📄 **Resourcify SRS (Software Requirements Specification)**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the system requirements for Resourcify, a web-based platform aimed at facilitating efficient resource management and sharing among communities, NGOs, and universities.

**1.2 Scope**

Resourcify allows users to view, request, donate, and track various resources across different locations. The system ensures efficient allocation and transparent tracking of resources, with features like request management, real-time updates via WebSockets, and Google Maps integration.

**1.3 Definitions, Acronyms, and Abbreviations**

| **Term** | **Definition** |
| --- | --- |
| UI | User Interface |
| API | Application Programming Interface |
| CRUD | Create, Read, Update, Delete |
| REST | Representational State Transfer |

**1.4 References**

* IEEE 830-1998 SRS Template
* Resourcify project documentation
* SRS and SDS content guides

**1.5 Overview**

This document contains functional, non-functional, and external interface requirements of Resourcify.

**2. Overall Description**

**2.1 Product Perspective**

Resourcify is an independent web-based system integrating a backend (Spring Boot & MySQL) and a frontend (HTML, CSS, JavaScript). It interacts with Google Maps API for resource location and WebSocket for real-time communication.

**2.2 Product Functions**

* User authentication (admin and normal users)
* CRUD operations on resources
* Request submission and management
* WebSocket-based real-time notifications
* Google Maps for resource locations
* Admin dashboard for monitoring activities

**2.3 User Characteristics**

| **User Role** | **Description** |
| --- | --- |
| Admin | Manages resources, views requests |
| Normal User | Requests and donates resources |

**2.4 Constraints**

* Dependent on internet connectivity
* Google Maps API key usage limits
* Backend requires MySQL and Spring Boot

**2.5 Assumptions and Dependencies**

* Users have basic web navigation knowledge
* Dependencies: Google Maps API, SockJS, STOMP, Spring Boot

**3. Specific Requirements**

**3.1 Functional Requirements**

* User registration and login
* Submit, update, and delete resource requests
* View available resources
* Track resource distribution on Google Maps
* Real-time updates via WebSocket

**3.2 Non-Functional Requirements**

* System uptime of 99%
* Response time < 2s
* Secure authentication with encrypted passwords

**3.3 External Interface Requirements**

* RESTful APIs for frontend-backend communication
* WebSocket for real-time updates
* MySQL for database management
* Google Maps API for location visualization

**3.4 Performance Requirements**

* Handle 100 concurrent users
* Load resource data within 3 seconds

**3.5 Software System Attributes**

* Security: JWT or Spring Security
* Usability: Intuitive frontend
* Scalability: Can add new features easily
* Reliability: Error handling and fallback mechanisms

**3.6 Database Requirements**

* MySQL relational database
* Tables: Users, Resources, Requests
* Foreign key relationships between requests and resources

**4. Appendices**

* ER Diagrams
* UI mockups
* Swagger/OpenAPI docs (optional future)

📄 **Resourcify SDS (Software Design Specification)**

**1. Introduction**

**Purpose**

This SDS describes the architectural design and detailed component design of the Resourcify system.

**Scope**

Covers backend architecture, frontend architecture, database design, WebSocket integration, and security considerations.

**Definitions, Acronyms, and Abbreviations**

(Refer to SRS)

**References**

* Resourcify SRS
* IEEE 1016-1998 SDS Template

**2. System Architecture Design**

**Overview**

* Backend: Spring Boot (REST APIs, WebSocket, JPA for persistence)
* Frontend: HTML, CSS, JavaScript
* Database: MySQL
* External APIs: Google Maps, SockJS, STOMP

**Architectural Diagram**

[Frontend] ---> [Backend REST APIs] ---> [MySQL Database]

|

[WebSocket (STOMP)]

|

[Frontend Listeners]

**Design Patterns**

* MVC for backend
* Observer pattern for WebSockets
* Singleton for DB connection (handled by Spring Boot)

**3. Component Design**

| **Component** | **Responsibility** |
| --- | --- |
| AuthController | User authentication & authorization |
| ResourceController | CRUD operations on resources |
| ResourceRequestController | Submit/view resource requests |
| WebSocketConfig | Configure WebSocket endpoints |

**Component Diagrams**

* REST controllers
* Services (business logic)
* Repositories (database access)

**Pseudocode/Flowcharts**

**Submit Request Flow:**

User submits request --> Validate resource quantity -->

Deduct quantity --> Save request --> Send WebSocket notification

**4. Interface Design**

**User Interfaces (Based on Project Files)**

* Home.html
* About.html
* Services.html
* Terms.html
* Login.html
* dashboard.html

**System Interfaces**

* REST APIs (JSON format)
* WebSocket (/ws) endpoint
* External: Google Maps API

**5. Data Design**

**Data Structures**

* ResourceItem (id, name, description, quantity, location)
* UserRequest (id, resourceId, userId, status, requestDate)

**Database Design**

* ERD connecting Users, Resources, Requests
* Relationships: One resource can have many requests

**6. Algorithm Design**

**Algorithms**

* Request validation and deduction
* Real-time notification via WebSocket
* Resource availability check

**Performance Considerations**

* Indexing on resource name and location
* Lazy loading where necessary

**7. Security and Privacy Considerations**

* Password encryption (BCrypt)
* Role-based access control
* CSRF protection (Spring Security)
* Data privacy on sensitive user info

**8. Error Handling and Logging**

* Custom exception handlers
* Logs stored using Spring Boot logging
* WebSocket errors logged separately

**9. Performance Considerations**

* Asynchronous calls where applicable
* Pagination for large datasets
* Efficient WebSocket handling

**10. Testing Strategy**

* Unit tests for service layer
* Integration tests for API endpoints
* Frontend testing via manual QA
* WebSocket testing with mock clients

**11. Deployment Strategy**

* Local deployment on Apache NetBeans
* Future: Docker containerization
* MySQL and Spring Boot dependencies

**12. Maintenance and Updates**

* Git for version control
* CI/CD pipeline for deployment (future)
* Bug tracking via issue boards

**13. Appendices**

* Sequence diagrams
* Screenshots of frontend
* Postman API collections (optional)

**Diagrams (Embedded/Attached)**

**ER Diagram**

(ERD connecting Users, Resources, Requests)

**Sequence Diagram**

1. Submit Request
2. Real-time Notifications
3. Resource Allocation

**API Endpoints Documentation**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/auth/register | POST | Register a new user |
| /api/auth/login | POST | Login and receive token |
| /api/resources | GET | Get all resources |
| /api/resources/{id} | GET | Get resource by ID |
| /api/resources | POST | Add a new resource |
| /api/resources/{id} | PUT | Update resource |
| /api/resources/{id} | DELETE | Delete resource |
| /api/requests | GET | Get all requests |
| /api/requests/{id} | GET | Get request by ID |
| /api/requests | POST | Submit a new request |
| /ws | WS | WebSocket endpoint for notifications |

End of Document.