Data ScienceTech Institute

Data Pipeline Part I

Mira BEN HALIMA - Favour OKIKE - KAKY SUZY Joelly Magalie - Kokou Aubain VIGLO - Real Estate Platform Report

Github repository :

https://github.com/DSTI-XMLProject/Collaborative-Real-Estate-Platform.git

Introduction :

This report describes the creation of an XML database for a collaborative real estate platform connecting users like buyers, sellers, agents and agencies. The XML Schema defines a structured way to organize this information. We worked mainly in VS Code using the XML Tools extension for editing and validation, and used an online XSLT processor for testing transformations. Joelly and Aubain created the XML Schema and database. Mira developed four transformation scenarios and their outputs , and Aubain did the 5th scenario with Python. The 6th scenario was done by Joelly and Favour, and Favour also handled the 7th scenario in JSON. Contributions were equally split: 25% each.

Our schema defines several key parts, including users, properties, transactions, agencies, listings, reviews, and appointments. Our platform includes different users with different roles. For example **User 9**: Isabelle Morel, a **buyer**, is registered with the email *isabelle.morel@example.com*. Isabelle is currently inactive, as indicated by the isActive field being set to false. She is located at *100 Rue de Lyon, Lyon, France*. **User 10**: Julien Garcia, a **seller**, is registered with the email *julien.garcia@example.com*. Julien is an active user, as the isActive field is true. His address is *77 Boulevard de la République, Nantes, France*.

The platform also manages different types of properties, each with unique characteristics. For example : **Property 101**: A **3-room apartment** in the heart of Paris, listed at a price of **350,000 EUR**. The property features a **balcony** and has a view of the Eiffel Tower. It is located at *45 Rue de Rivoli, Paris, France*. **Property 102**: A **large house** with a garden located on the outskirts of Lyon, priced at **450,000 EUR**. The house includes a **garage** and is situated at *12 Rue des Fleurs, Lyon, France*.

The platform tracks various types of transactions, such as purchases, rentals, and sales. For example:**Transaction 1**: A **purchase** of Property 101 by User 1 from User 2, with an agent (User 3) facilitating the transaction. The total amount of **350,000 EUR** was paid via **bank transfer**, and the transaction is marked as **completed**. **Transaction 4**: A **purchase** of Property 104 by User 2 from User 6, with User 7 acting as the agent. The transaction is currently **pending**, with a total amount of **800,000**

The real estate platform also manages agencies that list properties. For example:**Agency 1**: *Paris Real Estate Agency* is located at *10 Rue de la Paix, Paris, France*. The agency can be contacted via the phone number **0147852369** or through their website www.parisimmobilier.com.

The platform allows properties to be listed with specific statuses. For example:**Listing 1**: Property 101 is listed by User 3, and the status is **active**. The description reads, "Ad for a bright apartment in the heart of Paris."**Listing 3**: Property 103 was listed by User 3 but is marked as **expired**. The description reads, "Advertisement for a luxury villa with a swimming pool in Cannes."

The platform also tracks user reviews for both properties and agents. For example:**Review 1**: A review by User 1 for Property 101, with a rating of **4.0** out of 5. The comment states, "Very nice apartment, but a bit expensive."**Review 2**: A review by User 4 for Agent 2 (User 5), with a rating of **4.5** out of 5. The comment reads, "Excellent agent, very responsive and professional."

Appointments are scheduled for property viewings. For example: **Appointment 1**: A **confirmed** appointment for User 4 to view Property 105 with Agent 7, scheduled for **2023-08-15** at **10:00 AM**. **Appointment 2**: A **pending** appointment for User 10 to view Property 108 with Agent 3, scheduled for **2023-09-10** at **2:30 PM**.

When we designed the XML schema for our real estate platform, we wanted it to be clear and easy to update. A good part of our choices was using reusable blocks like for addresses and user info—this helped avoid repetition. For example, we used the same address type for users, properties, and agencies. We also connected elements with unique IDs to keep the data well organized. However, the schema can be a bit complex for new users, and XML files tend to get large because of all the tags. It's also hard to set some rules, like making sure a buyer doesn't act as a seller in the same deal.One problem we had was how to show users with different roles. At first, we thought of making separate user types, but that would be messy. Instead, we created one "User" type with a "roles" section. For example, Isabelle Morel is a user with the role of "buyer." This way, it's simpler, cleaner, and easier to manage people with one or more roles.

Scenario 7: Export User Data to JSON Format

The goal of the **7th scenario** was to **export user data** from the XML database of the real estate platform into a **structured JSON format**.
This transformation was necessary to support the integration of the platform's user data into modern web applications and APIs, where JSON is the preferred data exchange format due to its lightweight and easily parsable structure.

To achieve this transformation, an **XSLT 1.0 stylesheet** was developed that extracts user information from the XML database and formats it into valid JSON.

The XSLT program specifically outputs: An array called "users" where each item is a **user object**. Each user object includes detailed user information such as: id (integer), name (full name, combining first and last names), email, phoneNumber, role (e.g., buyer, seller, agent), isActive (boolean status), rating (decimal value or null if missing), registrationDate, profilePicture (URL or null if missing), address (nested object with fields for street, city, state, postal code, country, latitude, and longitude). Additionally, If optional fields like rating, profilePicture, latitude, or longitude are missing, the XSLT ensures that **null values** are inserted correctly in the output. A **comma** is automatically placed between JSON objects, but not after the last object to ensure valid JSON syntax.

Complexity of the Solution

This scenario required addressing multiple complex challenges during the implementation:

1. **Mismatch between XML and JSON structures**: XSLT templates were designed carefully to construct JSON objects manually since XSLT 1.0 was not originally built for JSON output.
2. **Handling Optional Fields**: xsl:choose, xsl:when, and xsl:otherwise instructions were used to check for missing XML elements like rating or profilePicture, and insert null into the JSON when absent.
3. **Avoiding Trailing Commas**: xsl:if test="position() != last()" was used inside the loop to add commas **only** between JSON objects, not after the last one. This ensures that the JSON output is syntactically correct.
4. **Combining Values**: First name and last name needed to be combined into a single "name" field in the output, requiring multiple <xsl:value-of> with embedded <xsl:text>.
5. **Nested Structures**: The address field had to be rendered as a **nested object** inside each user JSON object, mirroring the nested <address> XML structure.

Advantages of This Solution

1. **Interoperability**: JSON output can be easily consumed by JavaScript, React, Node.js, and REST APIs.
2. **Extensibility**: If additional user attributes are introduced in the future, the XSLT can be easily extended.
3. **Validation**: The JSON output structure can be strictly validated against the generated **JSON Schema**.
4. **Robustness**: The solution gracefully handles missing or optional fields without breaking the JSON format.

This scenario demonstrates how XSLT can be effectively used even beyond its traditional XML-to-XML transformations to generate modern data formats like JSON.

Careful handling of conditions, string formatting, and structure generation was crucial to building a valid, complex JSON output from a deeply nested XML source.

AI Assistance in the Project

During the course of this project, we utilized AI assistance tools to enhance our efficiency and explore alternative approaches. Specifically, we engaged with ChatGPT, GitHub Copilot and others at various stages.

Example : Improving XSLT Readability

- **Prompt:** "Can you suggest ways to make this XSLT code more readable? "
- **What we got:** The AI provided suggestions such as adding more descriptive comments, using more meaningful variable names, and restructuring the code for better flow.
- **What we corrected:** We reviewed the suggestions and selectively implemented those that improved clarity without altering the functionality of the stylesheet. For instance, we added comments to explain the purpose of specific <xsl:template> and <xsl:value-of> elements.

Conclusion

In conclusion, this group project successfully designed and implemented an XML database solution for a collaborative real estate platform. We effectively modeled various entities and their relationships using an XML Schema, populated it with a representative XML database, and demonstrated the versatility of this data through seven distinct scenarios. These scenarios showcased the ability to visualize data for different purposes using XSLT stylesheets and to transform the XML data into alternative formats like XML and JSON, catering to diverse application needs. The project highlighted the strengths of XML in structuring complex data and the power of XSLT in manipulating and transforming it. While acknowledging the potential complexity and verbosity of XML, our modeling choices, particularly the unified user representation with role-based differentiation, aimed for clarity and maintainability. The successful completion of all assigned tasks, including the optional Python implementation, underscores the team's collaborative effort and technical proficiency in XML technologies