

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

# Managementul unui Service IT

Tuns Andrei  
12 ianuarie 2026

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>3</b>
1.1	Context general	3
1.2	Obiective	3
1.3	Lista MoSCoW	3
1.4	Cazuri de utilizare	4
<b>2</b>	<b>Arhitectura</b>	<b>5</b>
2.1	Front-End: React cu Bootstrap	5
2.2	Back-End: Spring Boot	5
2.3	Comunicare între Front-End și Back-End	6
2.4	Comunicare în timp real: WebSockets	6
2.5	Baza de Date	6
2.6	Beneficii ale acestei arhitecturi	6
<b>3</b>	<b>Implementare</b>	<b>6</b>
3.1	Tehnologii Frontend	6
3.2	Tehnologii Backend	7
3.3	Integrarea API-ului WhatsApp	7
3.4	Generare PDF și Cod QR	7
<b>4</b>	<b>Structura Proiectului</b>	<b>9</b>
4.1	Structura Backend	9
4.2	Structura Frontend	9
<b>5</b>	<b>Modelul de Date</b>	<b>11</b>
5.1	Entitatea User	11
5.2	Entitatea Client	11
5.3	Entitatea Order	11
5.4	Entitatea Device	12
5.5	Entitatea OrderLog	12
<b>6</b>	<b>Securitate</b>	<b>13</b>
6.1	Autentificare JWT	13
6.1.1	Fluxul de autentificare	13
6.1.2	Structura Token-ului JWT	13
6.2	Configurarea Spring Security	13
6.3	Protecția rutelor în Frontend	14
6.4	Criptarea parolelor	14
<b>7</b>	<b>API REST</b>	<b>15</b>
7.1	Endpoint-uri principale	15
7.1.1	Autentificare (/api/users)	15
7.1.2	Comenzi (/api/orders)	15
7.1.3	Clienți (/api/clients)	15
7.1.4	Dispozitive (/api/devices)	16
7.1.5	Endpoint-uri publice (/api/public)	16
7.2	Formatul răspunsurilor	16

7.3	Gestionarea erorilor . . . . .	16
<b>8</b>	<b>Interfața Utilizator</b>	<b>17</b>
8.1	Dashboard . . . . .	17
8.2	Pagina Comenzi . . . . .	17
8.3	Pagina Clienți . . . . .	17
8.4	Pagina Dispozitive . . . . .	17
8.5	Portal Client . . . . .	18
8.6	Skeleton Loading . . . . .	18
<b>9</b>	<b>Comunicare în Timp Real</b>	<b>19</b>
9.1	Implementare WebSocket . . . . .	19
9.2	Notificări pentru comenzi . . . . .	19
9.3	Client WebSocket în React . . . . .	19
<b>10</b>	<b>Baza de Date și Migrări</b>	<b>20</b>
10.1	Configurare MySQL . . . . .	20
10.2	Migrări Flyway . . . . .	20
10.3	Configurare JPA/Hibernate . . . . .	20
<b>11</b>	<b>Concluzii</b>	<b>21</b>
11.1	Rezumatul proiectului . . . . .	21
11.2	Obiective atinse . . . . .	21
11.3	Tehnologii folosite . . . . .	22
11.4	Posibile îmbunătățiri . . . . .	22
11.5	Concluzii finale . . . . .	22

# 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
<ul style="list-style-type: none"><li>• Autentificare utilizatori</li><li>• Management comenzi</li><li>• Management clienți</li><li>• Management dispozitive</li></ul>	<ul style="list-style-type: none"><li>• Portal client</li><li>• Generare PDF și cod QR</li><li>• Notificări automate (WhatsApp)</li><li>• Istoric reparații</li><li>• Căutare avansată</li></ul>
Could have	Won't have
<ul style="list-style-type: none"><li>• Programări online</li><li>• Rapoarte statistice</li><li>• Management inventar piese</li><li>• Chat intern</li></ul>	<ul style="list-style-type: none"><li>• Modul de facturare</li><li>• Integrare cu soft de contabilitate</li><li>• Suport multi-lingvistic</li></ul>

## 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.

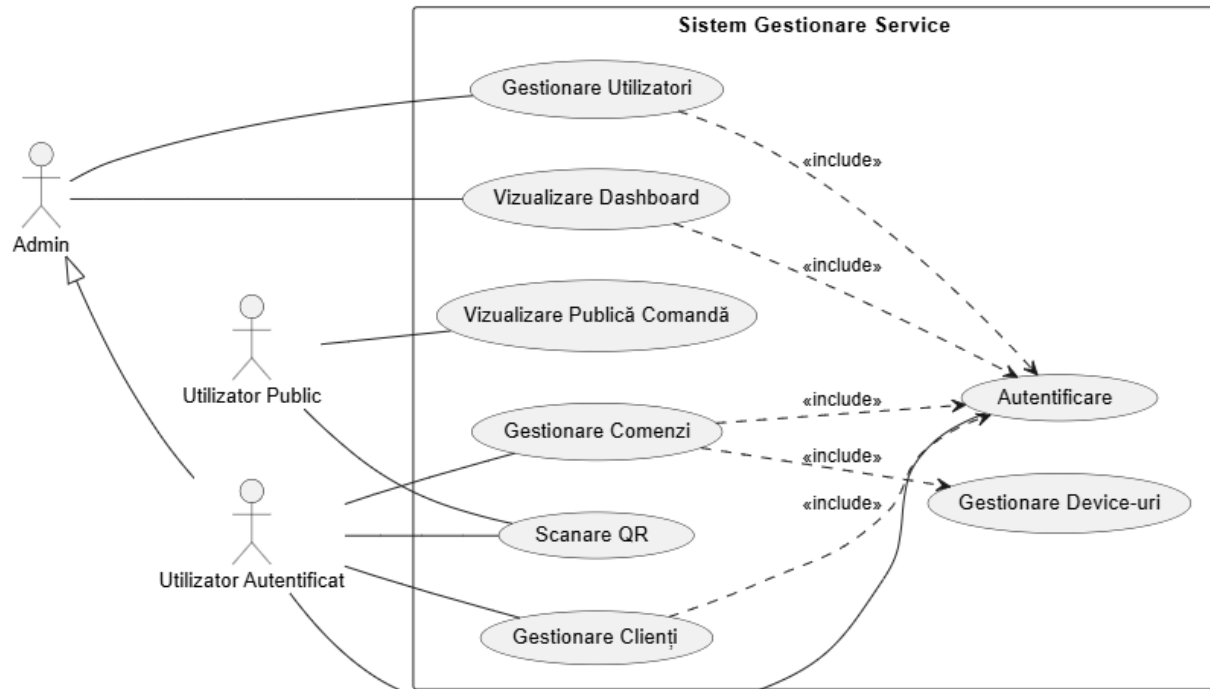


Figura 1: Diagrama cazurilor de utilizare pentru KIVA NET Service.



## 2.3 Comunicare între Front-End și 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 aceluși 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`:

```
1 @Bean
2 public SecurityFilterChain filterChain(HttpSecurity http) throws
   Exception {
3     http
4         .csrf(csrf -> csrf.disable())
5         .cors(cors -> cors.configurationSource(corsConfigurationSource(
6         )))
7         .authorizeHttpRequests(auth -> auth
8             .requestMatchers("/api/users/login", "/api/users/register").
9             permitAll()
10            .requestMatchers("/api/public/**").permitAll()
11            .requestMatchers("/ws/**").permitAll()
12            .anyRequest().authenticated()
13        )
14        .sessionManagement(session ->
15            session.sessionCreationPolicy(SessionCreationPolicy.
16            STATELESS))
17        .addFilterBefore(jwtAuthFilter,
18            UsernamePasswordAuthenticationFilter.class);
19     return http.build();
20 }
```

Listing 1: Configurarea `SecurityFilterChain`

## 6.3 Protecția rutelor în Frontend

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

```
1 const ProtectedRoute = ({ children }) => {  
2   const { user, loading } = useAuth();  
3  
4   if (loading) return <LoadingSkeleton />;  
5   if (!user) return <Navigate to="/login" />;  
6  
7   return children;  
8 };
```

Listing 2: Componenta `ProtectedRoute`

## 6.4 Criptarea parolelor

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

```
1 @Bean  
2 public PasswordEncoder passwordEncoder() {  
3   return new BCryptPasswordEncoder();  
4 }
```

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:

```
1 {
2   "content": [...],           // Lista de elemente
3   "totalElements": 100,      // Total elemente
4   "totalPages": 10,          // Total pagini
5   "size": 10,                 // Elemente per pagina
6   "number": 0                 // Pagina curent (0-indexed)
7 }
```

Listing 4: Structura răspunsului paginat

## 7.3 Gestionarea erorilor

Erorile sunt gestionate centralizat prin `GlobalExceptionHandler`:

```
1 {
2   "timestamp": "2026-01-12T23:00:00",
3   "status": 404,
4   "error": "Not Found",
5   "message": "Order not found with id: 123",
6   "path": "/api/orders/123"
7 }
```

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:

```
1 @Configuration
2 @EnableWebSocketMessageBroker
3 public class WebSocketConfig implements WebSocketMessageBrokerConfigurer
4 {
5     @Override
6     public void registerStompEndpoints(StompEndpointRegistry registry) {
7         registry.addEndpoint("/ws")
8             .setAllowedOriginPatterns("*")
9             .withSockJS();
10    }
11
12    @Override
13    public void configureMessageBroker(MessageBrokerRegistry registry) {
14        registry.enableSimpleBroker("/topic");
15        registry.setApplicationDestinationPrefixes("/app");
16    }
17 }
```

Listing 6: Configurare WebSocket

### 9.2 Notificări pentru comenzi

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

```
1 @Service
2 public class OrderService {
3     private final SimpMessagingTemplate messagingTemplate;
4
5     private void notifyOrderUpdate(String type, Long orderId) {
6         OrderEventMessage event = new OrderEventMessage(type, orderId);
7         messagingTemplate.convertAndSend("/topic/orders", event);
8     }
9 }
```

Listing 7: Trimitere notificare WebSocket

### 9.3 Client WebSocket în React

Frontend-ul se conectează și ascultă pentru actualizări:

```
1 useEffect(() => {
2     const client = createWsClient();
3     client.onConnect = () => {
4         client.subscribe('/topic/orders', (message) => {
5             const event = JSON.parse(message.body);
6             // Re ncarc lista de comenzi
7             fetchOrders();
8         });
9     };
10    client.activate();
11    return () => client.deactivate();
```

```
12 }, []);
```

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`:

```
1 spring.datasource.url=jdbc:mysql://localhost:3307/service_db
2 spring.datasource.username=dev
3 spring.datasource.password=devpass
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5
6 # HikariCP Connection Pool
7 spring.datasource.hikari.maximum-pool-size=10
8 spring.datasource.hikari.minimum-idle=5
9 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 <code>created_at</code> la <code>DATETIME</code>
V002	Adăugare câmp <code>hostname</code> pentru dispozitive

Tabela 12: Migrări Flyway aplicate

### 10.3 Configurare JPA/Hibernate

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

Listing 10: Configurare JPA

## 11 Concluzii

### 11.1 Rezumatul proiectului

Aplicația KIVA NET Service reprezintă o soluție completă pentru managementul unui service IT, dezvoltată utilizând tehnologii moderne și urmând cele mai bune practici din industrie. Proiectul demonstrează implementarea unei arhitecturi client-server robuste, cu separare clară între responsabilități.

Principalele componente dezvoltate includ:

- Un **backend RESTful** implementat cu Spring Boot, care gestionează logica de business, persistența datelor și securitatea aplicației
- Un **frontend SPA** (Single Page Application) dezvoltat cu React, oferind o interfață modernă și responsivă
- Un sistem de **autentificare și autorizare** bazat pe JWT (JSON Web Tokens)
- **Comunicare în timp real** prin WebSockets pentru notificări instant
- Integrare cu **servicii externe** (WhatsApp API, generare PDF și coduri QR)

### 11.2 Obiective atinse

Toate cerințele din categoria **Must have** au fost implementate cu succes:

- Autentificare și management utilizatori cu roluri diferențiate
- CRUD complet pentru comenzi, clienți și dispozitive
- Interfață intuitivă cu dashboard și statistici
- Portal public pentru clienți cu acces prin cod QR

De asemenea, funcționalitățile din categoria **Should have** au fost integrate:

- Generare automată de documente PDF cu cod QR
- Istoric complet al reparațiilor (OrderLog)
- Căutare și filtrare avansată
- Notificări în timp real prin WebSockets

### 11.3 Tehnologii folosite

Domeniu	Tehnologii utilizate
Backend	Spring Boot, Spring Security, Spring Data JPA, Spring WebSocket
Frontend	React 18, Bootstrap, Vite, Axios, SockJS/S-TOMP
Bază de date	MySQL 8.0, HikariCP, Flyway
Securitate	JWT, BCrypt, CORS
DevOps	Maven, npm, Git

### 11.4 Posibile îmbunătățiri

Pentru versiuni viitoare ale aplicației, se pot lua în considerare următoarele extensii:

- Implementarea unui modul de facturare și integrare cu sisteme de contabilitate
- Dezvoltarea unei aplicații mobile native pentru tehnicieni
- Adăugarea unui sistem de ticketing pentru suport tehnic
- Implementarea de rapoarte și analize avansate cu grafice interactive
- Suport pentru multiple locații/puncte de lucru
- Integrare cu sisteme de inventar pentru piese de schimb

### 11.5 Concluzii finale

Proiectul KIVA NET Service îndeplinește cu succes obiectivele propuse, oferind o platformă funcțională și scalabilă pentru managementul unui service IT. Arhitectura aleasă permite extinderea ușoară a funcționalităților, iar tehnologiile utilizate sunt actuale și bine documentate, facilitând mentenanța pe termen lung.

Aplicația demonstrează aplicarea practică a conceptelor teoretice studiate, de la proiectarea bazelor de date relaționale până la implementarea pattern-urilor de design în aplicații enterprise. Experiența acumulată în dezvoltarea acestui proiect constituie o bază solidă pentru proiecte viitoare de complexitate similară sau superioară.