Detailed Project Plan for Team 2: Configuration Database Microservice

Responsibilities

- Store current and historical configuration data of network devices.
- Implement version control and change tracking mechanisms.

Day 1: Initial Setup and Development

1.1 Set Up the Development Environment

- **Install Necessary Tools:**
  - Java Development Kit (JDK)
  - Maven or Gradle for project dependency management
  - IDE (e.g., IntelliJ IDEA, Eclipse)
  - Git for version control
  - Database management system (e.g., PostgreSQL, MySQL)
  - ORM framework (e.g., Hibernate)
- **Create Project Structure:**
  - Initialize a new Maven or Gradle project.
  - Set up a Git repository and push the initial project structure.
- **Install Libraries:**
  - Add dependencies for the database driver, ORM framework, and any necessary utilities in the pom.xml (for Maven) or build.gradle (for Gradle).

xml

Copy code

```xml
<!-- Example for Maven -->
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <version>42.2.18</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>5.4.30.Final</version>
</dependency>
```

1.2 Review Version Control Mechanisms

- **Version Control in Databases:**
  - Understand concepts of version control and change tracking in databases.
  - Review strategies such as temporal tables, audit tables, and versioned entities.

1.3 Implement the Configuration Database Microservice

- **Design Database Schema:**
  - Create tables to store current and historical configuration data.

    - Implement version control mechanisms using temporal tables or versioned entities.

sql
Copy code
```
CREATE TABLE configurations (
   id SERIAL PRIMARY KEY,
   device_id VARCHAR(255) NOT NULL,
   config_data JSONB NOT NULL,
   version INTEGER NOT NULL,
   created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE config_versions (
   id SERIAL PRIMARY KEY,
   config_id INTEGER REFERENCES configurations(id),
   device_id VARCHAR(255) NOT NULL,
   config_data JSONB NOT NULL,
   version INTEGER NOT NULL,
   created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```
- **Implement ORM Entities:**
    - Map the database tables to Java entities using Hibernate.

java
Copy code
```
@Entity
@Table(name = "configurations")
public class Configuration {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(name = "device_id", nullable = false)
   private String deviceId;

   @Column(name = "config_data", nullable = false)
   private String configData;

   @Column(name = "version", nullable = false)
   private int version;

   @Column(name = "created_at", nullable = false)
   private LocalDateTime createdAt;
```

```
    // Getters and setters
}

@Entity
@Table(name = "config_versions")
public class ConfigVersion {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "config_id")
    private Configuration configuration;

    @Column(name = "device_id", nullable = false)
    private String deviceId;

    @Column(name = "config_data", nullable = false)
    private String configData;

    @Column(name = "version", nullable = false)
    private int version;

    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    // Getters and setters
}
```

- **Implement Repository Interfaces:**
    - Create repository interfaces for accessing the database.

java
Copy code
```
public interface ConfigurationRepository extends JpaRepository<Configuration, Long> {
    List<Configuration> findByDeviceId(String deviceId);
}

public interface ConfigVersionRepository extends JpaRepository<ConfigVersion, Long> {
    List<ConfigVersion> findByDeviceId(String deviceId);
}
```

Day 2: Integration and Testing

2.1 Test Integration with the Device Configuration Microservice

- **Set Up REST Endpoints:**
  - Implement RESTful endpoints to interact with the Device Configuration Microservice.
  - Define endpoints for creating, reading, updating, and deleting configurations.

java
Copy code
```
@RestController
@RequestMapping("/api/configurations")
public class ConfigurationController {
   @Autowired
   private ConfigurationService configurationService;

   @PostMapping
   public ResponseEntity<Configuration> createConfiguration(@RequestBody Configuration
configuration) {
      Configuration savedConfig = configurationService.saveConfiguration(configuration);
      return ResponseEntity.ok(savedConfig);
   }

   @GetMapping("/{deviceId}")
   public ResponseEntity<List<Configuration>> getConfigurations(@PathVariable String
deviceId) {
      List<Configuration> configurations = configurationService.getConfigurations(deviceId);
      return ResponseEntity.ok(configurations);
   }
}
```

2.2 Validate Data Storage and Version Control Functionalities

- **Unit Testing:**
  - Write unit tests for repository methods and service layer using JUnit or TestNG.
  - Ensure that version control mechanisms are correctly implemented and tested.
- **Integration Testing:**
  - Test the integration with the Device Configuration Microservice.
  - Ensure that configuration data is correctly stored and versioned.

java
Copy code
```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
```

```
@SpringBootTest
public class ConfigurationServiceTest {
   @Autowired
   private ConfigurationService configurationService;

   @Test
   public void testSaveConfiguration() {
      Configuration config = new Configuration();
      config.setDeviceId("device1");
      config.setConfigData("{...}");
      config.setVersion(1);

      Configuration savedConfig = configurationService.saveConfiguration(config);
      assertEquals("device1", savedConfig.getDeviceId());
   }
}
```

Day 3: System Integration and Documentation

3.1 Assist with System Integration

- **Collaborate with Other Teams:**
  - Work closely with the Device Configuration Microservice team to ensure seamless data flow.
  - Verify that the Configuration Database Microservice is correctly integrated into the overall system.

3.2 Comprehensive Testing

- **End-to-End Testing:**
  - Perform end-to-end testing across all integrated components.
  - Validate the overall functionality, performance, and reliability of the system.
- **User Acceptance Testing (UAT):**
  - Conduct UAT to ensure the system meets the requirements and expectations of end-users.

3.3 Documentation and Demonstration Preparation

- **Document System Design and Implementation:**
  - Create detailed documentation for the Configuration Database Microservice.
  - Include code snippets, API specifications, and usage instructions.
- **Prepare Demonstration:**
  - Develop a presentation to demonstrate the microservice's capabilities.
  - Highlight key features, integration points, and data version control mechanisms.

Conclusion

By following this detailed project plan, Team 2 can successfully develop the Configuration Database Microservice, ensuring it stores current and historical configuration data with robust version control and change tracking mechanisms. Integration with the Device Configuration

Microservice and comprehensive testing will ensure a reliable and effective solution for network configuration management.