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

Step 1: Pull Citus Docker Image

```
docker pull citusdata/citus:11.2
```

Step 2: Prepare Docker Compose (`docker-compose.yml`)

```yaml
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`)

```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
```

Step 5: Access the Database

```
docker exec -it citus_master psql -U postgres -d userdb
```

Step 6: Stop and Remove Containers

```
docker-compose down -v
```

Step 7: Remove Stopped Containers (if needed)

```
docker rm -f citus_master citus_worker
```

Step 8: Troubleshoot Docker Issues

Issue	Fix
Container exits immediately	Check logs: `docker logs <container_name>`; verify environment variables and 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

---

4. Spring Boot Configuration

`application.yml` **Example:**

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/userdb
    username: postgres
    password: citus_master
  jpa:
    hibernate:
      ddl-auto: none
    show-sql: true
```

---

5. Java Code Snippets

Entity: `User.java`

```java
package com.ad.user.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@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;
}
```

Repository: `UserRepository.java`

```
package com.ad.user.repository;

import com.ad.user.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

Service: `UserService.java`

```
package com.ad.user.service;

import com.ad.user.entity.User;
import com.ad.user.repository.UserRepository;
import org.springframework.beans.BeanUtils;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class UserService {

    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
```

```
    }

    public User createUser(User user) {
        return userRepository.save(user);
    }

    public User getUser(Long id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("User not found"));
    }

    public Optional<User> getUserByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    public User updateUser(Long id, User updatedUser) {
        User user = getUser(id);
        BeanUtils.copyProperties(updatedUser, user, "id");
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

Controller: `UserController.java`

```
package com.ad.user.controller;

import com.ad.user.entity.User;
import com.ad.user.service.UserService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping
```

```java
    public ResponseEntity<User> createUser(@RequestBody User user) {
        return ResponseEntity.ok(userService.createUser(user));
    }

    @GetMapping("/id/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        return ResponseEntity.ok(userService.getUser(id));
    }

    @GetMapping("/username/{username}")
    public ResponseEntity<User> getUserByUsername(@PathVariable String
username) {
        return ResponseEntity.ok(userService.getUserByUsername(username)
                .orElseThrow(() -> new RuntimeException("User not found")));
    }

    @PatchMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody
User updatedUser) {
        return ResponseEntity.ok(userService.updateUser(id, updatedUser));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteUser(@PathVariable Long id) {
        userService.deleteUser(id);
        return ResponseEntity.ok("User deleted successfully");
    }
}
```

DTO Example: `UserRequestDTO.java`

```java
package com.ad.user.dto;

import lombok.Data;

@Data
public class UserRequestDTO {
    private String username;
    private String email;
    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	Updated User object
DELETE	/api/users/{id}	Delete user	None	Success message

---

7. JWT Security

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

---

8. 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.

---

9. Common Issues & Solutions

Issue	Solution
Docker container exits immediately	`docker logs <container_name>`; verify environment vars and ports
Container name conflict	Remove previous containers: `docker rm -f citus_master citus_worker`
`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

Issue	Solution
Hibernate `ALTER TABLE` errors	Set `spring.jpa.hibernate.ddl-auto=none`; manage schema manually
Ambiguous handler methods	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) on `citus_master` 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.