**Build a full-featured Academic Text Editor with Grammar Checking & Plagiarism Detection (React + FastAPI)**

You are a senior full-stack engineer building a production-ready web app: **“ScholarStudio”** — a rich academic text editor with citation insertion, tables, graphs, equations, exports (.docx, .pdf, .tex, .md), and integrated plagiarism/AI-text detection using **ZeroGPT** and **CopyLeaks**. Also include an **open-source grammar checker** accessible via a FastAPI backend for easy integration (suggestion: LanguageTool or grammalecte for multilingual support). Deliver clean, well-documented code, tests, Docker setup, and deployment instructions. Follow best practices for security, privacy, and modularity.

**1) High-level goals**

* Elegant, accessible WYSIWYG editor supporting long form academic writing.
* Citation manager with import (BibTeX/CSL) and in-editor insertion (author–year & numbered styles).
* Native support for tables, plots (interactive), equations (LaTeX/KaTeX), and images.
* Export to .docx, .pdf, .tex, and .md preserving citations, equations, and figures.
* Integrated plagiarism/AI detection pipeline: requests go to ZeroGPT and CopyLeaks; results shown in UI with highlighted passages and summary score.
* Grammar checker: open-source service (LanguageTool recommended) wrapped by FastAPI for centralized control and easy swapping.
* Clean REST API and WebSocket support for realtime features (auto-save / export status).
* Modular, testable, and dockerized; sample deployment on Docker Compose.

**2) Tech stack (recommended)**

* Frontend: **React** (Vite), TypeScript, Tailwind CSS, TipTap (or ProseMirror) as rich text core, KaTeX/MathJax for rendering, React-Plotly / Chart.js for graphs, react-table or Tiptap table extension.
* Backend: **FastAPI** (Python, async), Uvicorn/Gunicorn, Pydantic models.
* Exports & document processing: python-docx (docx), pandoc (md ↔ tex ↔ pdf via wkhtmltopdf or LaTeX), WeasyPrint or wkhtmltopdf for PDF generation, latex toolchain for .tex → .pdf optionally.
* Grammar: **LanguageTool** server wrapped by FastAPI (or integrate its HTTP API). Provide fallback to local grammar engines.
* Plagiarism/AI-detection: integrate **CopyLeaks** (official API) and **ZeroGPT** (API), with orchestration & caching.
* Data store: SQLite (dev) and PostgreSQL (prod) via SQLAlchemy or Tortoise ORM for documents, users, citations, and scan results.
* Authentication: JWT + OAuth2 (optional) for API auth.
* Containerization: Dockerfiles for frontend and backend; Docker Compose for local dev.
* Tests: Pytest (backend), Jest/React Testing Library (frontend).
* CI: GitHub Actions for lint, tests, build, and Docker image push.

**3) Functional requirements (user stories)**

* As an author, I can write and format text (bold, italic, headers, lists).
* I can insert citations from an imported BibTeX file; citations appear inline and build a reference list.
* I can create tables, add/remove rows/cols, and export them correctly.
* I can insert equations using LaTeX syntax (rendered in-editor).
* I can upload or create plots from CSV or JSON data (plot builder modal); plots are interactive in the editor and export as images.
* I can run grammar checks and see inline suggestions; accept/reject suggestions.
* I can run plagiarism and AI-authorship checks via ZeroGPT & CopyLeaks; results show per-paragraph flags and overall scores.
* I can export the document to .docx, .pdf, .tex, and .md with citations resolved to reference section in desired style (e.g., IEEE, APA).
* I can save/load documents; autosave every N seconds; version history.
* Admins can configure API keys (ZeroGPT, CopyLeaks, LanguageTool) via env vars or admin UI.

**4) UX / UI details (concrete)**

* Use Tiptap as the editor core (content stored as HTML/JSON). Provide toolbar with:
  + Text formatting, lists, headings.
  + Equation button: opens modal with LaTeX input, preview via KaTeX.
  + Citation button: opens modal search (autocomplete from imported BibTeX) and inserts inline citation token.
  + Table button: create table, right-click cell menu.
  + Chart button: “Create Chart” modal: upload CSV or paste JSON, pick plot type, preview chart, insert.
  + Plagiarism/AI-check button and Grammar-check button (both can also be run automatically on demand).
  + Export button dropdown for .docx, .pdf, .tex, .md.
* Editor shows inline highlights for grammar suggestions and plagiarism/AI flags. Hover reveals details, click opens side panel.
* Sidebar: Reference manager (list of imported citations, add/edit), Document outline, Version history.
* Mobile responsive but desktop-first.

**5) Backend API design (suggested endpoints)**

* POST /api/v1/auth/login — returns JWT
* POST /api/v1/documents — create new document
* GET /api/v1/documents/{id} — fetch document JSON/HTML
* PUT /api/v1/documents/{id} — save document
* POST /api/v1/documents/{id}/export — body: {format: "pdf"|"docx"|"tex"|"md", citation\_style: "IEEE"}, returns signed URL or stream
* POST /api/v1/grammar/check — body: {text: "...", language: "en"} → calls LanguageTool; returns suggestions
* POST /api/v1/plagiarism/scan — body: {text: "...", providers: ["zerogpt","copyleaks"]} → orchestrates scans, returns aggregated report and per-paragraph flags
* GET /api/v1/plagiarism/result/{scan\_id} — fetch results
* POST /api/v1/citations/import — upload BibTeX; parse and save entries
* GET /api/v1/citations — list citations
* POST /api/v1/plot/render — body: {data: CSV|JSON, type: "line"|"bar"|"scatter", options} → returns image/svg or embeddable JSON for plotly
* WebSocket /ws/docs/{id} — autosave & real-time collaboration hooks (optional)

**6) Plagiarism / AI detection orchestration**

* Implement an async worker queue (Redis + RQ or Celery) to handle scans (ZeroGPT + CopyLeaks) to avoid blocking HTTP requests.
* Normalize inputs: split document into paragraphs/segments, send segment-wise requests, cache results.
* Define aggregated scoring logic: e.g., weighted average of provider scores; show provider-by-provider breakdown.
* Respect rate limits: exponential backoff and retry, queueing, and batching.
* Securely store scan metadata and not store full document text in logs. If storing text is required by API, encrypt at rest.

**7) Grammar checker (open-source) integration**

* Recommend **LanguageTool** (server mode) for robust grammar checks and multi-language support.
* FastAPI wrapper:
  + POST /api/v1/grammar/check calls local or remote LanguageTool server, transforms results into editor-friendly format, returns suggestions with offsets and replacement text.
* Provide fallback: if LanguageTool unavailable, return helpful error message and allow offline client-side checks (lightweight rules).

**8) Export pipeline specifics**

* Store document in canonical internal representation (Tiptap JSON or HTML + metadata for citations).
* Export flow:
  + Convert editor content → Markdown using unified / Turndown or Tiptap serializer.
  + Use pandoc or python-docx for advanced conversions:
    - .md — direct from serializer.
    - .tex — via pandoc from md with citation processing (pandoc-citeproc or citeproc).
    - .docx — pandoc or python-docx template to embed citations as footnotes/endnotes or as in-text references.
    - .pdf — from .tex (LaTeX) or HTML → PDF (WeasyPrint/wkhtmltopdf). Prefer LaTeX for best equation rendering.
  + Resolve citations: insert reference list using CSL styles — use citeproc-py or pandoc’s citeproc.
  + Include figures/plots as embedded PNG/SVG with captions and labels.
* Offer preview endpoint that streams a PDF for quick inspection before final export.

**9) Security & privacy**

* Never log API keys or PII. Use env vars and secrets manager.
* Use HTTPS everywhere; helmet-like headers on frontend.
* API auth: JWT with short expiry + refresh tokens, secure cookie for browser flows.
* Data residency: allow opt-out of storing document text for plagiarism providers; allow user to delete scans/documents.
* Rate limiting & per-user quotas for expensive operations (scans, exports).

**10) Tests & acceptance criteria**

* Unit tests covering backend endpoints, citation parsing, export pipeline (sample documents).
* End-to-end tests: create doc → insert citation → insert equation → run grammar → run scan → export PDF → check presence of citation and equation.
* Performance test: export large document (~100 pages) within acceptable time limit (e.g., PDF in <60s on prod hardware).
* Security tests: ensure no API key leakage and proper auth on endpoints.
* UX acceptance: editor renders equations and plots, citation insertion works, and plagiarism UI highlights passages correctly.

**11) Developer deliverables**

* Frontend repo: React + TypeScript, components for editor, citation modal, grammar UI, plagiarism UI, plot builder.
* Backend repo: FastAPI with routes, workers, DB models, export utilities, and wrappers for ZeroGPT + CopyLeaks + LanguageTool.
* Docker Compose files for dev and prod.
* README with setup, env vars, sample API key usage, and example requests.
* Postman collection or API spec (OpenAPI/Swagger).
* Tests & CI pipeline config.
* Short demo: sample document showing all features.

**12) Sample API payloads & examples (toy)**

* Grammar check:

POST /api/v1/grammar/check

Authorization: Bearer <JWT>

Content-Type: application/json

{

"text": "This is a sentence with bad grammar.",

"language": "en-US"

}

* Plagiarism scan:

POST /api/v1/plagiarism/scan

Authorization: Bearer <JWT>

Content-Type: application/json

{

"document\_id": "abc123",

"text": "Full document or paragraph array...",

"providers":["zerogpt","copyleaks"]

}

* Export:

POST /api/v1/documents/abc123/export

Authorization: Bearer <JWT>

Content-Type: application/json

{ "format":"pdf","citation\_style":"ieee" }

**13) Non-functional requirements & trade-offs**

* Cost: scanning every save is expensive. Default: on-demand scans with optional scheduled scans.
* Latency: grammar checks should be near-real time; plagiarism scans are async.
* Scalability: use Celery/Redis & autoscaling for workers.
* Open-source grammar keeps stack portable; allow swapping to commercial grammar later.

**14) Implementation notes & suggestions**

* Use Tiptap ProseMirror for table and equation extensions; it stores document as JSON and is easy to serialize.
* For citation handling, store each citation as a unique token (e.g., [[cite:citekey]]) then resolve during export.
* For equations, store LaTeX inside nodes and render client-side with KaTeX; for PDF export prefer LaTeX path to retain fidelity.
* For plots, create embeddable HTML + JSON (plotly) but export static PNG for docx/pdf pipelines.
* Provide a lightweight client-side grammar highlight fallback (simple rules via regex) if server unavailable.

**Final lines for the contractor/AI**

Build a minimal viable product (MVP) that demonstrates all major flows (write → cite → grammar check → plagiarism scan → export) within 2–3 sprint cycles. Deliver frontend and backend repos, Docker Compose for local dev, an OpenAPI spec, and a small demo document showing correct export with citations and equations. Use LanguageTool for grammar, integrate ZeroGPT & CopyLeaks for scanning, and document where API keys should be placed. Provide tests and CI.