DIM3 Web Application Report

Ross Eric Barnie,
Craig Cuthbertson,
Ross Taylor
Just. No.
DIM3
0901758, 1002386, 1002858
0901758b@student.gla.ac.uk
1002386c@student.gla.ac.uk
1002858t@student.gla.ac.uk

ABSTRACT
Provide a concise summary of the design of the application

1. AIM OF APPLICATION

- What is the purpose of the application?
- What are the assumptions about the aims and objectives?
- Describe the design goals and objectives of the application.
- What are the constraints of the project?
- Functionality List: i.e. what is the required and desired functionality?
- Reflective Questions:
- Is the scope of the application appropriate?
- Are the design goals realistic/achievable?
- How complex is the application?
- Is distribution across the web appropriate?

2. CLIENT INTERFACE

- Draw a wireframe of the user interface
- this may require several wireframes depending on the complexity of the application and the interfaces
- Describe the user interface.
- i.e. Label key input and output components: describe them.
- Provide a Walkthrough and explain the user interactions with application.
- i.e. use cases

Copyright is held by the author/owner(s).
DIM3 2010, Glasgow, UK
ACM X-XXXXX-XX-X/XX/XX.

- Describe the interactions associated with the dynamic components on the user interface.
- What calls are required to dynamically update the data on the client side?
- How does the user interface help the user achieve their goal, or complete their task?
- Is the user interface intuitive, appealing, usable, etc?
- What technologies are used on the client side?
- What are the reasons for your choices? i.e. what is the advantages and disadvantages of using this technology?
- What other options are there?

3. APPLICATION ARCHITECTURE

We used the N-tier diagram shown in figure 1 which shows the client (Web Browser), middleware (Django), the database and the external service (Youtube API), to design our architecture.

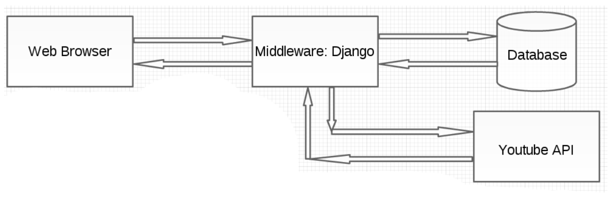

Figure 1: N-Tier Diagram

As can be inferred from the diagram, the clients sends requests for data to the middleware, this then retrieves the appropriate data from the database or the external services as required and the data is sent back through the middleware to the client.

Typically a request follows a structure similar to the following: the client makes a request for a web page; the middleware retrieves this request via specified url patterns which call particular view methods for the app; this view will then make calls to the database to retrieve appropriate data; the data will be taken by the view and then passed back to the web browser via a http response according to the template associated with the view; the client receives this and displays the information.

In order to design the data models for our application we created an Entity-Relationship diagram based on Chen notation [1]. This has been reconstructed and can be seen in figure 2.

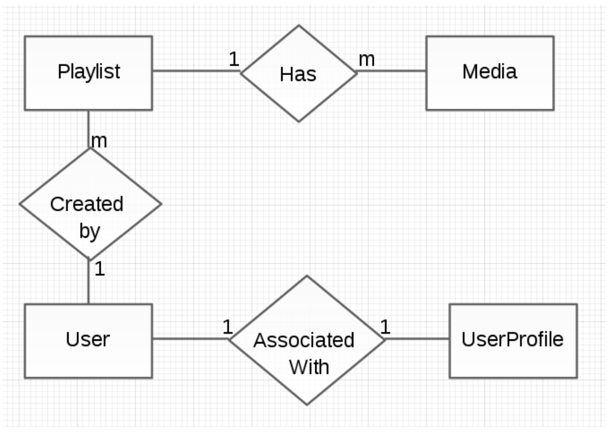

Figure 2: Entity-Relationship Diagram

- Playlist
  - Title: name of this playlist
  - Creator: reference to the User who created this playlist
  - Views: a count of the number of times this playlist has been viewed
  - Url: the title of this playlist encoded for use in a url
- Media
  - Playlist: reference to the playlist this media is associated with
  - Name: name of this media
  - Url: source url of this media
  - Source: the domain of this media eg. YouTube
  - Thumbnail: thumbnail image to be used for this media
- User
  - (default User model from Django, the following fields are the specific fields taken from this model)
  - Username
  - Password
  - Email
- UserProfile
  - User: reference to User assosciated with this UserProfile
  - Picture: image to be used as a profile picture

A key design element in consideration from the start of development was separation of concerns. Using many technologies (Django, Python, JavaScript, HTML, CSS etc.) it was paramount that each of these be kept as seperate as possible so that the source could remain readable and maintainable. Thanks to Django using the DRY (Don't Repeat Yourself) Principle [2] much of the separation of concerns was handled simply by using Django. However within this we set a static directory for the cascading stylesheets (CSS) and for images to be used for the thumbnail of the UserProfile data model. This kept the style of the HTML markup away from the markup itself and away from any javascript within that markup.

4. MESSAGE PARSING

- On the architecture diagram, Identify and label the main messages that will be parsed through the application.
- or alternatively (and preferably) include sequence diagrams to denote the sequence of communications parse between clients and servers.
- Describe the messages that are parsed back and forth through the application.
- For the main transactions - describe the payload of the messages
- i.e. What are the contents of the messages? i.e. include sample XML, XHTML, JSON, etc of one or two messages.
- What is the format of the messages?
- Why this format?
- What other formats could be used, what are the advantages and disadvantages of these other formats?

5. IMPLEMENTATION NOTES

5.1 Views

The main views we created for the project were as follows:

- Index - This view loads the index.html template and populates it with data for the current user. It gets a list of all of the playlists and then gets a seperate list of playlists which created by the current user. This view also deals with registering users and logging users in by saving the data entered into the relevant form on the index page.
- Playlist - The playlist view deals with the add_media function by verifying the form passed to the function is valid, extracting the data from the form and then performing the appropriate function calls to add the media to the database. This view also populates the context dictionary with the correct media when the playlist page is opened.

5.2 URL Mapping Schema

The URL mapping schema we decided to use was designed to be fairly easy to read and aquire information from. Every URL begins with the IP then '/allthemedia/'. The next section contains information about the type of page the user is accessing, for example '/login/', '/register/', '/about/', '/user/' etc. If this section contains '/user/' then the user is accessing a page with information relating to a specific user. This is followed by '/user/[username]/' to view the user profile of a particular user. This can be followed by a playlist title to view a playlist created by the specified user i.e. '/allthemedia/user/joe/coolsongs/' would display the playlist called 'coolsongs' which was created by the user called 'joe'. Anything beyond this stage specifies actions to be performed on a particular playlist, such as add_media.

5.3 External Services

The main external service used by our application is YouTube. This is used extensively by our 'Playlist' page, the main feature of which is a large YouTube player which plays the playlist media. This is created using the YouTube API which then displays the media from the current playlist.

5.4 Functionality

The folowing functionality is completed:

- Login/Logout/Register
- Create playlists
- View playlists
- Adding media to a playlist from the 'add_media' page
- Auto-playing videos in a playlist
- Skipping forwards and backwards through a playlist

The following functionality is unfinished, buggy or unimplemented:

- Adding to a playlist from the form on the playlist page - currently re-loads the playlist page with no playlist loaded and without adding the new media.
- CSS missing from add_media page - we had intended to add media from the playlist page and the add_media was supposed to be a temporary page. However, because this feature remains unfinished, the add_media page is the only way to add media to a playlist.
- After adding media to a playlist the re-direct goes to an empty playlist page with an incorrect URL.
- Adding media from sites other than YouTube is not currently supported.
- Collaborative playlist editing and Popular Playlists - unimplemented due to time constraints.

5.5 Technologies Used

For the implementation of this application we used the following technologies:

- Django - Django is the web application framework we used. This deals with a lot of the passing of data between pages and between the client and the server.
- Python - We used Python to write the files used by Django which define components such as the views, the models and the URL schema.
- HTML - Used to write the page templates.
- JavaScript - A number of scripts are used in the page templates, for example: the generation of the YouTube player on the playlist page is done within a script.
- CSS - The appearance and layout of the pages is handled using CSS.
- YouTube API - The YouTube API is accessed by the playlist page to allow our application to view YouTube videos and also to tell when the videos are finished to autoplay the next video.

6. REFLECTIVE SUMMARY

For the Implementation Report Only:

- What have you learnt through the process of development?
- How did the application of frameworks help or hinder your progress?
- What problems did you encounter?
- What were your major achievements?

7. SUMMARY AND FUTURE WORK

7.1 Summary

To summarise, our application currently functions as a video playlist system for Youtube videos. Users can register and log in which allows playlists to be saved and linked to a particular user. The user who created the playlist can also add more videos to the playlist. When a playlist is being viewed the videos will auto-play in order when each video ends. Users can also skip forwards and backwards through the playlist. The videos in the playlist will loop forever, provided the playlist which has been chosen contains videos.

Currently, adding to playlists can only be done through the add_media page and not from the playlist page as had been planned. Another limitation of the application in its current state is that only the creator of a playlist can add to it as opposed to our original idea which was to have multiple users able to collaborate on one playlist. We would also ideally have allowed users to remove media from playlists but this feature in unimplemented. The main limitation is that only media from YouTube is currently supported. We had initially intended for users to be able to add media from multiple sources but this wasn't implemented as it took us longer than we had originally expected to integrate the YouTube API.

7.2 Future Work

Given the limitations noted previously, there is plenty of scope for improvement and expansion of this application. One key feature we would have liked to integrate is support for media from multiple sources, not only YouTube. We had originally looked at sites such as GrooveShark and Soundcloud, however, streaming using the GrooveShark API requires you to pay so we decided this was not feasible. We would still like to include support for Soundcloud and similar services and may do so in the future.

Another feature we would like to implement in the feature would be the collaborative playlist editing as this was one of the features we liked the most when we first had the idea for the project. If this was implemented we would also need to add the ability for creators to remove videos from their playlists to prevent other users ruining their playlists.

8. ACKNOWLEDGEMENTS

Our thanks to the lecturers and demonstrators for their comments and suggestions. We would also like to thank the industry panel members for their comments and for taking the time to witness our demonstration.

9. REFERENCES

[1] P. P.-S. Chen. The entity-relationship modeltoward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, Mar. 1976.

[2] Cunningham and Cunningham, Inc. Dry principle. `http://c2.com/cgi/wiki?DontRepeatYourself`, 2013.