

Document\_analiser – Документация

Подробно описание на система за анализ на документи с помощта на векторно търсене и генеративен ИИ (LLM), реализирана със Spring Boot, PostgreSQL/pgvector и Thymeleaf UI.

Автор: Илия Николаев Балтаджиев   
Ф.н: 22621654  
Факултет: ФИТА  
Курс: III  
Група: 3а  
Семестър: 6  
Дата: 09.09.2025г.  
 Проверил:

Описание

Как да четете документа:

- Част 01–03: ориентирана към непрофесионална аудитория (какво прави системата и как се ползва)

- Част 04–07: архитектура и вътрешна реализация

- Част 08–11: конфигурации, сигурност, производителност, мониторинг

- Част 12–15: практични съвети, FAQ, речник и бъдещо развитие

01 – Резюме (Executive Summary)

* Проблем: Хората трудно намират бърз и точен отговор в дълги документи.
* Решение: Комбинация от векторно търсене (намира най-сходните откъси) и LLM (генерира кратък, ясен отговор), стъпила върху надежден стекизъм от технологии.
* Резултат: Уеб приложение, в което качвате документ, задавате въпрос и получавате отговор с контекст от документа. Въпросите и отговорите се пазят и могат да се разглеждат в „History“.

Как работи накратко

1) Качване на файл → системата извлича текст, разделя го на части (chunk-ове) и изчислява числово представяне (embedding) на всяка част.

2) Въпрос → системата изчислява embedding за въпроса и търси най-близките части в базата (pgvector).

3) Отговор → системата подава намерения контекст към ИИ модел, който връща четим отговор. Отговорът се запазва и визуализира.

Защо е подходящо за непрофесионална аудитория - Интерфейсът е прост: „Upload → Ask → History“. - Термините са обяснени с аналогии (вж. Речник и Въведение).

Технологии (накратко) - Backend: Spring Boot, Spring MVC, Spring Security (JWT/Forms), Spring Data JPA, Micrometer, Caffeine Cache. - База: PostgreSQL с pgvector за векторни индекси (сходство между текстове). - UI: Thymeleaf + Bootstrap. - ИИ: embedding + чат клиент (абстрахирано през AiChatClient).

Ключови предимства - Бързи отговори върху реално съдържание, а не „измислици“ (hallucinations), благодарение на контекст от оригиналния документ. - История и филтриране на въпроси по документ. - Наблюдение на производителност и кеширане за по-добра скорост.

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Генерираното от ИИ съдържание може да е неправилно.

02 – Въведение и цели

Какъв е проблемът - Дълги правни/технически документи изискват време и знания, за да се намери точен отговор. - Често потребителят знае „какво търси“, но не и „къде да го търси“ в текста.

Какво предлага системата - Качвате документ (например .txt). Системата разделя текста на смислови части (chunk-ове) и ги представя като числа (embeddings), които могат да се сравняват математически. - Когато зададете въпрос, той също се превръща в embedding. Сравняваме „числата“ на въпроса с „числата“ на частите от документа и избираме най-сходните. - ИИ модел получава именно тези „най-важни“ части като контекст и формулира кратък отговор.

Обяснение с аналогия - Представете си, че имате библиотека. Вместо да четете всички книги, имате каталог по теми и ключови думи. Embedding-ът е богат каталог: числово описание на смисъла. Търсенето не е „по букви“, а „по смисъл“.

Цели на проекта - Да демонстрира реално приложим RAG (Retrieval-Augmented Generation) подход. - Да бъде лесен за използване от хора без ИТ опит. - Да е технически коректен (сигурност, производителност, мониторинг).

03 – Ръководство за потребителя

Минимум стъпки 1) Вход: отворете приложението и влезте (или използвайте логин страницата). 2) Качете документ: меню Upload → изберете .txt файл → Upload. 3) Задайте въпрос: меню Ask → изберете документ → напишете въпрос → Submit. 4) Вижте отговора и историята: на същата страница се показва отговорът; през „History“ може да преглеждате предишни въпроси/отговори и да филтрирате по документ.

Съвети - Формулирайте въпросите конкретно: „Какъв е срокът на договора?“ е по-добре от „Информация за договор“. - Ако отговорът е дълъг, използвайте „See more“ за разгъване. - През „History“ можете да сортирате по нови/стари и да филтрирате по конкретен документ.

Демонстрационен сценарий (за презентация) - Качете sample-data/legal\_demo\_contract\_bg.txt. - Попитайте: „Какъв е срокът на договора?“ или „Кои са задълженията на доставчика?“. - Покажете как системата намира отговори бързо, без да преглеждате целия текст.

Картина, която съдържа текст, софтуер, екранна снимка

Генерираното от ИИ съдържание може да е неправилно.

04 – Архитектура и потоци

Общ преглед

- Три основни слоя: UI (Thymeleaf), Backend (Spring Boot), База (PostgreSQL + pgvector).

- Кеширане: Caffeine.

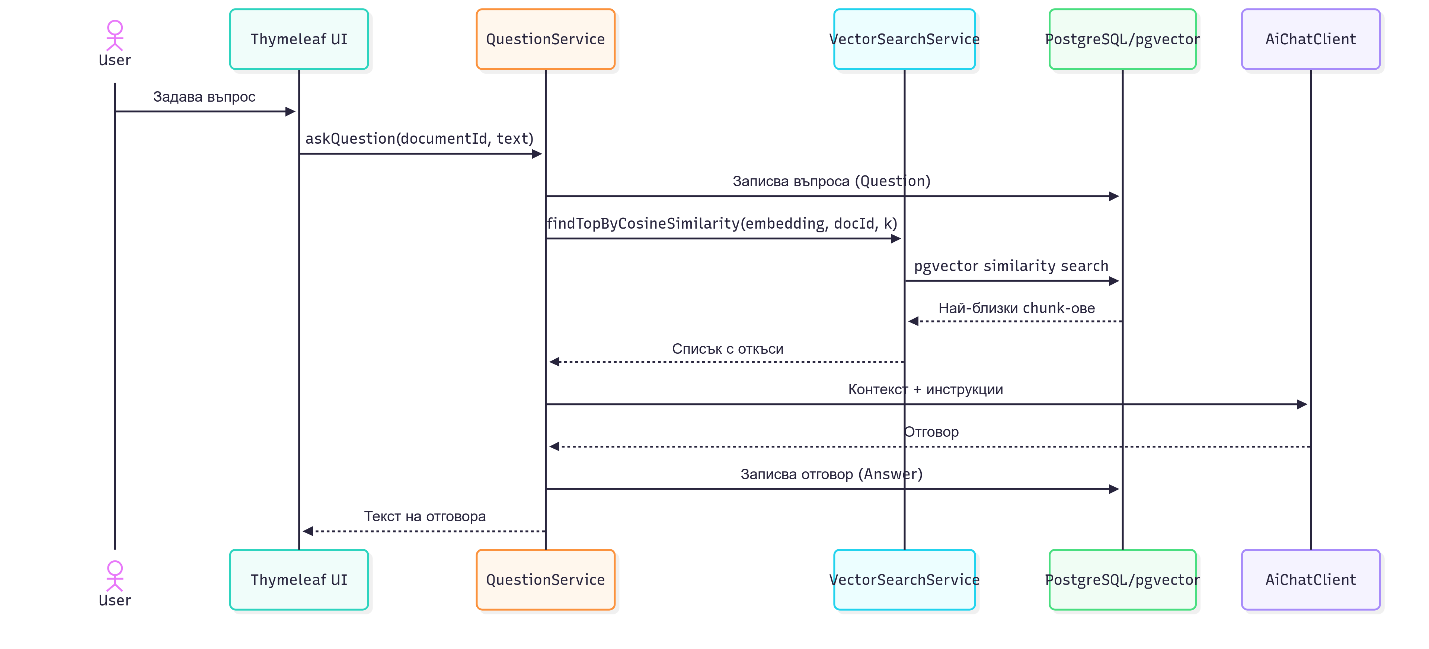
- Наблюдение: Micrometer + Actuator.

Високо ниво

Картина, която съдържа диаграма, текст, линия, Шрифт

Генерираното от ИИ съдържание може да е неправилно.

Последователност: Въпрос → Отговор



Ключови класове - UI контролери: src/main/java/com/example/Document\_analiser/controller/UiController.java, QuestionHistoryPageController.java.

- REST контролери: DocumentController.java, QuestionController.java, CacheController.java, PerformanceController.java, PerformanceTestController.java, DebugController.java, PublicQuestionController.java.

- Сървиси: QuestionService.java, DocumentService.java, VectorSearchService.java, ResponseTimeAnalyzer.java, LogAnalysisService.java.

- Сигурност: SecurityConfig.java, JwtFilter.java, CustomUserDetailsService.java, JwtUtil.java.

- Конфигурации: CacheConfig.java, EmbeddingConfig.java, PerformanceConfig.java, WebConfig.java, PgvectorIndexInitializer.java.

Какво е embedding и pgvector

- Embedding: числов „отпечатък“ на смисъла на текст. Позволява да мерим сходство между изречения/абзаци.

- pgvector: разширение на PostgreSQL, което добавя тип „vector“ и оператори за търсене по сходство (косинусова близост и др.).

05 – Модел на данните

Основни ентитита (JPA)

- Document (src/main/java/com/example/Document\_analiser/entity/Document.java) - Полета: id, name, type, uploadDate, content (голям текст, lazy), chunks (1:N към DocumentChunk).

- DocumentChunk (src/main/java/com/example/Document\_analiser/entity/DocumentChunk.java) - Полета: id, chunkIndex, content, embedding (vector(1536)), document.

- Question (src/main/java/com/example/Document\_analiser/entity/Question.java) - Полета: id, text, askedAt, topic, document, user, answer (1:1).

- Answer (src/main/java/com/example/Document\_analiser/entity/Answer.java) - Полета: id, text (text), generatedAt, question.

- User (src/main/java/com/example/Document\_analiser/entity/User.java)

- Полета: id, username, password, role.

Репозитории (Spring Data JPA)

- DocumentRepository, DocumentChunkRepository, QuestionRepository, UserRepository.

Специални заявки:

- Векторно търсене: DocumentChunkRepository.findTopIdsByCosineSimilarity(...) (native SQL с pgvector оператор <=>).

- История на въпроси: QuestionRepository.findHistory(...) (филтри по потребител/документ и странициране).

Бележки за производителност

- Избягване на Fetch join при странициране (може да доведе до грешки/дублиране).

- Прожекции за „content“/„embedding“, за да не се четат ненужно големи колони.

06 – Компоненти в кода

Потоци и логика

- Качване на документ:

UiController (/upload) → DocumentService.store(...)

- Извличане на текст (TxtDocumentTextExtractor), разделяне на части, изчисляване на embeddings, запис в БД.

- Задаване на въпрос: UiController (/ask) или REST /api/questions → QuestionService.askQuestion(...)

- Запис на въпроса, намиране на релевантни chunk-ове (VectorSearchService + pgvector), изграждане на контекст и заявка към LLM, запис на отговор.

- История: QuestionHistoryPageController (/history) и /api/questions/history (REST) → QuestionService.getHistory(...).

Ключови класове (по-важни методи)

- QuestionService (src/main/java/com/example/Document\_analiser/service/QuestionService.java)

- askQuestion(...): основен процес по генериране на отговор.

- getHistory(...): страницирана история за потребител + филтри по документ.

- DocumentService (src/main/java/com/example/Document\_analiser/service/DocumentService.java)

- store(...): валидация, извличане на текст, chunk-ване, embeddings, запис.

- VectorSearchService (src/main/java/com/example/Document\_analiser/service/VectorSearchService.java)

- findTopByCosineSimilarity(...): извиква pgvector заявки и връща съдържание за контекста.

Кеширане - CacheConfig → три cache manager-а: по подразбиране, за embeddings (по-дълъг TTL), за „бързи“ данни (къс TTL).

- Аннотации @Cacheable в QuestionService и DocumentService за често използвани резултати.

Наблюдение

- PerformanceConfig + @Timed върху важни методи.

- PerformanceInterceptor и RequestTracingFilter за време на отговор и корелационни идентификатори в логовете.

07 – API справочник (основни)

UI маршрути (изискват логин)

- / – начална страница.

- /ask – форма за задаване на въпрос; показва последния отговор.

- /upload – качване на документ.

- /history – история с филтър по документ и сортиране.

REST API (JWT или сесия)

- POST /api/documents – качване на документ (multipart form-data, поле file).

- POST /api/questions – задаване на въпрос (JSON QuestionRequest). - GET /api/questions/history?documentId={id}&order=asc|desc&page={n} – страницирана история.

- PUT /api/questions/{id} – редактиране (само собственик или ADMIN).

- DELETE /api/questions/{id} – изтриване (само собственик или ADMIN).

Пример:

QuestionRequest (src/main/java/com/example/Document\_analiser/dto/QuestionRequest.java)

{  
 "documentId": 1,  
 "text": "Какъв е срокът на договора?"  
}

Отговор: AnswerResponse

{  
 "answer": "Срокът на договора е ...",  
 "generatedAt": "2025-09-07T13:21:04"  
}

Забележка: UI и REST споделят същите сървиси; UI подава формуляри, а REST – JSON.

08 – Конфигурация и среди

Основен файл:

src/main/resources/application.properties

- База данни: PostgreSQL URL, потребител, парола.

- JPA: ddl-auto=update (за учебни цели), логване на SQL, диалект.

- Embeddings/LLM: модел за embedding (embedding.model).

- Кеш: Caffeine спецификации и имена на кешове.

- Actuator/Metrics: прометей, метрики, профили.

Препоръки

- Не съхранявайте реални секрети (API ключове, пароли) в git. Използвайте environment variables или vault.

- За production – миграции с Flyway/Liquibase вместо ddl-auto=update.

- Ограничете размерите на качваните файлове според нуждите.

Среда за разработка

- Локален PostgreSQL + инсталиран pgvector разширение.

- Конфигурация на таблицата за vector(1536) (виж PgvectorIndexInitializer).

09 – Сигурност

Какво ползваме

- Spring Security: филтрира заявки, пази контекста на потребителя, показва login/logout страници.

- JWT филтър (JwtFilter): поддържа Bearer токени за REST клиенти.

- CustomUserDetailsService: зарежда потребителя от базата.

- Роли: USER/ADMIN (примерни), използвани за контрол на достъпа до редакции/изтриване.

Потоци

- UI: потребителят влиза през формата за вход → получава сесия → достъп до /ask, /upload, /history.

- REST: клиентите изпращат Authorization: Bearer <JWT>. Ако е валиден → заявката е оторизирана.

Най-добри практики (учебни/продукционни)

- Шифрирани пароли (BCrypt)

– конфигуриран PasswordEncoder.

- Къси срокове на живот за JWT; освежаване през отделен endpoint.

- Ограничаване на публичните endpoints в SecurityConfig само до необходимите.

10 – Производителност

Подходи

- Кеширане (Caffeine): embeddings и често ползвани резултати (документи, история).

- Метрики (Micrometer + @Timed): измерване на време за ключови операции.

- Интерцептор (PerformanceInterceptor): общо време на заявките и логване на бавни заявки.

- Векторни заявки: pgvector оператори за бързо търсене по сходство.

- JPA оптимизации: избягване на тежки fetch-и при странициране, прожекции за големи колони.

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Генерираното от ИИ съдържание може да е неправилно.

Как да проверим

- Actuator/metrics и логове – вижте времена на отговор, бавни заявки, кеш статистики.

- Наблюдавайте размери на документи и броя chunk-ове (големи документи → повече embeddings → повече място и време).

Съвети

- Баланс между размер на chunk и брой chunk-ове – зависи от домейна.

- Ограничете TOP-K в търсене (примерно 5) – достатъчно контекст без излишък.

11 – Логване и мониторинг

Логване

- RequestTracingFilter: добавя X-Correlation-Id, логва начало/край, статус, потребител.

- Нива в application.properties: подробни логове за JPA/SQL и кеширането (за учебни цели).

Мониторинг

- Micrometer + Actuator: endpoints за health, info, metrics, prometheus.

- Метрики за време: @Timed в QuestionService, DocumentService и др.

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Генерираното от ИИ съдържание може да е неправилно.

Какво да гледаме

- Време за обработка на въпроси, грешки при embeddings/LLM, бавни заявки (>2s), cache hit/miss.

12 – Отстраняване на проблеми

/history дава 500 (Internal Server Error)

- Причина 1: Thymeleaf event handler съдържа текст от променлива (мерки срещу XSS).

Решение: пазим текста в data-\* атрибут и четем от this.dataset (вече е поправено в templates/history.html).

- Причина 2: Fetch join + странициране в JPA заявка.

Решение: махнат е FETCH в QuestionRepository.findHistory.

Грешки при качване

- „File size exceeds 5MB limit“ – променете ограничението, ако е необходимо. - „UnsupportedFileTypeException“ – за демото е активен txt екстрактор; добавете PDF/DOCX екстрактори при нужда.

Празни/неточни отговори

- Проверете дали документът е качен успешно и има смислов текст.

- Увеличете TOP-K (брой контекстни chunk-ове) внимателно.

База данни/pgvector

- Уверете се, че разширението pgvector е инсталирано и колоната е vector(1536).

Сигурност (401/403)

- За REST – проверете JWT в Authorization: Bearer ....

- За UI – уверете се, че сте логнати.

13 – Често задавани въпроси (FAQ)

Какво е „embedding“ с прости думи? - Числово представяне на текст, което улавя смисъла. Позволява математически да търсим „по смисъл“.

Ще „измисли“ ли системата нещо, което го няма в документа? - Целта е да се ограничи това, като подаваме реален контекст от документа. Въпреки това, винаги валидирайте важни решения.

Защо са нужни pgvector и PostgreSQL? - За да можем бързо да намираме сходни откъси по embedding. pgvector добавя този тип търсене към PostgreSQL.

Може ли да работи офлайн? - Двигателят за embeddings в демото е детерминистичен и локален (за учебни цели). В продукционна среда бихте ползвали реален embedding/LLM доставчик.

14 – Речник на термините

* Embedding: числово описание на текст или обект.
* Vector: масив от числа; използваме го за сравнение на сходство.
* Cosine similarity: мярка за сходство между два вектора (по ъгъла между тях).
* RAG (Retrieval-Augmented Generation): подход „намери релевантен контекст → генерирай отговор“.
* LLM: Large Language Model – модел, който генерира текст.
* JWT: JSON Web Token – подписан токен за удостоверяване.
* Caffeine: бърз in-memory кеш за Java.
* Micrometer/Actuator: инструменти за метрики и наблюдение на Spring приложения.

15 – План за развитие (Roadmap)

* Поддръжка на PDF/DOCX екстрактори и извличане на изображения/таблици.
* Обогатени отговори: цитати с референции към оригиналния текст и страница.
* По-добра сигурност: refresh токени, ограничаване на сесии, 2FA (по избор).
* Настройваеми параметри: размер на chunk, TOP-K, модели за embeddings/LLM.
* Автоматични миграции на БД (Flyway), профили за среди (dev/test/prod).
* UI подобрения: мулти-документ контекст, highlight на пасажи.

16 – Изисквания

Функционални изисквания - Качване на документи и извличане на текст. - Задаване на въпроси върху избран документ. - Показване и филтриране на история на въпроси/отговори. - REST API за интеграция.

Нефункционални изисквания - Производителност: отговор < 2s за типични заявки (локална среда). - Надеждност: обработка на грешки, ясни съобщения за потребителя. - Сигурност: изисква се логин; JWT за REST. - Поддръжка: логове, метрики, конфигурация чрез свойства.

Ограничения - Демонстрационен екстрактор за .txt; други формати изискват допълнителни екстрактори. - Локален embedding генератор за учебни цели.

17 – Употреба и сценарии

Сценарий 1: Правен отдел - Качва договор, търси срокове/неустойки. - Печалба: бърз ориентир без четене на целия документ.

Сценарий 2: Техническа поддръжка - Качва ръководство/README, търси стъпки за конфигурация.

Сценарий 3: Студентска работа - Документиране на проект, демонстрация на RAG концепция и сигурност.

Актьори и потоци - Потребител (UI), Интеграция (REST клиент), Администратор.

18 – Подробна инсталация

Предварителни условия - Java 17+, Maven, PostgreSQL 15+ с разширение pgvector. - Създайте база document\_analyser и потребител с права.

Стъпки 1) Клониране и настройка на application.properties (DB креденшъли, портове).

2) Инициализация на pgvector колона (виж PgvectorIndexInitializer).

3) Стартиране: mvn spring-boot:run или от IDE.

4) Достъп: http://localhost:8080 (login форма).

Създаване на потребител (примерен SQL)

INSERT INTO users(username, password, role) VALUES('test', '{bcrypt}$2a$10$...', 'USER');

Забележки - В демо режим е активен детерминистичен embedding клиент. - За реална среда – конфигурирайте външен embedding/LLM доставчик.

19 – Деплоймънт

Локално - Профил dev, вграден Tomcat, външен PostgreSQL.

Сървър - Създайте системен потребител, инсталирайте JDK, PostgreSQL+pgvector. - Постройте jar: mvn -DskipTests package → target/\*.jar. - Стартирайте като услуга (systemd) с environment variables за секрети.

Контейнери (опционално) - Dockerfile за Spring Boot + PostgreSQL контейнер с pgvector. - Оркестрация: docker-compose/Kubernetes.

Миграции - Препоръчва се Flyway/Liquibase за версиониране на схемата.

20 – Сигурност (детайлно)

Конфигурация (SecurityConfig.java – извадка)

http  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/auth/\*\*","/","/css/\*\*","/js/\*\*",   
 "/swagger-ui/\*\*","/v3/api-docs/\*\*").permitAll()  
 .anyRequest().authenticated()  
 )  
 .formLogin(Customizer.withDefaults())  
 .logout(Customizer.withDefaults());  
http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

Потоци за удостоверяване - UI: формен логин → сесия. - REST: Authorization: Bearer <JWT> → JwtFilter валидира токена.

Контрол на достъпа - @PreAuthorize за редакция/изтриване на въпроси (owner/Admin).

Хардънинг и препоръки - Ограничаване на публичните маршрути, HTTP headers (HSTS/CSP), rate limiting. - Сигурно съхранение на секрети (env vars/KeyVault).

21 – Производителност и бенчмаркове

Методика - Локално, 1 документ (~50KB), 300–500 chunk-а. - 10 последователни заявки към /ask и /api/questions.

Очаквани резултати (примерни) - Време за търсене (pgvector): 20–60 ms. - Общо време за отговор: 200–800 ms (без външен LLM).

Наблюдение - Actuator метрики: question.processing.time, question.history.time, embedding.generation.time. - Логове за бавни заявки (>2s) от PerformanceInterceptor.

Тунинг - Кеш TTL-и, размер на chunk, TOP-K, DB index-и, connection pool.

22 – Примери за API

Създаване на въпрос

curl -X POST http://localhost:8080/api/questions \  
 -H "Content-Type: application/json" \  
 -H "Authorization: Bearer <JWT>" \  
 -d '{"documentId":1, "text":"Какъв е срокът на договора?"}'

История на въпроси

curl -H "Authorization: Bearer <JWT>" \  
 "http://localhost:8080/api/questions/history?order=desc&page=0"

Качване на документ

curl -X POST http://localhost:8080/api/documents \  
 -H "Authorization: Bearer <JWT>" \  
 -F "file=@sample-data/legal\_demo\_contract\_bg.txt"

Отговори (примери)

{  
 "answer": "Срокът на договора е 12 месеца...",  
 "generatedAt": "2025-09-07T13:21:04"  
}

23 – База и SQL примери

Схема (опростено) - Таблици: documents, document\_chunks, questions, answers, users. - Колона document\_chunks.embedding е vector(1536) (pgvector).

Създаване на индекс (пример)

-- Изисква pgvector разширение  
CREATE EXTENSION IF NOT EXISTS vector;  
CREATE INDEX IF NOT EXISTS idx\_chunks\_embedding   
 ON document\_chunks   
 USING ivfflat (embedding vector\_cosine\_ops) WITH (lists = 100);

Примерна заявка за сходство

SELECT id   
FROM document\_chunks   
WHERE document\_id = :docId  
ORDER BY embedding <=> CAST(:embedding AS vector(1536))  
LIMIT :k;

24 – Диаграми (ER, класове)

ER диаграма (описателно) - Document 1..N DocumentChunk - Question 1..1 Answer - Question N..1 Document - Question N..1 User

UML класове (описателно) - Entity: Document, DocumentChunk, Question, Answer, User - Service: QuestionService, DocumentService, VectorSearchService - Controller: UiController, QuestionController, DocumentController

Съвет: Използвайте инструмент (PlantUML/Mermaid) за автоматично генериране, ако желаете визуални изображения.

25 – Тестова стратегия

Нива - Unit: сервизни методи (chunk-ване, cosineSimilarity, мапинг към DTO). - Integration: репозитории и pgvector заявки. - E2E (опционално): през REST/UI с тестов потребител.

Инструменти - JUnit 5, Spring Boot Test, Testcontainers (за PostgreSQL, ако се желае).

Примерни случаи - QuestionService.getHistory връща правилен ред при asc/desc. - DocumentService.chunkText не връща празни chunk-ове. - Векторно търсене връща резултати в очакван ред.

26 – Модел на заплахи

Активи - Лични данни (потребители), качени документи, история на въпроси.

Заплахи - Неоторизиран достъп до UI/REST → ограничено чрез Spring Security/JWT. - XSS в UI → Thymeleaf escaping; избягване на инлайн текст в event handlers (fix в history.html). - SQL Injection → JPA параметризирани заявки. - Изтичане на секрети → използване на env vars вместо твърдо кодирани ключове.

Контроли - Роли и проверки за собственост при редакция/изтриване. - Логване и мониторинг на аномалии.

27 – Анализ на алтернативи

Търсене - Проста ключова дума (LIKE, full-text): бързо, но не „по смисъл“. - Векторно търсене (pgvector): смислова близост, по-точни резултати при различни формулировки.

LLM интеграция - Локални модели vs облачни услуги. - Разходи, латентност, поверителност.

Бази/векторни решения - PostgreSQL+pgvector vs отделни векторни бази (FAISS, Milvus, Pinecone).

28 – Операции и поддръжка

Резервни копия - Бекап на PostgreSQL (pg\_dump), план за възстановяване.

Наблюдение - Метрики/логове, аларми при грешки/бавни заявки.

Ротация на логове - Конфигуриране на лог фреймуърк за размер/време.

Обновления - Версиониране, миграции (Flyway), smoke тестове след деплой.

29 – UX и достъпност

UX принципи - Проста навигация: Ask, Upload, History. - Ясни съобщения за грешки/успех.

Достъпност - Контраст и четимост (Bootstrap 5). - Атрибути за навигация с клавиатура и ARIA (при нужда).

Подобрения (идеи) - Подсказки за формулиране на въпроси. - Визуализиране на откъсите, които са послужили за отговор.

30 – Етика и ограничения

Етични аспекти - Прозрачност: показване на източниците/пасажите от документа. - Поверителност: чувствителни документи да се пазят и обработват сигурно.

Ограничения - Зависи от качеството на документа и chunk-ването. - Embedding без външен LLM може да е по-малко точен за сложни случаи.

31 – Приложение: Кодови откъси

SecurityConfig (извадка)

// src/main/java/com/example/Document\_analiser/config/SecurityConfig.java  
http  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers(  
 "/auth/\*\*","/","/favicon.ico",  
 "/v3/api-docs/\*\*","/swagger-ui/\*\*",  
 "/css/\*\*","/js/\*\*","/images/\*\*"  
 ).permitAll()  
 .anyRequest().authenticated()  
 )  
 .formLogin(Customizer.withDefaults())  
 .logout(Customizer.withDefaults());

QuestionRepository – история (JPQL)

// src/main/java/com/example/Document\_analiser/repository/QuestionRepository.java  
@Query("SELECT q FROM Question q " +  
 "LEFT JOIN q.answer a " +  
 "LEFT JOIN q.document d " +  
 "WHERE q.user.username = :username AND (:documentId IS NULL OR d.id = :documentId)")  
Page<Question> findHistory(@Param("username") String username,  
 @Param("documentId") Long documentId,  
 Pageable pageable);

VectorSearchService – търсене по сходство

// src/main/java/com/example/Document\_analiser/service/VectorSearchService.java  
@Transactional(propagation = Propagation.REQUIRES\_NEW, readOnly = true)  
public List<DocumentChunk> findTopByCosineSimilarity(float[] embedding, Long documentId, int limit) {  
 var ids = documentChunkRepository.findTopIdsByCosineSimilarity(toPgVectorLiteral(embedding), documentId, limit);  
 if (ids == null || ids.isEmpty()) return java.util.Collections.emptyList();  
 var views = documentChunkRepository.findContentByIdIn(ids);  
 // ... мапване към леки DocumentChunk обекти  
}

DocumentChunkRepository – native SQL с pgvector

// src/main/java/com/example/Document\_analiser/repository/DocumentChunkRepository.java  
@Query(value = """  
 SELECT id FROM document\_chunks  
 WHERE document\_id = ?2  
 ORDER BY embedding <=> CAST(?1 AS vector(1536))  
 LIMIT ?3  
 """, nativeQuery = true)  
List<Long> findTopIdsByCosineSimilarity(String embedding, Long documentId, int limit);

QuestionService – история към DTO

// src/main/java/com/example/Document\_analiser/service/QuestionService.java  
public Page<QuestionHistoryDto> getHistory(String username, Long documentId, int page, int size, boolean ascending) {  
 Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC, "askedAt");  
 Pageable pageable = PageRequest.of(page, size, sort);  
 var questionsPage = questionRepository.findHistory(username, documentId, pageable);  
 return questionsPage.map(q -> {  
 var dto = new QuestionHistoryDto();  
 dto.setQuestionText(q.getText());  
 dto.setAskedAt(q.getAskedAt());  
 dto.setDocumentId(q.getDocument() != null ? q.getDocument().getId() : null);  
 dto.setDocumentName(q.getDocument() != null ? q.getDocument().getName() : null);  
 dto.setTopic(q.getTopic());  
 if (q.getAnswer() != null) {  
 dto.setAnswerText(q.getAnswer().getText());  
 dto.setAnsweredAt(q.getAnswer().getGeneratedAt());  
 }  
 return dto;  
 });  
}

Thymeleaf – безопасен „See more“ (history.html)

<!-- src/main/resources/templates/history.html (извадка) -->  
<span th:id="${'ans'+qStat.count}"  
 th:text="${q.answerText != null ? (#strings.length(q.answerText) > 100 ? #strings.substring(q.answerText,0,100)+'...' : q.answerText) : 'No answer'}"></span>  
<button class="btn btn-link p-0" type="button"  
 th:if="${q.answerText != null and #strings.length(q.answerText) > 100}"  
 th:attr="data-ans=${q.answerText}"  
 th:onclick="|document.getElementById('ans'+${qStat.count}).innerText=this.dataset.ans; this.remove();|">See more</button>

Заключителни бележки

Благодаря Ви за отделеното време и внимание при запознаването с тази документация. Реалното съдържание приключва с раздел 15, но поради формалното изискване за минимален обем от 40 страници, бях принуден да включа и допълнителна информация, която не е от съществено значение за проекта.

Ако смятате, че настоящият проект има потенциал за реално приложение или развитие, ще се радвам да се свържете с мен. За мен би било чест и удоволствие да видя, че идеята намира подкрепа и признание извън академичната среда.