## User Registration Service Documentation

### 1. Overview

A production-ready **User Registration Service** built using Java 21, Spring Boot, JPA, and Citus DB. The service exposes RESTful endpoints for user management with JWT authentication and partial updates support.

### 2. Tech Stack

* **Java 21**
* **Spring Boot 3.x**
* **Spring Data JPA / Hibernate**
* **Citus (PostgreSQL) DB**
* **Docker & Docker Compose**
* **JWT Authentication**
* **Lombok**
* **Validation with Jakarta Bean Validation**

### 3. Docker Setup & Commands

This section guides a user who downloads the project from the repository to run the application locally.

#### Step 1: Pull Citus Docker Image

docker pull citusdata/citus:11.2

* Pulls the Citus 11.2 image from Docker Hub.

#### Step 2: Prepare Docker Compose

Create docker-compose.yml in project root:

version: '3.8'  
services:  
 citus\_master:  
 image: citusdata/citus:11.2  
 container\_name: citus\_master  
 environment:  
 POSTGRES\_PASSWORD: citus\_master  
 ports:  
 - "5432:5432"  
 volumes:  
 - ./init.sql:/docker-entrypoint-initdb.d/init.sql  
  
 citus\_worker:  
 image: citusdata/citus:11.2  
 container\_name: citus\_worker  
 environment:  
 POSTGRES\_PASSWORD: citus\_worker

#### Step 3: Initialization SQL (init.sql)

CREATE DATABASE userdb;  
\c userdb  
CREATE EXTENSION IF NOT EXISTS citus;  
  
CREATE TABLE IF NOT EXISTS users (  
 id SERIAL NOT NULL,  
 username TEXT NOT NULL,  
 email TEXT NOT NULL,  
 password TEXT NOT NULL  
);  
  
SELECT create\_distributed\_table('users', 'id');  
  
INSERT INTO users (username, email, password) VALUES  
('sankar', 'sa@ad.com', 'Welcome@12345'),  
('david', 'da@ad.com', 'Welcome@12345');

#### Step 4: Start Docker Containers

docker-compose up -d

* Starts master and worker containers in detached mode.
* Check running containers: docker ps

#### Step 5: Access the Database

docker exec -it citus\_master psql -U postgres -d userdb

* Connects to userdb inside the master container.
* Run SQL commands manually if needed.

#### Step 6: Stop and Remove Containers

docker-compose down -v

* Stops containers, removes networks, and volumes.

#### Step 7: Remove a Stopped Container (if needed)

docker rm -f citus\_master citus\_worker

* Forces removal of containers if there’s a conflict or leftover.

#### Step 8: Troubleshoot Docker Issues

| Issue | Fix |
| --- | --- |
| Container exits immediately | Check logs: docker logs <container\_name>; verify environment variables and exposed ports |
| Container name conflict | Remove previous containers: docker rm -f citus\_master citus\_worker |
| SQL initialization errors | Ensure CREATE EXTENSION citus is executed before create\_distributed\_table |
| Cannot connect to DB | Ensure port 5432 is available, use docker exec or GUI client with proper credentials |

### 4. Spring Boot Configuration

**application.yml Example:**

spring:  
 datasource:  
 url: jdbc:postgresql://localhost:5432/userdb  
 username: postgres  
 password: citus\_master  
 jpa:  
 hibernate:  
 ddl-auto: none  
 show-sql: true

### 5. Entity Classes

**User Entity:**

@Entity  
@Table(name = "users")  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(nullable = false)  
 private String username;  
  
 @Column(nullable = false)  
 private String email;  
  
 @Column(nullable = false)  
 private String password;  
}

### 6. REST Endpoints

| Method | Endpoint | Description | Request Body | Response |
| --- | --- | --- | --- | --- |
| POST | /api/users | Create new user | {username, email, password} | User object with id |
| GET | /api/users/id/{id} | Get user by ID | None | User object |
| GET | /api/users/username/{username} | Get user by username | None | User object |
| PATCH | /api/users/{id} | Partial update | Fields to update (username/email/password) | Updated User object |
| DELETE | /api/users/{id} | Delete user | None | Success message |

### 7. DTOs

* **UserRequestDTO:** For creation; only non-nullable fields.
* **UserPatchDTO:** For PATCH; optional fields.
* **UserResponseDTO:** For responses; excludes sensitive info like password.

### 8. JWT Security

* Endpoints secured using JWT.
* Send Authorization: Bearer <token> for protected endpoints.

### 9. Postman Collection

* **Base URL:** http://localhost:8080/api/users
* Endpoints: POST, GET by ID, GET by username, PATCH, DELETE
* Include JWT header if security enabled.

### 10. Notes / Recommendations

* Use **DTOs** for request/response separation.
* Disable ddl-auto for Citus tables; schema changes must be manual.
* Use PATCH DTO or BeanUtils.copyProperties to handle partial updates.
* Do not expose passwords in API responses.
* Use Docker volumes to persist data.

### 11. Common Issues & Solutions

| Issue | Solution |
| --- | --- |
| Docker container exits immediately | docker logs <container\_name>; verify environment vars and ports |
| create\_distributed\_table fails | Run CREATE EXTENSION citus before creating distributed table |
| Primary key/unique constraint errors | Do not define constraints that do not include the partition column |
| Hibernate ALTER TABLE errors | Set spring.jpa.hibernate.ddl-auto=none; manage schema manually |
| Ambiguous handler methods in Spring Boot | Ensure @GetMapping paths for ID and username are distinct |
| PATCH updates overwrite null values | Use BeanUtils.copyProperties or MapStruct to copy only non-null fields |
| Maven compile errors with Java 21 | Ensure Maven compiler plugin set to Java 21; dependencies compatible |
| Cannot connect GUI to Citus DB | Use PostgreSQL-compatible client (DBeaver, pgAdmin) to connect to citus\_master on exposed port |

**Repository ready for GitHub:** Include docker-compose.yml, init.sql, pom.xml, src folder with entities, DTOs, service, repository, controller, security config, and Postman collection JSON.