

Kierunek: **Informatyka Techniczna (ITE)**  
Specjalność: **Inżynieria Systemów Informatycznych (INS)**

## **PROJEKT**

### **System do zarządzania biblioteką wykorzystujący rozproszone serwery bazodanowe**

inż. Bartosz Błyszcz  
276951@student.pwr.edu.pl

Prowadzący zajęcia  
**dr inż. Robert Wójcik**



## Spis treści

<b>1. Wstęp teoretyczny</b>	4
1.1. Cel projektu	4
1.2. Zakres projektu	4
1.3. Wymagania funkcjonalne	4
1.4. Wymagania niefunkcjonalne	5
<b>2. Baza danych</b>	7
2.1. Model danych	7
2.2. Replikacja bazy danych	7
2.3. Widok tabel	8
<b>3. Docker</b>	9
3.1. Obrazy	9
3.1.1. Nginx	9
3.1.2. PostgreSQL	9
3.1.3. Symfony	11
3.2. Docker Compose	12
<b>4. Aplikacja</b>	14
4.1. Podsumowanie	15

# 1. Wstęp teoretyczny

## 1.1. Cel projektu

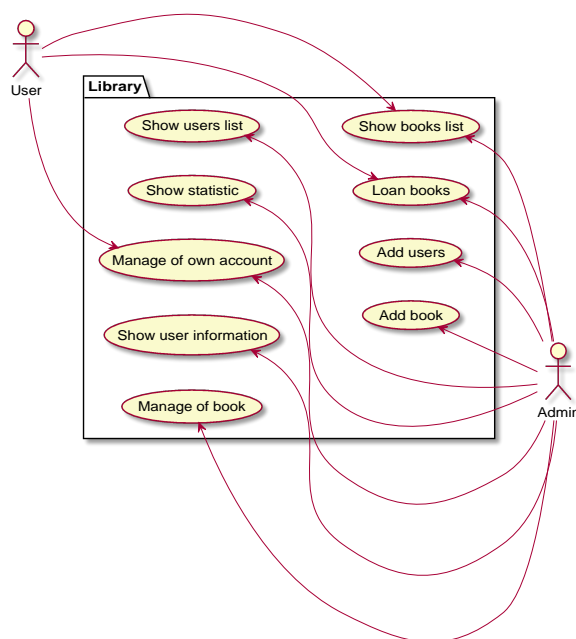
Celem projektu było wykorzystanie rozproszonej bazy danych w celu zwiększenia dostępności danych w systemach bibliotecznych.

## 1.2. Zakres projektu

W zakres projektu wchodziła analiza problemu, dobór technologii oraz rozwiązanie problemu replikacji baz danych. Zbudowanie ekosystemu pozwalającego na replikację oraz obsłużenie go za pomocą load balancera. Aby zbudować ekosystem przeprowadzono gruntowne poszukiwania odpowiedniej technologii pozwalającej na wirtualizację aplikacji oraz ruchu sieciowego oraz planowanie budowy ekosystemu. Na koniec zbudowano oraz skonfigurowano cały ekosystem aplikacji oraz podpięto je do wirtualnej sieci, tak by na zewnątrz pokazać jedynie adres load balancera i poszczególnych aplikacji klienckich (do testów replikacji).

## 1.3. Wymagania funkcjonalne

W skład wymagań funkcjonalnych systemu wchodzi możliwość przez dwóch aktorów obsługi aplikacji „Księgarni”. W zależności od uprawnień aktora można obsłużyć aplikację na dwa sposoby, tak jak pokazuje **rysunku 1.1**.

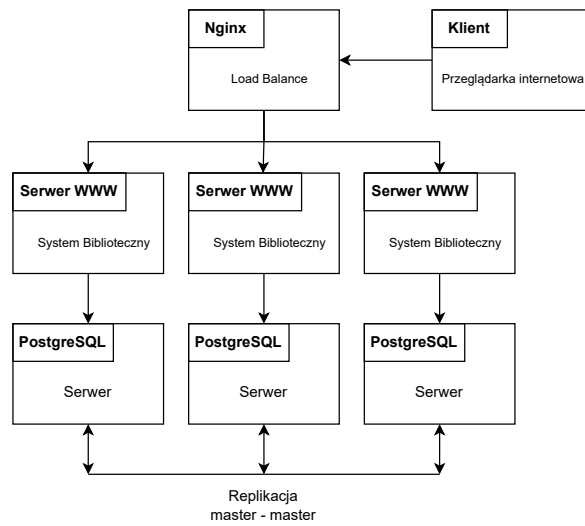


**Rys. 1.1.** Wymagania funkcjonalne systemu  
Źródło: Opracowanie własne

Użytkownik ma do swojej dyspozycji jedynie przegląd, wypożyczanie książek, a także edycję swoich danych. Administrator posiada już więcej możliwości, konto o tej roli pozwala użytkownikowi dodatkowo na zarządzanie książkami oraz oglądanie statystyk wypożyczeń. Administrator może również dodawać i usuwać użytkowników oraz przeglądać ich listę. Na liście książek, do której również ma dostęp, może przeglądać tytuły ich statystyki oraz modyfikować statusy książek i dodawać ich "kopie"

## 1.4. Wymagania niefunkcjonalne

W skład wymagań niefunkcjonalnych wchodzi technologia jaka została wykorzystana do stworzenia środowiska potrzebnego do sprawdzenia możliwości replikacji bazy PostgreSQL [Pos2023]. Do jej obsługi skorzystano aplikacji klienckiej napisanej w języku PHP 7.4 [1], przy użyciu frameworka Symfony 5 [2]. Całość została uruchomiona za pomocą kontenerów Dockerowych [3], całość została obsługiwana przez konfigurację napisaną przy użyciu biblioteki dla Docker: Docker Compose. Dodatkowo Load Balancer został uruchomiony na serwerze Nginx [4]. Wszystkie kontenery zostały wpięte do wirtualnej sieci utworzonej przez Dockera. Schemat aplikacji został zaprezentowany na **obrazie 1.2**.

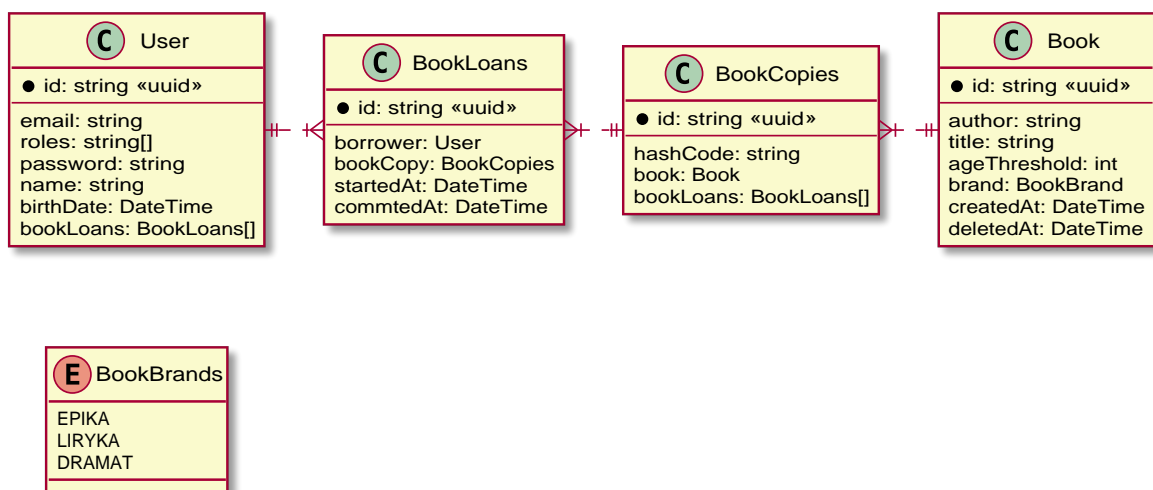


**Rys. 1.2.** Schemat ekosystemu apikacji  
Źródło: Opracowanie własne

## 2. Baza danych

### 2.1. Model danych

Model danych w aplikacji składa się z 4 tabel znajdujących się w bazie danych oraz enuma, który trzyma klasę obiektu. Cechą wspólną obiektów jest posiadane przez nie pole **Id**, które służy za identyfikację danych. Pole to jest typu *string*. Jak można zobaczyć na **modelu 2.1**



**Rys. 2.1.** Model danych  
Źródło: Opracowanie własne

obiekt **BookBrands**, który jest enumem, określa informacje dotyczące gatunku książki. Tabela **User** przechowuje informacje o użytkownikach, oraz łączy się z tabelą **BookLoans** relacją jeden do wielu, co oznacza, że jeden użytkownik, może wypożyczyć wiele książek. Tabela **BookLoans** łączy się relacją wiele do jednego z tabelą **BookCopies** co oznacza, że jeden egzemplarz książki może być wypożyczony wiele razy. Dodatkowo tabela **BookCopies** łączy się z tabelą **Books** relacją wiele do jednego, ponieważ jedna książka może mieć wiele kopii.

### 2.2. Replikacja bazy danych

Replikacja bazy danych to umożliwienie automatycznego rozgłoszenia danych, przez jednostkę nadrzędną (*master*) do jednostek podrzędnych (*slave*). W projekcie wykorzystującym bazę danych PostgreSQL [Pos2023] zastosowano replikację master-master. Replikacja ta pozwala na to, aby każdy serwer bazy danych był

serwerem równorzędnym, co pozwala na zapisywanie i odczytywanie danych z każdego serwera, dzięki czemu podczas awarii jednego mastera, drugi może przejąć jego ruch. Do uruchomienia takiej usługi wykorzystano oprogramowanie Bucardo [5]. Bucardo jest to system wspomagający replikację bazy danych. Łączy się do wielu baz danych i umożliwia działanie na zasadzie "Wyzwalacza", który propaguje dane z jednej bazy na wszystkie pozostałe. Oprogramowanie to trzeba zainstalować dodatkowo na serwerach baz danych, aby mógł on rozgłaszać dane, którą są dodawane, usuwane, zmieniane. Wykorzystanie Bucardo okazało się koniecznym krokiem, przez wzgląd na to, że PostgreSQL [Pos2023], nie posiada wbudowanej replikacji master-master. Dodatkowo Bucardo nie umożliwia replikacji danych znajdujących się przed jego uruchomieniem na serwerze bazodanowym.

## 2.3. Widok tabel

Tabele w bazie danych wyglądają tak jak na **zdjęciu 2.2**. Dodatkowo tabela **Book**, prezentuje się jak na **zrzucie 2.2**.

```
user_d=# \d
```

List of relations			
Schema	Name	Type	Owner
public	book	table	user_d
public	book_copies	table	user_d
public	book_copies_id_seq	sequence	user_d
public	book_id_seq	sequence	user_d
public	books_loans	table	user_d
public	books_loans_id_seq	sequence	user_d
public	user	table	user_d
public	user_id_seq	sequence	user_d

(8 rows)

**Rys. 2.2.** Zrzut tabel  
Źródło: Opracowanie własne

```
user_d=# SELECT * FROM book
```

id	author	title	age_threshold	brand	created_at	deleted_at
1	Kowalski Jan	Jan Kowalski	12	Epika	2023-11-10 09:56:10	
2	Kozek	Jozek	15	Epika	2023-11-10 10:01:00	
3	Wiara Wierzy Wieży	Krzywa Wieża Wiary	16	Epika	2023-11-10 10:02:15	

(3 rows)

**Rys. 2.3.** Wyniki zapytania SELECT na tabeli book  
Źródło: Opracowanie własne



## 3. Docker

Docker jest to narzędzie tworzące wirtualne kontenery. Kontenery te mogą odpowiadać za pojedynczą usługę, bądź całe systemy operacyjne. Zadaniem Dockera jest wspomagać pracę programisty nad skomplikowanym ekosystemem aplikacji. Dodatkowo wspiera on pracę z mikroserwisami.

### 3.1. Obrazy

W projekcie wykorzystano 3 rodzaje obrazów:

#### 3.1.1. Nginx

Obraz dla serwera Nginx, jest napisany bardzo prosto, ze względu na to, że zadaniem aplikacji jest jedynie bycie Load Balancerem, dlatego kopiuje on konfigurację Load Balancera, oraz uruchamia serwer Nginx, tak jak to przedstawiono na **wycinkach kodu 3.1, 3.2**.

```
1 FROM nginx
2 RUN rm /etc/nginx/conf.d/default.conf
3 COPY nginx.conf /etc/nginx/conf.d/default.conf
```

**Fragment kodu 3.1.** Plik Dockerfile dla Nginx

```
1 upstream loadbalancer {
2     ip_hash;
3     server 172.17.0.1:8081 weight=2;
4     server 172.17.0.1:8082 weight=2;
5     server 172.17.0.1:8083 weight=2;
6 }
7
8 server {
9     location / {
10         proxy_pass http://loadbalancer;
11     }
12 }
```

**Fragment kodu 3.2.** Plik konfiguracyjny dla Nginx

#### 3.1.2. PostgreSQL

Obraz dla serwera PostgreSQL, wykorzystuje gotowy predefiniowany obraz dla PostgreSQL, dodając instalację Bucardo. Plik Dockerfile dla PostgreSQL został przedstawiony na **wycinku kodu 3.3**, a konfiguracja Bucardo znajduje się w **plik 3.4**.

```

1 FROM postgres
2 COPY init.sql /docker-entrypoint-initdb.d/10-init.sql
3 COPY bbb.sh /bbb.sh
4 WORKDIR /
5
6 RUN apt update
7 RUN apt install -y wget curl postgresql-plperl-16
8 RUN apt install -y make libdbix-safe-perl libboolean-perl libdbd-mock-perl libdbd-pg-perl libanyevent-db-d-pg-perl libp
9 RUN wget -q https://bucardo.org/downloads/Bucardo-5.6.0.tar.gz
10 RUN tar xf Bucardo-5.6.0.tar.gz
11
12 RUN cd Bucardo-5*/ && perl Makefile.PL && make install
13 RUN mkdir -p /var/run/bucardo /var/log/bucardo
14 RUN touch /var/log/bucardo/log.bucardo

```

### Fragment kodu 3.3. Plik Dockerfile dla PostgreSQL

[illegible]

### Fragment kodu 3.4. Skrypt konfiguracyjny dla Bucardo

### 3.1.3. Symfony

Obraz Symfony bazuje na systemie alpine, który jest minimalną instalacją systemu bazującego na jądrze Linux. Jego zadaniem jest zainstalować odpowiednie sterowniki, a następnie uruchomić kontener i uruchomić aplikację. Konfiguracja została przedstawiona w **pliku 3.5**.

```
1 FROM alpine:3.18.4
2 RUN apk add curl php php-xml php-curl bash php-common php-pgsql php-iconv php-mbstring php81-ctype
3 RUN apk add php-session
4 RUN apk add php-dom
5 RUN apk add php-tokenizer
6 RUN apk add php-pdo
7 RUN apk add php-pgsql
8 RUN apk add postgresql
9 RUN apk add php-pdo_pgsql
10 RUN apk add php-simplexml
11 COPY . /app
12 COPY ./php.ini /etc/php81
13 WORKDIR /app
14 RUN bash ./setup.alpine.sh
15 RUN apk add symfony-cli
16 CMD symfony server:start --no-tls
```

**Fragment kodu 3.5.** Plik Dockerfile dla Symfony

## 3.2. Docker Compose

Docker Compose [DocCom2023] to oprogramowanie wykorzystywane do tworzenia konfiguracji obsługujących wiele aplikacji, które są łączone w jeden ekosystem z zależnościami między nimi. Konfiguracja wykorzystywana w projekcie została opisana w **pliku 3.6**. Konfiguracja pozwala na uruchomienie trzech kontenerów aplikacji, trzech kontenerów bazy danych oraz load balancera. Dodatkowo Docker Compose umożliwia utworzenie zależności między aplikacjami dzięki czemu, aplikacje uruchamiają się w odpowiednim czasie.

```
1 version: "3.8"
2 services:
3   app1:
4     image: lsm
5     ports:
6       - "8081:8000"
7     environment:
8       - DATABASE_URL=postgresql://user_d:user_d@db1:5432/user_d?serverVersion=13&charset=utf8
9     depends_on:
10      - db1
11   app2:
12     image: lsm
13     ports:
14       - "8082:8000"
15     environment:
16       - DATABASE_URL=postgresql://user_d:user_d@db2:5432/user_d?serverVersion=13&charset=utf8
17     depends_on:
18      - db2
19   app3:
20     image: lsm
21     ports:
22       - "8083:8000"
23     environment:
24       - DATABASE_URL=postgresql://user_d:user_d@db3:5432/user_d?serverVersion=13&charset=utf8
25     depends_on:
26      - db3
27   nginx:
28     build: ./nginx
29     ports:
30       - "8080:80"
31     depends_on:
32      - app1
33      - app2
34      - app3
35   db1:
36     build: ./sql
37     ports:
38       - "5432:5432"
39     environment:
40       POSTGRES_PASSWORD: user_d
41       POSTGRES_USER: user_d
42       POSTGRES_DB: user_d
43       POSTGRES_HOST_AUTH_METHOD: trust
44     depends_on:
45      - db2
46      - db3
47
```

```
48 db2:
49   build: ./sql
50   ports:
51     - "5433:5432"
52   environment:
53     POSTGRES_PASSWORD: user_d
54     POSTGRES_USER: user_d
55     POSTGRES_DB: user_d
56     POSTGRES_HOST_AUTH_METHOD: trust
57   depends_on:
58     - db3
59 db3:
60   build: ./sql
61   ports:
62     - "5434:5432"
63   environment:
64     POSTGRES_PASSWORD: user_d
65     POSTGRES_USER: user_d
66     POSTGRES_DB: user_d
67     POSTGRES_HOST_AUTH_METHOD: trust
```

**Fragment kodu 3.6.** Plik Docker Compose ekosystemu aplikacji

## 4. Aplikacja

Uruchomiony ekosystem prezentuje logi tak jak na **schemacie 4.1**. Konfigurację uruchamiania się za pomocą **polecenia 4.1**.

- \$ docker compose up

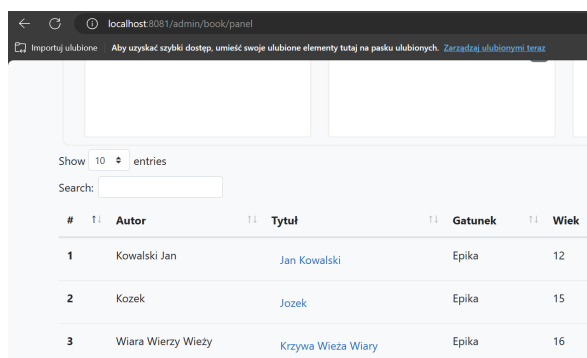
### Fragment kodu 4.1. Uruchomienie Docker Compose

```
ribute should be used to temporarily suppress the notice
roisb_pwr-app1-1 | [Application] Oct 14 12:55:10 [INFO] | PHP | Deprecated: Return type of Symfony\Component\VarDumper\Cloner\Dat
a::offsetSet($key, $value) should either be compatible with ArrayAccess::offsetSet(mixed $offset, mixed $value): void, or the #[\Retu
rnTypeWillChange] attribute should be used to temporarily suppress the notice
roisb_pwr-app1-1 | [Application] Oct 14 12:55:10 [INFO] | PHP | Deprecated: Return type of Symfony\Component\VarDumper\Cloner\Dat
a::offsetUnset($key) should either be compatible with ArrayAccess::offsetUnset(mixed $offset): void, or the #[\ReturnTypeWillChange]
attribute should be used to temporarily suppress the notice
roisb_pwr-app1-1 | [Application] Oct 14 12:55:10 [INFO] | PHP | Deprecated: Return type of Symfony\Component\VarDumper\Cloner\Dat
a::count() should either be compatible with Countable::count(): int, or the #[\ReturnTypeWillChange] attribute should be used to temp
orarily suppress the notice
roisb_pwr-app1-1 | [Application] Oct 14 12:55:10 [INFO] | PHP | Deprecated: Return type of Symfony\Component\VarDumper\Cloner\Dat
a::getIterator() should either be compatible with IteratorAggregate::getIterator(): Traversable, or the #[\ReturnTypeWillChange] attr
ibute should be used to temporarily suppress the notice
roisb_pwr-app1-1 | [Web Server] Nov 10 09:31:37 [INFO] | PHP | Listening path="/usr/bin/php81" php="8.1.23" port=34517
roisb_pwr-app1-1 | [PHP] [Fri Nov 10 09:31:37 2023] PHP 8.1.23 Development Server (http://127.0.0.1:34517) started
roisb_pwr-db2-1 | done
roisb_pwr-db2-1 | server stopped
roisb_pwr-db2-1 |
roisb_pwr-db2-1 | PostgreSQL init process complete; ready for start up.
roisb_pwr-db2-1 |
roisb_pwr-db2-1 | 2023-11-10 09:31:39.369 UTC [1] LOG: starting PostgreSQL 16.0 (Debian 16.0-1.pgdg120+1) on x86_64-pc-linux-gnu,
compiled by gcc (Debian 12.2.0-10) 12.2.0, 64-bit
roisb_pwr-db2-1 | 2023-11-10 09:31:39.370 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
roisb_pwr-db2-1 | 2023-11-10 09:31:39.370 UTC [1] LOG: listening on IPv6 address "::", port 5432
roisb_pwr-db1-1 | waiting for server to start...2023-11-10 09:31:39.373 UTC [49] LOG: starting PostgreSQL 16.0 (Debian 16.0-1.pg
dg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-10) 12.2.0, 64-bit
roisb_pwr-db2-1 | 2023-11-10 09:31:39.377 UTC [49] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
roisb_pwr-db2-1 | 2023-11-10 09:31:39.377 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
roisb_pwr-db2-1 | 2023-11-10 09:31:39.385 UTC [67] LOG: database system was shut down at 2023-11-10 09:31:39 UTC
roisb_pwr-db1-1 | 2023-11-10 09:31:39.387 UTC [52] LOG: database system was shut down at 2023-11-10 09:31:38 UTC
roisb_pwr-db1-1 | 2023-11-10 09:31:39.398 UTC [49] LOG: database system is ready to accept connections
```

Rys. 4.1. Uruchomiony Docker Compose

Źródło: Opracowanie własne

Po wprowadzeniu nowej książki do aplikacji pod adresem *localhost:8081* oraz przejściu pod adres *localhost:8082*, *localhost:8083* można zauważyć, że zmiany wprowadzone po uruchomieniu aplikacji Bucardo, zostały rozpropagowane dalej. Widać to na **zdjęciach: 4.2, 4.3**.



#	Ti	Autor	Ti	Tytuł	Ti	Gatunek	Ti	Wiek
1		Kowalski Jan		Jan Kowalski		Epika		12
2		Kozek		Jozek		Epika		15
3		Wiara Wierzy Wieży		Krzywa Wieża Wiary		Epika		16

Rys. 4.2. Dodanie dwóch wpisów do tabeli po uruchomieniu Bucardo

Źródło: Opracowanie własne

#	Autor	Tytuł	Gatunek	Wiek
1	Kozek	Jozek	Epika	15
2	Wiara Wierzy Wieży	Krzywa Wieża Wiary	Epika	16

**Rys. 4.3.** Rozpropagowanie dwóch wpisów do tabeli po uruchomieniu Bucardo

Źródło: Opracowanie własne

## 4.1. Podsumowanie

Wykorzystanie replikacji baz danych pozwala na swobodne tworzenie kopii zapasowych oraz rozładowanie ruchu sieciowego. Pozwala to na swoistą regionalizację aplikacji, umożliwiając użytkownikom korzystanie z serwerów znajdujących się w ich regionie. Dodatkowo do stworzenia tego typu instancji bardzo przydatny okazał się *Docker*, pozwolił on na stworzenie jednakowych obrazów aplikacji, które mogą posłużyć do uruchomienia na odpowiednim serwerze, co zostało zobrazowane a pomocą narzędzia *Docker Compose*, które umożliwiło zrobienia pseudoregionalizacji, tworząc cały ekosystem aplikacji tak jak na **schemacie 1.2**. Wykorzystanie narzędzia *Bucardo* pozwoliło na utworzenie replikacji ”master-master”, która nie znajduje się domyślnie w bazie danych *PostgreSQL*. Wykorzystana replikacja umożliwiła na niezależną synchronizację danych pomiędzy bazami danych. Synchronizacja ta dzieje się niezależnie od instancji aplikacji klienckich oraz bazy danych. Ponieważ *Bucardo* to narzędzie reagujące na wydarzenie, w tym wypadku zmiana w bazie danych, a następnie propagujące to wydarzenie na pozostałe serwery bazodanowe. Aplikacja ta dzięki temu, że znajduje się na serwerze bazodanowym, wykonuje się jedynie w momencie kiedy ta baza ”działa”.

# Wykaz rysunków

1.1	Wymagania funkcjonalne systemu . . . . .	5
1.2	Schemat ekosystemu aplikacji . . . . .	6
2.1	Model danych . . . . .	7
2.2	Zrzut tabel . . . . .	8
2.3	Wyniki zapytania SELECT na tabeli book . . . . .	8
4.1	Uruchomiony Docker Compose . . . . .	14
4.2	Dodanie dwóch wpisów do tabeli po uruchomieniu Bucardo . . . . .	14
4.3	Rozpropagowanie dwóch wpisów do tabeli po uruchomieniu Bucardo . . . . .	15



## Wykaz tabel

## Wykaz fragmentów kodu

3.1	Plik Dockerfile dla Nginx . . . . .	9
3.2	Plik konfiguracyjny dla Nginx . . . . .	9
3.3	Plik Dockerfile dla PostgreSQL . . . . .	10
3.4	Skrypt konfiguracyjny dla Bucardo . . . . .	10
3.5	Plik Dockerfile dla Symfony . . . . .	11
3.6	Plik Docker Compose ekosystemu aplikacji . . . . .	12
4.1	Uruchomienie Docker Compose . . . . .	14

# Bibliografia

- [1] PHP. „*PHP*”. URL: <https://www.php.net/> (term. wiz. 2023-11-04).
- [2] Symfony. „*Symfony*”. URL: <https://www.symfony.com/> (term. wiz. 2023-11-04).
- [3] Docer. „*Docker*”. URL: <https://www.docker.com/> (term. wiz. 2023-11-04).
- [4] Nginx. „*Nginx*”. URL: <https://www.nginx.com/> (term. wiz. 2023-11-04).
- [5] Bucardo. „*Bucardo*”. URL: <https://bucardo.org/> (term. wiz. 2023-11-04).