

# MikroBiblioteka

System do archiwizowania plików



Zespół: Alesia Filinkova, Diana Pelin, Weronika Maślana



# Wymagania funkcjonalne

1 milestone (M1 z Jiry): MVP – Minimum Viable Product

The screenshot shows a Jira board with three columns: TO DO, IN PROGRESS, and DONE.

- TO DO:** Contains two items:
  - func: show a list of files with name and size (KAN-14, AF)
  - func: download file from list (KAN-16, D)
- IN PROGRESS:** No items listed.
- DONE:** Contains one item:
  - func: add file from device (KAN-12, WM)

A header at the top indicates "KAN-22 M1 (3 work items) TO DO". A plus sign icon is in the top right corner.



# Wymagania funkcjonalne

2 milestone (M2 z Jiry): Management & UI

The image shows a digital Kanban board interface for managing functional requirements. The board is divided into three columns: **TO DO**, **IN PROGRESS**, and **DONE**.

**TO DO** column (3 items):

- func: delete file  
KAN-15 (status: D)
- func: filter files by name  
KAN-19 (status: WM)
- func: show how many times a file was downloaded  
KAN-20 (status: AF)

**IN PROGRESS** column (0 items)

**DONE** column (0 items)

At the top of the board, there is a header bar with a purple lightning bolt icon, a purple square icon, the text "KAN-23 M2 (3 work items)", and a "TO DO" button.

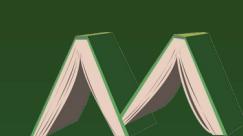
At the bottom left of the board, there is a "+ Create" button.



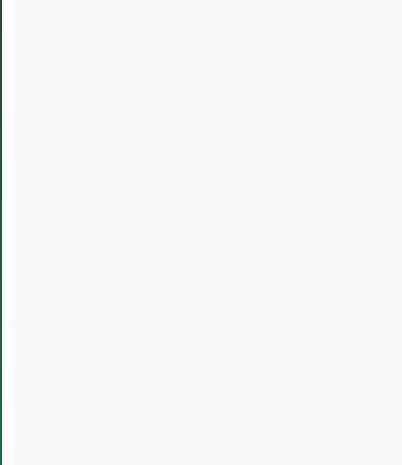
# Raport z pozostałych zadań

⚡ KAN-24 M0 (12 work items) **TO DO**

TO DO	IN PROGRESS	DONE
		12 ✓
+ Create		
		Jenkins setup <input checked="" type="checkbox"/> KAN-2 ✓ D
		make Github repo <input checked="" type="checkbox"/> KAN-7 ✓ WM
		Oracle server setup <input checked="" type="checkbox"/> KAN-11 ✓ D
		Jira setup <input checked="" type="checkbox"/> KAN-4 ✓ AF
		make app skeleton <input checked="" type="checkbox"/> KAN-5 ✓ WM
		add basic unit tests <input checked="" type="checkbox"/> KAN-8 ✓ WM
		code coverage setup <input checked="" type="checkbox"/> KAN-1 ✓ WM
		automated Deployment Pipeline <input checked="" type="checkbox"/> KAN-26 ✓ D



# Raport z pozostałych zadań



add functionalities to Jira - seperated for 2 milestones	<input checked="" type="checkbox"/> KAN-17	✓ WM
test_Jira	<input checked="" type="checkbox"/> KAN-6	✓ AF
nexus setup	<input checked="" type="checkbox"/> KAN-3	✓ AF
Maven setup	<input checked="" type="checkbox"/> KAN-25	✓ AF

▼ No Epic (3 work items)

TO DO 1

add integration tests

KAN-21

D

+ Create

IN PROGRESS 1

make a 7min presentaion - etap 2

Dec 16, 2025

KAN-9

WM

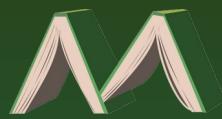
DONE 1 ✓

write technologies (stos technologiczny) and architecture documentation

Dec 16, 2025

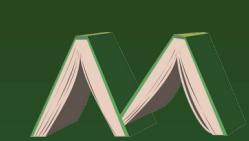
KAN-10

✓ AF WM



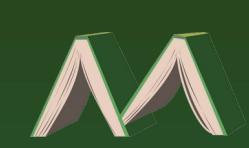
# Stos Technologiczny - opis

- Projekt MikroBiblioteka to aplikacja webowa umożliwiająca użytkownikom przesyłanie, przechowywanie, przeglądanie, pobieranie i usuwanie plików.
- System opiera się na architekturze wielowarstwowej z podziałem na frontend, backend i warstwę baz danych.
- Całość została uruchomiona w kontenerach Dockera i wdrożona na maszynie wirtualnej w chmurze Azure.
- Proces wytwarzania i testowania oprogramowania wspierają narzędzia GitHub, Jenkins, Nexus, JaCoCo oraz Jira.



# Stos Technologiczny - frontend

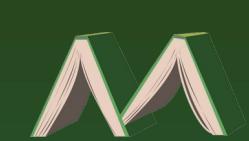
- **1. Frontend**
  - • Technologia: Angular
  - • Język: TypeScript, HTML, CSS
  - • Funkcjonalność:
    - ○ interfejs użytkownika (upload, pobieranie, przeglądanie, usuwanie plików),
    - ○ komunikacja z backendem przez REST API,
    - ○ walidacja danych po stronie klienta.



# Stos Technologiczny - backend

- **2. Backend**

- • Technologia: Spring Framework (Spring Boot)
- • Język: Java
- • Funkcjonalność:
  - ○ obsługa logiki aplikacji,
  - ○ zarządzanie metadanymi plików,
  - ○ komunikacja z bazami danych (PostgreSQL i MongoDB),
  - ○ udostępnienie REST API dla frontendu.



# Stos Technologiczny – bazy danych i infrastruktura

- **3. Baza danych**
  - ● PostgreSQL – relacyjna baza danych przechowująca metadane o plikach (np. nazwa, data dodania, rozmiar pliku).
  - ● MongoDB – nie relacyjna baza danych przechowująca same pliki (binary data).
- **4. Konteneryzacja i infrastruktura**
  - ● Docker – konteneryzacja każdej warstwy systemu (frontend, backend, PostgreSQL, MongoDB).
  - ● Docker Compose – orkiestracja czterech kontenerów i konfiguracja sieci między nimi.
  - ● Wirtualna maszyna w chmurze Azure – środowisko uruchomieniowe hostujące aplikację, narzędzia Jenkins oraz Nexus.



# Stos Technologiczny – c.d.

- **5. System kontroli wersji**
  - ● GitHub – repozytorium kodu źródłowego, zarządzanie wersjami, pull requestami i współpracą zespołową.
- **6. Zarządzanie projektem**
  - ● Jira – system do zarządzania zadaniami i iteracjami; wspiera planowanie i monitorowanie postępów w projekcie.



# Stos Technologiczny – c.d.

## 6. CI/CD, testy i repozytorium artefaktów

- ● Jenkins – narzędzie CI/CD do automatyzacji budowania, testowania i wdrażania aplikacji.
- ● JaCoCo – narzędzie zintegrowane z Jenkins do pomiaru pokrycia kodu testami w Javie.
- ● Nexus Repository Manager – repozytorium artefaktów (np. plików .jar, .war), wykorzystywane do przechowywania wersji aplikacji po procesie budowania.



# Architektura aplikacji

Diagram  
warstwowy

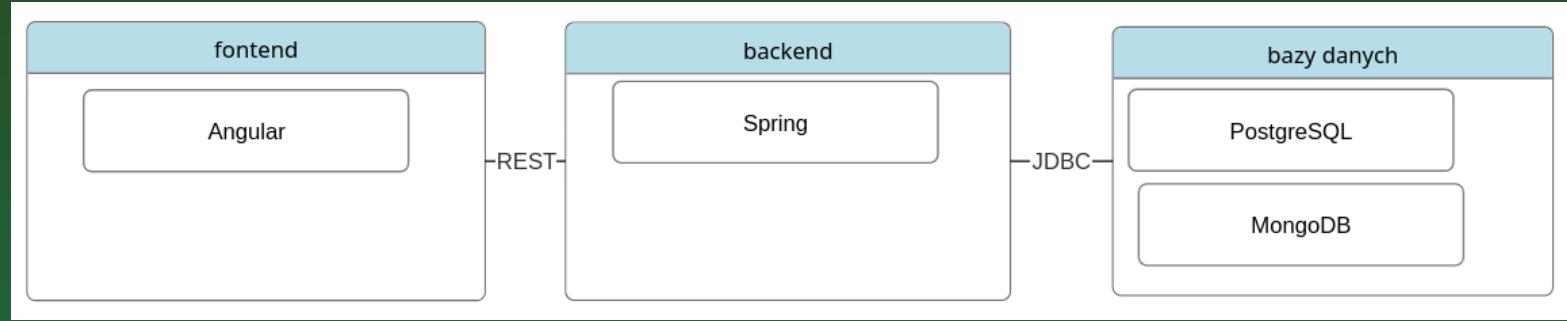
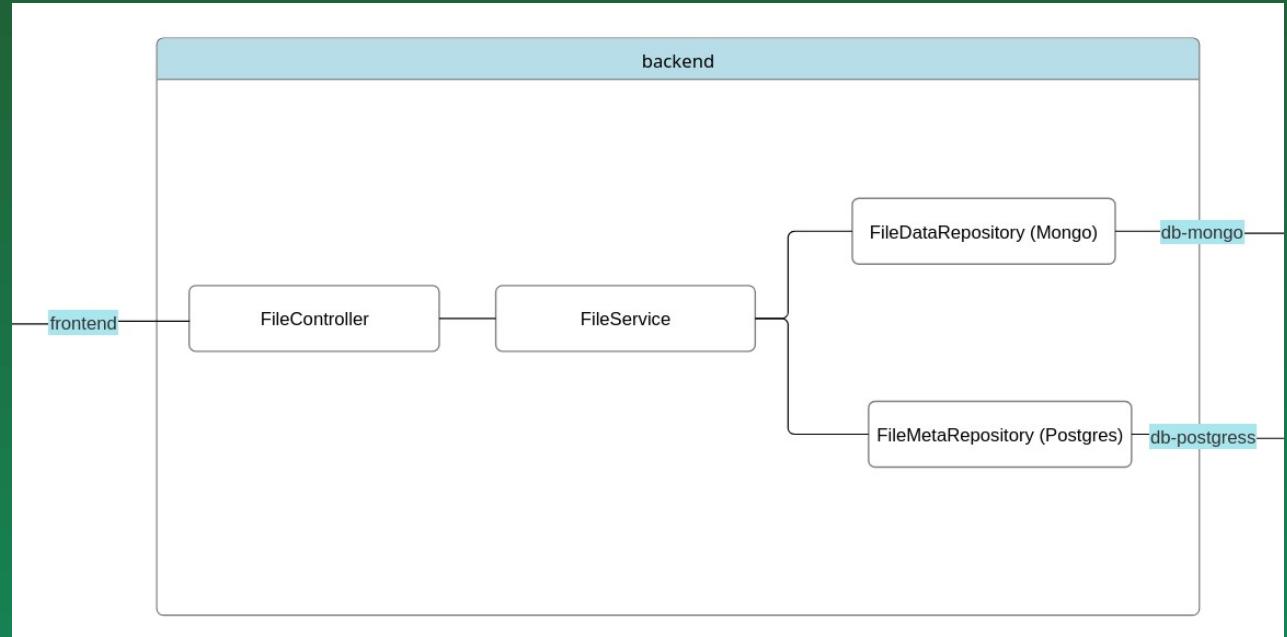


Diagram  
komponentów

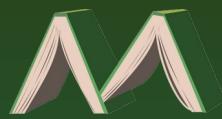




# Architektura aplikacji - opis

Architektura aplikacji składa się z:

- **1. Warstwy prezentacji (frontend)** – zrealizowanej w technologii Angular. Odpowiada za interfejs użytkownika, umożliwia przesyłanie, pobieranie i przeglądanie plików. Komunikuje się z backendem za pośrednictwem REST API.
- **2. Warstwy logiki biznesowej (backend)** – zaimplementowanej w technologii Spring Boot. Odpowiada za przetwarzanie danych, walidację, logikę biznesową oraz komunikację z warstwą baz danych.
- **3. Warstwy danych** – składającej się z dwóch baz: relacyjnej PostgreSQL (metadane plików) i nierelacyjnej MongoDB (przechowywanie plików binarnych).



# Architektura aplikacji - opis

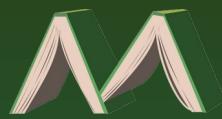
- **Komunikacja** między warstwami odbywa się w modelu klient–serwer z wykorzystaniem protokołu HTTP oraz interfejsów REST API.
- Aplikacja została skonteneryzowana w środowisku Docker, co umożliwia łatwe wdrożenie i skalowanie systemu.



# Zrąb projektu

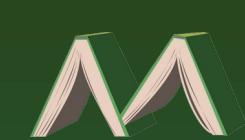
Zaprezentujemy:

- Implementację jednej funkcjonalności w docelowych technologiach – upload file
- Wszystkie warstwy
- Testy



# Lista materiałów szkoleniowych

- Oficjalna dokumentacja Spring Boot
- MongoDB University courses
- Angular.io
- Docker docs
- Baeldung (Spring)
- YouTube – Amigoscode, Java Brains



# Instrukcja budowania projektu

Zawarta w README na Githubie:

## Uruchomienie

z głównego katalogu zwołaj w terminalu:

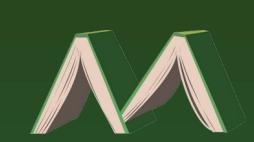
```
docker-compose build  
docker-compose up
```



(można dopisać frontend lub backend lub db-postgres by uruchomić tylko dany obraz z dockera) Program korzysta z biblioteki Material UI, zainstaluj ją komendą:

```
npm install @mui/material  
npm install @hello-pangea/dnd  
npm install @angular/core @angular/common @angular/platform-browser @angular/c
```





Dziękujemy za uwagę :)