RANT - Social Networking and Blogging Application

Vera Berninger, Michael Clelland, Vicki Cole, Paul Finnighan, and Kelly Marshall

Group B - iTech

Vera Berninger 0604672b@student.gla.ac.uk, Michael Clelland 0307417c@student.gla.ac.uk, Vicki Cole 1009777c@student.gla.ac.uk,

Paul Finnighan 0107467f@student.gla.ac.uk, Kelly Marshall 1000930m@student.gla.ac.uk

ABSTRACT

RANT will be a social networking and blogging service that enables users to post and read messages called rants via a website. These rants will be text entries restricted to 200 characters and displayed on a user's profile page.

1. AIM OF THE APPLICATION

The aim our application is to allow users to provide text content that can be viewed by others. These posts are public and are not restricted for viewing, however a user must log on to post.

Required Goals
1. to allow users to view rant texts
2. register create and edit a user profile that enables them to create and store their own content
3. to allow users to post their own rants
4. subscribe to RSS feeds of rants from other users
5. rate other user's rants, view a list of top-rated rants and search for specific rants, and access those rants.

Additional Desired Goals
1. export the RSS feeds to other services or to formatted display such as occurs on Twitter's TweetDeck
2. provide a search function to find rants based on content
3. provide tagging functions for keywords for each rant

Assumptions
- our application will not be as scalable or have many of the functions such as those that exists on the popular blogging application, Twitter.
- It is assumed that the application will only be visited by adult users and will not provide functionality to moderate content, block foul language or screen out younger users, although this would ideally be something to implement in the future.

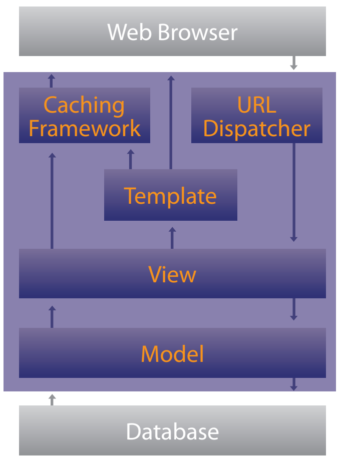
Constraints
- the allotted time of the course
- our team of five members
- the learning curve of using the Django framework [1] will be a technical constraint, since we have only recently been introduced to the framework via a tutorial.

It is hoped that by implementing only basic functionality, the goals of our application will be achievable in the time allotted for our course.

2. APPLICATION ARCHITECTURE

Our application architecture is expressed in the N-Tier diagram in **Figure 1.** A separation of concerns is provided between the display of the content, control functions and the data model. The client/user interacts via the web browser. When a user clicks to send authentication, registration or RSS feed requests, these are sent to the Middleware tier where the Django framework will handle these elements. The Django framework also handles generating queries to the SQLITE database. The Data Access and Integration tier will hold usernames, passwords and text of rants in SQLITE, returning any requests for data to the Middleware Server. The Middleware Server in turn updates the display back to the client/user.

Figure 1. N-Tier Diagram



Client - the client/user interacts with the system via a web browser

Django / Middleware - A URL dispatcher (urls.py) maps a requested URL to a view function and calls it. An optional caching framework can return an already cached page, if available.

The view function (views.py) performs the requested action, which usually involves reading or writing to the database. It may include other tasks as well. After performing any requested tasks to the model below, the view returns an HTTP response object.

The model (models.py) defines the data in Python and interacts with the database. The view then returns an HTTP response object through a template, which is responsible for the presentation, including references to any CSS style sheets. The templates return HTML pages.

Database - Our database will be implemented in SQLite3 and will store the data expressed in the ER diagram (Figure 2), including user information and individual rants and their associated attributes.

The Data Model

Our data model is expressed in the Entity Relationship diagram in **Figure 2.** Our main entities include auth_user and rant_rants. The auth_user consists of users, permissions, groups, and messages. It includes variables to store user details, user activity, anonymity, authentication, and methods for creating or managing users.

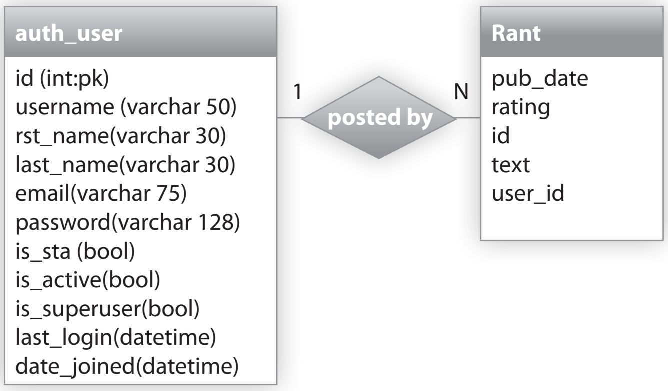


Figure 2. Entity Relationship Diagram

Django Development Framework

We have decided to proceed with Django [1] as a development framework due to its particular adherence to modern application methodologies that distribute system responsibilities according to Model-View-Controller elements (MVC). This provides a clean separation of concerns among the code that interfaces with data resources and databases (The Model), the logic for the various types of pages a user can visit (the Controller) and the HTML, CSS or other responses output to the client or user (the View). Django is particularly useful in rapidly creating models. There are many other frameworks that could have been used, including a combination of PHP and myAdmin, however our introduction to Django lead us to expect that creating data models and administrative views in Django would much more straightforward than if we were to use PHP.

Technology required for User Authentication

Through use of Django's in-built user authentication system we intend to provide our users with a registration, authentication and membership mechanism. [2] It allows us to create User objects with attributes such as username, first name, last name, password, email records, whether they can access the admin of the site, when their last login in was, the date they joined the site, and also whether the user account is active.

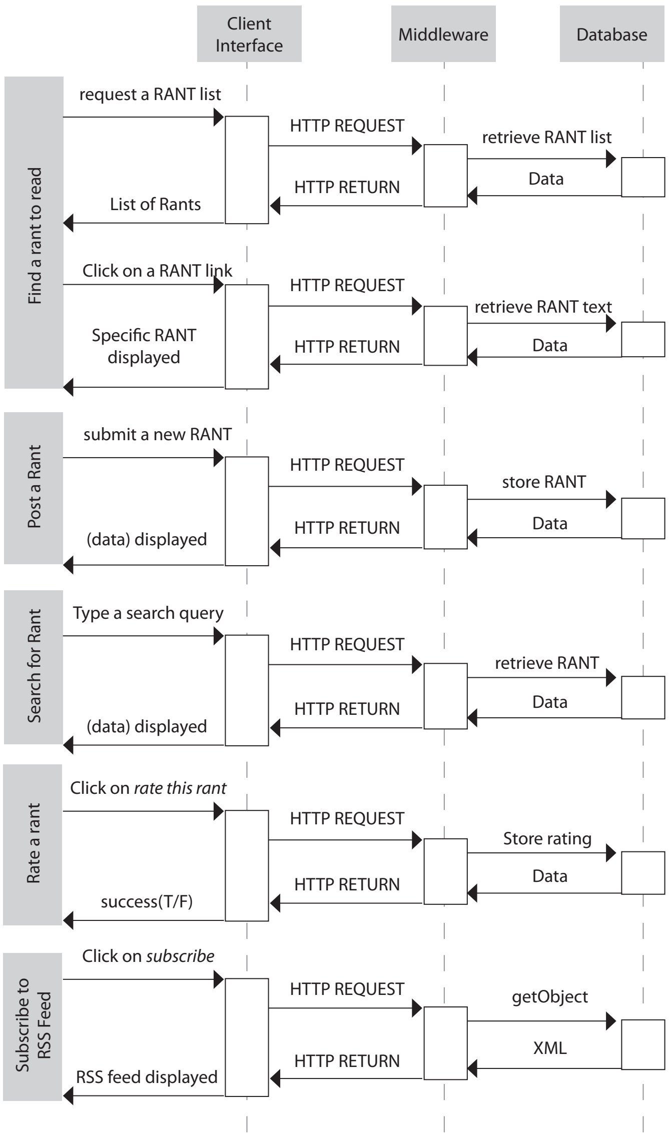
Django also provides us with several very useful bundled methods which perform a variety of functions including being able to set and check passwords, get profiles, and get messages for the user. Finally, the user authentication system also provides a series of pre-built django views for pages with user registration forms and other useful pieces of functionality.

3. MESSAGE PASSING

Http Requests and Responses

Examples of the various http requests and responses are expressed in the sequence diagram in **Figure 3.** When a user wants to find a rant to read, post a rant, search for a specific rant or rate a rant, an HTTP request is sent to the Middleware Server which in turn passes a query to the database to retrieve, store or rate that rant. This data is in turn fetched back with an HTTP response from the Middleware Server and updated in the user's browser.

Figure 3. Sequence Diagrams



RSS

Django offers it's own framework to include RSS feeds in a website. To implement this technology, all we would have to do would be to create a class defining which parts of our website we would like to include and an HTML file organising the XML code that Django generates automatically [3]. The technology would allow users to access new posts in a different way, outside of the website. It would be interesting to look into turning individually searched for items (someone types in "cat", they receive a page with rants about cats) into RSS feeds and thus giving the user some individual choices on what he or she receives. We may or may not have time to investigate a feature like this. JSON would have been another alternative, but Django automatically generates the feeds in XML. Using something else would mean changing the format of the feeds which would add an extra layer of complexity to our system.

4. CLIENT INTERFACE

XHTML

We will likely be using XHTML [4], or eXtensible Hypertext Markup Language, to create our project, which a subset of XML markup that extends HTML while reducing most of the complexity. XHTML differs from HTML in that all tags must be closed and all tags must use lower case letters. If we had the time, trying to create our application using strict HTML5 would have ensured that we were complying with the latest web standards. HTML5 differs from HTML4 in that it provides a selection of additional tags, for instance for navigation, includes multimedia elements, and has stricter parsing rules to handle errors. HTML5 is still not compatible with all browsers, however, so trying to implement it would have required providing alternate markup to provide backwards compatibility on unsupported browsers.

CSS

CSS, or Cascading Style Sheets, will be used to style the final presentation of our pages. [5] Django can access HTML and CSS files via any templates directory, or third-apps such django.contrib.staticfiles can be used to organize a large amount of static files, [6] (images, CSS, Javascript, etc.).

The User Interface Design

An effort has been made to keep the layout simple and consistent. This includes having the various content sections clearly divided. The top section contains a logo and login/signup area, and this area is kept small in height to allow plenty of room for the main content areas below.

The main content area contains the user profile information and that particular users rants. While the surrounding areas of the page use a darker colour scheme, the background is kept lighter on the main content area to draw the users in. A cartoon of a monster provides branding to the site and adds humour and focus around the main content area.

Top Rated rants are displayed in a smaller content area to the left, with a darker colour scheme with the aim of providing secondary focus for the user. A lighter text is used for contrast to provide easy reading, and a horizontal rule is used to provide separation between the individual rant listings.

The final implementation of the homepage and administration pages will be designed in a way that attempts to retain consistency in colour scheme, the grid divisions of the page and areas for main content and sidebars, and location of the login/signup link and user profile information. Our site will not provide alternate or user-customised CSS views, but this might be a function a future version of the site would benefit from.

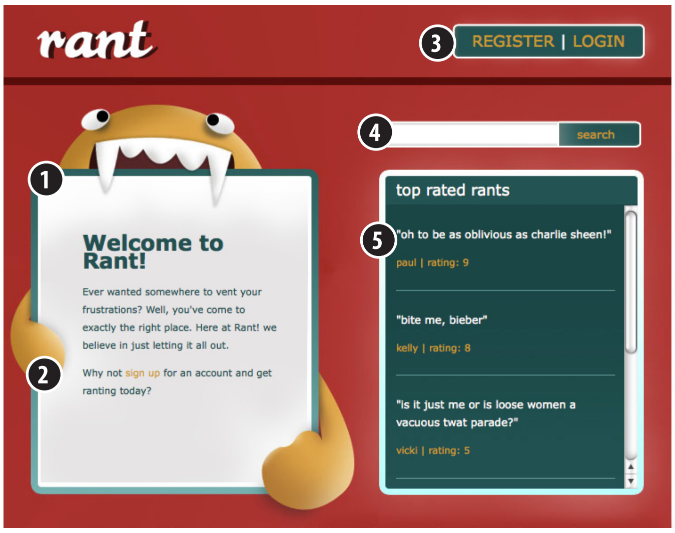
Use Case: Search for A Rant

- User navigates to the homepage
- User searches using keywords
- User chooses a rant to read from a selection of relevant displayed rants
- User reads the rant

Use Case: Post a Rant

- User navigates to the homepage (Figure 4.)
- User logs in and is taken to their profile/admin page (Figure 7.)
- user enters a title for their RANT
- User enters the text of their RANT
- User submits the RANT

Figure 4. Client interface showing the default view



Welcome to Rant!

Ever wanted somewhere to vent your frustrations? Well, you've come to exactly the right place. Here at Rant! we believe in just letting it all out.

Why not sign up for an account and get ranting today?

"oh to be as oblivious as charlie sheen!"
paul | rating: 9

"bite me, bieber"
kelly | rating: 8

"is it just me or is loose women a vacuous twat parade?"
vicki | rating: 5

Homepage Interface Design

❶ Upon loading the main homepage, a welcome message appears at the top of the main content area.

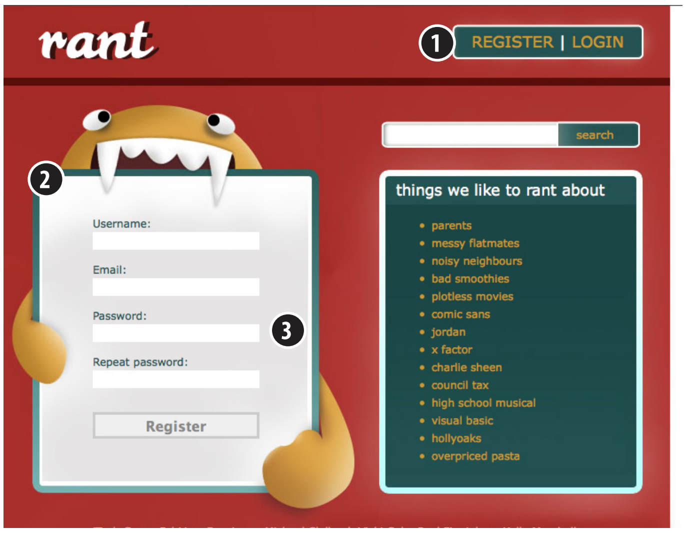
❷ A visitor is invited to sign up

❸ A visitor can also register or login with the button at the top right corner

❹ A search box allows a user to search for a rant topic

❺ Rants are loaded from the database and ranked in order of popularity

Figure 5. Screen capture of user registration form



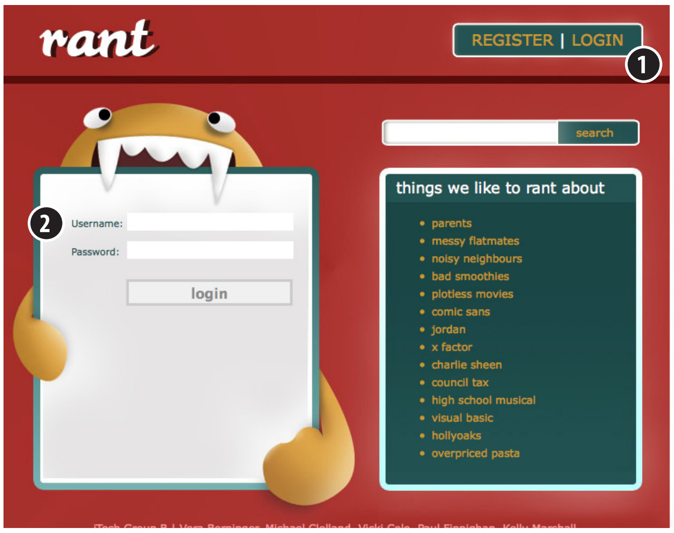
User Registration

❶ Upon clicking on the login button, this page is displayed.

❷ A form allows a user to sign-up to the site.

❸ The form fields validate to check that the username is unique, and the passwords entered in the two password fields match. The form also validates that an e-mail entered in the e-mail field is a valid e-mail address.

Figure 6. Screen capture of user login

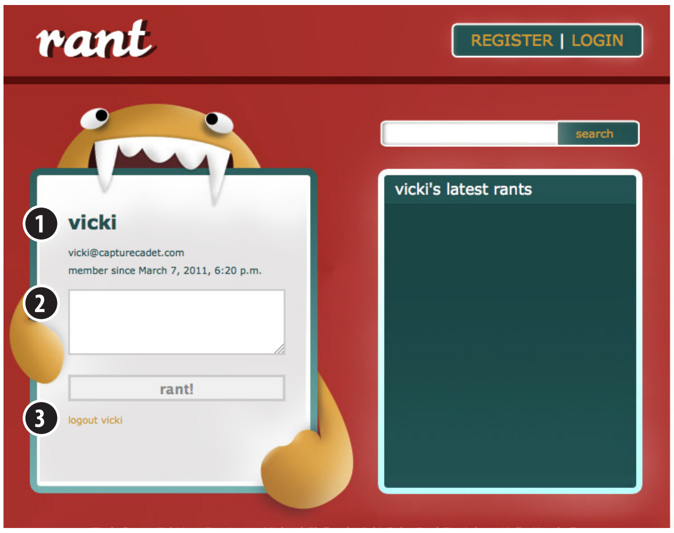


User Login

❶ Upon clicking on the login link, the following page is displayed so that a user can log in.

❷ When the user enters the username and password, it is checked to find a match in the database. If an invalid username or password is entered, an error message is displayed on this page and the user can make another attempt to log in.

Figure 7. Screen capture of rant entry form


vicki
vicki@capturecadet.com
member since March 7, 2011, 6:20 p.m.

Entering a rant

❶ When a user is logged in, their name appears at the top of the content area. Below that is their e-mail and the date they became a member.

❷ A text field is displayed where a user can enter a rant.

❸ A link appears for a user to log out.

5. DESIGN REVISION / FEEDBACK

Feedback:
We received feedback that some sections were a little verbose, or description was given where it wasn't really necessary and that the constraints section did not prioritize specific features. It does not explain which features are 'must haves', and which features can be dropped if time is short. There should be a clear list of requirements, ordered by priority.

Response:
Many sections were re-constructed into concise bullet points. Sections that were too technical were removed. Required goals have now been listed in numerical order. A list of additional desired goals is also provided in numerical order.

Feedback:
We received feedback that the sequence diagrams were overwhelming and or too large and that they should show the XML/HTTP messages that would be sent. There was also a question whether the sequence diagram "SQLite3" message passed to database was correct.

Response:
The sequence diagrams were greatly reduced in size. An example of an XML/HTTP message is now provided. The SQLite3 text was removed from the diagram.

Feedback:
One person gave feedback stating that we hadn't mentioned how we would make the app child friendly.

Response:
Our assumptions now include assuming that only adult users will visit the site and that no attempt to moderate the content or screen out younger users will be implemented at this point, due to time constraints and the complexity of adding this feature.

Feedback:
There was criticism that our ER diagram should include the attributes as part of the diagram, not just in a table underneath and that the explanation of the diagram was not very good, and the diagram itself didn't seem very well thought out.

Response:
The ER diagram was re-designed and now includes the attributes and attribute types listed under each entity. The Profile entity was removed, as it can be enveloped into the entity "auth_user".

Feedback:
We received advice that the wireframe descriptions could include a few more references to other HCI heuristics, such as keeping layout simple and having sections clearly divided and that it was shown in the diagrams but not really explained.

Response:
Description was added to the User Interface Design section in an effort to more clearly explain various design choices.

Feedback:
Both the N-Tier diagram and the sequence diagrams could (in parts) use more specificity in what technologies are being used in what directions.

Response:
Both Diagram have now been extended to include more technology specifications.

Feedback:
No ability to search for users based on all of a users rants, for example, top rated ranters.

Response:
It was intended for the usernames to be automatically dealt with as a tag and thus give users the possibility to search for them.

Feedback:
It is often mentioned that this project aims to imitate Twitter and that it will probably not contain as many features, but it does not highlight what new and interesting features this project will contain that Twitter does not have. If it does not have any new features, then it's not a very good idea.

Response:
In our assumptions section we explain that the application will not be moderated, leaving more freedom for users to write about whatever they want. It is also implied in the title that the main objective of the page is to contain negative, "ranting" content, which differentiates it greatly from Twitter's objectives.

Feedback:
No explanation of the impact Twitter has had on the world, and why web based blogs are important for keeping people up to date. It would make the project seem more worthwhile if they could explain it's intended uses and potential impact.

Response:
While the type of impact, Twitter has is important for that specific application, we do not believe that our own application will be comparable to this. As mentioned above, the main objective is to have people leave negative comments rather than inform or discuss as Twitter intends.

Summary:
As visible through our responses, we were able to use a large amount of the comments received from our peers to change and improve both our report and implementation. While we decided not to implement some ideas, as they did not seem feasible to us, they helped us reformulate our ideas in a way that readers were able to understand our decisions better.

6. IMPLEMENTATION NOTES

Working functionality:

- Login
- Rant model validates, restricting rant text to 200 characters
- Implementation of Django Session framework
- Implementation of Django User_Auth framework
- Username/password/email validation
- Implementation of Django template language to parse pages
- The register page checks for unique username
- Log in view dynamically displays username, email and member since date
- Showing existing Rants from the database on the main page in order of ranking, limited to ten entry per page and ordered highest to lowest
- Creation of Rant entries for every registered user via admin login
- Log out option makes use of Django default login view in auth.contrib
- CSS implemented successfully with each page template inheriting presentation view from base template.

Functionality that is not working:

- Posting a Rant
  - o Text box exists on the page
  - o Post button exists but form action not complete
- Ranking Rants (through website. It is possible to add a ranking through the database.)
- Viewing Rants by subject
- RSS feeds
- Adding a picture to the profile
- Searching for a Rant

Current problems and error messages:

There are currently no error messages as non-functioning sections are commented out. One issue of the current status of the site is that when the click rant button is pressed on the user profile it loads to another template called /postrant/ in the URL conf. This loads a view called def postRant, which is meant to create a new Rant object in the database. However, the main issue we had was that the loading of the new page kept logging the user out and causes there to be no current user ID available to insert as part of the Rant object. Another problem we encountered was grasping how to take information from the form on the profile page and use that inputted text block as the text attribute in the Rant object. After not managing to get this to work we tried implementing the ModelForm function on Django to create a form that way, but again there were errors relating to some of the field types and we were unable to fix these.

7. REFLECTIVE SUMMARY

While our group was able to design a very advanced and intuitive interface, we found it difficult to implement all the functionality we originally intended. Very early on we considered changing our topic but decided against it due to the work that had gone into the design. In retrospect it might have been useful to follow a simpler path to allow for more functionality.

Most of our group members had experience in interface design, while none of us had worked with Django or PHP before this semester. To make a decision on which technology to use the team was split up into two attempts and ended up with two half-working applications. With more insight and experience we would have been able to make a more informed decision faster and concentrate on the actual implementation.

Nonetheless the process made us all more familiar with many of the technologies used, including Python, PHP, CSS, XHTML, HTML, RSS and other we looked into during the planning period (such as JSON). We have been made aware of the importance to plan for implementation and programming more intensely and the importance to split work up between the team rather than working individually.

Most members of the group found the Django framework a great help to understanding a new language but had issues with understanding tutorials and other aids in connection with it. The main concern was to understand the Python syntax rather than the system itself.

It retrospect however it was very useful to all of us to have a tool that made separation of concerns obvious and easy to apply and thus helped split up the work and understand individual steps.

Our main success was the implementation of basic functionalities. This included the database, which was supplying output and user data storage as well as the login and registration functionalities. The greatest success in our opinion was the design of a professional-looking, intuitive user-interface.

8. SUMMARY AND FUTURE WORK

Our group found it hard to come to grips with a new language like Python having not used it before. In the future, we would like learn a lot more about Python syntax, how to construct ModelForms and pass their information into databases using POST methods, and also read up more on Sessions to properly implement the login system.

9. ACKNOWLEDGEMENTS

We would like to thank the lecturers and peer reviewers for their help in designing and implementing our application. We were also particularly grateful to our demonstrator and Leif for bringing us back on track and taking the time to go through specific issues with us.

10. REFERENCES

[1] Django - http://www.djangoproject.com
[2] User Authentication
http://docs.djangoproject.com/en/dev/topics/auth/
[3] XML Feeds http://validator.w3.org/feed/
[4] (X)HTML http://validator.w3.org/
[5] CSS http://jigsaw.w3.org/css-validator/
[6] http://docs.djangoproject.com/en/dev/howto/static-files/
[7] http://tinymce.moxiecode.com/
javascript rich text plugin for Django