Databases coursework 3: The IoT Fridge

Steve Kerrison

26th March – 1st May 2015

The Internet of Things (IoT) is a term given to the emergence of embedded electronic devices that provide useful services to us, with the help of an internet connection. All kinds of things, from your car, the phone in your pocket, down to light bulbs[1], for the IoT. It is envisaged that there will be many *billions* of internet connected devices in the future, significantly outnumbering humans on the planet.

Lots of these devices will need a local data store to perform their tasks effectively. This coursework supposes that it is your job to implement a database for an *intelligent, internet enabled fridge*. You will design, implement, and construct a demo of this database and a small amount of software around it. You will not have to make a real fridge, however.

1 Goals of this coursework

- Familiarise yourself with SQLite, an embedded database engine.
- Identify the data storage needs that might be required for an internet fridge.
- Propose a schema for the database.
- Implement the database in SQLite.
- Write a program that can act as a simple API between the fridge user-interface and the database.
- Provide sample data and API calls that demonstrate the features of your internet fridge and its embedded database.
- Evaluate the longevity of the design with respect to performance over time and future integration of new features.

2 Format and marking scheme

This coursework has three main parts: the database schema design and implementation, the API implementation and demonstration and a short report giving you an opportunity to justify the design and implementation decisions you have made, as well as to discuss issues such as performance and longevity.

All three parts will be assessed to determine your final mark. The distribution of marks will be as follows:

- **35%** for the schema design and SQLite implementation.
- **35%** for the features provided by the database, demonstrable through your API implementation.
- **30%** for the report.

As an indication of marking brackets, please consider the following criteria, which will be used to guide the marking process:

- **40% or less** is indicative of minimal functionality in the API, a very simple database implementation and no attempt made at the report.
- **Up to 50%** indicates that you have applied some reasoning to the report, constructed some form of database following reasonable practices, but have not demonstrated any particularly sophisticated database or API features.
- **Up to 60%** shows that your database schema is sound and your API demonstration works without problems. A good attempt has been made to follow good database practices. The report provides insight into the decisions made during the design process.

[1] http://www.theregister.co.uk/2013/03/29/zigbee_ip/

- **Up to 70%** will be awarded to work that has either a sophisticated feature set or a very well designed database that ensures good data integrity as much as is feasible, with a detailed report to back the work up. There are some novel ideas in the work.
- **Over 70%** indicates excellence, combining both a well structured database and a widely innovative and well implemented feature set in the API. You have demonstrated an ability to visualise a broad set of ways in which the product can be used and catered to these possibilities. The report clearly communicates the design and implementation processor, also including further research or experimental results to form a well reasoned evaluation of your own work.

Some ideas that may help you to achieve higher marks are given in the full description of the coursework that follows.

3 Submission

You will need to submit the following:

- An SQLite compatible schema, in a file titled `fridge.sql`.
- Your API demo code, in whatever language(s) you have chosen to use.
- A README.txt file, providing instructions on how to compile your software and run a demo. It should only contain simple instructions, descriptions should be put into the report document instead.
- Any additional data files (sample data, for example) or scripts that are needed in order perform the demo.
- A report, of file name `report.pdf`, containing no more than *four A4 pages*.

Do not submit an SQLite database file. If you do it will be ignored. The database will be constructed and pre-populated from the SQL file you submit, then your API demo access/update it, etc.

Submit via SAFE all of the above by the deadline of **1st May, 23:59:59** to avoid a late penalty. The SAFE page for this unit is https://wwwa.fen.bris.ac.uk/COMS20700/

Remember, the submission **must be your own work**, and plagiarism checks will be performed against both your code and your report to ensure this. If you use any external data sources or software libraries, please state this in your report and include any appropriate copyright and licensing notices with your submission, as per the requirements of the license(s).

4 Instructions

These instructions give you a very simple framework to get you started. It is recommended you take these and build upon them. However, if you have an alternative approach that you wish to follow, you may do so.

4.1 Overview of interaction

You should view your *fridge* as a device with some kind of *barcode scanner* and a touchscreen interface with which the user can interact. For the purposes of this coursework, of course, your *imagine* this to be the case, and instead operate at one level below these devices, in an API.

Figure 1 shows roughly how the fridge would be constructed, if it really existed. You can make reasonable assumptions about what the fridge, display and scanner can do, but try to make sure your assumptions are practical in the real world (e.g. you can't just put a bag full of products in the fridge and expect the fridge to identify all the contents... at least not with current tagging technology). You are given almost complete freedom with what the API can do, but some basic examples will now be provided.

4.1.1 API request and response format

JSON (JavaScript Object Notation) is a commonly used format for flexible data encoding. It is also relatively easy to read as a human, and fairly easy to interpret in code, with lots of software library support available. JSON, therefore, is the recommended format for this coursework.

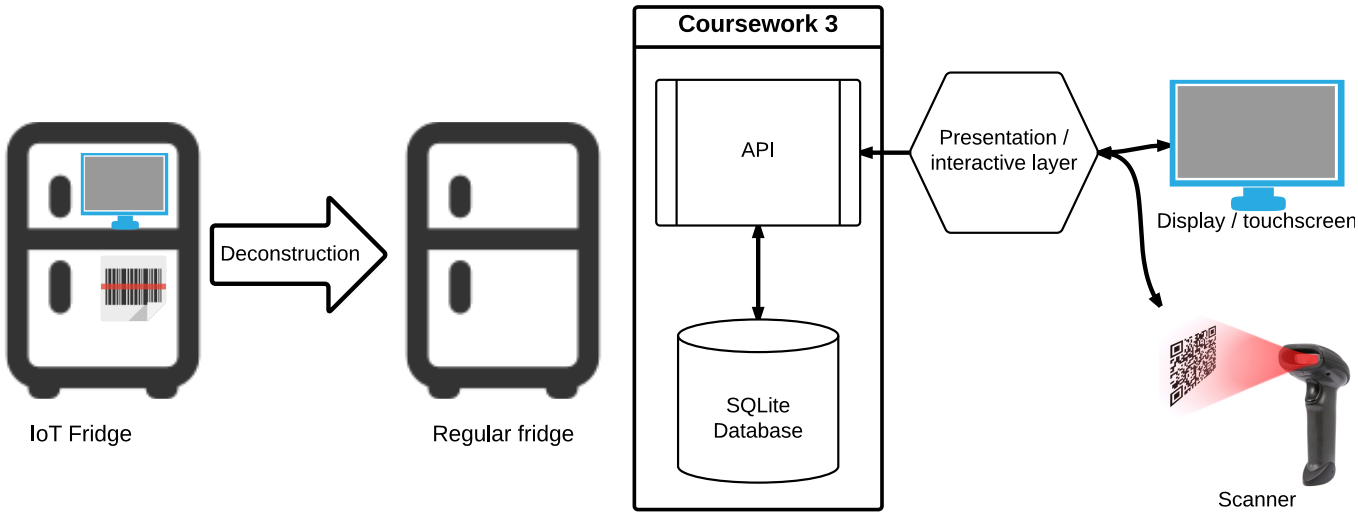

Figure 1: Your IoT fridge. You will work solely on the database and API, making reasonable assumptions about the capabilities of the rest of the system.

If an error occurs, ideally the program should respond with an error message but then accept new input, rather than crash. **Try to ensure this!**

Here are some very simple request/response examples that you can start to think about:

Adding an item to the fridge

Request:

```
{ 'request': 'insert', 'id': '12345',
    'data': { 'name': 'Tuna steak, two pack',
        'manufacturer': 'Fishy industries',
        'weight': 400,
        'expires': '2015-04-20' } }
```

Response:

```
{ 'response': 'OK', 'success': true }
```

Removing an item from the fridge

Request:

```
{ 'request': 'remove', 'id': '12345' }
```

Response:

```
{ 'response': 'OK', 'success': true }
```

Listing the items in the fridge

Request:

```
{ 'request': 'list' }
```

Response:

```
{ 'response': [ { 'id': '12345',
    'name': 'Tuna steak, 2 pack',
    ... } , { 'id': 'abcdef',
    'name': 'Unsalted butter',
    ...}, ...
    ],
    'success': true }
```

Exactly what data goes *into* and comes *out of* the requests is up to you. You are tasked with determining what data would be useful to this device, and when how it should then present the data.

For example, the list of items in the fridge could be taken and displayed on a screen. However, including every detail of a product in that list would be inappropriate. It may be sensible to add a new request for extended details on a particular product, thus creating an API request that can handle the equivalent of clicking "more details" on an item in the list.

4.2 Designing the database

With an idea of what activities you'd like your fridge to perform, think about what data needs to be stored within it and how it should be structured. As with previous courseworks, you will have to think about the following:

- Sensible data types
- Appropriate sizing
- Data integrity constraints
- Normalisation

You may wish to produce an E/R diagram of your database and include it in your report.

4.3 Implementing the database

You will need the SQLite client installed on the machine that you are using. Either get this from the SQLite website, or your Operating System's package manager, **or use the lab machines/snowy, which have the correct sotware installed**. Assume that the database creation process happens *outside* of your demo application — the application only manipulates data within the database, it doesn't drop or create tables.

As such, the process for *bootstrapping* the fridge database, would be along the lines of:

```
sqlite3 fridge.db < fridge.sql
```

This will create a file, `fridge.db` (if it doesn't exist already), and run the SQL script `fridge.sql`, which will contain the `CREATE TABLE` and other statements required to construct the database that you have designed.

If you want to pre-seed the database with some demonstration data, this can also be done in this file, or with a second sql script that is loaded afterwards.

4.4 Implementing the interface

If you use the suggested request/response format, then a suggested method for implementing the interface is as follows:

- Start your demo by running the compiled program or script, with the bootstrapped database as its only argument. For example:

  ```
  python iotfridge.py fridge.db
  ```

- Provide JSON requests via stdin and responses via stdout. It is recommended that the inputs come from a script.
- Put a blank line between requests so the API can parse individual requests easily and you can write requests over multiple lines if you want.

Internally, your application will use an SQLite library to construct queries and interact with the database. You may need to install this, for example SQLite for python.

A simple template is given to you at https://www.cs.bris.ac.uk/Teaching/Resources/COMS20700/cw/cw-03.zip. In it, you will find:

- `test.sql`, a test schema that you can run with sqlite3 to create a single, very simple table.
- `test.json`, a very simple set of commands to get you started.
- `iotfridge.py`, a skeleton for the API, that has very simple implementations of list and insert commands.
- `test.sh`, a script that will run test.sql against sqlite3 and test.json against iotfridge.py - if this cannot run, then your software or environment isn't set up correctly. Note: this script will create a database file called test.db if it doesn't exist already.

If you write your own implementation from scratch, you will have to use whatever library is available for the language you are using.

4.5 Writing the report

Your report should cover the *design*, *implementation* and an *evaluation* of your work, including possible future work and improvements. Figures may be useful, such as an E/R diagram. Keep screenshots to a minimum; behaviour should be observable via the demonstration run(s) instead.

Keep the report to **4 pages or less** and no less than **11pt** font (the size of this document's body text).

5 Ideas

This coursework is intentionally open-ended, giving you an opportunity to explore the database design, implementation and use process with more depth than previous coursework. It is your responsibility to set your own goals for what you want to get out of this coursework, and what features you want to demonstrate in your submission. A selection of ideas to consider are now given. Your own ideas and creativity are welcomed and encouraged!

You can, in theory, construct a good submission for this coursework without your API ever using the internet. However, in the spirit of the internet fridge, one should try to find a feature that benefits from an internet connection.

- Consider an IoT fridge/freezer; not just a fridge.
- Products typically have different cold storage requirements. Consider this, including aspects such as whether the product has been opened.
- What can I cook with what's in my fridge?
- Recommend that I find something to make with a set of items in my fridge that will soon expire.
- Imagine an internet fridge in a shared office space. Who does this milk belong to? Can I "borrow" some?
- I want to check what's in my fridge, but I'm not at home. Assuming the API is open to access over the internet, can I find out without letting everybody in the world have access to the list of my fridge's contents?
- What do I need to remember to buy today?
- Given the EAN13 number[2] of product (for example, from a barcode scanner), automatically look-up the product details from an online resource[3]
- What was in my fridge this time last year?
- Use an ORM to interact with the database, rather than SQLite directly.

The report is also relatively open, particularly in the evaluation section. However, some things you may want to think about:

- How long a history of items can the database store, without degrading performance?

[2] http://en.wikipedia.org/wiki/International_Article_Number_%28EAN%29
[3] http://www.outpan.com/

- How storage space efficient is the database?
- How might the database & software perform on a low-power embedded device, such as a small ARM or Arduino?
- How well does the database and API cope with the variability of meta-data with each item (for example, some products may have a weight, whilst others have a volume).
- If you propose future work that could be added to the system, how disruptive would that be; can you expand the schema in a non-destructive way?

6 Questions

As always, make use of the lab sessions and the forums: https://www.cs.bris.ac.uk/forum/index.jsp?title=COMS20700