PROJECT REPORT

WORKLOAD DISTRIBUTION

Name	Contribution Percentage	Tasks
Mathilde Roman	33.33%	- XML Database - XML Schema - XSLT (Scenario 1, 6, 7) - Project Report
Amiirah Codabaccus	33.33%	- XML Database - XML Schema - XSLT (Scenario 4, 5) - Project Report
Alexandra Curran	33.33%	- XML Database - XML Schema - XSLT (Scenario 2, 3) - Project Report

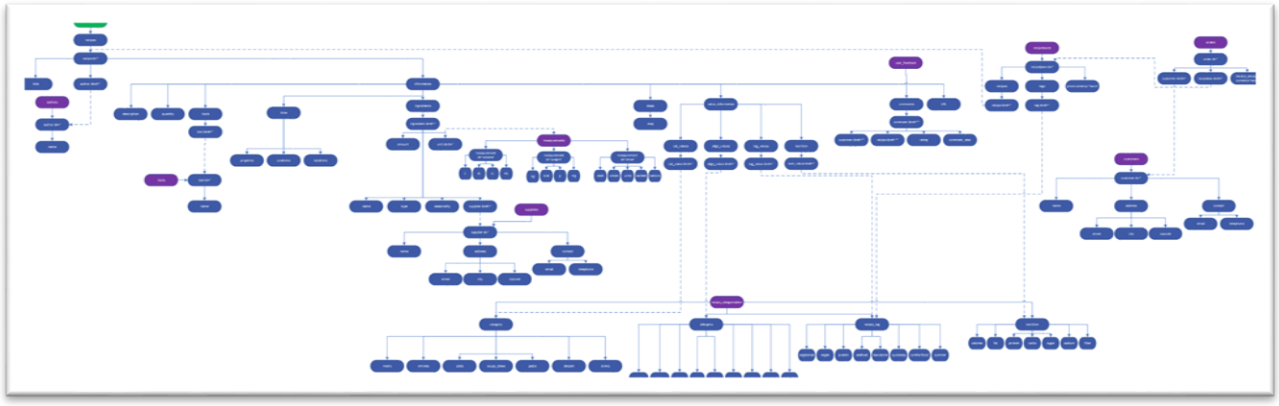
A group project is the sum of its parts. Each member contributed to the best of their abilities, contributed their own personal insight, ensuring a real collaborative effort on this project.

WORKING ENVIRONMENT

Tool	Usage
VSCode	- For writing code - Extensions “XML” (Red Hat) and “XML Tools” (Josh Johnson) allowed us to validate our XML and XSD files - Extension “Live Share” (Microsoft) allowed us to live code together
GitHub Desktop/GitHub	- For collaboration (pushing and pulling edits)
Codebeautify.com	- For validating our scenarios
xsltransform.net/	- For visualizing the HTML output of our scenarios

MODEL OVERVIEW

The flowchart documenting the Parent-Child relationships in our model can be found below. For better readability, a PDF version is included in the project zip file.



The **core principles** of our model are as follows:

- **Flexibility**: our model allows the easy addition of data via id referrals to different sections e.g. ingredients, tools, etc. It allows some wiggle room for absent data e.g. no comments, no ratings, etc.
- **Maintainability**: we attempted to envision our model in the long term. Maintenance is key, and our model should be clear enough for any new incomer.
- **Real-world Scenarios:** our love for cooking pushed us to open our model to as many scenarios as we could, even further than the ones we did e.g. recipes by author, ingredients by seasonality, etc.
- **Granularity:** we wanted our model to have clearly defined sections for any element that could be duplicated in the other sections e.g. tools, measurements, etc.

The **pros** and **cons** of our model are as follows:

Pros	Cons
- **Referential Integrity:** the inclusion of references e.g. id/idref ensures consistency of the relationships throughout - **Clear Structure:** having separated sections makes it easier to organize and visualize the relationships between them - **Modularity:** separate sections allow for updates to specific sections without affecting the rest of the model. - **Detailed Data Points:** data such as seasonality, type, tools, etc. provides additional context	- **Query Complexity**: due to the granularity of the sections, the queries may become more complex the more sections are added. - **Scalability:** if the database were to grow, the model may need to be revisited and separated into distinct files per section for easier parsing

One **modelling problem** we addressed was the representation of ingredients and their associated metadata in a recipe.

Solution:

We decided on using separate elements for both ingredients, amounts and units.

- Ingredients: in a recipe, the ingredients are referenced via an ID. Ingredients were defined with their own unique ID, as well as name, type, seasonality and supplier.
- Amount: in a recipe, the amount is specified
- Unit: in a recipe, the unit is referenced via an ID

This allowed us to maintain a separate database for ingredients without needing to alter recipe entries. This increased the filtering possibilities. The inclusion of a unit section would allow in the future measurement conversion.

SCENARIOS OVERVIEW

Scenario	Description
1. Low Glycemic Index Recipes	Retrieve recipes that have a low sugar content
2. Quick Recipes	Retrieve recipes that are easy to make by retrieving those with a specific “quickeasy” tag
3. Summer Seasonal Recipes	Retrieve recipes that are tagged as “summer”
4. Comfort Food Recipes	Retrieve recipes that are tagged as “comfort food”
5. Recipes Ranking	Retrieve recipes and rank them by average rating
6. XML: Recipe box and ingredient list	List the recipe boxes, their recipes, and the ingredient list per recipe
7. JSON: Recipe box and ingredient list	List the recipe boxes, their recipes, and the ingredient list per recipe

XSLT TRANSFORMATION USAGE

With our XSLT transformation, we can implement specific filtering, be it by recipe tag, dietary needs, ingredient seasonality, etc. The interoperability between the clearly defined sections offers the possibility of an app implementation for ordering and using recipe boxes e.g. buy a recipe box, see the complete ingredient list for a specific box, etc.

COMPLEX SCENARIO

One of the scenarios that required a more complex solution than anticipated was scenario 5: addressing a ranking of the recipes. The solution required defining a key, variables, and implementing them in aggregations:

- We first search for all recipes that have at least one comment
- We then sort them based on the average rating calculated from their comments
- We define variables to store the recipe ID, comments, total ratings, and average rating
- These variables are then used to calculate outputs in the table for recipe rank, recipe name, and its average rating.

AI QUERIES

OpenAI (ChatGPT) was used for troubleshooting our code when Stackoverflow provided no viable answer.

Query #1

Input	*[Scenario 5.xsl inserted]* For some reason, the table shows 3 extra lines with nothing
Output	If you're seeing extra lines with no content in the output, it's likely due to the presence of empty <recipe> elements in your XML data. To prevent these empty lines from appearing in the output table, you can add a condition to filter out such elements. *[Scenario 5.xsl modified line]* `<xsl:for-each select="//recipes/recipe[.//comment]">` The expression [.//comment] checks if there are any <comment> elements within the current <recipe> element.
Action	We added the expression to our code and it resolved our issue. We mistakenly thought .// and // were the same.

Query #2

Input	*[Scenario 6.xsl inserted]* Could the error be because in recipes and recipeboxes there are tags called recipe?
Output	Yes, you're correct. The issue might arise because both the <recipes> and <recipeboxes> elements contain child elements named <recipe>, which can lead to ambiguity in XPath expressions. To resolve this, we can use relative XPath expressions to ensure that we're selecting the correct <recipe> elements within their respective contexts. *[Scenario 6.xsl modified line]*
Action	The expression //recipes/recipe was changed to //recipe in the output. It did not solve our issue at all and was disregarded. We started from the ground up and found the issue came from how our template match was organized.