

# National College of Ireland

## Teamit Technical Report



Due 16<sup>th</sup> May 2021

BSHCSDE4  
Software Development  
2020/2021  
Kevin Brennan  
16149823  
[x16149823@student.ncirl.ie](mailto:x16149823@student.ncirl.ie)

Supervisor: Aqeel Kazmi

1	Executive Summary.....	4
2	Introduction .....	5
2.1	Background.....	5
2.2	Aims .....	5
2.3	Technology .....	7
2.4	Structure .....	8
3	System.....	10
3.1	Requirements Specification.....	10
3.2	Functional Requirements .....	10
3.2.1	Use Case 1 .....	14
3.2.2	Use Case 2 .....	16
3.2.3	Use Case 3 .....	18
3.2.4	Use Case 4 .....	20
3.2.5	Use Case 5 .....	22
3.2.6	Use Case 6 .....	24
3.2.7	Use Case 7 .....	26
3.2.8	Use Case 8 .....	28
3.2.9	Use Case 9 .....	30
3.2.10	Use Case 10 .....	32
3.2.11	Use Case 11 .....	34
3.2.12	Use Case 12 .....	36
3.2.13	Use Case 13 .....	38
3.2.14	Use Case 14 .....	40
3.3	Non-Functional Requirements .....	42
3.3.1	Security .....	42
3.3.2	Maintainability.....	42
3.3.3	Reliability .....	42
3.3.4	Recoverability .....	42
3.3.5	Extendibility .....	42
3.4	Data Requirements.....	43
3.4.1	Teams Hierarchy .....	43
3.4.2	Users Hierarchy.....	43
3.4.3	Class Diagram.....	43
3.5	User Requirements.....	44
3.6	Environmental Requirements.....	45
3.7	Usability Requirements .....	45
3.7.1	Learnability .....	45

3.7.2	Presentation.....	45
3.7.3	Maintainability.....	45
3.8	Design & Architecture .....	46
3.9	Implementation.....	46
3.9.1	Registration API .....	48
3.9.2	Sentiment Analysis API .....	50
3.9.3	Angular routing .....	53
3.9.4	Navigation and Behaviour Subjects .....	53
3.9.5	Search .....	54
3.9.6	Metric Computation .....	55
3.9.7	Content Rendering.....	57
3.10	Graphical User Interface (GUI) .....	58
3.10.1	Colour Palette .....	58
3.10.2	Logo Design.....	58
3.10.3	Registration.....	58
3.10.4	App Navigation.....	60
3.10.5	Users Home Dashboard (Journal) .....	61
3.10.6	Admin Dashboard .....	62
3.10.7	Search Dashboard .....	64
3.10.8	Content Creation.....	65
3.10.9	Commenting .....	66
3.10.10	Tooltips .....	67
3.10.11	Popup Notifications .....	68
3.11	Testing .....	68
3.11.1	Manual Testing .....	68
3.11.2	API Testing .....	69
3.11.3	UI Testing .....	71
3.11.4	Usability Testing.....	71
3.12	Evaluation .....	72
4	Conclusions .....	73
5	Further Development or Research .....	74
6	References .....	75
7	Appendices.....	76
7.1	Appendix A: Midpoint Submission .....	76
7.2	Appendix B : Development Machine Specifications.....	84
7.3	Appendix C: API Module Structure .....	85
7.4	Appendix D: User Request Validation .....	86
7.5	Appendix E: CORS Issue Resolution .....	87

7.6	Appendix F: Behaviour Subjects .....	88
7.7	Appendix G: Value Passing Between Components .....	89
7.8	Appendix H: Query Creation.....	90
7.9	Appendix I: Tag and Collaboration Analysis .....	91
7.10	Appendix J: Showcase Poster .....	92
7.11	Appendix K: Reflective Journals.....	93
7.11.1	Introduction .....	93
7.11.2	October 2020 .....	94
7.11.3	November 2020 .....	95
7.11.4	December 2020.....	97
7.11.5	January 2021 .....	98
7.11.6	February 2021 .....	101
7.11.7	March 2021 .....	102
7.11.8	April 2021.....	104
7.11.9	May 2021 .....	106
7.12	Other materials used .....	107
7.12.1	Viability Survey .....	107

# 1 Executive Summary

This report contains comprehensive documentation detailing all aspects of work undertaken as part of this final fourth-year project. Theoretically and practically, the project builds upon four years of studying computer science while architecting applications using learned software design patterns and development best practices. **Teamit's** problem domain pertains to many of the shortcomings that professional teams find themselves navigating due to the unprecedented and unique challenges posed by remote working arrangements introduced during Covid-19. In particular, the impact that these new restrictions have on the collaborative profile and capabilities of once office-based teams. **Teamit** is a browser-based platform designed to overcome such collaborative and communicative impediments by offering a feature set that better aligns with the evolving needs of small-sized distributed teams. A team's productivity depends jointly on the cohesion of its members as well as raw talent of individual contributors. A team comprised of highly skilled professionals working to capacity is desirable. However, harmoniously aligned, highly trained professionals who can leverage each other's talents, feeding on each other's learnings and professional growth is optimal. Understanding team dynamics, while driving productivity and monitoring operational efficiencies are regular tasks performed by team leaders and managers. **Teamit** provides various metrics that will assist a team's management with achieving clarity and insight into individuals performance in addition to the collaborative profile and health of their team. This document discusses at length each deliverable of **Teamit**. From initial ideation, through design and planning phases, with in-depth discussion and record of the projects technical implementation.

## 2 Introduction

This section borrows from the previously completed work submitted as part of the initial project proposal (**Appendix A: Midpoint submission**). Several areas needed no additional clarification, while others, *as is the case with technologies used*, are discussed in a deeper level of detail in an upcoming section. While a full, comprehensive record of all technical and implementation details will unfold as the report progresses, the ensuing introduction provides a high-level introduction to the purpose of **Teamit**, and the tools and technologies used to take it from concept to production.

### 2.1 Background

Formal writing aside for a moment, the motivation for selecting the problem domain discussed above is simple and personal. I have worked professionally as part of a small geographically dispersed team for several years. During this time, regular face to face interaction with colleagues has not been possible due to the remote working and travel restrictions. However, I have keenly observed daily communications between many of my colleagues, including acting as a participant in much of the dialogue. My observations highlight an apparent variability in how a remote team communicates compared to a similarly sized office-based team. Removing the face-to-face component of office-based working provides remote employees with the opportunity to hide behind a wall of unavailability by distancing themselves from awkward or challenging situations within the team. Simply put, individuals become slack, and the practice of communication along with the willingness to assist others declines. Throughout 2020/2021, Covid-19 has impacted large portions of the professional world and workforce, pushing many employees into similar scenarios whereby all interactions with teammates, managers and colleagues are now carried out online by default as offices remain closed (*How to reduce the pandemic impact on employees: A guide for company leaders | Deloitte in Ukraine, 2021*). Managing a team of remote employees also has additional challenges that make it increasingly difficult to fully observe teams working dynamics, in turn making it harder to identify unbeneficial interactions between team members, making it more challenging to persist sustained levels of productivity.

### 2.2 Aims

**Teamit** aims to provide small, remote teams with a platform that allows for richer, more inclusive collaboration, increased productivity and operational velocity. **Teamit** aims to serve two types of end user, namely management and the individual team contributors. While both users will interact with what is essentially the same system, **Teamit** aims to provide each with a uniquely tailored end user experience. The project objectives as detailed in the project plan succinctly defines what **Teamit** aims to achieve under the following headings.

#### Productivity

The **productivity** aims of **Teamit** include the following;

- provide small, remote teams with a suite of tools that encourages cross-team collaboration, communication and cohesion.
- provide users with a journal for tracking daily tasks, providing frequent updates and seeking help when blocked.
- reduce the time required to run daily stand-ups recursively providing managers with lengthy and redundant updates.

- allow users search from a back catalogue of previously discussed issues and find resolutions to their questions, quicker and more efficiently than waiting for a colleague to assist. The system aims to guide platform users when seeking help, providing intelligence on who within the team is most qualified to answer their query.

### **Performance**

A teams performance is key when assessing how well a group of professionals function as a unit. **Teamit** aims to provide management and team leaders with a contextual dashboard that drills down into the performance characteristics of teams under their leadership. The platform and product generally strive to achieve this in a fashion that does not alarm the individual contributor, making he or she feel micro-managed.

The **performance** aims of **Teamit** include the following;

- provide administrators with the functionality to perform administrative activities such as adding/removing users to the teams they manage. As part of the performance aims of Teamit, the platform aims to provide managers with functionality that allows for reviewing engagement and collaboration metrics generated by the platform.

### **Efficiency**

The **efficiency** aims of **Teamit** include the following;

- provide users with a searchable and well documented catalogue of archived communications. **Teamit** aims to provide teams with the tools needed to unlock maximum efficiency while limiting downtime.
- provide a set of tools, allowing teams to revisit previously discussed tasks by reviewing archived communications and previously documented dialogue. **Teamit** aims to intuitively structure and deliver the resulting queried data through a user interface that is easily navigable, intuitive and pleasurable to use.

### **Inclusivity**

The **inclusivity** aims of **Teamit** include the following;

- reduce the social barriers between team members who find themselves forced to work online as a by-product of the Covid-19 pandemic. Some individuals are more resilient to working remotely. However, it is not for everyone. **Teamit** aims to overcome several of these obstacles by providing personal information about team members, delivering more emotionally led opportunities to bond with fellow colleagues. **Teamit** aims to achieve this by including the following detail on user profiles, time zones, hours of work, interests, birthdays, hobbies and location to name a few.

## 2.3 Technology

**Teamit** will be developed using a combination of industry leading technologies leveraged for both backend and frontend development. The upcoming section contains information pertaining to many of the third-party frameworks and libraries that will be leveraged while developing **Teamit's** front and backend components.

### Backend

Go, a language developed at Google by several highly influential computer scientists a little over a decade ago, will be used to write **Teamit's** backend APIs. Designed for scalability and speed Go is an appropriate choice of language for backend API implementation (*Go: Perhaps the Best Language for Building Scalable Code*, 2021).

Go is a statically typed language, which when built, compiles programs into static binaries that are executable on a wide variety of hardware, running on a number of different operating systems. **Teamit's** backend is comprised of several smaller micro-services, which when combined serve as a larger unit of overall functionality. This type of architectural design has been carefully chosen for several reasons, most of which allow for easier, less bug-prone development of the individual units of functionality.

An example of this service segregation in action has been to separate the Sentiment Analysis API and Registration API into isolated binaries, running independently of one another while providing different services to the main platform. Concerning production workloads, an issue with the Registration API should not prevent existing users from accessing the platform. Compared to other server-side languages, Go has a very strong, feature-rich standard library that will reduce the requirement to rely on excessive use of external dependencies resulting in operational efficiencies and a reduced resource footprint.

The following table reveals several modules, libraries and packages that will be leveraged while working through the associated implementation of both Registration and Sentiment Analysis APIs.

Standard Library (Go v1.16)	External Dependencies
encoding/json net/http context errors fmt os	github.com/grassmudhorses/vader-go/sentitext github.com/grassmudhorses/vader-go/lexicon github.com/okta/okta-sdk-golang/v2/okta google.golang.org/api/iterator cloud.google.com/go/firestore google.golang.org/api/option github.com/sirupsen/logrus

### Frontend

**Teamit's** UI will rely on a popular front-end framework, Angular. Angular is an widely adopted, enterprise-grade framework primarily designed to facilitate development of single-page web applications (SPA). In recent years the maintainers of Angular framework adopted Typescript as the officially supported development language (*Angular 2: Built on TypeScript | TypeScript*, 2021).



Typescript has additional benefits over regular JavaScript, most notably type checking, a welcome addition to a browser-based scripting language. The Typescript transpiler converts Typescript codes into vanilla JavaScript, allowing execution compatibility with most modern web browsers. Angular is intended as a component-based architecture that facilitates the development of reusable, modular components. Combined with a powerful templating engine, Angular allows for dynamically rendered views. In addition to Angular, PrimeNG, a highly performant component library will provide building blocks that will ease the front-end development workload. PrimeNG ships with a suite of icons, themes and other stylistic features that will make **Teamit** attractive and intuitive.

<https://angular.io>

<https://www.primefaces.org/primeng/>

### **Persistence**

Firestore is a NoSQL, non-relational database, available as part of the Firebase suite of tools provided by Google. Firestore will provide data storage functionalities that **Teamit** will leverage to persist platform data. The Firebase Admin SDK provides a set of convenient tools for reading, writing and manipulating data. **Teamit's** data model implements a non-relational schema, so a highly performant object storage (Firestore) is the preferred choice. Data accessibility, reliability, availability, redundancy and security are complex undertakings to implement from scratch. Offloading this overhead is highly advantageous, making **Teamit** more cost-effective, fault tolerant and straightforward to develop (*What are the key benefits of a Database Managed Service?, 2021*).

### **Development Environment (IDE's)**

GoLand, and Webstorm are the preferred IDE's that will be used to develop **Teamit**. Both these IDE's offer a powerful set of development features and support for their respective languages.

## **2.4 Structure**

The remainder of this report has been divided into a series of logical sections presented in the following order for ease of reading.

### **System**

**System** consists of the main body of research, system design documentation, requirements elicitation, implementation detail, testing and evaluation. All aspects of **Teamit's** internals and externals are extensively documented throughout this section and should be used as the definitive go-to for learning more about any of the deeper technical aspects of **Teamit's** delivery.

### **Conclusions**

A section containing in-depth retrospective and reflection regarding the relative success of **Teamit** throughout the design, implementation and testing phases. **Conclusions** contain impressions and feedback from the developer of the platform, along with several user accounts of their interactions with the platform, revealing whether or not the implemented scope of work adequately delivers appropriate solutions to the initially identified problem domain.

### **Further Research and Development**

A section dedicated to exploring and discussing any realisations and alternate ideas that manifested while working on **Teamit** that could further enhance the production readiness of the product, making **Teamit** a more viable choice for use in enterprise organisations.

### **References**

A comprehensive record of all supporting reference material consulted while developing **Teamit**.

### **Appendices**

Reflective journals, initial opening project plan and any other materials that were identified as beneficial while working on **Teamit's** design and implementation.

## 3 System

### 3.1 Requirements Specification

Before detailing both functional and non-functional requirements of **Teamit**, the high-level overarching requirements of the systems are listed as follows;

1. The system must be intuitive and support ease of use. The systems functionality should be self-documenting. There should no ambiguity as to what the system is capable of. A user should feel empowered by the systems functionality after a few short minutes of interaction.
2. The system must be capable of delivering tailored content and views for both of the expected user types, team administrators and team contributors.
3. The administration (metrics) view must provide system owners, managers and team admins with the capability to view performance metrics related to their team's productivity, in addition to providing the ability for managers to communicate with their team through the system. The system must support real time communication between users.
4. The system must provide a searchable catalogue of queries and issues previously resolved using the system as a collaboration medium.
5. The system should allow team admins and contributors to share and consume contextual information about each other's environment such as geographical location, time zone, local weather, hobbies etc.

### 3.2 Functional Requirements

Functional requirements are defined as a set core actions and capabilities that a given system must be capable of accomplishing to be considered functional. **Teamit's** functional requirements are split between two types of identified users expected to interact with the system. Most of these requirements are applicable to all users, with some additional requirements defining actions specific to team administrators.

It is worth noting that the systems design dictates that all *users are administrators* in a specific capacity. When an individual creates a **Teamit** account, by default they are assigned one team on creation, their own, which they administer and can invite other members to join. In this way, all functional requirements specific to administrators, are by default inherited by all users of at least one team.

#### Admin User Requirements

FR No.	FR Title	Actor
FR-1	Admin User should be able to add users to the teams they administer.	Admin
FR-2	Admin User should be able to remove users from the teams they administer.	Admin
FR-3	Admin User should be able to view team specific performance metrics.	Admin
FR-4	Admin User should be able to delete teams they administer.	Admin
FR-5	Admin User should be able to view/manage their team's from a dedicated team's page.	Admin

## **General User Requirements**

<b>FR No.</b>	<b>Functional Requirement</b>	<b>Actor</b>
FR-6	User should be able to create an account.	User
FR-7	User should be able to login to their account.	User
FR-8	User should be able to logout of their account.	User
FR-9	User should be able to delete their account.	User
FR-10	User should be able to create additional teams.	User
FR-11	User should be able to add/modify profile information from a preferences page.	User
FR-12	User should be able to add/modify content on their own personal journals.	User
FR-13	User should be able to comment on contributions posted by other users.	User
FR-14	User should be able to choose from a selection of templated contribution types [to-do, query, blog].	User
FR-15	User should be able to tag their contributions.	User
FR-16	User should be able to search through all contributions of a team from a dedicated search page.	User
FR-17	User should be provided with suggestions based on their search query.	User

### **FR-1**

Refers to the process of allowing an authenticated user with administrator permissions within a team to add additional users. This action is reserved for team administrators and will not be available to general members of the team.

### **FR-2**

Refers to the process of allowing an authenticated user to remove existing users from a team, which is a reserved action, allowable only by team admins and will not be available to general members.

### **FR-3**

Refers to the process of providing metrics to an authenticated user with admin permissions within the team. Visibility of these metrics is reserved for the team administrator and will not be available for general consumption.

### **FR-4**

Refers to the process of allowing team deletion from the database. This is an administrator reserved action which is not available for general members of a team.

### **FR-5**

Refers to the process of facilitating team management within the system, factoring in a number of previously documented requirements. Specifically, this requirement should provide centralized access to team management activities.

### **FR-6**

Refers to the process of allowing account creation, which is accessible to the user via the public internet. Appropriate validation and verification checks are factored into the scope of this requirement.

### **FR-7**

Refers to the process of allowing a user to log into an existing account provided they have successfully presented credentials that satisfy the users unique login requirements.

**FR-8**

Refers to the process of allowing an authenticated user to securely terminate their session. This action is final. Subsequent attempts to interact with the system will require the user to re-authenticate.

**FR-9**

Refers to the process of enabling a user to permanently delete their account. Deletion of a user's account should not result in their individual contributions being permanently removed from the database.

**FR-10**

Refers to the process of allowing an authenticated user to create additional teams within the system. This action should result in a newly created team whereby the author is the both team administrator and primary owner.

**FR-11**

Refers to the process of allowing an authenticated user to author modifications to their personal information retained by the system and viewable by other team members.

**FR-12**

Refers to the process of allowing an authenticated user to post content to their journal. The content must assume one of the following object types

- To-do,
- Query
- Blog

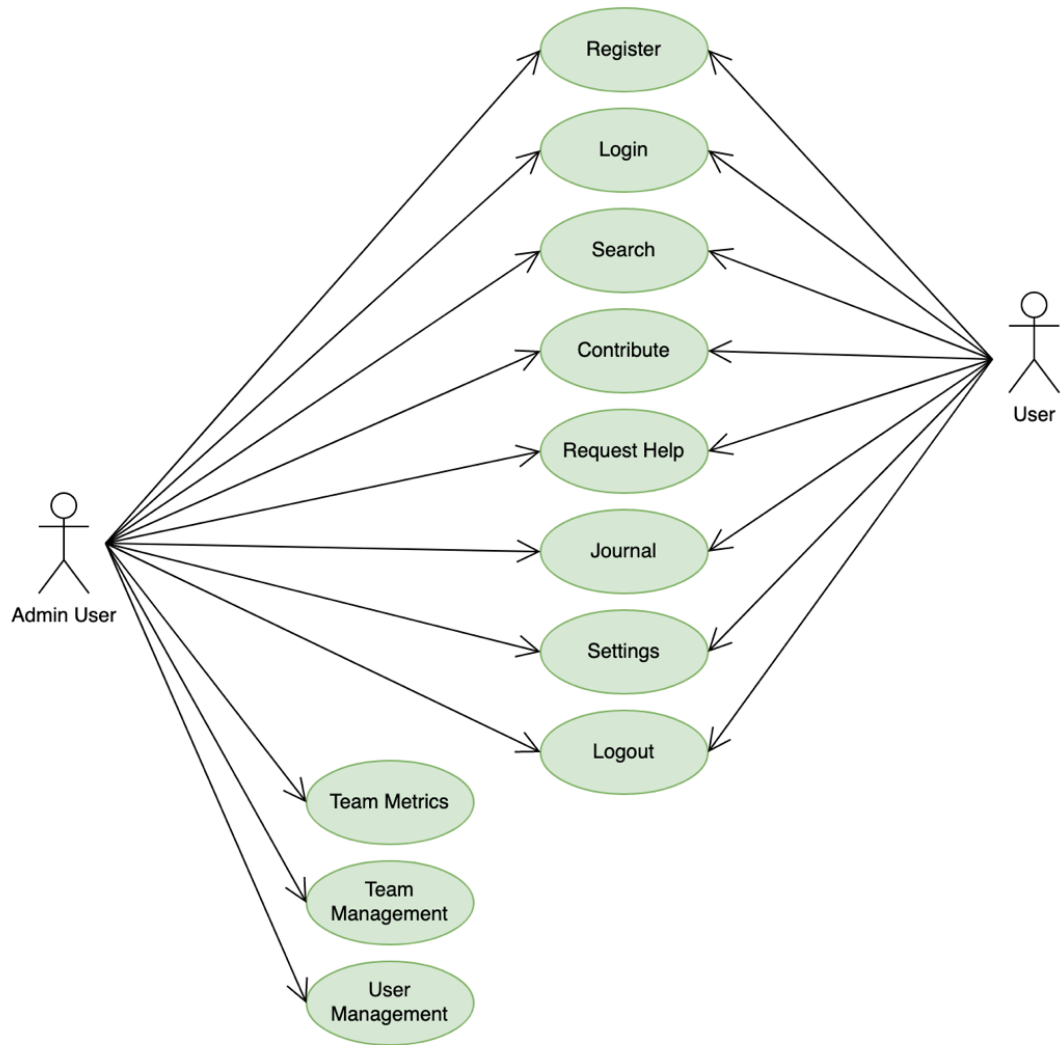
Included within the scope of this definition is the requirement for an authenticated user to author modifications to previously posted content.

**FR-13**

Refers to the process of providing an authenticated with functionality that allows them comment on content posted by their teammates.

**FR-14**

Refers to the process of providing an authenticated user with functionality that allows easy searching of previous conversations and interactions between team members and individual contributors.



*Figure 1 Teamit's Use Case Diagram*

### 3.2.1 Use Case 1

#### NAME ADDING MEMBERS TO A TEAM

ID	UC_01
DESCRIPTION	Refers to the process of allowing an authenticated user with administrator permissions within a team to add additional users. This action is reserved for team administrators and will not be available to general members of the team.
ACTORS	Admin User
USE CASE DIAGRAM	<pre> graph LR     subgraph "Adding members to a team"         AdminUser[Admin User] --&gt; TMD[Team Management Dashboard]         TMD -.-&gt; « includes »  TC[Team Creation]         TC -.-&gt; « extends »  Team[Team]         Team -.-&gt; « includes »  AUT[Add user to team]         AUT -.-&gt; « extends »  Reg[Registration]         User[User] --&gt; AUT     end </pre> <p style="text-align: right;"><i>Figure 2 Adding Members to a Team</i></p>
TRIGGERS	Use case commences when a team administrator attempts to add an additional user to a team they administer.
PRECONDITIONS	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> <li>▪ Initiating user is logged in.</li> <li>▪ Initiating user has the required administration permissions needed to complete the action described.</li> <li>▪ Initiating user has knowledge of the invitees email address.</li> </ul>
POSTCONDITIONS	<ul style="list-style-type: none"> <li>▪ A visual indicator is presented to the user as successful confirmation that the user has been added to the specified team.</li> <li>▪ Newly added user will become visible to existing team members.</li> <li>▪ User will see a new team in their list of teams.</li> <li>▪ User will be able to contribute within scope of the newly added team.</li> </ul>
MAIN FLOW	<ol style="list-style-type: none"> <li>1. Team management page is opened by the admin user.</li> <li>2. Admin user chooses a team to manage by clicking on the appropriate team icon.</li> </ol>

	<ol style="list-style-type: none"> <li>3. Admin user enters an email address that identifies the user to be added.</li> <li>4. Admin user submits the request [A1: User Exists, A2: Empty identifier provided ]</li> <li>5. The system adds the user.</li> <li>6. The system updates the team members contained in the database to reflect the addition of the newly added user.</li> <li>7. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	<p>A1 : User Exists</p> <ul style="list-style-type: none"> <li>▪ The system notifies the initiating user that a matching identifier is already present within the database.</li> <li>▪ Use case continues from position 3 of the main flow.</li> </ul> <p>A2 : Empty identifier provided</p> <ul style="list-style-type: none"> <li>▪ The system notifies the user that a valid email identifier is required.</li> <li>▪ The use case continues from position 3 of the main flow.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a new user has been added and the initiating user navigates away from their teams management page.</li> <li>▪ The system goes into a wait state</li> </ul>



### 3.2.2 Use Case 2

#### NAME REMOVING USER FROM A TEAM

ID	UC_02
DESCRIPTION	Refers to the process of allowing an authenticated user to remove existing users from a given team, which is a reserved action, allowable only by team admins and will not be available to general members.
ACTORS	Admin User
USE CASE DIAGRAM	<pre> graph LR     AdminUser[Admin User] --&gt; TMD[Team Management Dashboard]     TMD -.-&gt; &lt;&lt;includes&gt;&gt;  Team[Team]     Team -.-&gt; &lt;&lt;includes&gt;&gt;  RemoveUser[Remove user]     RemoveUser -.-&gt; &lt;&lt;includes&gt;&gt;  ConfirmAction[Confirm Action]     UserAdded([User Added]) -.-&gt; &lt;&lt;extends&gt;&gt;  RemoveUser     </pre> <p>The diagram illustrates the process of removing a user from a team. It starts with an 'Admin User' actor interacting with the 'Team Management Dashboard' use case. This dashboard includes a 'Team' use case, which in turn includes a 'Remove user' use case. The 'Remove user' use case includes a 'Confirm Action' use case. Additionally, a 'User Added' use case extends the 'Remove user' use case. The entire process is titled 'Removing user from a team'.</p>
TRIGGERS	Use case commences when a team administrator attempts to remove a user from an existing team under their administration.
PRECONDITIONS	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> <li>▪ Initiating user is logged in.</li> <li>▪ Initiating user has the required administration permissions needed to complete the described action.</li> <li>▪ Initiating user has navigated to the team management page.</li> <li>▪ Initiating user has selected a team they wish to administer.</li> </ul>
POSTCONDITIONS	<ul style="list-style-type: none"> <li>▪ A visual indicator is displayed to the initiating user.</li> <li>▪ Initiator is provided an information message on completion.</li> <li>▪ The user is removed from the specified team</li> <li>▪ The user will have no further access to the team.</li> </ul>
MAIN FLOW	<ol style="list-style-type: none"> <li>1. Admin user searches for the team-member(s) they wish to remove.</li> <li>2. Admin user selects the team-members(s) they wish to remove</li> <li>3. Admin user submits the request for processing.</li> </ol>

Figure 3 Removing Users from a Team

	<ol style="list-style-type: none"> <li>4. The system requests that the user confirm their intent. [E1: No confirmation]</li> <li>5. The user confirms that their intent is correct.</li> <li>6. The system removes the specified user from the team.</li> <li>7. The system updates the database to reflect the changes</li> <li>8. Admin user is presented with a success message. [E2 : Transient Error]</li> <li>9. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	
<b>EXCEPTIONAL FLOW</b>	<p>E1 : No confirmation</p> <ul style="list-style-type: none"> <li>▪ The user declines to confirm the action.</li> <li>▪ User removal cannot successfully complete without confirmation from the initiator.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul> <p>E2 : Transient Error</p> <ul style="list-style-type: none"> <li>▪ Error occurs during task execution.</li> <li>▪ System informs the initiator than an error occurred.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates successfully when the specified user(s) user have been removed and the initiating user navigates away from their teams management page.</li> <li>▪ The system goes into a wait state</li> </ul>

### 3.2.3 Use Case 3

<b>NAME</b>	<b>PERFORMANCE METRICS DASHBOARD</b>
<b>ID</b>	UC_03
<b>DESCRIPTION</b>	Refers to the process of providing metrics to an authenticated user with admin permissions within the team. Visibility of these metrics is reserved for the team administrator(s) and will not be available for general consumption.
<b>ACTORS</b>	Admin User
<b>USE CASE DIAGRAM</b>	<pre> graph LR     AdminUser[Admin User] --&gt; Team((Team))     Team -.-&gt; &lt;&lt; includes &gt;&gt;  PerformanceDashboard((Performance Dashboard))     PerformanceDashboard -.-&gt; &lt;&lt; includes &gt;&gt;  DailyPostCount((Daily post count))     PerformanceDashboard -.-&gt; &lt;&lt; includes &gt;&gt;  MostFrequentContributor((Most frequent contributor))     PerformanceDashboard -.-&gt; &lt;&lt; includes &gt;&gt;  QueryResolutionTimer((Query Resolution timer))     PerformanceDashboard -.-&gt; &lt;&lt; includes &gt;&gt;  SentimentAnalyzer((Sentiment Analyzer))     PerformanceDashboard -.-&gt; &lt;&lt; includes &gt;&gt;  MostFrequentResolver((Most frequent resolver))     </pre> <p style="text-align: right;"><i>Figure 4 Performance Metrics Dashboard</i></p>
<b>TRIGGERS</b>	Use case commences when a team administrator navigates to the performance dashboard of a given team.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> <li>▪ Initiating user is logged in.</li> <li>▪ Initiating user has the required administration permissions needed to complete the described action.</li> </ul>
<b>POSTCONDITIONS</b>	
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. Admin user navigates to the performance dashboard page of the currently active team.</li> <li>2. Admin user browses through performance metrics generated by the system. [A1 : No metrics available]</li> <li>3. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	A1 : No metrics available

	<ul style="list-style-type: none"> <li>▪ No metrics are available</li> <li>▪ The system prompts the user to encourage more team engagements so that metrics can generate.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates successfully when valid performance metrics have been calculated and presented to the user.</li> <li>▪ The system goes into a wait state</li> </ul>

### 3.2.4 Use Case 4

<b>NAME</b>	<b>REMOVING AN ENTIRE TEAM</b>
<b>ID</b>	UC_04
<b>DESCRIPTION</b>	Refers to the process of allowing team deletion from the database. This is an administrator reserved action which is not available for general members of a team.
<b>ACTORS</b>	Admin User
<b>USE CASE DIAGRAM</b>	<pre> graph TD     AdminUser[Admin User] --&gt; TMD[Team Management Dashboard]     TMD -.-&gt; &lt;&lt; includes &gt;&gt;  Team[Team]     TMD -.-&gt; &lt;&lt; extends &gt;&gt;  TC[Team Creation]     Team -.-&gt; &lt;&lt; includes &gt;&gt;  DT[Delete Team]     TC -.-&gt; &lt;&lt; includes &gt;&gt;  DT     DT -.-&gt; &lt;&lt; includes &gt;&gt;  CA[Confirm Action]     </pre> <p>The diagram illustrates the process of removing an entire team. It features an actor 'Admin User' who interacts with the 'Team Management Dashboard' use case. From the dashboard, the flow can either include the 'Team' use case or extend to 'Team Creation'. Both paths lead to the 'Delete Team' use case, which then includes a 'Confirm Action' use case. The entire process is titled 'Removing an entire team'.</p>
<b>TRIGGERS</b>	Use case commences when a team administrator tries to delete a team from the system.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> <li>▪ Initiating user is logged in.</li> <li>▪ Initiating user has the required administration permissions needed to complete the described action.</li> <li>▪ Initiating user has navigated to the team management page.</li> <li>▪ Initiating user has selected a team they wish to administer.</li> </ul>
<b>POSTCONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ A visual indicator is displayed to the initiating user.</li> <li>▪ Initiator is provided an information message on completion.</li> <li>▪ The specified team is removed.</li> <li>▪ All remaining members will lose access to the team, and all of the teams historic data will be deleted.</li> </ul>
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. Admin user searches for the team they wish to remove.</li> <li>2. Admin user selects the team they wish to remove</li> <li>3. Admin user submits the request for processing.</li> <li>4. The system requests that the user confirm their intent. [E1: No confirmation]</li> </ol>

Figure 5 Remove Entire Team

	<ol style="list-style-type: none"> <li>5. The user confirms that their intent is correct.</li> <li>6. The system removes the specified team.</li> <li>7. The system updates the database to reflect the changes</li> <li>8. Admin user is presented with a success message. E2 : Retry]</li> <li>9. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	<p>A1 : Retry</p> <ul style="list-style-type: none"> <li>▪ An error occurred during task execution.</li> <li>▪ The system alerts the user that an error has occurred.</li> <li>▪ The system will prompt the user to try again.</li> <li>▪ Admin user chooses to retry the action.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	<p>E1 : No confirmation</p> <ul style="list-style-type: none"> <li>▪ The user declines to confirm the action.</li> <li>▪ Team removal cannot successfully complete without confirmation from the initiator.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul> <p>E2 : Retry</p> <ul style="list-style-type: none"> <li>▪ An error occurred during task execution.</li> <li>▪ The system alerts the user that an error has occurred.</li> <li>▪ The system will prompt the user to try again.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a team is successfully deleted from the system.</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.5 Use Case 5

<b>NAME</b>	<b>GENERAL TEAM MANAGEMENT</b>
<b>ID</b>	UC_05
<b>DESCRIPTION</b>	Refers to the process of team management within the system, factoring in a number of previously documented requirements. Specifically, this requirement should provide centralized access to team management activities.
<b>ACTORS</b>	Admin User
<b>USE CASE DIAGRAM</b>	<pre> graph LR     AdminUser[Admin User] --&gt; TMD[Team Management Dashboard]     TMD -.-&gt; MD[Metrics Dashboard]     TMD -.-&gt; TM[Team Management]     TM -.-&gt; UM[User Management]     TM -.-&gt; CT[Create Team]     UM -.-&gt; DU[Delete User]     UM -.-&gt; AU[Add User]     CT -.-&gt; DT[Delete Team]     </pre> <p style="text-align: right;"><i>Figure 6 General Team Management</i></p>
<b>TRIGGERS</b>	Use case commences when a team administrator wishes to perform maintenance or update a team they administer.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> <li>▪ Initiating user is logged in.</li> <li>▪ Initiating user has the required administration permissions needed to complete the described action.</li> </ul>
<b>POSTCONDITIONS</b>	Any changes made to teams or users are reflected throughout the system.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. Admin User navigates to the teams management dashboard.</li> <li>2. Admin User is presented with a tile view of the teams they administer.</li> <li>3. Admin User chooses a team from the list [E1 : No Changes]</li> <li>4. Admin User selects the desired action to perform.</li> <li>5. Admin User confirms intent.</li> <li>6. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	

<b>EXCEPTIONAL FLOW</b>	<p>E1 : No Changes</p> <ul style="list-style-type: none"> <li>▪ Admin user decides no changes are required.</li> <li>▪ Admin user leaves team management dashboard</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully performed a maintenance action on one of the teams they administer.</li> <li>▪ The system goes into a wait state.</li> </ul>



### 3.2.6 Use Case 6

<b>NAME</b>	<b>USER REGISTRATION</b>
<b>ID</b>	UC_06
<b>DESCRIPTION</b>	Refers to the process of allowing account creation, which is accessible to the user via the public internet. Appropriate validation and verification checks are factored within the scope of this requirement.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; Register(Register)     subgraph "User Registration"         Register -- "&lt;&lt; includes &gt;&gt;" -.-&gt; Email((Email))         Register -- "&lt;&lt; includes &gt;&gt;" -.-&gt; Password((Password))         Password -- "&lt;&lt; extends &gt;&gt;" -.-&gt; Validation((Validation))     end </pre> <p style="text-align: right;"><i>Figure 7 User Registration</i></p>
<b>TRIGGERS</b>	Use case commences when a user navigates to the applications home page with intent to register.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> </ul>
<b>POSTCONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ User account is created</li> <li>▪ User will be logged in and directed to their default team management dashboard</li> </ul>
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. User navigates to the applications home page</li> <li>2. User provides an email address</li> <li>3. User provides a password.</li> <li>4. The system validates the provided user credentials [A1 : Invalid credentials], [A2 : Existing Account]</li> <li>5. The system redirects the user to their default team management dashboard.</li> <li>6. The use case terminates successfully</li> </ol>
<b>ALTERNATE FLOW</b>	<p>A1 : Invalid Credentials</p> <ul style="list-style-type: none"> <li>▪ The system notifies the user of invalid credentials</li> <li>▪ The user amends the provided credentials</li> </ul>

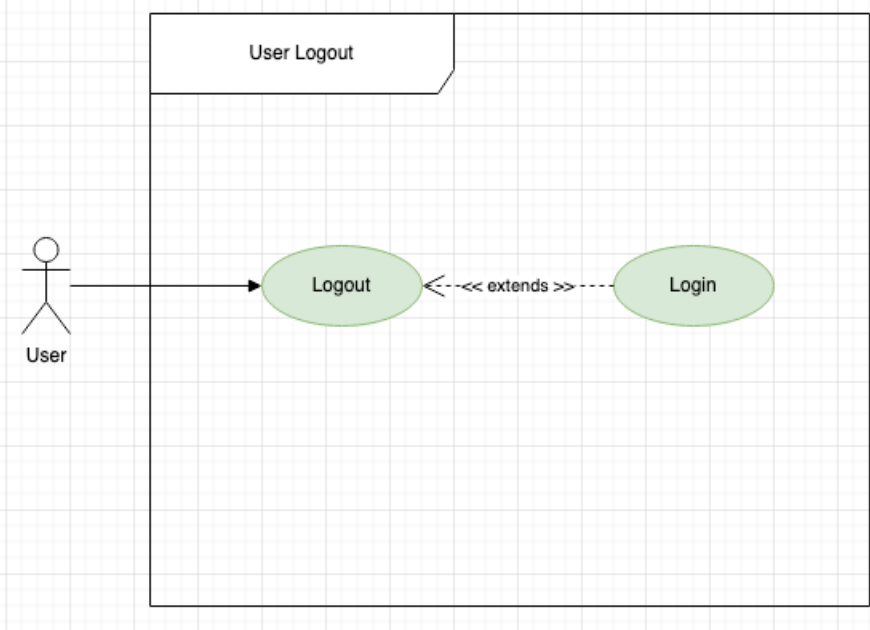
	<ul style="list-style-type: none"> <li>▪ The system validates and accepts the updated credentials</li> <li>▪ Use case continues from step 5 of the main flow</li> </ul> <p>A1 : Existing Account</p> <ul style="list-style-type: none"> <li>▪ Account already exists with the credentials provided</li> <li>▪ The system notifies the user of the conflict.</li> <li>▪ The user amends the provided credentials</li> <li>▪ The systems validates and accepts the updated credentials</li> <li>▪ Use case continues from step 5 of the main flow</li> </ul>
<b>EXCEPTIONAL FLOW</b>	<p>E1 : User Cancels</p> <ul style="list-style-type: none"> <li>▪ The user ends the registration process by leaving the registration page.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully created an account.</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.7 Use Case 7

<b>NAME</b>	<b>USER LOGIN</b>
<b>ID</b>	UC_07
<b>DESCRIPTION</b>	Refers to the process of allowing a user to log into an existing account provided they have successfully presented credentials that satisfy the users unique login requirements.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; Login(Login)     subgraph "User Login"         Login -- "&lt;&lt; includes &gt;&gt;" --&gt; Email((Email))         Login -- "&lt;&lt; includes &gt;&gt;" --&gt; Password((Password))         Registration((Registration)) -- "&lt;&lt; extends &gt;&gt;" --&gt; Login     end </pre> <p style="text-align: right;"><i>Figure 8 User Login</i></p>
<b>TRIGGERS</b>	Use case commences when a user navigates to the applications home page with intent to login.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account.</li> </ul>
<b>POSTCONDITIONS</b>	The user is logged in and able to interact with the system
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to the applications login view.</li> <li>2. The user provides credentials</li> <li>3. The system validates the users credentials [A1 : Incorrect Credentials], [E1 : User Cancellation], [E1 : Invalid User]</li> <li>4. The system redirect the user to their main dashboard</li> <li>5. The use case terminates successfully</li> </ol>
<b>ALTERNATE FLOW</b>	<p>A1 : Incorrect Credentials</p> <ul style="list-style-type: none"> <li>▪ The system validates that the user exist, but credentials were provided incorrectly.</li> <li>▪ The system prompts the user to update provided credentials.</li> <li>▪ The user provides valid credentials</li> <li>▪ Use case continues from step 4 of the main flow.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	E1 : User Cancellation

	<ul style="list-style-type: none"> <li>▪ The user terminates the login flow by navigating away from the login page</li> <li>▪ The use case ends unable to complete the main flow</li> </ul> <p>E2: Invalid User</p> <ul style="list-style-type: none"> <li>▪ The system attempts to validate the provided credentials</li> <li>▪ The system is unable to find an account matching the provided credentials</li> <li>▪ The system notifies the user of the error</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully logged into an existing account.</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.8 Use Case 8

<b>NAME</b>	<b>USER LOGOUT</b>
<b>ID</b>	UC_08
<b>DESCRIPTION</b>	Refers to the process of allowing an authenticated user to securely terminate their session. This action is final. Subsequent attempts to interact with the system will require the user to re-authenticate.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	 <pre> graph LR     User((User)) --&gt; Logout((Logout))     Login((Login)) -.-&gt; &lt;&lt; extends &gt;&gt;  Logout     </pre> <p style="text-align: right;"><i>Figure 9 User Logout</i></p>
<b>TRIGGERS</b>	Use case commences when a user wishes to securely terminate their current session with the platform.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> </ul>
<b>POSTCONDITIONS</b>	The user is logged out and no further actions with the system are possible.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user clicks logout button which is available from all views and components.</li> <li>2. The systems terminates the current users authentication session.</li> <li>3. The system redirects the user to the login page.</li> <li>4. The use case terminates successfully</li> </ol>
<b>ALTERNATE FLOW</b>	
<b>EXCEPTIONAL FLOW</b>	
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully logged out.</li> <li>▪ The system goes into a wait state.</li> </ul>



### 3.2.9 Use Case 9

<b>NAME</b>	<b>USER ACCOUNT DELETION</b>
<b>ID</b>	UC_09
<b>DESCRIPTION</b>	Refers to the process of enabling a user to permanently delete their account. Deletion of a user's account should not result in their individual contributions being permanently wiped from the database.
<b>ACTORS</b>	All users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; US((User Settings))     US -.-&gt; &lt;&lt; includes &gt;&gt;  DA((Delete Account))     DA -.-&gt; &lt;&lt; includes &gt;&gt;  C((Confirmation))   </pre> <p style="text-align: right;"><i>Figure 10 User Account Deletion</i></p>
<b>TRIGGERS</b>	Use case commences when a user wishes to delete (remove) themselves from the system.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> </ul>
<b>POSTCONDITIONS</b>	The user is logged out and no further actions with the system are possible. The user will not be able to access the platform again until a new account is created.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to their user settings component.</li> <li>2. The user locates the 'delete account' option.</li> <li>3. The user chooses to delete their account</li> <li>4. The system requests that the user confirm the action with a simple challenge that requires additional input from the user as an accidental deletion projection. [E1 : Failed Confirmation]</li> <li>5. The user confirms</li> <li>6. The users account is marked as deleted (but not removed).</li> <li>7. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	

<b>EXCEPTIONAL FLOW</b>	E1 : Failed Confirmation <ul style="list-style-type: none"> <li>▪ The user fails to successfully complete the confirmation challenge.</li> <li>▪ The system prompts the user to try again, or cancel the account deletion entirely.</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully deleted their account</li> </ul>



### 3.2.10 Use Case 10

<b>NAME</b>	<b>TEAM CREATION</b>
<b>ID</b>	UC_10
<b>DESCRIPTION</b>	Refers to the process of allowing an authenticated user to create additional teams within the system. This action should result in a newly created team whereby the author is the both team administrator and cardinal member.
<b>ACTORS</b>	All users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; TMD((Team Management Dashboard))     AC((Account Creation)) -.-&gt; &lt;&lt; extends &gt;&gt;  TMD     TMD -.-&gt; &lt;&lt; includes &gt;&gt;  CT((Create Team))     CT -.-&gt; &lt;&lt; includes &gt;&gt;  STN((Select Team Name))   </pre> <p>The diagram illustrates the 'Team Creation' use case. It features a 'User' actor who interacts with the 'Team Management Dashboard' use case. An 'Account Creation' use case is shown as an extension of the 'Team Management Dashboard'. The 'Team Management Dashboard' use case includes the 'Create Team' use case, which in turn includes the 'Select Team Name' use case.</p>
<b>TRIGGERS</b>	Use case commences when a user wishes to create a new team within the system
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> </ul>
<b>POSTCONDITIONS</b>	A new team is created, and the user will see this change reflected in their team management dashboard with the addition of a new team added to the list of teams they administer.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to their team management dashboard.</li> <li>2. The user locates the 'Create Team' option.</li> <li>3. The user chooses to create a new team.</li> <li>4. The system requests the user choose a unique name for the team. [E1 : User Cancellation]</li> <li>5. The user names the new team and clicks create. [A1 : Team Name Exists]</li> <li>6. The new team is created in the system.</li> <li>7. The new team will be viewable from the initiating users team management dashboard.</li> <li>8. The use case terminates successfully.</li> </ol>

Figure 11 Team Creation

<b>ALTERNATE FLOW</b>	<p>A1 : Team Name Exists</p> <ul style="list-style-type: none"> <li>▪ The system identifies that the provided team name matches an existing teams of the same name.</li> <li>▪ The system prompts the user to choose a globally unique name for the team.</li> <li>▪ The user re-enters a name for the team.</li> <li>▪ The selected name satisfies validation</li> <li>▪ Use case continues from step 6 of the main flow.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	<p>E1 : User Cancellation</p> <ul style="list-style-type: none"> <li>▪ The user decides a new team is no longer required.</li> <li>▪ The user navigates away from the team creation functionality.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully created a new team</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.11 Use Case 11

<b>NAME</b>	<b>PROFILE MODIFICATION</b>
<b>ID</b>	UC_11
<b>DESCRIPTION</b>	Refers to the process of allowing an authenticated user to author modifications to their personal information retained by the system and viewable by other team members.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; US((User Settings))     subgraph ProfileModification [Profile Modification]         US --&gt; &lt;&lt; includes &gt;&gt;  TZ((Timezone))         US --&gt; &lt;&lt; includes &gt;&gt;  WH((Working hours))         US --&gt; &lt;&lt; includes &gt;&gt;  L((Location))         US --&gt; &lt;&lt; includes &gt;&gt;  I((Interests))         US --&gt; &lt;&lt; includes &gt;&gt;  B((Birthday))         US --&gt; &lt;&lt; includes &gt;&gt;  H((Hobbies))         AC((Account Creation)) -.-&gt; &lt;&lt; extends &gt;&gt;  US     end </pre> <p style="text-align: right;"><i>Figure 12 Profile Modification</i></p>
<b>TRIGGERS</b>	Use case commences when a user wishes to add or update their profile information within the system.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> </ul>
<b>POSTCONDITIONS</b>	The users profile information will be updated and the changes will be visible by other members of the team.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to the user setting section of their account.</li> <li>2. The user identifies the field they wish to modify [E1 : User Cancellation]</li> <li>3. The user submits the newly authored changes for the system to persist [A1 : Field Validation Error]</li> <li>4. The changes are persisted by the system and visible to other members of the team.</li> <li>5. The use case terminates successfully.</li> </ol>
<b>ALTERNATE FLOW</b>	A1 : Field Validation Error

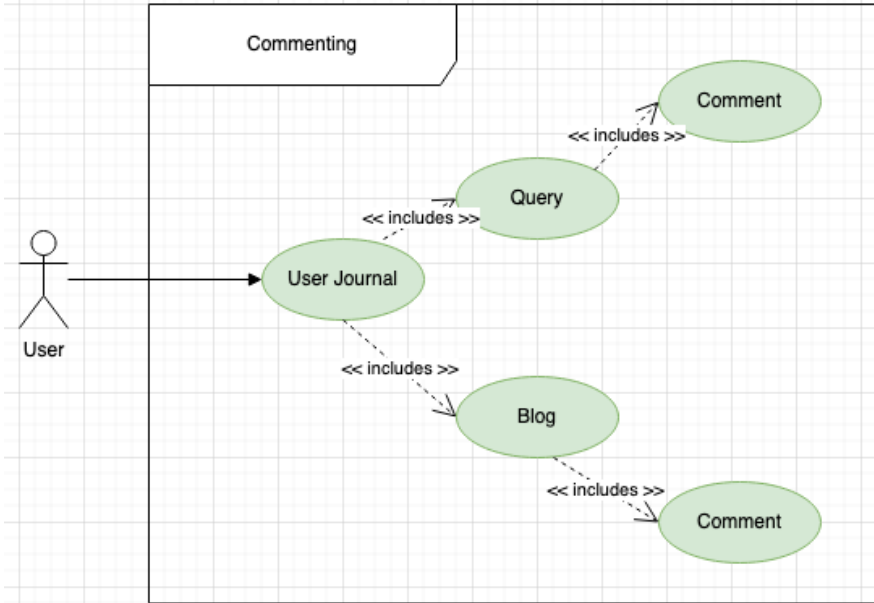
	<ul style="list-style-type: none"> <li>▪ The systems alerts the user that field specific validation has not been successful and the changes will not be saved.</li> <li>▪ The user updates the original input based on the systems feedback.</li> <li>▪ The user re-submits the profile changes</li> <li>▪ The use case continues from step 4 of the main flow.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	<p>E1 : User Cancellation</p> <ul style="list-style-type: none"> <li>▪ The user decides not to make profile changes.</li> <li>▪ The user navigates away from their profile settings page.</li> <li>▪ The use case ends unable to complete the main flow.</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully altered the details of their profile within the system.</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.12 Use Case 12

<b>NAME</b>	<b>CONTENT CREATION</b>
<b>ID</b>	UC_12
<b>DESCRIPTION</b>	Refers to the process of allowing an authenticated user to post content to their journal. The content must assume one of the following object types, to-do, query or blog item.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; Journal((Journal))     Journal -.-&gt; &lt;&lt;includes&gt;&gt;  Query((Query))     Query -.-&gt; &lt;&lt;includes&gt;&gt;  Title((Title))     Query -.-&gt; &lt;&lt;includes&gt;&gt;  Tags((Tags))     Query -.-&gt; &lt;&lt;includes&gt;&gt;  Content1((Content))     Journal -.-&gt; &lt;&lt;includes&gt;&gt;  Blog((Blog))     Blog -.-&gt; &lt;&lt;includes&gt;&gt;  Content2((Content))     </pre> <p style="text-align: right;"><i>Figure 13 Content Creation</i></p>
<b>TRIGGERS</b>	Use case commences when a user wishes to post new content within the system.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> </ul>
<b>POSTCONDITIONS</b>	The user will have successfully created new content within the system which is visible to all users.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to their home dashboard.</li> <li>2. The user chooses a submission type. [E1 : User Cancellation]</li> <li>3. The user proceeds to fill complete all required fields. [A1 : Field Validation Error]</li> <li>4. The user submits the provided details for the system to persist.</li> <li>5. The system displays the newly created information.</li> <li>6. The use case terminates successfully</li> </ol>
<b>ALTERNATE FLOW</b>	<p>A1 : Field Validation Error</p> <ul style="list-style-type: none"> <li>▪ The system informs the user about the missing values.</li> <li>▪ The user updates the submitted data.</li> </ul>

	<ul style="list-style-type: none"> <li>▪ The system continues from step 4 of the main flow.</li> </ul>
<b>EXCEPTIONAL FLOW</b>	<p>E1 : User Cancellation</p> <ul style="list-style-type: none"> <li>▪ The user decides they no long want to share their initial thought.</li> <li>▪ The use case ends unable to complete the main flow</li> </ul>
<b>TERMINATION</b>	<ul style="list-style-type: none"> <li>▪ The use case terminates when a user has successfully submitted some content for creation</li> <li>▪ The system goes into a wait state.</li> </ul>

### 3.2.13 Use Case 13

<b>NAME</b>	<b>COMMENTING</b>
<b>ID</b>	UC_13
<b>DESCRIPTION</b>	Refers to the process of providing an authenticated with functionality that allows for commenting on content posted by their teammates.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	 <pre> graph LR     User((User)) --&gt; UserJournal((User Journal))     UserJournal -.-&gt; &lt;&lt; includes &gt;&gt;  Query((Query))     Query -.-&gt; &lt;&lt; includes &gt;&gt;  Comment1((Comment))     UserJournal -.-&gt; &lt;&lt; includes &gt;&gt;  Blog((Blog))     Blog -.-&gt; &lt;&lt; includes &gt;&gt;  Comment2((Comment))     </pre> <p>The diagram illustrates the 'Commenting' use case. An actor labeled 'User' is connected to a use case labeled 'User Journal'. From 'User Journal', two dashed arrows labeled '&lt;&lt; includes &gt;&gt;' point to 'Query' and 'Blog'. From 'Query', a dashed arrow labeled '&lt;&lt; includes &gt;&gt;' points to a 'Comment' use case. From 'Blog', a dashed arrow labeled '&lt;&lt; includes &gt;&gt;' points to another 'Comment' use case. The entire diagram is enclosed in a box labeled 'Commenting'.</p> <p style="text-align: right;"><i>Figure 14 Commenting</i></p>
<b>TRIGGERS</b>	Use case commences when a user wishes to comment on another team members post.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> <li>▪ Another users post exists to comment</li> </ul>
<b>POSTCONDITIONS</b>	The user will have successfully contributed to another team members post.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user navigates to a team members dashboard.</li> <li>2. The user chooses a post that they wish to comment on.</li> <li>3. The user proceeds to create a response, submitting upon completion. [E1 : User Cancellation]</li> <li>4. The system displays the newly created comment in a chronological fashion.</li> <li>5. The use case terminates successfully</li> </ol>
<b>ALTERNATE FLOW</b>	
<b>EXCEPTIONAL FLOW</b>	E1 : User Cancellation

**TERMINATION**

- The user decides they no long want to contribute to the selected post.
  - The use case ends unable to complete the main flow
- 
- The use case terminates when a user has successfully commented another users post.
  - The system goes into a wait state.



### 3.2.14 Use Case 14

<b>NAME</b>	<b>CONTENT SEARCHING</b>
<b>ID</b>	UC_14
<b>DESCRIPTION</b>	Refers to the process of providing an authenticated user with functionality that allows easy searching of previous conversations and interactions between team members and individual contributors.
<b>ACTORS</b>	All Users
<b>USE CASE DIAGRAM</b>	<pre> graph LR     User((User)) --&gt; SearchPage(Search Page)     SearchPage -.-&gt; « includes »  SearchBar(SearchBar)     SearchBar -.-&gt; « extends »  UserBlog[User Blog/Queries content]     subgraph ContentSearching [Content Searching]         SearchPage         SearchBar         UserBlog     end </pre> <p style="text-align: right;"><i>Figure 15 Content Searching</i></p>
<b>TRIGGERS</b>	Use case commences when a user navigates to the search dashboard with the intent of searching for a particular term.
<b>PRECONDITIONS</b>	<ul style="list-style-type: none"> <li>▪ The system is hosted and accessible from the internet.</li> <li>▪ Initiating user has an active account</li> <li>▪ Initiating user is logged in</li> <li>▪ Blog and query posts exist within the system so that searching finds data.</li> </ul>
<b>POSTCONDITIONS</b>	The user will have successfully validated if the system contains any information of use to them based on specified search term.
<b>MAIN FLOW</b>	<ol style="list-style-type: none"> <li>1. The user inputs a search term within the search box</li> <li>2. The system returns an updated list of possible matches in response to each keystroke.</li> <li>3. The user has access to information once loaded. [E1: No match]</li> </ol>
<b>ALTERNATE FLOW</b>	
<b>EXCEPTIONAL FLOW</b>	<p>E1: No Match</p> <ul style="list-style-type: none"> <li>▪ The system is unable to located any records matching the specified search term.</li> </ul>

**TERMINATION**

- The user is prompted to try an alternate search phrase.
- The use case terminates successfully when a user is provided with a set of search results specific to the specified search term.
- The system goes into a wait state.

## 3.3 Non-Functional Requirements

### 3.3.1 Security

Many security requirements initially appear to be non-functional. However, many of these requirements in fact transition into concrete functional requirements in some capacity with regards to implementation. An example of a non-functional security requirement concerning **Teamit** relates to the applications multitenancy. The system must protect users data, avoiding situations whereby a given user is exposed to information that is not their own.

All users of the application interact with what is essentially the same instance, meaning it is the responsibility of the application to guard user data and only return data specific to the current user's session. The application must also guard specific routes, allowing access to only those users who are authenticated. Concerning user authentication, a reputable third-party authentication service has been leveraged (Okta). A user exchanges credentials for an OIDC token which is used for future interaction with the system, ensuring that both authorisation and authentication are being duly performed upon login. The same authentication framework leverages the users token to implement route guarding within the application itself, further increasing the applications gated accessibility and security posture. (*What is Single Tenant vs Multi-Tenant Software? | Liquid Web, 2021*)

### 3.3.2 Maintainability

The maintainability of **Teamit** is vital to ensuring a high release cadence whereby newly developed features can be deployed to production regularly without requiring significant refactoring of the entire code base. The maintainability of **Teamit** is demonstrated by using a highly modular component-based architecture for front end development. Coupled with additional custom API's supporting user registration and sentiment analysis, a great deal of care and attention has been afforded to the future maintainability and extendibility of the application.

### 3.3.3 Reliability

The reliability of any software is paramount if its customers are to remain loyal and satisfied with the product. **Teamit** leverages several third-party services which deal with application and database hosting, each of which offers impressive enterprise-level SLA's (Service-Level Agreements). In doing so **Teamit** can present an impressive level of uptime by eliminating all self-hosted services.

### 3.3.4 Recoverability

The recoverability requirements of **Teamit** primarily concern user data and platform data as they are most mission-critical data sets. Adequate recoverability of each must be in place before this non-functional requirement can be considered satisfied. Both Okta and Firebase offer regular automated backups as part of their cloud service offering, meaning all data can be recovered by point-in-time timestamp in the event of accidental deletion or perhaps a more deliberate and sinister act of malice.

### 3.3.5 Extendibility

Similar to many of the points already outlined within maintainability, **Teamit** has been developed with extensibility in mind, reducing the barrier to entry for new developers contributing to the application's codebase.

### 3.4 Data Requirements

This section details the data requirements of **Teamit**. As previously discussed, Firestore has been chosen as the platform's data persistence layer. In order to better understand the specifics of **Teamit's** data model, and how it has evolved while keeping compatibility with Firestore's data structuring in mind, understanding Firestore data structure is important. Firestore is a document oriented database which stores data by collection reference, which in turn holds document references. More information is available by referring directly to Firestore's documentation. (*Cloud Firestore Data model | Firebase, 2021*)

#### 3.4.1 Teams Hierarchy

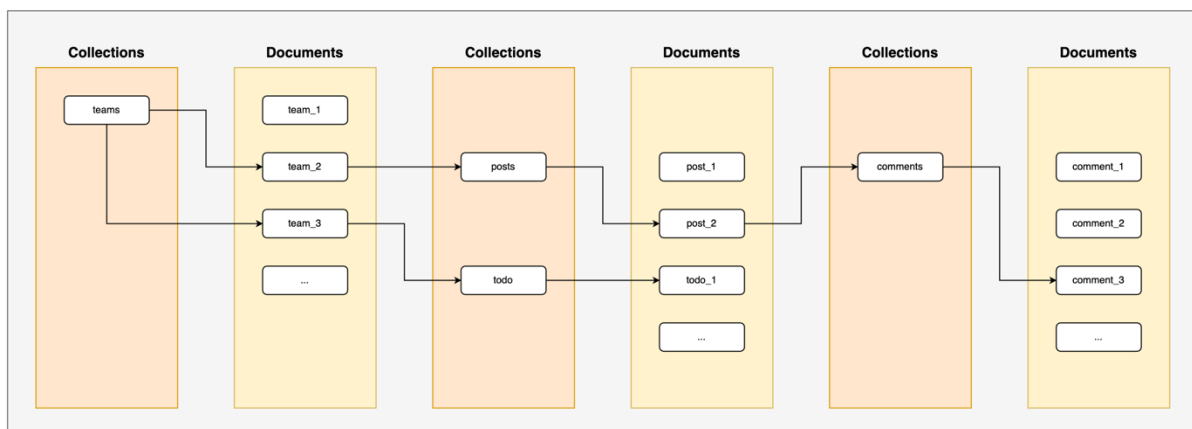


Figure 16 Teamit Team Structure

#### 3.4.2 Users Hierarchy

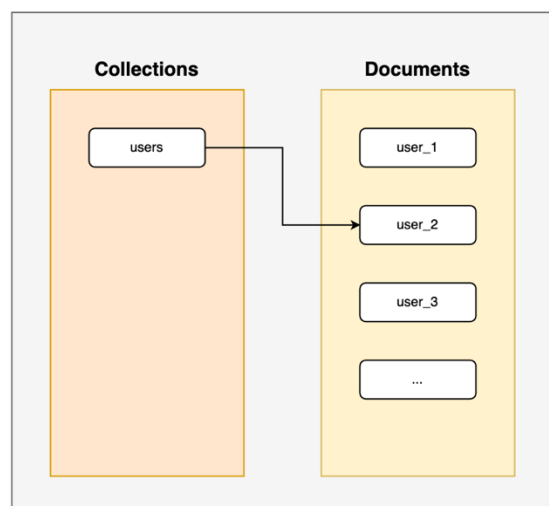


Figure 17 Teamit User Structure

#### 3.4.3 Class Diagram

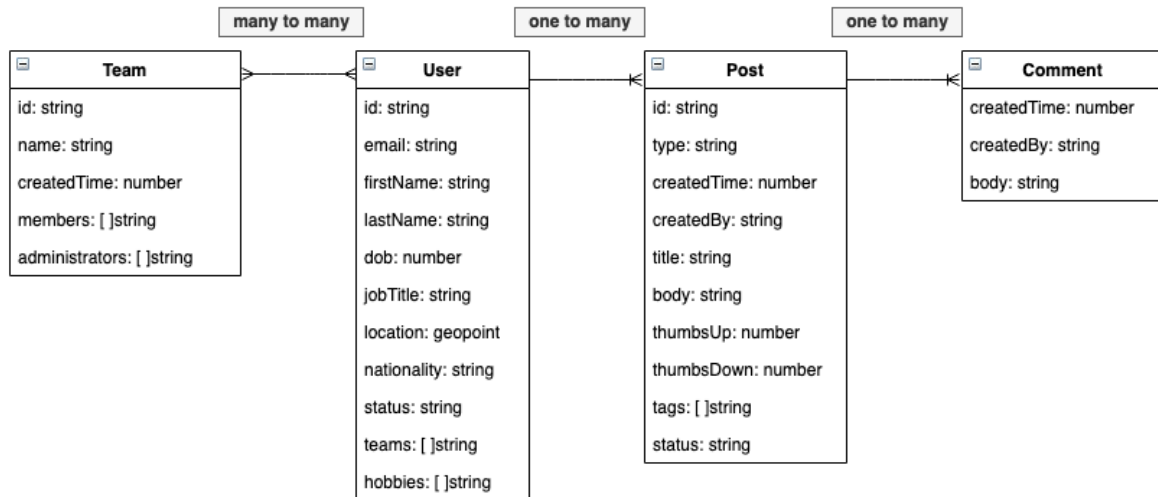


Figure 18 Teamit Class Diagram

### 3.5 User Requirements

The following represents a list of user requirements that **Teamit** must satisfy to be classified production-ready.

#### Administrator Specific

1. An administrator must be able to create teams within the system.
2. An administrator must be able to add users to the teams they currently administer.
3. An administrator must be able to disable users access to a team they administer.
4. An administrator must be able to view metrics relating to a team's performance under their administration.

#### All Users

1. Users must be able to create an account by providing the requested details and successfully fulfilling registration validation.
2. Users must be presented with access to the teams in which they are members.
3. Users must be able to update their profile freely and observe the changes propagate throughout the system.
4. Users must be able to post content to their journal.
5. Users must be able to comment on other team members posts, contributing advice and feedback without restriction.
6. Users must be able to mark a query as resolved such that its state changes within the system.
7. Users must be able to search a catalogue of historic query and posts to narrow in on any information that may present them with value.

## 3.6 Environmental Requirements

While there are no specific environmental requirements per se, there are some resource-intensive processes that require consideration during development, namely the Angular Ivy compiler. Introduced with Angular 9, the Ivy compiler features AOT (Ahead of Time) compilation, which in some environments can consume 100 per cent of the available memory, particularly during frequent live changes as the compiler continually recompiles the application while serving. No specific specifications in terms of resource usage could be located online as the figures would be circumstantial in conjunction with the overall size of the project being compiled. Specifications of the machine used throughout development are highlighted (**Appendix B: Development Machine Specification**)

## 3.7 Usability Requirements

### 3.7.1 Learnability

- The system should be highly intuitive to use, with a low learning curve. Most actions available within the system should be easy to accomplish, even for novice users.
- Features of the system should be easily located, and self-documenting. A user should not have to invest time relearning how to use the system.

### 3.7.2 Presentation

- The UI design should comply with design guidelines and accessibility best practices. Clarity, space and depth should be achieved through the use of shadows, layers and contrasting colours.
- The platforms UX and UI should delight and excite the end-user. Colours should be bright and eye-catching, making the platform easy to navigate and easy for the end-user to extract value through usage.
- The UI should be comprised of components, buttons and calls to action that make their intent explicit to the end-user.
- The UI should guide the user using non-intrusive popups as they attempt to navigate and complete actions available within the platform.
- Features of the system should be conveniently located, allowing for simplistic interactions. A user should not have to invest time in relearning how to use the system.

### 3.7.3 Maintainability

- Users must feel in control of their contributions and profile. Users of the platform should feel empowered to maintain their footprint within the system, keeping their information updated.

### 3.8 Design & Architecture

As demonstrated by the accompanying system architecture overview, several logical steps are executed upon users interactions with the platform. Before access is granted a user must first present their unique credentials to the authentication server. If the authentication server successfully confirms the user to be valid, a token is exchanged which the client can use to log in via a backchannel redirect.

Once an authenticated user is successfully logged in, the system will retrieve information from the main database using the user's unique identifier as the matching criterion. Upon selection of the team metrics page, an external sentiment analysis service running as a serverless cloud function is triggered, returning one of the many metrics used to profile a team performance.

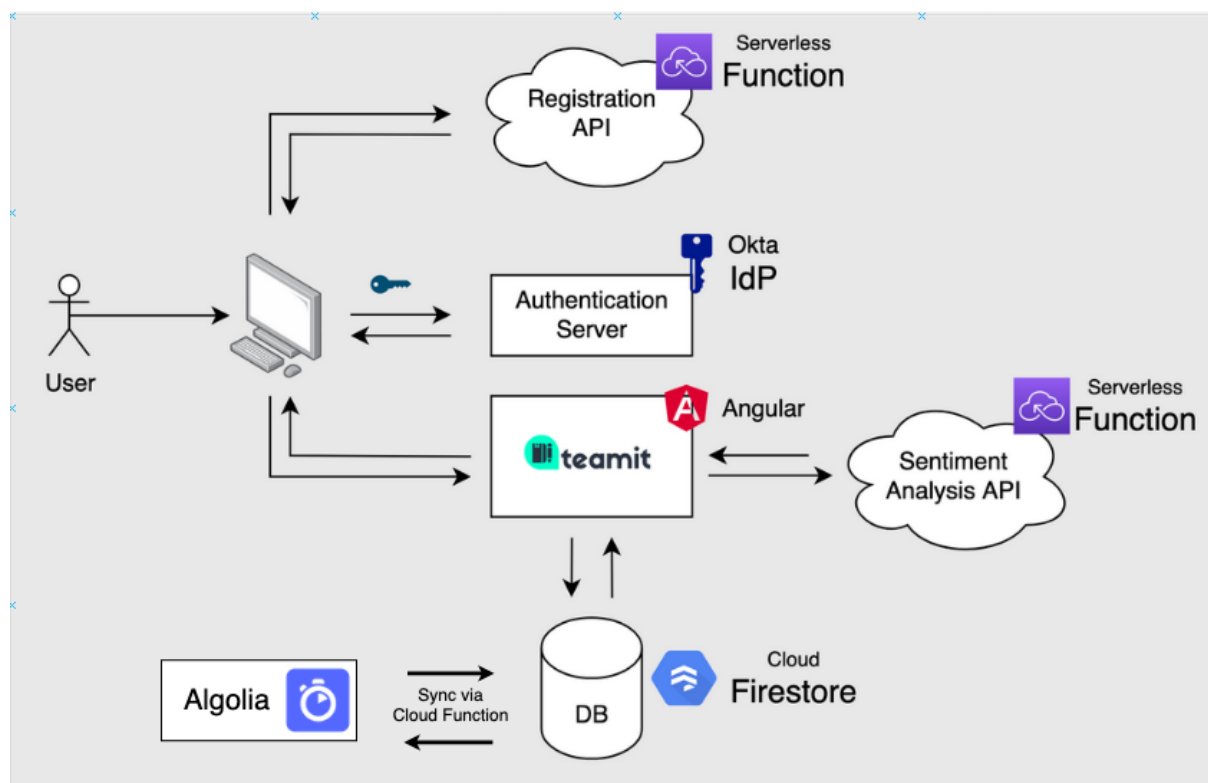


Figure 19 Teamit System Architecture

### 3.9 Implementation

It would be neither practical nor beneficial to discuss every single implementation detail concerning **Teamit's** development. The following section covers only the most time consuming and difficult problems encountered during development.

Development work commenced with two auxiliary RESTful API's that run separately alongside the main **Teamit** UI app, providing platform registration and sentiment analysis functionality. Developing each of these services external to the main **Teamit** UI was an important design decision, one that was taken to achieve separation of concerns amongst the various platform components. This approach was heavily influenced by modern distributed systems shifting towards microservices-based development patterns as opposed to the more traditional and outdated, monolithic approach to software development.

The development of these two API's is split between several Go modules encapsulating each unit of functionality respectively, with common dependencies of both APIs included under the modules directory (**Appendix C: API Module Structure**).

This approach streamlines development efforts allowing for custom implementation of third party libraries in a manner that simplifies the API code base reducing redundant duplication within the code. Configuring a Firebase client for database connectivity is an appropriate example of this pattern. Specifying the required logic to instantiate a Firebase client within the firebase module, it is then possible to simply call this implementation throughout the rest of the project without passing repeated configuration values upon each call (*How To Keep Your Code Dry, 2021*).

### Configuring client (once)

```
// GetClient returns a firestore client to the caller
func GetClient() *firestore.Client {
    ctx := context.Background()
    client, err := firestore.
        NewClient(
            ctx,
            projectID: "teamit-app",
            option.WithCredentialsFile( filename: "../modules/firestore/cred.json"),
        )
    if err != nil {
        _ = fmt.Errorf( format: "firestore client creation error: %v", err)
    }
    return client
}
```

Figure 20 Example Shared Client Logic

### Instantiating Clients (multiple)

Simple creation of client via GetClient(), providing all available functionality

```
// dependency inject firestore client
fsc := fs.GetClient()
fsc.
```

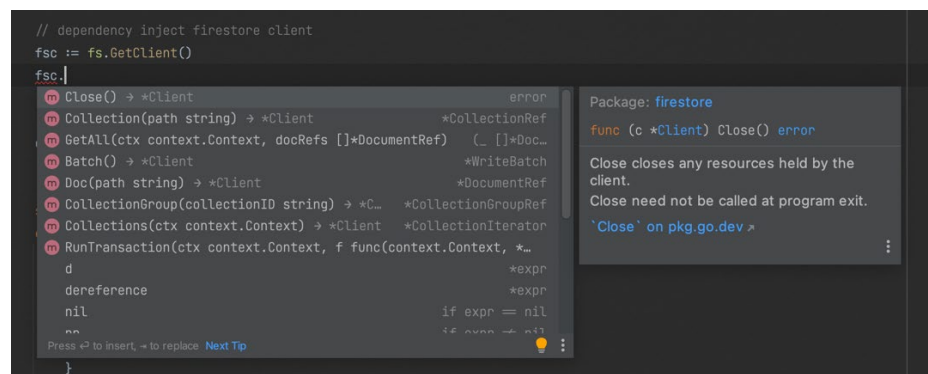


Figure 21 Instantiating Shared Client



### 3.9.1 Registration API

The main motivations behind the development of a separate registration API was to ensure that existing users remain able to interact with the **Teamit** platform, avoiding a situation where any additional development to the registration functionality could potentially take down the whole platform if tightly coupled with the main platform codebase (*Benefits, 2021*).

This separation ensures that already registered users will be less likely to experience service downtime. Also, for security reasons, it's more secure to have a backend service interacting with the authentication provider (Okta) rather than client-side logic handling this directly. If handled client-side, sensitive API credentials are stored within the bundled javascript that gets sent to the users browser, as such is viewable by anyone possessing the know-how to interact with the obfuscated code via the browsers console.

#### Registration Flow

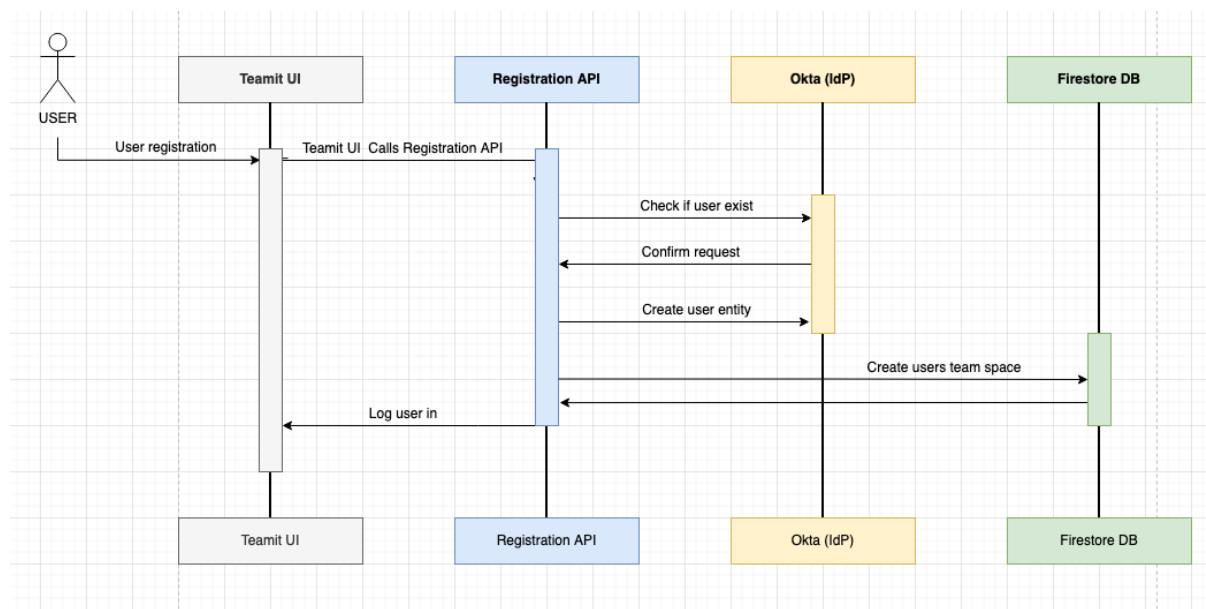


Figure 22 Teamit Registration Sequence

Upon calling the registration API, the incoming HTTP requests are handled via the APIs handler which is responsible for directing the requests based on specified query parameters and the HTTP Verb, also demonstrating dependency injection, passing (fsc Firestore Client) and **Okta client** objects to the respective handler functions. The respective handler function is then responsible for carrying out the requested action and returning a response status code to the client.

```

// logic is based on HTTP request verb
switch req.Method {
case "GET":
    if query["email"] != nil {
        checkUserExist(ctx, oktaClient, res, query["email"][0])
    }
    if query["team"] != nil {
        checkTeamExists(ctx, fsc, res, req)
    }
    //checkUserExist(ctx, oktaClient, res, req)
case "POST":
    createUser(ctx, oktaClient, res, req)
case "PUT":
    updateUser(ctx, oktaClient, res, req)
case "DELETE":
    removeUser(ctx, oktaClient, res, req)
case "OPTIONS":
    return
default:
    res.WriteHeader(http.StatusNotFound)
    fmt.Fprintf(res, a...: "Unknown API endpoint")
    err := fmt.Errorf("method #{req.Method} for #{req.URL} not supported")
    logger.Infof("registration API error: #{err.Error()}")
}
}

```

Figure 23 Registration API Handler

```

func createUser(ctx context.Context, oktaClient *okta.Client, res http.ResponseWriter, req *http.Request) error {

    // read the request body
    decoder := json.NewDecoder(req.Body)
    createRequest := &User{}
    err := decoder.Decode(createRequest)
    if err != nil {
        logger.Errorf("decoding payload error: #{err}")
        return err
    }

    // validate user request
    if err := createRequest.validateParams(); err != nil {
        logger.Error(err)
        res.WriteHeader(http.StatusBadRequest)
        fmt.Fprintf(res, err)
        return err
    }

    // create user
    err, _ = createRequest.createUser(ctx, oktaClient)
    if err != nil {
        logger.Error(err)
        res.WriteHeader(http.StatusInternalServerError)
        fmt.Fprintf(res, err)
        return err
    }
    res.WriteHeader(http.StatusOK)
    fmt.Fprintf(res, fmt.Sprintf("account created for: #{createRequest.Email}"))
    return nil
}

```

Figure 24 User Creation Execution

Strictly speaking, Go is not considered an object-oriented language, however, OOP principles can be modelled using structs (similar to C) which can be compared to classes and receiver methods declared on a struct perform the function of instance methods (Structs Instead of Classes - Object Oriented Programming in Golang, 2021). An example of this is demonstrated, building upon the previous diagram. A request object of type `user` is instantiated from an underlying `user` struct as demonstrated, and the request values are mapped onto the request object.

```
// read the request body
decoder := json.NewDecoder(req.Body)
createRequest := &User{}
err := decoder.Decode(createRequest)
```

Figure 25 User Object Creation and Request value mapping

As `createRequest` is of type `User`, now all of the struct instance methods defined within the `user.go` file become available. One such example is the validation of the previously created user object containing the decoded request parameters. Once a registration request has been validated and all required values have been supplied the remainder of the registration flow commences with multiple subsequent actions being performed to ensure that the specified user name is unique and also that the specified team name is unique within the system, before creating the users entity (**Appendix D: User Request Validation**).

### Encountered Issue

During development, Postman was used extensively for manually testing the registration API endpoints. Triggering the API locally via Postman did not cause any issues until the logic was integrated within the HTTP module of *Teamit's* UI. The subsequent issue related to CORS (Cross-Origin Resource Sharing) and meant that *Teamit's* UI running on port 4200, was unable to communicate with the Registration API running on port 2410.

The solution was two-fold. Firstly, a HTTP OPTIONS verb handler had to be implemented within the handler logic of the API, allowing for pre-flight checks between client and server. Secondly, the outgoing response had to be modified to include additional headers that would grant access to clients from all origins. Long term this solution is not secure and a future iteration of the registration API will address this potential security vulnerability (**Appendix E: CORS Issue Resolution**).

### 3.9.2 Sentiment Analysis API

Sentiment analysis is a very popular method of extracting the emotion or intent that lies behind a particular passage of text. This is a technique that has in recent years become very popular amongst product and marketing professionals, as it provides an ideal opportunity to capture impressions of new products and services as they are released and discussed via social media platforms and other popular communication mediums.

*Teamit* features a suite of performance gathering metrics, allowing team administrators to report the health of their teams. Central to this is sentiment analysis. Much like the previously discussed registration API, it was architecturally conceived that sentiment analysis should not be performed client-side for performance reasons. As the body of content grows around an active team, so too does the resource overheads of computing a sentiment metric. Offloading the computation to a backend

service, available on request results in not only quicker load time but also allows for server-side caching, which in turn further increases performance. Implementation began with the APIs handler which receives the client request and routes accordingly. Initially, the intended strategy was to develop an endpoint that would compute an entire teams sentiment, period. However, as this piece of functionality came to a close, it was apparent that individual sentiment would be a very useful metric also. The API itself connects directly to the database, pulling a collection of data mirroring the team name specified within the client request. Once a collection reference is establish an iterator is responsible for crawling thought the teams data before passing it on for sentiment analysis.

```
// team collection path
collection := fmt.Sprintf("teams/#{team}/posts")

// fetch posts and all comments
iter := fsc.Collection(collection).Documents(ctx)
defer iter.Stop()
for {
    post, err := iter.Next()
    if err == iterator.Done {
        break
    }
    if err != nil {
        logger.Errorf("Reading collection error: #{err}")
    }

    // save posts to interactions array
    tmp := post.Data()["body"].(string)
    interactions = append(interactions, tmp)

    // get post comments
    docID := post.Ref.ID
    path := fmt.Sprintf("teams/#{team}/posts/#{docID}/comments")
    iter2 := fsc.Collection(path).Documents(ctx)
    comments, _ := iter2.GetAll()
    for _, eachReply := range comments {
        tmp2 := eachReply.Data()["body"].(string)
        interactions = append(interactions, tmp2)
    }
}

// compute sentiment score
for _, eachInteraction := range interactions {
    parsedText := sentitext.Parse(eachInteraction, lexicon.DefaultLexicon)
    sentiment := sentitext.PolarityScore(parsedText)

    // calculate average sentiment
    totalSentiment = totalSentiment + sentiment.Compound
}

sentimentScore = totalSentiment / float64(len(interactions))
logger.Infof("Team #{team} scored #{sentimentScore} during sentiment analysis")
return sentimentScore
```

Figure 26 Triggering Analysis

Individual sentiment follows an identical approach with the minor difference being an additional query clause specifying a particular user in addition to the team name parameter. Initial sentiment analysis was performed using a package called Sentiment, which is a pre-trained machine learning sentiment analyser written in Go which uses a sizeable collection of IMDB reviews as its source of reference

material. However, upon testing this library with sample input the resulting output was inconsistent and questionable at best. Providing input that contained commonly regarded negatives resulted in output that depicted sentiment as positive. This resulted in switching from Sentiment as a source of analytical logic to a package called *Vadar-go*. *Vadar-go* takes a more simplistic approach to weighing sentiment, assigning a numerical value to each word based on a lexicon of commonly used English words (*SENTIMENTAL ANALYSIS USING VADER, 2021*).

### **API Conclusions**

The Registration and Sentiment API's combined added approximately 700-850 additional lines of code to the project, and required a considerable amount of planning, learning, debugging and frustration, however, with perseverance, eventually paid off. As highlighted, several rather time-consuming issues required serious head-scratching during the first few months of *Teamit's* development.

Retrospectively, a reasonable alternative to the implemented approach would have been to leverage Firebase's authentication backend which would remove Okta and the Registration API from the equation. Concerning the Sentiment analysis API, this too could be further simplified by computing a sentiment score client-side, reducing the amount of work involved, additionally removing the dependency on a running sentiment service.

However, *Teamit* as a product, was architecturally designed and built using implementation strategies that one would expect to find while building similar production-grade systems in the 'real world' for lack of a better term. As a computing student specialising in Software Development, difficulty and ingenuity garnered a tremendous opportunity to learn and grow in parallel with the platform itself which is worth every bit as much as the working artifacts themselves.

### 3.9.3 Angular routing

**Teamit** is a SPA (Single Page Application), such that the user interface is comprised of several dynamic components, each rendered conditionally, on demand, based on the desired outcome of the user. The accompanying truncated image demonstrate a number of interesting configurations specific **Teamit's** routing, such as authentication guards prohibiting specific path access, call back component for handling Okta registration integration and finally redirects.

```
const CALLBACK_PATH = 'login/callback';

const appRoutes: Routes = [
  // Landing Page
  {
    path: '',
    component: CreateAccountComponent
  },
  // Auth Routes
  {
    path: 'login',
    redirectTo: '/home',
    pathMatch: 'full'
  },
  {
    path: 'logout',
    component: LogoutComponent,
  },
  {
    path: CALLBACK_PATH,
    component: OktaCallbackComponent
  },
  {
    path: 'home',
    component: HomeComponent,
    canActivate: [OktaAuthGuard],
  },
],
```

Figure 27 Angular Routing

### 3.9.4 Navigation and Behaviour Subjects

Upon building the users home dashboard an issue arose whereby the selected value of one component was required by another second component before data in the second neighbouring component could be requested and rendered. However, in keeping with maintainability and extensibility, topics discussed earlier, the components of the application were broken down into the smallest possible units of functionality containing only the minimum controller and view logic required by each component to fulfil its duty. The following diagram demonstrates the issue in further detail.

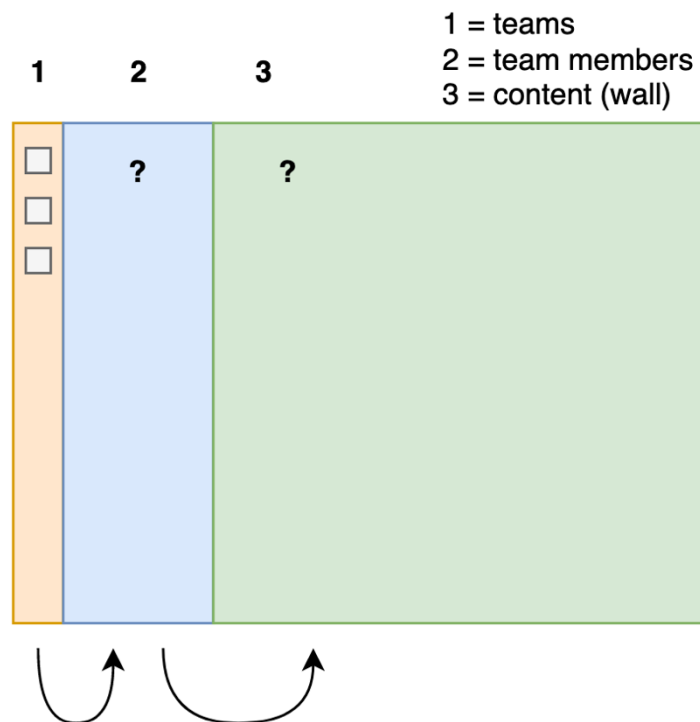


Figure 28 Passing Values dynamically between components

As shown, the selected value from the teams component is not readily available to subsequent components (team members, wall) but is required by the resulting chain of requests that ensues. Depending on which team is selected, dynamic team members are pulled from the database. Then, depending on the selected team member, content specific to that user is fetched from the database and displayed. An easier approach to this would have been to fetch all data and perform conditional rendering client-side across the whole dataset. However, doing so is not scalable requiring far too many redundant database reads, which over time would slow down the front end-user experience and be an inefficient use of database connection capacity.

The solution was observables, more specifically Behaviour Subjects, a construct of RxJS (**Appendix F: Behaviour Subjects**). Behaviour subjects programmatically support capturing and broadcasting specific values over-time in response to a given event. In this case selection of a team triggers the team name to be broadcast, making it available for other components to subscribe to, and use. Same applies to the team-members component, broadcasting the selected team-members id, in turn providing the required value needed to fetch that selected users posts data. Code sample included (**Appendix G: Value Passing Between Components**).

### 3.9.5 Search

Implementing real-time search functionality within **Teamit** was not as straightforward as initially expected. Firestore does not support real-time search indexes, which was disappointing. Google is better equipped than most companies to provide a product that supports this feature.

The recommended solution was to use a third-party cloud service, Algolia. Setting up an account with Algolia was straightforward. Creating the index that would be used also took no more than a few clicks. The complexity arose however concerning populating the Algolia index with data written to Firestore. Initially, the plan was to leverage the JavaScript SDK that provides support for programmatically

connecting to an Algolia servers. The plan was to write to both Firestore and Algolia in parallel upon user post creation. Before long it became apparent that the bundled Algolia JavaScript could not be used within a TypeScript component for reasons that are still not fully understood.

Following the recommended approach a cloud function was required that would trigger in response to Firestore document changes. Once triggered, the newly persisted data is read from the Firestore and pushed to Algolia. Doing so ensures that the searchable data jointly persisted to Algolia is always in sync with the data persisted to Firestore. A cloud function is running on Google Cloud Platform and is responsible for handling this syncing operation.

```
const APP_ID = functions.config().algolia.app;
const ADMIN_KEY = functions.config().algolia.key;
const ALGOLIA_INDEX_NAME = 'teamit';
const client = algoliasearch(APP_ID, ADMIN_KEY);

// Update the search index every time a blog post is written.
exports.onPostCreated = functions
  .region( regions: 'europe-west2')
  .firestore.document( path: 'teams/{teamId}/posts/{postId}')
  .onCreate( handler: (snap : QueryDocumentSnapshot , context : EventContext ) => {
    // Get the post document
    const post = snap.data();
    // Add an 'objectID' field which Algolia requires
    post.objectID = context.params.postId;
    console.log(post)
    // Write to the algolia index
    const index = client.initIndex(ALGOLIA_INDEX_NAME);
    return index.saveObject(post);
  });
```

*Figure 29 Cloud Function - Algolia Sync*

It is important to note the objectID of the Firestore document is used to configuring the object reference in Algolia. Doing so allows for subsequent targeted actions across the Algolia index such as updates and deletes in response to the other actions within Firestore. Images of the search UI are featured in an upcoming section.

### 3.9.6 Metric Computation

A key component of **Teamit's** value proposition relates to metrics generated by quantifying certain interactions within the system. Once the team-specific data has been aggregated, **Teamit** displays a variety of performance measurements specific to the selected team and its members. Current platform implementation supports the following metric types, with scope for many more with future iterations. Both Individual Sentiment and Team Sentiment have already been discussed. That leaves the remaining three in addition to sentiment analysis, which were implemented as follows.



- Query Collaboration
- Contributions
- Tag Analysis

**Query Collaboration** represents the ratio of open to closed queries within the system. Upon creating a Post of type Query, the system by default marks the content as open (unresolved). The time at which the query was opened is also noted. See more on query creation (**Appendix H: Query Creation**).

Upon opening the metrics section within the UI, several processes are executed, gathering data from Firestore, and computing the required aggregations before passing the resulting arrays to the template for rendering. Charts and graphs are provided by PrimeNG which under the hood implements Chart.js.

```
fetchQueryCollaborationMetrics(team) {

  let openCount = 0;
  let closedCount = 0;

  this.metricsService.getQueryCollaboration(team).subscribe( next: (resp: IPost[]) => {
    resp.forEach(post => {
      if (post.status === 'open') {
        openCount++;
      } else if (post.status === 'closed') {
        closedCount++;
      }
    });
    this.queryCollaborationData = {
      labels: [
        'open',
        'closed',
      ],
      datasets: [
        {
          data: [openCount, closedCount],
          backgroundColor: [
            '#db2955',
            '#00ffcd',
          ],
          hoverBackgroundColor: [
            '#db2955',
            '#00ffcd',
          ]
        }
      ]
    };
  });
}
```

Figure 30 Query Collaboration Template Model

Tag analysis and Collaboration metrics follow a very similar set of steps whereby the data in question is retrieved, aggregated and finally passed to the HTML template for rendering. Similar implementation details concerning the remaining metrics are referenceable from the appendices with minor difference relating to dynamic colouring of team members based on variable member count per team. (**Appendix I: Tag and Collaboration Analysis**)

### 3.9.7 Content Rendering

During the implementation of an early content creation piece of functionality, an issue arose whereby it was not apparent how the saved content should be re-rendered upon loading. As a result of the WYSIWYG Editor used, content is saved in the following format.

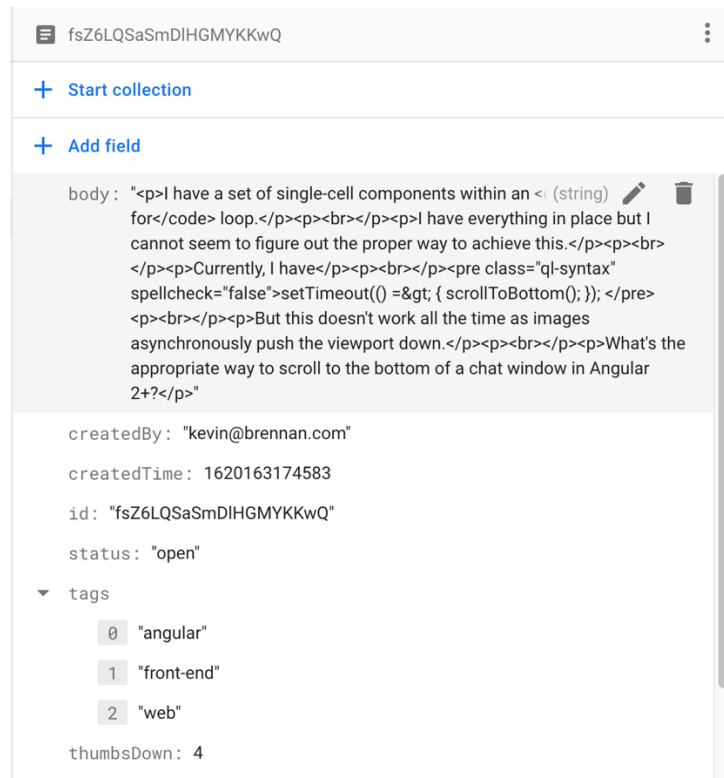


Figure 31 Persisted Body Content

Body data is stored as plain HTML. It is PrimeNG's editor module that provides the editor's functionality, however, their documentation regrettably fails to clearly explain the process of how to handle the resulting data string upon re-rendering. With no clue how to work with the formatted data, it was easier to bypass PrimeNG's documentation completely and go directly to source documentation for the underlying editor dependency of PrimeNG's Editor, Quill.js. Upon reviewing the documentation it was noted that a special component made available by the Quill module directly provided a custom HTML element with several directives enabling the HTML unwrapping into a renderable, plain-text counterpart.

```
<quill-view-html [content]="post.body"></quill-view-html>
```

Figure 32 HTML Rendering Element

## 3.10 Graphical User Interface (GUI)

### 3.10.1 Colour Palette

**Teamit's** target audience is comprised of a variety of end-users from perspectives such as age demographic, interests and social background to name a few. That said, paying particular attention to something simple such as colour plays an important role in ensuring the application is widely accessible and appealing to use. **Teamit's** colour schema has been created from scratch using a range of striking colours from several different complementary colour palettes.

<https://coolors.co/293241-00ffc9-829399-db2955-3993dd>

The following online resource was used to assemble the colour palette, with the respective HEX values being imported and parameterised in **Teamit's** root CSS style sheet.

```
/* SCSS HEX */
$gunmetal: #293241ff;
$sea-green-crayola: #00ffc9ff;
$cadet-grey: #829399ff;
$raspberry: #db2955ff;
$tufts-blue: #3993ddff;
$hover_color: #1c2830;
```

Figure 33 Teamit Colour Palette

### 3.10.2 Logo Design

Alongside a striking colour palette, a professional logo is crucially important when striving to achieve brand identity and product recognition. **Teamit's** logo has been carefully designed from scratch with specific attention and care given to the selected font-face and layout configuration. Canva, an online media creation platform was leveraged when designing **Teamit's** logo. Different configuration of the logo exists, supporting placement in a variety of online locations and settings.



Figure 34 Teamit Logo Design

### 3.10.3 Registration

There are several possible scenarios whereby a user navigates to **Teamit's** registration page. In each case, whether on purpose or otherwise, a lasting first impression will be formed based on this initial interaction with the platform. It's vitally important to maintain user satisfaction by providing an onboarding flow that is seamless and intuitive. As demonstrated the user will be prompted for several required values before account creation is executed. Each field features validation such that upon

entering an invalid value or simply omitting to enter a value, the user will be prompted to resolve any perceived errors before continuing.

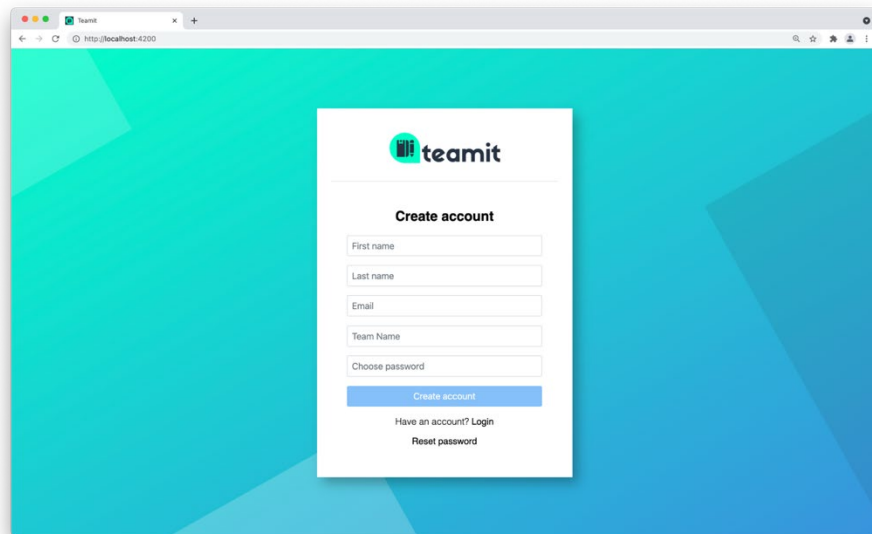


Figure 35 Teamit Registration

Most of the technical implementation details concerning the registration flow, specifically regarding backend operation, along with several technical challenges have already been discussed at length during a previous section of the report. In addition to the custom registration API, a reactive forms module included as part of the Angular framework serves as the basis for collecting each of the details provided by the user, packaging the payload that is sent via HTTP to the registration API. Upon choosing a password, the registration component will present the user with feedback regarding the strength of the provided password. While not strictly an essential feature, it does serve as an additional piece of functionality that further enforces the feeling of quality and attention to detail.

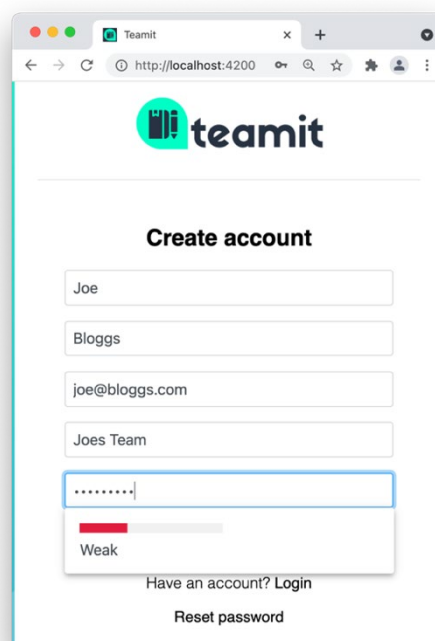


Figure 36 Registration Validation

### 3.10.4 App Navigation

Many layers of navigation exist within **Teamit**.

#### Team Navigation

A list of given user teams are displayed within the teams navigation menu, featured to the far left of data returned is specific to the logged user, as is the case for all data rendering components.

#### Team Member Navigation

Depending on the users selection from the team navigation menu, a list of team specific members are displayed, along with the team title.

#### Main Navigation

Provides a user with the ability to navigate between the applications various different pages.

#### Content/Comment Navigation

Features several clickable icons, allowing a user to view comments or modify/deleted content.

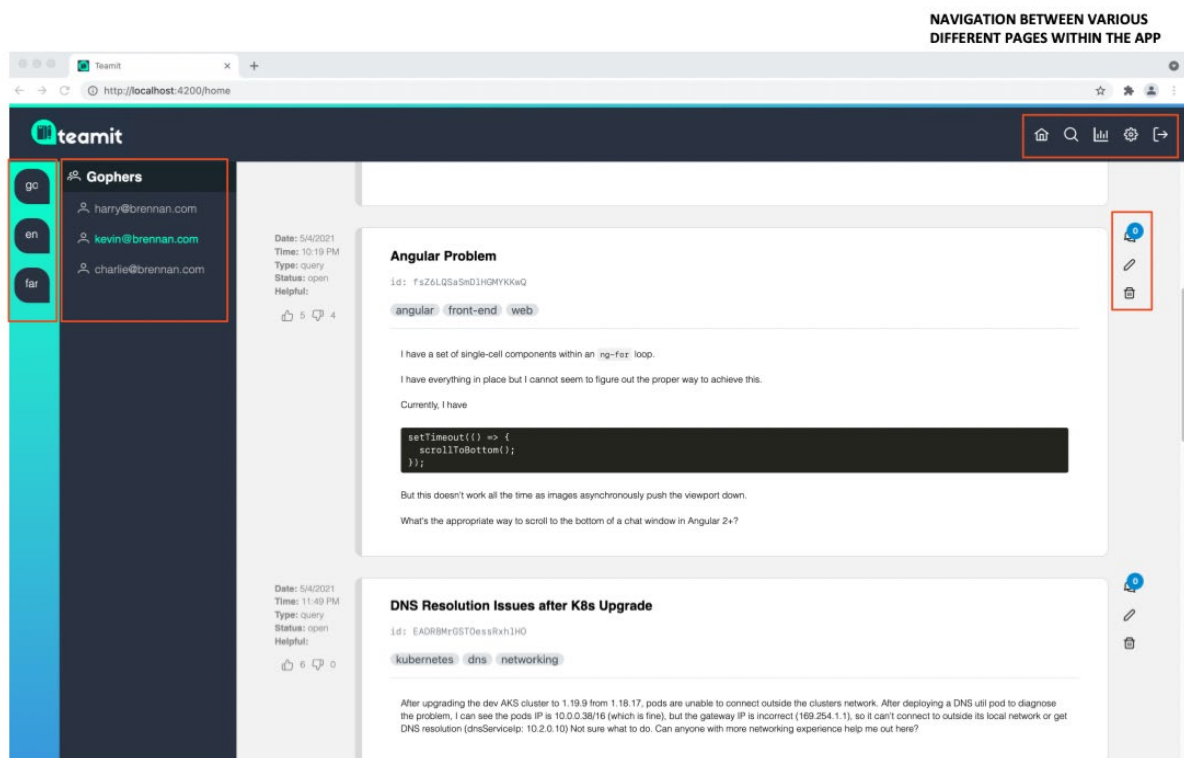


Figure 37 Teamit Navigation

### 3.10.5 Users Home Dashboard (Journal)

**Teamit** provides each team member with a personal space that is fully operated and moderated by them. Whether seeking help or simply posting an interesting article for their colleagues to enjoy, this is where a user will spend most of their time while interacting with **Teamit**.

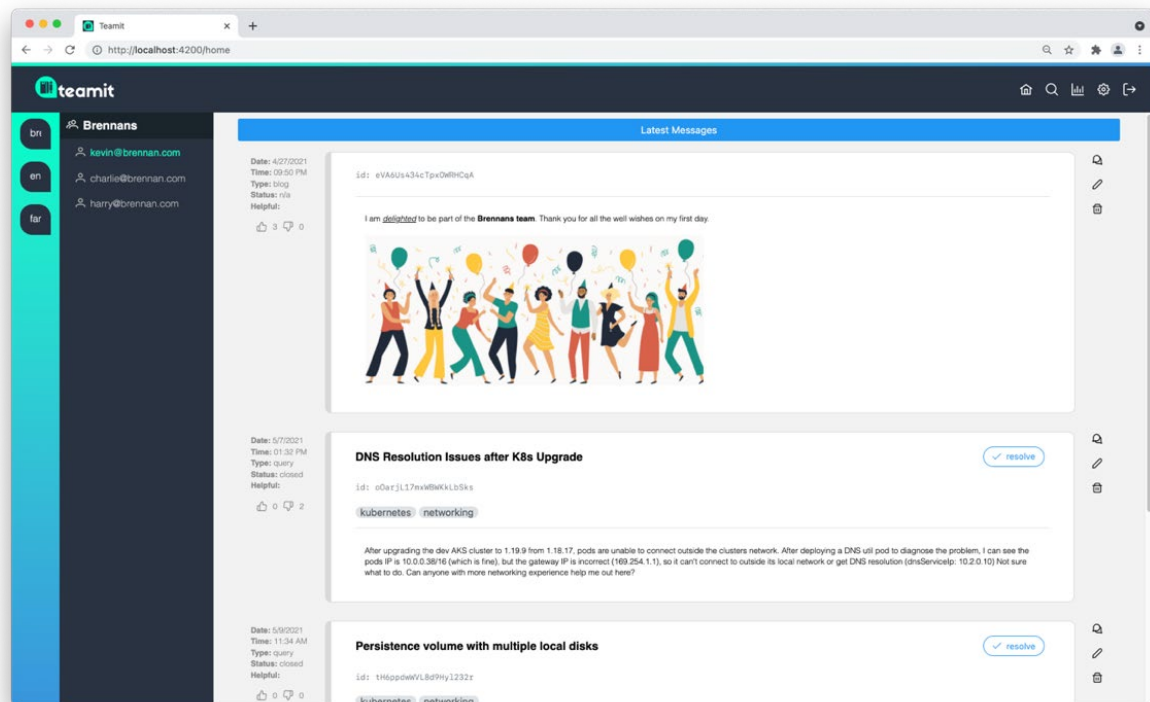


Figure 38 Teamit User Dashboard

### 3.10.6 Admin Dashboard

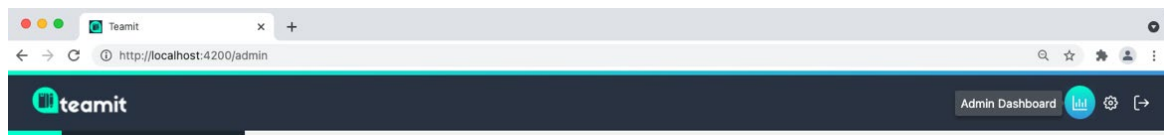


Figure 39 Teamit Main Navigation

The administration dashboard is accessible via the main menu icons located in the top navigation bar. Tooltips provide helpful guidance. Upon choosing to visit the admin section, several options are available, namely *Team Metrics* and *Team Management*. From here a user can navigate to the metrics page and select from a list populated with teams they administer.

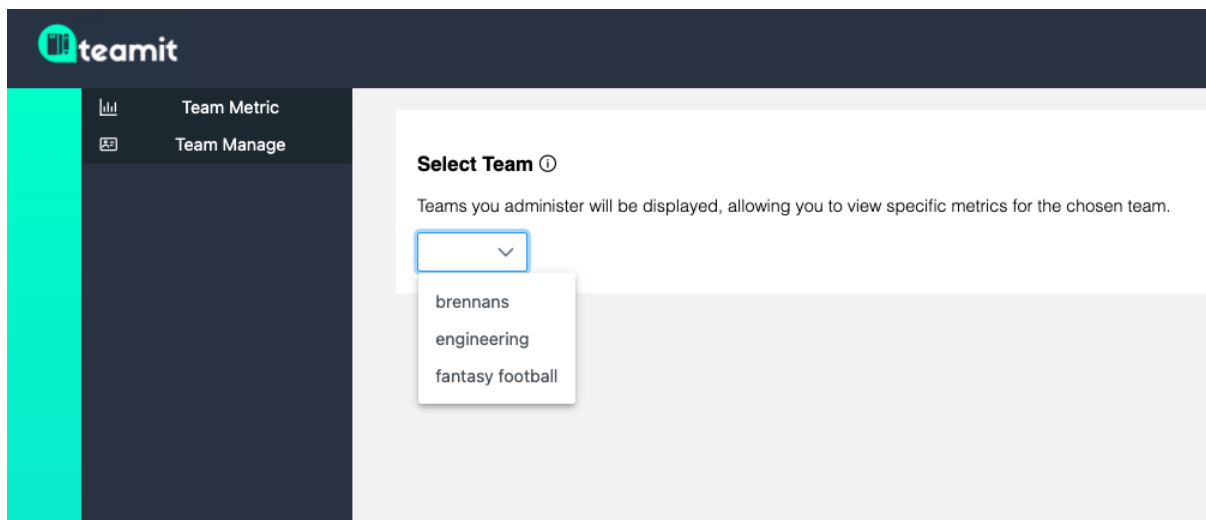


Figure 40 Metric, Management and Team selection

The upcoming image demonstrates the data available via the metrics dashboard. It should by now be possible to fully appreciate the main motivations behind **Teamit** as most of what has been discussed previously has been leading up to this point. Currently, the metrics available are as follows;

- Query Collaboration
- Contributions
- Individual Sentiment
- Team Sentiment
- Tag Analysis

Importantly, and excitingly, the demonstrated metrics only begin to scratch the surface of what is possible. Many other ideas were not fully explored or implemented due to time constraints and submission deadlines. However, an upcoming version of the project will support the ability to gather metrics per specified period, further supporting team administrators with historic access to performance data, which can be useful to retrospectively report on productivity opening up a whole additional set of benefits.

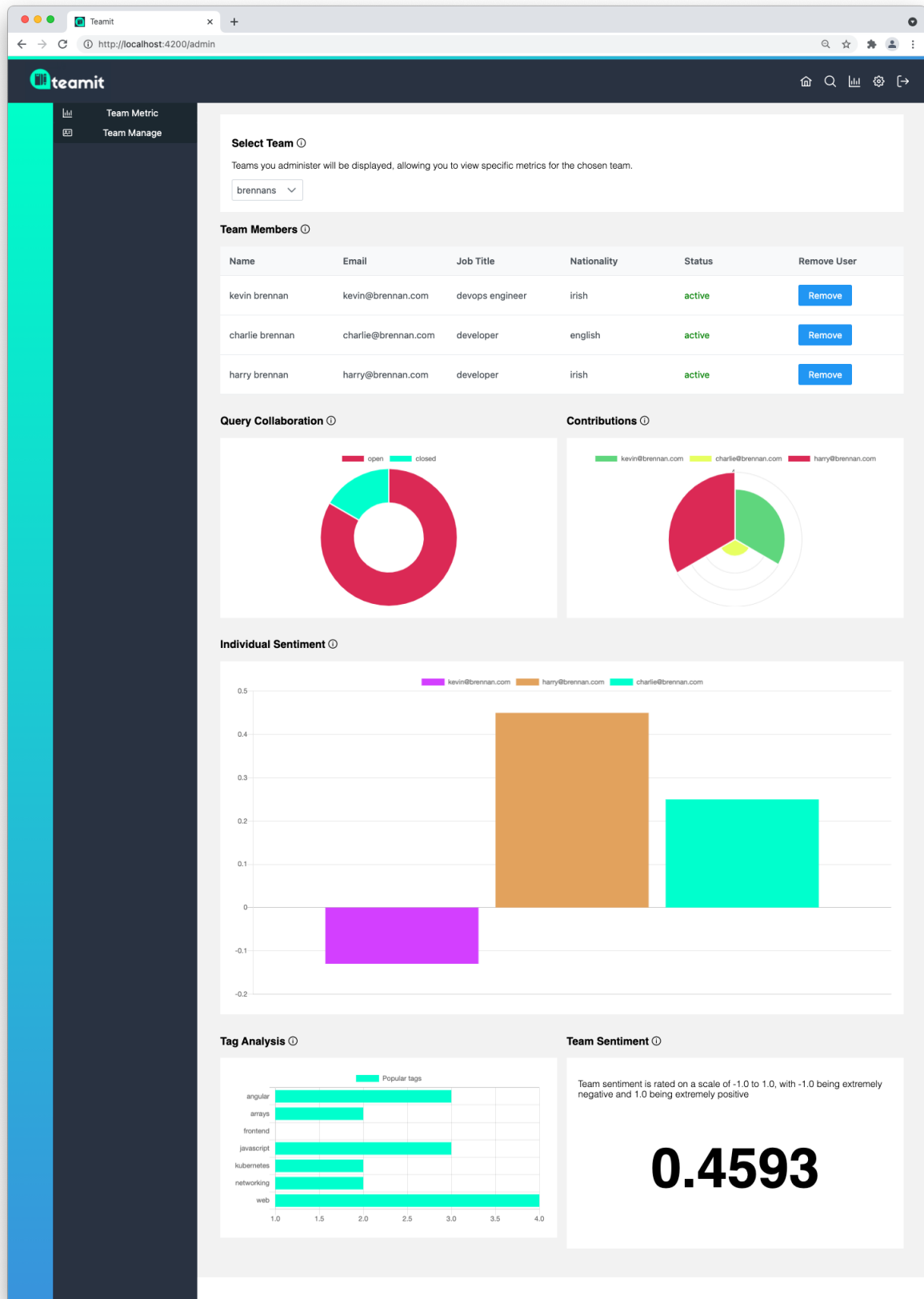


Figure 41 Metrics Dashboard



### 3.10.7 Search Dashboard

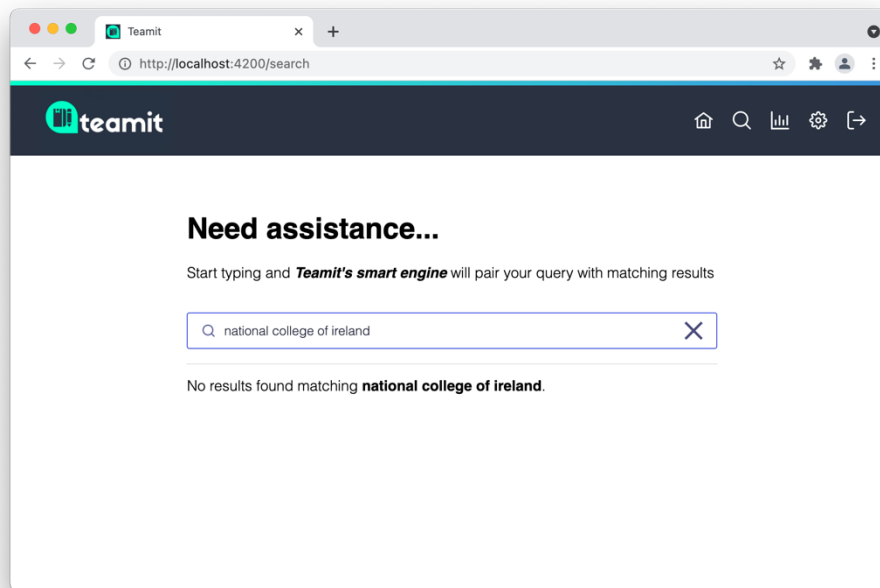


Figure 42 Search Functionality - waiting for user input

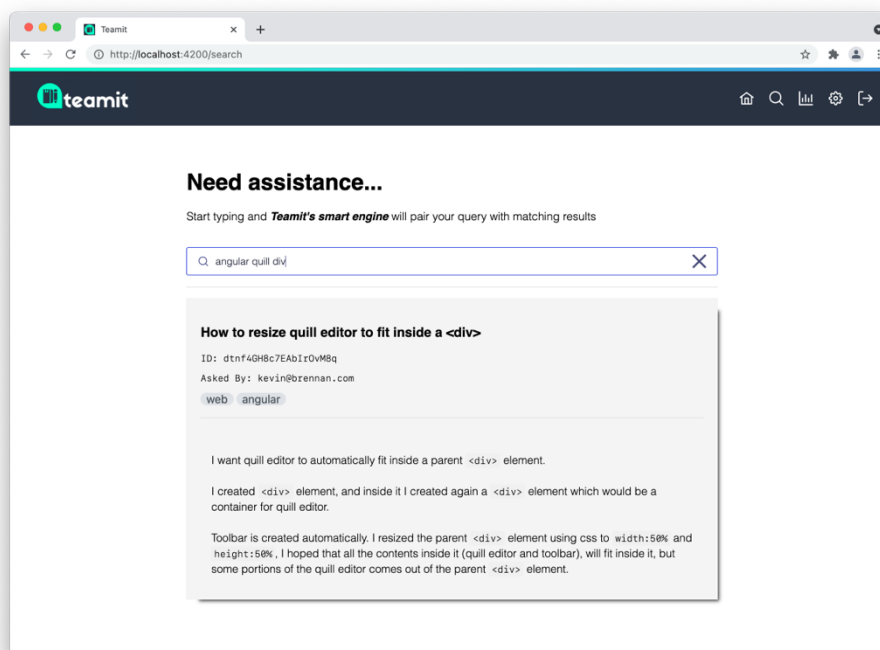


Figure 43 Search Functionality - response to specific search term

### 3.10.8 Content Creation

**Teamit** features a WYSIWYG style editor which provides the end-user with a range of functionality, allowing them to author text in a way that feels natural along with remaining visually appealing. The editor itself is an implementation of PrimeNGs editor module, which in turn implements Quill.js, the underlying library responsible for providing all of the editor's functionality. To recap, PrimeNG is a component-based framework used extensively throughout the project for the implementation of a whole range of various UI based functionality. As demonstrated in the accompanying images, the editor is visually appealing, providing an extensive range of editing power to the user.

The main composition component consists of a parent component that has several nested child components, providing a unique editing experience for each of the platforms interaction methods (Blog, Query, To-do). Importantly, each of these composition models is intended to store differing content based on the users intent.

- The blog allows a user to record their progress throughout the day. Importantly, blogs do not require further action from other team members, therefore it is not possible to resolve blog-style posts, nor do they require a heading, tags and so forth.
- The query allows a team member to communicate blocking issues with the rest of the team and request that other team members assist them.
- To-do items allow a user to track their work items. A future version of the platform will support to-do items per individual. Additionally, to-do items would feature within the overall team metrics, allowing team administrators to gauge the speed at which members of a given team are progressing through their work items.

The screenshot displays a web form titled 'TAB SELECTION COMPONENT'. At the top, there are three tabs: 'Blog', 'Query' (which is selected and highlighted in blue), and 'Todo (coming soon)'. Below the tabs, the form is labeled 'DYNAMICALLY RENDERED'. The form contains several input fields: a 'Title (summary)' field, a 'Tags' field, and a 'Editor' field. The 'Editor' field is a WYSIWYG editor with a toolbar containing various formatting options like bold, italic, underline, text color, background color, bulleted list, numbered list, link, unlink, and source. At the bottom of the form, there is a blue button labeled 'Request help' and a red button labeled 'Clear'.

Figure 44 Content Creation Component

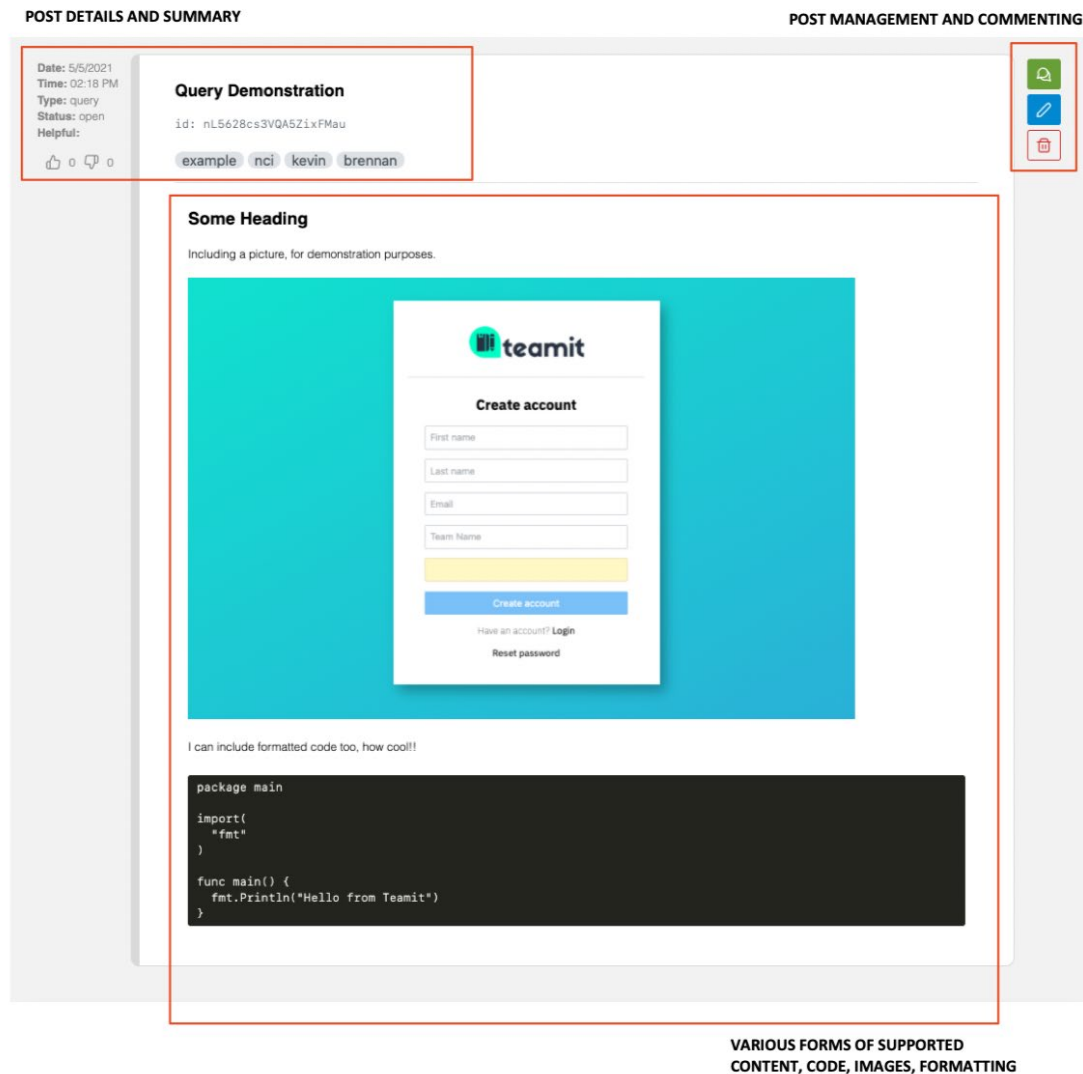


Figure 45 Content Rendering

### 3.10.9 Commenting

Facilitation of cross-team collaboration is important functionality to include within any communication style application, and is a core component of **Teamit's** functionality. **Teamit** provides users with a full suite of communication tooling which supports initial posting of queries, additionally facilitating multithreaded communications amongst interested parties.

Comments are toggled upon request and appear via a slide out window. This was an important design consideration preventing the main screen from becoming overly saturated with responses to a given post, potentially distracting from the main content itself.

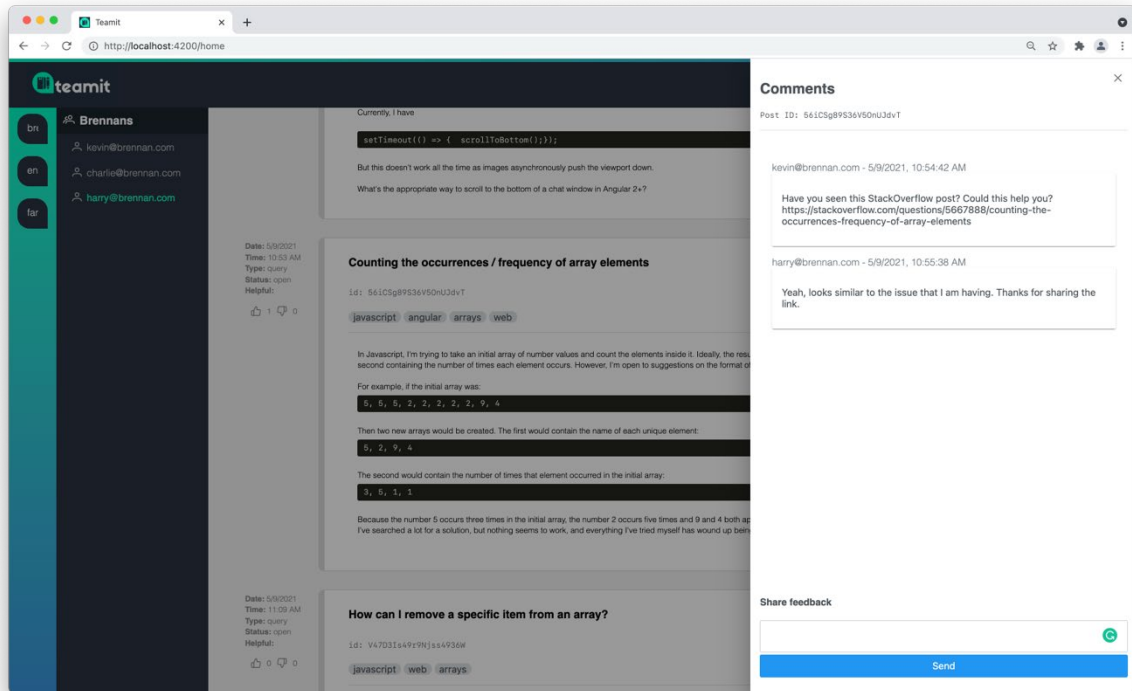


Figure 46 Slide out Comment Drawer

### 3.10.10 Tooltips

**Teamit** uses informational tooltips extensively throughout, with the sole purpose of improving a user's comprehension and understanding of how to interact with the system.

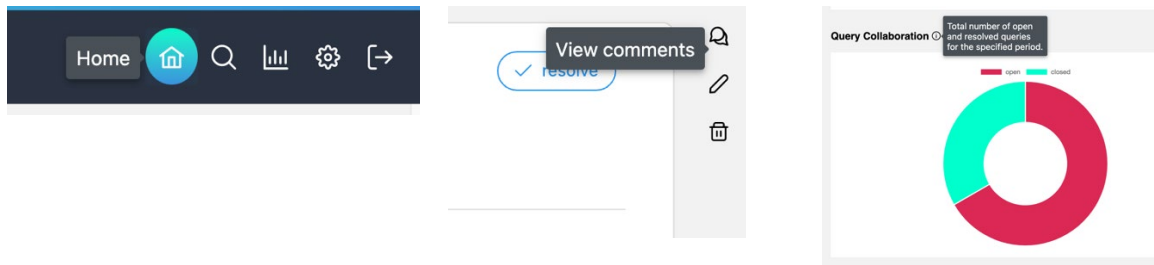
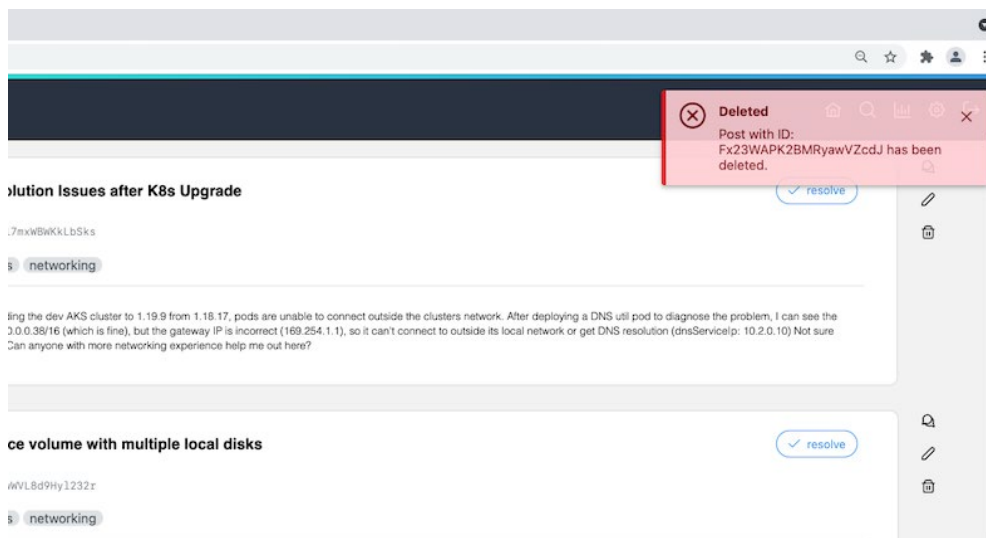


Figure 47 Tooltip Examples

### 3.10.11 Popup Notifications

**Teamit** uses a range of informational popups which for the most part are triggered in response to user actions, such as content creation, modification, and deletion.



## 3.11 Testing

### 3.11.1 Manual Testing

Manually testing the application remained a steady constant throughout **Teamit's** lifecycle. As each feature was developed, a range of scenarios was tested, providing the opportunity to document the outcomes of each, allowing for comparison with the expected outcome. At each stage, a full set of tests were performed to ensure that each item of functionality behaved as intended. Included below is a snapshot of the testing matrix used, along with a hosted link allowing for full access to view the criteria and results of each test case.

[https://studentncirl-my.sharepoint.com/:x:/g/personal/x16149823\\_student\\_ncirl\\_ie/EXBR-nTSFJ9NohZdVFypZx4Bt0cZ2rF7IkMzBJLFqP41mw?e=9k1AJ1](https://studentncirl-my.sharepoint.com/:x:/g/personal/x16149823_student_ncirl_ie/EXBR-nTSFJ9NohZdVFypZx4Bt0cZ2rF7IkMzBJLFqP41mw?e=9k1AJ1)

	A	B	C	D	E	F	G	H	I
10					TC-0009	Sentiment Analysis API			Kevin Bren
11					TC-0010				Kevin Bren
12					TC-0011				Kevin Bren
13					TC-0012		Register for a new Teamit account		Kevin Bren
14					TC-0013	User Interface	Sign in to an existing account	24/04/21	Kevin Bren
					TC-0014	User Interface	Delete my user account		Kevin Bren
15									
16					TC-0015	User Interface	Logout of currently active account	24/04/21	Kevin Bren
					TC-0016	User Interface	Navigate to user settings dashboard	24/04/21	Kevin Bren
17									
					TC-0017	User Interface	Navigate to admin metrics dashboard	24/04/21	Kevin Bren
18									
					TC-0018	User Interface	Navigate to searchable content dashboard	24/04/21	Kevin Bren
19									
					TC-0019	User Interface	Create a blog entry within the system.	24/04/21	Kevin Bren
20									

Figure 48 Testing Matrix

### 3.11.2 API Testing

#### Postman

Both Registration and Sentiment Analysis API's were heavily tested throughout development. Testing consisted of using Postman to trigger the respective endpoints, making it possible to observe the response bodies, headers and status codes. Doing so allowed proper error handling to be included in response to errors revealed through testing. For ease of use and efficiency a set of collections was created in Postman containing the various accepted methods for each API's respective endpoints.

Demonstrated in the accompanying image, currently being tested is the Sentiment Analysis API. It can be seen that the API is properly versioned, with two parameters being passed. These are the required parameters when performing individual sentiment analysis.

<http://localhost:2411/api/v1/sentiment?team=brennans&user=charlie@brennan.com>

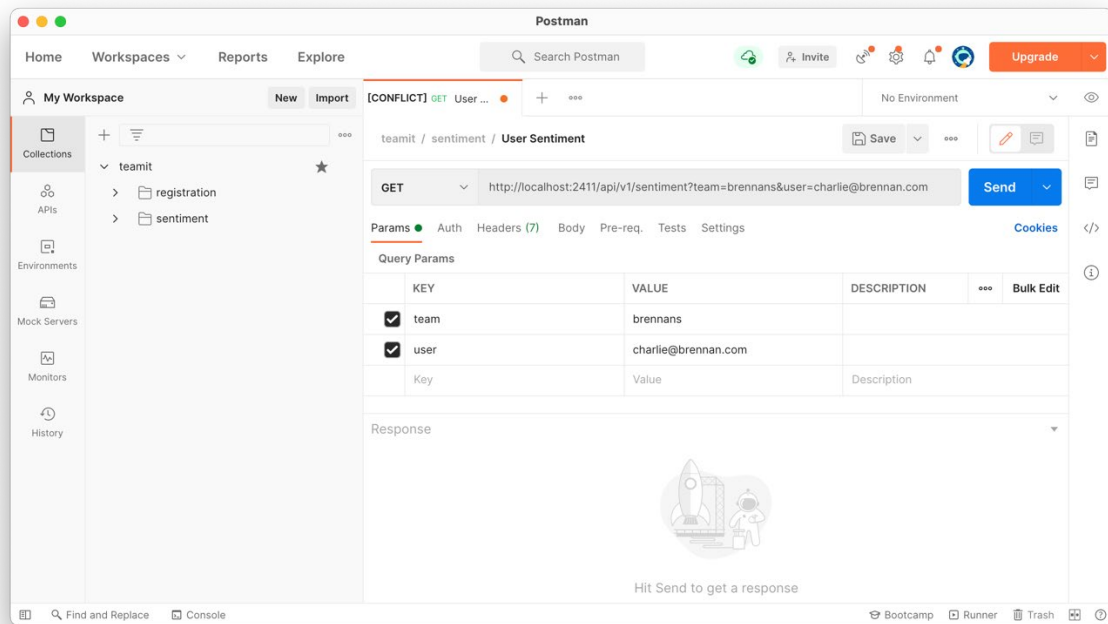


Figure 49 Testing with Postman

## Unit Testing

Go features a suite of tools for unit testing that come bundled with the language's standard library. Unit tests were written for several of the API's handler and helper functions to ensure that the expected output was returned. One such example includes unit testing the `GeneratePassword` function that is used within the system.

```
func TestGeneratePassword(t *testing.T) {

    got := GeneratePassword(20)
    want := len(got)

    if len(got) != want {
        t.Errorf("got %q want %q", got, want)
    }
}
```

Figure 50 Go Unit Test Example

```
~/Hidden/teamit/modules/util | on user-profiles !3
} go test -v
=== RUN   TestGeneratePassword
--- PASS: TestGeneratePassword (0.00s)
PASS
ok      github.com/kevinbrennanio/teamit/modules/util    0.071s
```

Figure 51 Triggering Test with Output

### 3.11.3 UI Testing

Angular comes bundled with Jasmine, which is a behaviour-driven testing framework for JavaScript. Each component is tested to ensure they are correctly instantiated and created. The tests can be run using the 'ng test' command. More emphasis was placed on manually testing the UI components for several reasons. User Interface testing can be very complex. Many of the benefits of UI testing are realised when a project has multiple contributors. **Teamit**, for now at least does not, so managing and maintaining a suite of manual tests was sufficient in addition to a basic level of component validations using Jasmine. Test can be run individually per component as demonstrated below, or in a batches.



Figure 52 Jasmine Sample Output

### 3.11.4 Usability Testing

Throughout development and post-development various individuals were asked to interface with the platform. On several occasions, valuable feedback was onboarded and implement within the project. One such request was to include tooltips on the metrics dashboard, that would help clarify what each data point represented. Implementation is demonstrated in the accompanying image.



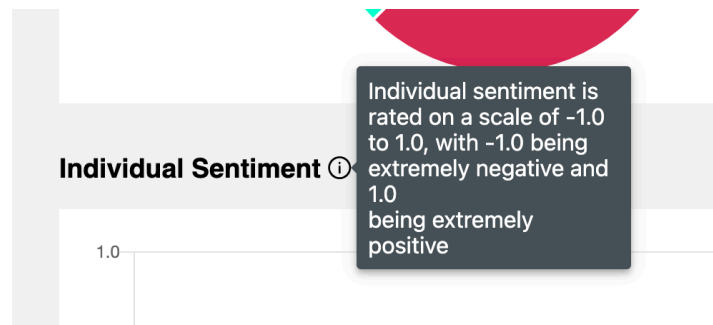


Figure 53 Example Implementation Usability Feedback

### 3.12 Evaluation

Several impartial actors (six in total) asked to provide their feedback and impressions of the platform once the implementation phase had concluded. Each of the participants is currently actively employed, and working in some professional capacity that involves remote working as a consequence of the Covid-19 pandemic. To summarise, the platform was very well received, with each user demonstrating an interest in the metrics collection, stating that it is a very nice idea, currently not offered in pure communication style platforms. It was also noted that the UI/UX of the platform was unanimously regarded as fetching and very satisfying to the eye. However, there was one major caveat. Captured below is a summary of the main points captured during the evaluation.

**Question:** Based upon first impressions, are the intentions of the platform clear?

In this case, all participants agreed that it was an application facilitating conversations with other platform members. The verdict was such that the user interface felt familiar, with possible usage patterns being immediately recognisable.

**Question:** Based upon interactions with the platform, would you say the platform is easily navigable and simple to extract value from?

In this case, all participants were highly satisfied with the UX/UI experience, in part due to the implemented feedback discussed in the previous usability section.

**Question:** Based upon interactions with the platform, would you consider adopting the tool within your teams?

In this case, for the most part, participants showed some reluctance. Not because of concerns with the implemented functionality, but rather because adding additional tools to an already full stack of software products can cause more distraction in teams than it's worth. Another major point that was raised related to existing tools providing support natively for video and conference style calls. **Teamit** currently does not offer this functionality.

## 4 Conclusions

**Teamit** set out as a project to create a Web Application for use by small to mid-size teams with the primary objectives of facilitating team communication while also providing insight into a team's productivity profile. The project has delivered on all of these objectives. Once planning had commenced it became increasingly evident that the scope of work involved could easily spiral out of control if all conceived possibilities were to be included in the projects short term deliverables. For this reason, careful planning was required to ensure the included deliverable remained achievable.

**Teamit's** overall value propositions are quite extraordinary. As a result of Covid-19, millions of people now operate from remote locations, reducing the dependency on office space. This in turn increases the need for software that facilitates team productivity and encourages the combined efforts associated with completing a team's workload. **Teamit** is directly aimed at these professionals. Many established competitors already operate within this space, however, due to the vast number of teams now working remotely, capturing even a small percentage of the market results in a profitable and robust business model.

Overall, the project has been a great success from the perspectives of opportunities to learn and the final product delivery. As previously mentioned throughout this document many technical challenges were encountered along the way. Importantly, all of these challenges were overcome leaving no area in a less than satisfactory implementation state, as per the original requirements specifications. However, using this as an opportunity to reflect, if challenged with a similar project undertaking again, the approach would differ such that off-the-shelf parts would be used where possible, streamlining the overall work involved, thus shortening time to completion and the resulting feedback loops. This is vitally important delivery factor for a relatively unknown product like **Teamit**, as the delivery speed of additional feature will be a very important aspect of encouraging new users to sign up.

To conclude, several months' worth of effort have been invested working towards getting the platform to where it is today. With further research and development and continuous improvements that extend upon existing foundations, there is no conceivable reason why anything but a healthy uptake and adoption of **Teamit** will follow.

## 5 Further Development or Research

Since the initial project pitch, the work involved in designing and building **Teamit** has proven to be both rewarding and highly challenging. A realisation that became apparent mid-way through implementation was that the platform is more useful than initially imagined and could have actual real-world value beyond the scope of a college project. That being said, many additional modifications and features would be required to ensure **Teamit** could confidently be marketed as production-grade software. The following list captures these items and will be used as a backlog of tasks upon commencing future development. Future aspirations for the project involve releasing it as an open-sourced platform, encouraging contributions from the community, ultimately making it a better service while reducing the length of time taken for customers to benefit from its functionality.

### Future Work Items

- Video and conference call support would be implemented. During evaluation of the platform it was apparent that a lack of native support on this front was enough to potentially defer prospective custom's from trialling the platform.
- The database structure would be revisited to include namespaces. Namespaces would sit a layer above teams in the hierarchy, allowing the system to support multiple teams with the same name.
- The systems would be further divided up into well-defined microservices allowing future functionality to be developed separately from the platform itself. This is an important consideration, making the aforementioned community contributions easier to accomplish, in turn encouraging continued development of the platform.
- A different editor library would be used in place of Quill.js. Little was known of this area of development during implementation. Since then, knowledge of a broader range of comparable WYSIWYG libraries exists, with functionality and many powerful customisations surpassing what is currently available within the implemented tooling. One such example includes <https://www.tiny.cloud/tinymce/>
- Proper pipelines would be established, enabling CI/CD and automatic testing upon contributions and code merges.
- An exhaustive list of metric datapoints would be conceived in advance and assigned to various upcoming releases. In this way a product roadmap could be formed, allowing interested customers to remain updated with upcoming features and functionality.

## 6 References

TypeScript. 2021. *Angular 2: Built on TypeScript | TypeScript*. [online] Available at: <<https://devblogs.microsoft.com/typescript/angular-2-built-on-typescript/>> [Accessed 16 May 2021].

Benefits, M., 2021. *5 Major Benefits of Microservice Architecture*. [online] Skelia. Available at: <<https://skelia.com/articles/5-major-benefits-microservice-architecture/>> [Accessed 16 May 2021].

Firebase. 2021. *Cloud Firestore Data model | Firebase*. [online] Available at: <<https://firebase.google.com/docs/firestore/data-model>> [Accessed 16 May 2021].

Open Source For You. 2021. *Go: Perhaps the Best Language for Building Scalable Code*. [online] Available at: <<https://www.opensourceforu.com/2020/01/go-perhaps-the-best-language-for-building-scalable-code/>> [Accessed 16 May 2021].

Drycode.io. 2021. *How To Keep Your Code Dry*. [online] Available at: <<https://www.drycode.io/>> [Accessed 16 May 2021].

Deloitte Ukraine. 2021. *How to reduce the pandemic impact on employees: A guide for company leaders | Deloitte in Ukraine*. [online] Available at: <<https://www2.deloitte.com/ua/en/pages/human-capital/articles/impact-of-covid-19.html>> [Accessed 16 May 2021].

<http://www.fusionoh.com>, F., 2021. *The effects of COVID-19 on employees and the workplace*. [online] Fusion Operational Health. Available at: <<https://www.fusionoh.com/blog/effects-covid-19-employees-workplace>> [Accessed 16 May 2021].

Medium. 2021. *SENTIMENTAL ANALYSIS USING VADER*. [online] Available at: <<https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664>> [Accessed 16 May 2021].

Go Tutorial - Learn Go from the Basics with Code Examples. 2021. *Structs Instead of Classes - Object Oriented Programming in Golang*. [online] Available at: <<https://golangbot.com/structs-instead-of-classes/>> [Accessed 16 May 2021].

Content.dsp.co.uk. 2021. *What are the key benefits of a Database Managed Service?*. [online] Available at: <<https://content.dsp.co.uk/what-are-the-key-benefits-of-a-database-managed-service>> [Accessed 16 May 2021].

Liquid Web. 2021. *What is Single Tenant vs Multi-Tenant Software? | Liquid Web*. [online] Available at: <<https://www.liquidweb.com/kb/what-is-single-tenant-vs-multi-tenant-software/>> [Accessed 16 May 2021].

## 7 Appendices

### 7.1 Appendix A: Midpoint Submission

#### Objectives

**Teamit** aims to provide small teams with a platform that allows for a richer, more inclusive collaboration, increased productivity and velocity within a team. **Teamit** aims to serve two types of end user, namely management and the individual teams contributors. While both users will interact with what is essentially the same system, **Teamit** aims to provide each with a uniquely tailored end user experience.

#### Productivity

In terms of productivity, **Teamit** aims to provide small, remote teams with a suite of tools that encourages cross-team collaboration, communication and cohesion. **Teamit** aims to provide users with a journal for tracking daily tasks, providing frequent updates and seeking help if blocked. **Teamit** aims to reduce the time required to run daily stand-ups recursively providing managers with lengthy and redundant updates. Product name aims to search from a back catalogue of previously discussed issues and find resolutions to their questions, quicker and more efficiently than waiting for a colleague to respond with help. The system aims to guide platform users when seeking help, providing intelligence on who within the team is most qualified to answer their query. The platform aims to provide this type of AI like experience by offering smart searching functionalities as part of the included tool-chain.

#### Performance

A teams performance is key when assessing how well a group of professionals function as a unit. **Teamit** aims to provide management and team leaders with a contextual dashboard that drills down into the performance characteristics of the teams they manage. The platform and product generally strive to achieve this in a fashion that does not alarm the individual contributor, making he or she feel micro-managed. **Teamit** aims to provide administrators with the functionality to perform administrative activities such as adding/removing users to the teams they manage. As part of the performance aims of **Teamit**, the platform should provide managers with functionality that allows for reviewing engagement, collaboration metrics generated by the platform.

#### Efficiency

Quick access to team-specific data is crucial for providing a great user experience. By using data specific to the each team, **Teamit** aims to provide a searchable and well-presented catalogue of archived communications. **Teamit** aims to provide teams with the tools needed to unlock maximum efficiency while limiting downtime. **Teamit** strives to provide a set of tools, allowing teams to revisit previously discussed tasks by reviewing archived communications and previously documented dialogue. **Teamit** aims to intuitively structure and deliver the data to the end-user through a user interface that is easily navigable and a pleasurable to consume.

#### Inclusivity

**Teamit** aims to reduce the social barriers between team members who find themselves forced to work online as a by-product of the Covid-19 pandemic. Some individuals are more adept with working remotely. However, it's not for everyone. **Teamit** aims to navigate many of these obstacles by

providing personal information about team members, allowing for more emotionally led opportunities to bond with colleagues. **Teamit** aims to achieve this by including the following detail on user profiles i.e. time zones, hours of work, interests, birthdays, hobbies and location.

## **Background**

### **Product Domain**

**Teamit** is operating in the communication and productivity space for small to medium sized teams. Researching alternate products in this space reveals there are 3 favourites currently being used by remote teams. Microsoft Teams, Slack, Asana.

### **Competitors**

- **Microsoft Teams**
- Developer: Microsoft
- Operational Since: 2017
- No Users. 75 million

Microsoft Teams is the youngest competitor, and it's also the most popular in terms of user numbers. Teams is part of Office365 suite making it the go-to for many professionals already invested in the Microsoft Ecosystem.

- **Slack**
- Developer: Microsoft
- Operational Since: 2009
- No Users. 10 million

Slack has been around for just over a decade, and is intended to be the centre of workplace collaboration by increasing productivity through simplification of communication.

- **Asana**
- Developer: Asana
- Operational Since: 2008
- No Users. Unknown

Asana is the oldest competitor, having been around longer than other similar platforms. Asana much like Microsoft teams has a core user base gained through other mature products the company offers.

## **Market History**

Since the early twenty-tens there has been an irreversible shift in how organisations communicate. This shift involves more organisations migrating from email based communication to tools like slack for managing internal communications. Taking slack as a use case, the company founded in 2009 raised \$42million dollars in an early round of funding. Fast forward nearly a decade and in early December of 2020, Salesforce announced they would be purchasing Slack for the colossal sum of \$27.7 billion. Slack are not the only player in this space. Other communication tooling such as Asana, Monday.com and the newer MS Teams have had, and continuing to have a big impact on the modernisation of professional communications.

## **Differentiator**

The most notable differentiator separating **Teamit** from its competition exists in how it leverages interactions between team members. Comparable messaging service as mentioned above primarily focus on sending and receiving message, with no additional functionality being offered beyond basic archival and searching.

**Teamit** differs in this respect as the primary focus of system since inception has been to document a back catalogue of contextually relevant conversations amongst members of a team, offering what becomes a self-documenting journal of problems and solutions. Additional value add comes in the form of inter-team performance metrics that can be ingested by management to better understand the health of their teams using metrics such as engagement reports.

When identifying the domain in which **Teamit** will operate, it's been difficult to pinpoint a tool that has the proposed feature set available natively. Furthermore, existing solutions have gone after a different market segment, offering solutions that are best suited to be rolled out org wide, whereas **Teamit** is designed for small teams, and can be used by a single team within a larger company if that's how the company wishes to implement the platform.

## Technical Approach

Modern software development frequently follows a microservices architecture design, opposed to the use of a lesser relevant, traditional monolithic pattern. Using microservices supports the decoupling of discrete units of functionality, making the development/deployment of each a more controlled and well defined process.

**Teamit** relies on a number of different services combined, allowing for full end-to-end functionality. A brief example is provided and highlights the technical considerations of the system.

A user must first have an account which will either involve being added by an admin or creating an account then requesting to be added to a specific team.

Okta handles user accounts and authentication, both for admin and regular users.

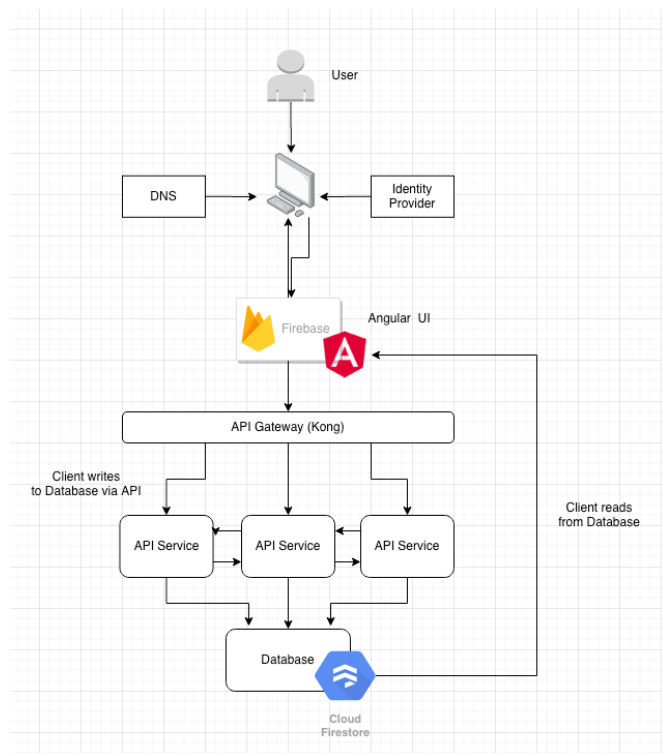


Figure 54 Initial Architecture Design

Once validated a user will have access to the UI through a browser and will carry a bearer token that allows them to make requests to the backend API's.

**Teamit's** UI can read directly from Cloud Firestore database using tooling available as part of the Firebase SDK. Proxying all content through **Teamit's** API for rendering is not required in this instance. Only after a user has logged into the platform will the data be retrieved from the Firestore database using Firebase SDK.

Writing to the database could follow a different approach. Mainly suggested for reasons of validation and added functionality where validation is overly complex for the client to handle. Example, when a user posts a query, **Teamit** will endeavour to suggest a most appropriate team member to provide a response. For this to be achieved an extensive database search and keyword extraction is performed via the API. Once a suggestion has been generated the API then writes the result to Firestore which in turn can be read by the client following the applications standard methods for data retrieval.



## **Resources Required**

### **Local Development**

- **Laptop**

A decent spec'd laptop is required, needed for running CI/CD pipelines locally. A number of resource intensive software's are being used during the build and testing process. Minikube, which is discussed below will be used to host a self-managed Jenkins in addition to running the application binaries in a series of containers.

- **Minikube**

Minikube facilitates running a Kubernetes cluster locally. With regard to production, the various API's that make up **Teamit's** backend will be hosted on a cloud-based Kubernetes offering, Azure AKS. However, this is not ideal for development as avoidable cost will be incurred. Minikube is a scaled replica of a fully functioning Kubernetes cluster making it suitable for building and predicting outcomes of Kubernetes workloads locally.

- **Jenkins**

Jenkins is a well establish, open source platform for orchestrating software builds and tests. Using Jenkins is preferable to other currently available CI/CD tooling as it can be self-hosted, eliminating the resource usage limits of other free tier offerings from competitors. The Kubernetes plugin for Jenkins will provide all of the functionality needed for running Jenkins as a container-based workload.

### **Hosting**

- **Azure AKS**

Kubernetes running on Azure. As highlighted in a previous section (Minikube) this service comes at a cost, thus, will only be leveraged at end of the project when **Teamit** needs to be externally hosted for grading purposes.

- **Docker Hub**

Docker Hub is a hosted container repository offered by Docker. This service will be used for storing the built containers containing **Teamit's** compiled binaries. Deployments will fetch container images from this storage location.

- **GoDaddy**

The domain registrar of choice. **Teamit's** unique TLD will be purchased and managed through this domain provider.

- **Version Control**

GitHub will be utilized for storing **Teamit's** source code. A private repository will be used during development, and will be made public upon project completion, unless access is requested in the interim.

- **Okta**

Okta is an identity provider that will be utilized for handling user management and authentication.

## **Project Plan**

Waterfall methodology will be used to track progress during development of **Teamit** over the coming months. There may be sections that need reworking as the project unfolds, so it will be somewhat agile, however it's not favorable to use full agile methodology for the management of this project.

Waterfall is the better choice for this project because of the large amount of documentation required for submission along with working software.

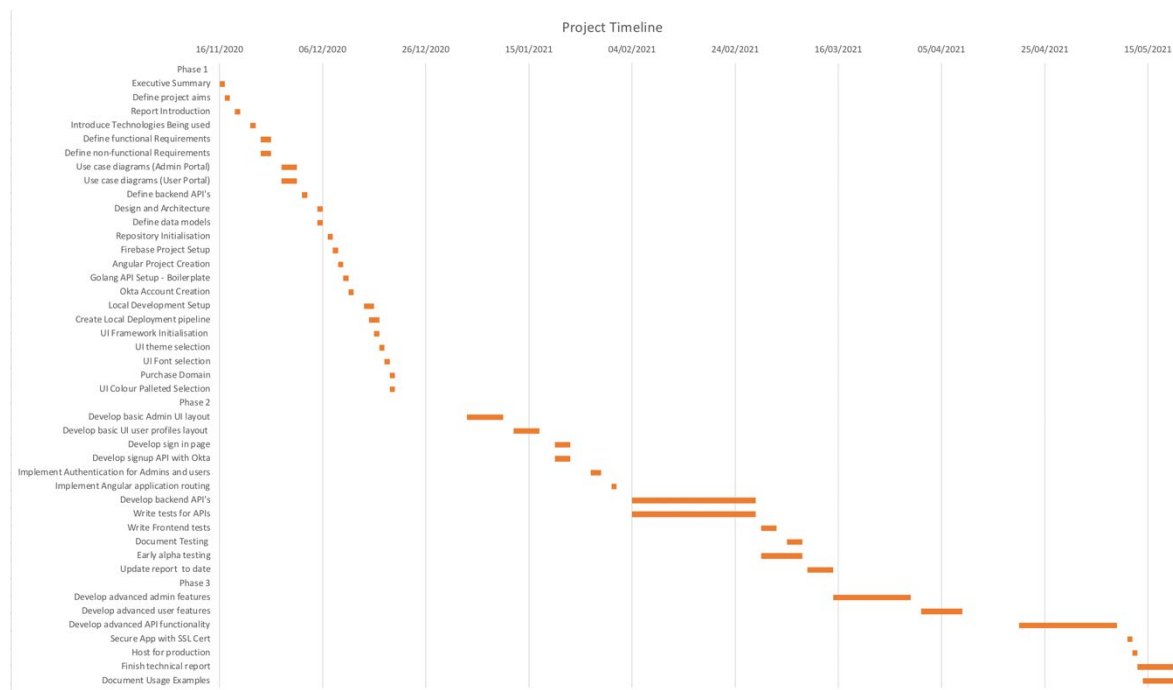


Figure 55 Project Plan Breakdown

## **Technical Details**

**Teamit** will be developed using a combination of industry leading technologies for both backend and frontend components. The following collection is not an exhaustive list of all third-party libraries that will be implemented, but rather a high level summary of the main components accompanied by a justifications for choosing each.

### **Backend**

The backend language of choice is Golang. Built with speed in mind it seems a logical choice. Golang compiles to static binaries that can be run on a wide variety of hardware and operating systems. **Teamit's** backend consists of a collection micro services written in Go, once combined make one large API responsible for handling queries and providing the services that are at the core of **Teamit's** functionality. Golang has a powerful HTTP package included as part of its standard library. The built-in HTTP library facilitates most of the RESTful services required as part of client server communication.

<https://golang.org/>

### **Frontend**

**Teamit's** frontend will be built using Angular. Angular is an Enterprise grade framework for developing single page web applications (SPA) using Typescript. Angular is comprised of a component based architecture that facilitates the development of reusable, modular components. A powerful templating engine also makes possible displaying components that dynamically change the content of an application. In addition to Angular, PrimeNG is a highly performant component library that will provide the building blocks of **Teamit's** user interface. Additionally, PrimeNG provides a suite of icons, themes and other stylistic features that will make **Teamit** attractive to use.

<https://angular.io>

<https://www.primefaces.org/primeng/>

### **Persistence**

Firestore will be used to persist **Teamit's** data. The Firebase Admin SDK provides a set of convenient tools for accessing the data. **Teamit** uses a non-relational data model so a highly performant object storage (Firestore) is preferable. Data storage and availability can be a complex area. Offloading this is highly advantageous making it a more cost effective and straightforward area.

### **Development Environment**

GoLand, and Webstorm are the IDE's that will be used when building **Teamit**. Both these IDE's offer a powerful set features and support for their respective languages.

## **Evaluation**

A number of different approaches will be explored when testing **Teamit's** functionality. Testing is major pillar of software development and necessary for delivering a product that has your full confidence. The following types of testing will be implemented;

### **Stubbing**

Stubbing will be used to validate that API's handle responses in the correct manner. Also known as mocking, this technique is vital for testing how the combined components of a larger system will handle failures and other un-expected requests / responses from the system.

### **Unit Testing**

Unit tests will be implemented where possible. I say where possible, because; as a primarily web base API product with a skin, many of the functions needed to ensure a working platform are made up of RESTful request that expect valid HTTP verbs along with a payload as an action. Secondly a response with a potential payload and status report. Using a unit style approach allows for a function's return value to be compared with an expected HTTPS response status. For example, should a user post to their wall, once the user clicks submit the freeform text should be written to persisted data store and a successful completion of the task should present the user with a 200-status report

### **Integration Testing**

*Teamit's* functionality will primarily be validated using integration tests. during development I will ad hoc test the platform and observe outcomes of certain actions. Based on these learnings the executed sequence of steps can be replicated and crafted into an automated pipeline that should run post deployment. As the product develops and functionality is added the number of integration tests will also increase.

## 7.2 Appendix B : Development Machine Specifications



*Figure 56 Development Machine Specification*

### 7.3 Appendix C: API Module Structure

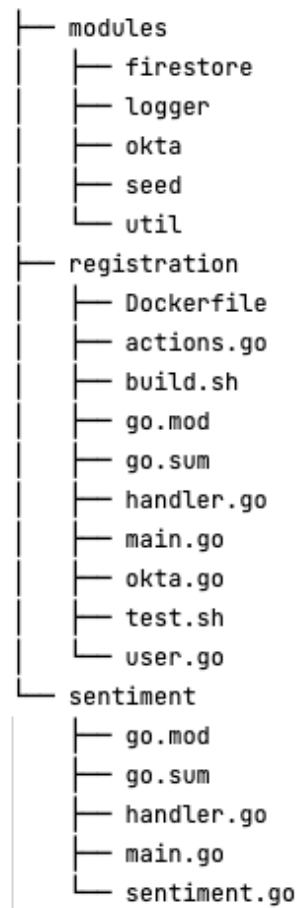


Figure 57 API Module Structure

## 7.4 Appendix D: User Request Validation

```
// User struct defines a user request
type User struct {
    FirstName string `json:"firstname"`
    LastName  string `json:"lastname"`
    Email     string `json:"email"`
    Password  string `json:"password"`
}

// configure logging
var logger = lg.CreateLogger()

// validateParams validates a user request
func (ur *User) validateParams() error {
    if ur.FirstName == "" : errors.New("firstname is a required field") ↗
    if ur.LastName == "" : errors.New("lastname is a required field") ↗
    if ur.Email == "" : errors.New("valid email address is a required field") ↗
    if ur.Password == "" : errors.New("password is a required field") ↗
    return nil
}
```

Figure 58 User Request Validation

## 7.5 Appendix E: CORS Issue Resolution

### **Browser Error (Similar – not exact example)**

✖ Access to XMLHttpRequest at 'http://localhost:5000/global\_config' step1:1 from origin 'http://localhost:8080' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

*Figure 59 Example CORS Console Error*

### **Resolution**

```
// ConfigureCors - values currently set are only suitable for Development
func ConfigureCors(res *http.ResponseWriter) {

    (*res).Header().Set( key: "Access-Control-Allow-Origin", value: "*")
    (*res).Header().Set( key: "Access-Control-Allow-Methods", value: "POST, GET, OPTIONS, PUT, DELETE")
    (*res).Header().Set(
        key: "Access-Control-Allow-Headers",
        value: "Accept," +
            "Content-Type," +
            "Content-Length," +
            "Accept-Encoding," +
            "X-CSRF-Token," +
            "Authorization," +
            "username-from-header")
}
```

*Figure 60 CORS Custom Headers*



## 7.6 Appendix F: Behaviour Subjects

<http://reactivex.io/rxjs/manual/overview.html#behaviorsubject>

### BehaviorSubject

One of the variants of Subjects is the `BehaviorSubject`, which has a notion of "the current value". It stores the latest value emitted to its consumers, and whenever a new Observer subscribes, it will immediately receive the "current value" from the `BehaviorSubject`.

*BehaviorSubjects are useful for representing "values over time". For instance, an event stream of birthdays is a Subject, but the stream of a person's age would be a BehaviorSubject.*

In the following example, the `BehaviorSubject` is initialized with the value `0` which the first Observer receives when it subscribes. The second Observer receives the value `2` even though it subscribed after the value `2` was sent.

```
var subject = new Rx.BehaviorSubject(0); // 0 is the initial value

subject.subscribe({
  next: (v) => console.log('observerA: ' + v)
});

subject.next(1);
subject.next(2);

subject.subscribe({
  next: (v) => console.log('observerB: ' + v)
});

subject.next(3);
```

With output:

```
observerA: 0
observerA: 1
observerA: 2
observerB: 2
observerA: 3
observerB: 3
```

## 7.7 Appendix G: Value Passing Between Components

```
<div *ngFor="let team of userTeams" class="team-icon">
  <div class="two-chars">
    <a
      (click)="broadcastSelectedTeam($event)"
      pTooltip="{{ team }}"
      tooltipPosition="top">{{ team }}
    </a>
  </div>
</div>
```

Figure 61 Trigger Broadcast function on click

```
broadcastSelectedTeam($event) {
  this.teamService.selectedTeam($event.target.innerText);
}
```

Figure 62 Send broadcasted value to shared service

```
private teamDataSource = new BehaviorSubject<any>(_value: '');
currentlySelectedTeam = this.teamDataSource.asObservable();
```

Figure 63 Broadcast value from shared service

```
// load the users of a selected team
this.teamService.currentlySelectedTeam.subscribe( next: team => {
  this.teamService.getTeamMembers(team).subscribe( next: (t: any) => {
    if (this.selectedTeam !== '') {
      this.selectedTeamMembers = t[0].members || [];
      this.teamName = t[0].name;
    }
  });
});
```

Figure 64 Subscribe to value from destination component

## 7.8 Appendix H: Query Creation

```
onSubmit() {
  this.queryPost = {
    id: randomString( len: 20),
    body: this.queryPost.body,
    createdBy: this.loggedInUser,
    createTime: Date.now(),
    status: 'open',
    tags: this.queryPost.tags,
    thumbsDown: 0,
    thumbsUp: 0,
    title: this.queryPost.title,
    type: 'query'
  };
  console.log(`DEBUG OUTPUT → ${this.queryPost}`);
  this.contentService.writePostToDB(this.queryPost, this.selectedTeam);
  this.reset();
}
```

Figure 65 Query Creation

```
writePostToDB(postObject: IPost, teamName: string) {
  // Add a new query or blog record with a generated id.
  this.db.collection( path: `teams/${teamName}/posts`).doc(postObject.id).set(postObject)
    .then((d :void ) => {
      console.log(`${postObject.type} written with ID:`, postObject.id);
    })
    .catch((error) => {
      console.error(`Error adding ${postObject.type}: `, error);
    });
}
```

Figure 66 Persist Query Object to Database

## 7.9 Appendix I: Tag and Collaboration Analysis

```
getContributionData(team, userID) {  
  return this.db.collection<IPost>( path: `teams/${team}/posts`, queryFn: ref =>  
    ref.where( fieldPath: 'createdBy', opStr: '=', userID)) AngularFirestoreCollection<IPost>  
    .valueChanges() Observable<IPost[]>  
    .pipe(take( count: 1));  
}
```

Figure 67 Fetch Collaboration Data

```
fetchContributionMetrics(team) {  
  
  const dataArray = [];  
  const labelsArray = [];  
  const randColors = [];  
  
  this.metricsService.getTeamMembers(team).subscribe( next: resp => {  
    const members = resp[0].members.toString().split( separator: ',');  
    members.forEach(member => {  
      this.metricsService.getContributionData(team, member).subscribe( next: data => {  
        dataArray.push(data.length);  
        labelsArray.push(member);  
        randColors.push(this.colors[Math.floor( x: Math.random() * this.colors.length)]);  
        this.contributionData = {  
          datasets: [{  
            data: dataArray,  
            backgroundColor: randColors,  
          }],  
          labels: labelsArray  
        };  
      });  
    });  
  });  
}
```

Figure 68 Compute Collaboration Metrics

## 7.10 Appendix J: Showcase Poster





Kevin Brennan **BSHCSDE4**

**Teamit** is a productivity application aimed at small to medium-sized, remote teams. **Teamit** includes a suite of tooling specifically designed to encourage employee productivity, while providing management with the required metrics needed to maintain full visibility of their remote employee's productivity.

- Enterprise-grade authentication
- Create/Update/Delete Teams
- Invite teammates
- Team Collaboration
- Querying & Journaling support
- Commenting & Conversation
- Searchable Team Content
- Management Metrics Suite



Need assistance...

Start typing and Teamit's smart engine will pair your query with matching results

national college of ireland

No results found matching national college of ireland

teamit

0.4593

[IDE's]

[Languages]

[Frameworks]

[Cloud Services]

[VCS]

[IdP]



GoLand



WebStorm



TypeScript



JavaScript



Golang



Angular



PrimeNG



Serverless



Google Cloud



Algolia



Firebase



GitHub



Okta



honorable mention (Spotify)

Figure 69 Teamit Showcase Poster

## 7.11 Appendix K: Reflective Journals

### 7.11.1 Introduction

My name is Kevin Brennan (x16149823). I am a 4th year student studying a Bachelor of Science in Honours Computing at the National College of Ireland. This journal aims to fulfil two purposes. Firstly, and probably most important; it will serve as an outlet for me to document and share my thoughts and experiences over the coming 8 months as I work towards completing my project. Personally, I place a high value on documentation and journaling activities in general. Doing so regularly makes the act of reflection a more convenient exercise. Secondly, producing a journal is a mandatory part of the project brief so I will meet that requirement with my best efforts.

At the time of writing I am an employee of a trade finance company, TradeIX. My employment with the company started in 2018 and I have been working remotely since March 2020. I am a Devops Engineer working as part of a small team responsible for managing and maintaining the deployment lifecycle of our products.

I have a wife Charlotte and a little boy Harry. Recently we bought a house and are delighted to have our own space where we can grow as a family. If not already obvious, life is very busy. Between being a Dad, working with TradeIX, attending college part-time and working on the house, time is a precious resource and it is important that every minute is spent wisely.

I enjoy project work. I regard my skillset as being well suited to tasks that involve research, planning, and forward thinking. I am a self-motivator. Mid-way through 2nd year I started gathering ideas for my 4th year project. Information gathering involved making note of notable projects I came across while browsing online, collecting links to interesting frameworks and other resources I deemed potentially beneficial at the time.

So, when the project brief was assigned I fully expected that an idea would jump at me and I wouldn't need to give it any thought. I was wrong! Initially I found myself trying to create problems that would lend themselves to being easily solved using technologies and processes that I am already familiar with. This was a poor approach and quickly I realised that trying to invent a problem so I could solve it was essentially putting the cart before the horse.

I tried to understand areas in my own daily activities that I find abrasive or problematic and realised that (full credit to the software industry) there are already so many great services and products that address many of these issues.

Through reading my journal you will learn about the struggles and setbacks I experience along the way, I expect there to be many, along with the progress that will follow after many late nights and long weekend's overcoming the problems encountered during development.

### 7.11.2 October 2020

**15/10/2020**

Started writing a transcript that will be used during the recording of my 4<sup>th</sup> year project pitch. There is no requirement for this to be typed but I will benefit from having it documented and referenceable during the recording.

**18/10/2020**

I recorded and uploaded the video link for my project pitch this evening. All in all I am reasonably happy with how it turned out. I had planned on shooting the video over a series of smaller sections and editing them together. However, I decided against this when the time came as I was approaching the submission deadline and I definitely did not want to miss the upload window over something simple like having the footage recorded but not presentable. I opted to use Zoom for recording as I have a greater level of familiarity with its features compared to MS Teams.

**21/10/2020**

Downloaded the project Technical report template from Moodle. Looking through some of the headings and how the document is structured. This will be helpful when I start setting out the structure of my report. I like that the college have shared this resource.

**27/10/2020**

Received an email from the college today with information on assigned project supervisors. I have been assigned Aqeel Kazmi. Did a quick search on LinkedIn, unless there is a different Aqeel Kazmi not on LinkedIn I am happy with his credentials. He holds a PhD and master's in computer science. Hopefully this is the correct guy.

**31/10/2020**

Have not received an email from Aqeel yet regarding my project pitch video. I have emailed him requesting a status update. Starting to feel a bit anxious as we have an upcoming deadline of the 8<sup>th</sup> November for the submission of the project proposal document which I have not started. Obviously, there is no merit in starting this document without knowing if my pitch was successful. Hopefully he will make contact with good news in the next day or so.

**31/10/2020**

Over and out for October. Will not be making any further updates to this month's journal. See you in November!



### 7.11.3 November 2020

**02/11/2020**

Received feedback from my project supervisor today. His response was short. My pitch has been successful. He is happy for me to proceed with the full proposal. He highlighted that my idea sounded great, and he is interested to know which competitors I have research, and the features I will offer that they do not. Time to get busy drafting the full proposal.

**05/11/2020**

I have completed the project objectives and technical approach sections of the proposal. One observation while working on the objectives was that it's challenging to continually write about the proposed platform in greater level of detail while it still remains nameless, without an identity or domain. I found myself briefly journeying down a rabbit hole looking for a suitable, catchy product name. Before long I was able to identify that my efforts were not in line with the priorities so I stopped immediately and will set aside some time to complete at a later date.

**07/11/2020**

Eugene O'Loughlin's excel template for creating a detailed project plan has proven very helpful. With previous experience creating Gantt charts using purpose-built tools such as Microsoft Project I have found they can be overbearing with requirements for high levels of detail. Eugene's YouTube video resource shared by the college nicely simplify this part of the proposal.

**08/11/2020**

Submitted a first draft of the proposal tonight, with 1.5 hours to spare. I am reasonably pleased with the level of detail included. I may revisit it at a later date to add some supplementary diagrams but for now the submission is sufficient. I will aim to include some architecture diagrams, along with perhaps some system overview diagrams before the final date for submissions closes.

**15/11/2020**

So far have not had any feedback from my supervisor. I am keen to learn his thoughts on the proposal. Specifically, I would like to know if I am overdoing it with the devops workload, regarding hosting, build pipelines, artifact management etc. This is what I do for a day job and I regard these activities as highly important, that said I am cognizant that marks are being awarded for the quality and originality of developed code, with far less regard for hosting or deployment.

**28/11/2020**

Reviewing the timeframes set out as part of the project timeline earlier this month and I am noticing that I have already fallen behind on the tasks I aimed to have completed by this date. I am excited to work on the project, however, this semester the overall workload is very taxing. The additional projects and TABA's in place of terminal examinations are very demanding on time, focus and energy. Not to mention work is hectic as per usual, and with building work commencing on our house before we can move in (work which I am responsible for completing) I am having a hard time keeping all the balls in the air. I keep telling myself if it was easy everybody would be doing it. I had a nice message from a close friend the other day simply saying,



“Tough times do not last, but tough people do!”

I will make up for the lost time over the coming weeks and get the project back on track ahead of the mid-term presentation.

**30/11/2020**

Done. Journal is complete for November. Not as insightful as originally hoped, nevertheless it's complete for another month. Over and out for November. See you in December. Tis the season to be jolly!

#### 7.11.4 December 2020

**31/12/2020**

Not much to contribute to the journal for December. This month's entry takes on the form of a reflective record rather than a regular update. Reason being, with all the CA's due over the past few weeks I simply have not had time to work on anything project related. An exception to this was the mid-term submission (due on the 22nd December) which was a frantic race to the deadline.

Night after night I burned the midnight oil trying to prepare my submission, which I did manage to deliver on-time. However, I was not completely pleased with the quality of work. Put simply, the workload of the past few weeks has been relentless, and with more immediate deadlines the project has suffered.

A video submission accompanied the report upload on the 22nd December, where I noted being time-poor as the single biggest challenge encounter thus far while attempting to work on the project. With the first semester coming to a close, I can only hope that time will be more plentiful next year so that I can spend the time that is desperately required to make some progress on the project front.

### 7.11.5 January 2021

**10/01/2021**

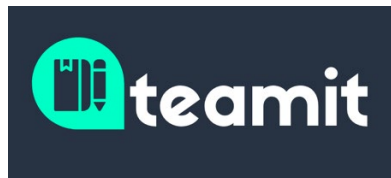
Finally, I have decided on a title for the project, **Teamit**. It's catchy, to the point, relates to the product and most importantly still has a good top-level domain available, which I have just purchase.

<https://teamit.app>

I secured the domain using GoDaddy domain registrar. Now that I have settled on a name for the product, I can get cracking on designing a logo and picking some colours that will be used to build a brand around the platform.

**12/01/2021**

Really happy with how professional the newly designed logo looks. I used a tool called Canva to create it. Designing the logo was a simple task, but a necessary one; which gives the project an identity. Also, this evening I have settled on a colour palette that will be used to keep appearance consistent throughout the UI and project media.



*Figure 70 Teamit Logo*

Additionally, this evening I initialized a GitHub repository that will be used to store the projects code base, while I also configured a new Firebase project that will provide hosting and a data persistence layer for the application.

**13/01/2021**

This evening I was due to have a meeting with Aqeel, project supervisor. However, due to technical difficulty he encountered the meeting was postponed.

**18/01/2021**

Pretty happy with how the main landing page of the application is starting to look. It took a little bit of time to understand the best practices around structuring an Angular application for scalability and maintainability but is since resolved. I am happy with how the project directory structure looks now. I have borrowed from the advice put forward in this article <https://itnext.io/choosing-a-highly-scalable-folder-structure-in-angular-d987de65ec7>

**23/01/2021**

The final semester commences this a.m. However, we have a cancellation for the morning class. Our lecturer is feeling unwell. I will use the time wisely to make some progress on the project. Setting up an Okta Developer account this morning and working on building authentication into the application.

Following the developer documentation available at <https://developer.okta.com/docs/guides/sign-into-spa/angular/before-you-begin/>

**Update:** The following screen grabs highlight the progress made on the login functionality. As can be seen, the application is running of localhost:4200. After the user clicks Log in, the flow is directed to Okta to perform user validation, before returning the user to the application. Once redirect back to **Teamit**, the user is displayed the their unique landing page, hosting their posts and interactions.

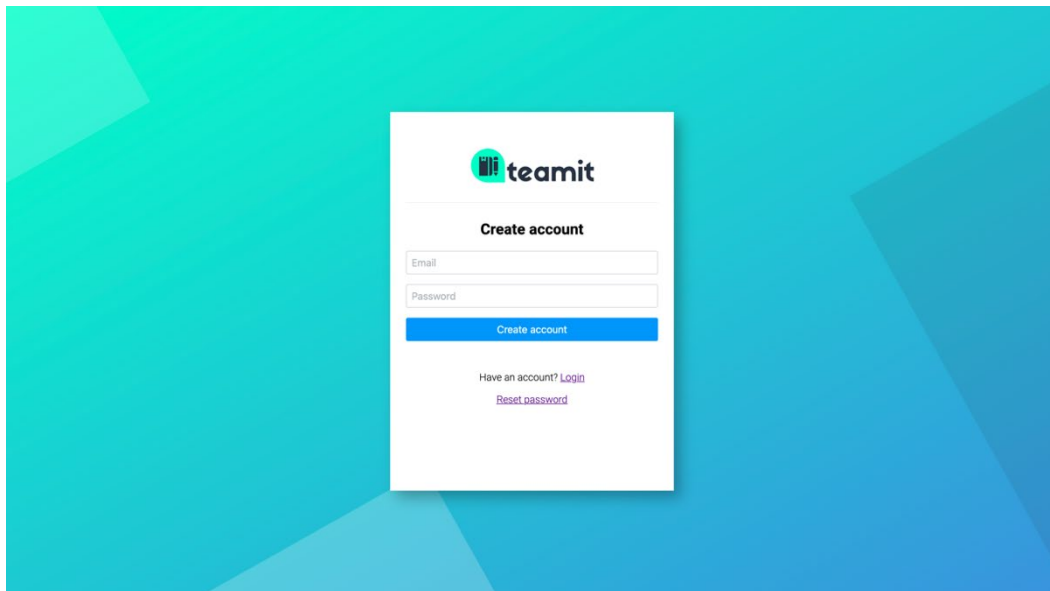


Figure 71 Initial Login

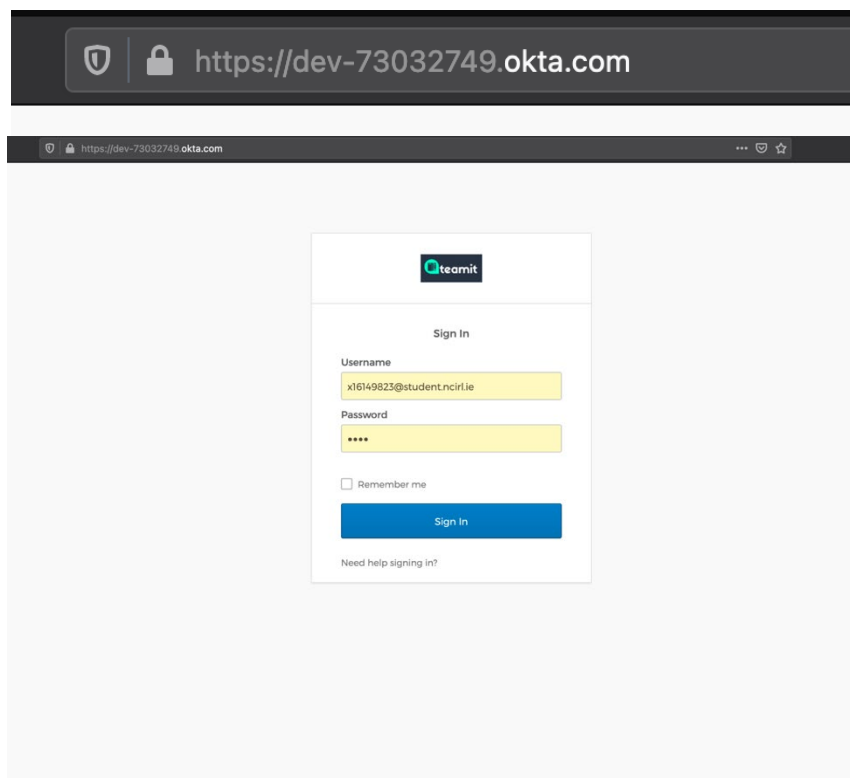


Figure 72 Okta Auth Redirect

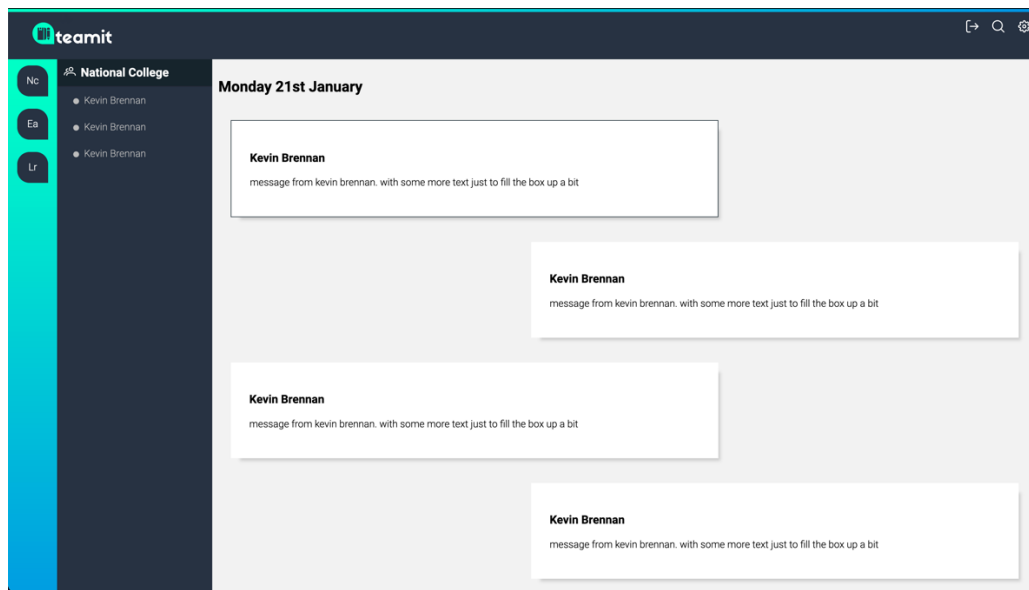


Figure 73 Home Dashboard

**27/01/2021**

This evening I met with my project supervisor to demo some of the functionality I have been building since we last spoke. My primary focus has been to configure the UI login and routing functionality. As highlighted during the technical requirements section, I am using Okta for authentication, and Angular as the frontend framework. In addition to successfully implementing both login and routing components, I created the main dashboard view returned once a user successfully authenticates. Each of these items took several hours, and I was delighted with the progress made on each. However, despite best efforts, I got the impression Aqeel wasn't as enthused by the work. Commending me for my effort, he quickly moved the conversation on, keen to understand what functionality I would be delivering next.

**28/01/2021**

Having had the opportunity to work on the codebase recently, I need to make an effort to update my report, including this month's journal along with some information about the newly added functionality.

### 7.11.6 February 2021

#### **10/02/2021**

Met with my project supervisor this evening. Demonstrated some newly added functionality, specifically relating to the backed API code that will support user registration and login/logout functionalities.

#### **13/02/2021**

All modules are now back in full swing, each with their respective development projects. Getting busy again, however, after 4 long years there is finally light at the end of the tunnel! Just as well too, between working from home, college, and lockdown in general, the past number of months have been increasingly challenging mentally. The longer days and summer cannot come quick enough this year.

#### **14/02/2021**

This evening I have been working on the report, adding more functional requirements and use case diagrams. I'm not sure why these are taking so long to complete, seems like they have been hanging over me for too long. I did say to myself that I would take one a day and within 2 weeks they would be all completed. Pity I didn't stick to that as I'd be close to done with them now if I had.

#### **21/02/2021**

Finally, I'm done with functional requirements and use cases/diagrams. Next, spend some time implementing new functionality in code ahead of sync project supervisor later this week.

#### **24/02/2021**

Unfortunately, I couldn't make tonight's sync with Aqeel. Some stuff came up at home that took priority. Besides making some additional progress with the technical documentation I didn't have much to demo in terms of new feature/functionality. Some heavy projects in two of the other modules have ramped up which I am keen to knock out as quickly as possible. However, the areas pertain to gRPC and Ruby on Rails, both technologies I have zero experience with so the learning curve is steeper than I would like.

### 7.11.7 March 2021

#### 03/03/2021

Development has started on an API that will be used for performing sentiment analysis as one of the metrics that its reported to a given teams administration dashboard. Similar to the registration API, this service will also be built using Go. Right, quick update done, let's get cracking on the code.

#### 07/03/2021

Work on the sentiment analysis API is coming along nicely. I enjoy working with Go. It's a nice language and there is a really helpful community that has formed around it. It helps a lot when there are good resources available online that assist when you are unsure of how a particular piece of logic should be implemented.

Initial sentiment analysis was performed using a library Sentiment, which is a pre-trained sentiment analyser written in Go and uses a rather large collection of IMDB reviews as its learning source. However, upon testing this library with sample input I found the output to be pretty questionable. Providing sentences that contained commonly regarded negatives resulted in output that displayed sentiment as positive. I'm not sure about continuing any further with the particular library. <https://github.com/cdipaolo/sentiment>

#### 14/03/2021

Pretty bogged down with other modules and their respective assignments. Not much project progression to report since my previous update. I did however come across an alternative to the previously implemented sentiment analysis library that should work better. Linking here for record <https://github.com/grassmudhorses/vader-go>.

This library takes a more simplistic approach to weight sentiment, assigning a numerical value to each word based on a lexicon of commonly used English words. Initial testing has yielded promising results so my next steps will be to implement this library within the sentiment API in place of the previously used analyser.

#### 23/03/2021

It's been over a week or so now, however, work on the sentiment analysis API is complete. Development work took longer than I would have initially liked, however, its functionality is important and ultimately will become a core metric used by platform metrics. As part of the implementation, I was able to create functionality that allows for reporting on both team sentiment, as well as individual sentiment. Included are the method signatures for each function as a snapshot, with full technical implementation steps fully discussed during the implementation section of the report.

```
// IndividualSentiment takes a team name and user email
// as input, returning an overall sentiment score for
// the specified user.
func IndividualSentiment(team string, userEmail string, fsc *firestore.Client) (sentimentScore float64) {
```

*Figure 74 Individual Sentiment Method Signature*

```
// TeamSentiment takes a team name as input, returning
// an overall sentiment score for the teams interactions
func TeamSentiment(team string, fsc *firestore.Client) (sentimentScore float64) {
```

*Figure 75 Team Sentiment Message Signature*



### 7.11.8 April 2021

**02/04/2021**

With user authentication sign-in/signup implemented and supported via the registration API, next I need to work on what will become the main chat functionality of the application. I'm a bit anxious about this piece of work. The plan is to display dynamic content specific to the logged-in user, using their email as the unique identifier. This will involve showing different information for teams, team members and chats, all based on the logged users ID. Additionally, the logged user should be able to respond to other team members posts, as well as perform other functions such as marking a given post as helpful or resolved for instance.

**11/04/2021**

I have had relative success so far with the implementation of the chat functionality within **Teamit**. Currently, I am running into a wall with an issue that relates to asynchronous calls and the order of execution and returning of data via javascript in the browsers. For example, function-B requires data returned from function-A, before being successfully able to make the next call in the flow that loads data. In this case, function-A is an asynchronous call that has yet to return a value before function-B continues with execution. This is causing me a lot of bother and I have been trying different solutions for several days now with little success. The best solution I have currently is to make the OnInit class of each component tasked with loading data asynchronous using ES6 async/await functionality. I have had some success but the whole flow is not correctly implemented yet. The example below shows a working solution to this particular piece of the puzzle, whereby firstly the logged-in user's identifier is fetched. The OnInit function will pause execution until the loggedInUser function returns a value. The next step is to call observable steam that is responsible for broadcasting the selected team by name based on an event fired by the user clicking. Once the observable stream has returned a value then another observable stream is called and upon subscribing the component passes the raw post data on to the template for rendering.

```
async ngOnInit() {
  this.userEmail = await this.userService.loggedInUser();
  // get currently selected team
  this.teamService.currentlySelectedTeam.subscribe( next: team => {
    // load teams specific posts
    this.selectedTeam = team;
    this.posts = this.contentService.fetchTeamPosts(team, this.userEmail);
    this.loading = false;
  });
}
```

*Figure 76 Asynchronous OnInit Function*

**18/04/2021**

Next step is to start development of the component that will be used by the user to post content to their journal. I have a design in mind that will handle this as once single component with child components for each of the different functions (blog, query, to-do).

28/04/2021

Well, that was not trivial. Seems there is a general theme forming around my project whereby I appear to have made everything overly complicated, requiring days' worth of work and effort that I simply had not accounted for when conceiving the various functional components of the platform.

Again, while it took longer than I would have liked, I am very happy with the results. I have a component that implements a WYSIWYG editor, whereby the user can format input in a variety of different ways (including uploading pictures that are base64 encoded) before saving them to the database as one long string. I had not accounted for the work that would be required to re-render the HTML formatted data upon page load, but I did find a parser as part of the QUILL library that allows for this. Tooltips add a nice touch to the experience.

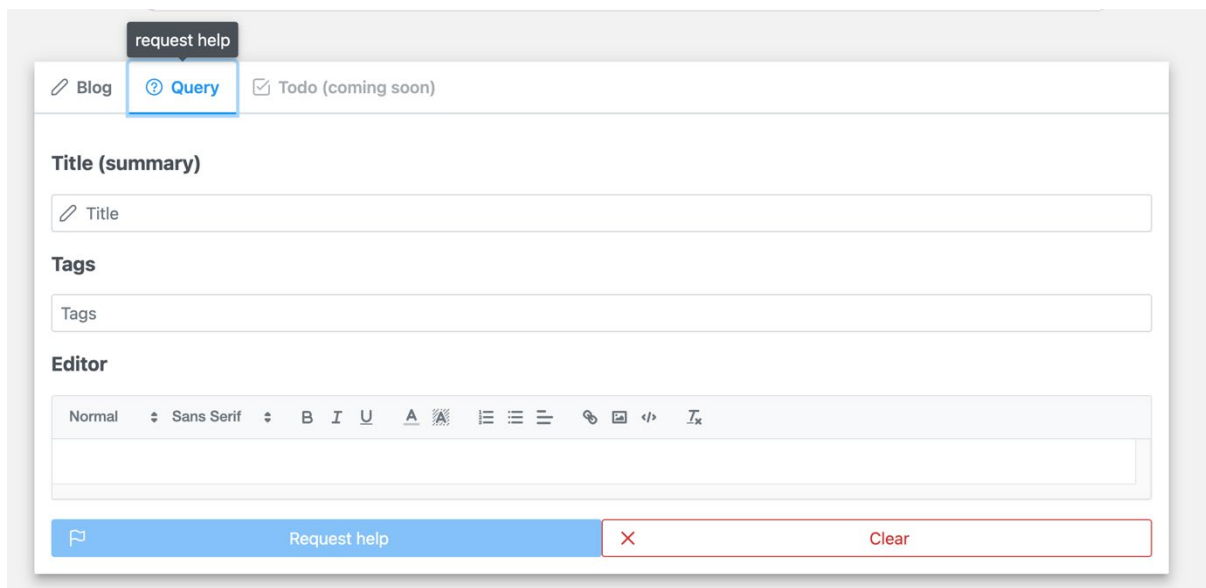


Figure 77 Compose Component

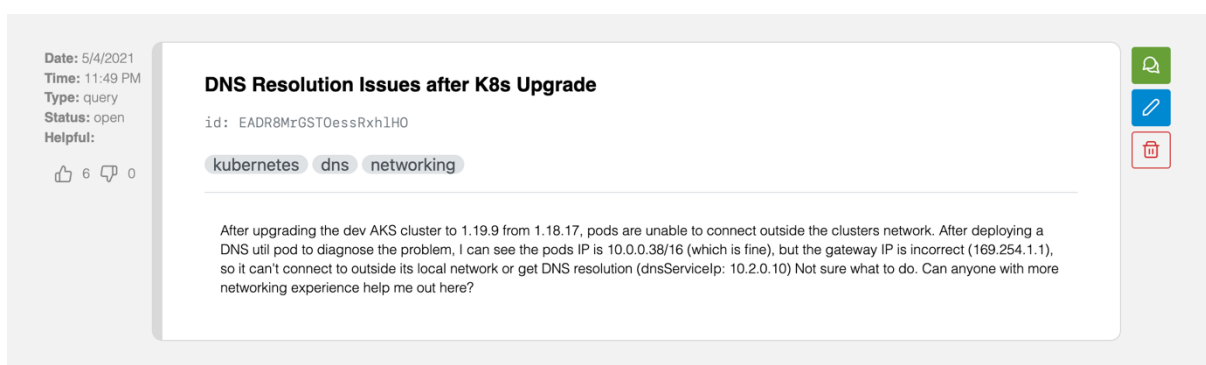


Figure 78 Rendered Post

### 7.11.9 May 2021

**02/05/2021**

It's May already, where have the months gone! Initially, I was eagerly counting the days until 4th year would be complete. Now I'm realising that an extra couple of days will most probably be required to get this project over the line. Unfortunately for the college, they fell victim to a ransomware attack last month, resulting in all IT systems being offline (including Moodle) for almost 10 days. Luckily for me, the college was obliged to allow extra days on all submission deadlines to account for the extended loss of service.

**03/05/2021**

User commenting functionality complete, with some additional tweaks added around resolving posts, assigning thumbs up, thumbs down and overall presentation. Next step is to commence work on the metrics section

**08/05/2021**

Will keep the update brief as there is still much to do. Work on metrics section is complete. Fortunately, hooking up the external sentiment analyser worked as expected, which was a great relief, and encouraging to finally see the different components of functionality coming together to perform as one service.

**12/05/2021**

Final call with my project supervisor this evening. There have been several occasions throughout the project where it felt like I was making excuses as to why the project wasn't being worked on consistently. Retrospectively, better time management could have helped to a certain extent, however, my excuses were all reasonable and valid. I am not the type of person to unjustly make excuses. Regardless, on this occasion, Aqeel was very happy with the recent progress made and commended me on my efforts. I know what I am capable of when I get the time, so to learn that Aqeel was also impressed was great to hear.

**15/05/2021**

Nothing like a last-minute scramble to finish. I had hoped to be done by now but such is life. I am reasonably confident that between now and Sunday midnight I will have completed the remaining items. If anything is still outstanding it will be mentioned within the documentation.

**15/05/2021**

Working on the final implementation of the search functionality. A third party service Agolia is being leveraged for this feature. I find it quite odd that Firebase doesn't provide real-time search functionality out of the box, especially given that it's created by engineers at Google, the most advanced search engine in existence.

## 7.12 Other materials used

### 7.12.1 Viability Survey

Results from an initial user survey.

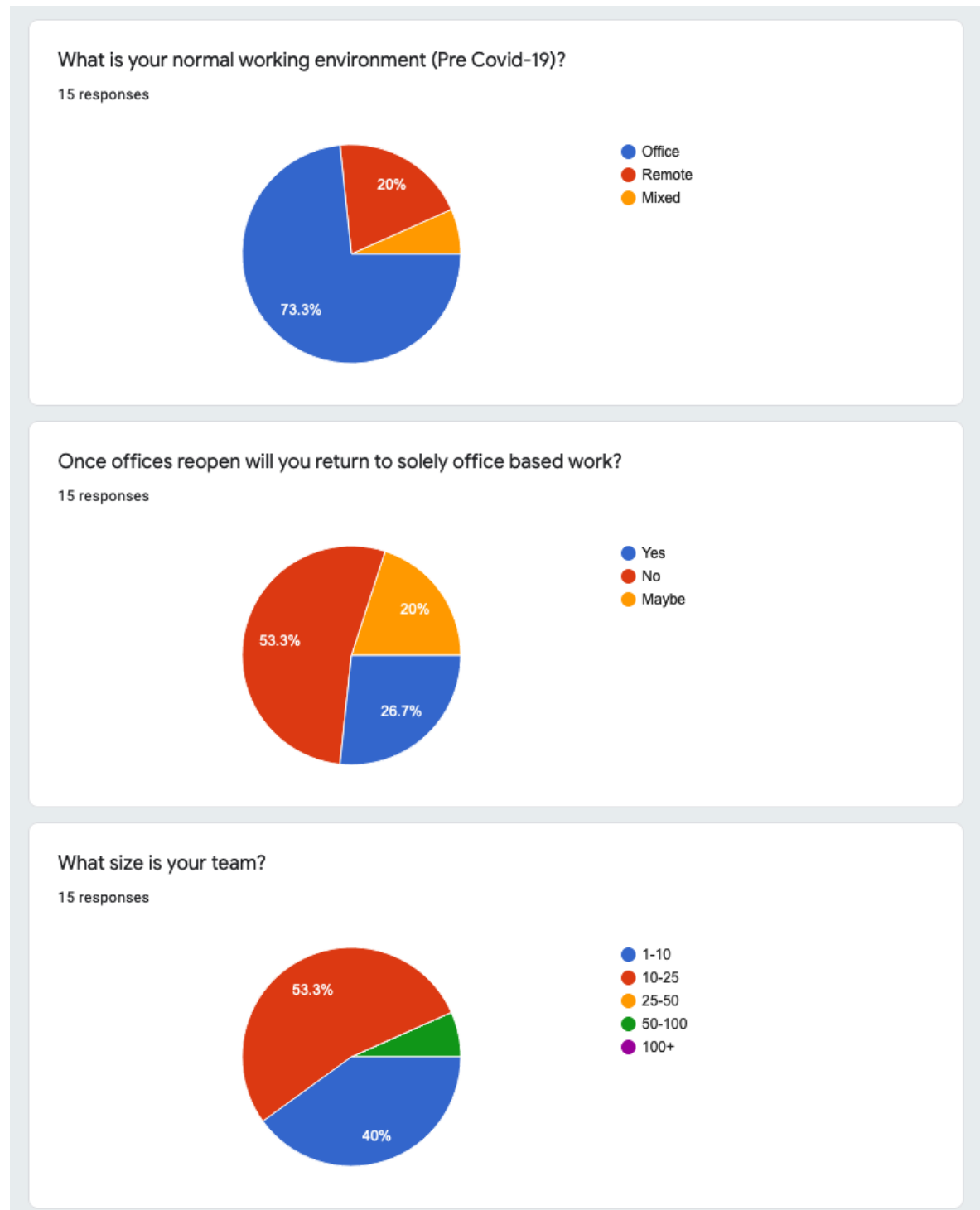
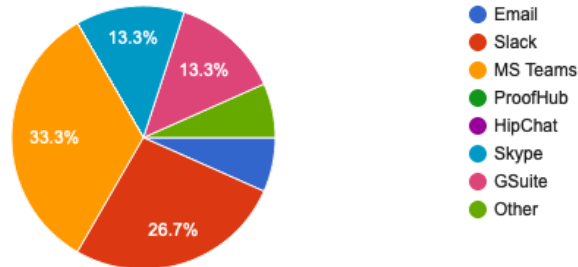


Figure 79 Survey Pt.1

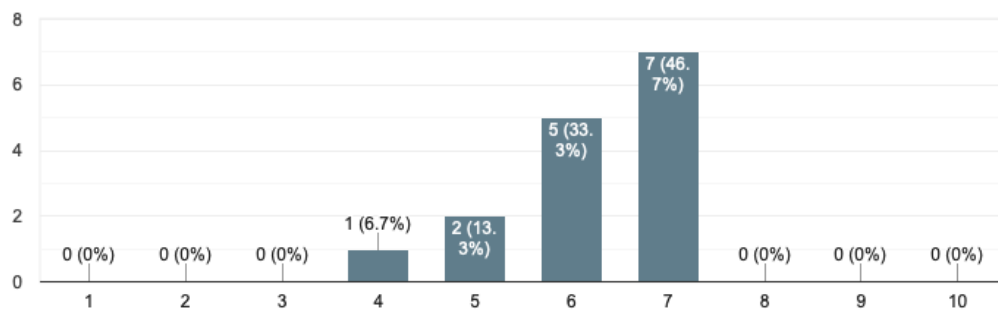
What communication tool does your team use currently?

15 responses



How satisfied are you with the chosen communication tool?

15 responses



Do you think productivity overall has suffered as a result of remote working?

15 responses

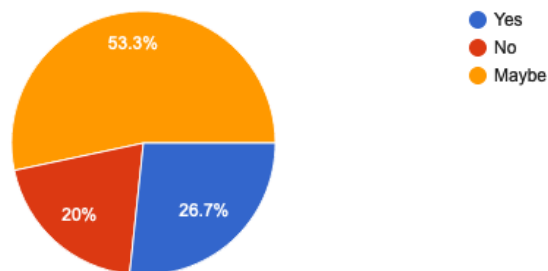


Figure 80 Survey Pt.2

Are you more or less inclined to help your teammates while working remotely?

15 responses

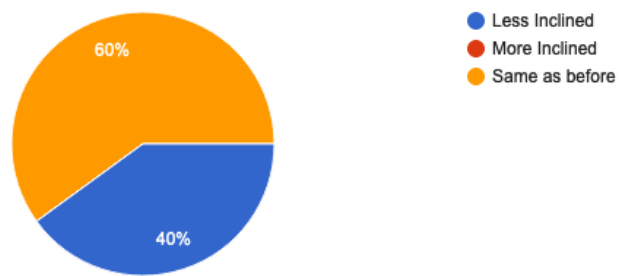


Figure 81 Survey Pt.3