ECM3408 Enterprise Computing
Continuous Assessment 2023-24

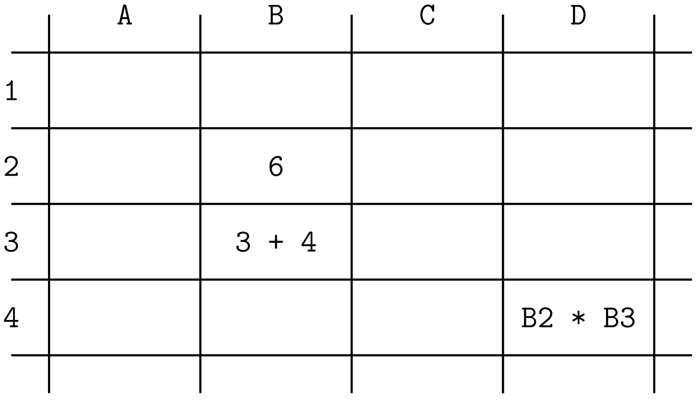
D. Wakeling

Handed Out	Handed In
Thursday 8th February 2024 (T2:04)	Wednesday 6th March 2024 (T2:08)

This Continuous Assessment is worth 40% of the module mark. Your attention is drawn to Faculty and University guidelines on collaboration and plagiarism.

1 Background

More than forty years ago, Dan Bricklin, a Harvard University MBA student, was pondering how he might use a computer to help him to make some complicated financial projections that he needed for one of his assignments. Bricklin was supposed to do his calculations by hand using ledger sheets, also called *spreadsheets*. A spreadsheet consists of a number of *cells* identified by a column letter and row number that may contain *formulas* to calculate their values. See Figure 1.



	A	B	C	D
1				
2		6		
3		3 + 4		
4				B2 * B3

Figure 1: An example spreadsheet.

Although Bricklin knew that spreadsheets were essential for doing his calculations, he also knew that the computers could do the work far more quickly and accurately than humans could. Calculating the value in one spreadsheet cell might involve calculating the values in some other cells, which might in turn involve calculating the values in some others, and so on. In large spreadsheets, many such calculations would have to be done, and probably later redone.

Despite the scepticism of his professors, Bricklin and a computer programmer friend, Bob Frankston set about creating the first spreadsheet program for the then-brand-new Apple II. On release, their VisiCalc (for "visible calculator") program immediately became the "killer app" for that machine. Many people went to computer shops to buy VisiCalc for $100 and an Apple II for $2,000 as "something to run it on."

Specification

A spreadsheet MVP may be implemented as a SC microservice with a RESTful interface and the following behaviour.

Creating Cells

The SC microservice allows cells to be created. See Figure 2. Here, the `PUT` method

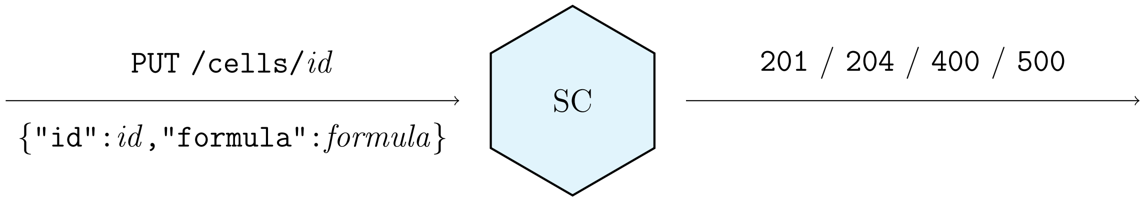


Figure 2: Creating cells.

takes a JSON object that has an `id` property whose value is a string representing the cell identifier, and a `formula` property whose value is a string representing its formula. Successful results are `201 Created` and `204 No Content`. Unsuccessful results are `400 Bad Request` and `500 Internal Server Error`.

Reading Cells

The SC microservice allows cells to be read, and their values to be calculated. See Figure 3. A successful result is `200 OK` accompanied by a JSON object that has an `id` property whose value is a string representing a cell identifier, and a `formula` property whose value is a string representing the result of a formula calculation. To keep things simple, we shall assume that formulas are made up of only integer or floating-point numbers, the

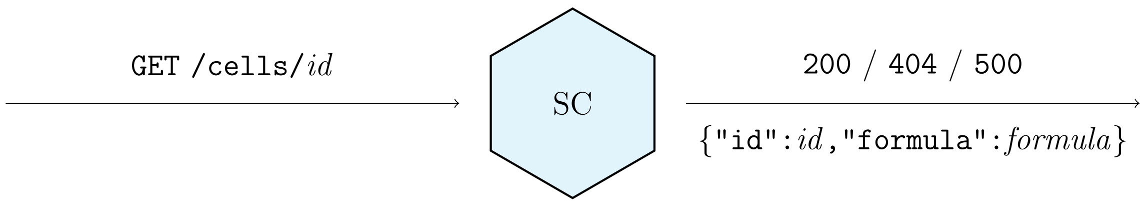


Figure 3: Reading cells.

operators '+', '−', '×', '/' and unary '−', with parentheses for grouping. The values of formulas will be calculated as in Python. There is no to check for cycles between cells or division by zero. Unsuccessful results are `404 Not Found` and `500 Internal Server Error`.

Deleting Cells

The SC microservice allows cells to be deleted. See Figure 4. A successful result is `204 No Content`. Unsuccessful results are `404 Not Found` and `500 Internal Server Error`.

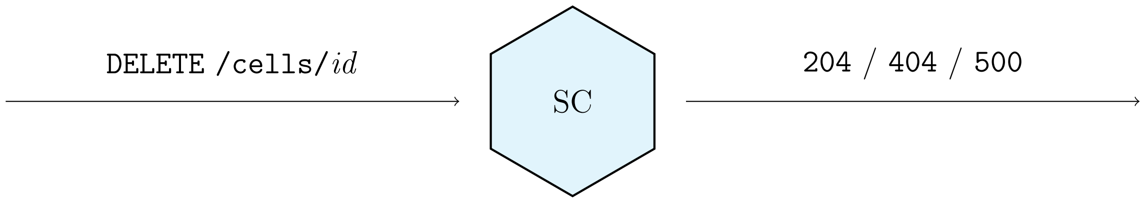


Figure 4: Deleting cells.

Listing Cells

The SC microservice allows all cells to be listed. See Figure 5. A successful result is `200 OK` accompanied by a list of strings. An unsuccessful result is `500 Internal Server Error`.

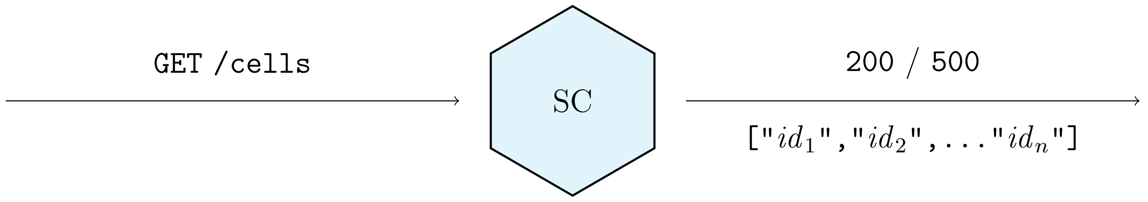


Figure 5: Listing cells.

An Example

The following sequence of commands creates the spreadsheet of Figure 1.

```
curl -X PUT -H "Content-Type: application/json" \
     -d "{\"id\":\"B2\",\"formula\":\"6\"}" localhost:3000/cells/B2
curl -X PUT -H "Content-Type: application/json" \
     -d "{\"id\":\"B3\",\"formula\":\"7\"}" localhost:3000/cells/B3
curl -X PUT -H "Content-Type: application/json" \
     -d "{\"id\":\"D4\",\"formula\":\"B4 * B3\"}" localhost:3000/cells/D6
```

The following command reads a cell and calculates its value

```
curl -X GET localhost:3000/cells/D4
{"formula":"42","id":"D4"}
```

and the following command lists all cells in the spreadsheet

```
curl -X GET localhost:3000/cells
["B2","B4","D4"]
```

Assessment

This assessment involves implementing the SC microservice.

Part 1: On-Premises Storage (50%)

The SC microservice can store cells on-premises in a SQL database. The microservice must listen on port 3000, and in this case must use a Sqlite database. A flag indicates that a Sqlite database is to be used when the microservice is started

```
    python3 sc.py -r sqlite
```

This part will be assessed by a script "`test50.sh`" that runs 50 tests, and prints a score reflecting how many of those tests were passed. A script "`test10.sh`" that runs the first 10 tests of these tests will be made available for training.

Part 2: Cloud Storage (50%)

The SC microservice can also store cells in the cloud in a No-SQL database. The microservice must still listen on port 3000, but in this case must use a Firebase Realtime Database. A flag indicates that a Firebase Realtime Database is to be used when the microservice is started

```
python3 sc.py -r firebase
```

There are three possible locations for a Firebase Realtime Database: `United States (us-central1)`, `Belgium (europe-west1)` and `Singapore (asia-southeast1)`. Choose `Belgium (europe-west1)`. The database can run in *locked mode* or *test mode*. Choose *test mode*. Your Firebase Realtime Database name will look like `toybase-3d1c1`, which gets incorporated into a URL like

```
https://toybase-3d1c1-default-rtdb.europe-west1.firebasedatabase.app
```

Please ensure that your database name is read from an environment variable, set up with

```
export FBASE=toybase-3d1c1
```

This part will be assessed by the same script "`test50.sh`" that runs 50 tests, and prints a score reflecting how many of those tests were passed. The same script "`test10.sh`" that runs the first 10 tests of these tests will be available for training.

Test Environment

The test environment for all code will be the machine "`blue18.ex.ac.uk`", where the version of python3 to be used is `Python 3.6.8`. The standard libraries will be extended with a virtual environment containing only the `flask` and `requests` libraries, as follows

```
rm -rf venv
python3 -m venv venv
source venv/bin/activate
(venv) python3 -m pip install flask
(venv) python3 -m pip install requests
```

Any code that does not run in this environment will be considered not to run at all.

Submission

You should submit a single ".zip" file (note, *not* 7zip or WinRAR) containing the Python source code of your microservice in a directory "`sc`".