International Journal of Engineering Sciences & Research Technology

(A Peer Reviewed Online Journal)
Impact Factor: 5.164

IJESRT

RESEARCHERID
THOMSON REUTERS

[Joseph *al.*, 14(4): April , 2025]
IC™ Value: 3.00

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

DEVELOPING A CONVERSATION-BASED PROJECT MANAGEMENT TOOL

Jennifer Joseph
*Keck Graduate Institute, United states
Corresponding Author Email: jenniferjoseph2050@gmail.com

ABSTRACT

The paper presents the development of a novel project management tool that integrates conversation-based functionalities with traditional ticket management systems. Traditional project management tools often segregate communication and ticket management, leading to redundant processes and fragmented information. The primary objective of this project is to create a system that streamlines project management by embedding conversation features directly into the ticket management process. This approach aims to reduce duplication of effort, enhance communication efficiency, and consolidate functionalities within a single platform. The development process involved researching existing project management systems, planning requirements, implementing the system using Django and various JavaScript libraries, and evaluating the tool through user feedback. The findings indicate that integrating conversation and ticket management can simplify workflows and improve project tracking, though challenges in balancing communication and process remain. Future work will focus on refining these integrations and exploring additional features further to enhance the tool's effectiveness in project management.

KEYWORDS: Management tool; Ticket management system; Conversation; Traditional project management

1. INTRODUCTION

In today's dynamic and ever-evolving work environments, successful project management plays a crucial role in helping teams meet objectives, stick to timelines, and use resources effectively. While traditional tools—like task managers and Gantt charts—are widely used to organize work, they often miss a key element: the importance of human interaction. These tools tend to emphasize tasks, schedules, and resource management, but they don't adequately support the ongoing communication that's essential for team collaboration.

The popularity of platforms such as Slack and Microsoft Teams highlights how central communication is to modern workflows. As remote work and distributed teams become more common, the need for seamless communication tools integrated into project management systems grows. Unfortunately, current tools often operate in silos, requiring users to switch between task management software and communication platforms, which can lead to inefficiencies and fragmented workflows.

This research aims to design a project management tool that integrates task management with conversation. The goal is to embed real-time, context-aware communication directly into the project management process, so team members can discuss tasks, make updates, and track progress without leaving their conversation space. By incorporating natural language processing (NLP), the tool will allow users to create and modify tasks through natural, intuitive dialogue, enhancing collaboration and making project management more fluid.

This paper explores how embedding conversational capabilities into project management software can improve team dynamics, streamline processes, and reduce delays. By fostering a conversation-centric approach, I seek to bridge the gap between managing tasks and fostering human connection, offering a more unified solution for today's collaborative, digital workplace.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

2. PRELIMINARIES

A project management tool is a program that allows a team of developers, managers, and customers to monitor the state of a project. These tools tend to be used to manage tickets. A ticket is a collection of items of data used to detail a fault, task, or feature. A ticket management system is a type of project management system that manages and maintains lists of issues documented through tickets. These are notices that members of a development team can set up, discuss code on, and see information about a problem, such as its severity and who is responsible for fixing it. Issues and bugs would be found and recorded in tickets. They can also hold information about tasks or features the team has to work on. Metadata refers to pieces of data that can be attached to a ticket and contain small pieces of information relating to the topic of discussion. Records of conversations in ticket comment threads help developers understand a ticket's significance and serve as a platform for discussion on active tickets.

3. MOTIVATION

When writing a software project as a team, it can be difficult to manage issues and bugs. This is generally done in the form of tickets as discussed in the previous section. However, it is often found that programmers discuss these tickets in an informal environment (such as Facebook or Skype) and, once a conversation detailing the issue is created, this work must be carried out a second time in the creation and comments section of a ticket. This is a problem as it duplicates the work that has already been done and wastes time. Consequently, project management tools can be cumbersome to use, making it difficult for users to integrate them into their work schedule. Existing management schemes also result in a fragmentation across the different tools being used. Although attempts have been made to make external metadata available within conversations, the opposite approach of using the conversations as a source of metadata remains unexplored.

AIMS

The project's primary objective is to build a conversation-based project management tool. This tool can then investigate the effectiveness of conversation-based tools in managing a project. Another aim is to examine how data generated within conversations can be used to simplify ticket management, without compromising the utility of traditional systems. Because traditional ticket and project management focuses very little on communication, a third aim is to reduce the number of tools a team must employ by providing many functionalities within one service.

OUTLINE

This dissertation will discuss what was achieved while developing the system.
Section 6- **Research**: This section discusses research on similar products and what was noted about their design regarding system planning.
Section 11- **Initial Planning:** This section discusses how the requirements were developed, and how, using these user stories, a user interface mockup and an ER diagram for the system were developed.
Section 16- **Implementation:** This section provides an overview of the system's implementation process, detailing the creation of different components such as the interface, chat system, and model layer and how the project progressed over time.
Section 27- **Evaluation**: This section details how the project was evaluated. It will also detail the results of the dogfooding, peer evaluation from another team using the system, and improvements identified from this feedback.
Section 31- **Conclusion:** This section will detail the current state of the system, whether the project's aims were achieved, and potential future work that could be done.

RESEARCH

Numerous systems have been developed which aim to improve the way in which software projects are managed. It was found that ticket management systems tend to fall into three categories: ticket centric systems, conversation-based systems, and systems that focused on task management rather than tickets specifically.

TICKET CENTRIC SYSTEMS

A. Bugzilla

Bugzilla [1] is a bug tracker and testing tool created and used by Mozilla. It integrates with several version control systems such as Git and SVN. Some interesting features include automated reminders, whereby when bugs are

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

reported but left unresolved the system can be configured to send regular emails to specific people informing them of these bugs. This system is known as the "whine" system. When any change is made to a known bug, this change is added to an automated bug report. Bugzilla integrates with e-mail, whereby these reports can be e-mailed to the user. Although comparatively simple to use, criticisms of Bugzilla have been made that it is hard to manage errors and there exist many bugs in the program itself.

B. **Jira**

JIRA [2] is a well-used tool for project management, made by Atlassian. Atlassian have a history of working with various aspects of project management, and JIRA integrates with many of these to produce a coherent management environment. However, this makes JIRA less useful on its own, so there is also an identifiable need for a standalone solution. Many users of JIRA complain of its difficulty to use, and say that it can be a burdensome way to manage their projects.

CONVERSATION FOCUSED SYSTEMS

A. Slack

Slack [3] is an instant messaging system focused around project management. It can be used in any team-based project, but it excels at managing software projects due to the large number of external services it integrates with, such as GitHub. Despite Slack not providing a mechanism for directly attaching project-related issues to conversations, it was decided that the excellent messaging functionality it provides is something the project should aim to emulate.

TASK-BASED SYSTEMS

A. Asana

The unique selling proposition offered by Asana [4] is effective task management without reliance on email. However, its web interface is difficult to understand, and thus results in a significant learning curve. Asana also lacks a real-time communication system, relying on comments on tasks for users to communicate. Whilst analyzing this product, it was found that relying on commenting rather than instant messaging resulted in a lack of urgency, and users were not as inclined to respond immediately. Consequently, users may choose to rely on an external messaging system for communication.

B. **Producteev**

Producteev [5] provides a feature set similar to the aforementioned products, but allows skilled users to quickly navigate the system using numerous shortcuts. A critical feature of Producteev is the ability to convert an email into a task using a Microsoft Outlook extension, thus allowing users to create tasks without have to repeat themselves.

C. Trello

Trello [6] is a web-based task management platform that employs the Japanese "Kanban" system, originally developed by Toyota. Kanban uses a hierarchical system of cards, contained within lists, which themselves are contained within boards, to manage projects. Kanban is often used to understand data flow within the real world, such as in supermarket production lines or in law cases, but also in project management using Trello. The team used Trello in the early stages of the project before eventually migrating to use the project itself.

MISC

A. Codebook

Codebook [7] was a project management system theorized by Microsoft to visualize the connections between people and their software projects. The project failed to become a public product due to its high degree of social networking and understanding of metadata.

Codebooks most significant lacking in reference to this project's goals was ticket management. It was not possible to manage a project entirely through Codebook, because it had no ticket management or communication between groups of people; instead, Codebook mapped out projects so workflows could be better understood.

INITIAL PLANNING

Having researched other products on the market, the team began planning the project itself.

TEAM ORGANISATION

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

In managing the development of the project the team determined a set of tools that would be used. A means for team members to communicate, store code, divide tasks and document the process was decided upon.

A. Communication

Trello [8] was chosen as the team's initial task management software, with the intent to swap to the tool being developed when the project had advanced sufficiently. Facebook chat was used to communicate and arrange meetings. This split of organization between Facebook and Trello frustrated some team members which further reinforced the aim of the project - bypassing the repetition of certain tasks over different tools.

B. Version Control

The team elected to use Git [9] as the version control system to manage changes in the source code of the software. Git was used due to multiple team members being familiar with it, and for the integration it has with Github [10]. Making use of version control was vital during development of the project, since features were often being developed simultaneously.

C. Team Roles

A set of roles were identified, and different team members were provisionally allocated these at the start of the project, as follows.

- Project Manager: Tom
- Secretary: Euan
- Tool Smith: Darren
- Test Manager: Leo
- Chief Architect: Gustavo

These roles were distributed informally and over time each member's duties changed as pertained to their strengths.

APPROACHING THE PROBLEM

The problem that the project aims to solve was established in the initial meetings with the project supervisor. However, developing a set of requirements that would implement a solution for this problem proved more difficult. One of the primary aims of this project is to realize a system that would reduce the duplication of effort that is often required when managing tickets: a team discusses an issue or bug on one platform and then logs the result in a ticket. This is a problem that existing solutions often make very little or no attempt to solve. While analyzing existing products and ideas, it was found that Producteev and Codebook attempt to minimize the effects of this issue, and the other systems choose to ignore it completely.

Codebook attempted to solve this problem by closely integrating a social network with software project artifacts. The social network aspect of Codebook was noted by the team as one that may encourage users to use the system both as their primary communication method, and as their project management tool. It was also identified that by giving the software access to tickets and the discussions related to them, a wealth of additional data is available which would otherwise have to be manually entered by a user.

The approach taken by Producteev on the other hand, is to attempt to reduce duplication of effort by allowing users to convert emails into tasks. By allowing users to create a task in this way, the need to copy information from email to the task manually is overcome.

Both of these features helped inspire the primary requirements of the software. It was deemed a vital requirement that the software tightly integrate social features in order to make use of the data generated through users communicating with each other. In addition, the team decided that the software needed to use this data in order to minimize the effort required in managing a software project. Agile software development aims to prioritize interactions over process, and so it would be prudent to design the software to maintain some degree of process without overly interfering with communication.

It was noted that the products that were looked at often had native apps and websites designed for desktop, but very few employed a responsive design to cater to their mobile users. It was decided that the design of the project must provide for the variety of devices the user may use to navigate the site, and that the website should accommodate for all possible cases.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Several different ideas for the development of the system were discussed. One suggested approach was that an existing product be extended to attempt to meet the requirements. Suggestions for implementing this approach included:

- *Interfacing a custom ticket management system with Skype*

This approach was rejected due to the lack of control that the Skype API gives with regards to accessing data contained within conversations.

- *Creating an instant messaging extension for Jira*

This approach was rejected due to the emphasis that Jira places on the ticket-based approach. Although the required design would have to maintain aspects of this approach, it would have been difficult to focus on communication when there is a major dependence on a colossal, process-reliant ticket management system.

- *Use the Facebook API as a basis for a chat system*

It was found that the Facebook API for integrating with the chat system is extremely limited and therefore would not have been suitable for achieving the requirements of this project.

With none of these approaches offering the desired flexibility, it was decided to build the system using Django, a Python web development framework. Django has a thriving developer community and its popularity means that it is easy to find tutorials and answers to questions should problems arise. It is also exceptionally powerful when it comes to interfacing with database management systems, providing an easy-to-use interface allowing Python code to treat database records as Python objects by defining mapping classes known as "models". As a result of this decision, and in addition to the extensive research of existing project management systems, an initial set of user stories were developed.

USER STORIES

The user stories initially created are listed below. Over time, they were added to, modified, and removed depending on various system development decisions.

1. As a *user*, I want to *create an account* on the system so that I can *use the features it offers*.
2. As a *user*, I want to *create a conversation with another user* so that I can *discuss a proposed change*.
3. As a *project manager*, I want to *change other users privileges* so that I *limit other users access*.
4. As a *developer*, I want to *tag a conversation as a ticket* so that *it becomes managed*.
5. As a *developer*, I want to *attach additional meta-data to a ticket* to allow me to keep track of priority, progress and other issues.
6. As a *developer*, I want to *end a conversation or ticket* so that I *don't have to look at irrelevant tickets*.
7. As a *developer*, I want to *be able to change priorities of my tickets* so that *more important tasks get priority*.
8. As a *QA manager*, I want to *be able to change the priorities of other peoples tickets* so that *I can decide the importance of certain tasks*.
9. As a *developer*, I want to *assign milestones to a conversation* so that *I can track project velocity*.
10. As a *developer*, I want to *split a conversation* so that *I can separate multiple issues if they appear in a single ticket*.
11. As a *user*, I want to *tag another user* to *add them to a discussion*.
12. As a *project manager*, I want to *view metrics that have been extracted from a ticket* so that I can *monitor my teams progress on an issue*.
13. As a *developer*, I want to *link to a version control repository* so that I can *quickly access the technical side of the ticket*.
14. As a *user*, I want to *attach multimedia to a conversation or ticket* so that I can *explain problems or ideas more easily*.
15. As a *developer*, I want to *assign resolution to an owner* so that *it will get done*.
16. As a *user*, I want to *add other people* to help *resolve the issue*.
17. As a *user*, I want to *be able to autocomplete names of other users* so that I can *tag other users correctly and quickly*.
18. As a *user*, I want to *create labels* so that I can *cross-reference other conversations*.
19. As a *developer*, I want to *view previous tickets* so that I can *view what has been changed previously*.
20. As a *project manager*, I want to *track the activity of other users* so that I *know what is happening in the project*.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

21. As a *user*, I want to *search the website* so that I *can find things within the site quickly*.

From these user stories a set of high, medium and low-priority tasks were determined. The high-priority user stories were: **1, 2, 5, 7, 12, 19**. Medium-priority: **11, 15, 18**. Low-priority: **10, 13, 14, 17**.

DESIGN

Once the initial requirements were established, an entity-relationship (ER) diagram was developed. After several iterations, the diagram shown in Figure 1 was produced. This was used to assist the development by identifying the crucial components of the system. Additionally, the ER diagram greatly aided in modelling the data using the Django Object Relational Mapper since there is a close correlation between the features of the diagram and the structure of data models in Django.

Due to the project's experimental nature, rapidly changing requirements caused the final model to vary from the design in the diagram.Next, a series of paper-based prototypes were created to plan the user interface. Paper prototyping was used due to quick, easy and inexpensive nature whereby multiple different possible designs can be quickly produced and contrasted. Several initial designs were brainstormed. The clutter, intuitiveness and overall appeal of different designs were compared. Examples of different designs that were considered are shown below.

Figure 2a shows the initial prototype for the design, which displayed visualizations relating to the project on the main page. A project was chosen from a drop-down menu and all the tickets related to that project were listed below. Each ticket consisted of the graphs, metadata and the associated conversation.

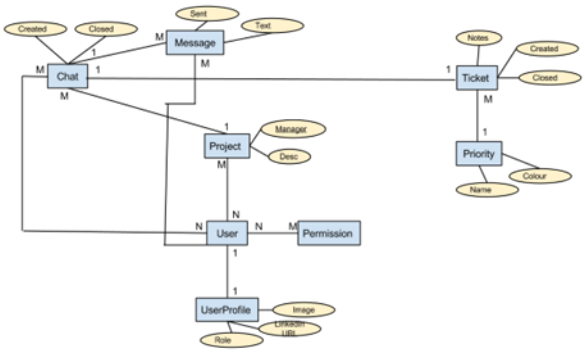
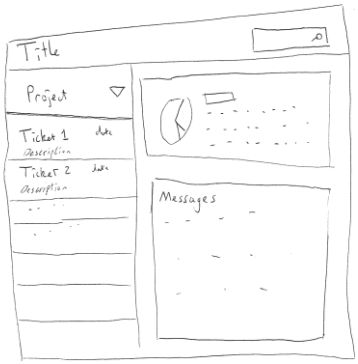
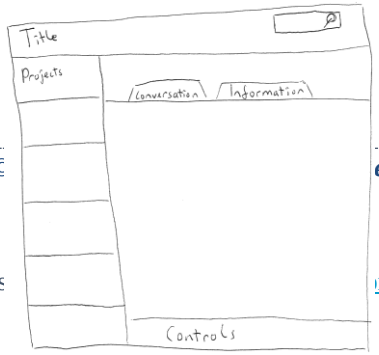


Fig. 1: ER Diagram

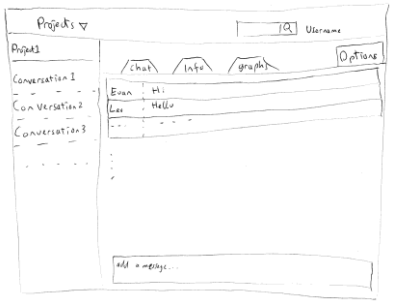


(a) Mockup 1



ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

(b) Mockup 2



(c) Mockup 3

Fig. 2: Three mockups

In Figure 2b, the ticket appearance was changed whereby the information section was removed and two tabs were added, a 'conversation' tab which was used to display the messages, and an 'information' tab which displayed visualizations and metadata related to the project. This change was made to reduce clutter in the interface and A controls section was added which would be used to add new metadata such as due date, priority, assignee etc.

The final design, Figure 2c, moved the project's drop-down menu into the navbar, with the selected project shown below alongside the relevant conversations. The controls section was removed, and an options menu was added above the conversations. The information tab was divided into an info tab and a graphs tab, whereby info contained metadata relating to the conversation and graphs contained the visualizations that had been generated with the data.

IMPLEMENTATION

Once the requirements had been identified and the team's organizational structure formalized, implementation could begin.

The application's server-side is written in Python, using the Django web application framework and several additional Python modules. A second server exists purely for handling real-time communication. The client side is written using the jQuery Javascript library [11], with several other Javascript libraries being used to handle different components such as chat and graphs.

WORKFLOW

A. Git

Several branches were maintained within the Git repository. Two of these branches were particularly important, as they represented the project's primary and secondary deployments. The master branch contained the most stable build, and the beta branch was used as a deployment staging area for new features. Some code modifications may work as intended in a development environment and break when deployed. To overcome this risk, the code was first deployed to the beta branch before merging into the master branch.

One aspect of version control in which the team was eventually diligent in applying was continuous integration, in which feature branches are regularly merged into the master branch. Using continuous integration meant that the project's live version never strayed too far from the overall set of features being developed at a given time. Frequent merging of feature branches also provided the added benefit that breaking changes were easier to find,

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

because less code was being added in each merge. During the initial stages of the project, branches often went unmerged for several weeks, making integration of features exceptionally difficult. After realizing the difficulties this workflow caused, this flaw was amended by substantially increasing the rate at which features were merged with the master branch. Diagram 3 shows how features were frequently merged into the master branch (shown as the black line at the top of the diagram) during the week of March 1st, 2015.
For comparison, diagram 4 displays the lack of feature branch integration exhibited during the project's early stages, showing branching activity between November 24th and 30th, 2014.

B. Continuous Delivery

A service called Codeship was used to ease the deployment of new builds. Codeship [12] is a continuous deployment platform configured so that any changes to the master or beta branches of the Github repository were automatically built and deployed to their respective live websites.

DEPLOYMENT

An unforeseen benefit of using the combination of Git and Github was the ease by which the system could be deployed. Two instances of the project [13] were hosted on Heroku [14], a platform for hosting dynamically scalable web applications. The first instance hosted the code held within the master branch of the Git repository. The second instance ran the code present in the "beta" branch of the project. Heroku provides a mechanism which enables deployment using Git, so the deployed version of the product is always equivalent to the most recent commit on the master branch.

Since the default web server provided by Django is not suitable for production use, the Gunicorn [15] server was used instead. Heroku made it easy to use Gunicorn through the use of a "Procfile", which specifies the Bash commands that should be executed whenever a write is made to the master branch of the repository. Thus, using Gunicorn was as simple as adding the initialization command to this file.

A Python library known as WhiteNoise [16] was used to index static assets, such as images, Javascript, and CSS files. Since these assets are indexed into a dictionary, the server can perform an O(1) lookup to retrieve a file requested by the client instead of an O(n) file system traversal, resulting in a performance increase.

MODEL IMPLEMENTATION

The data model was implemented as shown in Figure 5 using Django's built-in Object Relational Mapper (ORM). During the initial stages of development, these models were mapped to a SQLite [17] schema, but as development progressed, the PostgreSQL database management system was used instead. PostgreSQL [18] was used due to the ease with which it can be connected to a Heroku-hosted application, whilst SQLite is only suitable for desktop or mobile applications. Another factor contributing to this decision is that SQLite does not scale well on web applications because concurrent access rates can change unpredictably and are often much higher than client-side applications.

USER INTERFACE

Due to the system being a web application, the main technologies used in creating the user interface were HTML5 and CSS3, which are standard in the vast majority of web-based software.

Twitter's Bootstrap library was used to manage the layout of web pages, and to improve compatibility with mobile devices. The intuitive grid layout system provided by this library greatly improved the rate at which the UI was developed, since it minimized the extent to which the team had to rely on custom CSS rules.

Bootstrap satisfied the requirement of responsive design due to its feature of being mobile-first when necessary. This meant that the design of the web pages could be laid out in such a way as to accommodate small screens and expand to larger sizes of screens when necessary.

Certain decisions were made to enhance the user friendliness and quality of user experience during the design of the user interface. When appropriate, the team followed design patterns in line with the German Bauhaus movement of design. Bauhaus design revolves around the idea that the function of a project should directly influence the form of a design. In this case, the function of the application is to allow ticket management with a focus on communication and conversation; therefore, design decisions were made that would ingrain this

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

functionality into the user interface. A tabbed design allowed for immediate access to conversations while allowing access to metadata and metrics. Furthermore, tabbed designs are very common, so this was also easy to use. Another example of the design's form following its function is that the conversations are almost always available via Bootstrap accordions in a sidebar. This allows for easy and constant access to conversations, as a user's focus, but does not clutter the user's view at any time due to the accordion's collapsibility and Bootstrap's minimalist look.

It was identified that the application must be reactive to user interaction, and to accommodate this, the jQuery Javascript library was used. jQuery assists almost every element of the application, from displaying new messages on the screen, to highlighting selected issues.

Perhaps the most important feature of the jQuery library used to make the website more responsive was Ajax (Asynchronous Javascript and XML). Ajax allows reloading only small portions of the page when required, rather than reloading the entire page and sending multiple requests for objects that have already been transferred. Ajax is the basis for several application features that would otherwise respond slowly, such as the search and sort features. It was deemed that a constantly reloading page would be frustrating to a user, particularly a mobile user with potentially slow internet connections. Ajax was an appropriate way of circumventing this problem in a way that would be aesthetically pleasing to the user.

The project's resulting design, shown in Figure 6, was a set of pages that were responsive to user input and intuitive to use. These pages focused on the application's functionality. In addition, the pages were mobile-ready, meaning that the intuitive design could be shared across any platform.

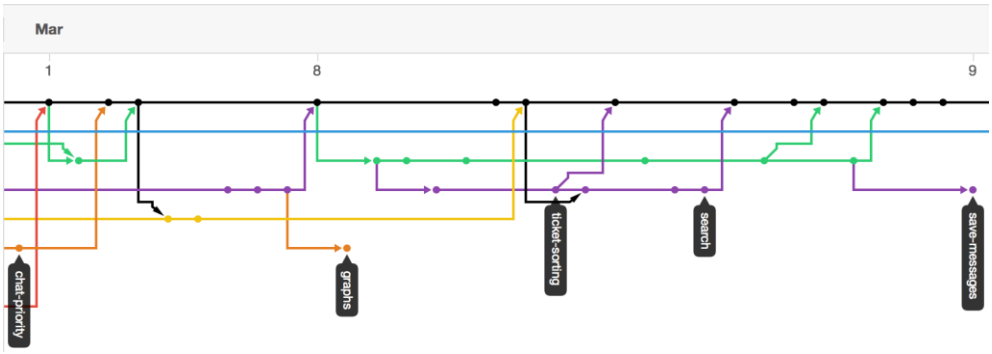


Fig. 3: Branching Diagram 1

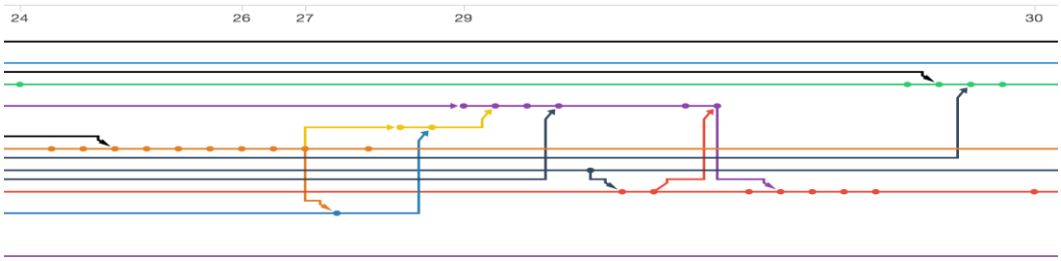


Fig. 4: Branching Diagram 2

MESSAGING SYSTEM

The instant messaging aspect of the project was implemented using a Javascript library and backend data storage service called Firebase [19]. An alternative option considered by the team was to implement a server-side mechanism for handling messaging using the WebSocket [20] protocol. However, several disadvantages to this approach were found on investigation:

- The default Django web server relies on the Web Server Gateway Interface (WSGI) [21] for handling communication between the server and the framework. Due to the request handling mechanism employed

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

by WSGI, it is not possible to support long-term connections through the likes of WebSockets, and so an additional server which provides support for the WebSocket protocol would have to be set up to achieve the required functionality.
- The WebSocket protocol is extremely complex in comparison to the Firebase library, and so it would have been difficult to produce a working system within the time constraints of the project.

Another option considered was working within the constraints of the Django framework. Creating an instant messaging system within Django would require using an inefficient method such as asynchronous polling, in which the client repeatedly asks the server if a new message has been sent.
The decision to use Firebase was not without its drawbacks:

- There is an overhead in ensuring that the data in the external storage provided by Firebase remains consistent with the data held within the relational database.
- Employing a third-party storage service added a point of failure. On several occasions throughout development, the Firebase API became unavailable, leaving the application unusable for a short period of time
- Each member of the team had to understand the API exported by Firebase. Consequently, the project uses two REST APIs for communication with the server side rather than one, increasing the complexity of the design. The Firebase API also proved difficult to work at times, with simple queries departing from the widely understood semantics of SQL.
- The free access to Firebase seemed to randomly throttle connection to its servers, occasionally resulting in slow resolution of API calls.

In general, the use of an external service to implement the messaging system greatly improved the rate of development of the project and allowed the team to focus on other aspects of the system, which may not have been possible using one of the other options.

Due to the fact that some messages may try to convey much information, it was felt that the messaging system should allow some level of formatting, enabling users to write formatted lists, headings, emphasized text, and more. This was achieved using a Javascript library known as Showdown.js [22], which enables users to write messages using the Markdown [23] formatting language. Before messages are displayed on screen, they are passed through the Markdown parser provided by Showdown.js and converted to valid HTML. Figure 7 shows how Markdown formatted text can improve the readability of what would otherwise be a comma-separated list.

REST API

A REST API provides an interface between the client and the server. A Django plugin known as TastyPie [24] was used to convert the models that were previously defined into REST endpoints that the client can call to request data from the database. After a quick initial configuration of the plugin, specified portions of the database schema were available to the client side of the application. For example, if the browser requests the URL /api/v1/chat/, the server will respond with a Javascript Object Notation (JSON) object containing information about all conversations in the database. A truncated typical response from accessing this endpoint is shown below.

This JSON object can then be parsed using Javascript and injected into the Document Object Model (the tree representing the layout of the document) using Ajax, therefore making it visible to the user without requiring a page refresh.

The system relies heavily on client-side data processing, so having a flexible and extensive REST API, which enables client-server communication, was an essential component of the application.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

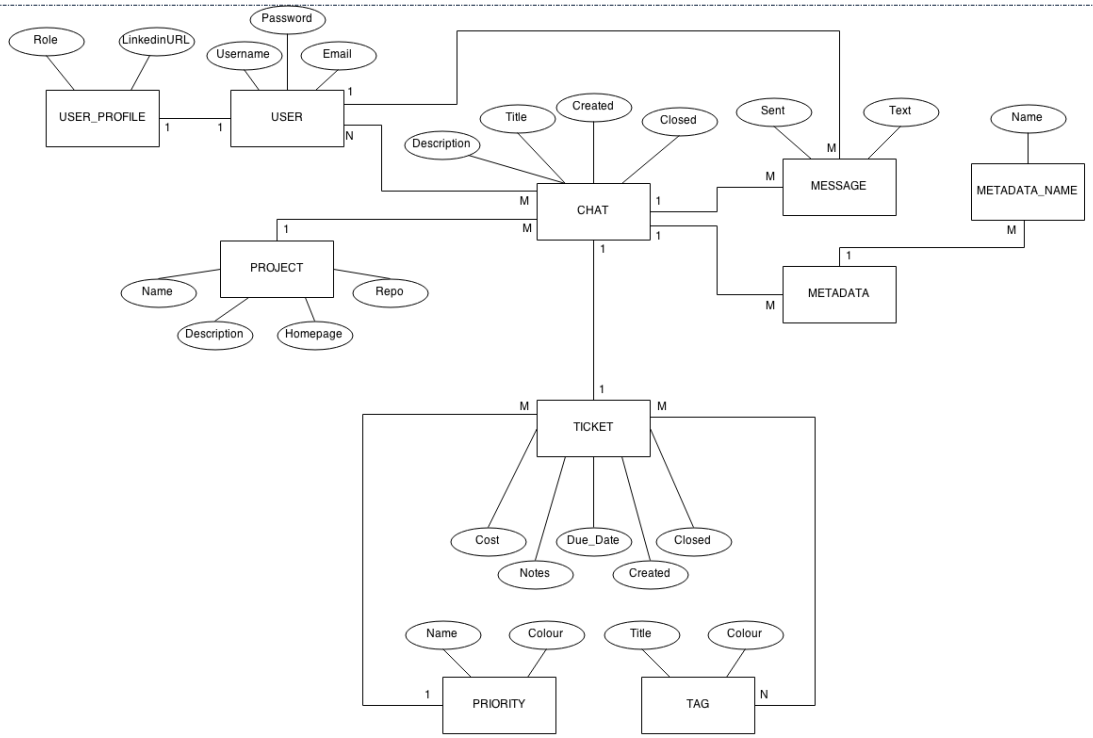


Fig. 5: Final ER Diagram

PROTOTYPE

A prototype was prepared for demonstration after an initial ten weeks of development. The interface shown in Figure 8 is simple but fully functional. The system had basic functionalities such as:

- User registration and login
- Project creation
- Creation of independent conversations
- Closing conversations

The aim of developing the prototype was to create a basic proof-of-concept display that could then be built upon instead of implementing multiple user stories. This resulted in a fragmented program with many isolated features but no overall functionality. The final demonstration, therefore, lacked the polish and features to be delivered by the end of the project. The tradeoff for this was that features were much easier to implement in the future, and solid groundwork had been laid upon which the team could now build.

METADATA

The ability to attach metadata to conversations is a core feature that differentiates the software from standard instant messaging applications. It is also the feature that makes the application suitable for use as a project management tool. Metadata provides essential information for developers and managers regarding a conversation and is accessible through the information tab. Metadata is a crucial feature of the final product, so a prominent position was given in the UI. A drop-down menu to the right of the screen, shown in Figure 9, allowed for the easy addition of six metadata types through a menu shown in Figure 10.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

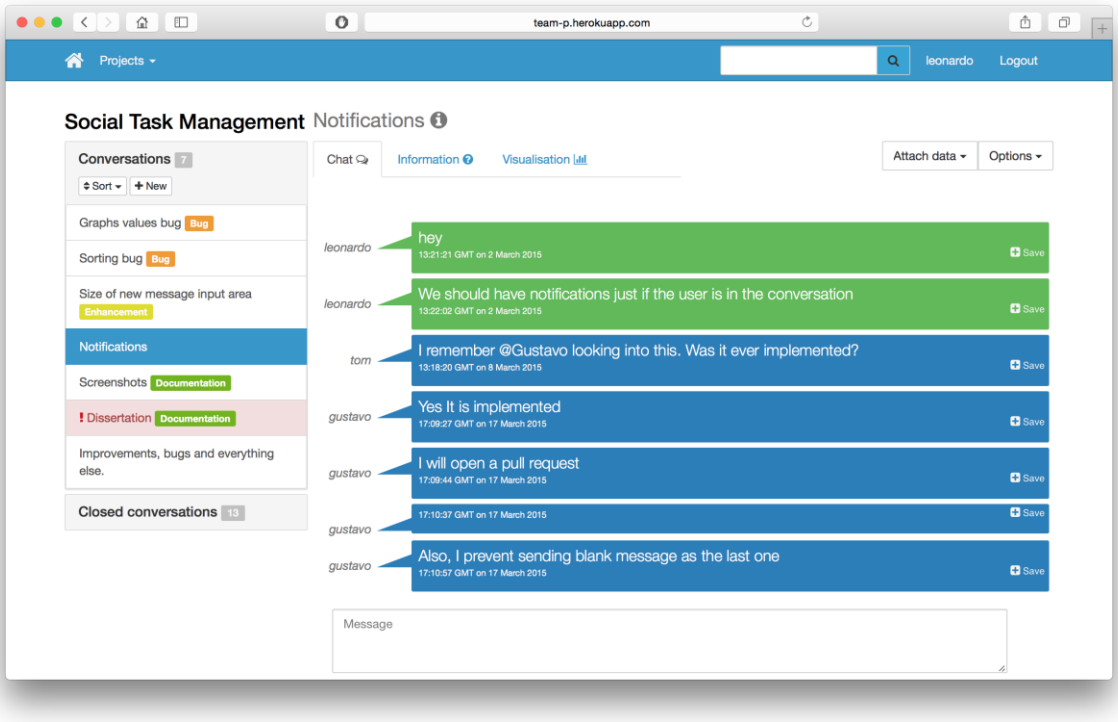


Fig. 6: Final Design

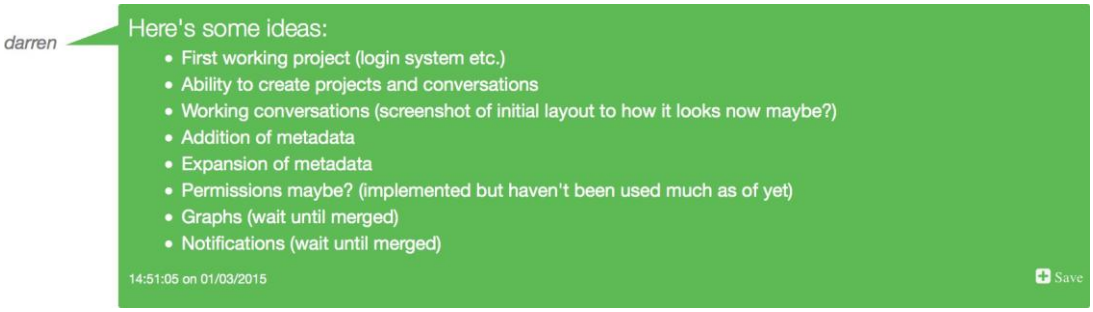


Fig. 7: Markdown Example - Unordered List

The metadata types are defined below and shown in Figure 11.

- *Assignee*: By default, the user who creates the conversation is its assignee. However, users can edit this later and choose another user to be assigned from a drop-down list.
- *Cost*: A decimal field that can be used flexibly by development teams to indicate the effort required to complete a task. The field is unitless and therefore does not enforce a specific cost prediction method. Thus, teams can define their own baseline from which to judge the cost of a particular task.
- *Due Date*: The date and time the issue discussed in the conversation should be resolved.
- *Notes*: A text field primarily used for linking to resources related to the conversation. This field supports Markdown notation, meaning that teams can use it to store almost any type of well-formatted miscellaneous information.
- *Priority:* This field can indicate the importance of a conversation. It can contain any one of the following values: “Low,” “Medium,” or “High.” Conversations marked as high priority are displayed with a red background in the conversations list.
- *Tags:* Users can set tags for a conversation. Every tag has a special label color and is displayed beside the conversation title to help users identify the nature of the conversation.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

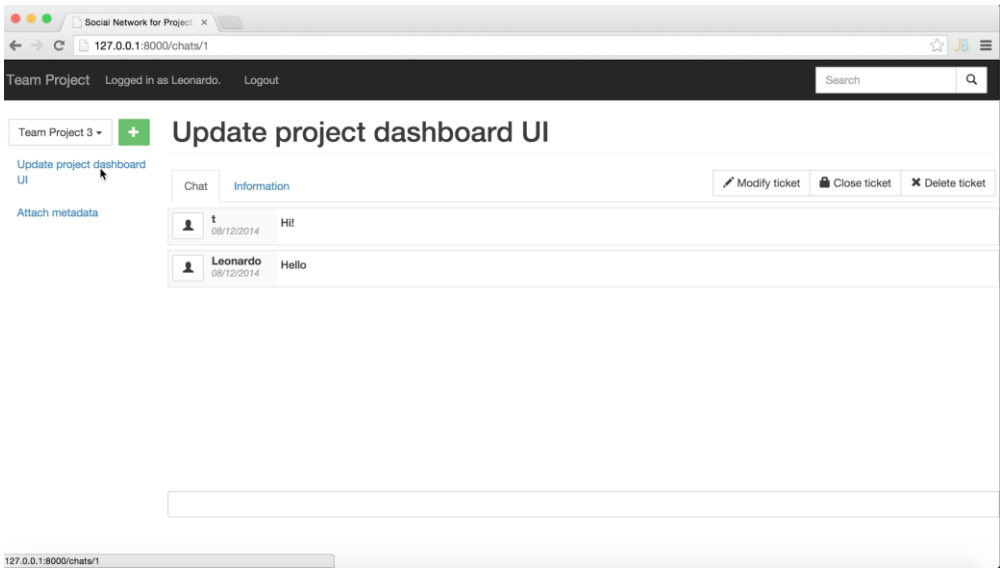


Fig. 8: Prototype

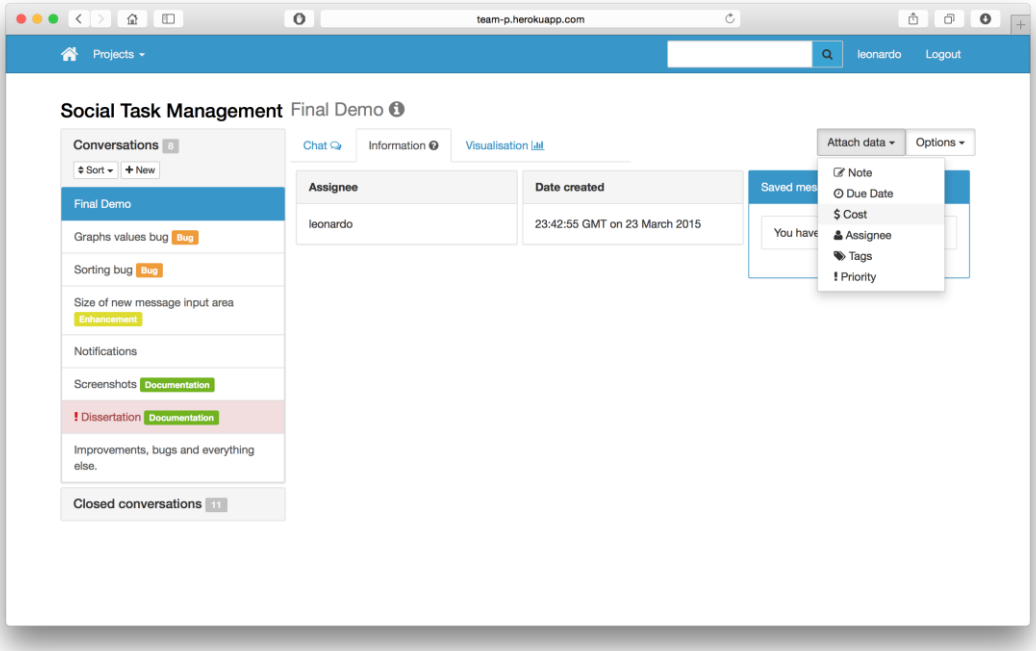


Fig. 9: Add metadata dropdown menu

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

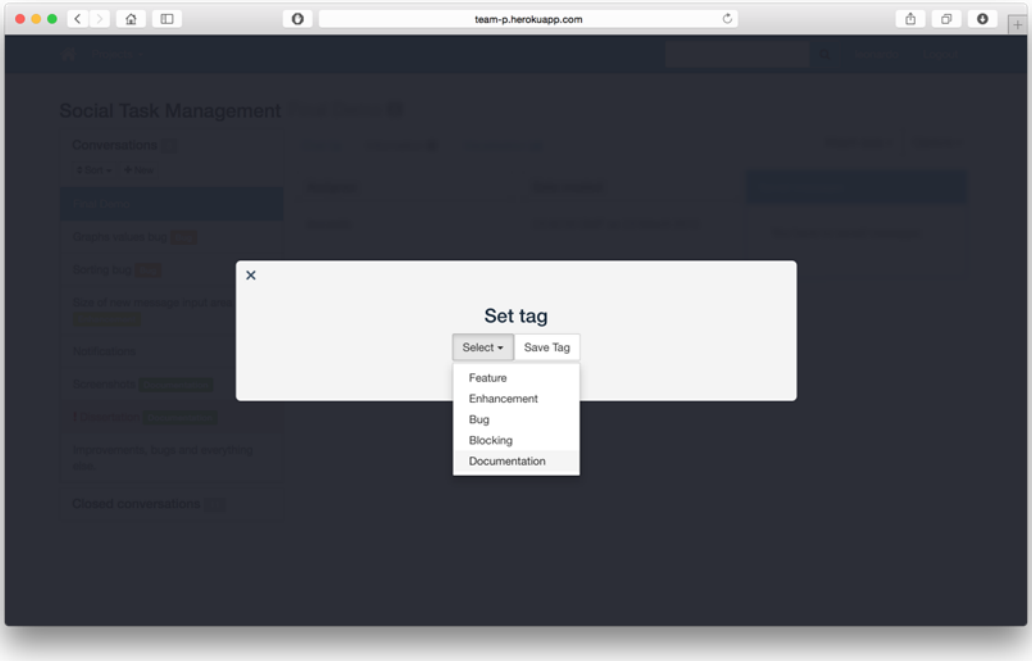


Fig. 10: Add metadata

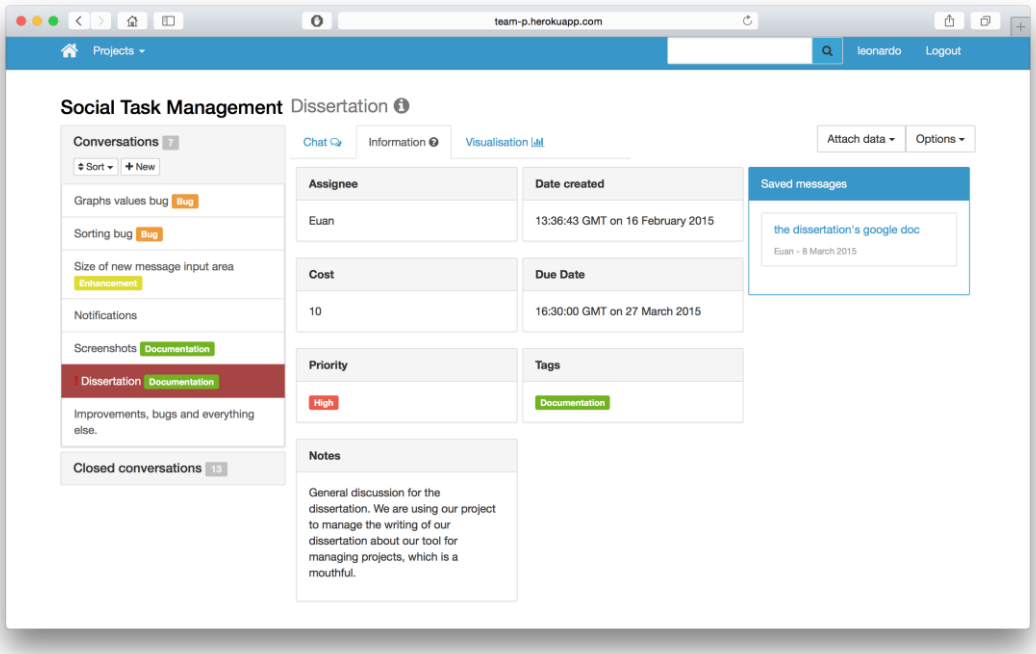


Fig. 11: All metadata

STATISTICS

The visualization of statistics regarding a team's activity was deemed an important feature of the product. Providing metrics to users allows for the identification of successes and failures within a project. Although the statistics provided cannot give an absolute indication of the exact state of the project, they may give hints to project managers of issues regarding progress or levels of developer contribution. For example, a conversation that has

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

been open for a week without any activity may indicate that a problem is being ignored, and the graphs intend to aid in identifying situations like this.

Three graphs were implemented to show a proof-of-concept of how visually representing information would enhance a user's understanding of their project. All visualizations were implemented as simple bar charts. It was found that, in all three cases, the data being displayed was a mapping of ordinal data and quantitative value. Users' familiarity with bar charts made them an obvious choice for representing the site's data. The graphs provided were:

- Messages sent per user in a conversation
- Messages sent per user in a project
- Messages contained in each conversation in a project

These graphs were intended to show user participation and chat activity in a way that would allow an observer to judge the importance of different chats within the system and to see the contributions a user was making on both a project level and within specific conversations.

The graphs, shown in Figure 12 and Figure 13, were constructed using the D3 [25] Javascript library. D3 allowed for the creation of graphs with relative ease, powered by the conversation data from Firebase. In addition, D3 allowed for the creation of tooltips that contained data regarding points on the graphs [26]. It was identified that containing exact data on each bar of the graphs could make them cluttered, making data more difficult to understand at a glance [27, 28]. Therefore, the graph was designed to provide a simple visualization of data, and tooltips would explain the data provided when points on the graph are hovered over [29,30].

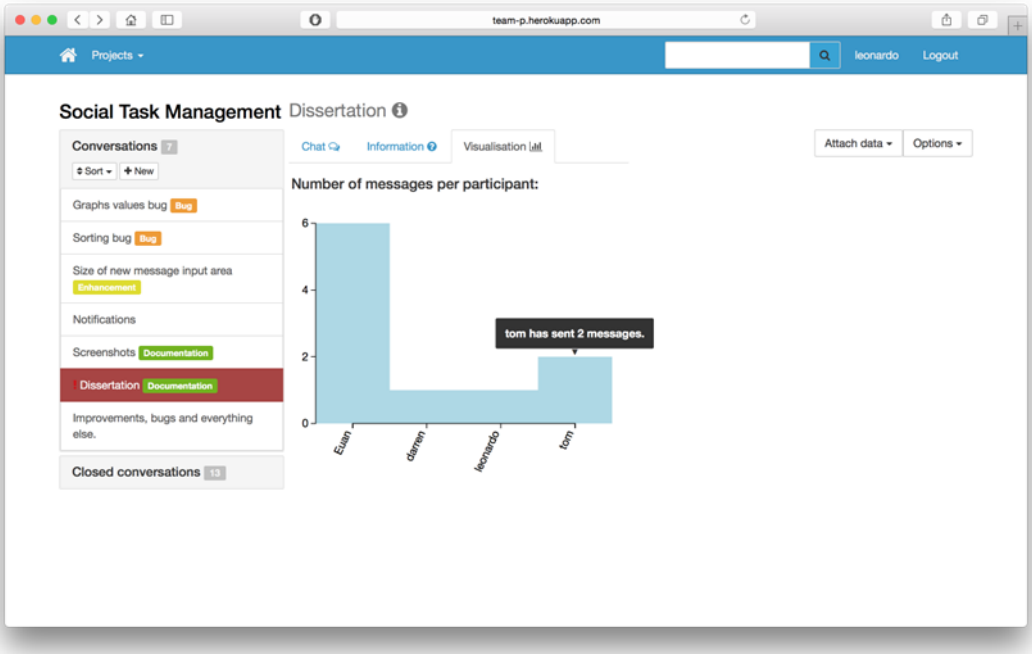


Fig. 12: Chat graph

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

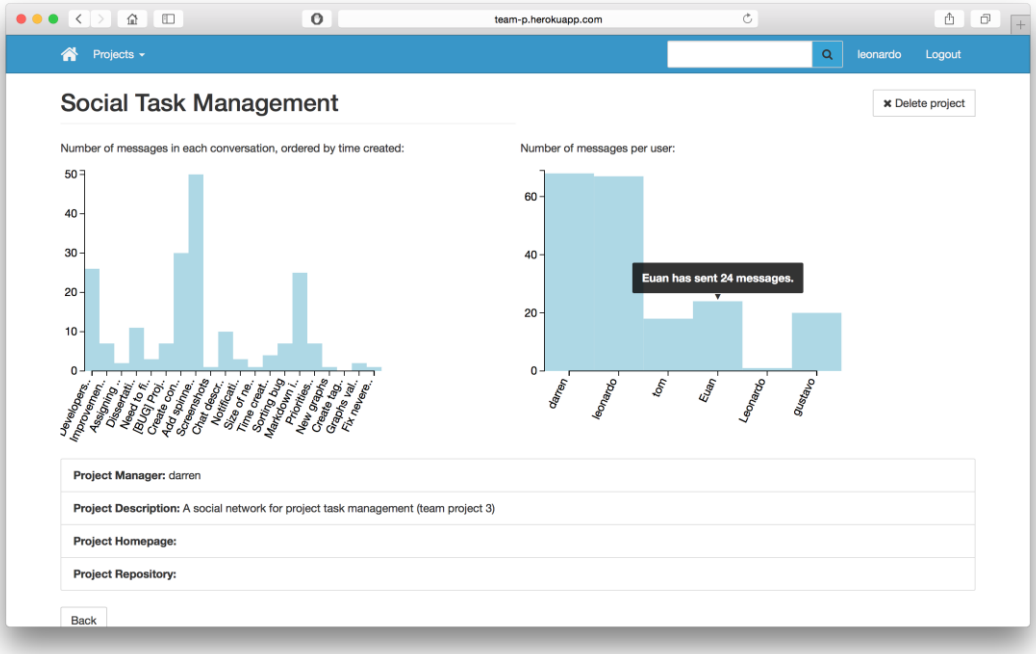


Fig. 13: Project graphs

ADDITIONAL FEATURES

With the software aimed at software development projects, the team was in an excellent position to evaluate it themselves by using it to manage the development. After this self-evaluation, the following features were identified and then implemented [31-33].

A. **Notifications**

Users can participate in a significant number of different chats at the same time. Without a notification system, the user does not know when a chat has updates or if the user was mentioned in a conversation. The user could check each conversation manually, but it is impractical and time-consuming. To address this problem, the web notification feature was implemented [34, 35].

The Notification API specification [36] is currently under development. However, the technology is supported by most of the modern browsers, as shown in Table I. Notifications consist of four main attributes: title, body, icon, and tag (i.e. an identifier for the notification).

TABLE I: Support for web notifications API

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic Support	5 webkit 22	4.0 moz 22	Not supported	25	6

There is a script in the project that listens for updates in chats. When a change occurs, a new notification is created showing the id of the updated chat and the content of the latest message, and it is displayed to the user [37, 38]. While the notification is displayed, the user has three options: dismiss the notification, wait for the automatic closing time of ten seconds, or click the notification and be redirected to the updated chat. A listener awaits changes in any conversation. When the listener detects that a new message has been sent, a new notification is created showing the ID of the chat and the content of the new message [39, 40]. This notification is then sent to all participants, as shown in Figure 14. With the notification on screen, the user has three options: dismiss the notification, wait for the automatic closing time of ten seconds, or click the notification and be redirected to the conversation [41,42].

Currently, the notification system does not save updates from the user's offline periods. However, it addresses the issue of users having to manually check for updates by delivering useful, real-time information.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

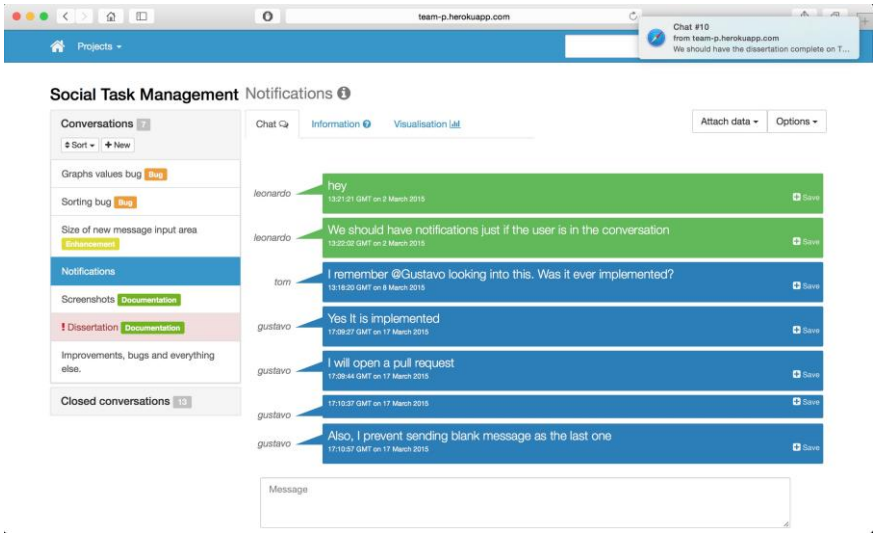


Fig. 14: Notification example

B. **Saved Messages**

The number of messages in each chat can be considerable, so it can be quite difficult to locate important messages. The user could just add a note to avoid this problem; however, this would involve copying the information from the message into the conversation notes. Doing this each time important information was sent would be a slow and repetitive process [43, 44].

To solve this problem, the ability to save messages that are of value as metadata was added. This feature is implemented through a small button Figure 15a located in the bottom right corner of each message. If pressed, the message will be saved as an item of metadata in the information tab as shown on Figure 15b.

With this feature, the user can see a list of all the relevant information in a chat. Thus, there is no need to read the entire conversation to understand what is happening. Thereby, there is no duplication of effort, and every relevant content can be saved as the conversation occurs [45, 46].

C. **Improved Navigation**

In extremely active projects, many conversations can exist at any given time. Without any navigational assistance, finding a conversation could be cumbersome, even if the name of the conversation was known.

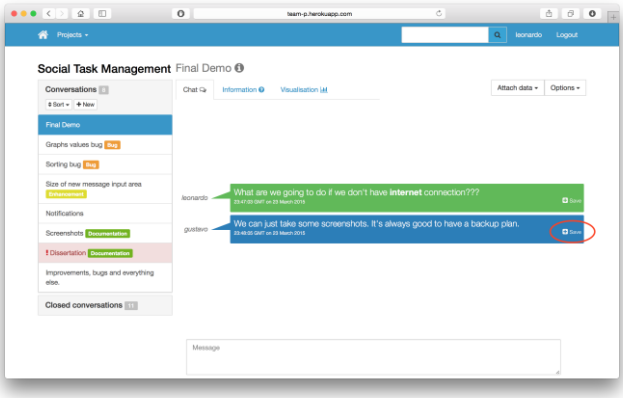
Filtering and sorting functionality was added to assist the user in finding what they need. The user can sort conversations based on selected metadata, such as due date and priority. As a result, conversations of particular interest can be identified with ease [47, 48].

Additionally, the user can filter the conversations based on their names. As the user types a string into the search input box, only conversations whose names contain this string as a substring will remain on screen, and the rest will be immediately hidden from view [49].

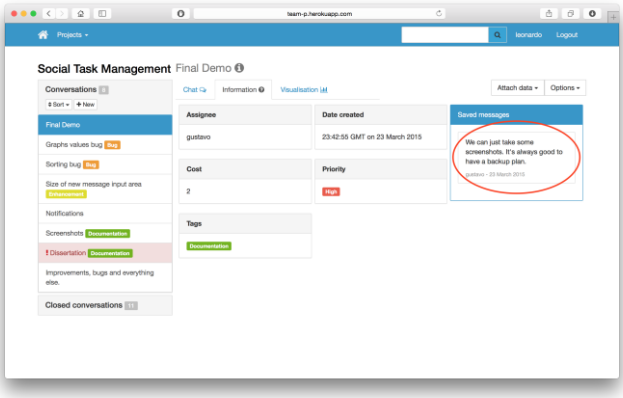
EVALUATION

The project was formulated with the intent to create a ticket management system that emphasized communication between developers over coordination of data on a ticket. The resultant product does this effectively by providing a means of associating additional information with a conversation. The name of a conversation equates to a ticket name, and the data that would be stored in a ticket using a conventional ticketing system is contained within the information tab of the chat. This means that despite appearing at a glance like a primary chat system, with metadata almost an afterthought, it provides the full functionality of a ticketing system. This allows the system to be used as a self-contained system for project management without the need for external systems to ‘fill in the gaps’.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7



(a) Saving a message



(b) Message saved

Fig. 15: Saved Messages feature

EXTERNAL EVALUATION

The team enlisted the help of another project team to evaluate the application. This revealed several areas where the system performed well and areas of potential improvement.

The testers liked the overall layout and believed the graphs could reveal useful information about a project.

However, several possible changes that were identified included:

- Metadata added was anonymous and thus open to abuse - it does not say who saves messages, who added notes, etc. Who added what should be shown in the information tab.
- System is open to attacks due to major security issues. API is not secure in that users can view and modify database objects through it.
- An inclusion of a list of recent activities could greatly improve the usability of the site, allowing users to see what has been recently done on each project.
- Allowing the user to customize the appearance, such as choosing a color scheme or changing the font, allows them to personalize it as they prefer and adds a sense of identity for different users.

DOGFOODING

As a result of the team using the application to manage their own project, they were able to evaluate the effectiveness of the tool. As discussed in the previous section, the identification of key weaknesses allowed the team to fix these and improve the user experience as a whole. As the team is part of the target market, they were in the optimal position to evaluate the usefulness of the system as a whole, and with the inclusion of the additional features, it was found that the tool functions as a wholly inclusive management system, while remaining easy to use.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Some key additions were made to the functionality of the application as a result of the team testing it as users. For example, the need for the functionality of a saved message was quickly identified, and its inclusion proved very useful. Also, notifications were added after this part of the project's evaluation. It became evident that the messaging system suffered from a lack of updating users on activity, meaning that issues could go untreated as no developers were aware of new information.

FUTURE GOALS

A. Security and Stability

As noted in the external evaluation section, the API remains public and so if someone can access the server that the application is deployed on, they will gain limited access to the REST API for that deployment instance. Therefore, checking whether the request is from an authenticated source is a high-priority goal for the future. Additionally, using the "security rules" feature provided by Firebase, conversations can be made more secure by ensuring only users participate in a conversation and managers can see the messages within. In order to improve the robustness of the application, the user interface could be tested using a tool such as Selenium WebDriver. Selenium [50] automates web browsers by automatically performing user-defined actions to ensure that web applications work as expected.

B. Update Feed

The application's landing page is currently lacking content and could be updated to provide a "birds-eye" view of all things happening within the user's projects. Displaying a feed summarizing the latest messages, conversations, and high-priority issues would vastly improve the software's capabilities as a project management suite. Managers would no longer have to check individual conversations or rely solely on the metrics provided, but could also receive up-to-the-minute updates on the latest happenings within their projects. A feature like this would be beneficial in large-scale software projects, where manually checking each conversation for progress updates would be infeasible.

Additionally, mobile users currently do not receive notifications, but users are frequently away from their keyboards and may need to be aware of an issue immediately. Native mobile applications would provide a richer mobile experience and would also be capable of delivering notifications straight to a user's smartphone or tablet.

C. Integrating with Third Parties

Feedback from the prototype demonstration suggested that integration with a service such as Jenkins may be a valuable addition to the application. Jenkins [51] is a continuous integration server used to monitor and execute repeated jobs, such as building and testing software being developed. This could be implemented by treating Jenkins as a user who automatically creates a ticket based on each build. It could also automatically create a high-priority ticket when a build fails and tag the relevant developers. Jenkins provides a REST API, which would make this integration relatively simple.

Another potential addition could be integrating with GitHub or version control systems like Subversion. Version control systems often allow for issues to be noted within commits, and discussing these issues as tickets within this application would be useful. For example, a conversation could be created automatically when a merge request (a request to merge one branch into another) is made on GitHub. This could be used to discuss the changes and suggest possible modifications.

D. Scalability

Currently, the project's setup may not easily scale to large software development teams of hundreds or thousands of developers. One solution to this problem would be to filter conversations only to those a user has interacted with. To add another developer to a conversation, a user may "tag" the other developer using a notation such as "@username." This would allow for a user to see the conversations that concern them, easily, but would come at the cost of the user's awareness of the project at large, as to list all conversations within an excessively large project would take an enormous amount of screen space.

A second solution to this problem would be to allow for sub-projects. This would permit work to be broken into smaller chunks (an instance of the application managing a game could include a game engine project with a game physics sub-project). This would require some alterations to the user interface, but would allow for effective management of projects in a way that is currently difficult with many popular project management tools such as Trac. Additionally, this would work well with the previously suggested feed of recent content, as users could see

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

recent content in the context of its more specific sub-project title rather than the context of an all-encompassing project title.

4. CONCLUSION

As shown by the team's self-evaluation of the application, the project is currently stable and useful enough to monitor and track issues within a software development team.

The final project serves well as an example of how project management can benefit from a focus on communication in a ticket management scenario. In integrating communication and ticket management, it has been found through the team's usage of the application that communication is a vital component of a ticket's lifecycle. Furthermore, the application shows that ticket management can benefit from an increased focus on communication over what the currently used products provide. The project has achieved its aim of providing a ticket management system that treats an ordinary conversation as a ticket and is a viable enough solution to be practical for at least small software development teams.

While some problems persist in the project, as exposed in the external evaluation, its utility was evident to those who viewed it. This shows a definite lack of integration of ticket management and instant messaging within project management tools. It also indicates that current tools unnecessarily complicate communication between developers.

REFERENCES

1. Bugzilla.org, "Home - bugzilla - bugzilla.org," 2015, http://www. bugzilla.org/.
2. Atlassian, "Jira - issue and project tracking software - atlassian," 2015, https://www.atlassian.com/software/jira/.
3. Slack, "Slack: Be less busy," 2015, https://slack.com/.
4. Asana, "Asana - teamwork without email," 2015, https://asana.com/.
5. Producteev.com, "Task management software - producteev by jive," 2015, https://www.producteev.com/.
6. F. Creek, "Trello," [Online; accessed 25-March-2015 from http://trello. com/].
7. Research.microsoft.com, "Codebook - microsoft research," 2015, http: //research.microsoft.com/en-us/projects/codebook/.
8. T. P, "Team p trello page," 2015, [Online tool; used 25-March-2015 from http://trello.com/b/TwY1Htmk/team-project-3/].
9. Github, "Github," 2015, http://github.com.
10. T. P, "Project github repository," 2015, http://github.com/darrenburns/ TeamProject3.
11. jquery.org, "jquery," 2015, https://jquery.com/.
12. Codeship, "Continuous delivery for free - codeship," 2015, https:// codeship.com/.
13. T. P, "A social network for project management," 2015, http://team-p. herokuapp.com/.
14. Heroku.com, "Heroku - cloud application platform," 2015, https://www. heroku.com/.
15. Gunicorn.org, "Gunicorn - python wsgi http server for unix," 2015, http: //gunicorn.org/.
16. Pypi.python.org, "Whitenoise: Python package index," 2015, https:// pypi.python.org/pypi/whitenoise.
17. Sqlite.org, "Sqlite home page," 2015, https://sqlite.org/.
18. Postgresql.org, "Postgresql: The world's most advanced open source database," 2015, http://www.postgresql.org/.
19. Firebase.com, "Firebase - build realtime apps," 2015, https://www. firebase.com/.
20. Tools.ietf.org, "Rfc 6455 - the websocket protocol," 2015, https://tools. ietf.org/html/rfc6455.
21. Python.org, "Welcome to python.org," 2015, https://www.python.org/ dev/peps/pep-0333/.
22. Showdown, "Showdownjs/showdown," 2015, https://github.com/ showdownjs/showdown.
23. J. Gruber, "Daring fireball: Markdown," 2015, http://daringfireball.net/ projects/markdown/.
24. Tastypieapi.org, "Tastypie - restful apis for django," 2015, http:// tastypieapi.org/.
25. M. Bostock, "D3.js - data-driven documents," 2015, http://d3js.org/.
26. Hood, K. and M. Al-Oun, Changing performance traditions and Bedouin identity in the North Badiya, Jordan. Nomadic Peoples, 2014. 18(2): p. 78-99.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

27. Ibadin, F., et al., Comparative Analysis of Granulation Tissue Formation and Progression in Elderly Patients with Fractures. American Journal of Medical Science and Innovation, 2025. 4(1): p. 11-17.
28. Ibadin, F.E. and E.O. Ernest-Okonofua, SYSTEMATIC LITERATURE REVIEW OF COGNITIVE-BEHAVIORAL THERAPY (CBT) EFFECTIVENESS IN TREATING DEPRESSION AMONG ADULT MENTAL HEALTH PATIENTS. The American Journal of Medical Sciences and Pharmaceutical Research, 2024. 6(12): p. 113-126.
29. Jangid, J., Efficient Training Data Caching for Deep Learning in Edge Computing Networks. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2020. 7(5): p. 337-362.
30. Jangid, J. and S. Dixit, Enhancing security and efficiency in wireless mobile networks through blockchain. Available at SSRN 5167791, 2023.
31. Karani, V., P. Madisetti, and J. Tyagi, Method and system for intelligent priming of an application with relevant priming data. 2019, Google Patents.
32. Mahroq, O.A., et al., Neuro Navigation Versus Conventional Spinal Techniques in Analyzing Nerve Injury and Anatomical Accuracy: A Systematic Review. Cureus, 2024. 16(9).
33. Shakirat Oyindolapo Ganiyu, Rithish Nimmagadda, Vemparala Priyatha, BushraFirdous Shaik, Excel Ernest-Okonofua, Omar Mahroq, Dissecting the Obesity Paradox in Relation to Morbid Obesity and Heart Failure Outcomes. ARC Journal of Cardiology, 2024. 9(1): p. 5-13.
34. Vigneshwaran, E., et al., Prevalence and predictors of cervical cancer screening among HIV-positive women in rural western Uganda: insights from the health-belief model. BMC cancer, 2023. 23(1): p. 1216.
35. Viraj Soni, S.G., Fraud Detection in Credit Card Transactions: A Machine Learning Approach. Viraj Soni, Sumit Gupta, 2024. 20(17).
36. M. D. Network, "Notification," 2015, https://developer.mozilla.org/en/ docs/Web/API/notification.
37. Anjum, N. and S. Alam, A comparative analysis on widely used web frameworks to choose the requirement based development technology. Int. Adv. Res. J. Sci. Eng. Technol, 2019. 6(9).
38. Brankovic, A., UAE Tolerance Framework as a Base for Coexistence in a Multicultural Society. International Journal Of Civilizations Studies & Tolerance Sciences, 2025. 1: p. 1-115".
39. Anjum, N. and R. Chowdhury, Revolutionizing Cybersecurity Audit through Artificial Intelligence Automation: A Comprehensive Exploration. International Journal of Advanced Research in Computer and Communication Engineering, 2024. 13(5): p. 493-502.
40. Chan, M., et al., Embedded view of third-party data and template generation for a communication platform. 2025, Google Patents.
41. Anjum, N., M. Habiba, and M.R. Islam. A Security Application for Smart Phone and Mobile Device. in International Conference on Informatics, Electronics & Vision. 2013.
42. Deguine, J.-P., et al., Agroecological crop protection for sustainable agriculture. Advances in agronomy, 2023. 178: p. 1-59.
43. Anjum, N. and A. Kabir, Introducing Refined Agile Model (RAM) in the context of Bangladesh's Software Development Environment concentrating on the Improvement of Requirement Engineering Process. International Journal of Software Engineering & Applications (IJSEA), 2019. 10(4).
44. Dhal, K., et al., Vision-based guidance for tracking multiple dynamic objects. Journal of Intelligent & Robotic Systems, 2022. 105(3): p. 66.
45. Dhal, K., A. Kashyap, and A. Chakravarthy. Collision avoidance and rendezvous of quadric surfaces moving on planar environments. in 2021 60th IEEE Conference on Decision and Control (CDC). 2021. IEEE.
46. Ho, Q., et al. Rearing Metarhizium anisopliae fungi at the household level for management of brown planthoppers in rice fields. in 28th International Rice Research Conference, Hanoi, Vietnam. 2010.
47. Dhal, K., A. Kashyap, and A. Chakravarthy, Collision avoidance of 3-dimensional objects in dynamic environments. arXiv preprint arXiv:2203.09037, 2022.
48. Dixit, J.J.a.S., The AI Renaissance Innovations, Ethics, and the Future of Intelligent Systems. Vol. 1. 2023: The International Open Access Publisher. 1-300.
49. Goruntla, N., et al., Evaluation of rational drug use based on who/inrud core drug use indicators in a secondary care hospital: a cross-sectional study in western Uganda. Drug, Healthcare and Patient Safety, 2023: p. 125-135.
50. Seleniumhq.org, "Selenium - web browser automation," 2015, http:// www.seleniumhq.org/X. Li, R.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

51. Jenkins-ci.org, “Welcome to jenkins ci! - jenkins ci,” 2015, http:// jenkins-ci.org/.