Mash
Specification, Design and Implementation Report

Calum McCall, Euan Freeman, Aidan Smeaton, Simon Judd, Colin Ross
Team K
Dim3
0800410, 0808016, 0800972, 0707914, 0806035
{0800410m, 0808016f, 0800972s, 0707914j, 0806035r}@student.glasgow.ac.uk

ABSTRACT

Mash is a simple search engine which "mashes up" different types of multimedia results for a users' search query. Media is gathered from a variety of sources (see Figure 4) and presented to the user in a structured layout.

This web service aims to provide the user with easier access to different formats of search results, by combining the top results for a variety of different types of multimedia.

1. AIM OF APPLICATION

Mash is a web application which will allow users to search for a variety of different media types with a single query. This "mashup" search engine will provide the user with web results, images, videos, twitter updates and news stories.

The main design goal for Mash is to allow users to find the types of result they want. Different types of results will have to be combined and presented in a way which helps the user to locate results without difficulty. Result types will be customisable, allowing the user to hide certain types of media from results.

Time and resources are the largest constraints on this project. A limited time is available for implementation and the scope of this project will have to be manageable by a team of five. The team believe that the scope of the Mash project is reasonable and that the design goals are achievable in the time available. This project is reasonably complex in that it combines a wide variety of different web services and technologies to provide the different types of search results.

Functional Requirements

- Present to the user web results, images, videos, tweets and news articles relevant to their search.
- Allow the user to decide which types of multimedia they would like included in their search results.
- Maintain a record of a users last 10 queries so that they can revisit their recent search results.
- Provide different styles to the user to allow them to customise the appearance of their results.
- Allow the user to perform an "instant-search" - searching as they type

DIM3 2010, Glasgow, UK
ACM X-XXXXX-XX-X/XX/XX.

2. APPLICATION ARCHITECTURE

2.1 N-Tier Architecture

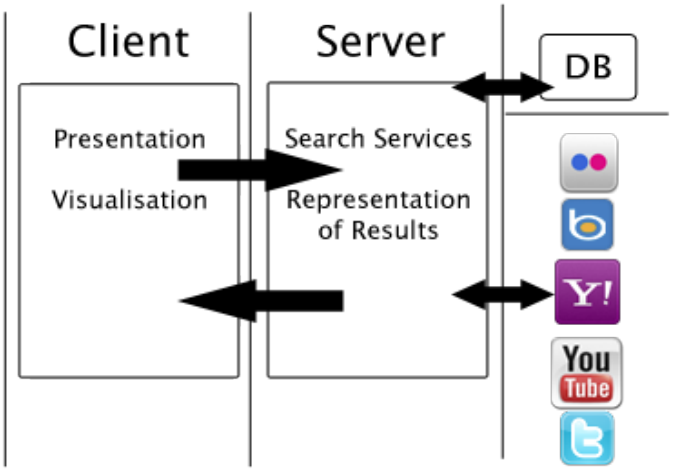

Figure 1: Diagram showing the tiered architecture of Mash.

Mash will be building on top of the PuppyIR framework [2]. PuppyIR already provides some simple web search services, providing support for Google, Bing and Yahoo. This project will build on that framework, providing support for different types of media.

2.1.1 Tier 1: Client

The Client Tier, as shown in Figure 1, is responsible for presentation and visualisation of search results. The visualisation of search results is concerned with the way results are structured and shown to the user. The client, primarily through the use of Cascading Style Sheets, also provides the presentation of results to the user.

2.1.2 Tier 2: Server

The Server Tier receives queries from the client and takes required action to search relevant search services to service the user. User preferences are fetched from the Database in Tier 3. These preferences include which services to use in the search. Results from the different backend services are prepared by this tier for presentation by the client.

2.1.3 *Tier 3: Search Services*

This tier consists of the variety of backend services used by Mash. More information about the services used is provided in Section 2.4.

2.1.4 *Database*

The database is responsible for storing user preferences, such as layout preferences and user accounts. The data model of Mash is described in further detail in Section 2.3

2.2 Data Flow

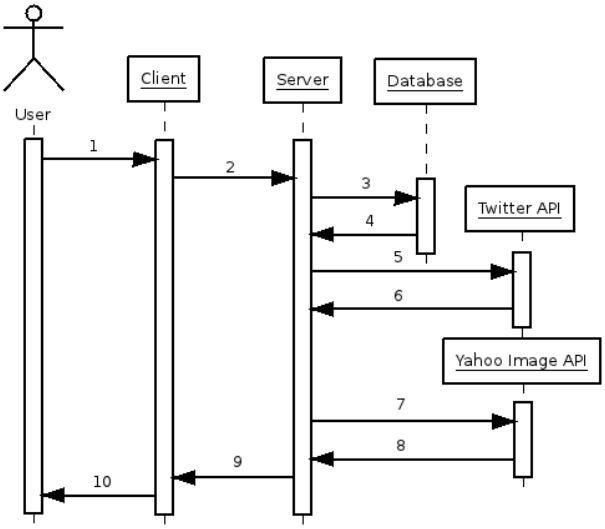

Figure 2: Example sequence of messages for searching tweets and images.

Figure 2 demonstrates an example of the communication which takes place when the user performs a search for tweets and images. Each message is as follows:

1. User enters a query, for example ”potato”
2. Web browser sends a POST request to the middleware
3. Ask database about user preference for results
4. Database indicates which type of results to get (tweets and images in this example) and how to display them
5. Send GET request to Twitter search API (see Section 3.2)
6. Receive and parse JSON response from Twitter search API [3]
7. Send GET request to Yahoo image search API (see Section 3.3)
8. Receive and parse XML response from Yahoo image search API [4]
9. Send a representation of the results to the client
10. Results are presented to the user

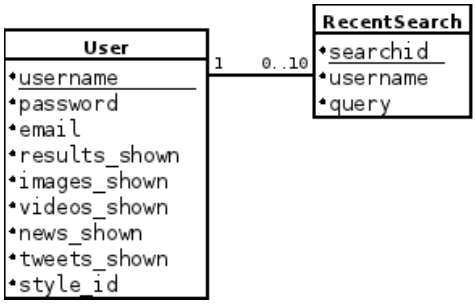

Figure 3: Entities within the system.

2.3 Data Model

Figure 3 shows the persistent data required by Mash. Users can opt to register with Mash in order to customise their content. The **User** entity contains authentication information in addition to user preferences for search result content.

The *results_shown, images_shown, videos_shown, news_shown* and *tweets_shown* attributes determine whether or not the user wants this type of result to be shown as part of their query results. Further information on how the user can customise their results is given in Section 4.

The *style_id* stores the identifier of the theme preferred by the user. Several content themes will be provided to the user to allow them to customise the presentation of their Mash results.

2.4 Backend Services

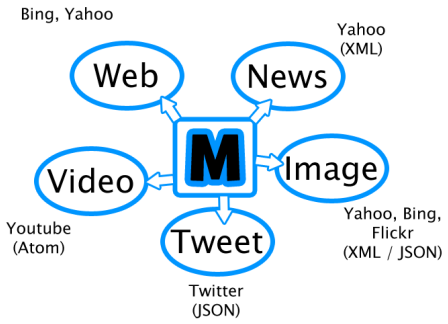

Figure 4: Different type of results and their sources.

As shown in Figure 4, search results will be coming from a variety of different backend services. Each of these data services uses different technologies and protocols for receiving requests and sending results. A more detailed explanation of the messages exchanged is provided in Section 3.

2.5 Design Choices

2.5.1 *Separation Of Concerns*

From the top down, separating Mash into tiers ensures there is a separation of concerns. This simplifies design as changes to one tier should usually require no changes to any of the other tiers, as the message passing between tiers uses

standard formats which are independent from the implementation details of each tier.

In addition to the above, separation into tiers allows Mash to scale more easily. Tiered applications are easy to split over physically different locations, increasing scalability.

Finally, the client side of Mash also separates the appearance of results from the structure of the web page. The use of Cascading Style Sheets (CSS) ensures that appearance is kept separate from the HTML structure of results.

2.5.2 Web Application Framework

We decided to use a web application framework, instead of developing the entire site from scratch as the use of a framework can reduce development time by freeing developers from writing boilerplate code.

However, these frameworks have a learning curve and developers must become familiar with each framework's practices and structure. Also, using frameworks can reduce flexibility as the developer must create their application in the way which the framework designers intended.

Django [1] was mainly chosen due to the team being familiar with Python. Other frameworks, such as ruby on rails, would require more effort to learn. As this project has a limited schedule, we cannot afford the extra time needed to learn a new programming language in addition to a new web framework.

3. MESSAGE PARSING

3.1 Messages

Mash will use a variety of web services to fetch information for the user. This section of the report will describe some examples of the messages sent to and from these web services. General web result searches will be powered by the PuppyIR framework. This section will only describe the services which additional work is required for. Example responses are shortened for brevity.

3.2 Tweets

3.2.1 Request

The Twitter search API [3] will be used for searching tweets. This service receives HTTP GET requests, of which an example follows:

```
http://search.twitter.com/search.json?rpp=1&q=hello
```

This request asks that results be returned in Java Script Object Notation (JSON), that only one result is displayed per page (**rpp**) and queries for the term "hello" (**q**).

3.2.2 Response

An example message received for this request is:

```
{"results":
[{"from_user_id_str":"25011349",
"from_user":"park1996",
"created_at":"Thu, 03 Feb 2011 08:21:15+0000",
"text":"Hello Hong Kong!",
"id":33077542797713408,
}]}
```

From this message it can be seen that "park1996" (**from_user**) tweeted "Hello Hong Kong!" (**text**). Twitter offers results in 2 formats: JSON and Atom. The JSON format was chosen as we believe it is easier to read than Atom.

3.3 Images

3.3.1 Request

Yahoo provide a web service known as BOSS (Build your Own Search Service) [4] which allows developers to use Yahoo to search for web results, news articles and images. BOSS uses a REST (Representational State Transfer) interface which allows querying through HTTP GET requests. An example of an image query for images of "Glasgow" is:

```
http://boss.yahooapis.com/ysearch/images/v1/
Glasgow?appid=APIKEY&count=1&format=xml
```

As specified by the **format** parameter, this request asks for a response in eXtensible Markup Language (XML). Only 1 image result is requested (**count**).

3.3.2 Response

A simplified example message received for this request is:

```
<?xml version="1.0" encoding="UTF-8"?>
<ysearchresponse>
  <resultset_images count="1" start="0">
    <result>
      <title>Glasgow From The Air 28</title>
      <url>http://farm3.static.flickr.com/2507/
           3768312403_b75574b262.jpg</url>
    </result>
  </resultset_images>
</ysearchresponse>
```

This message contains the search result for an image named "Glasgow From The Air 28" (**title**). Yahoo provide responses in 2 formats: XML and JSON. XML was chosen for variety, giving the team greater exposure to two competing technologies.

3.4 News

3.4.1 Request

Mash will use the Yahoo BOSS web service, previously described in Section 3.3, to search for news results. An example query for news results about "Australia" follows:

```
http://boss.yahooapis.com/ysearch/news/v1/
Australia?appid=APIKEY&count=1&format=xml
```

3.4.2 Response

A simplified example message received is:

```
<?xml version="1.0" encoding="UTF-8"?>
<ysearchresponse>
  <resultset_news count="1" start="0">
    <result>
      <date>2011/02/03</date>
      <source>Business Spectator</source>
      <time>04:37:24</time>
      <title>Australia scores record trade surplus
```

```
            for 2010</title>
      <url>http://www.businessspectator.com.au/bs.nsf/
           Article/UPDATE-1-Australia-scores-record-
           trade-surplus-for-DQ5BX</url>
    </result>
  </resultset_news>
</ysearchresponse>
```

This message contains a result for a news article titled "Australia scores record trade surplus for 2010" (**title**), from "Business Spectator" (**source**) which was published at 4:37am on the 3rd of February 2011 (**time** and **date**).

3.5 Videos

3.5.1 *Request*

Video searches on Mash will be performed by YouTube. The YouTube API [5] is queried using an http GET request. An example of such a request is:

```
http://gdata.youtube.com/feeds/api/videos?q=guitar
&orderby=viewCount&start-index=1&max-results=1&v=2
```

This request is searching for the term "guitar" (**q**) and asks for results to be ordered by their view count (**orderby**).

3.5.2 *Response*

The YouTube API returns results as an Open Search 1.1 compliant Atom feed. A shortened example response for the request in Section 3.5.1 is:

```
<?xml version='1.0' encoding='UTF-8'?>
<feed>
  <openSearch:totalResults>13740</openSearch:total...>
  <openSearch:startIndex>1</openSearch:start...>
  <openSearch:itemsPerPage>1</openSearch:items...>
  <entry gd:etag='...'>
    <title>guitar</title>
    <media:group>
      HIDDEN
    </media:group>
  </entry>
</feed>
```

As the **openSearch:** elements suggest, this query has returned 1 out of 13,740 results for "guitar". The **entry** element describes a search result. Within the **media:group** element is a large amount of data describing and providing resources about the image.

4. CLIENT INTERFACE

4.1 Description

At the top of every page is a header comprising of the Mash logo, a search bar (Figure 5, label 1), and a "Search" button. The purpose of this section is allowing the user to query Mash consistently from any page.

The left section allows the user to authenticate with Mash (Figure 5, label 3) by logging in to the system, and contains a panel which allows the user to determine which type of results they would like to see (Figure 5 label 2). This panel contains icons for each type of search result which, when selected or deselected, shows or hides that type of media from search results.

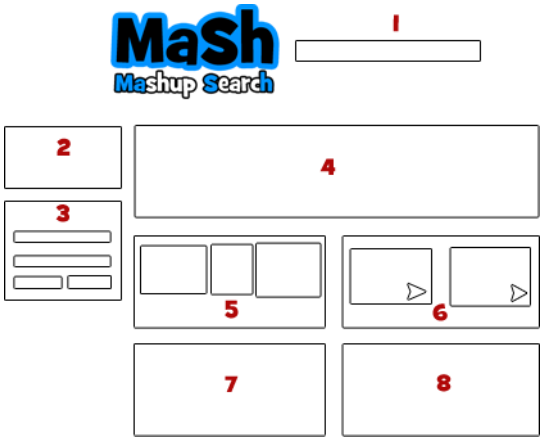

Figure 5: Wireframe showing an example layout of search results.

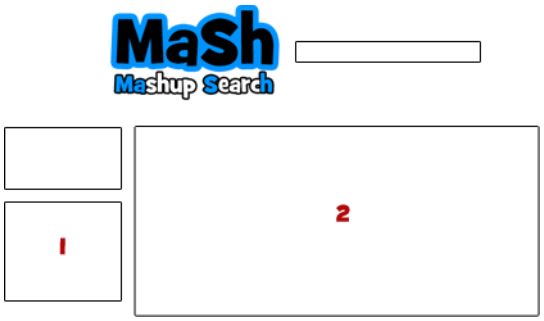

Figure 6: Wireframe showing a logged-in user displaying just one result type.

The main section is for presentation of results. It is divided into 5 different blocks, one for each of the result types. Figure 5 label 4 displays traditional web results such as the page title, a short description, and URL.

Figure 5 label 5 displays image thumbnails of image results. The next box displays video results (Figure 5, label 6) as a mini YouTube video thumbnail.

Below images and video results is a further box for news results (Figure 5, label 7), displayed as a news banner headline with brief synopsis. Figure 5 label 8 displays tweet results.

Figure 6 is how the interface looks both when the user is logged in, and when they have chosen to "See more" of a type of result. Now that the user is logged in to the system, the user preferences panel(Figure 6, label 1) allows them to log out, change password, and change the theme they are using.

Figure 6 label 2 represents a single result type. When the user chooses to view one type of result (for example, selecting a "See more" link), they are given more results than in the small sections depicted in Figure 5.

4.2 Walkthrough Example

- The user clicks on the search bar, and enters their query, "Galaxy Caramel".
- The user then deselects the "news feed" icon; they hate news.
- The news section on the interface disappears, and the space is filled in by the tweets section.
- The user then deselects the "Twitter" icon, and the tweets disappear too, removing the row.
- The user hits the enter key or presses the "Search" button.
- The interface displays web results for "Galaxy Caramel" in the top row of the main section. It displays images related to the search in the images section, and videos in the video section.
- The user clicks on any of the specific results, and they are taken to that webpage.

4.3 Dynamic Interface Components

As the user is able to hide particular types of results, the interface can dynamically change on request. For example, when the user "deselects" a result type, the icon for that result type is replaced with a desaturated one and the results of that type are removed from the main content pane.

As the user (de-)selects result types, their preferences in the database are updated so that Mash displays results as appropriate in future. When the result type is selected or deselected, Javascript will be used to dynamically alter the structure of the page. The server may be queried if necessary to get new types of results.

Users can drag-and-drop blocks of results into different positions on the screen, and results can be minimised rather than removed. The interface will update dynamically upon receiving instant search results.

4.4 User Experience

The interface is intuitive, it makes the search process simple using key words and icons. It makes use of standard conventions for search engines, such as the text entry bar and button with the "search" keyword, which users of other engines will be familiar with.

Icons are used to represent different types of search result. Each icon is simple, but informative, adding to the overall aesthetic appeal of the website, and making usability simple. The simple clickable icons with associated dynamic responses on the page, make the website highly interactive, and easy to learn and use.

4.5 Technologies Used

The user interface of the Mash web application will combine Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) to structure and present information to the user. The advantage of combining these technologies allows the separation of style from information. This makes it easier to design and maintain each of these components.

5. SUMMARY AND FUTURE WORK

5.1 Technologies Required

- **Django** Implementation of the Mash server-side
- **PuppyIR** Web search framework
- **HTML** Structure of search results
- **CSS** Style and appearance of the client interface
- **JSON** Search response from the Twitter Search API
- **JQuery** Client-side scripting
- **XML** Search response from Yahoo API
- **Atom** Search response from Youtube API
- **AJAX** Instant-search feature
- **OpenSearch** Standard for search results

5.2 Limitations

One limitation of this project is that only a limited number of search services will be utilised. While sufficient for the scope of this project, it would be ideal that a "mashup" search service uses a greater variety of sources for information.

A limitation of providing several different types of result at once is that there becomes less room for each category of content. Mash tries to mitigate this limitation by allowing the user to see more of a particular category of result.

6. PEER REVIEW FEEDBACK

The majority of the feedback our report received concerned the diagrams used. Example comments include:

> "The N tier architecture diagram could be clearer. I'd maybe add arrows just to show how the information flows between the different tiers - although this was made clear in your sequence diagram."

> "I'd also try to be consistent between diagrams. In the n- tier diagram tier 2 is called server, but in your sequence diagram and subsequent explanation it's called middleware which could be confusing for a client reading your report."

> "The ER diagram sets the cardinality of the User-RecentSearch relationship to 1 - 10. This gives the impression that there must be exactly 10 searches per user. Since each point on the link specifying the cardinality can have a range, I would have replaced the 10 with 0-10 (just a slight issue)."

> "I dont actually believe that this is a 4 tier design like you say. Tier 3 and tier 4 never communicate so they are essen- tially on the same level."

Each of these comments were taken on board and the appropriate diagrams updated. The architecture diagrams were modified to show flow of data and to remove confusion over the database and backend services being a different tier. The ER diagram was updated to show the cardinality of the relationship between users and recent searches. Some feedback, however, could not be taken into consideration.

> "Resolution of some images could be improved."

Unfortunately this is out of our control, as the imposed page limit required some images to be made smaller than we'd have liked.

Some comments which helped to guide implementation and inspired fixes to the report are:

> "...according to section 4.2 it seems as if users have to reload a search if they wish to see different types of content, while 4.3 indicates JavaScript is used..."

> "will AJAX be used to do it in the background?"

The team decided to settle on using AJAX to perform these requests - which would not require the entire search to reload when a new content type is enabled or disabled. The report was also updated to show this. A comment which, unfortunately, we did not have enough time to act upon was:

> "Allocation of space: While showing a variety of result types is useful, users (on smaller screens in particular) may not see many results of any one type."

We found the feedback to be mostly constructive and it helped to notice flaws in our document or system design which were perhaps not apparent to us. As documents like this are aimed at an audience which isn't the initial design team, it is important that content is not ambigious and having others review the document helps to identify these issues.

7. IMPLEMENTATION REPORT

7.1 Functionality Checklist

Web, image, video, tweets, news results
Complete.

Users can search for all of these type of results, and a further search type was added: Related Searches.

Web searches can be performed by Yahoo or Bing, and the user can choose to switch search engine dynamically.

Image searches can be performed by Flickr, Yahoo or Bing, and the user can change the result source, performing an instant search on their new choice.

Tweets, news results, video results and related searches all come from a single source: Twitter, Yahoo, Youtube and Bing, respectively.

Choose content types
Complete.

Users can show / hide result types. The data model is updated so that these search types are not performed until the user enables them.

Remember 10 last searches
Complete

Every time a (non-instant) search takes places, the search query is added to the data model. Only 10 recent searches are stored per user.

Different Styles
Partially-complete

The user may select from a handful of colour-schemes which dynamically updates the appearance of Mash. There was no time to integrate this with the data model, however. Theme choice is stored in a cookie instead.

Drag-and-drop results
Complete

JQueryUI was used to allow the user to show / hide result types and to drag-and-drop blocks of results to different positions on the screen.

Instant Search
Complete

Upon typing more than 3 characters, an AJAX request is sent to the middleware, allowing the user to perform instant searches.

View single result type
Incomplete

There was not enough time to implement this feature.

7.2 Issues

Searching works in most cases, however not all characters are escaped properly. Special characters such as '*', for example, are not dealt with. This issue is with the regular expression used for matching URLs. This causes an `HTTP 404` error, as no URL matches the pattern.

There was not enough time to add input validation to the Registration page, which may cause problems when users do not fill out all fields.

7.3 Future Development

Further work on Mash could see a larger number of search services used. Alternative sources of media could be made available to the user, for example providing the user with the choice of finding news from different websites.

A greater level of customisation could also be provided to the user. Although the user can drag and drop results-types into different positions, this layout is not persistent.

Another feature which could enhance the user experience of Mash is allowing users to specify which type of result they

would like to see a greater (or lower) proportion of. This could allow the user to request more news articles in their results, or less video results, for example.

8. SUMMARY

Using Django greatly accelerated the development of this web application. The Django platform allowed us to focus on providing functionality rather than having to write "boiler-plate" code to allow the application to run. The rapid approach to development helped greatly, as there were other technologies to learn as well.

A lot of technologies were used which were new to the team, including JQuery, Django and AJAX. These technologies were picked up quite quickly thanks to familiarity with other languages. JQuery and Javascript are similar to other imperative languages such as C and Java, and Django is just Python, with which the whole team is familiar.

We consider the greatest achievement of this project to be our peers voting us as the *Most Awesome Application* and *Best Demo*. It is rewarding that this application was so positively received.

Although we didn't have time to add all the features we would have liked, we are pleased that we managed to exceed our initial expectations of the project.

9. ACKNOWLEDGEMENTS

Thanks to the lab demonstrators for their feedback and advice on the design of this system. Thanks to the lecturer for his guidance and comments on the project. We would also like to thank the PuppyIR project for making the framework and documentation available for use in this project, and our classmates for their feedback on this report.

10. REFERENCES

[1] Django. http://www.djangoproject.com/.
[2] Puppyir framework. http://sourceforge.net/projects/puppyir/.
[3] Twitter search api. http://dev.twitter.com/doc/get/search.
[4] Yahoo boss. http://developer.yahoo.com/search/boss/.
[5] Youtube data api. http://code.google.com/intl/en/apis/youtube/2.0/reference.html.