MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Managementul unui Service IT

Tuns Andrei
12 ianuarie 2026

Cuprins

1 Introducere

1.1 Context general

Aplicația de service a fost concepută pentru a răspunde nevoii de digitalizare și eficientizare a proceselor de management într-un serviciu de reparații și mentenanță pentru echipamente IT. Scopul fundamental este de a centraliza gestionarea comenzilor, a clienților și a dispozitivelor, oferind o platformă unică, accesibilă și interactivă atât pentru personalul de service, cât și pentru clienți.

1.2 Obiective

Scopul principal al acestei aplicații este de a simplifica și automatiza fluxurile de lucru dintr-un serviciu IT. Prin dezvoltarea acestei platforme, se urmăresc următoarele obiective:

- Gestionarea eficientă a comenzilor de service, de la preluare până la finalizare.
- Centralizarea informațiilor despre clienți și dispozitivele acestora.
- Oferirea unui portal pentru clienți unde pot urmări statusul reparațiilor.
- Notificarea automată a clienților la schimbarea statusului comenzii.
- Generarea de fișe de service în format PDF cu coduri QR pentru acces rapid la informații.

1.3 Lista MoSCoW

Pentru a prioritiza cerințele proiectului KIVA NET Service, am utilizat metoda Moscow. Aceasta clasifică funcționalitățile în patru categorii esențiale, asigurând o dezvoltare concentrată pe aspectele critice ale aplicației.

Tabela 1: Lista de priorități MoSCoW pentru KIVA NET Service

Must have	**Should have**
• Autentificare utilizatori • Management comenzi • Management clienți • Management dispozitive	• Portal client • Generare PDF și cod QR • Notificări automate (WhatsApp) • Istoric reparații • Căutare avansată

Could have	**Won't have**
• Programări online • Rapoarte statistice • Management inventar piese • Chat intern	• Modul de facturare • Integrare cu soft de contabilitate • Suport multi-lingvistic

1.4 Cazuri de utilizare

Aplicația definește interacțiuni clare pentru două tipuri principale de utilizatori: personalul intern (administratori/tehnicieni) și clienți. Diagrama de mai jos ilustrează principalele fluxuri operaționale.

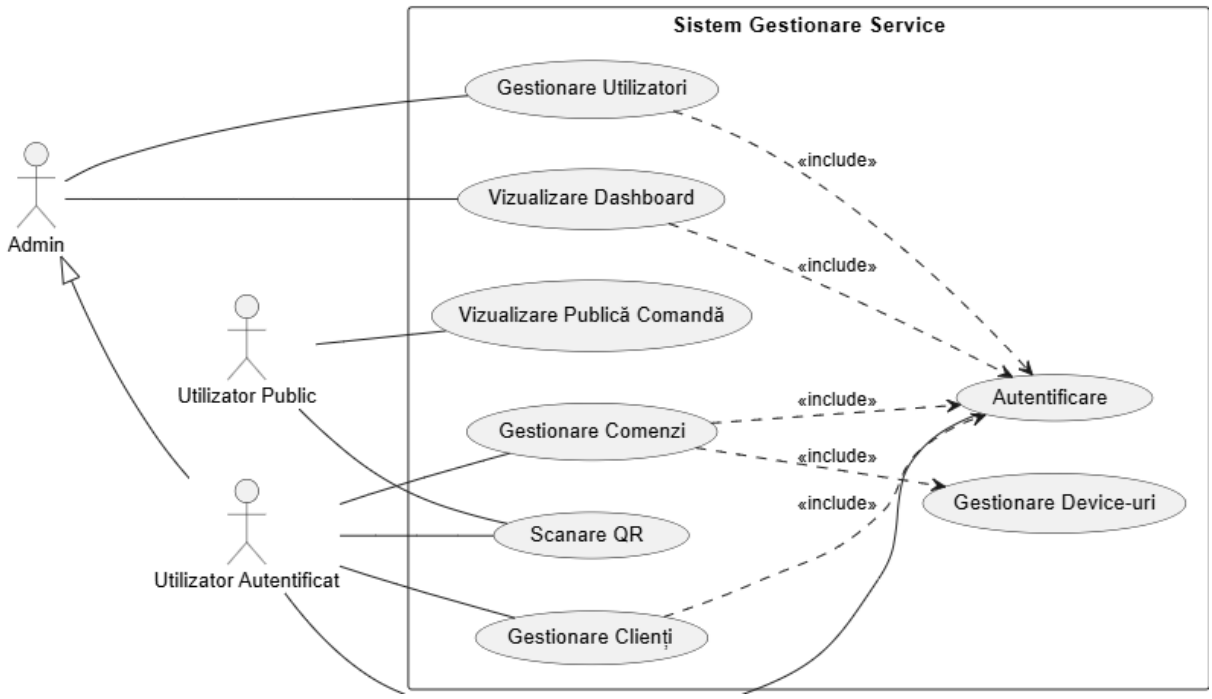

Figura 1: Diagrama cazurilor de utilizare pentru KIVA NET Service.

2 Arhitectura

Arhitectura aplicației este proiectată pentru a fi robustă, scalabilă și ușor de întreținut. Am optat pentru o arhitectură modernă, bazată pe un front-end reactiv și un back-end solid care gestionează logica de business și comunicarea cu baza de date și serviciile externe.

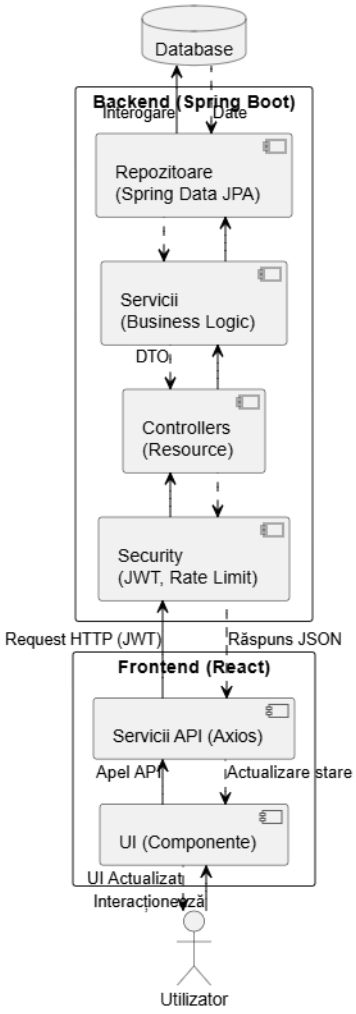

Figura 2: Diagrama arhitecturii aplicației KIVA NET Service.

2.1 Front-End: React cu Bootstrap

Partea de front-end este dezvoltată folosind biblioteca React, care permite crearea unei interfețe de utilizator dinamice și performante, bazată pe componente. Pentru stilizare și design responsiv, s-a utilizat framework-ul Bootstrap, asigurând o experiență consistentă pe diverse dispozitive.

2.2 Back-End: Spring Boot

Pentru back-end, s-a utilizat framework-ul robust Spring Boot (Java). Acesta expune o serie de API-uri RESTful pentru comunicarea cu front-end-ul. Logica de business, autentificarea utilizatorilor și procesarea datelor sunt gestionate la acest nivel.

2.3 Comunicare între Front-End şi Back-End

Comunicarea dintre cele două componente se realizează prin cereri HTTP asincrone (folosind Axios). Front-end-ul trimite cereri către API-ul back-end pentru a prelua, adăuga, modifica sau șterge date, iar back-end-ul răspunde cu date în format JSON.

2.4 Comunicare în timp real: WebSockets

Pentru actualizări în timp real, aplicația implementează WebSockets folosind protocolul STOMP peste SockJS. Aceasta permite:

- Notificări push către clienți când statusul comenzii se schimbă
- Actualizarea automată a listelor fără refresh manual
- Experiență utilizator îmbunătățită prin date live

2.5 Baza de Date

A fost ales un sistem de management al bazelor de date relațional MySQL, datorită fiabilității, popularității și suportului pentru tranzacții complexe. Structura bazei de date este concepută pentru a stoca eficient informații despre utilizatori, comenzi, clienți și dispozitive.

2.6 Beneficii ale acestei arhitecturi

- **Scalabilitate:** Separarea clară între front-end și back-end permite scalarea independentă a celor două componente.
- **Flexibilitate:** Tehnologiile pentru front-end și back-end pot fi actualizate sau înlocuite independent una de cealaltă.
- **Eficiență în dezvoltare:** Echipe diferite pot lucra în paralel la front-end și back-end, accelerând procesul de dezvoltare.
- **Performanță:** Arhitectura permite optimizarea performanței prin distribuirea sarcinilor între client și server.

3 Implementare

3.1 Tehnologii Frontend

În dezvoltarea interfeței cu utilizatorul, am adoptat o abordare modernă pentru a asigura o experiență fluidă și intuitivă.

- **React:** Biblioteca principală pentru construcția interfeței, permițând dezvoltarea bazată pe componente reutilizabile și gestionarea eficientă a stării aplicației.
- **Bootstrap:** Utilizat pentru stilizarea componentelor și pentru a asigura un design complet responsiv, adaptabil pentru desktop, tablete și telefoane mobile.

- **HTML și JavaScript (ES6+):** Limbajele fundamentale ale web-ului, utilizate pentru structura paginilor și logica dinamică a componentelor.
- **Vite:** Build tool modern pentru development rapid cu Hot Module Replacement (HMR).

3.2 Tehnologii Backend

Backend-ul aplicației gestionează logica de business, interacțiunile cu baza de date și comunicarea cu servicii externe.

- **Spring Boot:** Framework-ul Java care simplifică dezvoltarea de aplicații web robuste și servicii RESTful. Acesta gestionează configurarea, securitatea și accesul la date.
- **Spring Data JPA:** Utilizat pentru a simplifica interacțiunea cu baza de date MySQL, mapând obiectele Java la tabelele din baza de date.
- **Spring Security:** Asigură mecanismele de autentificare și autorizare pentru a proteja endpoint-urile API.
- **Maven:** Folosit pentru managementul dependențelor și pentru build-ul proiectului.

3.3 Integrarea API-ului WhatsApp

O funcționalitate cheie a aplicației este comunicarea proactivă cu clienții.

- **Notificări automate:** Atunci când statusul unei comenzi se modifică (ex: "în diagnoză", "așteaptă piese", "reparat", "finalizat"), sistemul trimite automat o notificare clientului.
- **API Extern:** Această funcționalitate este implementată prin integrarea cu un API de la WhatsApp Business. Backend-ul (Spring Boot) inițiază o cerere către acest API, transmițând numărul de telefon al clientului și mesajul corespunzător noului status al comenzii.
- **Comunicare eficientă:** Utilizarea WhatsApp asigură o rată ridicată de livrare și vizualizare a mesajelor, îmbunătățind considerabil experiența clientului.

3.4 Generare PDF și Cod QR

Pentru a eficientiza fluxul de lucru și a oferi transparență clienților, aplicația generează automat o fișă de service în format PDF pentru fiecare comandă.

- **Generare PDF:** La crearea unei comenzi, backend-ul generează un document PDF care conține detaliile comenzii și un cod QR unic. Pentru această funcționalitate se utilizează biblioteci Java precum iText sau Apache PDFBox.
- **Cod QR cu dublă funcționalitate:** Codul QR încorporează un link unic către o resursă din aplicație și are roluri diferite în funcție de utilizator:

– **Pentru tehnician/administrator:** Scanarea codului QR deschide o pagină securizată în aplicație care afișează un *to-do list* (listă de sarcini) specific reparației acelui dispozitiv.
– **Pentru client:** Scanarea aceluiași cod QR deschide o pagină publică, unde clientul poate vizualiza în timp real statusul actual al comenzii sale, fără a fi nevoie să se autentifice.

4 Structura Proiectului

4.1 Structura Backend

Proiectul backend urmează o arhitectură pe straturi (layered architecture), specifică aplicațiilor Spring Boot:

```
backend/
 src/main/java/com/example/backend/
    BackendApplication.java      # Punctul de intrare
    config/                      # Configurări
       AdminInitializer.java
       CacheConfig.java
       SecurityConfig.java
       WebSocketConfig.java
    domain/                      # Entități JPA
       Client.java
       Device.java
       Order.java
       OrderLog.java
       User.java
    dto/                         # Data Transfer Objects
    exception/                   # Gestionare erori
    repo/                        # Repository-uri JPA
    resource/                    # Controller-e REST
    security/                    # Securitate JWT
    service/                     # Logica de business
    specification/               # Filtrare dinamică
 src/main/resources/
     application.properties       # Configurare aplicație
     db/migration/                # Migrări Flyway
```

4.2 Structura Frontend

Proiectul frontend este organizat după principiile React moderne:

```
frontend/
 src/
    App.jsx                      # Componenta principală
    main.jsx                     # Punctul de intrare
    components/                  # Componente React
       Dashboard.jsx
       Orders.jsx
       Clients.jsx
       Devices.jsx
       Users.jsx
       Login.jsx
       PublicOrderStatus.jsx
    contexts/                    # Context React (Auth)
```

```
   services/api/                # Servicii API
   styles/                      # Stiluri CSS
   utils/                       # Utilități
index.html
```

5 Modelul de Date

5.1 Entitatea User

Reprezintă utilizatorii sistemului (administratori și tehnicieni).

Tabela 2: Structura entității User

Câmp	Tip	Descriere
id	Long	Identificator unic (PK)
username	String	Nume de utilizator (unic)
password	String	Parola criptată (BCrypt)
role	String	Rolul utilizatorului (ADMIN/USER)
createdAt	LocalDateTime	Data creării contului

5.2 Entitatea Client

Reprezintă clienții service-ului, care pot fi persoane fizice sau juridice.

Tabela 3: Structura entității Client

Câmp	Tip	Descriere
id	Long	Identificator unic (PK)
name	String	Numele clientului
email	String	Adresa de email
phone	String	Numărul de telefon
type	ClientType	Tipul (INDIVIDUAL/BUSINESS)
cui	String	Cod unic de identificare (pentru firme)

5.3 Entitatea Order

Reprezintă o comandă de service, legând un client de unul sau mai multe dispozitive.

Tabela 4: Structura entității Order

Câmp	Tip	Descriere
id	Long	Identificator unic (PK)
client	Client	Referință la client (FK)
status	OrderStatus	Statusul comenzii
createdAt	LocalDateTime	Data creării
deliveredAt	LocalDateTime	Data livrării
observations	String	Observații generale
estimatedAmount	BigDecimal	Suma estimată
deposit	BigDecimal	Avans plătit

5.4 Entitatea Device

Reprezintă un dispozitiv adus la reparat în cadrul unei comenzi.

Tabela 5: Structura entității Device

Câmp	**Tip**	**Descriere**
id	Long	Identificator unic (PK)
order	Order	Referință la comandă (FK)
deviceType	String	Tipul dispozitivului
brand	String	Marca
model	String	Modelul
serialNumber	String	Numărul de serie
problem	String	Problema raportată
status	DeviceStatus	Statusul reparației
accessories	String	Accesorii primite
observations	String	Observații specifice
hostname	String	Numele calculatorului (opțional)

5.5 Entitatea OrderLog

Reprezintă istoricul modificărilor unei comenzi.

Tabela 6: Structura entității OrderLog

Câmp	**Tip**	**Descriere**
id	Long	Identificator unic (PK)
order	Order	Referință la comandă (FK)
action	String	Acțiunea efectuată
details	String	Detalii suplimentare
createdAt	LocalDateTime	Data și ora acțiunii
createdBy	String	Utilizatorul care a efectuat acțiunea

6 Securitate

6.1 Autentificare JWT

Aplicația implementează un mecanism de autentificare bazat pe JSON Web Tokens (JWT), care oferă o soluție stateless pentru securizarea API-urilor RESTful.

6.1.1 Fluxul de autentificare

1. Utilizatorul trimite credențialele (username și password) către endpoint-ul `/api/users/login`
2. Backend-ul validează credențialele folosind `AuthenticationManager`
3. Dacă sunt valide, se generează un token JWT semnat cu o cheie secretă
4. Token-ul este returnat clientului și stocat în localStorage
5. Pentru fiecare cerere ulterioară, token-ul este trimis în header-ul `Authorization`
6. `JwtAuthFilter` interceptează cererile și validează token-ul

6.1.2 Structura Token-ului JWT

Token-ul JWT conține trei părți:

- **Header:** Algoritmul de semnare (HS256) și tipul token-ului
- **Payload:** Informații despre utilizator (username, rol, data expirării)
- **Signature:** Semnătura digitală pentru verificarea integrității

6.2 Configurarea Spring Security

Configurarea securității este realizată în clasa `SecurityConfig`:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
    Exception {
     http
          .csrf(csrf -> csrf.disable())
          .cors(cors -> cors.configurationSource(corsConfigurationSource()
    ))
          .authorizeHttpRequests(auth -> auth
              .requestMatchers("/api/users/login", "/api/users/register").
    permitAll()
              .requestMatchers("/api/public/**").permitAll()
              .requestMatchers("/ws/**").permitAll()
              .anyRequest().authenticated()
          )
          .sessionManagement(session ->
              session.sessionCreationPolicy(SessionCreationPolicy.
    STATELESS))
          .addFilterBefore(jwtAuthFilter,
              UsernamePasswordAuthenticationFilter.class);
     return http.build();
}
```

Listing 1: Configurarea SecurityFilterChain

6.3 Protecția rutelor în Frontend

În frontend, rutele protejate sunt gestionate prin componenta `ProtectedRoute`:

```
const ProtectedRoute = ({ children }) => {
  const { user, loading } = useAuth();

  if (loading) return <LoadingSkeleton />;
  if (!user) return <Navigate to="/login" />;

  return children;
};
```

Listing 2: Componenta ProtectedRoute

6.4 Criptarea parolelor

Parolele utilizatorilor sunt criptate folosind algoritmul BCrypt înainte de a fi stocate în baza de date:

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Listing 3: Bean pentru PasswordEncoder

7 API REST

7.1 Endpoint-uri principale

API-ul backend expune următoarele endpoint-uri RESTful:

7.1.1 Autentificare (/api/users)

Metodă	Endpoint	Descriere
POST	/api/users/login	Autentificare utilizator
POST	/api/users/register	Înregistrare utilizator nou
GET	/api/users	Listare utilizatori
PUT	/api/users/{id}	Actualizare utilizator
DELETE	/api/users/{id}	Ștergere utilizator

Tabela 7: Endpoint-uri pentru autentificare și management utilizatori

7.1.2 Comenzi (/api/orders)

Metodă	Endpoint	Descriere
GET	/api/orders	Listare comenzi (cu paginare și filtrare)
GET	/api/orders/{id}	Detalii comandă
POST	/api/orders	Creare comandă nouă
PUT	/api/orders/{id}	Actualizare comandă
PATCH	/api/orders/{id}/status	Schimbare status
DELETE	/api/orders/{id}	Ștergere comandă
POST	/api/orders/{id}/deliver	Marcare ca livrată

Tabela 8: Endpoint-uri pentru management comenzi

7.1.3 Clienți (/api/clients)

Metodă	Endpoint	Descriere
GET	/api/clients	Listare clienți (cu paginare)
GET	/api/clients/{id}	Detalii client
POST	/api/clients	Creare client nou
PUT	/api/clients/{id}	Actualizare client
DELETE	/api/clients/{id}	Ștergere client
GET	/api/clients/search	Căutare clienți

Tabela 9: Endpoint-uri pentru management clienți

7.1.4 Dispozitive (/api/devices)

Metodă	Endpoint	Descriere
GET	/api/devices	Listare dispozitive
GET	/api/devices/{id}	Detalii dispozitiv
POST	/api/devices	Adăugare dispozitiv
PUT	/api/devices/{id}	Actualizare dispozitiv
DELETE	/api/devices/{id}	Ștergere dispozitiv

Tabela 10: Endpoint-uri pentru management dispozitive

7.1.5 Endpoint-uri publice (/api/public)

Metodă	Endpoint	Descriere
GET	/api/public/order/{id}	Status public al comenzii
GET	/api/public/qr/{code}	Redirect din cod QR

Tabela 11: Endpoint-uri publice (fără autentificare)

7.2 Formatul răspunsurilor

Toate răspunsurile API folosesc formatul JSON. Pentru listări cu paginare, răspunsul include:

```
{
  "content": [...],         // Lista de elemente
  "totalElements": 100,     // Total elemente
  "totalPages": 10,         // Total pagini
  "size": 10,               // Elemente per pagina
  "number": 0               // Pagina curent  (0-indexed)
}
```

Listing 4: Structura răspunsului paginat

7.3 Gestionarea erorilor

Erorile sunt gestionate centralizat prin `GlobalExceptionHandler`:

```
{
  "timestamp": "2026-01-12T23:00:00",
  "status": 404,
  "error": "Not Found",
  "message": "Order not found with id: 123",
  "path": "/api/orders/123"
}
```

Listing 5: Structura răspunsului de eroare

8 Interfața Utilizator

8.1 Dashboard

Pagina principală oferă o vedere de ansamblu asupra activității service-ului:

- **Carduri statistice:** Total comenzi, comenzi noi, în lucru, finalizate
- **Grafic pie:** Distribuția comenzilor pe status
- **Grafic linie:** Evoluția comenzilor în ultimele 7 zile
- **Tabel:** Ultimele comenzi adăugate

8.2 Pagina Comenzi

Permite gestionarea completă a comenzilor de service:

- Listare cu paginare și sortare
- Filtrare după status, client, perioadă
- Căutare full-text
- Creare/editare comenzi cu modal
- Schimbare rapidă a statusului
- Vizualizare detalii și istoric
- Generare PDF și printare

8.3 Pagina Clienți

Gestionarea bazei de clienți:

- Listare cu paginare
- Filtrare după tipul clientului (Individual/Business)
- Căutare după nume, email, telefon, CUI
- Vizualizare comenzi asociate
- CRUD complet pentru clienți

8.4 Pagina Dispozitive

Evidența tuturor dispozitivelor:

- Listare cu toate dispozitivele din sistem
- Filtrare după status, tip, comandă
- Căutare după serie, model, brand
- Actualizare status și observații

8.5 Portal Client

Pagină publică accesibilă prin cod QR:

- Afișare status curent al comenzii
- Timeline cu istoricul modificărilor
- Detalii despre dispozitivele din comandă
- Design responsive pentru mobile

8.6 Skeleton Loading

Pentru o experiență utilizator îmbunătățită, aplicația implementează skeleton loading:

- Afișare placeholder animat în timpul încărcării
- Tipuri diferite: tabel, carduri, dashboard
- Reduce percepția timpului de așteptare

9 Comunicare în Timp Real

9.1 Implementare WebSocket

Aplicația folosește Spring WebSocket cu STOMP pentru comunicare bidirecțională:

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer
    {

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws")
                .setAllowedOriginPatterns("*")
                .withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }
}
```

Listing 6: Configurare WebSocket

9.2 Notificări pentru comenzi

Când o comandă este modificată, toți clienții conectați primesc notificări:

```
@Service
public class OrderService {
    private final SimpMessagingTemplate messagingTemplate;

    private void notifyOrderUpdate(String type, Long orderId) {
        OrderEventMessage event = new OrderEventMessage(type, orderId);
        messagingTemplate.convertAndSend("/topic/orders", event);
    }
}
```

Listing 7: Trimitere notificare WebSocket

9.3 Client WebSocket în React

Frontend-ul se conectează și ascultă pentru actualizări:

```
useEffect(() => {
    const client = createWsClient();
    client.onConnect = () => {
        client.subscribe('/topic/orders', (message) => {
            const event = JSON.parse(message.body);
            // Re ncarc  lista de comenzi
            fetchOrders();
        });
    };
    client.activate();
    return () => client.deactivate();
```

```
}, []);
```

Listing 8: Client WebSocket în React

10 Baza de Date și Migrări

10.1 Configurare MySQL

Conexiunea la baza de date este configurată în `application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3307/service_db
spring.datasource.username=dev
spring.datasource.password=devpass
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# HikariCP Connection Pool
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.connection-timeout=30000
```

Listing 9: Configurare bază de date

10.2 Migrări Flyway

Flyway gestionează versiunile schemei bazei de date:

Versiune	**Descriere**
V001	Actualizare câmp created_at la DATETIME
V002	Adăugare câmp hostname pentru dispozitive

Tabela 12: Migrări Flyway aplicate

10.3 Configurare JPA/Hibernate

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
    MySQL8Dialect
spring.jpa.properties.hibernate.jdbc.batch_size=20
spring.jpa.properties.hibernate.order_inserts=true
```

Listing 10: Configurare JPA

11 Concluzii

11.1 Realizări

Proiectul KIVA NET Service a atins toate obiectivele propuse inițial:

- **Arhitectură modernă:** Separare clară între frontend (React) și backend (Spring Boot)
- **Securitate robustă:** Autentificare JWT cu Spring Security
- **Comunicare în timp real:** WebSockets pentru notificări instant
- **Interfață intuitivă:** Design modern cu Bootstrap și skeleton loading
- **Scalabilitate:** Arhitectură pregătită pentru creștere

11.2 Tehnologii utilizate

Categorie	**Tehnologii**
Frontend	React, Bootstrap, Vite, Axios
Backend	Spring Boot, Spring Security, Spring Data JPA
Bază de date	MySQL, Flyway, HikariCP
Comunicare	REST API, WebSocket/STOMP, SockJS
Securitate	JWT, BCrypt
Build	Maven, npm

Tabela 13: Stack tehnologic utilizat

11.3 Îmbunătățiri viitoare

Pentru versiunile următoare se pot considera:

- Implementarea unui sistem de notificări push pentru mobil
- Adăugarea unui modul de raportare și statistici avansate
- Integrare cu sisteme de plată online
- Aplicație mobilă nativă (React Native)
- Implementarea unui chatbot pentru suport clienți