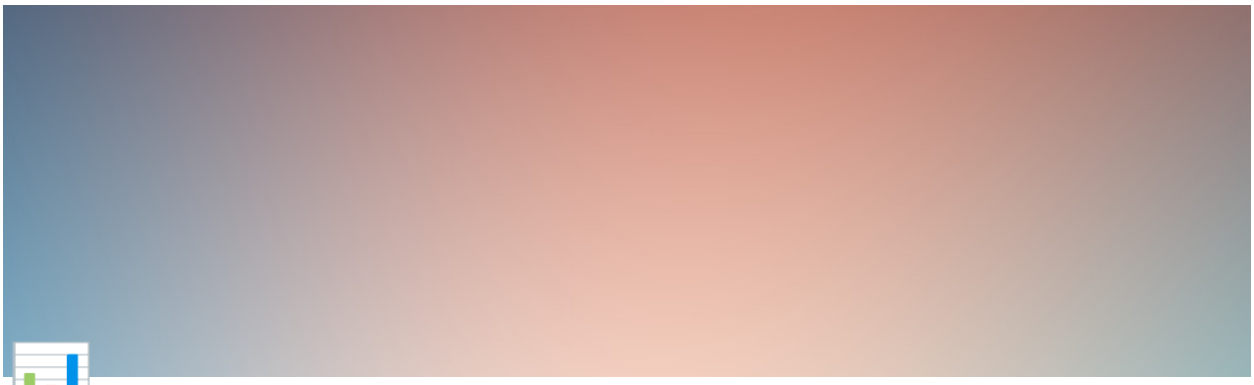


seriall

> Aktualny build: **SLL-260523REV1**

> Aktualna wersja: 1.1.2

Changelog:

SLL-260523REV1 Latest

Added database autoreconnect feature.

SLL-170523REV1

Added Unique codes table.

UI Fixes

SLL-170523REV0

First production grade release.

Podstawowa funkcjonalność aplikacji.

Aplikacja służy do zarządzania wpisanymi przez użytkownika kodami paskowymi. Kody paskowe które obsługuje mogą zostać automatycznie zaczytane z urządzenia typu skaner. Aplikacja obsługuje kody paskowe jak i kody QR.

Podstawową funkcjonalnością aplikacji jest możliwość filtracji wpisanych kodów paskowych na dwa podstawowe filtry: kod jest unikalny, kod jest powtarzający się.

Aplikacja została stworzona w celu rozwiązania problemu automatyzacji seryjnie wpisywanych kodów paskowych i ich filtracji.

Zastosowane technologie.

Aplikacja:

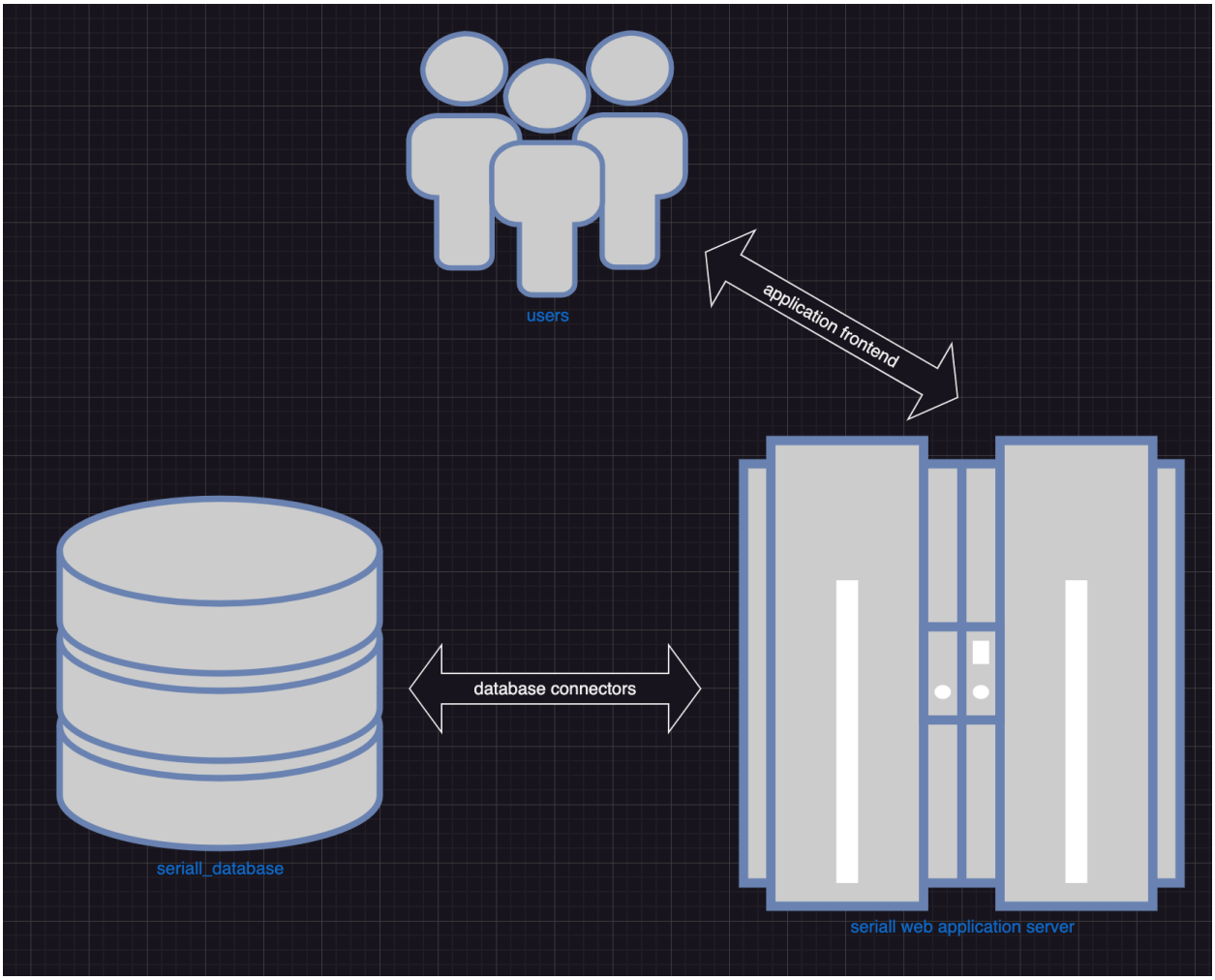
- JAVA 17
- HIBERNATE

- SPRING BOOT 2.8.3
- VAADIN 23.1

Baza danych:

- MYSQL SERVER 8.0.31

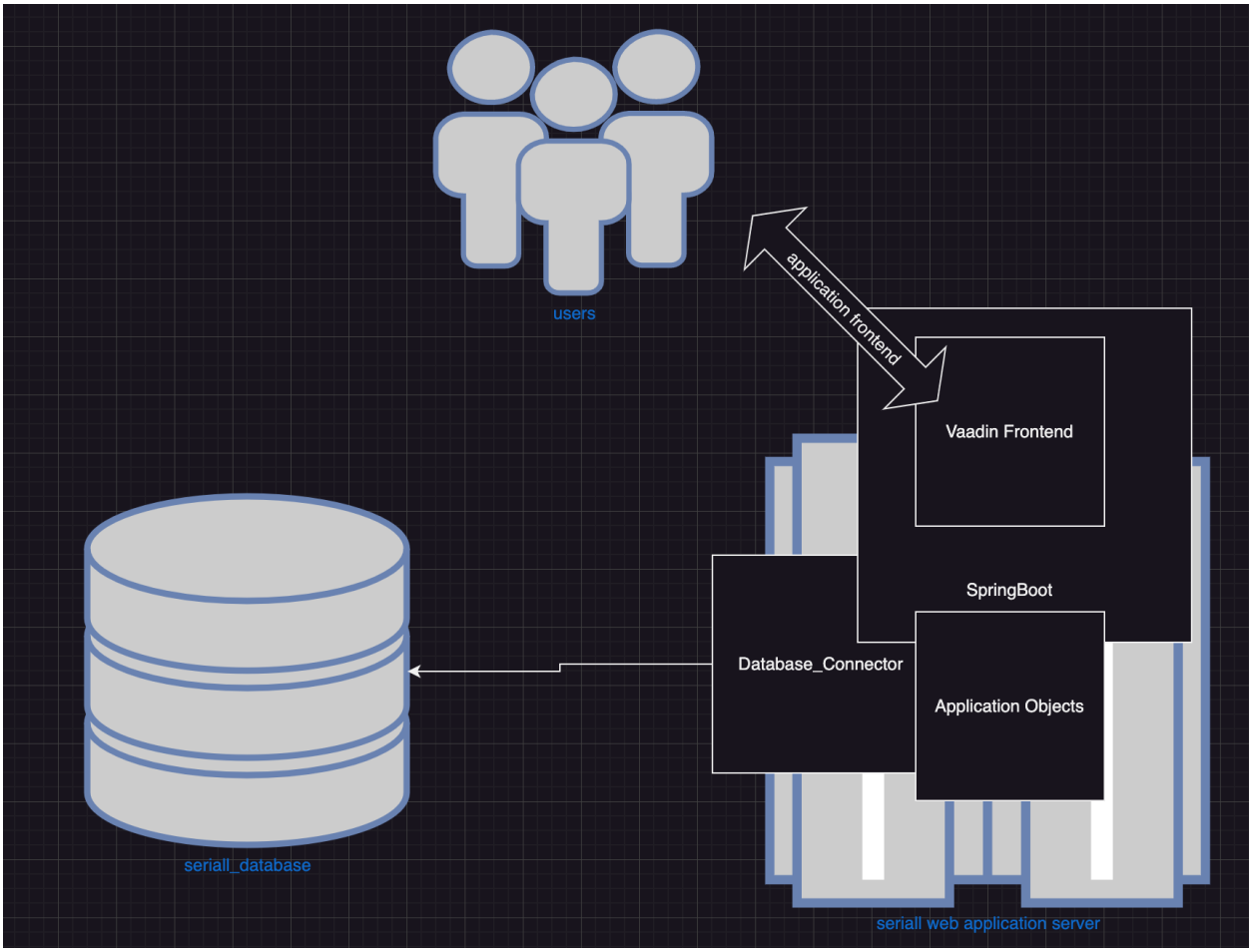
Schemat architektury.



Aplikacja opiera swoje działanie na dwóch podstawowych komponentach: bazie danych zawierającej wszystkie dane aplikacji i serwerowi aplikacyjnemu serwującemu aplikacje webową dla użytkowników. Aplikacja jest stworzona w taki sposób że baza i serwer może być na tej samej maszynie. Connector bazy danych umożliwia podłączenie się do dowolnego serwera bazodanowego. Connector jest napisany specyficznie do łączenia się jedynie z tą konkretną bazą. Przy połączeniu waliduje schemat bazy i wersję oprogramowania bazy.

Architektura umożliwia również kompletnie autonomiczne podłączenie aplikacji i bazy bez względu na sieci w której się znajdują.

Funkcjonalny schemat kodu.



Całą logikę aplikacji utrzymuje obiekt Database_Connector. Obiekt ten utrzymuje połączenie z bazą danych.

```
public class Database_Connector {

    public boolean connected;
    LocalDateTime run_time;
    public Connection con;

    private String user, password,name,ip;

    /**
     * Constructor with parameters
     * Parameters contains information needed to connection
     * @param user
     * @param password
     * @param name
     * @param ip
     */
    public Database_Connector(String user, String password,String name, String ip){
        this.user = user;
        this.password = password;
        this.name = name;
        this.ip = ip;
        connected = false;
        run_time = null;
    }
}
```

Utrzymanie połączenia z bazą danych jest utrzymywane przez obiekt Database_ConnectorUpdater. Updater relizuje reconnect z bazą danych co 8 godzin w celu wymuszenia nowego połączenia. Wynika to z faktu zablokowania po stronie serwera możliwości tworzenia połączeń trwających dłużej niż 8 godzin,

```
/**
 * Object for maintaing reconnection to database
 */
public class Database_ConnectorUpdater implements Runnable{

    int time;

    /**
     * Constructor
     * @param time
     */
    public Database_ConnectorUpdater(int time){
        SeriallApplication.database.log("DB-RECONNECT","Trying to reconnect to database!");
        this.time = time;
    }
    @Override
    public void run() {
        try{
            while(true){
                if (SeriallApplication.database != null){
                    if ( SeriallApplication.database.validate_object() ){
                        SeriallApplication.database.connect();
                        if ( SeriallApplication.database.connected ){
                            SeriallApplication.database.log("DB-RECONNECT","Reconnected successfully!");
                        }
                        else{
                            SeriallApplication.database.log("DB-RECONNECT-FAILED","Failed to reconnect application! Check log!");
                        }
                    }
                }
                Thread.sleep(time);
            }
        }catch(Exception ex){
            SeriallApplication.database.log("THREAD-RECONNECT-FAILED","Failed to auto reconnect database ("+ex.toString()+")");
        }


    }
}
```

Tworzy transakcje zapisu danych i utrzymuje dane pokazywane na frontendzie aplikacji. W przypadku danych aplikacji są one reprezentowane przez obiekty odpowiadające im funkcjonalnie.

Budowa sesji aplikacji.

Sesja aplikacji jest globalna. Oznacza to że każdy użytkownik posiadający dostęp do aplikacji widzi te same dane. Instnieje możliwość kolaboracji nad danymi. Użytkownik robiący zmiany na wybranym secie danych blokuje je na przeglądarce i dopiero po wykonaniu akcji są one aktualizowane na innych klientach. Dane są globalne ale zmiany są lokalnymi transakcjami.

Deployment aplikacji i pierwsze uruchomienie.

1. Przygotowanie środowiska bazy danych. Wykonanie skryptu tworzącego bazę danych na instancji MySQL.
2. Utworzenie użytkownika na serwerze MySQL z uprawnieniami SELECT INSERT do bazy danych.
3. Skopiowanie na maszynę archiwum aplikacji i wypakowania całej zawartości w jednej lokalizacji.
4. Uruchomienie archiwum javy komendą:

```
java -jar seraill.jar
```

Aplikacja została uruchomiona na porcie 6969. Wszystkie możliwe do wykonania kopie zapasowe wpisywanych danych można realizować za pomocą wbudowanych funkcjonalności MySQL. Dobrą praktyką jest utworzenie zasad na firewallu umożliwiających jedynie wybranym komputerom dostęp do strony wstawianej przez aplikacje.