ECE453/CS447/ECE653/CS647/SE465
Test Design for isin (v1)

Patrick Lam and Lin Tan

Django applications are database-backed web applications written in Python[1]. We'll describe the architecture of the sample `isin` app and its test cases. Most relevant for you are a Django app's *model* and *views*.

1 Application layout

The model defines the database schema. In this case, it is quite simple; there is a `Status` class, which contains three fields (one unused). As used, a status update consists of an 80-character string, `status`, and a publication date, `pub_date`. The database therefore stores a set of status updates.

The view presents and manipulates the information in the database. Since we're talking about web applications, the view produces HTML for output and accepts HTTP requests as input. Fortunately, you don't have to worry about any of that: Django can produce HTML from templates (see the `isin/templates/in` directory) and it can parse the HTTP requests for you.

The three public methods in `views.py` each correspond to an interface for the `isin` app. The `index` method simply outputs the most recent status. The `update` method allows the user to insert a new status into the database, stamped with the current time. The `quick_update` method uses the IP address of the requester to set a hardcoded status. See `urls.py` for the mapping between URLs and methods.

2 Test requirements

I find that it's useful to write down a bunch of test scenarios all at once and then to implement them. The benefit is that you're in the frame of mind where you're thinking of system behaviours and you can just write down a bunch of related behaviours without getting sidetracked by implementation details. Of course, you will go back and modify this list of test scenarios when you actually implement the tests.

I analyzed the software's functionality and created the following test scenarios:

- index: empty db;
- index: one status in db;
- index: two statuses in db, check that most recent one appears;
- index: programmatically add a status in the past, check that it doesn't appear;
- update: visiting page while not logged in doesn't display submit UI

[1]Django tutorials: `https://docs.djangoproject.com/en/1.7/intro/tutorial01/`.

- update: attempt to update while not logged in doesn't change status;
- update: attempt to log in a non-existent user must not succeed;
- update: attempt to log in with bad password must not succeed;
- update: visiting page while not superuser does not display update UI;
- update: visiting page while superuser displays update UI;
- update: POSTing an update while logged in as superuser results in a change to the status (empty db);
- update: POSTing two updates results in the newer update appearing in response to a request;
- update: putting a future update directly into db and then using the update UI doesn't replace the future update;
- quick-update: attempt to quick-update while not logged in does not display update UI nor does it update the status; and,
- quick-update: simulating a quick-update from cambridge's IP address updates the status.

3 Test case layout

Django supports a range of Python unit testing frameworks, including the default `unittest` framework. Unit testing frameworks are often quite similar; in Python's case, running the tests means running all methods in classes that subclass `unittest.TestCase`. As with all testing frameworks, each test contains some setup code followed by a number of assertions.

Django also supports the notion of a fixture. You used a fixture to import user data; it describes database contents, for instance in JSON format. Test classes can also import fixtures.

I created 4 test classes in the `tests.py` file: 2 for the index tests and 1 each for the update and quick-update tests. The `EmptyDBTest` tests the behaviour of the index where there is no fixture (i.e. an empty database). The `PopulatedDBTest` demonstrates the use of a fixture (`fixtures/isin_testdata.json`) and tests the index when there is already data in the database. We could have gotten by without that fixture but I wanted to illustrate the use of fixtures. The `UpdateTests` and `QuickUpdateTests` test the behaviour of the update and quick-update interfaces.

The contents of the tests themselves are fairly self-explanatory and you should be able to use them as a guide to complete Q4 of the assignment.