University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Web Application Development 2**			
Coursework Number	**4 (of 5) – Group Project Web App**			
Deadline	**Time:**	**6.30pm**	**Date:**	**1 April 2021**
% Contribution to final course mark	**25**		**This should take at most this many hours:**	**20**
Solo or Group ✓	**Solo**		**Group**	✓
Submission Instructions	**Via Moodle – see Page 4**			
Who Will Mark This? ✓	**Lecturer**	**Tutor ✓**	**Other**	
Feedback Type? ✓	**Written ✓**	**Oral**	**Both**	
Individual or Generic? ✓	**Generic**	**Individual ✓**	**Both**	
Other Feedback Notes				
Please Note: This Coursework cannot be Re-Done				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below. The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

Marking Criteria
See Page 5

Web Application Development 2

Group Project Web App (25%)

Introduction

The WAD2 Project is mainly based on the development of a web application of your choosing. Your web application should be developed using Python, Django, HTML, CSS and associated technologies including Javascript, JQuery and AJAX.

Implementation of your web app should draw on the skills built up during the development of Rango. Teams are free to use their own ideas, however there are some basic expectations:

- the app should involve user authentication;
- it should certainly interact with some kind of model stored in a database;
- it should be visually appealing and have an intuitive user interface;
- overall the functionality supported should be rich enough in order to allow you to demonstrate an understanding of the technologies listed above.

Beyond these guidelines, it is up to you, though if you do need some ideas, the following section lists some example projects from previous years.

Example project ideas from previous years

- *Rate My Beard*
  - Users with beards can upload photos inviting feedback; other users can login, view photos and give feedback
- *GU Grub Guide*
  - Users can leave ratings for restaurants and view other users' ratings
- *UnNoobMe*
  - Students seeking private tutors and tutors seeking clients are allocated to each other
- *Does my MSP Represent Me?*
  - Users can find out information about how their MSP is voting in Parliament
- *TripShare*
  - Users can post information about their upcoming trips, browse / search for trips and request to join a specific trip
- *Bargain Radar*
  - Small businesses can post their discounted deals; customers can browse deals and add them to baskets
- *SHAKESbeer*
  - Allows users to contribute cocktail recipes; users can browse and search over, and rate and comment on, cocktails
- *FilmBook*
  - Users can login and browse, search and rate movies; film producers can login and upload new films so that they can be rated
- *Federated Health Search Application*
  - People find out about particular conditions and to save the information that they find into different folders. The app lets people search across two different medical sites
- *Zombie Survival*
  - A game of search and survival. The web application needs to interface with a series of classes that create the world in which the player needs to survive.

The last two projects are described in more detail in Appendix 2.

Working in a team

You should think about how the tasks associated with the development of your team's web app should be divided. Try to ensure that activities are assigned so that every member of the team can be involved at all times. Here is a possible breakdown of responsibilities for a four-person team (this is just an example and you should feel free to organise things differently if you prefer. Remember, contributions can also be balanced with earlier design work and the forthcoming presentation):

1. user authentication, unit testing
2. models, Javascript, AJAX
3. views and URLs
4. templates and CSS

What should be submitted

You should host your application on PythonAnywhere and also ensure that the latest version of the code is stored in a public GitHub repository. The URLs for both of these locations will need to be submitted. You should ensure that your final application addresses the following requirements:

Deployment

- Application is deployed via PythonAnywhere and runs from there
- Requirements file is included and contains the correct packages
  - Within your main project folder you should include a file called `requirements.txt` that can be used to install the relevant packages within a newly-created virtual environment using `pip install -r requirements.txt`. You can create this file easily using `pip freeze > requirements.txt`.
- Database / migrations files not included
  - You should delete your database and migrations files so that when your tutor works on deploying your application on his/her local machine, the database can be recreated from scratch by making the relevant migrations and running your population script.
- Population script works and contains useful example data
  - Create a population script called `population_script.py` with enough relevant data in your main project folder.
- Application can be deployed on marker's own machine

Functionality

- Main functionality has been implemented (hopefully reflecting the design)
  - Here you should refer back to your design specification to guide development towards providing functionality consistent with what was envisaged.
  - If you have used external sources, you should acknowledge them in a README file in the top-level folder of your project. Failure to acknowledge external sources could result in a deduction of marks under the "main functionality" heading.
  - Although there are marks available for deploying your application on PythonAnywhere, the main functionality will be checked with respect to the version that your tutor deploys locally (i.e., the version on localhost).
- Application is bug-free / no error messages occur
  - The user should be able to access all of the functionality of the application without encountering Django error messages. You should thoroughly test your application to ensure that this is the case.
- Application includes some Javascript / JQuery / AJAX
  - Your application should not be based solely on Python and Django, and there should be evidence of usage of one or more of some of the technologies covered later in the course including Javascript / JQuery / AJAX.

Look and Feel

- Polished / refined interface, not clunky
- Uses a responsive CSS framework
  - CSS can be quite fiddly – to help with the layout and appearance of your web pages you are recommended to use a CSS framework such as Bootstrap.
- If browser window size changed, is the change handled neatly?
  - In the "Look and Feel" category, marks are available for an attractive and intuitive user interface with good use of CSS.

Code

- Templates inherit from base
- URLs are relative, i.e., use the {% url ..%} tag
- Code contains helper functions/classes (if required)
- Code is readable, clear and commented where appropriate
- CSS and Javascript are kept separate from templates (i.e., not inline)
- No repetition of code blocks in the views or templates
- Unit tests are included
  - In the "Code" category, marks are available for well-organised code, adhering to the principles that you will have learned during the development of the Rango app.

A sample marking scheme is included in Appendix 1.

Use of external sources

It is recognised that you may use external sources (e.g., Bootstrap) when developing your application. However these *must* be acknowledged when you submit (see below).

How to submit

After developing your web application you should host it on PythonAnywhere and ensure that the latest version of your code has been pushed to your team's GitHub repo.

One member of the team should then submit a one-page document via the "Project" submission icon on the Moodle page for the course, containing the following information:

- Name of web application
- Lab group number and team letter (e.g., Lab group 4, Team B)
- Team members (names and student numbers)
- GitHub URL for the app
- PythonAnywhere URL for the app
- Any external sources used (code, libraries, APIs), if applicable.

The person making the submission will be required to complete a Declaration of Originality on behalf of all team members when submitting via Moodle. If you have used any external sources, be sure to acknowledge them in your one-page document. For reference, the School's plagiarism policy is contained in Appendix A of the Undergraduate Class Guide.

Appendix 1: Sample marking scheme

The following marking scheme is intended to give a broad indication as to how marks will be apportioned. The actual marking scheme used in practice may deviate slightly from this.

Category	Marks
Deployment	
Application is deployed via PythonAnywhere and runs from there	4
Requirements file is included and contains the correct packages	2
Database / migrations files not included	1
Population script works and contains useful example data	3
Application can be deployed on marker's own machine	3
Functionality	
Site core functionality – an ambitious project has been undertaken	12
Application is bug-free / no error messages occur	3
Application includes some Javascript / JQuery / AJAX	5
Look and Feel	
Polished / refined interface, not clunky	6
Uses a responsive CSS framework	3
If browser window size changed, is the change handled neatly?	1
Code	
Templates make use of inheritance	3
URLs are relative, i.e., use the {% url ..%} tag	2
Code is readable, clear and commented where appropriate	3
CSS and Javascript are kept separate from templates (i.e., not inline)	1
No repetition of code blocks in the views or templates	3
Unit tests are included and are comprehensive, including testing views	5
Total	**60**

The total mark will be converted to a band which will be the team's mark for this component of the assessment.

Appendix 2: Example projects from previous years

In a previous year, there were four fixed project ideas and each team was required to select and work on one of those four. Specifications from two of these are provided below. These can help provoke your own ideas, and provide an example of the expected scope and complexity of the apps to be created in the project.
Note that these are examples to illustrate sample ideas and that diagrams etc might not be in the required format for any of your submissions

Federated Health Search Application

Description: The purpose of this application is to help people find out about particular conditions and to save the information that they find into different folders. The application lets people search across two different medical sites (medline and healthfinder) and the general web (bing). People using the application would like to self-diagnose, i.e. given some symptoms find out what are the likely conditions. They would also like to find out information about particular conditions, treatments and medicines.

Specifications:
Searchers need to be able to:

- create an account and profile
- edit, update and maintain their profile
- create a category (where they can save pages that they found)
- save pages that they find during the process
- search for conditions, treatments, medicines.
- share a list of links (via a category)
    - links and categories should be private, unless shared.

The search interface should provide:

- a mash up of results from the three search verticals
- tabs to enable access to the individual search verticals
- results should be annotated in terms of source (which vertical)
- a list of user defined categories should be available to select and browse
- an easy mechanism to save results to categories
- inline query suggestions to help uses with tricky words and spelling

When results are saved, the page should be fetched and stored. The readability of the page should be calculated, using TextStat's Flesch Reading Ease Score. The polarity and subjectivity of the text should also be calculated, using TextBlob.
When looking at saved pages, the searcher should be able to see the title of the page, the summary, the source, url, the reading score, polarity score and subjectivity score.

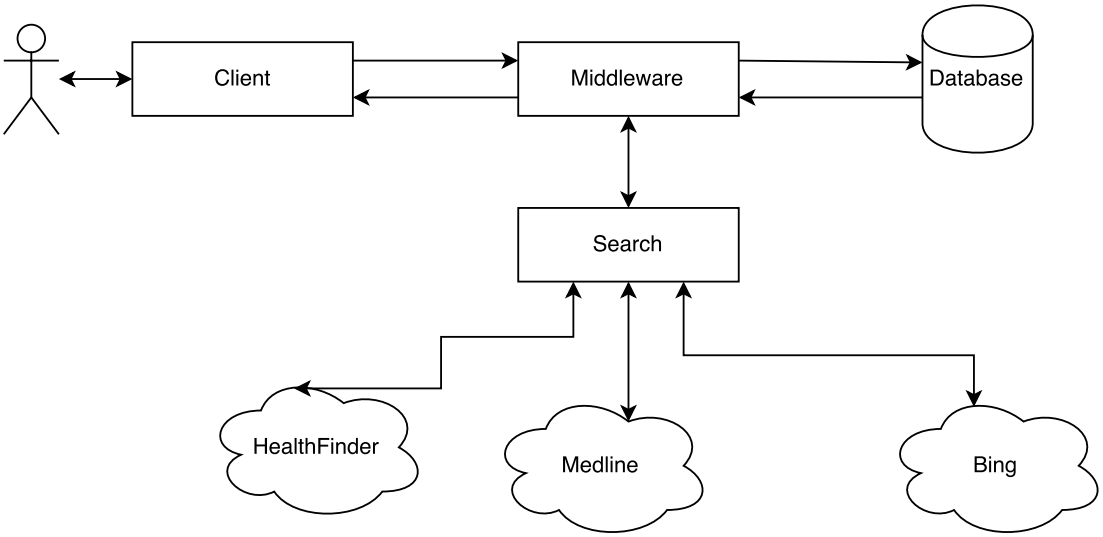
The API's for the federated search are:

- http://healthfinder.gov/developer/How_to_Use.aspx
- https://www.nlm.nih.gov/medlineplus/webservices.html
- https://datamarket.azure.com/dataset/bing/search

Additional resources:

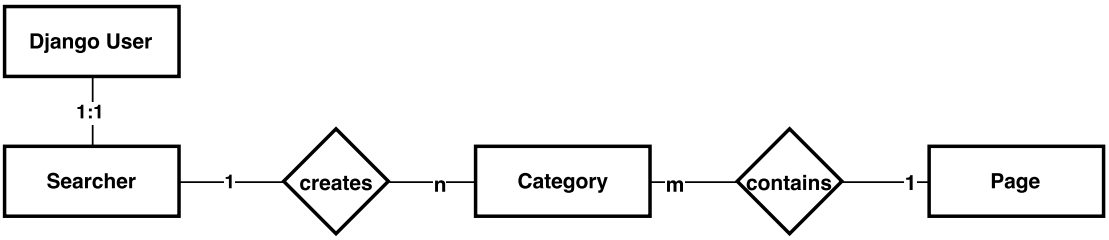
- https://pypi.python.org/pypi/textstat/0.1.4
- https://pypi.python.org/pypi/textblob/0.11.0

High level system architecture diagram:



HealthFinder, Medline and Bing are all external APIs. To provide some clarity I have split out the Search component that will interface with these APIs. And then my middle ware will use this service.

ER diagram:



Searcher(username, password, email)
Category(user, name)
Page (category, title, summary, url, flesch_score, polarity_score, subjectivity_score)

Zombie Survival: A Game of Search

Description: The purpose of this web application is to provide the frontend interface for a game of search and survival. The web application needs to interface with a series of classes that create the world in which the player needs to survive.

The game is as follows:

A player starts on day one of the zombie apocalypse. Their goal is to survive as long as possible (as measured by the number of days).

Their party size is 1. They have 3 units of food and 2 units of ammunition.

They enter a street. A street contains a number of houses. Each house has a number of rooms.

A room may contain other survivors, zombies, food and/or ammunition.

The player has a number of moves.

On the street:

(1) a player can choose which house they enter.

In a house:

(1) enter room
(2) move to next room
(3) leave house

In a room:

(1) search room
(2) leave room
(3) if zombies are present, they can fight or run.
    a. if they run, they leave the house and return to the street
    b. if they fight, they might lose the fight (lose party members), or win, in which case they claim whatever is in the room (survivors, food, ammo)
    c. the number of zombies killed is counted

The game mechanic for win/loss is based on the number of zombies, the amount of ammo and the size of the party.

During a day, the player has X units of time. Each action costs time. The day ends when they run out of time. At the end of the day, the game mechanic, decreases the amount of food depending on the size of the party. If there is not enough food, then party members might leave. Party size and food is updated, then they start the next day.

The web application needs to provide the following functionality.

Specifications:

Players need to be able to:

- create an account and profile
- view, edit, update and maintain their profile
- start a game and return back to a game in progress.
- Play the game according to the mechanics described above
- View the leaderboard

Visitors to the site:

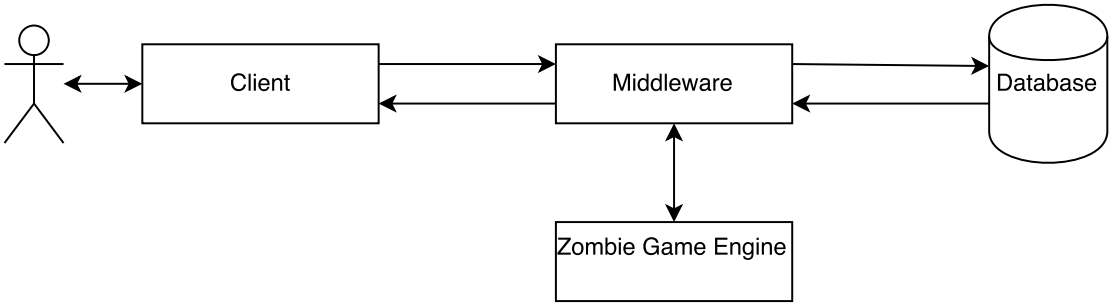
- Need to be able to view the leaderboard
- Read the instructions of the game
- Be able to register

The profile contains the players handle/username and picture along with how many games they have played, their average and best number of days survived and zombie kills. The profile also lists the badges that the player has achieved.

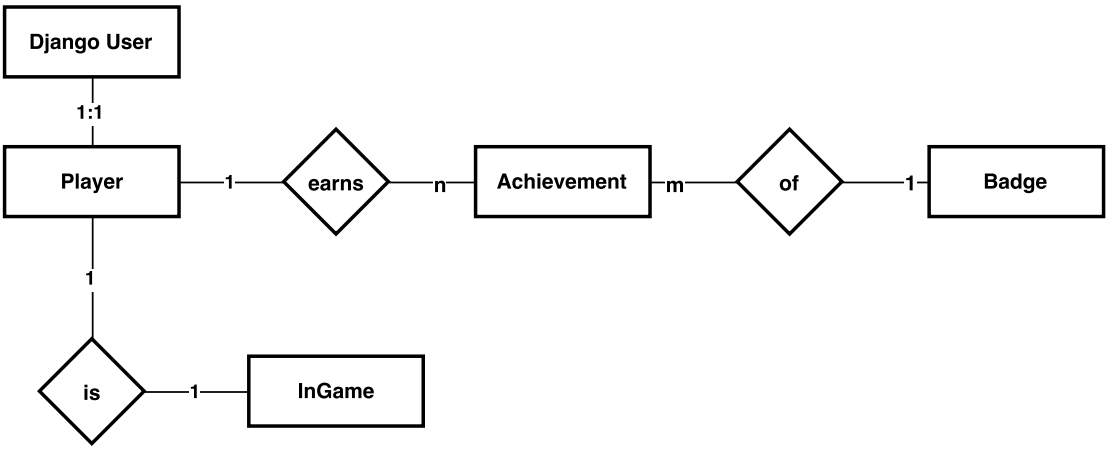
Badges: there are 4 badges with different levels (bronze, silver, gold): (i) Survival: survived 5, 10 and 20 days. (ii) Killer: killed 10, 20 and 50 zombies in one game. (iii) Stamina: played 5, 10 and 20 games, (iv) Party: had a party size that contained 10, 20, 40 people.

Leaderboard: there are two leaderboards: (i) top 20 players based on days survived, (ii) top 20 players based on zombie kills.

As the application will use the Game Engine provide we have made this explicit in the following high level system architecture diagram:



Here is an ER diagram:



We assume that a Player can only play one game at one time. Statistics on pasts games are recorded in the Player model.

Since they can only be in one game at one time, we can store the game in the player model, too.

Player(user, profile_picture, games_played, most_days_survived, most_kills, most_people, current_game)

username, password, email, are stored in the django user model.

Current_game stored the current instantiated game (tip: use the python pickle package to persist the game object, and save it in the Player model).

How the **Badge** is represented depends a lot on the implementation. My suggestion is below, but other justifiable versions are acceptable, especially if they work in an elegant fashion.

Badge(name, description, criteria, badge_type, level, icon)

Where *badge_type* (kills, people, days) and *criteria* is an integer specifying how many kills, people, days, etc have to be obtained in order to get the badge.

Icon is an image that represents the badge. The population script should ensure that the badges and their icons are all inserted into the database.

A class/handler is needed to determine if the *criteria* for a particular *badge_type* is met, or not.

Achievement (player, badge, date_awarded)