Assignment 3

Part 1:
Build a web application that implements a simple spreadsheet .
The client part is a Single Page Application: after loading the initial page, every user action that modifies a cell triggers an HTTP request. The server responds computing the variations, and returns all the cells which have been modified (in JSON format). The browser then updates the cells in the view.

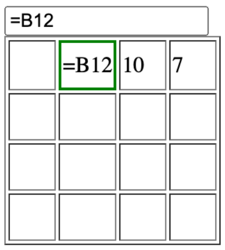
Part 2:
Other users may connect to the web site, obtaining a view of the spreadsheet created by the first user. Like the first user, they can act on the spreadsheet, modifying cells. Cell modification made by any user are propagated to all connected users.

Write a report with the usual structure and deliver report in pdf and code as zipped IntelliJ project.

Further specifications and suggestions

- On the Browser, a spreadsheet cell has the following properties:
    - An identifier, which also specifies the position in the matrix (e.g. B3)
    - A formula (see later)
    - A value
- The identifier is composed by a first, alphabetic char which identifies a column, while the rest of the token is numeric and identifies a row.

- The core of the client view presents an editable field and a matrix of non editable cells, which show the value of the cell.
- When the user clicks on a cell, its border is highlighted, and the field above the matrix shows the formula of the cell. The user can edit this field: every key typed is also reflected in the cell (which during editing shows the formula instead of the value).



During editing, when the user types the Return key or clicks elsewhere, the formula is submitted to the server. The server evaluates the formula and returns a JSON structure containing all the cells which were modified by the application of the formula, cells with their new values (e.g., if we edit cell B2, and cell C3's formula is =B2+6, both B2 and C3 are modified). The client updates all those cells, showing their values in the matrix (without reloading the page).

- The code for an engine usable on server side that implements the business logic is provided: the source code is downloadable from the course web site. It is in a package containing two java classes: SSEngine and Cell (see later).

- A formula can be:
    - A string that is parseable as a number
    - A string that starts with = and contains a reference
    - A string t that starts with = and contains numbers and or references separated by operators (+-*/)
- Valid formula examples: 5, =6, =A2, =A2+3, =A4/A2-B5+3
- You do not need to validate or evaluate formulas client side, since you can use the provided engine to do that. Formulas with syntax error or circular dependencies evaluate with 0.

- There is only one spreadsheet managed by the app: if another user connects to the webapp, s/he sees the spreadsheet in the present state. No authentication mechanisms are required.

  Second part:

- When a user modifies a cell, all connected users see the result. To achieve this, use the JS setInterval function (see https://www.w3schools.com/jsref/met_win_setinterval.asp). The browser should send to the server a request indicating the timestamp of the state of the spreadsheet that it has. If the current state on the server side has the same timestamp, the response notifies that nothing has changed, else the new timestamp is sent, together with the new state.

The source code of the business logic component SSEngine is provided. It lives server side, and it is instantiated with the following command:

SSEngine engine=SSEngine.getSSEngine();

This call creates (server side) a matrix of Cell, composed by 4 columns (A, B , C and D), and 4 rows, customizable by changing in its code the lines

```
private final static String columns[]={"A","B","C","D"};
private final static int NROWS=4
```

When the SSEngine is created, all cells are initialized as empty. Subsequent calls of getSSEngine do not change the state of the engine, nor create new instances.

A Cell is an object having:

- an id (Composed by row and column, such as A3 or B1)
- a formula, of type:
    - numeric value, such as 11
    - numeric value with equal, as in =5 (equivalent to the previous form)
    - a reference, as in =B1
    - a composition of numeric values, references and operators, such as =B1+3*C2
- a value (calculated by the engine on the basis of the formulas).

Numeric values are restricted to be integers.
Operators are +,-,*,/. They are evaluated left to right (there are no precedence rules)
A formula cannot be circular (even over multiple cells). Attempts to use formulas which create circular dependencies are not allowed, and have no effects (cells with illegal formulas or with circular dependencies evaluate with 0).

A Cell can be modified by calling

```
modifyCell(String id, String formula)
```

which returns the collection (`Set<Cell>`) of modified cells which were modified either directly or in the recursive cascade (after a cell is modified, all other cells which depend on it get modified).

The value of a Cell can be read by calling `getValue()` on it.

The formula of a Cell can be read by calling `getFormula()` on it.

A Cell's state can be printed by calling `toString()` on it.

All other methods in Cell or SSEngine should not be used.

Examples are present in the main program of SSEngine, which is runnable for a test/demo.

An integral part of the specification is the discussion hold in class during the Friday 28 lecture.