CinemaBuddy

COMP3207 Group Coursework: Team K

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 uses could search for films, view detailed information about the film (synopsis, poster, trailer, rating, classification) and its showing times. Users can 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 is also used to "match" CinemaBuddies 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 based on each users individual preferences, and **sourcing complementary information from multiple sources and utilising recommenders** from CinemaBuddy friends, ultimately giving it a competitive edge.

The prototype design was to have an asynchronous Javascript driven user interface, to make the process of searching for films quicker and more responsive. This would help 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.

Our application server side is written in Python. Standards compliant HTML, JavaScript, 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, which the majority of the team have experience in using. The speed of development and attractiveness of the design made Bootstrap ideal for developing a prototype.

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 server. 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. This platform supports a large number of languages, has fast database system, and provides good debugging.

Development methodologies

CinemaBuddy was developed using agile methods due to the desirable characteristics they possess, specifically shorter development cycles when compared to traditional methods. This is key in a project such as this with a fixed deadline and limited time available to 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.

3 Statistics

http://validator.w3.org/
http://www.jslint.com/
http://wave.webaim.org/
python has pep8 built in
for lines of code http://pep8online.com/

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 two common base handlers, because people are wierd.

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 get 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 and 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 to0 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 Javavscript on the front end, and the search interface.

The design could benefit from a more strongly AJAX based user interface, in particular with search. For this to operate quickly enough, a more thorough indexing system would be recommended.

Security concerns - API data is unfiltered.

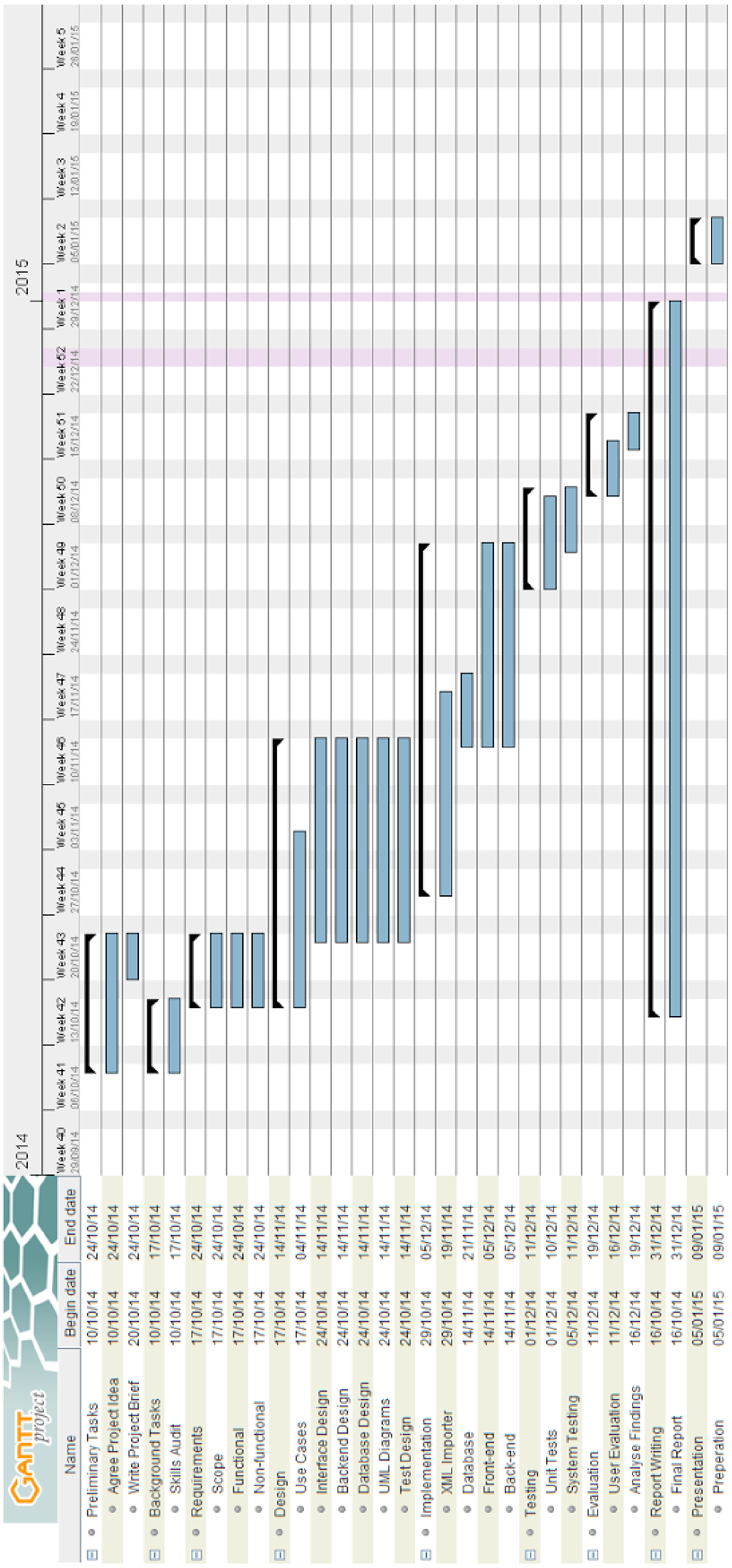
In terms of the group as a whole, overall we worked well together considering we did not have any prior experience of working together, and there was not any major conflict as often happens as new groups establish a working relationship.

The group however did face multiple problems, first of all it was often very difficult to arrange times when the group as a whole could meet due to differing 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 undoubtedly slowed development.

The group as a whole also failed to incorporate external factors when planning the project schedule, most significantly deadlines from other modules. This meant that very little progress was made in the weeks leading up to christmas vacation due to other deadlines taking precedence. 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.

PUT A SUMMING UP COUPLE OF SENTENCES HERE ONCE PROJECT IS DONE RELATING WHAT WE ACTUALLY GOT DONE, IN CONTRAST TO WHAT WE INTENDED TO DO, LINKING THAT AS A WHOLE WAS OK, AND THAT THINGS BASICALLY COULD'VE BEEN A LOT WORSE.

Appendix A: Gantt Chart



Name	Begin date	End date
Preliminary Tasks	10/10/14	24/10/14
Agree Project Idea	10/10/14	24/10/14
Write Project Brief	20/10/14	24/10/14
Background Tasks	10/10/14	17/10/14
Skills Audit	10/10/14	17/10/14
Requirements	17/10/14	24/10/14
Scope	17/10/14	24/10/14
Functional	17/10/14	24/10/14
Non-functional	17/10/14	24/10/14
Design	17/10/14	14/11/14
Use Cases	17/10/14	04/11/14
Interface Design	24/10/14	14/11/14
Backend Design	24/10/14	14/11/14
Database Design	24/10/14	14/11/14
UML Diagrams	24/10/14	14/11/14
Test Design	24/10/14	14/11/14
Implementation	29/10/14	05/12/14
XML Importer	29/10/14	19/11/14
Database	14/11/14	21/11/14
Front-end	14/11/14	05/12/14
Back-end	14/11/14	05/12/14
Testing	01/12/14	11/12/14
Unit Tests	01/12/14	10/12/14
System Testing	05/12/14	11/12/14
Evaluation	11/12/14	19/12/14
User Evaluation	11/12/14	16/12/14
Analyse Findings	16/12/14	19/12/14
Report Writing	16/10/14	31/12/14
Final Report	16/10/14	31/12/14
Presentation	05/01/15	09/01/15
Preperation	05/01/15	09/01/15

Appendix B: System Requirements

1. **Functional requirements:**
   - 1.1 The application must allow users to create an account
   - 1.2 The application must allow users to login using the account they created
   - 1.3 The application must allow users to search for their desired films
   - 1.4 The application must allow users to view details for the film they searched
   - 1.5 The application must allow users to view run time for the film they searched
   - 1.6 The application must allow users to find a match for their desired film(s)
   - 1.7 The application must display only films that are currently airing or they will be airing soon in cinemas.
   - 1.8 The application must display recommended films on the home page.

2. **Non-functional requirements:**
   - 2.1 The application must be easy to use
   - 2.2 The application must load relatively fast
   - 2.3 The application must offer support for various screen resolutions
   - 2.4 The application must run in different browsers