Project Proposal

1) Executive Summary

Build a multi-tenant SaaS that lets authenticated users generate full-fledged research papers from a topic prompt using a LangGraph agent with specialized tools. The product offers 3 free generations per user, then paywalled plans (Basic / Standard / Royal). The system is split into three independently deployed services: **Frontend (React)**, **Backend API (Node/Express + MongoDB)**, and a **LangGraph Agent Service**. Chat sessions are persisted (like ChatGPT) and responses are streamed token-by-token.

---

2) Goals & Success Criteria

- **Primary Goal:** Provide high-quality, citation-aware research papers via an explainable, tool-using agent.
- **Success Criteria:**
  - ≥ 95% uptime for free tiers of chosen platforms.
  - < 2s first-token latency on typical requests (under free hosting constraints).
  - ≥ 40% conversion from free to paid after 3 generations (target to optimize post-launch).
  - Zero PII leaks; all secrets stored in environment variables; JWT-based auth.

---

3) Core Features (MVP)

1. **Auth & Accounts**
   - Email/password sign-up & login (JWT + refresh tokens, HTTP-only cookies).
   - OAuth (Google) optional stretch.

2. **Usage & Paywall**
    - 3 free topic generations per user.
    - When limit reached, auto-redirect to Pricing.
3. **Plans**
    - **Basic**: N papers/month, standard model, queued processing.
    - **Standard**: Higher limits, citation export (BibTeX), PDF download.
    - **Royal**: Priority, long-context, advanced tools, team seats.
4. **Pages**
    - Home, Pricing, About, Contact, Login, Signup, **Chat/Workspace**.
5. **Chat + Streaming**
    - Realtime token streaming (SSE or WebSocket) with typing indicator.
    - Sidebar session list, rename/delete sessions, restore history.
6. **Research Paper Generator**
    - Agent writes a structured paper (Abstract, Intro, Related Work, Methods, Results, Discussion, Conclusion, References).
    - Include citations (inline numeric or author-year) and reference list.
7. **Exports**
    - Markdown & PDF (server-side renderer). BibTeX/EndNote for references.

---

4) Non-Functional Requirements

- **Security:** bcrypt hashing, JWT rotation, rate limiting, input validation, CORS policy, secret management.
- **Scalability:** Stateless services, horizontal scaling, queue for long jobs.
- **Observability:** Centralized logs, request tracing IDs, minimal metrics (requests, latency, errors).
- **Data Protection:** GDPR-style deletion endpoints, least-privilege DB roles.

---

5) System Architecture

Three services, independently deployed:

1. **Frontend (React + Tailwind + Router)**
    - Responsible for UI, auth flow, session management, chat UI, pricing page, account page.
    - Talks to Backend API for auth, usage, sessions; subscribes to SSE/WebSocket from Agent via Backend (proxied) or direct if allowed by CORS.
2. **Backend API (Node/Express + MongoDB Atlas)**
    - Auth, users, billing webhooks, usage limits, session CRUD.
    - Initiates/mediates agent runs; streams tokens to client.
    - Generates PDFs; stores message deltas and final artifacts.
3. **LangGraph Agent Service (Python or Node)**
    - Orchestrates tools: web search, PDF fetch, citation extraction, summarization, outline → drafting → polishing.
    - Stateless; returns streamed tokens + structured metadata.

Data Flow (happy path): Client → Backend `/chat/:sessionId/stream` → Backend forwards to Agent `/generate` (SSE/WebSocket) → streams tokens → Backend relays to Client; Backend persists messages and usage counters.

---

6) Free Deployment Strategy (Zero-cost friendly)

- **Frontend:** Vercel/Netlify (static React, free SSL, CI from GitHub).
- **Backend API:** Render (Free Web Service) or Railway free plan.
- **LangGraph Agent:**
    - **Option A (Python FastAPI + SSE)** on **Render Free**.
    - **Option B (Gradio/FastAPI)** on **Hugging Face Spaces (public)**.

    - **Option C (Docker)** on **Railway Free**.

- **Database:** MongoDB Atlas Free (Shared M0 cluster).
- **Queue (optional):** Upstash Redis Free (for job status & rate limit tokens).
- **Monitoring:** Logtail/Better Stack free tier; or simple JSON logs + Render/Railway dashboards.

Notes: Free tiers may sleep on idle; implement reconnectable SSE and request retries. Provide user feedback when cold-starting.

---

7) Tech Stack

- **Frontend:** React, Vite, Tailwind, React Router, Zustand/Redux, React Query; Markdown renderer; PDF viewer.
- **Backend:** Node.js 20, Express, Zod (validation), jsonwebtoken, bcrypt, Stripe SDK, Mongoose, BullMQ (optional), SSE.
- **Agent:** LangGraph + LangChain; FastAPI/Express; HTTP SSE for streaming; tools (Tavily/SerpAPI, arXiv/Crossref, PDF parsing with PyPDF/Unstructured); citation formatter.
- **Infra:** Docker (for Agent), GitHub Actions CI, environment variables via platform secrets.

---

8) Data Model (MongoDB)

users

- _id, email (unique), passwordHash, plan: 'free'|'basic'|'standard'|'royal', searchCount, createdAt, updatedAt, stripeCustomerId?, seats? (royal), profile?

chat_sessions

- _id, userId (index), title, createdAt, updatedAt, model, tokensUsed, status: 'active'|'archived'

messages

- _id, sessionId (index), role: 'user'|'assistant'|'system', content (markdown), deltas?[], citations?[], createdAt

generations

- _id, sessionId, userId, outline, sources[], paperMarkdown, paperPdfUrl?, bibtex?, qualityScores, createdAt

subscriptions

- _id, userId, plan, status, startedAt, renewsAt, stripeSubscriptionId, limits {monthlyPapers, tokenCap}

---

9) API Design (selected endpoints)

Auth

- POST `/auth/signup` {email, password}
- POST `/auth/login` {email, password}
- POST `/auth/logout`
- GET `/auth/me` -> user profile + plan + usage

Usage & Paywall

- GET `/usage` -> {searchCount, limit}
- POST `/usage/increment` (middleware-guarded per generation)

Sessions

- POST `/sessions` -> create session {title?}
- GET `/sessions` -> list
- GET `/sessions/:id` -> details + messages
- DELETE `/sessions/:id`
- PATCH `/sessions/:id` {title}

Chat/Generation

- POST `/chat/:sessionId/message` {content}
- GET `/chat/:sessionId/stream` (SSE) -> token events, citations, finish event

Billing (Stripe)

- POST `/billing/checkout` {plan}
- POST `/billing/portal`
- POST `/billing/webhook` (Stripe signature verified)

Artifacts

- GET `/generations/:id.pdf` (signed URL)
- GET `/generations/:id.bib`

---

10) LangGraph Agent Design

Graph Nodes (example):

1. **Intent & Scope** → normalize prompt, detect domain/keywords.
2. **Search** → web + academic (Tavily/SerpAPI, arXiv, Crossref, Semantic Scholar API).
3. **Source Fetch** → fetch PDFs/HTML; extract text & metadata.
4. **Evidence Grid** → chunk + rank passages per section.
5. **Outline Builder** → section headings with claims & supporting citations.
6. **Draft Writer** → section-wise drafting with inline citations.
7. **Citation Compiler** → format references (APA/IEEE/MLA selectable).
8. **Polisher** → style, coherence, dedup citations, abstract.
9. **Exporter** → emit Markdown; optional PDF & BibTeX.

Tools: search_api, fetch_pdf, extract_text, cite_format, summarize_chunk, fact_check (heuristic), anti-plagiarism (similarity check against fetched sources).

Streaming: emit events `{type: 'token'|'section'|'citation'|'meta', data: ...}` so the UI can display live progress and attachment chips for sources.

---

11) Frontend UX Notes

- **Pages:** Home, Pricing, About, Contact, Login, Signup, Chat.
- **Chat Workspace:**
  - Editor area with live stream, toolbar: Export PDF/MD/Bib, Copy.
  - Sidebar: sessions + search; hover to rename/delete.
  - Top bar: plan & remaining generations; Upgrade button.
- **Empty States:** helpful demos & templates (e.g., "Medical literature review", "Tech survey").
- **Accessibility:** Keyboard-first, ARIA labels, semantic HTML.

---

12) Security & Compliance

- Passwords: bcrypt(12+). JWT access short-lived; refresh via HTTP-only cookie.
- CSRF: SameSite=strict on cookies; CSRF token for state-changing forms if needed.
- Validation: Zod schemas for all inputs. Size limits for uploads.
- Ratelimits: per-IP + per-user sliding window via Redis (Upstash) to protect free tier.
- Secrets: platform env vars; no secrets in repo.
- Content Safety: filter disallowed topics (e.g., malware generation) if required; disclaimer on academic integrity.

---

13) Observability & Ops

- Structured logs (JSON) with requestId and userId.
- Metrics (lightweight): req/sec, p95 latency, token throughput, error rate.
- Alerts: email on error spikes; dead-letter queue for failed generations.

- Backups: Mongo Atlas snapshots (free tier nightly is limited; manual export cron optional).

---

14) Payment & Entitlements

- Stripe Checkout for plan selection; Webhooks to activate entitlements.
- Entitlement checks in middleware: `requirePlan('standard')` etc.
- Proration and cancellation handling; grace period setting in config.

---

15) Rollout Plan (Phased)

- **Phase 1:** Foundations — auth, sessions, SSE streaming, basic agent loop, 3-free limit.
- **Phase 2:** Citations & Exports — reference compiler, PDF/BibTeX export.
- **Phase 3:** Billing — Stripe checkout + webhooks, plan enforcement.
- **Phase 4:** Quality — source de-duplication, fact-checking heuristics, UI polish.
- **Phase 5:** Growth — analytics, templates, referrals, team seats (Royal).

(Phases are ordered milestones without time estimates.)

---

16) Risks & Mitigations

- **Cold starts on free hosts** → reconnectable SSE, user notice, warm-up pings.
- **Rate limits on external search/APIs** → backoff + caching; fallbacks.
- **Long contexts/token costs** → summarization chain + sectioned streaming.
- **Academic integrity concerns** → clear ToS; include citation requirements and originality notes.

---

17) Acceptance Criteria (MVP)

- Auth works with email/password; sessions persist across devices.
- Exactly 3 free generations enforced; paywall trigger renders Pricing.
- Streaming works reliably; users can export PDF/MD; chats saved/restored.
- Agent outputs structured paper with at least 5 citations when sources exist.
- Deployments run on free tiers; environment configs documented.

---

18) Deliverables

- Source code for **Frontend**, **Backend**, **Agent** (three repos).
- Infrastructure files: Dockerfile (Agent), Render/Railway config, Vercel config.
- ENV templates and secrets guide.
- API reference (OpenAPI/Swagger JSON).
- Minimal runbook for incident response.

---

19) Repository Structure (illustrative)

frontend/ (React)

- src/
  - pages/ (Home, Pricing, About, Contact, Login, Signup, Chat)
  - components/ (Chat, Sidebar, Message, PlanBadge, Loader)
  - hooks/, store/, api/
  - styles/
- vite.config.ts, package.json

backend/ (Node/Express)

- src/
  - index.ts (server)
  - routes/ (auth, sessions, chat, billing, usage)

    - middleware/ (authJwt, rateLimit, requirePlan)
    - models/ (User, Session, Message, Generation, Subscription)
    - services/ (pdf, stripe, sseRelay)
    - utils/ (zodSchemas, logger)
- dockerfile (optional), package.json

agent/ (LangGraph)

- app/
    - main.py (FastAPI + SSE)
    - graph.py (nodes, edges)
    - tools/ (search.py, fetch_pdf.py, citations.py)
    - writers/ (outline.py, draft.py, polish.py)
- requirements.txt, Dockerfile

---

20) Configuration & ENV

- `MONGODB_URI` (Atlas)
- `JWT_SECRET`, `JWT_REFRESH_SECRET`
- `STRIPE_SECRET`, `STRIPE_WEBHOOK_SECRET`
- `AGENT_BASE_URL`
- `TAVILY_API_KEY` / `SERPAPI_KEY`, `CROSSREF_MAILTO`
- `REDIS_URL` (optional)

---

21) Testing Strategy

- Unit: Zod schemas, utils, citation formatter.
- Integration: Auth, sessions, streaming endpoint (SSE tests).
- E2E: Cypress for signup → generate → paywall → upgrade → generate.

- Load: k6/Gatling for streaming concurrency on free hosts.

---

22) Next Steps (What will be built first)

1. Scaffold three repos with minimal CI.
2. Implement auth + sessions + SSE echo stream.
3. Implement LangGraph minimal agent (outline + one source + draft).
4. Enforce 3-free limit & Pricing gate.
5. Add citation tools + exports; then Stripe.

---

23) Appendix — Example Contracts

SSE Event shape

```
{ "type": "token", "data": "…" }
{ "type": "citation", "data": { "title": "…", "url": "…", "id
{ "type": "section", "data": { "name": "Methods" } }
{ "type": "done", "data": { "durationMs": 12345 } }
```

Message schema (Mongo)

```
{
  _id: ObjectId,
  sessionId: ObjectId,
  role: 'user' | 'assistant' | 'system',
  content: string, // markdown
  citations: [{ id: string, url: string, title: string }],
  createdAt: Date
}
```

Agent request

{ "topic": "AI in Healthcare", "style": "IEEE", "sections": [

Agent response (final)

{ "markdown": "# Title...", "citations": [{"id":"ref_1","titl