Project Documentation: Zoo Management System

1.Introduction

The **Zoo Management System** project is designed to handle information related to zoos, animals, and their conservation status. By leveraging various XML-based technologies, the system enables efficient storage, transformation, and querying of data. The application combines a well-formed XML file with validation schemas, XSLT transformations, XQuery queries, a REST API, and a SOAP API to ensure comprehensive management and presentation of the data.

2. Project Structure

2.1 Main XML File (zoo.xml)

The XML file contains information about:

- **Zoos:** includes attributes such as id, name, city, foundation, and location.
- **Animals:** stores data such as id, species, zooid, name, scientific_name, habitat, and diet.
- **Conservation Statistics:** related to animals, includes details on population_in_wild, population_in_capacity and status.
- **Messages:** news or announcements related to zoos or animals.

The hierarchical structure ensures clear and accessible organization of the data.

3. XML Validation

The zoo.xml file is validated using the XML schema defined in zoo.xsd. This ensures that the data adheres to the specified structural and semantic rules, such as correct associations between zoos and animals.

The validation process is automated using the validate_xml.py script, which employs Python's lxml library to verify conformity with the schema.

4. XSLT Transformations

The system uses XSLT to transform data into more readable or visually appealing formats. The transformations include:

- **zoos_to_html.xsl**: generates an HTML table with information about zoos and their animals.
- **herbivores_to_html.xsl**: Creates a list of herbivorous animals.
- **animals_by_habitat.xsl**: Groups animals by habitat.
- **endangered_animals.xsl**: Displays a report of endangered animals.
- **animals_count_by_zoo.xsl**: Shows a count of animals per zoo.

The transformations.py script automates these transformations, generating the corresponding HTML files.

5. eXistDB Queries

For advanced queries, eXistDB is used in conjunction with existdb_queries.py, which defines the following Xquery queries:

1. List of herbivorous animals
2. Information about zoos
3. Animals at risk of extinction
4. Count of animals per zoo
5. Animals in specific habitats.

These queries enable quick and efficient retrieval of relevant information.

6. API Interface

6.1. api.py

This is the REST API that is used to integrate with Xforms to invoke different operations that utilize said API.

- Retrieve animals filtered by zoo and species using query parameters.

6.2. API REST

The Flask application defines CRUD routes for zoos, animals, and conservation statistics. These functionalities are backed by functions in xml_utils.py for XML file manipulation.

The xml_utils.py file includes key functions such as:

- Zoos: retrieve, add, update and delete zoos
- Animals: manage animal information
- Statistics: handle conservation statistics

These utilities ensure efficient and secure data manipulation.

6.2. SOAP API

The SOAP API provides a structured interfaced for interacting with de system's data. Id supports operations such as:

- **Zoo Operations:**
  - Adding, updating, deleting and retrieving zoos by ID
  - Listing all zoos
- **Animal Operations:**
  - Adding, updating, deleting, and retrieving animals by ID.

    - Listing all animals.
- **Conservation Statistics Operation:**
    - Adding, updating, deleting, and retrieving conservation statistics by animal ID and year.
    - Listing all conservation statistics.

Unit tests for the SOAP are provided in test_zoo.py, test_animal.py, and test_statistics.py, ensuring the correctness and reliability of the operations.

7.Enterprise Integration Patterns (EIPs) used

7.1. Content-Based Router

- **Description**: This pattern routes messages to different destinations based on the content of the message (e.g., query parameters such as type, habitat, or zooName).
- **Where it is used**: In the direct:filterAnimals route, Camel evaluates the type, habitat, and zooName inputs to dynamically construct an XPath filter and select only the relevant data from the XML.
- **Related Code**:

```
if (type != null) {

  xpathQuery.append("@species='").append(type).append("'");

}

if (habitat != null) {

  xpathQuery.append("habitat='").append(habitat).append("'");

}

if (zooName != null) {

  xpathQuery.append("@zooid=//zoo[name='").append(zooName).append("']/@id");
```

7.2. Message Translator

- **Description**: This pattern transforms the format of a message. In this project, it transforms a full XML file into a filtered XML with relevant data.
- **Where it is used**: In the transformation routes direct:allAnimals and direct:filterAnimals, where XML is manipulated using XPath to retrieve specific nodes and generate a custom response.

- **Related Code**:

```
var nodeList = (org.w3c.dom.NodeList) xpath.evaluate(xpathQuery.toString(), xmlTree,
XPathConstants.NODESET);

StringBuilder response = new StringBuilder("<response><animals>");

for (int i = 0; i < nodeList.getLength(); i++) {

   response.append("<animal>")

         .append("<name>").append(name).append("</name>")

         .append("<species>").append(species).append("</species>")

         .append("</animal>");

}
```

7.2. REST Endpoint

- **Description**: This pattern is used to expose data through REST services. In this project, the REST endpoints (/zoo/animals and /zoo/animals/filter) allow interaction with Camel logic and the XML file content.
- **Where it is used**: In the REST configuration to expose routes as HTTP-accessible APIs.
- **Related Code**:

```
restConfiguration().component("netty-http").host("localhost").port(8081);

rest("/zoo")

   .get("/animals").to("direct:allAnimals")

   .get("/animals/filter").to("direct:filterAnimals");
```

8. XML-Based Technologies Documented

8.1 XPath

Used to dynamically query and filter the zoo.xml file based on parameters such as type, habitat, and zooName.

8.2 DOM Parsing

Used to load and manipulate the XML file in memory, accessing nodes and attributes.

8.3 Apache Camel Rest DSL

Facilitates the creation of REST APIs to interact with XML data through integration routes.

9.Web Interface

The interface defined in index.html allows to interact with the system intuitively. Some functionalities include:

- Listing zoos and animals
- Adding new entries
- Displaying detailed information

10. Generated Outputs

The system produces results in HTML and XML formats:

- Files generated through XSLT transformations
- API REST responses
- Outputs from Xquery queries executed in eXistDB.