Detailed Project Plan for Team 1: Device Configuration Microservice

Responsibilities

- Interface with network devices using NETCONF and RESTCONF protocols.
- Fetch and apply configuration data.
- Use libraries such as Netconf4J for NETCONF and Apache HttpClient for RESTCONF API interactions.

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
- **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 Netconf4J and Apache HttpClient in the pom.xml (for Maven) or build.gradle (for Gradle).

xml

```
<!-- Example for Maven -->
<dependency>
    <groupId>org.opendaylight.netconf</groupId>
    <artifactId>netconf4j</artifactId>
    <version>1.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.13</version>
</dependency>
```

1.2 Review NETCONF and RESTCONF Protocols

- **NETCONF:**
  - Understand the basic operations: <get-config>, <edit-config>, <get>, <rpc>.
  - Review the structure of NETCONF messages and their XML format.
- **RESTCONF:**
  - Understand the RESTCONF operations: GET, POST, PUT, DELETE.
  - Review the structure of RESTCONF requests and responses, focusing on JSON and XML formats.

1.3 Implement Basic NETCONF Operations

- **Create a NETCONF Client:**
  - Use Netconf4J to establish a session with a network device.
  - Implement basic operations to fetch configuration data.

java

```
import org.opendaylight.netconf.client.NetconfClient;
import org.opendaylight.netconf.client.NetconfClientSession;
import org.opendaylight.netconf.client.NetconfClientConfiguration;
import org.opendaylight.netconf.client.mina.NetconfClientMinaImpl;

public class NetconfExample {
    public static void main(String[] args) {
        NetconfClientConfiguration clientConfig = ... // Configure client settings
        NetconfClient client = new NetconfClientMinaImpl();
        NetconfClientSession session = client.connect(clientConfig);

        String getConfigRequest = "<rpc message-id=\"1\"><get-config><source><running/></
source></get-config></rpc>";
        String response = session.sendRpc(getConfigRequest);

        System.out.println("Response: " + response);
    }
}
```

1.4 Implement Basic RESTCONF Operations

- **Create a RESTCONF Client:**
  - Use Apache HttpClient to interact with a network device's RESTCONF API.
  - Implement basic operations to fetch configuration data.

java

```
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class RestconfExample {
    public static void main(String[] args) throws Exception {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpGet request = new HttpGet("http://<device-ip>/restconf/data/ietf-
interfaces:interfaces");
        request.addHeader("Accept", "application/yang-data+json");

        String response = httpClient.execute(request, responseHandler -> {
```

```
            int status = responseHandler.getStatusLine().getStatusCode();
            return status == 200 ? EntityUtils.toString(responseHandler.getEntity()) : null;
        });

        System.out.println("Response: " + response);
        httpClient.close();
    }
}
```

1.5 Develop Support for Configuration Changes and Fetches

- **Extend NETCONF Client:**
  - Implement <edit-config> operation to apply configuration changes.

```
java

String editConfigRequest = "<rpc message-id=\"2\"><edit-config><target><running/></
target><config>...</config></edit-config></rpc>";
String editResponse = session.sendRpc(editConfigRequest);
```

- **Extend RESTCONF Client:**
  - Implement POST, PUT, and DELETE methods to apply configuration changes.

```
java

import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;

public class RestconfPostExample {
    public static void main(String[] args) throws Exception {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpPost request = new HttpPost("http://<device-ip>/restconf/data/ietf-
interfaces:interfaces");
        request.addHeader("Content-Type", "application/yang-data+json");
        String jsonConfig = "{ ... }"; // JSON formatted configuration
        request.setEntity(new StringEntity(jsonConfig));

        httpClient.execute(request);
        httpClient.close();
    }
}
```

Day 2: Integration and Testing

2.1 Integrate with the Configuration Database Microservice

- **Set Up Communication:**
  - Define REST endpoints or RPC methods to interact with the Configuration Database Microservice.

    - Implement methods to read and write configuration data to the database.

2.2 Test and Validate Core Functionalities

- **Unit Testing:**
    - Write unit tests for NETCONF and RESTCONF operations using JUnit or TestNG.
    - Mock network devices for testing purposes.
- **Integration Testing:**
    - Test end-to-end functionality by fetching and applying configurations to actual or simulated devices.
    - Validate that data is correctly stored and retrieved from the Configuration Database Microservice.

java

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class NetconfClientTest {
    @Test
    public void testGetConfig() {
        String response = netconfClient.getConfig();
        assertEquals(expectedConfig, response);
    }
}
```

Day 3: System Integration and Documentation

3.1 Assist with System Integration

- **Collaborate with Other Teams:**
    - Ensure seamless integration of the Device Configuration Microservice with the other microservices.
    - Verify data flow and communication between microservices.

3.2 Comprehensive Testing

- **End-to-End Testing:**
    - Perform comprehensive tests across all integrated components.
    - Validate the overall system functionality, performance, and reliability.
- **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 Device Configuration 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 real-time configuration changes.

Conclusion

By following this detailed project plan, Team 1 can successfully develop the Device Configuration Microservice, ensuring it interfaces effectively with network devices using NETCONF and RESTCONF protocols. The integration with the Configuration Database Microservice and the comprehensive testing will guarantee a robust and reliable solution for network configuration management.