CinemaBuddy

COMP3207 Group Coursework: Team K

https://comp3207k.appspot.com/

James Crickmere (jc21g11)
Dom McCoy (dm2e11)
Dragos Popovici (pdp1g12)
Dan Rusu (dar1g12)
Stephen Griffith (sg26g11)
Bowen Zhong (bz4e14)
Yue Sun (ys6g14)

1 Prototype description

CinemaBuddy's original concept was to combine data from CineWorld, IMDB.com, and a potential number of other sources (BBFC, Rotten Tomatoes) to produce a web based social application aimed at movie lovers. Its users could search for films, view detailed information about the film (synopsis, poster, trailer, rating, classification) and screening times. Users would also select their favourite cinemas as well as the films they most want to see, which are given prominence on their personalised homescreen. This information could match other users who like the same films and are located nearby together, and can enjoy the film together.

CinemaBuddy offers a more personalised service than websites run by CineWorld or IMDB, tailoring its content to each user. Information can be combined from multiple sources, and recommenders can help users find new films in their interest area, giving the application a competitive edge.

The prototype design included an asynchronous Javascript driven user interface, to make the process of searching for films quicker and more responsive. This helps the application stand out from competing websites.

2 Tools and techniques

CinemaBuddy has been developed by a team of seven people. Developing as part of a team required working from the most up-to-date source code, and having a clear idea of who is responsible for completing a particular task.

The main difficulty in developing code together was working with differing timetables, deadlines and extra-curricular activities. We utilised the git source control system, hosted on GitHub.com, which was familiar to most of the team. We used a facebook group and google docs to keep track of who was doing each task and for general team communication.

Testing was continuous, and involved team members testing each other's code from a usability point of view. Unit tests, which are a form of black box testing, were written for the data importer during its initial development, which keeps the interface presented by this code as other team members expect.

The application server side is written in Python. Standards compliant HTML, JavaScript and CSS was used in the browser, with HTML generated dynamically from Jinja templates. Each member was allowed to choose their prefered editor with the following editors being selected: Notepad++, komodo edit, Sublime Text Editor and Atom Text Editor.

Libraries

Front end libraries used include Bootstrap and jQuery. The speed of development and attractiveness of bootstrap and the vast array of features included in the jQuery suit the needs of the project.

Server side libraries used, beside the standard Python and Google App Engine libraries, were Jinja and webapp2. webapp2 is a Model-View-Controller based framework for web applications, supported out of the box on GAE, but also portable to non GAE servers. Jinja is a templating language, allowing developers with little or no knowledge of Python to create HTML templates. It's a simple templating tool which is also supported out of the box by GAE, as well as being a library that works outside GAE.

The specification required the use of GAE. Some members of the team were very familiar with this platform, and everyone found its scalable database and easy debugging useful.

Development methodologies

CinemaBuddy was developed using agile methods, due to the shorter development cycles when compared to traditional methods. This is key in a project with a fixed deadline and limited time available for development.

Development was based around sprints, most commonly used in SCRUM to ensure that the most important features were implemented first and, most importantly maintaining a working product at the end of each sprint. This was crucial as it ensured there was always a working application that could be submitted.

Weekly meetings were held and were based on "Daily Scrum Meetings" in which every member of the team presented the progress they had made, and discuss with the group any problems they were experiencing, and receive extra help if needed.

The user interface was mocked up with wireframes, to establish an agreed layout before coding the frontend.

3 Statistics

Lines of code

The lines of code were calculated on *SourceA* and *SourceB* (original and modified code) using *cloc*.

```
james@james-xubuntu:~/git/comp3207k/zip$ cloc SourceA SourceB
      23 text files.
      23 unique files.
       3 files ignored.

http://cloc.sourceforge.net v 1.60  T=0.10 s (211.9 files/s, 19362.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                           7            287            181            609
HTML                            11             59             37            604
CSS                              1             20              0             94
YAML                             2             19              0             41
Javascript                       1             18              4             37
-------------------------------------------------------------------------------
SUM:                            22            403            222           1385
-------------------------------------------------------------------------------
james@james-xubuntu:~/git/comp3207k/zip$
```

Online tools such as Wave (http://wave.webaim.org/) W3C markup validator (http://validator.w3.org/) and JSlint (http://www.jslint.com/) were also used to assess the quality of code produced, identtifying accessibalitiy issues as well as HTML5, CSS3 and javascript errors.

4 Design

Model view controller

The database system used was Google's NDB, an object oriented database, which provides direct mapping to python objects, and makes expanding the database easy. Models defining the type of objects stored in the database were written as classes in *models.py*.

The hierarchical structure of the NDB was taken advantage of, with *FIlmTime* objects, representing the times of individual screenings, placed as children of the *Cinema* objects they belong to. This helped to improve the efficiency of queries as well as reducing the complexity of code.

Views, based on HTML templates, were placed in a dedicated *views* folder.

Controllers, which are *RequestHandler* subclasses, were placed in both *controllers.py* and *handlers.py*, with a common base handler for authentication functions.

Data Importer

An important part of the application is the 'importer', which fetches data from the CineWorld API and stores it in the application's database. A local copy of the data is essential for speed, reliability and to make sure the application can scale. This process is also important in reducing the load on CineWorld's API server; if this prototype gets

developed into a public application, it would become noticeable to CineWorld who could easily cut off access or start charging for their data.

Although this prototype only fetches data from CineWorld, a number of other sources, such as IMDB.com, could be used in the application. A 'framework' for developing other importers was considered, and the base class *Importer* has been designed to be a parent class for other imports beside *CineWorldImporter*.

The importer functionality is designed to run as a background process, using the google tasks system. There is also a feature which checks and stores the *Last-Modified* header in a *HEAD* request of the API server, so content is not necessarily downloaded and parsed when it hasn't been updated; this reduces load on both the application's servers and CineWorld's servers.

A major problem when deploying the application to Google App Engine was discovered when the importer task failed to run because it used up too much memory. This problem was traced down the the first part of the importer code which parses XML from CineWorld into a DOM tree, before any parsing or database storing. This means makes it harder to fix, as the external library *minidom* is responsible for using too much memory. This was not a problem that occurred on the development server.

A temporary fix has been implemented where the XML file from CineWorld has been copied to a private website, and made smaller in size, essentially creating a simulation of the CineWorld API with a reduced number of films and cinemas. In a public deployment of the application, computing resources beyond what is offered for free by Google App Engine would be needed to switch back to the real CineWorld API.

Search Engine

The search feature requires matching a search string to cinemas and films which. The Google Search API helped greatly in providing a way of indexing these objects, and text matching query string quickly enough to build a responsive AJAX interface for the search feature.

An indexing process, which, like the importer, is a background task, copies basic information from cinemas and films in the database to two indexes, one for cinemas and one for films. A simple request causes a rapid search of these indexes, and returns a JSON document, part of the AJAX mechanism used to display search results as the user types.

The Google Search API provides a scaleable way of full text searching that would be far too slow to run on the database, as well as expensive in the resources it uses (thereby also reducing costs).

5 Critical evaluation

This prototype fell short of the original specification; however important concepts from the original specification were still part of the work that was done. Complications with parts of the design, such as issues with the CineWorld importer using too much memory on GAE, were not predicted, and as such development went behind schedule and not everything was completed.

Our focus on producing a working prototype rather than a perfectly finished product, however, meant that the end result was something that looked and navigated very similar to the specification. Many of the potential pitfalls of deploying this application publicly have still been uncovered; these include the way importer scripts should be run, what other data sources to include, the effectiveness of Javascript on the front end, and the search interface.

The design could benefit from a more strongly AJAX based user interface, in particular with loading films. For this to operate quickly enough, the Google indexing system can be utilised.

Security concerns were not as rigorously enforced as they ought to be in a public application; the focus was on user interface and data harvesting. Sanitising the data from CineWorld and other APIs, especially HTML that could be injected, should be a part of the final application.

The group worked well together considering we haven't worked together before. There were no major conflicts and the group established a good working relationship.

However, one of the problems the group faced was arranging times when the group could meet, as there was different timetables and personal commitments. This resulted in members often working individually and simply updating the team at our weekly progress meetings instead of being able to ask for help and share ideas to problems, which slowed development.

Deadlines from other modules also slowed progress, especially in the weeks leading up to christmas vacation. If this project was to be repeated these outside factors would be incorporated into the project plan and the weekes at the start of term would have been used to reduce the workload required later.

Overall, the prototype demonstrates the most important part of the original specification, and a wide variety of issues have been considered by the team, if not also implemented in code. The aim of this project is to demonstrate the teams expertise in both server-side python, and client-side javascript. The prototype developed does demonstrates this, and the team as a whole has gained great experience to utilise in any future project.