CHAT 3GPP

27 czerwca 2025

Dokumentacja

Wykonawcy:

Mikołaj Kołodziejak	Product Owner
Bartłomiej Sawicki	Scrum Master
Krystian Kubica	Deweloper
Rafał Hresiukiewicz	Deweloper xd

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Spis treści

1 Tematyka projektu

Projekt powstał w celu ułatwienia pracy inżynierom. Wielu inzynierów zmaga się z problemem czytania i przeszukiwania norm 3GPP, w związku z czym dostaliśmy zadanie, aby stworzyć chat, który rozwiązuje tę niedogodność i odpowiada na pytania odnośnie tychże norm. Projekt był realizowany we współpracy z firmą Samsung R&D Institute Poland. Naszym zadaniem było skupienie się na normie LTE RCC, z możliwością dalszego rozwijania projektu na większą ilość norm.

2 Koncepcja rozwiązania

Rozwiązanie które wypracowaliśmy w trakcie rozwoju projektu, zakłada stworzenie servera oraz aplikacji webowej. Server ma za zadanie tworzyć bazy wektorowe z norm, a następnie wyszukiwać fragmenty, które zostaną przesłane do modelu językowego. Następnym kluczowym elementem jest wyświetlanie użytkownikowi, fragmentów normy wysłanych do modelu, jak i kluczowych miesjc w tym fragmencie które były najistotniejsze przy tworzeniu odpowiedzi.

Nasze rozwiązanie opiera się o backend stworzony z wykorzystnaiem framework Spring Boot oraz frontendu stworzonego w React oraz TypeScript. Projekt został zrealizowany tak, żeby był jak najbardziej skalowalny oraz gotowy do dalszego rozwoju.

3 Opis kluczowych fragmentów

Poniżej opisujemy ogólną koncepcję kluczowych rozwiązań zastosowanych w projekcie

Pobieranie informacji o dostępnych normach

Informacje o dostępnych normach są pobierane przy każdym starcie servera. Zdecydowaliśmy się na takie rozwiązanie, żeby automatyczne dodanie normy na oficjalnej stronie 3GPP powodowało automatyczne wyświetlanie tej normy u nas w aplikacji. Dzięki temu nie trzeba się martwić o nieaktualną listę norm i jej aktualizację.

Wyświetlanie norm

Normy są wyświetlane użytkownikowi w postaci drzewa Release -> Series -> Norm. Rozwiązanie zostało uznane za intuicyjne dla użytkownika obeznanego ze strukturą norm 3GPP.

Tworzenie baz

Użytkownik może wybrać każdą normę z drzewa i stworzyć dla niej bazę. Takowa baza jest tworzona po stronie backendu i jest trwale zapisywana na serwerze.

Procedura tworzenia bazy

Pobranie normy ze strony 3GPP

Norma jest pobierana z oficjalnej strony 3GPP, następnie jest rozpakowywana oraz zapisywana na naszym serwerze.

Ekstraktowanie zdjęć z normy

Kopia normy zapisanej w poprzednim kroku jest rozpakowana, (korzystamy z faktu, że docx to tak naprawdę archiwum ZIP) zdjęcia z niej są usuwane oraz zastępowane tak zwanymi photo code. Obrazy zapisywane są w katalogu na serwerze, a każdy photo kod zawiera informację o nazwie pliku, pod którą jest on zapisany.

Photo code, umożliwiają modelowi zdecydowanie czy dane zdjęcie należy dołączyć do sprawozdania baz przetwarzania i analizowania obrazu. (Korzystamy z faktu specyficznej, wyjątkowo strukturalnej budowy normy, pod każdym obrazkiem znajduje się opis oraz to, że obrazki nie znajdują się w losowych miejscach).

Robienie markdown

Norma z docx jest zamieniana na markdown, tabele są zamieniane na tabele HTML, usuwane są wszystkie znaki specjalne word. Nagłówki docx są zamieniane na nagłówki markdown.

```
#   - nagłówek pierwszego poziomu
##  - nagłówek drugiego poziomu
### - nagłówek trzeciego poziomu
```

Chunk Chunk

Tekst jest dzielony na fragmenty po N znaków.

Embedding chunków

Chunki są embeddowane.

Baza hybrydowa

Z zaembedowanych chunków jest robiona baza hybrydowa

Przetwarzanie zapytań

- Wyszukanie najbardziej pasujących chunków w bazie hybrydowej
- Obudowanie pytania oraz kontekstu w prompt z instrukcjami dla chatu
- Wysłanie pytania do modelu AI
- Zapisanie odpowiedzi na serwerze, zapisanie chunków użytych jako kontekst
- Owinięcie odpowiedź pierwszego modelu oraz kontekstu, specjalnym promptem który informuje, że jest on używany tylko do wskazania fragmentów w kontekście kluczowych dla geneorowania odpowiedź
- Wysłanie zapytania do drugiego modelu AI
- Sprawdzenie czy model odpowiedział poprawnie (odpowiednia struktura), jeżeli nie ponowienie (do 5 razy)
- Zapisanie highlight'ów w bazie danych
- Wysłanie odpowiedzi do użytkownika

PythonSererModel

Klasa abstrakcyjna po której dziedziczą, klasy które działają jako Springowe połączenie z mikrousługami pythona. Zapomocą dziedziczenia po PythonSererModel, klasa automatycznie uruchamia mikrousługę, gdy zlecenie pojawi się w kolejce, automatycznie za pomocą REST API przesyła wytyczne do zadania, automatycznie odbiera i wywołuje metodę publishResult, którą należy nadpisać w klasie pochodnej. Zapewnia to skalowalność oraz łatwość dodawania kolejnych mikrousług. Zdecydowaliśmy się na wykorzystanie mikrousług ze względu na większą dostępność bibliotek, w języku Python np. do uruchamiania modeli lokalnie oraz robienia embeddingów.

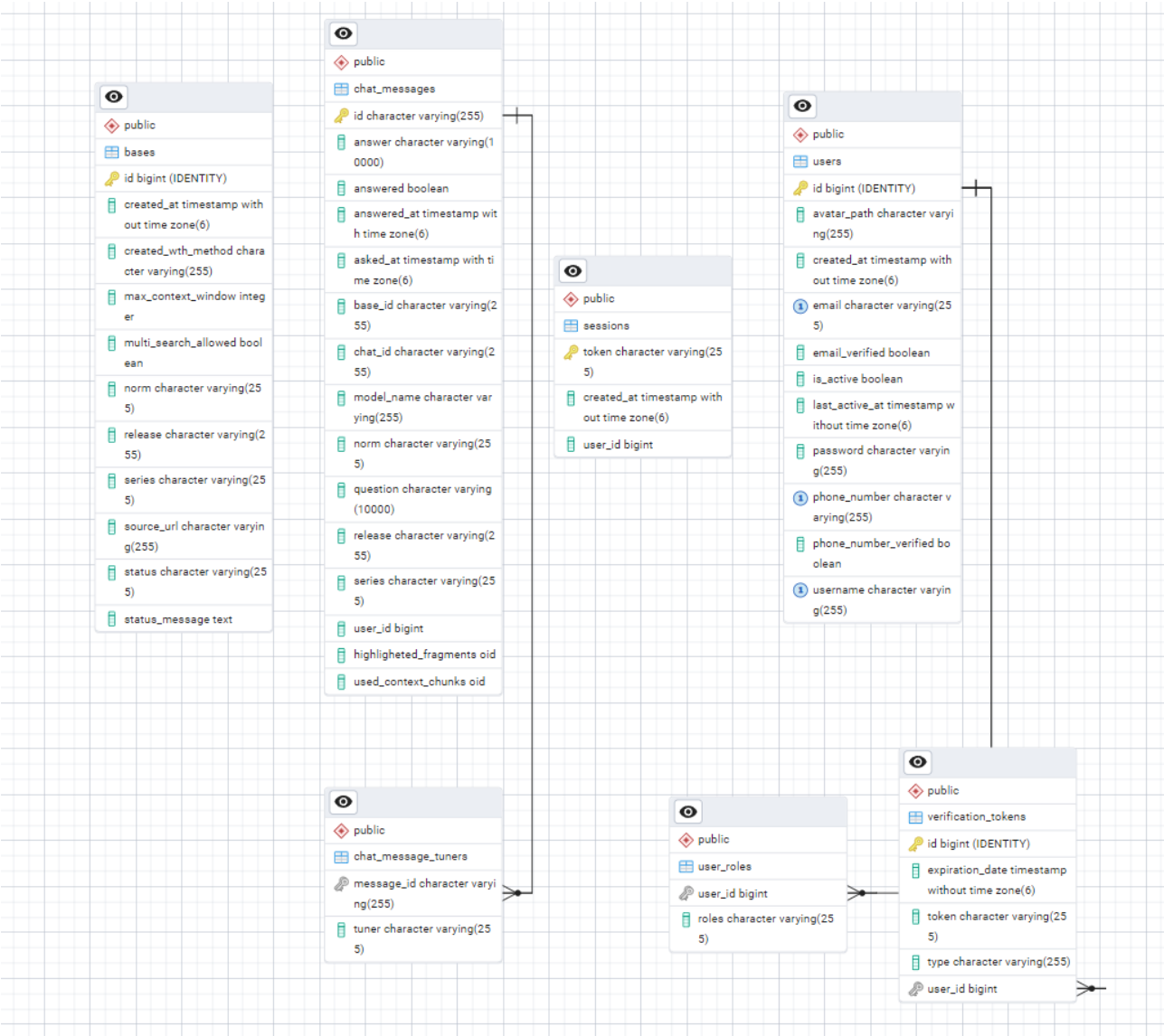
Koncepcja Wyświetlania użytkownikowi kontekstu na podstawie którego powstała odpowiedź

Użytkownik przy każdej wiadomości, na którą model udzielił odpowiedź widzi przycisk Spect Context. Po kliknięciu na ten przycisk wyświetla mu się Modal, na którym są dwie opcje.
Pierwsza umożliwia zobaczenie samych chunków na podstawie których odpowiedź została wygenerowana.
Druga umożwliwia zobaczenie całej normy, na której chunki użyte jako konetkst są wyszeczgólnione kolorem, oraz highlighty (fragmenty kluczowe dla powstania odpowiedź) są również wyróżnione kolorem.
W drugiej opcji na modalu znajduje się z kontener wymienionymi wszystkimi chunkami oraz highlight's, po kliknięciu na każdych z tych elementów, uzytkownik jest automatycznie przenoszony do danego miejsca w normie. Ratuje go to przed wielogodzinnym przeszukiwaniem długich norm.

4 Diagramy pomagające zrozumieć działanie architektury projektu

Struktura została zaprojektowana z wykorzystaniem JPA (Hibernate) i jest automatycznie synchronizowana z PostgreSQL.

4.1 Baza danych, PostgradeSQL



Rysunek 1: Diagram ERD bazy danych projektu

- **users** – tabela główna zawierająca dane użytkowników: e-mail, login, hasło, ścieżkę do avatara, statusy weryfikacji, daty aktywności i rejestracji. Każdy użytkownik może mieć przypisane role.
- **user_roles** – powiązanie wielu ról do jednego użytkownika (relacja N:1), np. rola `ADMIN`, `USER`. Klucz główny: połączenie `user_id` + `roles`. Pole stworzone z myślą o przyszłym rozdzieleniu ról admin oraz user.
- **verification_tokens** – przechowuje tokeny służące do weryfikacji konta użytkownika. Zawiera datę wygaśnięcia i typ tokena.
- **sessions** – reprezentuje aktywne sesje użytkowników (po zalogowaniu). Powiązane z użytkownikiem przez `user_id`.
- **bases** – zawiera metadane dotyczące baz utworzonych dla poszczególnych norm (3GPP). Zawiera m.in. informacje o metodzie tworzenia bazy, serii, release i statusie przetwarzania.
- **chat_messages** – kluczowa tabela przechowująca wiadomości (pytania i odpowiedzi) wprowadzane do chatu. Powiązana z użytkownikiem, normą i zawiera pełną historię wiadomości, model, fragmenty użyte w odpowiedzi, kluczowe fragmenty podczas generowania odpowiedź.
- **chat_message_tuners** – tabela asocjacyjna przechowująca informacje o tunerach przypisanych do wiadomości (np. tunery to wariację tworzenia bazy, można je włączać i wyłączać).

Relacje są zgodne z wymaganiami JPA i są odwzorowane poprzez adnotacje Hibernate w kodzie Kotlin.

5 Instrukcja rozwoju

5.1 Jak pracować w dev

W celu uruchomienia aplikacji w trybie developerskim (na czas rozwoju aplikacji) należy dla frontend'u przejść do katalogu 5GBemowoFrontend odtworzyć plik README.md oraz wykonać wszystkie kroki w nim opisane. Dla backend'u przejść do katalogu 5GBemowoBackend, odtworzyć plik README.txt, a następnie wykonać wszystkie czynności w nim opisane.

5.2 Jak ładować to do Docker?

Aby uruchomić cały projekt w kontenerach Docker (frontend, backend i baza danych PostgreSQL), należy wykonać następujące kroki:

> **Uwaga:** Upewnij się, że przed wykonaniem poniższych kroków przeczytałeś dokładnie punkt 3b w pliku `README.txt` w katalogu `5GBemowoBackend`, ponieważ wymaga on ręcznego utworzenia dwóch plików konfiguracyjnych z danymi środowiskowymi.

1. Upewnij się, że masz zainstalowane:
   - Docker: https://www.docker.com/
2. Przejdź w terminalu do katalogu głównego projektu:

   ```
   cd 5GBemowoMerged
   ```

3. Wykonaj polecenie budujące i uruchamiające aplikację:

   ```
   docker compose up --build -d
   ```

4. Po uruchomieniu:
   - Frontend będzie dostępny pod adresem: http://localhost:8080
5. Aby zatrzymać kontenery:

```
docker compose down
```

5.3 Jak wdrożyć projekt na zdalnym serwerze (deploy.sh)

Aby wdrożyć cały projekt (frontend, backend, baza danych) na zdalnym serwerze, wykonaj:

1. Skonfiguruj plik `deploy.sh`:
   - `SERVER_IP` – adres serwera
   - `SERVER_PORT` – port SSH
   - `SERVER_USER` – użytkownik z uprawnieniami (np. root)
2. Upewnij się, że masz uprawnienia do uruchamiania skryptu:

   ```
   chmod +x deploy.sh
   ```

3. Uruchom skrypt:

   ```
   ./deploy.sh
   ```

Skrypt automatycznie:

- pakuje projekt do archiwum,
- przesyła go na serwer,
- rozpakowuje i nadpisuje starą wersję,
- uruchamia kontenery Dockera.

6 Koncepcje rozwoju

Nasz koncepcja rozwoju projektu, zakłada jego stopniowe ulepszenie. Zastanawiam się w jakim kierunku chcemy rozwijać projekt, identyfikujemy problemy, planujemy zmiany na dany okres. Po zaimplementowaniu zmian, powtarzamy proces. Planujemy również dalsze zmiany, jednak one są obdarzone większym prawdopodobieństwem bycia zmienionymi w przyszłości.

6.1 Okres rozwoju projektu (4 tyg po 27.06.25)

1. ChunkyChunkerze Tabele HTML mogą zostać przecięte w środku (pomiędzy tagami <table> oraz </table>) w wyniku czego poprawne wyświetlenie tabel w frontend powoduje błędy.
Nie da się w pełni zredukować problemu przecinania tabel ponieważ ich długość czasem wielokrotnie przekracza maksymlaną długość chunki.
W tym etapie trzeba zadbać, żeby tabele były rozdzielane na mniejsze tabele, poprawnie sformatowane i każda z nich musi zawierać nazwy kolumn.

Przykład:

Tabela 1: Oryginalna tabela HTML

ID	Nazwa	Wartość
1	A	10
2	B	20
3	C	30
4	D	40
5	E	50
6	F	60
7	G	70
8	H	80

Po podziale przez ChunkyChunkera:

Tabela 2: Tabela rozcięta numer jeden

ID	Nazwa	Wartość
1	A	10
2	B	20
3	C	30
4	D	40

Tabela 3: Tabela rozcięta numer dwa

ID	Nazwa	Wartość
5	E	50
6	F	60
7	G	70
8	H	80

2. Photo extraction Photo extraction dostaje czasem błąd rozpakowywania normy jako ZIP. Nie występuje nigdy dla Release 18, Series 36, norma 36331. Błąd związany z inną wewnętrzną strukturą docx.

3. Optymalizacja embeddingów Aktualnie embeddingi są wykonywane pojedynczo, nawet jeżeli proces obciąża minimalną ilość dostępnych zasobów. Trzeba przebudować kod odpowiadający za robienie embeddingów, tak żeby zawsze był wykorzystywany pełen potencjał zasobów. Trzeba dodać automatyczne wykrywanie czy GPU jest dostepne, jeżeli tak embeddingi mają się wykonywać na GPU, w przeciwnym przypadku zostajemy przy CPU.

4. Optymalizacja mini-serwerów Python Po stronie Springa trzeba poprawić mechanizm wyłączania serverów, dodać ścieżki awaryjne co gdy nie uda się włączyć serwera. Po stronie pythona należy zastąpić Flask WSGI.

5. Server Boot Istniała koncepcja (aktualnie widoczna w kodzie) zgodnie z którą mini-servery Pythona miały się automatycznie wyłączać po ustawieniu parametru autoClose na true. Obecnie wydaje się to być rozwiązaniem absurdalnym, użytkownicy zbyt często zadają pytania, żeby obciążać ich trzema dodatkowymi sekundami czekania na odpowiedź. Ponadto, gdy autoClose = false, server jest uruchamiany wraz z pierwszym zapytaniem do niego kierowanym. Należy przyjmować, że gdy server jest włączony zapytania wpłyną do niego odrazu. Należy zmienić mechanizm uruchamiania mini-serverów, na taki, który uruchomi je wszystkie wraz z uruchomieniem servera. Ponadto trzeba stworzyć Bean, który będzie przechowywał informacje o uruchomieniu servera. W przyszłości w frontend będzie można wyświetlić na jego podstawie informację, czy server jest osiągalny, jeżeli tak to czy jest gotowy do pracy. W przyszłości warto pomyśleć o zablokowaniu przyjmowania zapytań do servera gdy nie jest on w pełni gotowy.

Aktualne procesy po uruchomieniu serwera

- Pobieranie norm

Planowane po poprawach

- Pobieranie norm
- Uruchamianie wszystkich mini-serwerów Pythona
- Sprawdzenie osiągalności, połączneie Web Socket'ów do modeli językowych
- *W przypadku na przejście na modele lokalne, połączenie Web Socketu ewentualne uruchominie*
- *Ewentualne sprawdzenie czy dla każdego rekordu bazy danych PostgreSQL, istnieją pliki bazy, zdjęcia i wszystkie potrzebne zasoby, czy działają prawidłowo (wyszukanie czegoś w bazie)*
- Zmiana stanu servera z BOOT, na FAILED albo ACTIVE

5. Poprawa MessageService Optymalizacja kodu klasy pod kątem dalszego rozwoju, zredukowanie dwóch bardzo podobnych funkcji robiących prawie dokładnie to samo.

6. UsedChunks Aktualnie w encji MessageEntity znajduje się pole usedContextChunks będące listą obiektów UsedChunk.

Listing 1: Pole `usedContextChunks` z adnotacjami JPA

```
@Lob
@Column(nullable = false)
@Convert(converter = UsedContextChunksConverter::class)
val usedContextChunks: MutableList<UsedChunk> = mutableListOf(),
```

Listing 2: Klasa danych `UsedChunk`

```
data class UsedChunk @JsonCreator constructor(
    @JsonProperty("text") val text: String,
    @JsonProperty("index") val index: Int
)
```

Nie ma sensu zapisywać pola text UsedChunk. Taka konstrukcja miała uławtić dane w frontend, jednak okazuje się kompletnie zbędna jako, że frontend i tak ma dostęp do wszystkich chunków i może je wyświetlać na podstawie indeksu. Klasę UsedChunks można zatem usunąć, a zmienna usedContextChunks może stać się listą Int'ów (indexów chunków).

7. Błędne wyświetlanie chunków w spect chunks W frontend, w widoku chatu, po otrzymaniu odpowidzi na zadane pytanie, każda wiadomość ma dostępny przycisk spect chunks. Po kliknięciu na ten przyciask, w modalu po mamy dwie opcje, po wybraniu chunki w zakładce only chunks, wyświetlane są chunki w postaci jaką dostaje model, zamiast w tej w której ma widzieć ją użytkownik. Należy wyświetlić chunki z indeksu chunki, zamiast z UsedChunk.text.

8. Poprawić logowanie użytkowników Poprawić wyświetlanie czerwonych informacji o nieprawidłowościach, dodać opcję odzyskiwania hasła.

9. Dodać obsługę profilu użytkownika absolutnie kluczowe

10. Poprawić rozłączający się web-socket podczas pisania z chatem. WebSocket pomiędzy frontendem, a backendem rozłącza się w wyniku czego użytkownik czeka na wiadomość tak długo jak nie odświeży strony.

11. Odświeżenie strony powoduje 404 Naprawić błąd występujący tylko w docker, po odświeżeniu strony, trzeba od nowa wpisywać bazowy url w przeglądarkę. Trzeba dodać konfigurację Spring.

6.2 Następny rozwoju (wstępny plan)

Rozszerznie funkcjonalności baz Należy dostosować tworzenie baz tak, żeby istniała możliwość zrobienia bazy z dowolnej kombinacji baz.

Stałe zapisywanie baz Dodanie do boot'a servera mechanizmu pobierającego bazy danych z repozytorium. Ułatwi to dalszy rozwój, po zmienie miejsca hoostownaia servera nie będzie trzeba od nowa tworzyć baz, zostaną one automatycznie pobrane z repozytorium.

Należy dodać możliwość zrobienia bazy z własnego dokumnetu Użytkownik powinien mieć możliwość dodania dowolnego pliku i zrobienia z niego bazy

6.3 Dalsze perspektywy

Po zakończeniu powyższych etapów rozwoju, należy obrać specjalizację w których nasz projekt ma największy potencjał biznesowy oraz dostosować projekt indywidualnie do tych gałęzi i wprowadzić na rynek. Warto rozważyć wykorzystanie projektu jako chatu AI dla inżynierów, chatu do nauki Prawa, chatu do nauki norm lotniczych (przedewszystkim Type Rating, podobnym problem jak z normami 3GPP, aktualnie dostępne chaty AI, często odpowiadają na podstawie dokumentu dotyczącego innej maszyny). W każdej z wyżej wymienionych specjalizacji, nasze rozwiązanie ma potencjał, który wynika przedewszystkim z faktu, że nasz chat ma dokładnie zawężony obszar poszukiwań w wyniku czego unikamy generowania dla użytkownika odpowiedzi dotyczących nieobowiązujących już przepisów, albo nie mających zastosowania w danym przypadku. Dodatkową zaletą jest wskazanie użytkownikowi, z jakiego dokładnie fragmentu pochodzi odpowiedź.

Planujemy wprowadzić rozwiązanie na rynek, a następnie szukać podmiotu, który jest w stanie finansować dalszy rozwój projektu w zamian za przyszły udział w jego zyskach.

7 Listę wykorzystanych bibliotek

Pełna lista wykorzystany bibliotek na dzień 27.06.25 r. znajduje się w index.html dla backend, oraz licenses.json dla frontendu. W katalogu przesyłanym razem ze sprawozdaniem.