WWW.IJRPR.COM

International Journal of Research Publication and Reviews

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

SPARK: An AI-Powered Collaborative Browser IDE for Instant Generation and One-Click Deployment of Small Software

Samyukt Nambiar R[1], Saravana Ganesh[2], Sasmitha KV[3] and Sadhana V[4]

[1]Member, CSE-D, SNS College of Technology, Coimbatore, Tamil Nadu
[2]Member, CSE-D, SNS College of Technology, Coimbatore, Tamil Nadu
[3]Member, CSE-D, SNS College of Technology, Coimbatore, Tamil Nadu
[4]Member, CSE-D, SNS College of Technology, Coimbatore, Tamil Nadu
Corresponding author: Samyukt Nambiar R (samyukt.r.cse.2025@snsct.org)

ABSTRACT—

Background— AI coding assistants let non-specialists create small tools within minutes, but sharing them still requires cloud accounts and deployment knowledge. Methods— SPARK (Small Platform for Agile Rapid Kreation) is a browser-based, AI-assisted development and deployment platform. A text or voice description is sent to an n8n-hosted Gemini workflow with a four-stage fallback chain, and the returned React code is parsed, reviewed, and rendered in a sandboxed preview. Teams collaborate on a shared canvas, and a one-click module publishes the app through the Vercel API. Results— The production build completed (2,072 modules in 3.28 s), and all six response-parsing tests passed. Generation success, latency, and deployment reliability were not measured. Conclusion— SPARK integrates generation, preview, collaboration, and publication in one workflow and is intended for academic demonstration only.

***Index Terms*—Artificial Intelligence, Browser IDE, Code Generation, Large Language Models, One-Click Deployment, React, Real-Time Collaboration, Small Software, Voice-Driven Development, Workflow Automation.**

***Plain Language Summary*—**People without programming training can now ask an AI assistant to write a small web tool, but sharing that tool with colleagues is still hard because it has to be placed on a server first. SPARK is a prototype website in which a user types or speaks what the tool should do, sees the result straight away, refines it together with teammates, and publishes it as a web link with one click. If one AI service fails, SPARK switches to backup options so that the user still receives a working result. In our tests, the system built correctly and read all six test AI replies correctly. We did not yet measure how often the AI produces correct tools, how fast it is, or how real users get on with it. SPARK is a demonstration and is not ready for everyday company use.

Word count: 2,460 words (main text, Sections I–V, excluding abstract, plain language summary, tables, figure captions, declarations, annexure, and references); abstract: 128 words.

I. INTRODUCTION

Software creation has become remarkably easy. A marketing specialist or an operations lead can describe a repetitive task to an AI coding agent and receive a working web tool within minutes, because large language models trained on source code have matured [1]–[3]. The difficulty appears at the next step: a tool on one person's laptop cannot be used by a teammate until it is deployed, and deployment is still designed for professionally operated systems.

Traditional cloud platforms expect users to provision servers, configure routing, package containers, and define access rules. For a tool generated in two minutes, this effort is disproportionate, so useful tools stay on local machines or are handed over as terminal commands that non-technical colleagues cannot follow. We call such purpose-built tools *small software*: applications that should be as easy to share as a shared document.

SPARK is a browser-based workflow for creating and delivering small software. A user opens a workspace and describes the required application by typing or speaking. SPARK generates React code through a Gemini-based pipeline, shows a live preview, lets the team refine it on a shared canvas, and publishes it to a public URL with one click. SPARK is an academic prototype, and generated code is treated as untrusted output that must be reviewed.

A. Problem Statement

A local tool cannot be shared without deploying it to a cloud provider, and conventional cloud services were designed for enterprise-scale software that needs manual setup of servers, networking, and security rules [4]. Existing workarounds fall short: clouds demand DevOps knowledge, screen sharing locks the tool on one laptop, and terminal hand-offs fail for non-technical colleagues.

The consequences are practical. Useful tools remain siloed, and IT administrators face shadow IT, in which company data is uploaded to unvetted third-party sites. The project's problem analysis estimates that deploying a tool generated in two minutes can take two or more hours; this practitioner estimate has not been validated experimentally.

A lightweight platform is therefore needed that lets a user describe a tool, preview it safely, collaborate on it, and publish it without infrastructure knowledge, while continuing to work when an individual AI service fails.

B. Research Gap

The major gaps in existing approaches are as follows.

1) **Deployment friction:** code-generation assistants stop at the editor; hosting remains a separate, expert task.
2) **Fragmented tooling:** prompting, previewing, collaboration, database setup, and deployment use different tools.
3) **Limited accessibility:** few environments let non-technical users create and share applications by voice.
4) **Unreliable output handling:** model responses vary in format and may be empty, malformed, or wrapped in service metadata.
5) **Weak recovery and governance:** error recovery, rollback, access roles, and secret handling are rarely connected to generation.

These gaps motivate SPARK, which integrates generation, review, preview, collaboration, database provisioning, and one-click deployment in one prototype.

II. LITERATURE SURVEY

Transformer-based sequence models [5] and scaled language models [6] established that natural-language instructions can be converted into program text. Code-tuned models solve many programming tasks [1], AlphaCode reached competitive contest performance [2], and a recent survey [3] organizes the field. Multimodal models such as Gemini [7] extend these abilities through public APIs, the integration route used by SPARK.

AI pair programmers can speed up routine development work [8], but generated code is not automatically safe: a study of Copilot-generated programs found vulnerabilities in a significant fraction of security-relevant suggestions [9], and applications that feed external content to a language model can be manipulated through indirect prompt injection [10]. Generated code should therefore be treated as untrusted.

Untrusted workloads are commonly isolated with lightweight virtual machines such as Firecracker [11], while WebAssembly [12] and WebContainers [13] allow Node.js to run inside a web page. SPARK uses a browser sandbox for previews and a managed hosting API for publication, as the serverless literature recommends [4].

Operational transformation [14] and conflict-free replicated data types [15] guarantee convergence when several users edit shared state. SPARK currently uses a simpler broadcast model, which is easier to implement but weaker under concurrent edits.

Practical web technologies make the system feasible: React [16], Vite [17], React Flow [18], the Web Speech API [19], Supabase [20], n8n [21], and the Vercel API [22] supply the interface, build, canvas, speech, database, automation, and publication layers. The back end uses salted password hashing [23] and JSON Web Tokens [24] and is compared against the OWASP risk categories [25]. These components exist individually; what is missing is a single workflow that connects them for non-specialist users while keeping generated code under human review.

III. METHODOLOGY

A. Workspace Registration and Authentication

A user registers with an e-mail address, password, display name, and developer type. The Express back end rejects duplicate addresses, hashes the password with bcrypt [23], and returns a JSON Web Token that expires after one day [24]. Protected routes verify the bearer token.

Each workspace is identified by a UUID in the page address, so the link doubles as an invitation. The creator becomes the owner, and later visitors become members. Three roles exist (owner, admin, member), and only an owner may change a member's role or delete the workspace. Unauthenticated users receive a local guest identity.

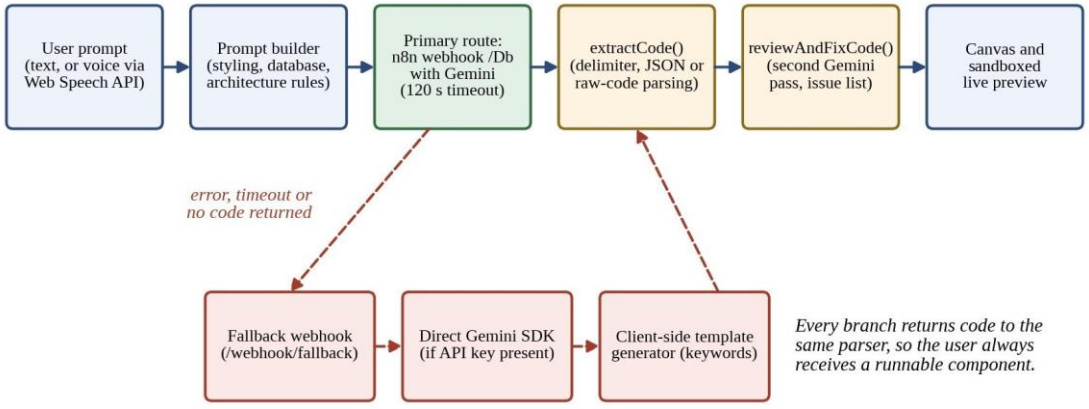
The server applies Helmet security headers, a global limit of 200 requests per 15 minutes, and a stricter limit of 10 per 15 minutes on authentication routes.

B. Intent-to-App Generation Pipeline

The user describes the application by typing or speaking; voice input uses the browser Web Speech API [19] and is converted to text. The user may also choose a project name and a styling preference (styled-components by default). A prompt builder then appends rules requiring a single-file component by default; multi-file output is introduced by a // FILE: Name.jsx delimiter.

The prompt is posted to an n8n webhook that calls Gemini, with a two-minute abort timeout. Because model services can fail, the orchestrator uses a four-stage fallback chain (Fig. 1): the primary webhook, a secondary webhook, a direct Gemini SDK call, and finally a client-side generator that selects a template by keyword. The user therefore always receives a runnable component, although the last stage is limited to built-in templates.

Fig. 1. Intent-to-app generation pipeline with the four-stage fallback chain.



C. Response Extraction and Code Normalization

The webhook may return plain code, fenced code, JSON, or service metadata, so a parser (extractCode) normalizes every response into a map from file name to source code. It tries, in order: (i) the multi-file delimiter; (ii) JSON, as a name-to-code map, an array of file objects, or a recursive search of likely keys; and (iii) raw code, after removing Markdown fences, accepted only if it contains a default export or a function with a return. If the response is only workflow metadata, the parser returns nothing, which triggers the next fallback stage.

D. Review, Auto-Fix, and Auto-Heal

Generated code is treated as untrusted. Before it is applied to the canvas, a review step sends the code and the original request to Gemini with a checklist covering request match, syntax errors, missing or disallowed imports, a default export, and use of the Supabase database instead of simulated storage. The model returns a JSON object with a status (ok or fixed), a list of issues, and the corrected code, which is shown in a review panel.

Two functions operate after generation. *Auto-heal* sends a runtime error log captured from the preview to the model and requests a corrected file. *Format-for-deploy* removes app.listen(), exports the Express application, and replaces hard-coded localhost addresses with relative paths.

E. Live Preview and In-Browser Execution

The preview component builds a complete HTML document inside an iframe. It loads React, the Supabase client, Tailwind, styled-components, and Emotion from public script hosts, strips import and export statements, and compiles JSX in the page with Babel. The iframe isolates the generated code from the SPARK interface. A StackBlitz WebContainer [13] can also boot an in-browser Node.js runtime. Runtime errors are captured as a crash log that can be passed to auto-heal or the watchdog workflow.

F. Visual Canvas and Real-Time Collaboration

The project is shown as a node-and-edge diagram using React Flow [18], in which users can add, connect, and delete components. A Supabase Realtime channel named after the workspace provides presence tracking and broadcast events that carry generated code, canvas updates, and role changes. Generated files are also logged to the database so that a later joiner can load them, and a Socket.IO room relays code changes as a second channel.

Conflict handling is deliberately simple: the most recent update wins. This suits a small team but is weaker than operational transformation [14] or CRDTs [15], which are listed as future work.

G. Database Provisioning for Generated Apps

When a workspace is opened, SPARK registers a database configuration in a shared Supabase table with the columns id, created_at, workspace_id, and a JSONB payload. The generation prompt instructs the model to keep all application data in the payload column, filtered by workspace. If the request mentions login or saving, a default table and Supabase Auth are used. Data is therefore separated by row scoping rather than by a separate database per app.

H. Deployment Pipeline

The deployment module publishes the generated files as a live site (Fig. 2). It first rejects an empty file set and builds a safe project name from the title and workspace identifier. Files in api/, backend/, or server/, and server entry files that contain Express patterns, are treated as back end; all others are front end. Dependencies are discovered from import and require statements.

The module then synthesizes a complete Vite project [17], including package.json, a vercel.json with single-page and API rewrites, and an api/index.js in which the server's listen call is neutralized for serverless execution. The project is submitted to the Vercel deployments endpoint with the production target [22], and its access protection is then patched so that the link opens publicly. The returned address is stored with a snapshot of the files. A *My Apps* panel pings each stored address every 120 seconds, marks it *online* or *down*, and can redeploy from the snapshot or delete the record.

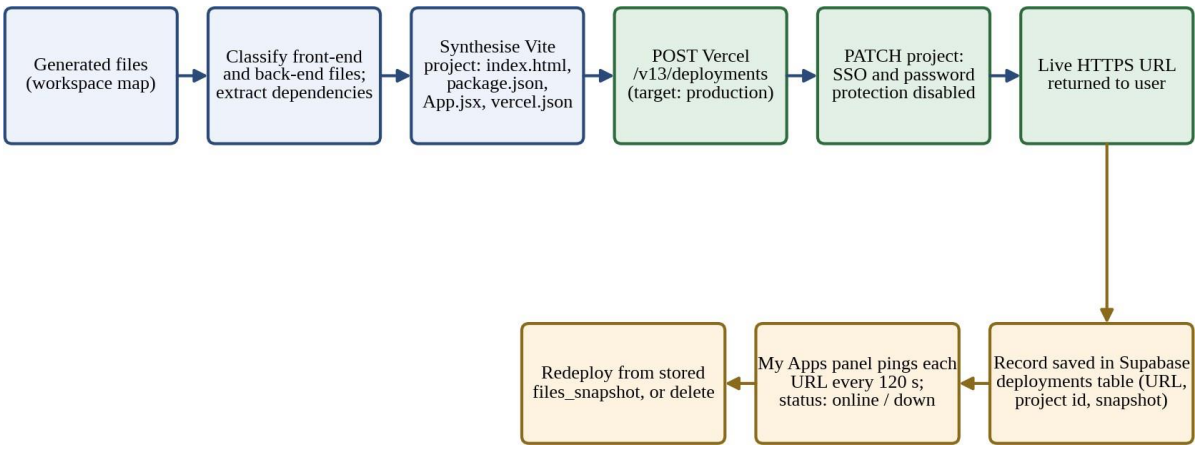


Fig. 2. One-click deployment pipeline and post-deployment monitoring.

I. Workflow Automation and Monitoring

An automation service posts JSON payloads to n8n webhooks [21]. Four workflows are defined: a *watchdog* that receives an error message and stack trace, requests a fix from the model, and returns a patch; a *deployment* workflow for continuous delivery to Vercel; an *error-alert* workflow that notifies a team channel if the watchdog fails; and a *version-control* workflow that receives a target state index from a time-travel slider. On the server, a job scheduled at 09:00 each day posts a digest of the previous day's activity to a configured webhook. In the submitted code, only the watchdog address

is a concrete endpoint; the other three are placeholders.

J. System Architecture

The system has five layers so that each can be tested and replaced independently (Fig. 3). The *presentation layer* is a React single-page application. The *AI orchestration layer* builds prompts, calls the webhooks and the Gemini SDK, and performs parsing, review, and repair. The *data layer* consists of MongoDB, accessed through an Express API [26], [27], for users, workspaces, and project storage, and of Supabase [20] for presence, deployment records, and the generated apps' data. The *automation layer* is the n8n server, and the *execution and hosting layer* consists of the iframe or WebContainer for previews and Vercel for public hosting.

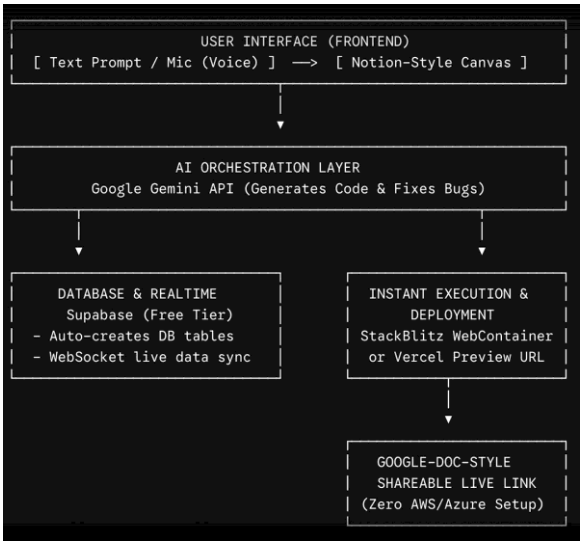


Fig. 3. High-level workflow: prompt or voice input, AI orchestration, database and real-time layer, instant execution, and a shareable live link.

K. Front-End and Back-End Implementation

JavaScript is used throughout: the front end uses React 19 and Vite 8, and the back end uses Node.js, Express 5, Mongoose, and Socket.IO. The main source files are summarized in Table I.

TABLE I. MAIN SOURCE MODULES OF THE SPARK PROTOTYPE

Module	File	Lines	Responsibility
Application shell	src/App.jsx	2,302	Layout, prompt panel, workspace state, commit, review, deploy, and assistant wiring
Dashboard	src/components/Dashboard.jsx	1,532	Home, workspaces, templates, deployments, members, settings, and upgrade views
AI orchestration	src/services/AIOrchestrator.js	1,420	Prompt building, fallback chain, response parsing, review, auto-heal, deploy formatting, chat
Canvas	src/components/CanvasEditor.jsx	494	React Flow nodes and edges, real-time canvas sync
Deployment	src/services/DeployService.js	446	Project synthesis, dependency extraction, Vercel API calls, deployment records
Live preview	src/components/FastPreviewIframe.jsx	318	Iframe document, JSX transformation, error capture
My Apps panel	src/components/MyAppsPanel.jsx	308	Deployment list, 120 s health check, redeploy, delete
Realtime service	src/services/SupabaseService.js	248	Presence, broadcast, identity, log persistence, projects table
Back-end server	server.js	424	Authentication, workspaces, roles, activity log, project storage, Socket.IO, cron digest
Automation	src/services/AutomationService.js	37	Webhook trigger for the four n8n workflows

L. User Interface

Fig. 4 shows the sign-in screen and Fig. 5 the home dashboard, which offers three entry points: *AI Builder*, *Templates*, and *Deploy*. A floating button opens the project assistant, which answers questions about the current workspace.

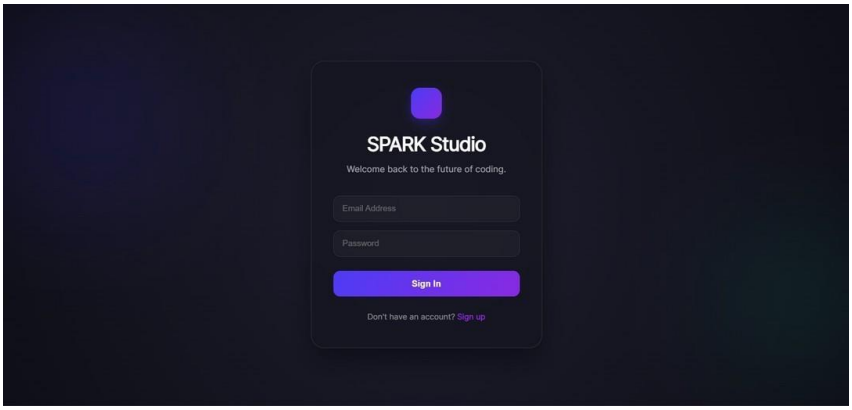


Fig. 4. SPARK Studio sign-in screen.

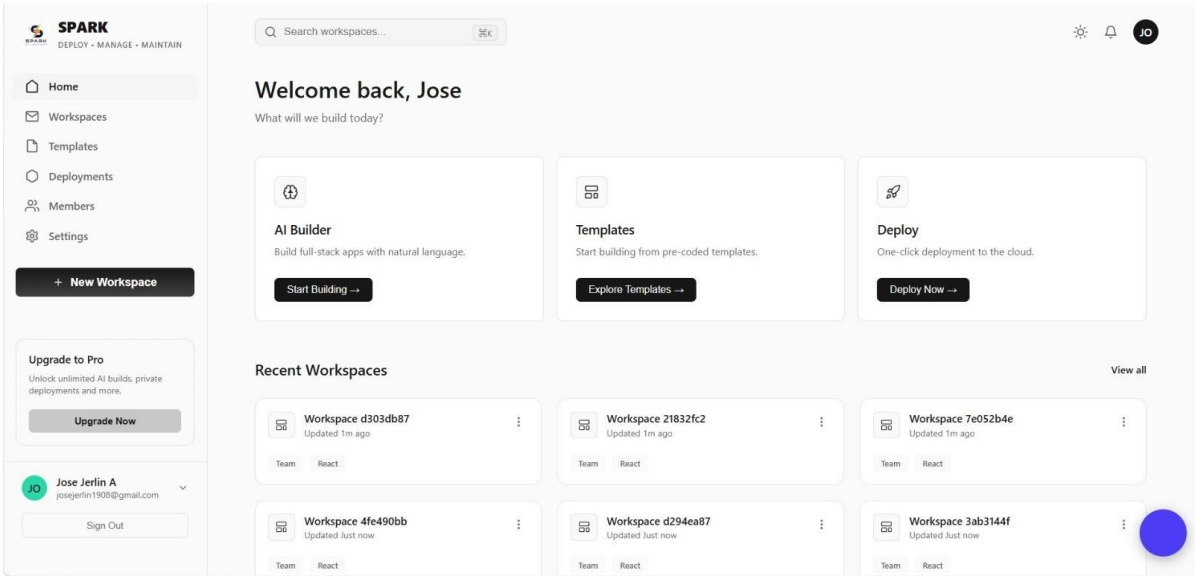


Fig. 5. SPARK home dashboard showing navigation, the three entry points (AI Builder, Templates, Deploy), and recent workspaces.

IV. TESTING AND RESULTS

A. Testing

Testing covers authentication, the parsing pipeline, deployment, and build quality (Table II). Scenarios marked *Executed* were run for this paper; those marked *Needs live services* depend on MongoDB, n8n, or Vercel access and were verified only by reading the code paths. The response parser was tested with six inputs (TC-05 to TC-10) and should return files for the first four and nothing for the last two, so that the fallback chain is triggered. The front end was installed from the lock file, compiled with a production build, and checked with a static analyzer.

TABLE II. TEST SCENARIOS AND EXPECTED RESPONSE

Test ID	Test scenario	Expected response	Status
TC-01	Register a new user with valid fields	Token and user profile returned	Needs live services
TC-02	Register with an existing e-mail	Error: e-mail already exists	Needs live services
TC-03	Log in with a wrong password	Error: invalid password	Needs live services
TC-04	Non-owner changes a member's role	Rejected with status 403	Needs live services
TC-05	Parse response with // FILE: delimiters (2 files)	Two files returned	Executed – passed
TC-06	Parse fenced single component	One file named after the component	Executed – passed
TC-07	Parse JSON map of file names to code	File returned with .jsx extension	Executed – passed
TC-08	Parse nested JSON (output.text)	One file returned	Executed – passed
TC-09	Parse workflow metadata only	No code; fallback triggered	Executed – passed
TC-10	Parse plain prose reply	No code; fallback triggered	Executed – passed
TC-11	Primary webhook exceeds 120 s	Fallback chain used	Needs live services
TC-12	Deploy with no files	Error: no files to deploy	Needs live services
TC-13	Production build of the front end	Build succeeds	Executed – passed
TC-14	Static analysis (oxlint)	No errors	Executed – passed (65 warnings)

B. Results

Table III reports only measured values; metrics that would need an evaluation set, live services, or participants are marked *Not measured* rather than estimated. The main bundle is a single 3.2 MB chunk, and the 65 analyzer warnings, mostly hook dependency lists, do not block compilation.

TABLE III. MEASURED RESULTS

Metric	Result
Dependencies installed (npm)	306 packages in 12 s
Modules transformed (Vite 8)	2,072
Production build time	3.28 s
Main JavaScript bundle	3,207.82 kB (829.60 kB gzip)
Style sheet	24.75 kB (4.73 kB gzip)
Archive library chunk	95.87 kB (28.44 kB gzip)
Source analyzed	35 files, 9,152 lines (src and server)
Static analysis (92 rules)	0 errors, 65 warnings
Parser tests passed	6 of 6
Generation success rate	Not measured
Generated-code correctness	Not measured
Generation latency	Not measured
Deployment success rate	Not measured
Concurrent-user capacity	Not measured
User study	Not performed

C. Comparison with the Existing Method

SPARK combines functions that are normally performed in separate tools (Table IV).

TABLE IV. COMPARISON OF THE CONVENTIONAL APPROACH AND SPARK

Feature	Conventional approach	SPARK
Creating the tool	Hand coding, or a local AI agent in an editor	Text or voice prompt in the browser, with fallback generation
Preview	Local development server	Sandboxed iframe or in-browser WebContainer
Sharing	Screen sharing or terminal hand-off; manual cloud setup	One-click deployment returning a public URL
Error handling	Manual debugging	Review pass, auto-heal, and watchdog webhook
Collaboration	File exchange or screen sharing	Presence, broadcast, and shared canvas
Database	Manual provisioning and schema design	Supabase configuration injected into the generation prompt
Access control	Separate cloud identity rules	Owner, admin, and member roles; token authentication
Monitoring	Separate monitoring tools	Health ping every 120 s and online/down status
Version recovery	Manual repository history	Time-travel slider triggering a version-control workflow

D. Advantages

1) One workflow covers creation, preview, and publication of small tools.
2) Typed or spoken requests lower the entry barrier for non-technical users.
3) The four-stage fallback chain keeps the platform working when an AI endpoint fails.

E. Limitations

1) Code quality depends on the model, and generation success has not been measured.
2) Model identifiers are hard-coded and inconsistent across modules (Gemini 2.5 Flash versus a 1.5 model).
3) Keys with the VITE_ prefix, including the deployment token, are embedded in client-side code, and default database credentials appear in the source; these calls should move to the server.
4) Project-storage routes require no authentication, cross-origin access is open, and deployed applications are public; single sign-on is not implemented.
5) Generated code has no formal isolation policy and is exposed to insecure code [9] and prompt injection [10].
6) Collaboration is last-write-wins, and three of the four automation endpoints are placeholders.
7) No user study, latency, cost, or load measurements were carried out.

V. CONCLUSION

SPARK is an academic, browser-based approach for turning a plain-language or spoken description into a shared, hosted web application, combining

authenticated workspaces, a fallback-protected generation pipeline, model-based review, a sandboxed preview, a collaborative canvas, and one-click deployment in a single workflow.

The prototype installs and builds reproducibly (306 packages, 2,072 modules, 3.28 s), and its parser handled all six test inputs correctly, including the two non-code responses that must trigger the fallback chain. The deployment pipeline, collaboration, and automation were reviewed at the code level but not exercised against live services. Generated code may be incorrect or insecure, secrets must be moved out of client-side code, and deployed links are public. The prototype is therefore intended only for academic demonstration.

Future work includes server-side credential proxying, single sign-on, isolation of generated code in micro-virtual machines [11], automated security scanning, conflict-free collaboration [15], working version-control and alert workflows, and a structured evaluation of generation success, latency, deployment reliability, and usability. With appropriate safeguards, such platforms can help non-specialist teams share the tools they build while keeping a human reviewer in control.

DECLARATIONS

Author contributions. All authors contributed to the conceptualization, software development, testing, and writing of this work. All authors read and approved the final manuscript.

Conflict of interest. The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

Data availability. No external datasets were used. All values reported in Tables II and III were generated by building and testing the prototype; the source code and test inputs are available from the corresponding author on reasonable request.

Ethics statement. This study did not involve human participants, personal data, or animals, and no user study was performed. Ethical approval was therefore not required.

Funding. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

ANNEXURE

This annexure presents the important source modules of the SPARK prototype, developed in JavaScript with React on the front end and Node.js with Express on the back end. Short excerpts are condensed for printing; credentials and keys have been omitted.

A. AI Orchestration Module (src/services/AIOrchestrator.js)

This module contains the prompt builders, the four-stage fallback chain, the response parser, and the review, auto-heal, deploy-formatting, and chat functions. The final stages of the fallback chain and the multi-file delimiter used by the parser are shown below.

```
// executeWithFallback (final stages)
const directText =
  await callDirectGemini(prompt, systemInstruction);
if (directText) return directText;
// last resort: client-side template generator
return generateSmartClientCode(prompt, 'App');
```

```
const fileRegex =
/\/\/\s*FILE:\s*([a-zA-Z0-9_.-]+)\n
([\s\S]*?)(?=\/\/\s*FILE:|$)/gi;
// (pattern split over lines for printing)
```

B. Deployment Module (src/services/DeployService.js)

This module classifies files, extracts dependencies, synthesizes the Vite project, calls the hosting API, and saves a deployment record. The submission call is shown below in condensed form.

```
const response = await fetch(
  `https://api.vercel.com/v13/deployments${teamParam}`,
  { method: 'POST',
  headers: { Authorization: `Bearer ${token}`,
    'Content-Type': 'application/json' },
  body: JSON.stringify({
    name: cleanName || 'spark-app', files: vercelFiles,
    projectSettings: { framework: 'vite',
      outputDirectory: 'dist' },
    target: 'production' }),
});
```

C. Back-End Server (server.js)

The server defines the user, workspace, activity-log, and project-storage schemas and the routes for registration, login, profile, workspaces, members,

commits, and project storage. It also starts the Socket.IO server and the daily digest job. The authentication middleware is shown below.

```
const auth = (req, res, next) => {
  const token = req.header('Authorization')
    ?.replace('Bearer ', '');
  if (!token)
    return res.status(401).json({ error: 'Access denied' });
  try {
    req.user = jwt.verify(token, JWT_SECRET); next();
  } catch (err) {
    res.status(400).json({ error: 'Invalid token' });
  }
};
```

D. Other Modules

src/App.jsx renders the layout, prompt panel, automation dashboard, tabbed views, and assistant, and coordinates database provisioning, generation, enhancement, review, auto-heal, commit, and deployment. SupabaseService.js manages identity, presence, broadcast, and file persistence; DatabaseService.js provisions the workspace database record; AutomationService.js triggers the n8n workflows; VercelService.js, NetlifyService.js, and WebContainerService.js provide alternative hosting calls and the in-browser runtime. Interface components include Dashboard.jsx, CanvasEditor.jsx, FastPreviewIframe.jsx, MyAppsPanel.jsx, VoiceAssistant.jsx (speech recognition), ProjectChatBot.jsx (workspace assistant), CodeReviewPanel.jsx (review results), and TimeTravelSlider.jsx (version rollback).

REFERENCES :

[1] M. Chen et al., “Evaluating large language models trained on code,” arXiv:2107.03374, 2021.

[2] Y. Li et al., “Competition-level code generation with AlphaCode,” Science, vol. 378, no. 6624, pp. 1092–1097, 2022.

[3] J. Jiang et al., “A survey on large language models for code generation,” arXiv:2406.00515, 2024.

[4] E. Jonas et al., “Cloud programming simplified: A Berkeley view on serverless computing,” arXiv:1902.03383, 2019.

[5] A. Vaswani et al., “Attention is all you need,” in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30, 2017.

[6] T. B. Brown et al., “Language models are few-shot learners,” in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 1877–1901.

[7] Gemini Team, Google, “Gemini: A family of highly capable multimodal models,” arXiv:2312.11805, 2023.

[8] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, “The impact of AI on developer productivity: Evidence from GitHub Copilot,” arXiv:2302.06590, 2023.

[9] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the keyboard? Assessing the security of GitHub Copilot’s code contributions,” in Proc. IEEE Symp. Security and Privacy (SP), 2022, pp. 754–768.

[10] K. Greshake et al., “Not what you’ve signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection,” arXiv:2302.12173, 2023.

[11] A. Agache et al., “Firecracker: Lightweight virtualization for serverless applications,” in Proc. USENIX Symp. Networked Syst. Design Implement. (NSDI), 2020, pp. 419–434.

[12] A. Haas et al., “Bringing the web up to speed with WebAssembly,” in Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI), 2017, pp. 185–200.

[13] StackBlitz, “WebContainers API documentation,” webcontainers.io.

[14] C. Sun and C. Ellis, “Operational transformation in real-time group editors: Issues, algorithms, and achievements,” in Proc. ACM Conf. Comput. Supported Cooperative Work (CSCW), 1998, pp. 59–68.

[15] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free replicated data types,” in Proc. Int. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS), LNCS 6976, 2011, pp. 386–400.

[16] Meta Open Source, “React documentation,” react.dev.

[17] Vite contributors, “Vite documentation: Next generation frontend tooling,” vite.dev.

[18] xyflow, “React Flow documentation,” reactflow.dev.

[19] W3C Community Group, “Web Speech API specification,” webaudio.github.io/web-speech-api.

[20] Supabase Inc., “Supabase documentation: Database, Auth and Realtime,” supabase.com/docs.

[21] n8n GmbH, “n8n workflow automation documentation,” docs.n8n.io.

[22] Vercel Inc., “Vercel REST API: Deployments,” vercel.com/docs/rest-api.

[23] N. Provos and D. Mazières, “A future-adaptable password scheme,” in Proc. USENIX Annu. Tech. Conf., FREENIX Track, 1999, pp. 81–91.

[24] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” IETF RFC 7519, May 2015.

[25] OWASP Foundation, “OWASP Top 10: 2021,” owasp.org/Top10.

[26] OpenJS Foundation, “Express and Socket.IO documentation,” expressjs.com; socket.io/docs.

[27] Mongoose contributors, “Mongoose ODM documentation,” mongoosejs.com.