**Architecture/Design (огляд і висновок)**

**Висновок:** показана структура проєкту та твоє бачення повністю відповідають цілі “інноваційна супер-система” (вище Luminance) — за умови, що ми **заморозимо SSOT-контракти**, добудуємо **детермінований rule-engine з 25–32 кроками** і з’єднаємо це зі **стабільними API** до панелі Word. Поточні файли фронту (word\_addin\_dev/taskpane.html, taskpane.bundle.js) вже містять правильні гачки: Office.js-брідж, cache-buster, кнопки Analyze/Draft/Apply, консоль/бейджі, селектори для Suggested edits — це хороший “кістяк” для UX і діагностики.   
Дерево каталогу підтверджує наявність “двигуна” (engine/), API (api/), GPT-модулів (з мок-проксі), legal\_rules та Word-addin — отже база достатня для поетапної збірки без “латок”.

**Головні прогалини:**

1. відсутність остаточної **SSOT-схеми** (поля/типи AnalysisOutput, Finding, Recommendation, Citation, Span, CrossRef) — фронт уже очікує score/risk/status/severity/findings/recommendations (див. бейджі й рендер списків). Потрібна жорстка узгодженість бек-відповідей з цим форматом.
2. **/api**: доповнити стабільні маршрути (/health, /api/analyze, /api/gpt/draft, /api/suggest, /api/qa-recheck) та гарантувати “завжди валідний JSON навіть на порожньому тексті” через compat-шар.
3. **Правила 25–32 кроків** і **cross-checks** — має бути детермінований скоринг + обов’язкові посилання на UK/UA/EU норми.
4. **Зв’язування з Word**: current UI дозволяє “Use selection/Use whole doc/Insert result”, але потрібні стабільні сценарії “Analyze → Annotate (коментарі лише на проблемах) → Draft → Apply (tracked) → QA-recheck”.

**Plan (порядок робіт без коду; файл→функція→вихід/вхід→перевірка)**

Нижче — **послідовний чек-лист**. Ідемо зверху вниз, файл за файлом. Де файл уже існує — уточнюю, що додати; де нема — що створити. Формат: **[Файл/папка] → призначення → що додати/перевірити → вхід/вихід → як тестувати**.

**0) Freeze SSOT (єдиний контракт даних)**

* **[core/schemas.py]** → SSOT-типи: AnalysisInput, AnalysisOutput, Finding, Recommendation, Citation, Span, CrossRef, RiskLevel, Status.  
  **Що додати:** повний опис полів, обов’язкові/опційні, діапазони (score: 0..100), інваріанти (critical ⇒ FAIL).  
  **Вхід/вихід:** використовується в усьому бекенді.  
  **Тест:** валідатор схем; “порожній текст” теж формує коректний AnalysisOutput.
* **[engine/pipeline\_compat.py]** (є) → **нормалізатор** виходів до SSOT (дефолти, обрізання довгих рядків/колекцій).  
  **Тест:** навіть якщо rule повернув мінімум — фронт не ламається, бейджі показують “—”. (UI це вже очікує).

**1) API (FastAPI)**

* **[api/app.py]** (є) → оголошення стабільних маршрутів:  
  **GET /health**, **POST /api/analyze**, **POST /api/gpt/draft**, **POST /api/suggest**, **POST /api/qa-recheck**.  
  **Що додати:** кореляційний заголовок x-cid, CORS, таймаути, ліміти, структурні логи.  
  **Вхід/вихід:** чіткі DTO (Pydantic) — використовують SSOT.  
  **Тест:** Invoke-WebRequest/Postman/панель “Test” показує статус 200 і валідний JSON (панель уже має кнопки Save/Test бекенду).
* **[api/orchestrator.py]** (є) → маршрутизація до engine/pipeline.py, gpt/\*, “QA-recheck” (диф метрик).  
  **Що додати:** graceful-errors → через compat гарантуємо валідний формат.

**2) Intake/Indexing (розбір документа)**

* **[engine/intake.py]** (є) → нормалізація, розбиття на клауза/пункти, побудова Span (offset/len) для кожного блоку.  
  **Що додати:** стабільні правила сегментації (заголовки, маркери списків), карта DocIndex для навігації з панелі.  
  **Тест:** однаковий уривок → однаковий Span у відповідях.

**3) Rule Engine + Cross-checks**

* **[legal\_rules/]** (є) → набір правил by clause, **25–32 кроків** кожне. Папка і registry.py присутні.   
  **Що додати:**
  + стандарт rule(text, ctx) -> list[Finding] + optional proposed\_text + citations.
  + cross\_checks.py — карта залежностей (Termination↔LoL, Jurisdiction↔Governing law, NDA↔DP тощо).
  + severity/risk/score — детермінована формула.  
    **Вхід/вихід:** текст клауза + ctx (сусідні клауза/дата/терміни) → Findings/Recommendations/Citations/Spans.  
    **Тест:** змінюємо умову (напр. прибираємо notice-period) → очікуваний Finding з’являється/зникає.
* **[engine/executor.py]** (є) → виклик правил по кожному блоку + агрегування cross-checks, формування AnalysisOutput.  
  **Що додати:** логи trace/diagnostics, інваріанти для status/risk/score.

**4) Orchestration (документ↔клауза)**

* **[engine/pipeline.py]** (є) → analyze\_document(text), збір усіх AnalysisOutput + карта посилань між клаузами; synthesize\_draft(analysis, mode).  
  **Що додати:** три режими драфту: friendly/standard/strict, підказки citations\_hint.  
  **Тест:** для одного блока всі 3 режими дають різні пропозиції (за суворістю/розкриттям ризиків).

**5) Draft/Proxy (LLM-off сумісність)**

* **[gpt/gpt\_proxy\_api.py, gpt\_prompt\_builder.py, gpt\_orchestrator.py]** (є) → зберігаємо як **мок**;  
  **Новий (опційно):** gpt/gpt\_draft.py — коли LLM увімкнено, але **з guardrails**: не перекривати rule-вердикти, без “вигаданих” норм.  
  **Тест:** /api/gpt/draft повертає draft\_text навіть без LLM (rule-based synthesis fallback). Панель вже очікує це.

**6) Word Add-in (панель)**

* **[word\_addin\_dev/taskpane.html]** (є) → заголовок, бейджі, розділи, кнопки, завантаження бандла з cache-buster (з манифесту). Перемикач “Show raw JSON”, блок Suggested edits, консоль — **ідеально для діагностики**.
* **[word\_addin\_dev/taskpane.bundle.js]** (є) →  
  **Що вже є:**
  + кеш BUILD, LS-збереження бек-URL, Test/Save, Office.js перевірка, читання selection/whole doc, вставка в Word, бейджі, рендер findings/recs, телеметрія кліків, POST /api/analyze//api/gpt/draft гачки.   
    **Що додати:**
  + **officeBridge**: addCommentAtSpan(span, text), jumpToSpan(span), applyTrackedChanges(diff);
  + **actions.AnnotateDocument:** ставить **коментарі тільки на проблемних** спанах (risk≥threshold), з коротким титулом [CODE] message.
  + **QA-recheck**: повторний виклик /api/qa-recheck і оновлення бейджів.  
    **Тест (ручний):** “Use whole doc → Analyze → коментарі з’явились лише там, де ризики; ‘Get AI Draft’ → ‘Insert result’ працює; ‘Re-analyze’ показує зменшення ризиків”.

**7) Learning (локальна адаптація)**

* **[learning/\*]** (нове) → replay\_buffer.jsonl, adaptor.py, політика.  
  **Що робить:** лог подій “прийняв/відхилив”, періодичний апдейт ваг шаблонів (без зовнішніх відправок).  
  **Тест:** кілька однакових прийнять → шаблон “наближається”.

**8) Дев-скрипти / самодіагностика**

* **[STEP1\_CLEAN\_START.ps1, STEP2\_BACKEND\_FRONT.ps1, STEP3\_OPEN\_WORD.ps1]** (є) → синхронізувати з cache-buster (BUILD у маніфесті/бандлі), повна чистка WEF-кешів.
* **[panel\_selftest.html]** (є у плані; створити) → автономний “пінг” API та емуляція відповіді для швидкої перевірки UI без Word.
* **[tests/test\_endpoints.ps1]** (нове) → смоук: /health, /api/analyze (2 уривки → різні findings), /api/gpt/draft (draft\_text ≥ 20 символів), /api/suggest (≥1 картка), /api/qa-recheck (диф метрик).

**Code (специфікація без коду)**

*Контракти DTO (узгодити для всіх ендпоїнтів):*

* AnalyzeIn { document\_name, text } → AnalyzeOut { document\_name, clauses[], results: AnalysisOutput[] }
* DraftIn { clause\_type, text, findings[], constraints?, jurisdiction? } → DraftOut { draft\_text, alternatives[], rationale?, citations\_hint[] }
* SuggestIn { clause\_id?, clause\_type, text, mode } → SuggestOut { suggestions: [ { suggestion\_id, clause\_id, action, proposed\_text, reason?, sources[], span?, hash } ] }
* QARecheckIn { document\_name, text, applied\_changes[] } → QARecheckOut { deltas: {scoreΔ, riskΔ, statusΔ}, residual\_risks[] }

*SSOT інваріанти:*

* score ∈ [0..100]; critical ⇒ status=FAIL; “без findings” ⇒ status=PASS.
* кожний Finding має code, message, severity, **опційно** evidence, citations[], span.
* citations: {system: UK|UA|EU, instrument, section, url?}.

**Tests (що перевіряємо на кожному кроці)**

1. **Схеми:** валідність DTO/SSOT, відсутність None у required, межі score.
2. **Rules:** property-тест — на будь-якому тексті analyze() не падає; зміна ключової умови ⇒ з’являється очікуваний Finding.
3. **Pipeline:** однаковий текст ⇒ однаковий Span; різні уривки ⇒ різні findings/recs.
4. **API:** /health=200; /api/analyze та /api/gpt/draft стабільні при timeouts/empty. Панель “Test” показує OK (Conn: 200).
5. **UX (manual):** “Analyze whole doc” додає тільки проблемні коментарі; “Get AI Draft → Insert” працює; “Re-analyze” зменшує ризик.

**Validation (критерії приймання)**

* Детермінований rule-аналіз (без LLM) + 25–32 кроки/клауза; cross-checks працюють.
* Відповіді API **строго** відповідають SSOT-контрактам; навіть на порожньому вводу — валідний JSON.
* У Word з’являються **коментарі лише на проблемах**; панель синхронно показує бейджі й списки; вставка/прийняття змін без збоїв.
* Цитування: UK/UA/EU із інструментом і секцією (URL — опційно).
* Безпека: нічого не йде назовні без прапора; локальна історія для learning.
* Продуктивність: Analyze ≤ 2–5 с на 10–15 сторінок; Draft (rule-based) ≤ 1 с.

**Next Steps (чітко, покроково)**

1. **Заморозити SSOT** (core/schemas.py) + **compat-шар** (розділ 0).
2. **Завершити API** (/health, /api/analyze, /api/gpt/draft, /api/suggest, /api/qa-recheck) — всі повертають валідний JSON завжди.
3. **Intake + Spans**: гарантувати стабільні Span/DocIndex.
4. **Rules v1 (10 клауз)**: Parties/Definitions, Term/Termination, Payment, Deliverables, Governing Law & Jurisdiction, Confidentiality, LoL, Indemnity, Force Majeure, Entire Agreement — 25–32 кроки + cross-checks.
5. **AnnotateDocument у панелі**: коментарі лише на проблемах, навігація jumpToSpan.
6. **Draft modes** (friendly/standard/strict) — вставка з tracked changes.
7. **QA-recheck** — швидкий повторний аналіз після застосування змін.
8. **Learning (локально)** — лог подій, адаптація шаблонів.

**Варіанти реалізації (2–3) з плюсами/мінусами/вартістю**

**A) Модульний моноліт (поточний шлях)**

* Простий деплой/дебаг, швидке MVP, одна кодова база.  
  − Потрібні дисципліна модулів та чіткі контракти.  
  **Вартість:** низька (швидко в пілот).

**B) Мікросервіси (API/Rules/Draft/DD/Reports)**

* Незалежне масштабування, ізоляція відмов.  
  − Складніша оркестрація/трасування, версіонування DTO.  
  **Вартість:** висока (після PMF).

**C) Serverless-гібрид (Functions + довгі задачі у черзі)**

* Pay-per-use, авто-масштабування, легко обмежити трафік.  
  − Колд-старти, ліміти виконання, локальна розробка складніша.  
  **Вартість:** середня.

**Рекомендація:** зараз **A**, з чіткою дорогою міграції на **B/C** після появи навантаження.

**STATE LOG**

**Мета:** супер-система юридичного аналізу/драфтингу (вище Luminance), готова для Innovator Founder Visa, із rule-engine 25–32 кроків, GPT-proxy, Word-панеллю, due diligence та CMS.  
**Артефакти:** робочий каркас панелі (HTML/JS) з правильними гачками та діагностикою; структура бекенду (engine/api/gpt/legal\_rules); скрипти запуску.   
**TODO (найближче):** 1) Freeze SSOT + compat; 2) стабілізувати API; 3) Intake+Spans; 4) Rules v1 (10 клауз) + cross-checks; 5) AnnotateDocument та QA-recheck у панелі.  
**Ризики:** несинхрон DTO↔UI; нестача/якість юридичних цитат; кеш/версіювання маніфестів у Word; продуктивність на довгих документах.  
**Примітка:** фронт уже вміє показувати бейджі/списки/консоль і має кнопки для повного сценарію — це треба лише підживити стабільними відповідями бекенду та spans/annotate.

Готовий перейти до **Кроку 1 — Freeze SSOT/DTO**: перелічу точні поля з типами/валідаціями та чек-лист “пройшло/не пройшло” для кожного ендпоїнта.

так, **КРОК 5 – завершено**. Нижче — стисле, але вичерпне оновлення: що зроблено, яка нині архітектура й логіка, та що ми модернізували порівняно з планом. Формат: **Architecture/Design → Plan → Code → Tests → Validation → Next steps → STATE LOG**.

**Architecture / Design (поточний зріз)**

**Модульний моноліт, SSOT-контракти, повна сумісність із панеллю Word.**  
Головні вузли й потоки:

* **Word Add-in (панель)** → POST /api/analyze, POST /api/gpt/draft, POST /api/qa-recheck.
* **API (FastAPI, app.py)**: CORS (https://localhost:3000 + 127.0.0.1), x-cid трасування, **envelope** {meta{cid,schema\_version,server\_time,cache?},data{...}}, ліміти (2.5MB), Semaphore(4), таймаути (15s analyze / 5s draft), **LRU idempotency** на /api/analyze.
* **Orchestrator (api/orchestrator.py)**: єдиний вхід для бізнес-логіки ендпоїнтів.
* **Pipeline (engine/pipeline.py)**: segment\_document → analyze\_index\_with\_rules → cross\_checks → DocumentAnalysis (SSOT) + synthesize\_draft(… , friendly/standard/strict).
* **Compat (engine/pipeline\_compat.py)**: **міст** SSOT → legacy (analysis/results/clauses) з clamp-політиками.
* **Rule Engine**: legal\_rules/rules/\* (клаузні правила, 12–15 швидких перевірок v1; дорожня карта до 25–32), **єдиний runner** (registry.run\_rule) з **per-rule timeout + circuit-breaker**; **cross\_checks.py** (GL↔JUR, TERM↔NOTICE/LoL, CONF↔DP, FM↔PAY, Survival, IP↔License).
* **LLM Drafting (gpt/\*)**:
  + gpt\_prompt\_builder.py: режимні промпти (friendly/standard/strict), no-hallucinations.
  + gpt\_draft.py: **decision tree** (rule-template → LLM(guarded) → rule-fallback), **guardrails** (без вигаданих норм, без markdown/disclaimer, clamp≤2000, стиль за режимом, вирівнювання під high/critical).
  + gpt\_orchestrator.py: єдиний виклик run\_draft(...).
  + gpt\_proxy\_api.py: стабільний мок (повертає draft\_text завжди).
* **Core/SSOT (core/schemas.py)**: risk-ordinal **RISK\_ORDER** + risk\_to\_ord/ord\_to\_risk (QA-дельти), DeltaMetrics.risk\_delta документовано як **різниця ординалів**.

**Plan (що саме ми закрили в КРОЦІ 5)**

1. **Промпт-білдер із режимами** (friendly/standard/strict) і whitelist-джерел (тільки з analysis.citations).
2. **Guarded drafting**: новий модуль gpt\_draft.py з guardrails + вирівнювання під високі ризики.
3. **Єдиний оркестратор драфту** (gpt\_orchestrator.run\_draft) з **decision tree** і гарантованим rule-fallback.
4. **LLM-проксі мок** (gpt\_proxy\_api.py) із легкими guardrails, стабільним draft\_text.
5. **API сумісність**: /api/gpt/draft повертає DraftOut незалежно від LLM-прапора; панель не ламається.

**Code (що модернізовано проти початкового плану)**

* **Decision tree для драфту**: пріоритет analysis.proposed\_text → потім LLM (якщо LLM\_ENABLED) → потім pipeline.synthesize\_draft.
* **Guardrails** (уніфіковано в одному місці):
  + no new sources (whitelist з citations),
  + no markdown/disclaimers,
  + стилі за режимом,
  + alignment з high/critical findings (вставляємо мінімальні обов’язкові формули, не знижуючи rule-рівень).
* **Єдиний runner правил** у Rule Engine (таймаут + CB) — без дублюючих threadpool-викликів.
* **SSOT-envelope** в API-відповідях + x-cid, **LRU idempotency** на /api/analyze.
* **Risk ordinal** інтегрований у QA-дельти; DeltaMetrics.risk\_delta = різниця ординалів (−3..+3).
* **Absolute spans & anchors** з intake → стабільні коментарі в Word.

**Tests (що має проходити)**

**Unit**

* Prompt builder: тільки дозволені джерела; різні режими дають різні стилістичні вимоги.
* Guardrails: видаляє disclaimers/markdown; нейтралізує невідомі посилання; clamp≤2000.
* gpt\_draft.generate\_guarded\_draft:
  + при proposed\_text повертає його (очищений),
  + при LLM-помилці → rule-fallback,
  + при critical у findings → додає вирівнювальні хінти (“shall”, timelines).

**Property**

* Будь-який валідний AnalysisOutput → draft\_text **ніколи** не порожній; довжина у межах.

**E2E**

* /api/gpt/draft з LLM\_ENABLED=False → rule-based текст (≥20 символів).
* /api/gpt/draft з LLM\_ENABLED=True (мок) → без нових джерел і disclaimers; sources[] відповідає citations.
* /api/analyze повторно з тим самим ключем → X-Cache: hit, той самий document + legacy.

**Validation (критерії приймання КРОКУ 5)**

* **Стабільність**: /api/gpt/draft завжди повертає валідний DraftOut, незалежно від LLM.
* **Точність**: не з’являються “вигадані” норми; тексти не знижують rule-ризики.
* **Сумісність**: контракти I/O незмінні для панелі (legacy + SSOT).
* **Швидкість**: rule-fallback ≤ ~1s/клауза (локально).

**Що саме покращено проти початкового плану**

* ✅ **Замість “LLM-перше”** — **rule-перше** (детермінізм, нижча вартість, безпека).
* ✅ **Guardrails централізовано** (раніше — частинами): whitelist джерел, no-disclaimer, no-markdown, clamp, стиль/ризик-alignment.
* ✅ **Єдиний runner правил** (таймаут/CB) → відсутність дубльованих пулів і race conditions.
* ✅ **Idempotency + envelope** у всіх ключових відповідях → краща діагностика й UX у панелі.
* ✅ **Risk ordinal** формалізовано → коректні QA-дельти та агрегування ризиків.

**Next steps (КРОК 6 → далі, файл за файлом)**

1. **Front Word (панель)**
   * officeBridge: addCommentAtSpan, jumpToSpan, applyTrackedChanges.
   * actions.AnnotateDocument: коментарі лише на проблемних spans (risk≥threshold, сортування за severity).
   * actions.QARecheck: виклик /api/qa-recheck, показ дельт.
2. **Rules v2**
   * Дорощуємо з 12–15 до **25–32** перевірок/клауза (Conf, GL, JUR, Term, LoL, Indemnity, FM, Definitions, IP, Payment).
   * Розширюємо cross\_checks (Survival, IP↔License, Payment↔FM тощо) + доповнюємо citations.
3. **Learning (локальна адаптація)**
   * learning/replay\_buffer.jsonl, adaptor.py: запис “прийнято/відхилено”, м’які ваги для шаблонів; **без** зовнішнього витоку.
4. **DevOps & QA**
   * tests/test\_endpoints.ps1 (smoke), panel\_selftest.html (автономна перевірка панелі), збір таймінгів і лічильників cache-hit.

**STATE LOG**

**Цілі:** гарантія драфту без LLM; при LLM — строгі guardrails; детермінізм, швидкість, прозорість.  
**Артефакти:**

* gpt\_prompt\_builder.py (режими+whitelist),
* gpt\_draft.py (decision tree + guardrails + alignment),
* gpt\_orchestrator.py (єдиний виклик),
* gpt\_proxy\_api.py (стабільний мок),
* API envelope + LRU idempotency, risk-ordinal у схемах.  
  **TODO:** КРОК 6 — анотації у Word та QA-дельти в UI; Rules v2 до 25–32 перевірок/клауза; learning.  
  **Ризики:** наджорсткі guardrails у вузьких кейсах (пом’якшується rule-fallback); продуктивність на дуже довгих документах (вирішуємо clamps + кеш + семафор).