AMUSE: A Child-Friendly Multimedia Search Engine

Authors:
James Douglas Dowie, 1004790, *1004790D@student.gla.ac.uk*
Geraldine Goldie, 1005345, *1005345G@student.gla.ac.uk*
Ben Watson, 0606439, *0606439W@student.gla.ac.uk*
Wing Hang Li, 1001759, *1001759L@student.gla.ac.uk*
Guillermo Valverde, 0605024, *06065024V@student.gla.ac.uk*
Shahnawaz Shariff, 1000954, *1000954S@student.gla.ac.uk*
Group: E
Course: ITECH

ABSTRACT

Amuse is a web multimedia search engine that will allow for children of varying literacy skills to browse the web in a safe environment. Amuse will filter inappropriate content and will display search results in a visually appealing way, ensuring that even young children can enjoy it.

1. AIM OF APPLICATION

Amuse is an application that aims to provide a fun environment for children to carry out web searches while filtering obscene content. Children should be able to use amuse even if they have low literacy skills, this means that it will retrieve both pictures and videos and display them with a minimum use of text. In order to appeal to children amuse will display results in a mosaic format, with image size relating to the relevance of the search result. Furthermore amuse will be a MASHUP search engine, this means that search results will be retrieved from more than one source [1].

1.1 Requirements

aMuSe will be required to:

- Accept text search terms and return relevant results
- Filter these results to an age-appropriate level
- Display these results in an easy to read manner
- Return video, image and text results
- Allow users to choose any combination of the three result types at one time
- Retrieve search results from multiple search engines

aMuSe should:

- Allow users to create their own account
- Allow users to customise the interface of the application e.g the colour of font of text and save these changes
- Allow users to save some favourite search terms
- Be fun for children to use
- Have a sliding scale of filter level dependant on age

1.2 Design Choices

The target audience of aMuSe is English speaking children between the ages of 4 and 12. This means that the website should be easy to use to children who cannot yet read. The mashup component of the application will be achieved by retrieving results from the world's most popular search engines; Google and Bing. Finally the application will be built by expanding the existing PuppyIR application using the Django framework [2]. The application is intended to be published on the web and be free to use, ensuring that as many children as possible get to benefit from the experience.

1.3 Assumptions

We assume that children will appreciate searching for multiple forms of media at once and that many, in particular younger ones, will appreciate using a text-light website. We also assume that parents will be willing to help young children in the use of the website by, for example, entering keywords in the search bar.

1.4 Constraints

The application will only be in English and will not be designed to conform to all accessibility standards [2]. The interface must be usable and appealing to children.

2. APPLICATION ARCHITECTURE

The application has four key components that are described below (see Appendix A for an accompanying diagram).

Front end: The Client – The user's own browser

Middleware (1): The Web Server – Handles communication with the client and the rest of the system. Implemented with Django.

Middleware (2): PuppyIR – Provides tools for collating and filtering search results.

Back end (1): External Search APIs (Required) - Provide search results from Google and Bing.

Back end (2): Database (Desired) - Holds user account data. Allows for extra functionality including saving favourites, customisation details, and age-dependent filter variation. The data stored will be kept to a minimum in accordance with the data protection act.

2.1 Data Model: User accounts

Name	Type	Rationale
UserID	String (3 characters minimum)	Due to the age range of the users they can have very short user names; the only condition is this minimum limit and uniqueness.
Password	String	While no sensitive data is stored the data still needs to be protected, so a password is required to access the users account. We will not be storing the password itself but instead will keep a hash of it.*
Name	String (3 characters minimum)	A name is stored to allow for customisation of the interface. ** This name can be whatever the user chooses i.e. their first name or a nickname.
Date of birth	Date	The reason this is stored is to allow the PuppyIR to filter the results to an appropriate level for the user.

Figure 1: Structure of the user account database

* – We are keeping password restrictions to a minimum in order to reduce the memory load on children. This means that we are allowing for weaker passwords but given the nature of the stored data this is not cause for concern

** – We are not storing full names because we want to keep the amount of stored data to a minimum. The purpose of storing a name at all is so that the system can adopt a personal approach when communicating with the user which will not require their full name.

2.2 Data Model: Favourites

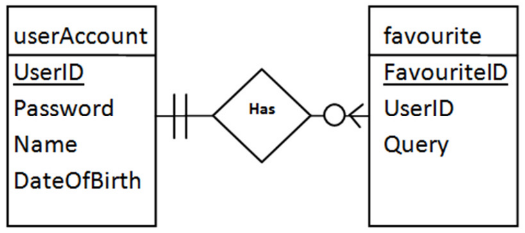
Favourites stored by users

Name	Data Type	Rationale
FavouriteID	Integer	This is a unique ID for the favourite in question; this is to ensure each favourite has a unique identifier in the database. It is not something the user of the system will ever see or have to interact with.
UserID	String (3 characters minimum)	A foreign key for linking to the user's account.
Query	String	This is the search query that has been added as a favourite.

Figure 2: Structure of the 'favourites' database

Originally the database was going to have a composite key between FavouriteID and UserID in order to speed up retrieving results. However, it was found that SQLite did not support composite keys so this was changed.

The system stores the favourite topics of users who have accounts. Each account can have no favourites associated with it or, as many as the user wants to have. The relationship between the two is as follows:



One and only one user account has zero to many favourites.

Default Favourites

In addition to the system storing favourites of users who have accounts there are also default favourites. These favourites are seen by all the users without accounts. Initially these will be defined by the team but it is expected that in the future these favourites will be dynamically updated either based on trending topics or usage data from users of aMuSe (i.e. to show popular queries).

Name	Data Type	Rationale
Default FavouriteID	Integer	This is a unique ID for the favourite in question; this is to ensure each favourite has a unique identifier in the database. It is not something the user of the system will ever see or have to interact with.
Query	String	This is the search query that has been added for this default favourite.

2.3 Description of Services Used

Django: A web development framework, using a pre-existing framework allows for rapid application development. There are some drawbacks to making this decision; it is necessary to learn the specifics of the chosen framework before starting work. It also creates a dependency on the framework where any flaws in it will be difficult to fix.

There are alternative technologies we could have used such as PHP, Struts, ASP.NET, Cold Fusion, Spring, Ruby on Rails and Tapestry. While any of these provide the necessary functionality to produce aMuSe, using Django allows us to build on the PuppyIR search engine. Allowing us to focus on the features that are exclusive to amuse instead of having to build an entire engine from the beginning.

Django also offers a pre-built system for creating and managing user accounts. aMuSe takes advantage of this fact and uses this system to handle all aspects of user accounts like the aforementioned tasks as well as authenticating login attempts.

PuppyIR: A search framework, funded by the European Union, providing tools that allow for the creation of search engines for children. It is built using the Django framework. Puppy IR offers a safe environment that is friendly to children, it is therefore an ideal foundation for aMuSe. PuppyIR is also used to ensure the results from the search APIs are in a common format (the open search format) that is usable by Django and by extension the rest of aMuSE and also that these results have inappropriate content filtered out[3].

The filtering used by aMuSe, as provided by PuppyIR, is as follows:

- Filtering the query itself and removing banned words, these are stored in a blacklist.
- Filtering the results retrieved by checking the title and description of the result against the same blacklist.
- Ensuring that textual content from the results (again the title and description) is containing banned words has these words censored i.e. if we banned the word 'tree' it would be displayed as ****.

Search Engine APIs
The system will use the search APIs of two major search engines (Google, and Bing) to retrieve results based upon the user's query [4, 5]. All three are used to ensure that the results are representative of the media related to the query in question available on the Internet.

3. MESSAGE PASSING

The format of our messages will be as follows:

- **Web Server Request Format** – HTTP GET requests; this is standard
- **Database Request Format –** Handled by Django, SQL requests are sent to the aMuSe database which uses SQLite. SQL Server or MySQL would be valid alternatives with extra functionality but Django provides more support for SQLite and we do not need the more powerful variants.
- **Search Query Message Format –** The messages sent to the search engine APIs are Restful requests, a full description of the format of these messages can be found on the respective pages about the APIs [4, 5]. This format was chosen because all three APIs support it.
- **Search Result Message Format –** The Results are returned from the APIs in the JSON format. The exact details are available on the APIs own pages [4,5]. JSON was selected as all three APIs as well as Django support it. SOAP and XML were possible alternatives. However, Django does not directly support SOAP so this would have added some overhead and XML parses more slowly than JSON [6].

4. CLIENT INTERFACE

The interface for amuse is designed to be both simple and fun to use. The most prominent feature is the mosaic display of search results, where each retrieved image or video is displayed as a thumbnail of varying size according to relevance. To the left of the mosaic is a list of favourite search topics or suggested topics, depending on whether the user is logged in or not. The user also has the option of selecting between different styles. Given the intended audience it is important that the interface is visually striking, yet it must also be easy to use. To ensure this we have made sure that all the buttons and input fields remain static.

4.1 User End Technologies

AMUSE will use CSS and HTML for the header and the topic menu, for the central panel with the multimedia results we will use AJAX.

CSS (Cascading Style Sheets): They are used to describe the presentation of an HTML document so that the presentation can be easily changed without having to alter the content. Using CSS will make it easy for the user to customise the design of the page. It will also help us make our website look good.
Alternatives: No other technology allows for easy customisation of a web page the way CSS does. We could conceivably use Flash, but it would be harder to implement and much less effective.

XHTML: (Extensible Hyper Text Markup Language): Will be used to define the content of the site.
Alternatives: HTML is also an option but XHTML encourages better practice and interfaces more effectively with AJAX.

AJAX (Asynchronous JavaScript and XML)**:** AJAX is a family of technologies that allows the developer to dynamically display information. It will allow for the
central panel of the website, containing the mosaic of multimedia information, to be dynamically refreshed without having to reload the entire webpage. This will be useful both to filter what type of information we want to be displayed (e.g. picture and videos only) and to display search results dynamically.
Alternatives: Flash or Java technologies. We concluded that using Flash would render the task needlessly complicated and using Java would hinder the performance of our website without bringing anything positive over Ajax. Furthermore, using flash would entail that our site could not be used by many iOS users.

4.2 Use Cases

Search
Description: The user enters a search term and receives relevant results of their chosen types.
Actors: Registered users, unregister users
Walkthrough: The user chooses any combination of video, image and text results using check boxes. The user then enters their search term into the search box and clicks 'Go.' They can then modify their selections of media type and, if they are a registered user, a new 'add to favourites' button is added to the
Add Favourites
Description: The user adds a favourite to the list on the side of the page.
Actors: registered users only
Walkthrough: The user types a search term on an empty text field above and saves it to the list of links under the "favourites" label on the side of the page.

5. DESIGN REVISION / FEEDBACK

Our feedback was complimentary about the level of detail we went into about our plans; however we were also criticised for being overly verbose and occasionally unclear. We have therefore tried to be more clear and concise in our final report.

A more specific but frequent comment in our feedback was about the lax restrictions on passwords. We had justified this by limiting the data we stored about users. However it was pointed out that user's age, name and favourites is data sensitive enough to merit higher security. Another commenter suggested that we use an alternative to text passwords to allow for higher security while remaining child-friendly. We have not implemented this but it is definitely a good idea and has been incorporated into our plans for future work.

Another common comment was that we had not made it clear how text was to be presented in the results. Due to a combination of this and our desire to avoid text where possible, we removed text results from our implementation. A related single comment identified that it was not possible to identify the difference between different types of result, this lead us to use coloured borders to differentiate between images and videos in our implementation.

We received a large number of comments about our wireframes being unclear. We have therefore made a point of using screenshots and labelling them more effectively.

Our sequence diagram was also criticised for not including details of what happened if a log in attempt failed, this has now been included as part of the diagram.

The feedback we received made it much easier to identify areas where the report lacked clarity, making it possible to refine what he had written. The feedback also made us realise some issues with our design itself which we were able to either solve or remain aware of for future work.

1. IMPLEMENTATION NOTES

1.1 2.1 Functionality Check List

Required Functionality

- Accept text search terms and return relevant results.
  - Successfully implemented
- Filter these results to an age-appropriate level.
  - A filtering system based on a list of unacceptable words is in place but the list has not been populated to suit any specific age group.
- Ensure results are displayed in an easy to read manner.
  - Results are presented as pictures which are sized according to relevance. Videos and pictures are differentiated using coloured borders.
- Return video and image results.
  - Successfully implemented
- Allow users to choose either or both of these two result types to display at one time
  - Successfully implemented
- Retrieve search results from multiple search engines.
  - Successfully implemented

Desired Functionality

- Allow users to create their own account.
  - The application does allow users to create their own account which stores a unique user name, a password and a nick-name.
- Allow users to customise the interface of the application.
  - The user can change the style of the page by switching the CSS used.
- Allow users to save favourite search terms.
  - Users can add search terms if they are logged in. This creates a button which will perform the search when pressed.
- Have a sliding scale of filter level dependant on age.
  - Unimplemented, we only have one filter list at this stage and do not store user's ages

1.2 Known Issues

- The site is known to have issues with Mac OS and Google Chrome
- The site has no admin, making it impossible to delete user accounts or recover passwords
- Users cannot remove favourites and without an admin section neither can administrators
- It is not possible for a user to retrieve a second page of results for a search
- There is no filtering of usernames. This means that is possible to register a username including profanity or a blank one. Currently there is a blank user in the user database which makes it possible to log in without giving any user details.
- The html does not conform to W3C Standards

- When a search is performed the whole page refreshes rather than just the results pane
- The layout is not as innovative as originally intended.

2. REFLECTIVE SUMMARY

The key lesson we learned from this project was the importance of version control. At the beginning and in the middle of the project, we did not follow an ordered process to keep versions consistent, this meant that merging our work was more time consuming than it should have been and features from earlier versions were sometimes lost in the later ones. By the end of the project we had developed a system which reduced the risk of this and the time required. One way to solve this for the future would be to use version control software like Subversion to track changes and perform merges. Subversion wasn't used in this case since it wasn't familiar to most of the group members.

The project also taught us how to work in parallel on different aspects of a system, using place holders for code which was still being written to allow people to keep working on dependant sections. As part of this we learned the importance of making sure everyone was aware of what the rest of the group was working on.

There were two desirable aspects which proved too difficult for us to implement. We had intended to use AJAX to ensure that only the results pane refreshed when a search was performed, however this has not been implemented. We got the server to return the correct JSON data, but could not find a way to make it call the necessary JavaScript to parse the results as well.

We had also originally intended to organise results in a more interesting and attractive way than we finally achieved. We experimented with using a TreeMap layout plug in for Jquery which produces a layout of rectangles resized according to a given variable. However this solution could not reliably keep the results understandable. We also attempted to use CSS commands to give results absolute positions but this proved too unpredictable without cropping/squashing images to a specific size.

One of the major issues with the project which we overcame was the team's lack of experience with Python. This meant that the code produced while functional, is not the most well designed code. There is a lot of code duplication between the search services classes for example. The next step in development, in this respect, would therefore be to revise all this code to improve code quality.

Regarding the databases, during development there was an issue getting the auto-incrementing primary key working (to be the primary keys of both the favourites database tables). SQLite by default adds in such a primary key if none are defined and so, this is what was done so there is no explicit primary key but this default one accomplishes the same objective.

We believe that the final display of aMuSe was a significant achievement as it is simple, friendly and colourful and so captures the user interface requirements of our audience. Furthermore we have made it possible for users to customise their interface in a quick and easy manner.

Another major achievement was scaling the results according to relevance and colour coding them according to type. This was significant because during the design phase we were not sure how this was going to be implemented and it was an important part of the system.

Our proudest achievement was including largely comprehensive error handling. Throughout the process we had problems where a search result returning no results or a user inputting incorrect log in details would cause an error page to show. As far as we are aware, there is no way to provide input which will lead to an error page in the current site.

5. SUMMARY AND FUTURE WORK

2.1 Future Work

Given further opportunity there are a number areas in this project we would like to improve and expand on:

- Solve the Known Issues
  - As described above there are a number of features such as browser compatibility that hinder the functionality and reduce the user's enjoyment of the site. Before incorporating any additional features into aMuSe we would like to resolve these initial problems.
- Scalable Filtering
  - At present each user has the same filtering level for their searches therefore we would like to develop methods to be able to alter this filter so that 12 year olds have more access, within reason, than 5 year olds.
- Make aMuSe more interactive
  - Currently aMuSe allows users to move the results and create collages however given further time we would like to make aMuSe much more engaging by including games e.g. they have to find a certain item on the page. Also, we would like to embed videos and allow users to save individual search results as opposed to the search term. These features and games would hopefully encourage users to stay on aMuSe, reducing the risk of them moving to a page that may potentially lead them to other inappropriate content.
- Enhance favourites menu
  - When a user is logged into their account they can only add a favourite by typing the search term

into the appropriate section which does not comply with our design goal of visualization over text. Therefore we would like to develop a method that would allow users to drag a video or image from the results into the menu bar and it would automatically save. This principle would also be preferable if we ever got single searches saving as well.

- Increase customization options
  - Currently there are only two themes available to users if they wish to customize their account; fiery and standard. We hope to expand on this to create a range of that user can pick from and even make it possible to allow them to make their own themes.
- Improve the login function
  - Currently our user account control relies on text-based passwords. This makes it difficult to ensure an adequate level of security while keeping the site accessible for children. We would like to use an alternative to text based passwords such as passdoodle, where users are asked to draw a picture on registration then identify which picture they drew to log in.

2.2 Summary

The application will provide a fun and useful mashup search engine for children. It will allow children to create an account, log in and out, customise the interface and save their favourite searches. Before the results are displayed they will be filtered to ensure they are suitable for children. After they are displayed the user will be able to restrict them according to media type if so desired. Search results will be presented in a mosaic with the size of tiles related to the relevance of the result.

The technologies used in this application are

Technology	Where it is used in application
HTTP	Used in message passing between client, web server and search engines
HTML	Web page format for normal content
XHTML	Web page format for Ajax content
CSS	Customisation of web pages for users
JavaScript	Client side scripting for dynamic content
Django	Framework for user accounts, accessing database and administration
PuppyIR	Extends Django framework to provide search engine and filtering results for children
Python	Programming language used by Django
Ajax	Used to dynamically update web page content
SQLite	Provides database for storing favourites information
Google and Bing APIs	Provide search results which will be mashed together
JSON	For exchanging data between search engine and PuppyIR

For future development the application could be opened for a wider range of users. These could be children who use a foreign language or children with disabilities. The ability to drag and drop would also be useful for the increasing number of touch screen users. It would also be possible to adapt amuse to be used for educational purposes. The safe environment and fun display would provide an excellent tool for children to learn about the internet and how to use it.

6. ACKNOWLEDGMENTS

Our thanks to the Leif Azzopardi, Owain Brown, and the anonymous peer-reviewers for their comments and suggestions.

7. REFERENCES

[1] Open Mashup Alliance, www.openmashup.org, accessed March 2011

[2] Web Content Accessibility Guidelines 1.0, http://www.w3.org/TR/WCAG10/, accessed March 2011

[3] Glassey, R. et al, 2011, PuppyIR unleashed: A Framework for Building Child Oriented Information Services.

[4] Google API, www.code.google.com/customersearch/v1/using_rest.html accessed March 2011

[5] Bing API, www.bing.com/developers/s/API%20Basics.pdf accessed March 2011

[6] Yahoo API, www.developer.yahoo.com/javascript/json.html accessed March 2011

Appendix 1 Design Diagrams

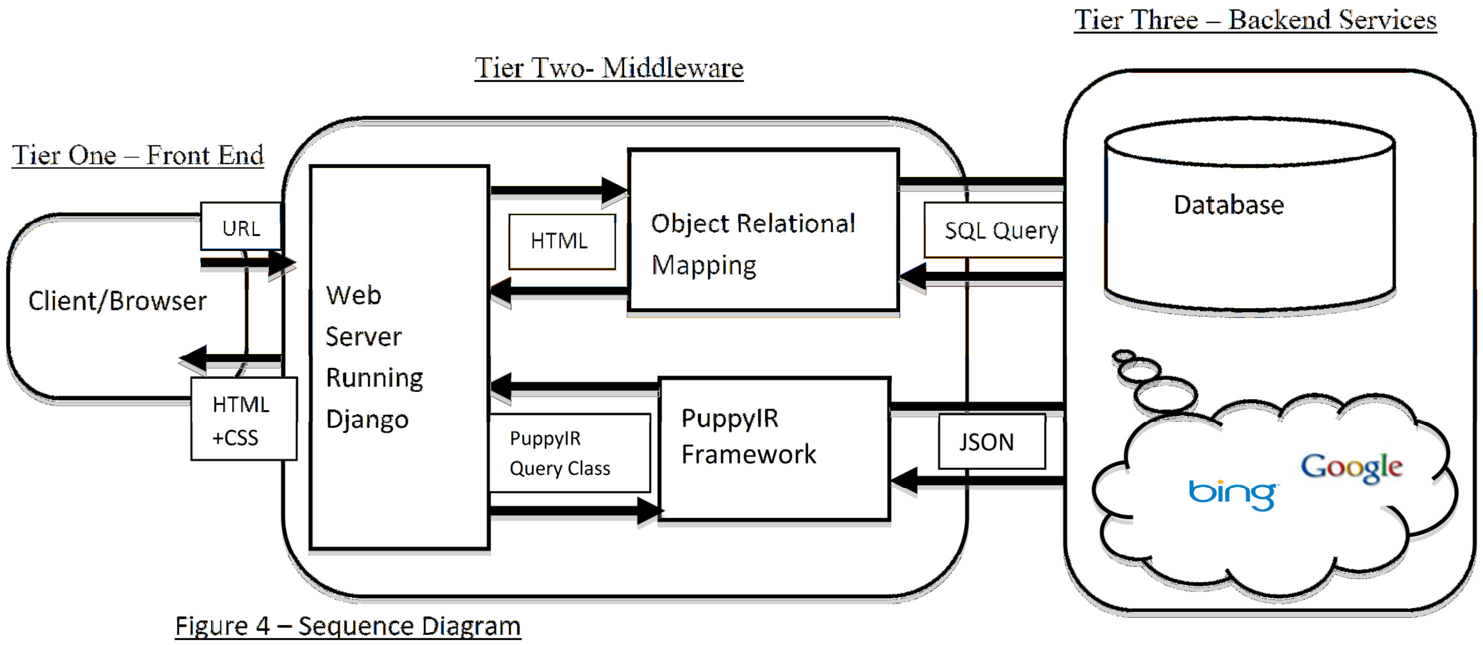


Figure 3 – N-Tier Architecture Diagram

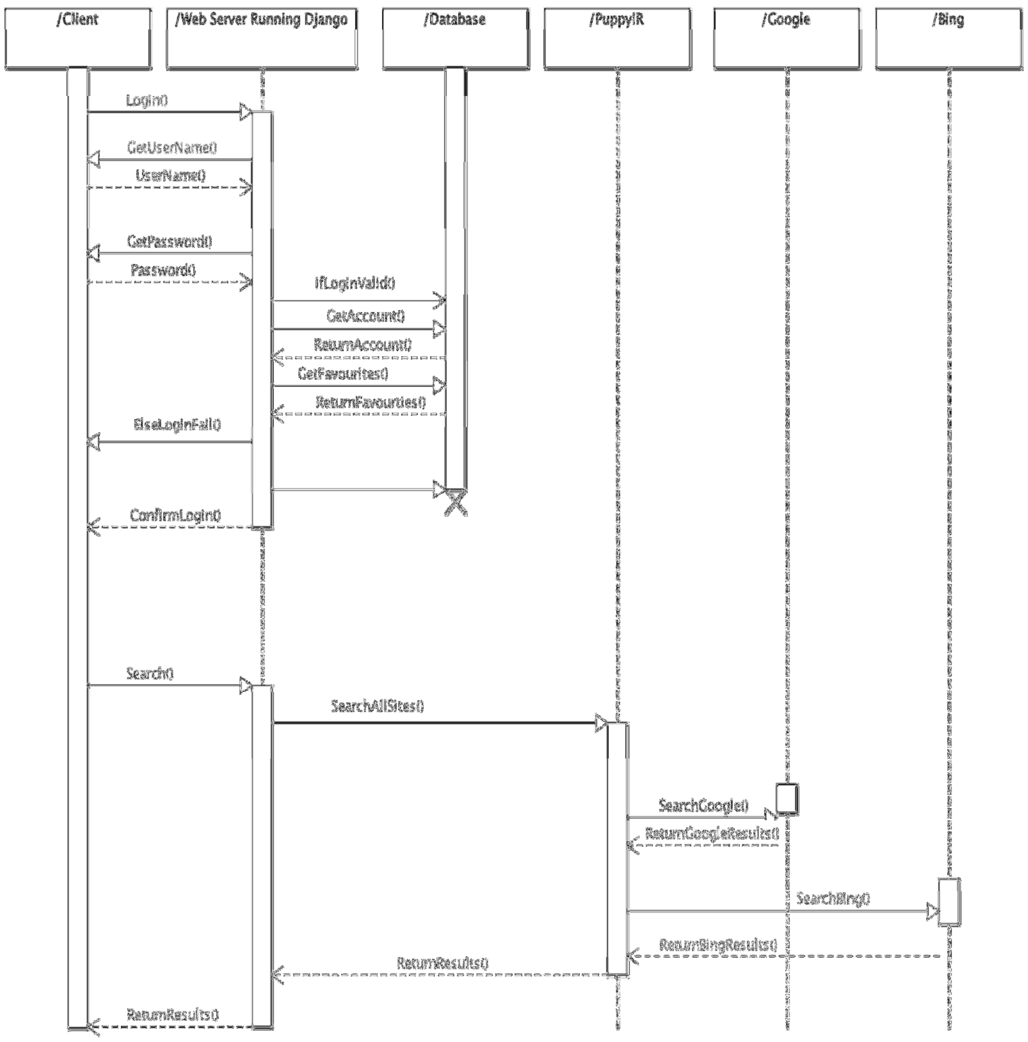
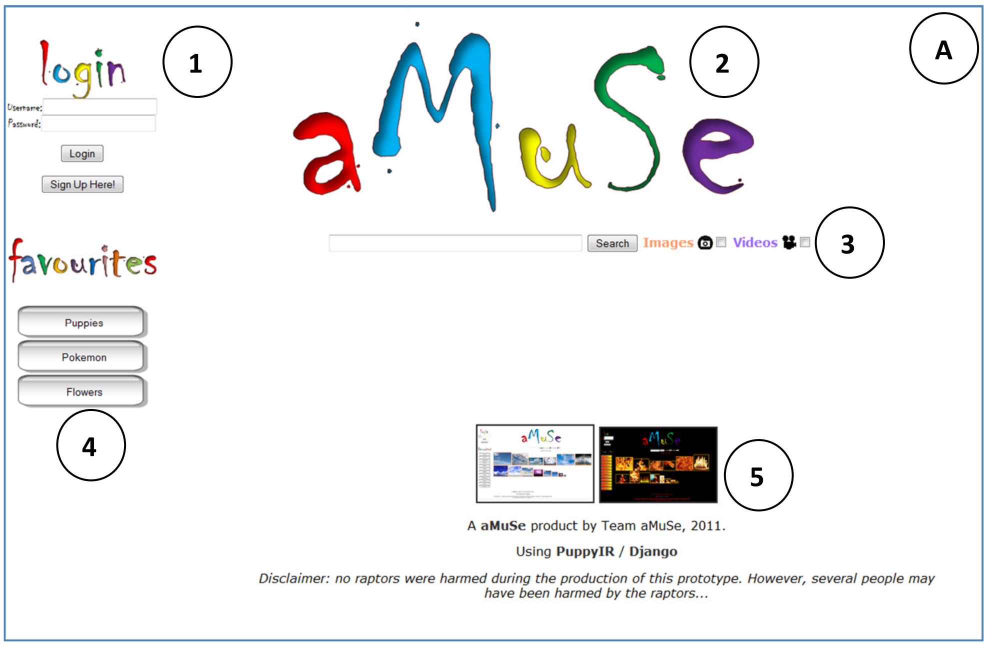


Figure 4 – Sequence Diagram

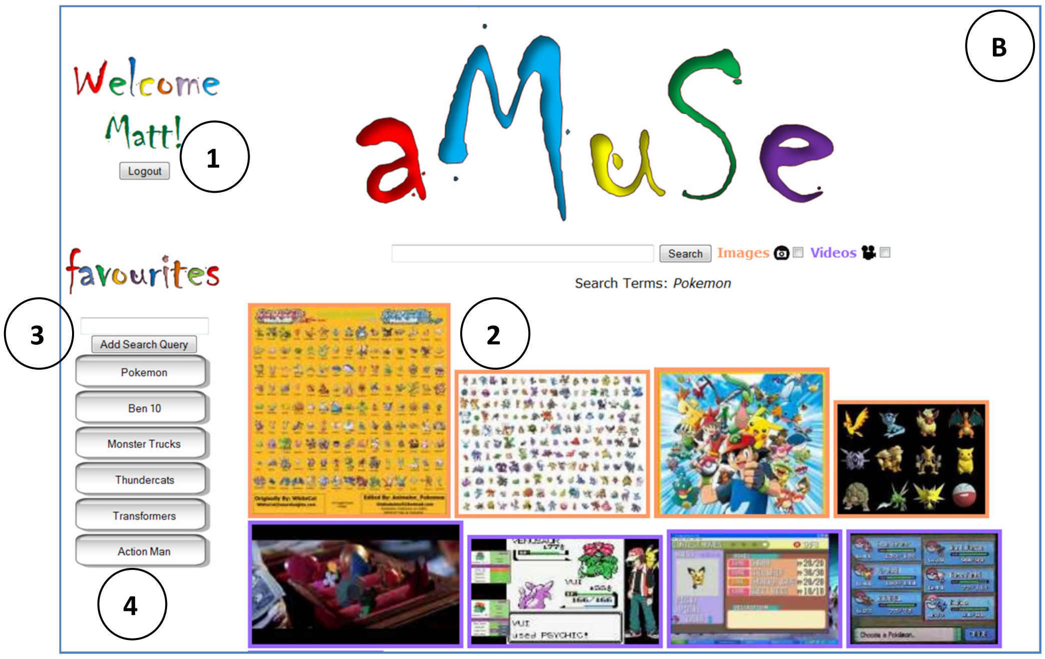
Appendix 2 Screenshots



A **aMuSe** product by Team aMuSe, 2011.

Using **PuppyIR / Django**

Disclaimer: no raptors were harmed during the production of this prototype. However, several people may have been harmed by the raptors...



<u>Screenshot A – Pre-search view, not logged in</u>

1. User can log in or sign up for a new account. **2.** Clicking on the aMuSe logo will return the user to the main page. **3.** Search bar where users and type search terms and select video or image only searches. **4.** Favourites sidebar which contains default searches since user is not logged in. **5.** Users can change the CSS used in this page by clicking one of the images.

<u>Screenshot B – Search view, logged in</u>

1. User is welcomed after logging in and can log out using the logout button. **2.** Search results are sized according to relevance and colour coded according to image or video. **3.** New favourite search terms can be added here. **4.** Saved search terms appear here and clicking on them will initiate the search.