcoming matches
Stock	300	Free API has inherent delays
Wikipedia / Factual	600	Content changes rarely
News / Trending	300	Balance freshness with requests

A known limitation is that this cache is process-local: on Vercel's serverless platform, each cold start initialises a new empty cache. Production-scale deployments would benefit from a shared Redis cache (e.g., Upstash Redis).

E. Prompt Construction

The enriched prompt is assembled by `geminiContextFormatter` and placed entirely in a user-role message. Using the system role was found to be unreliable with Gemini Flash via the OpenAI-compatible endpoint, as it sometimes ignored system messages. The prompt structure is:

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

```
[Context: Today is <date>. Time:
<time>. Timezone:

[REAL-TIME DATA]
<catego
ry
heading
>
<fetche
d data>

User Query: <original prompt>
```

VII. Deployment

A. Frontend Deployment

The frontend is deployed on Vercel with a `vercel.json` rewrite rule that redirects all paths to `index.html`, enabling client-side routing without 404 errors on direct URL access.

B. Backend Deployment

The backend is deployed as a separate Vercel project using the `@vercel/node` adapter, which wraps Express as a serverless function. The `app.listen()` call is ignored by Vercel in production but remains functional for local development.

C. Environment Variables

All secrets are managed through Vercel's project dashboard and are never committed to the repository. Key variables include `MONGODB_URI`, `JWT_SECRET`, `GEMINI_API_KEY`, `IMAGEKIT_PUBLIC_KEY`, `IMAGEKIT_PRIVATE_KEY`, `STRIPE_SECRET_KEY`, and `STRIPE_WEBHOOK_SECRET`.

D. Post-Deployment Issues Identified

- The client `.env` file retains `VITE_SERVER_URL=localhost:3000` from development; the
- The compiled `dist/` build folder is committed to the Git repository; this should be excluded via `.gitignore` and built by Vercel from source.
- Cold starts add 800–1500 ms to the first request after more than 30 minutes of inactivity.

VIII. Testing and Validation

A. Test Scripts

Over 20 test and diagnostic scripts were written during development, covering: quick end-to-end integration tests for all real-time fetchers, sports API consistency tests, full system production validation runs, and error scenario tests for API failures and network timeouts.

B. Manual Testing

Manual testing covered: user registration and login (including duplicate email rejection and JWT expiry), chat creation and deletion (verifying that users cannot delete other users' chats), text and image generation in both modes, real-time pipeline verification for each query category, and community image publishing.

C. Test Results Summary

Table VI presents the key test results from manual validation.

TABLE VI
Manual Test Results Summary

Feature	Type	Result
User registration (new)	Manual	Pass
Registration (duplicate email)	Manual	Pass
Login (correct credentials)	Manual	Pass
Login (wrong password)	Manual	Pass
JWT expiry redirect	Manual	Pass
Text message (no RAG)	Manual	Pass
Text message (weather RAG)	Manual	Pass
Text message (crypto RAG)	Manual	Pass
Text message (factual RAG)	Manual	Pass
Text message (sports RAG)	Manual	Partial
Text message (stock RAG)	Manual	Partial
Image generation	Manual	Pass
Image published to gallery	Manual	Pass
Chat deletion (own)	Manual	Pass
Chat deletion (other user)	Manual	Pass (blocked)
Dark mode toggle	Manual	Pass
Mobile sidebar toggle	Manual	Pass
Stripe checkout (test mode)	Manual	Pass
Stripe webhook (test mode)	Manual	Pass

Sports and stock results are marked partial due to inconsistent third-party API availability and the use of a demo key for Alpha Vantage, respectively.

IX. Results and Performance

A. Functional Outcomes

QuickGPT is fully deployed and operational at https:// quickgpt-2.vercel.app. All primary user flows are functional: account creation and login, multi-turn chat, AI image genera- tion, community gallery publishing, and dark mode. The Stripe

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

subscription system is implemented and tested in test mode; restoring full functionality requires only uncommenting credit deduction logic.

B. Response Time Observations

Table VII shows observed end-to-end response times for different request types.

TABLE VII
OBSERVED RESPONSE TIMES BY REQUEST TYPE

Request Type	Time (s)	Bottleneck
Text (no RAG)	1.5 – 3.5	Gemini API latency
Text (weather RAG)	2.5 – 5.0	Open-Meteo geocoding + fetch
Text (crypto RAG)	2.0 – 4.5	CoinGecko response time
Text (stock RAG)	1.5 – 3.5	Limited demo key data
Text (sports RAG)	3.0 – 7.0	Sports API response time
Image generation	6.0 – 15.0	ImageKit AI generation

Cold starts add a further 800–1500 ms to first requests. Streaming responses would substantially improve perceived latency by delivering tokens progressively.

X. CHALLENGES AND KNOWN ISSUES

A. Technical Challenges

1) Sports API Date Parsing: Cricket APIs returned match dates in three different formats: Unix seconds, Unix millisec- onds, and numeric strings. A unified normalisation function `isTodaysMatch()` was written to detect the format and convert all dates to a common local timezone representation.

2) Stripe Webhook Raw Body: Stripe webhook verification requires the original raw request bytes, which are consumed by `express.json()` before the webhook handler can read them. The resolution was to register the Stripe route with `express.raw()` before any global JSON parsing middleware.

3) Gemini System Role Behaviour: Gemini Flash via the OpenAI-compatible endpoint inconsistently ignored context injected via the `system` role. All context was consolidated into the user message to ensure reliable behaviour.

4) Serverless Cache Isolation: The in-memory cache is process-local. On Vercel's serverless platform, every cold start creates a new instance with an empty cache, limiting the effectiveness of the caching layer under real traffic. Upstash Redis is recommended for a production fix.

5) React Context Circular Dependency: In `AppContext.jsx`, `fetchUsersChats()` calls `createNewChat()` when the chat list is empty, and `createNewChat()` calls `fetchUsersChats()` after creation. Careful function declaration ordering was required to prevent infinite recursion.

B. Known Issues

Table VIII catalogues known issues with severity ratings and recommended fixes.

TABLE VIII
KNOWN ISSUES WITH SEVERITY AND FIXES

#	Issue	Severity	Fix
1	`VITE_SERVER_URL` set to localhost in .env	Critical	Set correct URL in Vercel dashboard
2	Credit deductions disabled in 3 controllers	High	Re-enable and test Stripe flow
3	Webhook endpoint unauthenticated	High	Remove or protect the route
4	Alpha Vantage uses demo key	High	Register and configure real key
5	Chat create uses GET method	Medium	Change to POST in chatRoutes
6	Test files committed to repo	Medium	Move to test directory
7	In-memory cache lost on cold start	Medium	Integrate Redis
8	No React Error Boundary	Medium	Add ErrorBoundary component
9	Build folder committed to Git	Medium	Add to .gitignore
10	getChats loads all messages at once	Medium	Add query limit to getChats
11	No streaming response	Medium	Implement token streaming
12	Unused imports in AppContext.jsx	Low	Remove unused imports

XI. CONCLUSION AND FUTURE WORK

A. Conclusion

QuickGPT demonstrates the feasibility of building a production-deployed, full-stack AI chat and image generation platform in a short development cycle. The project delivered all primary objectives: fast LLM-powered chat, AI image generation, user authentication with JWT, MongoDB-backed chat

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

persistence, real-time data enrichment via a custom RAG pipeline, and a working (though disabled) Stripe subscription system. The real-time data pipeline—classifying queries, fetch- ing live data, and injecting it into the Gemini prompt—is the most technically novel contribution of the project and represents a practical implementation of RAG without a vector database. The project provided hands-on experience across the full stack: database schema design, RESTful API development, React state management, serverless deployment, webhook handling, and third-party API integration. It was carried out under the guidance of Er. Jagannath Ray, Assistant Professor,
Dept. of CSE, GIFT Autonomous, Bhubaneswar.

B. Future Work

The following improvements are planned for future devel- opment:

- **Redis-Backed Cache**: Replace process-local in-memory cache with Upstash Redis for persistent, shared caching across serverless instances.
- **Multi-Turn Conversation History**: Send full chat history with each Gemini API call to enable contextual follow-up responses.
- **Re-enable Credit System**: Restore credit deductions, run full Stripe test-mode validation, and switch to live mode.
- **Streaming Responses**: Implement token-by-token stream- ing from Gemini to significantly reduce perceived response latency.
- **Auto-naming Chats**: After the first AI reply, request a 3–5 word title from Gemini based on the conversation content.
- **Document RAG with Vector Database**: Allow users to upload PDFs and query their contents using MongoDB Atlas Vector Search, Pinecone, or Weaviate.
- **Multi-LLM Routing**: Revisit the original vision of routing queries to multiple LLMs simultaneously using free or affordable API keys.

XII. System Evaluation and Discussion

A. Comparative Analysis of RAG Approaches

Traditional Retrieval-Augmented Generation systems rely on a vector database (e.g., Pinecone, Weaviate, or MongoDB Atlas Vector Search) to embed documents and perform se- mantic similarity searches at query time [19]. While powerful for document-grounded QA, these systems carry operational overhead: embedding models must be run on each ingested document, vector stores must be provisioned, and retrieval pipelines add latency.

QuickGPT's pipeline takes a lighter-weight, domain-specific approach: structured APIs with known schemas replace embedding-based retrieval. The trade-off is reduced generality (only supported domains can be enriched) in exchange for lower latency, zero vector infrastructure, and highly structured data. Table IX compares the two paradigms.

TABLE IX
Comparison: Vector-DB RAG vs. QuickGPT's API-Based RAG

Criterion	Vector-DB RAG	API-Based RAG
Infrastructure	Vector store required	No extra store
Latency	Medium (embed + search)	Low (direct API call)
Data freshness	Depends on ingestion	Always live
Domain coverage	Arbitrary documents	Predefined APIs only
Scalability	High	Bounded by API lim-its
Setup complexity	High	Low

B. Security Analysis

Table X assesses the security posture of the current deploy- ment and recommends mitigations for identified gaps.

TABLE X Security Assessment

Area	Status	Recommendation
JWT Authentication	Implemented	Reduce expiry to 7 days
Password Hashing	bcrypt (secure)	No change needed
Stripe Webhook	Signature checked	Remove debug endpoint
CORS	Enabled	Restrict to frontend do-main
Rate Limiting	Not implemented	Add express-rate-limit
Input Validation	Partial	Add Joi/Zod schemas
HTTPS	Vercel default	No change needed
Secrets Management	Vercel dashboard	Rotate keys periodically

C. Scalability Considerations

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Vercel's serverless model provides horizontal scaling auto- matically, but three bottlenecks limit throughput at scale:

1) **In-Memory Cache**: Each serverless instance maintains its own cache. Replacing it with Upstash Redis provides a shared, persistent cache across all instances.
2) **MongoDB Atlas Free Tier**: The current deployment uses the free M0 cluster, which limits concurrent connections and IOPS. Upgrading to M10 or above is necessary for production load.
3) **Third-Party API Rate Limits**: CoinGecko's free tier allows approximately 30 calls per minute. At high traffic, the crypto pipeline would need caching with longer TTLs or a paid API plan.

D. Academic Contribution and Learning Outcomes

This project contributes to the learning outcomes of the B.Tech. CSE programme at GIFT Autonomous in the following areas:

- **Software Engineering**: Full SDLC experience from requirements to deployed production system.
- **Web Technologies**: Practical application of React, REST APIs, and MongoDB in a production context.
- **Artificial Intelligence**: Integration of LLMs, prompt engineering, and lightweight RAG design.
- **Cloud Computing**: Serverless deployment, environment configuration, and cold-start optimisation.
- **Security**: JWT-based authentication, bcrypt hashing, and Stripe webhook signature verification.

E. Limitations

The following limitations are acknowledged in the current version of the platform:

- **No Conversation Memory**: Each Gemini API call receives only the latest user message and injected context; prior conversation turns are not included. This prevents contextual follow-up responses.
- **Sports API Fragility**: Live cricket match data depends on third-party APIs with inconsistent uptime. No authoritative fallback exists beyond ESPNcricinfo redirection.
- **Credit System Disabled**: The Stripe integration is complete and tested in test mode, but credit deductions have been commented out pending final QA. Users currently have unlimited access regardless of credit balance.
- **No Streaming**: Gemini's token-by-token streaming is not utilised; the entire response is buffered before transmission, increasing perceived latency for long responses.

ACKNOWLEDGMENT

The authors thank Er. Jagannath Ray, Assistant Profes- sor, Dept. of CSE, GIFT Autonomous, Bhubaneswar, for his invaluable guidance, mentorship, and continuous support throughout the development of this project. The authors also acknowledge the open-source communities behind React, Express.js, MongoDB, Tailwind CSS, and Vite, and Google for providing the Gemini Flash API.

REFERENCES

[1] Google, "Gemini API Documentation," 2026. [Online]. Available: https: //ai.google.dev/gemini-api/docs
[2] Google, "Gemini Flash Model Specifications," 2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/models/gemini
[3] OpenAI, "OpenAI Node.js SDK," GitHub repository, 2025. [Online]. Available: https://github.com/openai/openai-node
[4] Vercel, "Vercel Platform Documentation," 2025. [Online]. Available: https://vercel.com/docs
[5] React, "React Documentation – React 19," 2025. [Online]. Available: https://react.dev
[6] Vite, "Vite Documentation v7," 2025. [Online]. Available: https://vite. dev/guide
[7] Tailwind CSS, "Tailwind CSS v4 Documentation," 2025. [Online]. Available: https://tailwindcss.com/docs
[8] React Router, "React Router v7 Documentation," 2025. [Online]. Avail- able: https://reactrouter.com/en/main
[9] MongoDB, "MongoDB Atlas Documentation," 2025. [Online]. Available: https://www.mongodb.com/docs/atlas
[10] Stripe, "Stripe API Reference," 2025. [Online]. Available: https://stripe. com/docs/api
[11] Stripe, "Stripe Webhooks – Signature Verification," 2025. [Online]. Available: https://stripe.com/docs/webhooks
[12] ImageKit, "ImageKit Documentation – AI Image Generation," 2025. [Online]. Available: https://docs.imagekit.io
[13] CoinGecko, "CoinGecko API v3 Free Tier

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Documentation," 2025. [Online]. Available: https://docs.coingecko.com/reference/introduction

[14] Open-Meteo, "Open-Meteo Free Weather API," 2025. [Online]. Available: https://open-meteo.com/en/docs

[15] Alpha Vantage, "Alpha Vantage Stock Market API Documentation," 2025. [Online]. Available: https://www.alphavantage.co/documentation

[16] Wikipedia, "MediaWiki API," 2025. [Online]. Available: https://en.wikipedia.org/w/api.php

[17] PrismJS, "PrismJS Syntax Highlighting," 2025. [Online]. Available: https://prismjs.com

[18] J. M. Carstens, "react-markdown package," 2025. [Online]. Available: https://github.com/remarkjs/react-markdown

[19] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge- Intensive NLP Tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020. arXiv:2005.11401.

[20] W. X. Zhao et al., "A Survey of Large Language Models," arXiv preprint arXiv:2303.18223, 2023.

[21] A. R. Sahoo and P. Baral, "QuickGPT – Full-Stack AI Chat Application," B.Tech. Project Report, Dept. of CSE, GIFT Autonomous, Bhubaneswar, 2026. [Online]. Available: https://quickgpt-2.vercel.app

[22] Babburi, S. Privacy-Preserving Collaborative Framework with Auditable Federated Learning.

[23] Gaddam, S. (2024). Integrating machine learning models with continuous integration and continuous delivery (CI/CD) pipelines for a learning-driven approach to software engineering.

[24] Reddy, S. K. R. Developing a Modular AI Framework to Enhance Scalability and Personalization in Next-Generation Reward Platforms.

[25] Poojari, R. INTELLIGENT SYSTEMS+B108 AND APPLICATIONS IN ENGINEERING.

[26] Vasagam, M. (2024, August 30). Ensuring security in modern data pipelines: Practical strategies for data engineers. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2401.

[27] Santthosh Saai Reddy Purmani. (2026). Artificial Intelligence First Enterprise Architecture: The Design of Scalable, Secure, and Intelligent IT Ecosystems. American Journal of AI Cyber Computing Management, 6(1(2)), 1–8. https://doi.org/10.64751/ajaccm.2026.v6.n1(2).pp1-8

[28] Purmani, S. S. R. (2025). Streamlining IT operations and service management with agile frameworks. European Journal of Advances in Engineering and Technology, 12(4), 76–81.

[29] Kotte, G. (2025). Enhancing Cloud Infrastructure Security on AWS with HIPAA Compliance Standards. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283660

[30] Kumara, S. (2026, February). A Lightweight Deep Learning Based Classification Models for Non-Human Identity Threat Detection. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-6). IEEE.

[31] Kotte, G. (2025). Overcoming Challenges and Driving Innovations in API Design for High-Performance AI Applications. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.5283649

[32] Mahtabi, M., Roshan, M., Muhit, M. M. I., Behvar, A., & Haghshenas, M. (2026). Cryogenic ultrasonic fatigue: Mechanisms, advancements, and insights. Cryogenics, 153, 104257. https://doi.org/10.1016/j.cryogenics.2025.104257

[33] Viswanathan, V. (2023). AI-Augmented Decision Intelligence for Enterprise Systems: Integrating Cognitive Analytics for Resource and Talent Optimization.

[34] Viswanathan, V. (2025). Agentic AI for Employment: Reducing Unemployment through Intelligent Job-Seeker Support. LEX LOCALIS–Journal of Local Self-Government.

[35] Mudusu, S. (2025). Health Insurance Fraud Detection: The Role Of Advanced It Systems In Preventing And Identifying Fraud. International Journal, 16(1), 3769-3777

[36] Mudusu, S. K. (2026, February 9). AI-augmented data quality engineering. InfoWorld (Foundry Expert Contributor Network).

[37] Agrawal, A. M., Gajula, S., Shinde, R. P., Shah, H., & Ghosh, H. (2025, July). Machine Translation for Long Sequences with Enhanced Attention Mechanisms. In 2025 5th International Conference on Electrical, Computer and Energy Technologies (ICECET) (pp. 1-6). IEEE.

[38] Maturi, S. Y. (2023). Crowdsourced frontier: Unveiling autonomous adversarial cybercapabilities via open AI competition. International Journal of Intelligent Systems and Applications in Engineering, 11(1s), 275–284.

[39] Sikder, M. Z., Shakil, M. A. I., Ahad, A., Karim, M. F., Intakhab, B., & Islam, D. A. (2025, June). Microwave-Based Detection of Early-Stage Renal Cell Carcinoma Using UHF Range Antenna. In 2025 International Conference on Computer Systems and Technologies (CompSysTech) (pp. 1-6). IEEE.

[40] Manoharan, D. (2024). Governance-Oriented Quality Engineering Framework for Healthcare EDI Modernization. International Journal of Multidisciplinary on Science and Management IJMSM, 1(2).

[41] Ravishankara, M. (2026, February). PlotChain: Deterministic Checkpointed Evaluation of Multimodal LLMs on Engineering Plot Reading. In SoutheastCon 2026 (pp. 1-8). IEEE.

[42] Doragacharla, V. R. (2026). Building Real-Time Pricing Systems for Modern Retail. Available at SSRN 6451760.

[43] Adabala, P. K. (2024). Utilizing predictive analytics to improve efficiency and decision-making in ERP-connected supply chains. International Journal of Intelligent Systems and Applications in Engineering, 12(22s), 2465

[44] Kavuri, S. (2026). An Explainable Machine Learning Framework for Predicting Software Defects in Large-Scale Software Systems. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), 1–6. https://doi.org/10.1109/icaic67076.2026.11395777

[45] Venkata Pavan Kumar Gummadi. (2023). MuleSoft Batch Processing: High-Volume

International Journal of
LAW, ARTS AND HUMANITIES
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0523 www.zesterapublications.com Original Research Paper

Streaming Architecture. Computer Fraud and Security, 50–57. https://doi.org/10.52710/cfs.886

[46] Gummadi, V. P. K., Chilamkurthi, L. S., & Kavuri, S. (2026). Securing APIs in Government Clouds and Runtime Fabric Using FIPS-Enabled MuleSoft. 2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISET), 1–6. https://doi.org/10.1109/icaiset66439.2026.11542099

[47] Gajula, S., & Margam, M. (2026). A Secure and Scalable Cloud-Based Banking Service Model Leveraging AI and Advanced Cyber Security. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC), 1–5. https://doi.org/10.1109/icaic67076.2026.11395704

Received: 05-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026