WEB ARCHITECTURES
Assignment 3

Riccardo Gennaro

November 6, 2022

1 Introduction

1.1 Problem statement

This project aims at developing a single-page web application that shows to the users a simple spredsheet.

Regarding the application itself, it has to satisfy the following, shown synthetically, requirements:

- **Spreadsheet**. The application must show to the user a matrix of 16 cells. These cells can be read/written by anyone who access the web application. The spreadsheet implements the basic four arithmetical operators.
- **Asynchronous communication**. The client-server communication must be carried out in an asynchronous way. For this project, the communication was implemented through AJAX requests using JSON files.
- **Polling**. Once a user cause the value of cell to change, this operation must be propagated to all other users. To do so, a client-to-server polling was implemented.

2 Proposed solution

Figure 1 represents the server-side workflow depending on the Javascript function called client-side. Following, a description of the workflow of the application.

2.1 Connection to the web application

When a user connects to the web application, it's request is redirected to *SheetServlet*, as specified in the web.xml. The servlet send a response containing *spreadSheet.html*.

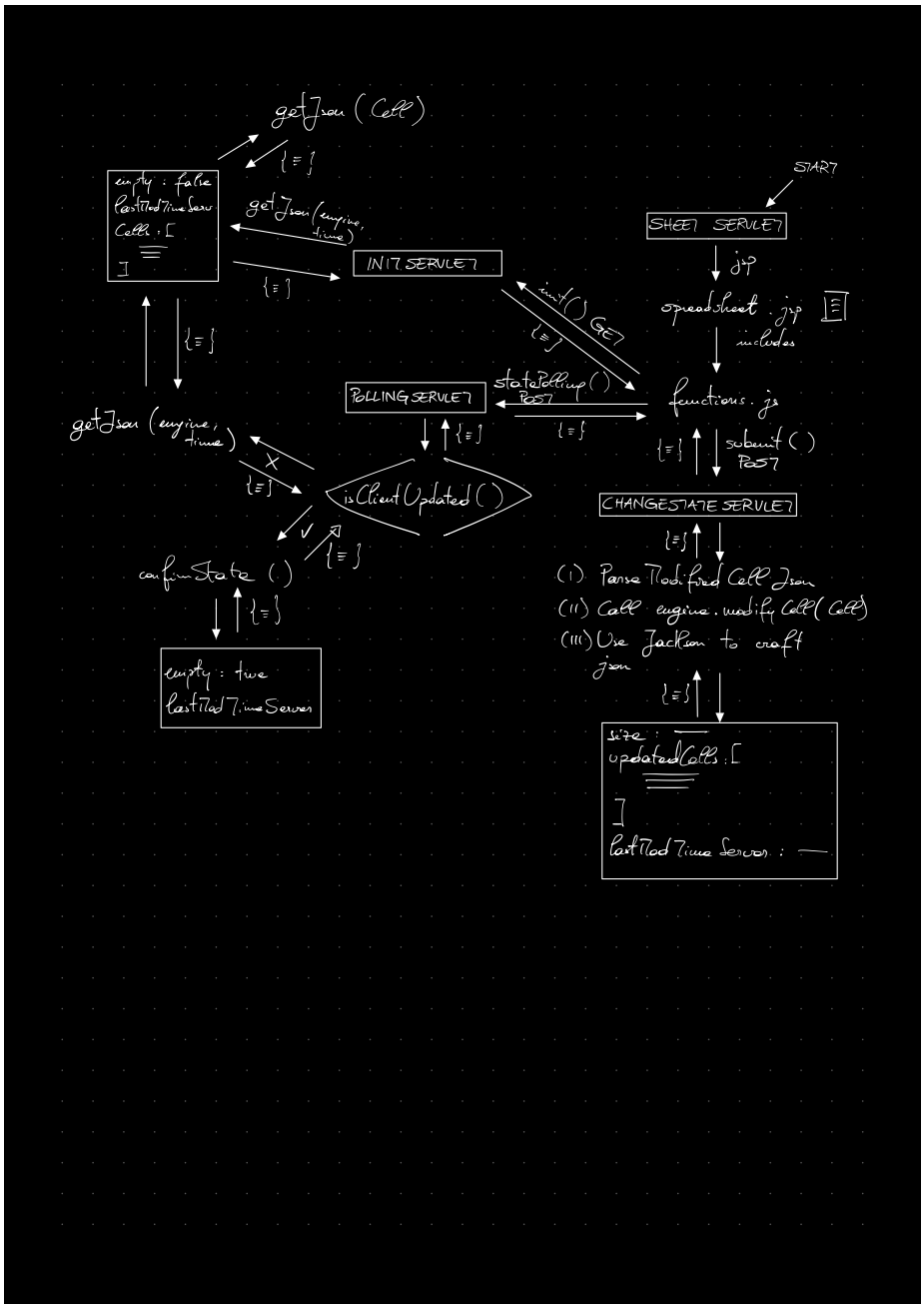

Figure 1: Server side workflow.

2.2 Spreadsheet

At this point, the .html page is rendered client-side. File spreadSheet.html includes file *js/functions.js*, containing the javascript code used to build the page and communicate with the server.

2.3 Client-side functions

File *js/functions.js* contains the below functions.

- `init()`. Called on page load. Sends a GET request to *InitServlet*. When the servlet answer with a JSON containing the cells, `init()` parse the JSON, initialize an array of cells, and render the html table containing the cells. This function also defines the event listeners related to input text area used to edit a cell.
- `submit(formula)`. This function is called via a wrapper, `submit_wrapper(formula)`, that checks if the formula to submit has been changed or if it is empty. Once `submit(formula)` is called, the function send a POST request to *ChangeStateServlet* with a JSON containing the modified cell's data. Server answer with a JSON containing the list of modified cells and the timestamp of the last modification. Changes are applied client-side.
- `statePolling()`. This function is called every five seconds. It sends a POST request to *PollingServlet* with a JSON containing the above mentioned timestamp of the last modification. If the last server-state modification occurred after the sent timestamp, server sends the updated state of the spreadsheet through a JSON file; otherwise, it sends a confirmation that the state has not been modified through a JSON containing an `empty` flag. Once client receives the JSON, it checks the `empty` flag: if it's true, the state is not modified; if it's false, the cells values and formulae are updated. This function is needed in order to keep track of other users' changes.

File *js/functions.js* also contains other auxiliary functions used to manage the page CSS and other events.

2.4 Server-side servlets

2.4.1 SheetServlet

As previously specified, this servlet is called at the connection to the web application. It sends *spreadSheet.html*.

2.4.2 InitServlet

As previously specified, this servlet is called through a GET request at the loading of *spreadSheet.html*. It initialize the provided application `engine`, build a

JSON containing the cells' values and formulae through `Json.getJson(SSengine, long)`, and sends it to client.

2.4.3 ChangeStateServlet

This servlet answer the POST request sent by client through `submit(request)`. It parses the JSON using the *Jackson* library, and uses `engine.modifyCell(String, String)` to update the server state. If state is modified, a JSON containing the modified cells and the timestamp of current modification is crafted and sent to client; otherwise a JSON containing attribute size set to zero is crafted and sent to client.

2.4.4 PollingServlet

This servlet answer the POST request sent by client through `statePolling(request)`. It checks if the client-side timestamp of the last modification is *before* the server-side one. If so, it crafts JSON containing the updated engine state and the last modification timestamp through `Json.getJson(SSengine, long)` and sends it to client. If last modification is *after* the server-side one, a confirmation that the state is up to date is sent.

3 Running application

3.1 Start

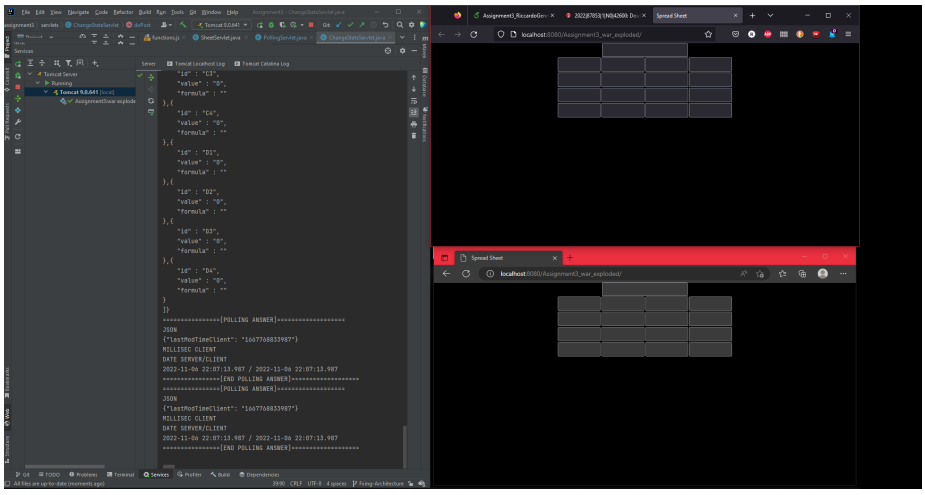

Figure 2: Initial page. No modifications

3.2 Successful submits

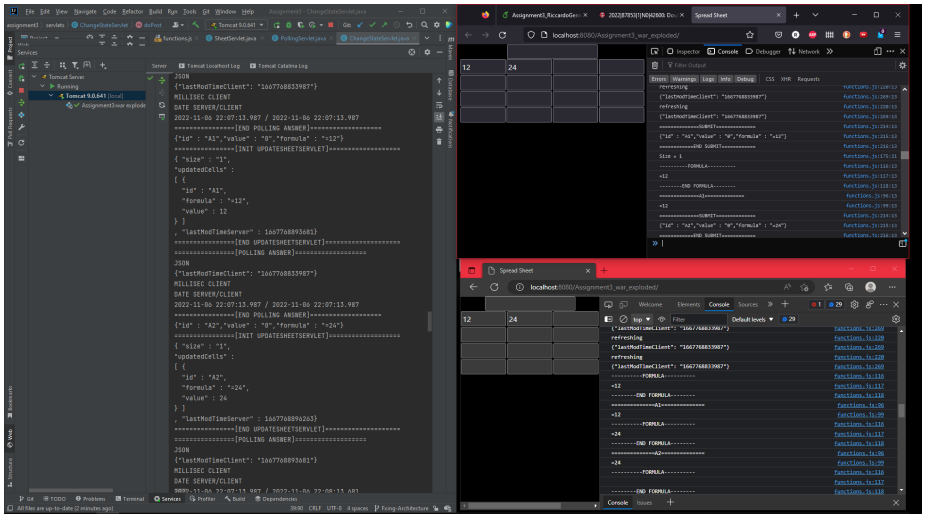

Figure 3: Successful submit of values 12 and 24 in cells A1 and A2

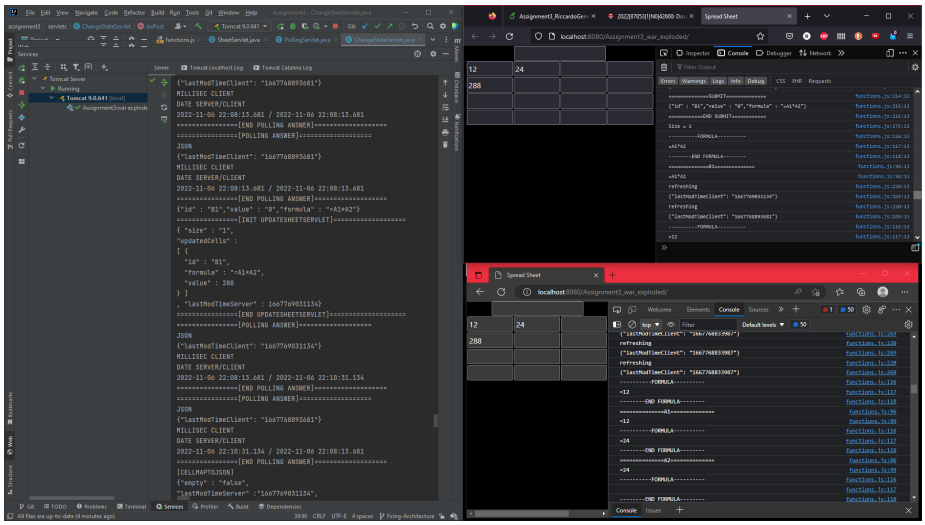

Figure 4: Successful submit of operation =A1*A2 in cell B1

3.3 Unsuccessful commit

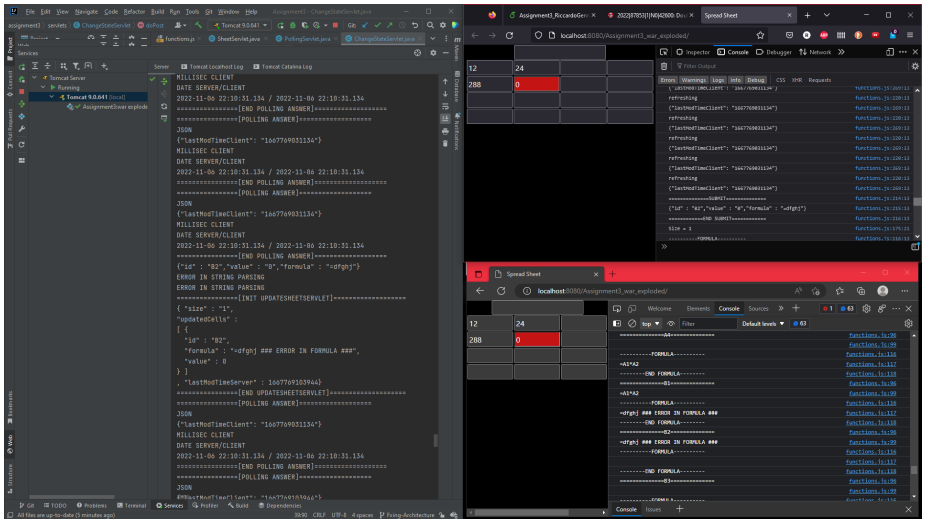

Figure 5: Error while trying to submit not valid formula

3.4 Comments and notes

Note that submitting a formula with circular dependencies will not change the server state. I tried to change some code in SSEngine::modifyCell(String, String) in order to obtain an error message on client-side, but doing so caused the Server to throw a StackOverFlow exception whenever a formula with nested circular dependency were evaluated. The changed code (now commented) can be found in SSEngine::modifyCell(String, String). The StackOverFlow exception was thrown by Cell::recursiveEvaluateCell(), since storing the value of the cell with circular dependency caused the recursive evaluation of the above mentioned cell.