Dokumentacja techniczna aplikacji Teletext

Sebastian Górski
Jakub Grzymisławski
Łukasz Szenkiel
Rafał Wilczewski

Spis treści

Opis

Dokumentacja dotyczy aplikacji *Teletext*, nowoczesnego systemu zarządzania telegazetą. Aplikacja została zaprojektowana z myślą o ułatwieniu tworzenia, edycji i publikacji treści telegazety w sposób szybki, intuicyjny i zgodny ze współczesnymi standardami cyfrowymi.

System umożliwia użytkownikom zarówno przeglądanie dostępnych stron telegazety, jak i zarządzanie jej zawartością w czasie rzeczywistym. Względem aplikacji postawiono następujące wymagania:

- użytkownik może przeglądać strony telegazety (struktura numerów stron!)
- użytkownik może wyszukiwać informacje po tytułach i kategoriach administrator może tworzyć własne strony telegazety
- strona ma numer, tytuł, kategorię i treść
- treść może zawierać tekst i proste elementy graficzne (ASCII) administrator może przypisywać
- wybrane integracje do numerów stron
- należy zaimplementować min. 7 integracji, np. pogoda, wyniki lotto, głosowania w Sejmie, ogłoszenia o pracę, kursy walut, ceny kruszców
- administrator widzi statystyki najczęściej odwiedzanych stron

Celem projektu jest stworzenie narzędzia, które nie tylko ułatwi pracę redaktorom i administratorom telegazety, ale również zapewni użytkownikom końcowym przyjazny i funkcjonalny interfejs do przeglądania informacji.

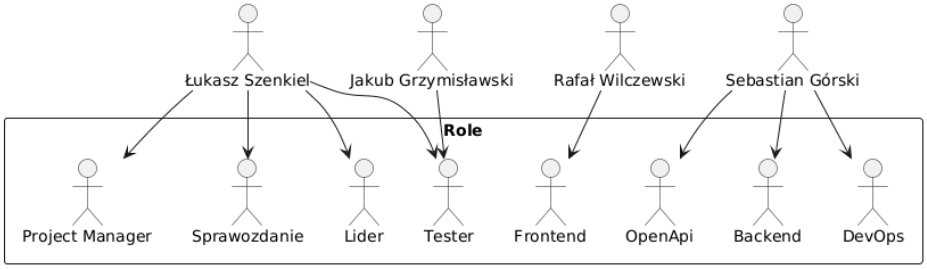
1 Zespół

1.1 Osoby w zespole

Zespół tworzą:

- Sebastian Górski
- Jakub Grzymisławski
- Łukasz Szenkiel
- Rafał Wilczewski

1.2 Role w zespole



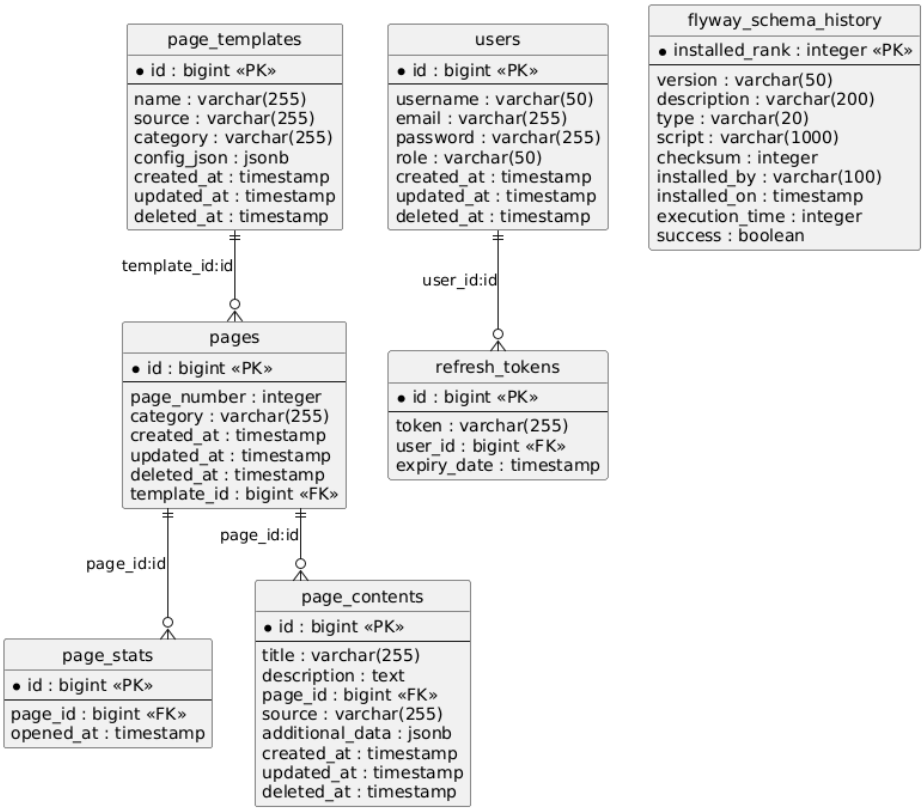
Rysunek 1: Podział ról w zespole

2 Architektura systemu

W tej sekcji przedstawiono opis architektury systemu *Teletext*.

2.1 Diagram encji

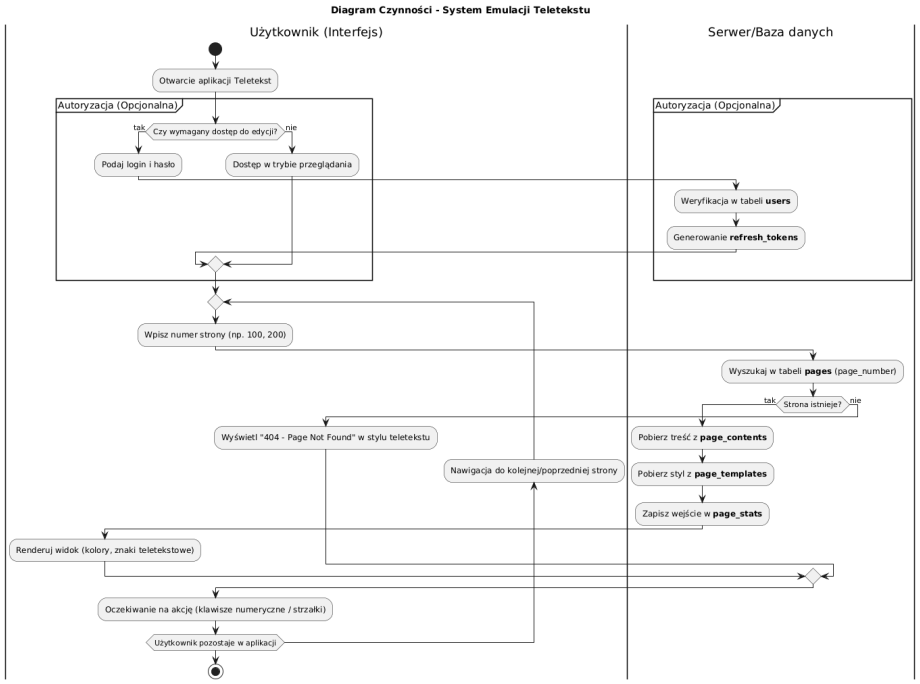
Diagram encji przedstawia strukturę bazy danych systemu telegazety.



Rysunek 2: Diagram encji

2.2 Diagram czynności

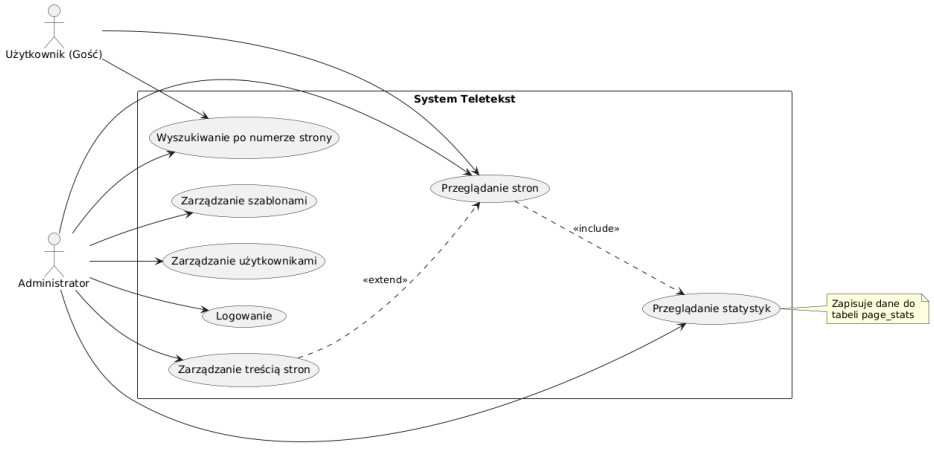
Diagram czynności przedstawia logikę biznesową systemu oraz przepływ sterowania podczas interakcji użytkownika z aplikacją. Ilustruje on procesy takie jak autoryzacja dostępu, nawigacja pomiędzy stronami serwisu na podstawie ich numeracji oraz mechanizm rejestrowania statystyk wyświetleń w bazie danych.



Rysunek 3: Diagram czynności

2.3 Diagram przypadków użycia

Diagram przypadków użycia ukazuje interakcje użytkowników z systemem, identyfikując główne funkcje dostępne w aplikacji.



Rysunek 4: Diagram przypadków użycia

2.4 Implementacja kodu

Aplikacja *Teletext* została zaimplementowana z wykorzystaniem frameworka Spring Boot, który ułatwia tworzenie aplikacji webowych w języku Java. Projekt został zorganizowany zgodnie z ar-

chitekturą MVC (Model-View-Controller), co zapewnia czytelność i separację odpowiedzialności w kodzie.

2.4.1 Stos technologiczny

- **Backend**: Java 21, Spring Boot 3.5.6
- **Frontend**: HTML5, CSS, React 19
- **Baza danych**: PostgreSQL (główny magazyn danych)
- **Cache**: Redis 7.4.2-alpine (przechowywanie danych tymczasowych i sesyjnych)
- **Zarządzanie migracjami**: Flyway 10.20.1 (wersjonowanie schematu bazy)
- **Serwer proxy**: nginx 1.29

2.4.2 Obsługa błędów

System posiada mechanizmy obsługi wyjątków

2.4.3 Przykładowe listingi kodu w języku programowania Java

```
@RestController
@RequestMapping("/api/admin/pages")
@RequiredArgsConstructor
@Tag(
name = "Admin Teletext Pages",
description = "Endpoints for managing teletext pages in the admin panel")
public class AdminTeletextPageController {

private final TeletextPageService pageService;

@GetMapping
@Operation(
summary = "Get all teletext pages",
description =
"Returns a list of teletext pages. Optionally filter by category and
   include inactive pages.")
public ResponseEntity<List<TeletextPageResponse>> getAllPages(
@RequestParam(required = false) TeletextCategory category,
@RequestParam(defaultValue = "false") boolean includeInactive) {
var results = pageService.getAllPages(category, includeInactive);
return ResponseEntity.ok(results);
}

@GetMapping("{id}")
@Operation(
summary = "Get teletext page by ID",
description =
```

```
"Returns a single teletext page by its ID, including its content. Shows even inactive pages.")
public ResponseEntity<TeletextAdminPageResponse> getPageById(@PathVariable long id) {
var result = pageService.getPageWithContentById(id);
return ResponseEntity.ok(result);
}

@PostMapping
@Operation(
summary = "Create a new teletext page",
description =
"Creates a new teletext page based on the provided data. Must declare type of the page (MANUAL or TEMPLATE).")
public ResponseEntity<Void> createPage(@RequestBody @Valid PageCreateRequest request) {
var result = pageService.createPage(request);
var uri = URI.create("/api/admin/pages/" + result.id());
return ResponseEntity.created(uri).build();
}

@PutMapping("{id}")
@Operation(
summary = "Update an existing teletext page",
description =
"Updates the details of an existing teletext page identified by its ID. Must declare type of the page (MANUAL or TEMPLATE).")
public ResponseEntity<TeletextAdminPageResponse> updatePage(
@PathVariable Long id, @RequestBody @Valid PageUpdateRequest request) {
var result = pageService.updatePage(id, request);
return ResponseEntity.ok(result);
}

@PatchMapping("{id}/activate")
@Operation(
summary = "Activate a teletext page",
description =
"Activates a teletext page identified by its ID. Can activate only previously deactivated pages.")
public ResponseEntity<Void> activatePage(@PathVariable Long id) {
pageService.activatePage(id);
return ResponseEntity.noContent().build();
}

@DeleteMapping("{id}")
@Operation(
summary = "Deactivate a teletext page",
description =
"Deactivates a teletext page identified by its ID. The page will no longer be visible in public API.")
```

```
public ResponseEntity<Void> deletePage(@PathVariable Long id) {
pageService.deactivatePage(id);
return ResponseEntity.noContent().build();
}
}
```

Listing 1: AdminTeletextPageController - kontroler do zarządzania stronami telegazety w panelu admina

```
@Entity
@Table(name = "pages")
@Data
public class TeletextPage {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(nullable = false)
private Integer pageNumber;

@Enumerated(EnumType.STRING)
@Column(nullable = false)
private TeletextCategory category;

@OneToOne(mappedBy = "page", cascade = CascadeType.ALL, orphanRemoval =
    true)
private TeletextPageContent content;

@OneToMany(mappedBy = "page", cascade = CascadeType.ALL, orphanRemoval =
    true)
private List<TeletextPageStats> stats;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "template_id")
private TeletextPageTemplate template;

@CreationTimestamp private Timestamp createdAt;

@UpdateTimestamp private Timestamp updatedAt;

private Timestamp deletedAt;

@PrePersist
@PreUpdate
private void validate() {
validatePageNumberRange();
validateCategory();
}

```

```
40 public String getTitle() {
41 if (this.content != null) return this.content.getTitle();
42 if (this.template != null) return this.template.getName();
43 throw new IllegalStateException(
44 "Brak tytu u strony o numerze " + this.pageNumber + ". Nieprawid owy
      stan obiektu.");
45 }
46
47 public String getType() {
48 if (this.template != null) return "TEMPLATE";
49 return "MANUAL";
50 }
51
52 private void validatePageNumberRange() {
53 int mainPage = this.category.getMainPage();
54 if (this.pageNumber < mainPage + 1 || this.pageNumber > mainPage + 99) {
55 throw new IllegalPageNumberException(
56 "Numer strony "
57 + this.pageNumber
58 + " jest poza zakresem dla kategorii "
59 + this.category.getTitle());
60 }
61 }
62
63 private void validateCategory() {
64 if (this.template != null) {
65 if (!this.category.equals(this.template.getCategory())) {
66 throw new IllegalStateException(
67 "Szablon strony nale y do innej kategorii ni  sama strona. Strona: "
68 + this.pageNumber
69 + ", kategoria strony: "
70 + this.category
71 + ", kategoria szablonu: "
72 + this.template.getCategory());
73 }
74 }
75 }
76 }
```

Listing 2: Encja TeletextPage - reprezentacja strony telegazety w bazie danych

```
1 @Mapper(
2 componentModel = "spring",
3 uses = {
4 TeletextPageMapper.class,
5 TeletextCategoryMapper.class,
6 })
7 public interface TeletextAdminPageMapper {
8
9 @Mapping(target = "type", expression = "java(page.getType())")
```

```
TeletextAdminPageResponse toResponse(TeletextPage page);

@Mapping(target = "content.title", source = "title")
@Mapping(target = "content.description", source = "description")
@Mapping(target = "content.source", constant = "MANUAL")
TeletextPage toPage(ManualPageCreateRequest request);

@Mapping(target = "template", ignore = true)
TeletextPage toPage(TemplatePageCreateRequest request);

@AfterMapping
default void linkContent(@MappingTarget TeletextPage page) {
if (page.getContent() != null) {
page.getContent().setPage(page);
}
}

@BeanMapping(nullValuePropertyMappingStrategy =
   NullValuePropertyMappingStrategy.IGNORE)
@Mapping(target = "template", ignore = true)
void updatePageFromTemplateRequest(
TemplatePageUpdateRequest request, @MappingTarget TeletextPage page);

@BeanMapping(nullValuePropertyMappingStrategy =
   NullValuePropertyMappingStrategy.IGNORE)
@Mapping(target = "content.title", source = "title")
@Mapping(target = "content.description", source = "description")
void updatePageFromManualRequest(
ManualPageUpdateRequest request, @MappingTarget TeletextPage page);
}
```

Listing 3: Mapper TeletextAdminPageMapper - konwersje między encją a DTO

3 Instrukcja uruchomieniowa

W celu uruchomienia aplikacji należy zainstalować na swoim urządzeniu następujące aplikacje: *Docker*, *GitHub CLI* oraz – w przypadku systemu Windows – aplikację umożliwiającą uruchomienie komendy *make*.

3.1 Sklonowanie repoizytorium

Należy sklonować repozytorium: `git clone https://github.com/lukaszsz1991/teletext-dev.git`

Repozytorium zawiera konfigurację infrastruktury projektu **Teletext**, w tym:

- repozytoria `teletext-backend` oraz `teletext-frontend`,
- folder `docker-files` z plikami obrazów Dockera oraz logami serwisów,
- plik `Makefile` ze skryptami pomocnicznymi,
- konfigurację serwisów Dockera w pliku `compose.yml`.

3.2 Praca z submodułami

Każdy submoduł jest osobnym repozytorium Git. Przechodząc do odpowiedniego podfolderu (`backend` lub `frontend`), pracujemy bezpośrednio w danym repozytorium. Oznacza to możliwość tworzenia gałęzi, aktualizacji kodu, wykonywania commitów oraz tworzenia Pull Requestów.

Uwaga Po wdrożeniu zmian w submodułach do gałęzi `main`, należy zaktualizować repozytorium `teletext-dev`.

```
make rebase            # (opcjonalnie)
make push-backend      # dla zmian w backendzie
make push-frontend     # dla zmian w frontendzie
```

Alternatywnie, w celu uzyskania większej kontroli (np. nad treścią komunikatu commita), można wykonać następujące kroki:

```
cd teletext-dev
make rebase
git add backend    # lub frontend, zależnie od zmian
git commit -m "chore: update submodules"
git push -u origin <branch>
```

Powyższy commit w repozytorium `teletext-dev` aktualizuje referencję submodułu do najnowszej wersji gałęzi `main`, umożliwiając wszystkim współpracownikom pobranie aktualnego stanu projektu.

3.2.1 Aktualizacja submodułów

Przed rozpoczęciem pracy zaleca się upewnienie, że wszystkie repozytoria są aktualne. W tym celu należy wykonać polecenie:

```
make rebase
```

Polecenie to uruchamia następujący skrypt:

```
git pull --rebase
git submodule update --init --recursive --remote --jobs 2
```

Wykonanie powyższych komend powoduje pobranie najnowszych wersji repozytoriów z gałęzi `main`.

3.3 Docker Compose

Projekt wykorzystuje Docker Compose do lokalnego uruchamiania wszystkich serwisów, w tym:

- bazę danych PostgreSQL (`postgres`),
- backend oparty o Spring Boot (`backend`),
- frontend oparty o React oraz Nginx (`frontend`).

3.3.1 Uruchamianie środowiska

Aby uruchomić środowisko lokalne, należy:

1. Upewnić się, że Docker jest zainstalowany w systemie.
2. Skopiować plik `.env.example` do `.env` i dostosować zmienne środowiskowe.
3. Skopiować plik `.env.webclient.example` do `.env.webclient`.
4. Uruchomić serwisy poleceniem:

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

Alternatywnie można skorzystać z polecenia:

```
make build-up
```

3.4 Makefile

Plik `Makefile` zawiera zestaw pomocnych komend usprawniających pracę nad projektem. Tabela 1 przedstawia dostępne polecenia.

Komenda	Opis
`make rebase`	Aktualizuje repozytorium i submoduły
`make push-backend`	Wypycha zmiany backendu do repozytorium zdalnego
`make push-frontend`	Wypycha zmiany frontendu do repozytorium zdalnego
`make build`	Buduje obrazy Dockera dla wszystkich serwisów
`make build-up`	Buduje i uruchamia obrazy Dockera
`make up`	Uruchamia serwisy Docker Compose
`make down`	Zatrzymuje serwisy Docker Compose
`make logs`	Wyświetla logi wszystkich serwisów
`make restart`	Przebudowuje i restartuje wszystkie serwisy
`make restart-backend`	Restartuje backend aplikacji
`make restart-frontend`	Restartuje frontend aplikacji

Tablica 1: Dostępne polecenia Makefile

4 Opis użytych technologii

4.1 Struktura bazy danych

Charakterystyka ogólna

System **Teletext** wykorzystuje relacyjną bazę danych **PostgreSQL 17.2**. Wybór ten podyktowany był potrzebą zapewnienia pełnej spójności danych (ACID), wsparcia dla zaawansowanych typów tekstowych oraz łatwej integracji z frameworkiem Spring Boot poprzez Hibernate/JPA.

Struktura danych i kluczowe encje

Baza danych została podzielona na logiczne obszary odpowiedzialne za treść, administrację oraz analitykę. Poniżej znajduje się opis najważniejszych tabel:

1. Zarządzanie treścią (`pages` & `categories`)

- **Strony telegazety (`pages`):** Główna tabela systemu. Przechowuje unikalny numer strony (klucz główny), tytuł, treść tekstową oraz meta-dane. Treść wspiera znaki specjalne i grafikę ASCII.
- **Kategorie (`categories`):** Słownik pozwalający na grupowanie stron (np. Sport, Gospodarka). Każda strona jest przypisana do jednej kategorii, co optymalizuje proces wyszukiwania i filtrowania treści.

2. Automatyzacja i Integracje (`integrations`)

- **Integracje zewnętrzne:** Tabela przechowująca konfigurację dla modułów automatycznych. Definiuje ona typ integracji (np. API pogodowe, wyniki giełdowe) oraz mapuje pobrane dane na konkretne numery stron telegazety. Dzięki temu system może bezobsługowo aktualizować treści w czasie rzeczywistym.

3. Bezpieczeństwo i Uprawnienia (`users`)

- **Użytkownicy i Role:** Tabela przechowująca dane dostępowe administratorów i redaktorów. System ról (Spring Security) definiuje zakres uprawnień – od możliwości podglądu statystyk po pełną edycję struktury stron i zarządzanie integracjami.

4. Analityka i Wydajność

- **Statystyki odwiedzin** (`statistics`)**:** Rejestruje zdarzenia wyświetlenia poszczególnych stron. Dane te służą do generowania raportów popularności w panelu administratora.
- **Warstwa Cache (Redis):** Mimo że Redis nie jest bazą relacyjną, stanowi integralną część modelu danych jako magazyn typu klucz-wartość. Przechowuje on zserializowane obiekty stron, co znacząco odciąża PostgreSQL przy dużym natężeniu ruchu.

Zarządzanie schematem

Integralność struktury bazy danych jest utrzymywana przez narzędzie **Flyway 10.20.1**. Wszystkie zmiany w modelu (tworzenie tabel, dodawanie kolumn) są wprowadzane poprzez skrypty migracyjne SQL, co zapewnia identyczną strukturę bazy na każdym środowisku uruchomieniowym.

4.2 Opis środowiska

4.2.1 Baza danych - PostgreSQL 17.2

System bazodanowy wykorzystywany w projekcie to PostgreSQL w wersji 17.2. PostgreSQL to wydajny, stabilny i bezpieczny system zarządzania relacyjną bazą danych, szeroko stosowany w środowiskach produkcyjnych.
Konfiguracja dostępu do bazy danych znajduje się w pliku docker-compose.yml.

4.2.2 Migracje schematu – Flyway 10.20.1

Do zarządzania migracjami schematu bazy danych wykorzystywane jest narzędzie Flyway w wersji 10.20.1. Flyway umożliwia wersjonowanie zmian w strukturze bazy danych i ich automatyczne stosowanie w środowiskach deweloperskich, testowych oraz produkcyjnych.
Cechy użycia Flyway w projekcie:

- migracje definiowane w postaci skryptów SQL w katalogu db/migration,
- automatyczne wykrywanie i stosowanie nowych migracji przy starcie aplikacji,
- wsparcie dla rollbacków i walidacji historii migracji.

4.2.3 Cache – Redis 7.4.2-alpine

Do przechowywania danych tymczasowych oraz buforowania odpowiedzi system korzysta z Redis w wersji 7.4.2-alpine. Redis działa jako zewnętrzny, szybki magazyn danych typu key-value, wykorzystywany m.in. do cache'owania wyników zapytań i danych sesyjnych.
Wersja alpine została wybrana ze względu na minimalny rozmiar obrazu oraz szybki czas uruchamiania kontenera.

5 Integracje z zewnętrznymi serwisami

5.1 Dostępne źródła

Aplikacja integruje się z siedmioma zewnętrznymi interfejsami API, które dostarczają dane do wyświetlenia w telegazecie:

- Narodowy Bank Polski (https://api.nbp.pl/) – kursy walut,
- OpenMeteo (https://open-meteo.com/en/docs) – dane pogodowe,
- Lotto (https://developers.lotto.pl/) – dane losowań lotto,
- News Data (https://newsdata.io/) – wiadomości,
- Jooble (https://help.jooble.org/en/support/solutions/articles/60001448238-rest-api-documentation) – oferty pracy,
- Mój codzienny horoskop (https://www.moj-codzienny-horoskop.com/webmaster/api-horoskop-xml-json.htm) – horoskop,
- Highlightly (https://highlightly.net/documentation/football/) – dane piłkarskie,
- TVP (https://www.tvp.pl/prasa) – program telewizyjny TVP.

5.2 Konfiguracja web clientów

W backendzie aplikacji, w pliku `application.properties`, znajduje się wydzielona sekcja przeznaczona do konfiguracji web clientów.

Wszystkie pola konfiguracyjne muszą zostać uzupełnione, aby aplikacja działała poprawnie. Właściwości związane z web clientami posiadają prefiks `webclient` i zaleca się dostarczanie ich do aplikacji poprzez zmienne środowiskowe.

Konfiguracja web clientów obejmuje:

- czasy timeoutów,
- bazowe adresy URL,
- klucze API (jeżeli są wymagane przez dane źródło).

Wszystkie połączenia z zewnętrznymi API realizowane są asynchronicznie, co poprawia wydajność aplikacji oraz doświadczenie użytkownika. Każde źródło danych posiada skonfigurowanego osobnego klienta.

General:

- `webclient.response-timeout-ms` (`WEBCLIENT_RESPONSE_TIMEOUT_MS`) – maks. czas oczekiwania na odpowiedź (ms), domyślnie `5000`.
- `webclient.connection-timeout-ms` (`WEBCLIENT_TIMEOUT_MS`) – maks. czas zestawienia połączenia (ms), domyślnie `5000`.

Base URL:

- `webclient.nbp-base-url` (`WEBCLIENT_NBP_API_BASE_URL`) – https://api.nbp.pl/
- `webclient.open-meteo-base-url` (`WEBCLIENT_OPEN_METEO_API_BASE_URL`) – https://api.open-meteo.com/
- `webclient.lotto-base-url` (`WEBCLIENT_LOTTO_API_BASE_URL`) – https://developers.lotto.pl/
- `webclient.news-data-base-url` (`WEBCLIENT_NEWS_DATA_API_BASE_URL`) – https://newsdata.io/
- `webclient.jooble-base-url` (`WEBCLIENT_JOOBLE_API_BASE_URL`) – https://jooble.org/
- `webclient.horoscope-base-url` (`WEBCLIENT_HOROSCOPE_API_BASE_URL`) – https://www.moj-codzienny-horoskop.com/
- `webclient.highlightly-base-url` (`WEBCLIENT_HIGHLIGHTLY_API_BASE_URL`) – https://sports.highlightly.net/
- `webclient.tvp-base-url` (`WEBCLIENT_TVP_API_BASE_URL`) – https://www.tvp.pl/

Secrets (wymagane):

- `webclient.lotto-secret` (`WEBCLIENT_LOTTO_SECRET`) – klucz API Lotto
- `webclient.news-data-secret` (`WEBCLIENT_NEWS_DATA_SECRET`) – klucz API NewsData
- `webclient.jooble-secret` (`WEBCLIENT_JOOBLE_SECRET`) – klucz API Jooble
- `webclient.highlightly-secret` (`WEBCLIENT_HIGHLIGHTLY_SECRET`) – klucz API Highlightly

5.3 Format zwracanego contentu

Wszystkie integracje wykorzystują ujednolicone DTO `ExternalDataResponse`, które zawiera następujące pola:

- `source` – źródło danych (`String`),
- `title` – tytuł treści (`String`),
- `description` – główny opis treści (`String`),
- `additionalData` – dodatkowe informacje zależne od źródła (`Map<String, Object>`).

Zawartość pól `source` oraz `additionalData` zależy od konkretnej integracji. Szczegółowy opis dla poszczególnych źródeł znajduje się w załącznikach:

- Narodowy Bank Polski,
- OpenMeteo,
- Lotto,

- News Data,
- Jooble,
- Mój codzienny horoskop,
- Highlightly,
- TVP.

5.4 Obsługa błędów

W przypadku niepowodzenia pobrania danych z zewnętrznego serwisu, błąd przekazywany jest dalej wraz z oryginalnym kodem błędu źródłowego API.

Błędy prezentowane są w tym samym formacie co pozostałe błędy aplikacji, zgodnie ze standardem `ProblemDetail` dostarczanym przez framework Spring (https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ProblemDetail.html).

Spis rysunków

Spis listingów