**Scenario favorite recipe dvp:**

**Challenges and Solutions**:

**Challenge 1**: Filtering the recipe with a rating greater than 3

**Solution**: in the XSLT we used the xsl:if tags allowing us to select nodes in function of a specific condition.

**Challenge 2**: Do the sum for each recipe of the total time in the kitchen, meaning the addition of the time for cooking and the time for preparing the recipe.

**Solution**: in the XSLT we create an element total\_time which contains variable cook\_time and prep\_time corresponding respectively to the value of the cooking\_time and preparing\_time element in the database. In creating those variables we were able to reuse them by putting a dollar sign before them and to do the sum by transforming the value of each into a number to do the sum (lines 35-46 of the file).

**Difficulty**: here we had to find where to declare the variable for getting them different for each loop. In declaring them outside the for each loop the variable gets all the value of the element and creates a list with them and does the sum for all what we don’t want. It’s for that we declare the variable inside the for-each loop in that way at each iteration the XSLT parser gets the value of the elements in the loop iteration and does the sum to then change the variable value in the next loop iteration.

Scenario JSON dvp:

**Challenges and Solutions**:

**Challenge 1**: Make a scenario to transform the customer part of the recipe box database in a JSON file format.

**Solution**: in the XSLT we structure the code to include the JSON style such as the { } to declare the JSON object, the [ ] to declare the array which contains the customer informations, the “ “ around the key, the : to declare the value attached at the key, and a comma after each element of the object to separate them. We used a xsl:if test for putting at the end of each object a comma except for the last customer where the comma isn’t requested.

**Challenge 2**: guide the users to complete the JSON file by creating a JSON schema.

**Solution**: The file Customer\_schema.json allows us to guide the users of this file to complete it. We detail in it the different types of JSON elements as well as if they are requested or not in the “required” element at the end of the properties declaration. We declare a pattern for the phone number to keep the same restriction as it was declared for the xml database schema with a regex expression. At the end we make a dependency relation with the “dependencies” element between the Customer\_PhoneNumber and Customer\_email. The 2 elements aren’t required but the element Customer\_email can’t exist if the customer\_PhoneNumber doesn’t exist.

Scenario menus\_book dvp:

**Challenges and Solutions**:

**Challenge 1**: extract the information from different parts of the database.

**Solution**: First we get the information from the menu part to describe it (number of recipes, price, and rating). Then we group the information from the recipe and nutrition part of the database corresponding to the recipe listed for this menu in the database. For that, we defined the XPATH with the position of the node we targeted in the first part of the XSLT for example the recipe “gaspacho” is at the 4th position in the recipe part of the database and its nutrition facts is at the 2nd position in the nutrition part of the database.

**Challenge 2**: Organize the HTML output to display the different elements.

**Solution:**  To better distinct the different parts of the HTML file we put text and use the blockquote tags to create a block of elements in the file. The titles are centered and we play with the element size with the h tags (h1 to h4) where h1 display the text element bigger than h2 and so on. This allows us to differentiate the document part. At least, for the ingredients and nutrition facts of each recipe we create tables.