Data Pipeline – Group Project

Stéphanie AGBODJOGBE - DE

Hortis DOUMAKPE - DE

Hamza MOSTEFAOUI - DA

Vincent PINNEAU - DA

PLAN

[I. Workload distribution 1](#_Toc1200829781)

[II. Working Environment and Tools 2](#_Toc2120365746)

[III. Modeling Principles and Choices 3](#_Toc867063159)

[IV. Scenarios and solutions 4](#_Toc910825335)

[V. Use of AI Assistance 6](#_Toc1122847914)

# Workload distribution

At the beginning, each of us thought of an example of a schema and a document, and we put everything together. And then we each share a part of the xml document to complete.

* Vincent PINNEAU :
* Task 1: XML Schema design
* Task 2: XML database :
  + Customers
  + A part of RECIPES
  + A part of REVIEWS
* Task 3: 2 visualizations scenarios (2 & 3)
* Hamza MOSTEFAOUI:
* Task 1: XML database
  + A part of ORDERS
  + A part of RECIPES
* Task 2: 1 visualization scenario (1) and scenario 7
* Task 3: Report writing
* Hortis DOUMAKPE
* Task 1: XML database
  + A part of ORDERS
  + A part of RECIPES
* Task 2: 1 visualization scenario (5)
* Task 3: A part of scenario 6
* Stéphanie AGBODJOGBE:
* Task 1: XML database
  + Delivery persons
  + A part of RECIPES
  + A part of REVIEWS
* Task 2: 1 visualization scenario (4) and a part of scenario 6
* Task 3: Report writing and XML document combination

# Working Environment and Tools

We worked in a collaborative environment using the following tools:

* Notepad++ for editing XML, XSLT, and other text files
* W3Schools.com for visualization of the scenarios
* Freeformater
* <https://xslttest.appspot.com/>
* Microsoft Word for report writing
* Stack Overflow
* Google
* [SVG Viewer - View, edit, and optimize SVGs](https://www.svgviewer.dev/)

# Modeling Principles and Choices

Each section of the database is organized hierarchically within the <RECIPE\_BOX\_DATABASE> root element, and individual records are represented as nested elements within their respective sections.

* CUSTOMERS:

Contains information about customers such as their full name, email, and phone number.

* RECIPES:

Contains details about recipes including recipe name, category, description, image URL, preparation time, total time, difficulty level, allergens, delivered ingredients, nutritional information, utensils required, cooking instructions, and unit price.

* REVIEWS:

Stores reviews given by customers for specific recipes, including star ratings and comments.

* DELIVERY EMPLOYEES:

Contains information about delivery employees such as their full name, email, and phone number.

* ORDERS:

Stores details about orders placed by customers, including the content of the order (selected recipes), serving size, delivery address, billing address, date of order, date of receipt, order status, and price.

**Advantages:**

* Rich Representation: The XML structure comprehensively captures various aspects related to recipes, customers, reviews, and orders, providing a detailed and structured representation of data.
* Easy Access: XML's hierarchical structure facilitates easy access to specific elements of interest, enabling efficient querying and extraction of relevant information.
* Self-Descriptive: The XML document is self-descriptive, with element names providing clear indications of their respective data, enhancing readability and understanding.

**Disadvantages:**

* Verbose Structure: The XML document exhibits a verbose structure, with extensive nesting and repetition of elements, potentially leading to larger file sizes and increased parsing overhead.
* Schema Complexity: Managing the XML schema for such a detailed representation can be complex, requiring careful consideration of the data model and potential changes over time.

One modeling problem we addressed was representing the relationships between entities. We chose to use “ref” attributes to establish references between them, as it provided a clear and efficient way to link related data without redundancy. For instance, between orders and delivery people we use “iddref” to reference the delivery person.

# Scenarios and solutions

We implemented solutions for various scenarios:

* **Scenario 1:** Display the recipes that are easy to make.
* **Scenario 2:** Create a synthetic and concise visualization to help a client to make precise and informed decision about a recipe from the italian recipes category. A client would like to choose an italian recipe but does not know which one.
* **Scenario 3:** The marketing team would like to know if some clients have given 5 stars to recipes, to be able to put these recipes forward, the next time they do a promotion campaign. Also, link the recipes to the client that gave the 5 stars so that the marketing team can tailor the promotion campaign to each client.
* **Scenario 4:** Create a visualization displaying the number of deliveries in which each driver is involved, so as to know who has never been called upon and who is too busy to help the delivery department head to balance the load.
* **Scenario 5:** Display the top 5 most ordered recipes for the company so they can suggest them as far recipes.
* **Scenario 6:** Write a svg document that displays the delivery addresses of each order in one block.
* **Scenario 7:** Calculate and display the average cost of meals by category.

The complex scenario we are going to explain is the scenario 7 about the JSON format.

We start by creating a JSON object called "average\_prices" to hold average price information for each recipe category.

For each unique recipe category, it calculates the average price of recipes within that category and includes the currency information. The output JSON structure contains category names as keys, with corresponding average price and currency information.

Finally, it ignores individual recipe elements, focusing solely on aggregating data at the category level.

**Few explanations:**

* Template Matching for RECIPE\_BOX\_DATABASE:

This template (<xsl:template match="RECIPE\_BOX\_DATABASE">) matches the root element RECIPE\_BOX\_DATABASE in the input XML. It signifies the beginning of the transformation and the start of the JSON object.

* "average\_prices" Object:

The line **"average\_prices”: {** is hard-coded into the output. It indicates the beginning of the object that will contain the average prices of recipes by category.

* Applying Templates to Each RECIPE:

<xsl:apply-templates select="RECIPES/RECIPE"/> applies templates to each RECIPE element under RECIPES.

* Template Matching for Each RECIPE:

The template <xsl:template match="RECIPE[not(CATEGORY = preceding-sibling::RECIPE/CATEGORY)]"> matches each RECIPE element that doesn't have the same CATEGORY value as its preceding sibling RECIPE element. This ensures that each category is processed only once.

* JSON Object for Each Category:

"CATEGORY": {This line generates the JSON key using the value of the CATEGORY element.

* average\_price and currency Values:

These lines calculate the average price for recipes in the current category and output it with a specified format. It also extracts the currency information from the first recipe in the current category.

* Template Matching for RECIPE:

<xsl:template match="RECIPE"/> This template matches all RECIPE elements and does nothing. It's effectively a template that suppresses output for RECIPE elements.

# Use of AI Assistance

During the project, we occasionally used ChatGPT for various concerns. The list of prompts used is as follows:

* Can you make up a comment about a recipe called X, which received a rating of X out of 5 stars?
* Considering the following xslt document can you comment the code to explain each line.
* What can you say about the advantages/disadvantages of my xml document?
* Translate all my comments in English.
* What is the error in my code?
* Give me a detailed explanation of the following xslt
* Can you correct my xslt (ça concerne mon code de transformation JsonAveragePrice)
* Suggest comments and ratings from 1 to 5 for recipes