

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: Kamil Lasecki
Student ID: X14100819
Supervisor: Muhammad Iqbal

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: Kamil Lasecki Date: 13/05/2018

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas

and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

Safe Boards: Technical Report

National College of Ireland

BSc in Computing

2017/2018

Kamil Lasecki

X14100819

X14100819@student.ncirl.ie

Safe Boards

Final Project Report



Table of Contents

Executive Summary	12
1 Introduction	13
1.1 Dictionary	13
1.2 Background	13
1.3 Aims	14
1.4 Technologies	15
1.5 Structure	15
2 Requirements	16
2.1 User Requirements Definition	16
2.2 Functional requirements	16
2.2.1 [FREQ-01] Authentication	16
2.2.2 [FREQ-02] Edit user's settings	17
2.2.3 [FREQ-03] Starting new post.....	17
2.2.4 [FREQ-04] Replying to the post	17
2.2.5 [FREQ-05] Replying to others response	17
2.2.6 [FREQ-06] Accessing closed group post.....	18
2.2.7 [FREQ-07] Accessing encrypted post.....	18
2.2.8 [FREQ-08] Changing privacy settings.....	18
2.2.9 [FREQ-09] Changing the encryption settings	18
2.2.10 [FREQ-10] Inviting user to access "closed group" post	19
2.2.11 [FREQ-11] Removing user from the "closed group"	19

Safe Boards: Technical Report

- 2.2.12 [FREQ-12] Requesting an access to the closed group post..... 19
- 2.2.13 [FREQ-13] Edit the post..... 19
- 2.2.14 [FREQ-14] Delete the post 19
- 2.2.15 [FREQ-15] Edit the comment 19
- 2.2.16 [FREQ-16] Delete the comment20
- 2.2.17 [FREQ-17] Voting.....20
- 2.2.18 [FREQ-18] Viewing replies hidden due low score20
- 2.2.19 [FREQ-19] Search for the post20
- 2.2.20 [FREQ-20] Access to notifications20
- 2.2.21 [FREQ-21] Access to user’s own posts21
- 2.2.22 [FREQ-22] Functionality to change displayed user name21
- 2.2.23 [FREQ-23] Functionality to change the password.....21
- 2.2.24 [FREQ-24] Functionality to add new categories.....21
- 2.2.25 [FREQ-25] Functionality to delete categories21
- 2.2.26 [FREQ-26] Functionality to move the post to another category21
- 2.2.27 [FREQ-27] Functionality to log out21
- 2.3 Non-Functional Requirements.....21
 - 2.3.1 [NFR-01] Performance/Response time requirement.....21
 - 2.3.2 [NFR-02] Availability requirement.....22
 - 2.3.3 [NFR-03] Robustness requirement22
 - 2.3.4 [NFR-04] Authentication.....22
 - 2.3.5 [NFR-05] Authorisation22
 - 2.3.6 [NFR-06] Reliability requirement23

Safe Boards: Technical Report

- 2.3.7 [NFR-07] Maintainability requirement23
- 2.3.8 [NFR-08] Easy access to search23
- 2.3.9 [NFR-09] Easy access to logout.....23
- 2.3.10 [NFR-10] XSS prevention23
- 2.3.11 [NFR-11] SQL infection prevention23
- 2.3.12 [NFR-12] Handling encryption phrase.....23
- 2.3.13 [NFR-13] Users list autocomplete24
- 3 Use cases 25
 - 3.1 [USE-01] Creating an account using Safe Boards [FREQ-01].....25
 - 3.2 [USE-02] User logging in [FREQ-01]27
 - 3.3 [USE-03] Linking social media account [FREQ-02]29
 - 3.4 [USE-04] Unlinking social media account [FREQ-02].....30
 - 3.5 [USE-05] New post creation [FREQ-03]31
 - 3.6 [USE-06] The user responds to the post [FREQ-04]32
 - 3.7 [USE-07] The user responds to the reply [FREQ-05]33
 - 3.8 [USE-08] The user accessing closed group post [FREQ-06]34
 - 3.9 [USE-09] The user accessing encrypted post [FREQ-07]35
 - 3.10 [USE-10] Post author changing the privacy setting [FREQ-08].....36
 - 3.11 [USE-11] Post author enabling encryption [FREQ-09]37
 - 3.12 [USE-12] Post author enabling encryption [FREQ-09]38
 - 3.13 [USE-13] An author of the post inviting user [FREQ-10]39
 - 3.14 [USE-14] Removing user’s access to the post [FREQ-11]40
 - 3.15 [USE-15] The user requests an access to closed group [FREQ-12]41

Safe Boards: Technical Report

- 3.16 [USE-16] An author edits the post [FREQ-13].....42
- 3.17 [USE-17] An author deletes the post [FREQ-14]43
- 3.18 [USE-18] The user edits his reply [FREQ-15]44
- 3.19 [USE-19] The user deletes his reply [FREQ-16]45
- 3.20 [USE-20] The user votes for the post or on replies [FREQ-17]46
- 3.21 [USE-21] The user viewing hidden messages [FREQ-18]47
- 3.22 [USE-22] User search for post [FREQ-19]47
- 3.23 [USE-23] The user accessing their notifications [FREQ-20].....48
- 3.24 [USE-24] The user accessing list his posts [FREQ-21]49
- 3.25 [USE-25] The user changing his display name [FREQ-22]50
- 3.26 [USE-26] The user changing his password [FREQ-23]51
- 3.27 [USE-27] An administrator adding new category [FREQ-24]52
- 3.28 [USE-28] An administrator deletes a category [FREQ-25]53
- 3.29 [USE-29] An administrator moves the post to different category [FREQ-26]
54
- 3.30 [USE-30] The user logs out from the session [FREQ-27].....54
- 4 Design and Architecture 56
 - 4.1 Data models - overview56
 - 4.2 Data models57
 - 4.2.1 Category data model.....57
 - 4.2.2 Notification data model58
 - 4.2.3 Post data model58
 - 4.2.4 Reply data model59
 - 4.2.5 User data model59

Safe Boards: Technical Report

- 4.3 Flow charts60
 - 4.3.1 New user registration:60
 - 4.3.2 Creating new post:61
 - 4.3.3 Opening an existing post62
 - 4.3.4 Editing the post.....63
- 5 Implementation 64
 - 5.1.1 [IMP-01] Authentication [FREQ-01].....66
 - 5.1.2 [IMP-02] Linking social media account [FREQ-02]67
 - 5.1.3 [IMP-03] Unlinking social media account [FREQ-02].....67
 - 5.1.4 [IMP-04] Starting new post [FREQ-03]67
 - 5.1.5 [IMP-05] Replying to the post [FREQ-04]68
 - 5.1.6 [IMP-06] Replying to the replies [FREQ-05]69
 - 5.1.7 [IMP-07] Accessing closed group post [FREQ-06].....69
 - 5.1.8 [IMP-08] Accessing encrypted post [FREQ-07].....70
 - 5.1.9 [IMP-09] Changing privacy settings [FREQ-08]71
 - 5.1.10 [IMP-10] Removing encryption from the post [FREQ-09]72
 - 5.1.11 [IMP-11] Adding an encryption to the post [FREQ-09].....73
 - 5.1.12 [IMP-12] Inviting users to closed group [FREQ-10].....74
 - 5.1.13 [IMP-13] Requesting access to the post [FREQ-12]75
 - 5.1.14 [IMP-14] Post edition [FREQ-13].....76
 - 5.1.15 [IMP-15] Post update [FREQ-13]76
 - 5.1.16 [IMP-16] Post deletion [FREQ-14].....76
 - 5.1.17 [IMP-17] Reply edit [REFQ-15].....77

Safe Boards: Technical Report

- 5.1.18 [IMP-18] Reply deletion {FREQ-16} 77
- 5.1.19 [IMP-19] Voting [FREQ-17]..... 78
- 5.1.20 [IMP-20] Hiding / un-hiding low score replies [FREQ-18] 79
- 5.1.21 [IMP-21] Post search [FREQ-19]..... 80
- 5.1.22 [IMP-22] Notifications [FREQ-20] 81
- 5.1.23 [IMP-23] Accessing own posts list [RFEQ-21]..... 82
- 5.1.24 [IMP-24] Changing display name [RFEQ-22] 82
- 5.1.25 [IMP-25] Changing master account password [RFEQ-23]..... 82
- 5.1.26 [IMP-26] Adding new category [RFEQ-24] 83
- 5.1.27 [IMP-27] Adding new category [RFEQ-25] 83
- 5.1.28 [IMP-28] Logout [RFEQ-27] 84
- 5.2 Graphical User Interface (GUI) Layout 84
 - 5.2.1 Authentication 84
 - 5.2.2 Managing account 85
 - 5.2.3 Create new post 86
 - 5.2.4 Single post view 87
 - 5.2.5 Viewing list of posts 89
 - 5.2.6 Category management 90
- 6 Testing..... 91
 - 6.1 User acceptance testing 91
 - 6.2 Integration testing 91
 - 6.3 End-to-end testing 91
 - 6.4 Customer testing 91

Safe Boards: Technical Report

- 6.5 Version control92
- 7 User manual..... 93
 - 7.1 Creating master account.....93
 - 7.2 Logging in with your created account93
 - 7.3 Login in with social media account.....94
 - 7.4 Creating new post94
 - 7.5 Accessing posts96
 - 7.6 Accessing closed group when not a member97
 - 7.7 Accessing encrypted post when member of the group97
 - 7.8 Replying to the post or comment98
 - 7.9 Inviting users to the closed group99
 - 7.10 Changing post privacy settings99
 - 7.11 Switching encryption on or off100
 - 7.12 Editing the post.....101
 - 7.13 Deleting post or comment.....102
 - 7.14 Editing comment.....102
 - 7.15 Displaying your posts102
 - 7.16 Notification103
- 8 Conclusions 104
- 9 Further development or research 105
- 10 References..... 106
- 11 Appendix 107

Safe Boards: Technical Report

11.1 Project Proposal107

11.2 Project Plan107

11.3 Monthly Journals107

11.4 Survey.....107

Executive Summary

We are living in the electronic data times when people generate more content than any time before. Internet users are eager to share their knowledge in the internet as well as seek the information in the internet. The Safe Boards aim to provide users with the platform, where users can discuss, share or ask questions on any topic, with minimum intervention from the administrators. Safe Boards is going to base the quality of the published posts and responses on the wisdom of crowd, by applying a voting system for both posts and replays. This way a content with poor quality can be hidden from users. The second most important point of the project is to secure user's privacy and anonymity. Whenever users decide that it is necessary, they should be able to secure their topics in the way that only users authorised by them can read their posts. An additional security can be applied by adding the encryption to the post. This will require users to distribute a secret key among themselves, but it will protect the post from being read by system administrator or even authorities. This way Safe Boards will be the boards to exchange the information but in the way that users can control their security and visibility of their posts.

1 Introduction

Safe Boards (SB) project is focus on providing the place to exchange an information in highly adjustable spectrum between sharing with public and sharing with only authorised users. Especially high pressure has been put on creating the functionality thanks to which no 3rd party can access the information, this is including anyone who has administrator access to the database. This functionality will require users to distribute a secret phrase between themselves, therefore it is not a default setting for every topic started on the SB, but rather a functionality which is available for user's disposal.

For the security reasons Safe Boards does not store an encryption phases and does not take a part of distributing them to other users. It is recommended that encryption phases are distributed using separate channel of communication such as text message or email that is not associated with the boards account.

The page does provide possibility to login with two major social media accounts, that is Facebook, and Google, as well as creating a Safe Boards personal account.

For the users' comfort, any of above accounts can be linked and unlinked at any time.

Deployed project can be accessed under:

<https://finalproject-kamillasecki.c9users.io/notifications>

1.1 Dictionary

Encryption phrase or **secret phase** is easy to remember, password like phase that is then used by the system to generate encryption key.

Closed Group post – a post with the security setting set to closed group – hidden or closed group – open. Both of which requires an author's invitation to access the post.

USP – unique selling point, a functionality that is unique to the application

1.2 Background

The Reddit is now 4th most popular site in the US, locating itself right after such giants like Google, YouTube and Facebook (Alexa.com, 2017). It gained its popularity from its anonymity and allowing freedom of speech. However, pages like Reddit are only

anonymous to some degree and lacking an access control, that is to whom the posts are visible or not.

The lack of anonymity issue.

For administrative purposes pages like Reddit stores users' IP addresses, as well as they are stored on the web server with very precise timestamp. This means that a page administrator would have to share that information to the authorities which has the right to ask for it. Knowing the IP address and a time will allow authorities to track down the address of property from where the comment or topic has been posted. Possible solution for users who would like to secure their IP exposure could be using the VPN, or Virtual Private Network. This works as a man in the middle, and all the traffic from the user is directed to the VPN provider and it is a VPN provider that is making the request to the server. Next the traffic is reversed, from server to VPN provider back to the user. This way server logs would show a VPN provider's IP address as an address from which connection has been made. Unfortunately, very often this may serve only as a delay of law enforcements. Even that VPN providers claiming they do not store users' logs, many cases show that this may not necessarily be the case for all of them. And if asked by authorities, they will have no other choice but to share those logs (Cimpanu, 2017).

The lack of privacy

Majority of the boards allows user to only post information that are accessible and can be read by all, or less often only by registered users.

1.3 Aims

To provide better anonymity and privacy, the Secure Boards allows user to choose who will be able to read the post. The user can choose from the following options:

Public – means all registered and unregistered users will be able to read this post.

Closed group - public – the post itself will be visible on the boards and it will appear in search, but access to read it will be only granted to invited users. This will allow users to request access to the post they are interested in, and after post author's verification the access can be granted or revoked.

Closed group - hidden – the post is not listed on the main boards and only invited users will be able to see it. In this case only post's author, or people promoted by the author, can invite other users.

Both private modes have an option to protect its content with an encryption phase that is to be shared only between the users and will be unknown to the boards administrator. Therefore, even in case that the administrator is requested to reveal those information, he will only have access to encrypted data and will be unable to decrypt it.

1.4 Technologies

The project has been built using Node.js platform which provides a JavaScript server-side functionality. The database which stores the user data as well as the boards content is the MongoDB. A non-relational, object-oriented database has been chosen due the fact that it handles better highly nested data as conversation history or the post / comments. The authentication has been handled by Passport using OAuth2.0 for access delegation for Google, and Facebook account holders.

The client side is served with HTML5. The real-time events are handled by Angular, with additional UI plugin called Angular Material. In addition, some of the page mechanics is served by jQuery library. Page formatting is provided mainly with use of Bootstrap3 and custom CSS stylesheets. Client-side encryption and decryption is done using a JavaScript library CryptoJS using AES with 256-bit key.

1.5 Structure

The document structure is as follows. This chapter contains executive summary on the application. Chapter 2 contains all functional and non-functional requirements marked [FREQ-XX] for functional requirements and [NFR-XX] for non-functional requirements. This sort of marking has been used for easy reference in the document. Next, in chapter 3 each requirement has been transformed into at least one use case those are marked [USE-XX]. Chapter 4 contains details on the design and architecture such as database structure or most important files and their role. And finally, chapter 5 contains detail on how each requirement has been implemented. Each of these is marked with [IMP-XX] and they all have a reference to their FREQ. Chapter 6 contained details on what testing were performed on the application and what was an outcome of these. Chapter 7 is the user manual, followed by Chapter 8 conclusions about the application, and Chapter 9 further development which contains all the functionality which I would like to implement in the future for which I had not enough time during the final year of my college.

2 Requirements

2.1 User Requirements Definition

The project required to have a following functionality to meet the user's needs.

Basic forum functionality.

- The user should be able to register new account or sign in with Google or Facebook account.
- The user should be able to add new posts.
- The user should be able to edit their posts.
- Unless someone already reply to the post, user should be able to delete it.
- The user should be able to search through all public and Closed Group – Open posts.
- The user should be able to reply to the public posts.
- The user should be able to reply to the Closed Group posts of which he is a member.
- The users should be able to like or dislike posts or replies.
- Replies with a negative vote number should be hidden but still accessible.

Functionality specific for this project

- The user should have a possibility to maintain their post privacy, that is whom is able to view their posts.
- The user should be able to hide the post from public view and assign users which will be able to read it.
- The user should be able to apply an encryption on his / her post.

2.2 Functional requirements

2.2.1 [FREQ-01] Authentication

The user should be able to create a new Safe Board local account or use his Google, or Facebook, authentication details. Upon the registration users should get random username assign to him to prevent any private information such as Facebook or Google user name or email. User should be able to change its username at any time if new username is not in use already.

2.2.2 [FREQ-02] Edit user's settings

A logged in user should be able to change his password as well as link or unlink his Safe Boards master account with Facebook or Google account. If accounts are linked, login in with any method will lead to the same master account.

2.2.3 [FREQ-03] Starting new post

Only logged in users can start a new post. When a new post is started the user should be ask for the subject and the post content, as well as a field where user can navigate through existing categories to choose the one best describes his post. Next the user should have an option to choose the security level of the post. The following options for the post privacy should be available: public, closed group – open, and closed group – hidden. Whenever one of the closed group security level is chosen, an additional option should become available to set an encryption on or off. When the encryption is enabled a new field for the encryption secret key should become visible. On this stage user should be clearly warn that the encryption key must be distributed to other users allowed to read this post in order to encrypt it, and that the key will never be send or stored by Safe Boards, therefore if lost it will not be possible to restore it. When the submit button is clicked all the fields should be validated and if there are any missing user should be clearly notified about it. If all fields are in order the post should be saved to the database and user should be redirected to his post. In the situation when the encryption key has been provided the user should be prompted to write down the encryption key and keep it in the safe place. Published post should capture the time it was created and the username of the owner, as well as all the privacy settings.

2.2.4 [FREQ-04] Replying to the post

Logged in users should be able to reply to Public posts and to the Closed Group to which they were granted an access. When viewing the post user should have a reply field available where he can write his reply and submit it. Once the reply is submitted the comment should appear on the page. Displaying the replied text, username of an author and when was the reply added.

2.2.5 [FREQ-05] Replying to others response

If the user is logged in and can access the post, it should be able to post the replies to the replies that refers to the main post. Each reply should have a reply field available where the user can write his reply to it and submit it. There should be only two levels of replies. That is; replies to the main post and replies to replies. No further replies should be allowed to avoid off-topic conversations. Those replies should be organised in a tree

structure where children replies are slightly shifted to the right relatively to their parent reply.

2.2.6 [FREQ-06] Accessing closed group post

When user is accessing Closed Group post its access should be validated. If the user can access the post the content should be loaded. If the user is not on the list of allowed users, this should be communicated with appropriate message, and form where he can request the access.

2.2.7 [FREQ-07] Accessing encrypted post

When the user accesses closed group post to which he is authorised, and the post is encrypted he should be presented with information about above and a field to provide the encryption phrase. If the phrase is correct an encrypted content of the post should be loaded. If the phrase is incorrect this should be communicated to the user with an error message should be displayed. post the post itself as well as all messages should be displayed.

2.2.8 [FREQ-08] Changing privacy settings

If logged in and user is an author of the post, he should be able to change the privacy settings of the post. A detail information on privacy levels should be available so the user is aware of pros and cons of each setting. Changing privacy from Closed group – hidden to Closed group – open and vice versa should only affect if the post is or is not visible on search list. When changing from one of the closed group settings to Public the user should be inform that all the access settings will be removed and if he wishes to change it bac he will have to resubmit all the permissions again. Also, when changing from Public to closed group setting the user should be inform about the fact that only allowed users will be able to access the post going forward. Once new settings are saved page should reload showing or hiding privacy specific settings and fields.

2.2.9 [FREQ-09] Changing the encryption settings

If logged in and user is an author of the post, he should be able to change the encryption settings of the post. A detail information on encryption should be available so the user is aware of pros and cons it. If the encryption has been disabled user should have option to enable it and to provide encryption phrase. When enabling encryption user should be inform that, once this is done users will need encryption phrase to access the post. Once new settings are saved page should reload.

2.2.10 [FREQ-10] Inviting user to access “closed group” post

If logged in and user is an author of the post, he should be able to invite other users, so they will be able to read and comment on the post. This function should be available on the post site and only to the author of the post. The user should be able to search through the users and send an invitation to the chosen one. Once invitation is send, an invited user should find the invitation in his notifications from where he can accept or decline an invitation.

2.2.11 [FREQ-11] Removing user from the “closed group”

The owner of the post should be able to remove users from the post that is visible to the “closed group” only. The owner should be able to view all the users that can view the post and remove any of them. Once user is removed he should receive a notification about that fact.

2.2.12 [FREQ-12] Requesting an access to the closed group post

If the post is closed group but listed, its subject will appear on users’ search. The user should be able to request access to the group which then will appear in author’s notifications, from where it can be accepted or declined. If the user access the closed group post it should be presented with form where he can provide a message field, this can contain a justification or message to the owner of the topic to help him make the decision.

2.2.13 [FREQ-13] Edit the post

The owner of the post should be able to fully edit the post but only until the first comment is added, after that the owner should be only able to update the original post by adding an edit note to it.

2.2.14 [FREQ-14] Delete the post

The owner of the post should be able to delete the post but only until the first comment is added.

2.2.15 [FREQ-15] Edit the comment

The users should be able to edit their comments at any time.

2.2.16 [FREQ-16] Delete the comment

The users should be able to delete their comments at any time. Deleted comments should not be displayed anymore unless someone else replied on it already, in this situation deleted comment should be replaced by text: "This comment has been removed by its author".

2.2.17 [FREQ-17] Voting

Logged in users should be able to vote for the replies or posts by up voting or down voting them. Replies with negative number of votes should be hidden and replaced with the "This comment has been hidden due its poor content" message. The user then should be able to click on this message and reveal the comment. The user should be only able to vote for each item once but should be able to change his vote at any time. A vote score should be visible to the users allowing them to evaluate the response or the post.

2.2.18 [FREQ-18] Viewing replies hidden due low score

An application should hide replies which have more negative than positive votes, however, users should be able to still view those if wanted.

2.2.19 [FREQ-19] Search for the post

The user whenever logged in or not should be able to search the boards. The search results should include the posts that contain the searched phrase in the subject or in the post body. The results should also display subject, time when post has been added, number of comments, attributes, vote scores and category of the post.

2.2.20 [FREQ-20] Access to notifications

Logged in user should have easy access to their notifications from any place on the page. In addition any new notifications should be clearly indicated. In the notification area users should be able to interact with their notifications that is: accept or decline both post invitations and post access requests. As for the other notifications such as information about invitation or request being rejected or accepted, user should be able to discard them. In addition, notification which were not yet read should be clearly indicated to the user.

2.2.21 [FREQ-21] Access to user's own posts

User should be able to display only the posts that he is an author of. This should provide the same information as REQ-19

2.2.22 [FREQ-22] Functionality to change displayed user name

User should be able to change his displayed user name. This is separate from the username used to login to the page and is used to sign user's posts and replies. When display name is changed it should be used to sign all new, as well as those added in past posts and replies.

2.2.23 [FREQ-23] Functionality to change the password

The user should be able to change his password. By providing old and new password.

2.2.24 [FREQ-24] Functionality to add new categories

The page administrator should be able to add new categories to the category tree.

2.2.25 [FREQ-25] Functionality to delete categories

The page administrator should be able to delete categories to the category tree.

2.2.26 [FREQ-26] Functionality to move the post to another category

The page administrator should be able to move post from one category to the category which in his opinion better suits to its subject or to the new category created to accommodate subject which didn't existed previously.

2.2.27 [FREQ-27] Functionality to log out

User should be able to logout his session from any place on the website.

2.3 *Non-Functional Requirements*

2.3.1 [NFR-01] Performance/Response time requirement

The posts and comments added or edited should be visible to the users right after being sent.

2.3.2 [NFR-02] Availability requirement

By initial release, the Safe Boards shall provide users with a minimum operational availability of 99%. Going forward with next releases the system shall provide users with a minimum operational availability of 99.9%.

2.3.3 [NFR-03] Robustness requirement

The system should handle invalid inputs without crashing. Safe boards should detect invalid input and notify user and should not accept invalid inputs to be saved to the database.

2.3.4 [NFR-04] Authentication

The system should be able to verify the identity of users. Users that are not authenticated should be treated as guests and only have access to posts search and reading public posts.

2.3.5 [NFR-05] Authorisation

The system should only perform actions that are allowed for current user status:

Non-registered user:

1. Viewing the public posts
2. Searching forum
3. Creating account

Registered user:

1. Searching forum
2. Reading public posts
3. Reading closed group posts when the user is listed in that group
4. Creating new post
5. Maintaining post when the logged-in user is the author of the post
6. Adding comments to the public posts
7. Adding comments to the closed group posts when the user is listed in that group
8. Editing the comments when the logged-in user is the author of the comment
9. Voting on the posts

Administrator:

1. Create new category
2. Delete the category
3. Move post to another category

2.3.6 [NFR-06] Reliability requirement

The mean time between the failures should exceed 2 months.

2.3.7 [NFR-07] Maintainability requirement

All updates to the system should be backward compatible and all maintenance should be handles without down time.

2.3.8 [NFR-08] Easy access to search

The users should have easy access to search functionality regardless which section of the page he is accessing

2.3.9 [NFR-09] Easy access to logout

The users should have easy access to logout functionality regardless which section of the page he is accessing

2.3.10 [NFR-10] XSS prevention

The website should handle user inputs in the way that under no circumstances users input should be executed as JavaScript

2.3.11 [NFR-11] SQL infection prevention

The website should handle all user inputs in the way that it is not possible to force the database to return more data than it is intended to.

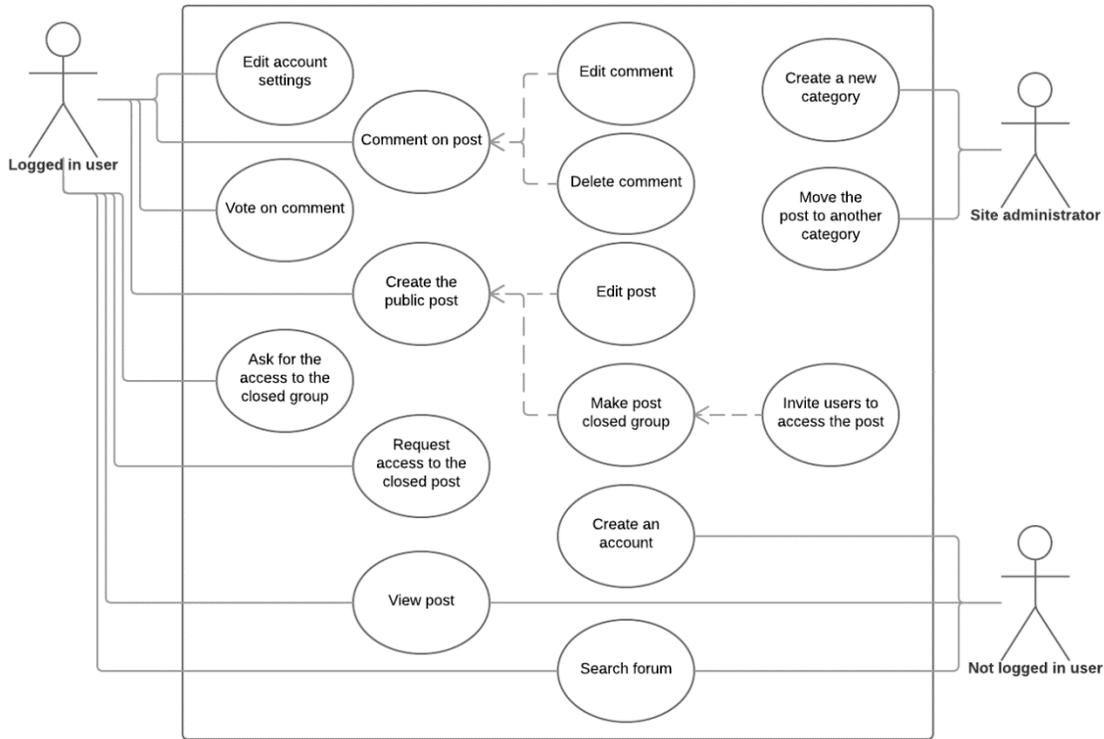
2.3.12 [NFR-12] Handling encryption phrase

User's encryption phrase should never be passed to the server side. All encryption and decryption should be handled on the front end. This is USP, allowing users to have a full control over who can read it the content of encrypted messages.

2.3.13 [NFR-13] Users list autocomplete

When searching for the user to invite for the closed group user field should support autocomplete to allow users to easily search for the user.

3 Use cases



3.1 [USE-01] Creating an account using Safe Boards [FREQ-01]

Description: User registration is a key component that must be implemented to separate each user’s boundaries, in other words that each user can only edit or access components which he is authorized to edit or access, and that system can confirm identity of the user. Safe boards should allow user to create Safe Boards specific account or login using their Google or Facebook account.

Priority: high

Scope: The scope of this use case is to provide user with the area on the website where the user will be able to create a new account.

Precondition: The user is not logged in.

Activation: Users clicks on the “Register” button available when browsing the boards when not logged in.

Main flow:

1. The user opens the login page
2. The user follows to the registration page
3. The application displays registration form
4. The user populates the username field on the form
5. The application checks if the username is not already taken [E1]
6. The user provides the password
7. The application checks if the password meets the minimum requirements [E2]
8. The user repeats the password
9. The application checks if both passwords match [E3]
10. The user provides the email address [A1]
11. The application checks if the email format is appropriate [E4]
12. The application enables the "Submit" button
13. The user clicks the "Submit" button
14. The application registers a new user
15. The application loads login page.

Alternate flow:

A1: Missing email address

1. The user does not provide the email address.
2. The system displays the warning about lack of possibility to recover the password if it is lost.
3. The use case continues at position 12 of the main flow

Exceptional flow:

E1: The username is already taken

1. The user provides username that is already taken
2. The application displays a warning that the user should pick another username
3. The use case goes back to the position 4 of the main flow

E2.1: The password is too short

1. The user provides password that is shorter than 6 characters
2. The application displays a warning that the user should pick longer password
3. The use case goes back to the position 6 of the main flow

E2.2: The password is not alphanumeric

Safe Boards: Technical Report

1. The user provides username that is not alphanumeric
2. The application displays a warning that the user should pick password that contains letters and numbers
3. The use case goes back to the position 6 of the main flow

E3: Passwords don't match

1. Second password does not match the first one
2. The application displays a warning that both passwords should be identical
3. The use case goes back to the position 8 of the main flow

E4: Email format incorrect

1. The user provides the email address that is in invalid format
2. The application displays a warning that email address should be in a format name@domain.xx
3. The use case goes back to the position 10 of the main flow

Termination: The user presses the "Submit" button.

Post condition: The system registers a new user and loads login page.

3.2 [USE-02] User logging in [FREQ-01]

Description: User login is a key component that must be implemented to separate each user boundaries. This is to allow the system to confirm user's identity, to allow access to only those areas which should be accessible by this user.

Priority: High

Scope: The scope of this use case is to provide user with the area on the website where the user will be able confirm its identity.

Precondition: The user is not logged in.

Activation: Users clicks on the "Login" button

Main flow:

1. The user clicks on the "Login" button
2. The application displays login form
3. The user populates the username field on the form [E1, A1, A2]

Safe Boards: Technical Report

4. The user populates the password field on the form [E2]
5. The user clicks “Login” button
6. The application validates if the username and password are correct [E3, E4]
7. The application is loading main page with user logged in

Alternate flows:

A1: User authenticates with Facebook

1. The user clicks Facebook authentication button
2. The application displays Facebook’s authentication form
3. The use case continues at position 3 of the main flow

A2: User authenticates with Google

1. The user clicks Google authentication button
2. The application displays Google authentication form
3. The use case continues at position 3 of the main flow

Exceptional flow

E1: Missing username

1. The user does not provide the username
2. The application displays the warning message about missing username
3. The use case goes back to the position 4 of the main flow

E2: Missing password

1. The user does not provide the password
2. The application displays the warning message about missing password
3. The use case goes back to the position 5 of the main flow

E3: Invalid login

1. The login provided by the user does not exist
2. The application displays the warning that the username or password are incorrect
3. The use case goes back to the position 3 of the main flow

E3: Invalid password

1. The password provided by the user does not match the one saved in the system

2. The application displays the warning that the username or password are incorrect
3. The use case goes back to the position 3 of the main flow

Termination: The user presses the “Login” button.

Post condition: An Application authenticates the user.

3.3 [USE-03] Linking social media account [FREQ-02]

Description: The users would like to link his Google or Facebook account to his Safe Boards master account. This will allow user to easier, one click login.

Priority: Low

Scope: The scope of this use case is to provide user with the area on the website where the user will be able to link his Google or Facebook account.

Precondition: The user must have Safe Boards Master account as well as Google or Facebook account.

Activation: Users clicks on the “Account settings” link

Main flow:

1. The user clicks on the “Account settings” button
2. The application displays account setting page.
3. The user clicks link Google account [A1]
4. An application loads Google login page
5. User provides login details
6. User agrees to what Google account details will be sheared with Safe Boards [E1]
7. An application loads the account setting page.
8. Name of the linked account is now displayed on the page

Alternative flow

A1: User uses his Facebook account:

1. The user clicks link Facebook account
2. An application loads Facebook login page
3. User provides login details
4. User agrees to what Facebook account details will be sheared with Safe Boards [E1]

5. The use case continues at position 7 of the main flow

Exceptional flow

E1: User disagree with the details that are going to be shared with Safe Boards

1. The user cancels login to social media account
2. Safe Boards returns to the account settings page

Termination: Name of the linked account is now displayed on the page

Post condition: Social media account has been liked and login in with it will take user to his Safe Boards master account.

3.4 [USE-04] Unlinking social media account [FREQ-02]

Description: The users would like to unlink his Google or Facebook account from his Safe Boards master account.

Priority: Low

Scope: The scope of this use case is to provide user with the area on the website where the user will be able to unlink his Google or Facebook account.

Precondition: The user must have his Google or Facebook account linked with Safe Boards master account.

Activation: Users clicks on the “Unlink” link next to the linked account.

Main flow:

1. The user clicks “unlink” button next to his social media account
2. An application displays a confirmation prompt to confirm an action
3. The user confirms the prompt [E1]
4. An application unlinks chosen social media account
5. An application reloads the account settings page
6. Unlinked account is no longer displayed on the page.

Exceptional flow

E1: User does not confirm the prompt

1. An application closes the prompt

Termination: The user confirms the willingness to unlink the account

Post condition: Chosen social media account is no longer linked to the Safe boards master account.

3.5 [USE-05] New post creation [FREQ-03]

Description: Posts and their replies will be a main content of the application therefore adding new posts is a key feature.

Priority: High

Scope: The scope of this use case is to add a new post to the system.

Precondition: User must be logged in

Activation: The user accesses new post form

Main flow:

1. The user accesses new post form
2. The user fills in the post subject input
3. The user fills in the post message input
4. The user picks the category of the post [A1]
5. The user sets the privacy level to public [A2, A3]
6. The user submits the form
7. The system validates the fields [E1, E2]
8. The system publishes the post.
9. The system loads newly added post

Alternate flow:

A1: Default category

1. The user was unable to find a matching category
2. The user leaves the category default setting: "Main"
3. The use case continues at position 6 of the main flow

A2: Closed group – public post

Safe Boards: Technical Report

1. The user sets the privacy to closed group - public
2. The user search users which should have access to the post.
3. The user sets the encryption (See A4)
4. The user sets the encryption secret.
5. The use case continues at position 7 of the main flow

A3: Closed group – hidden post

1. The user sets the privacy to closed group - hidden
2. The user search users which should have access to the post.
3. The user sets the encryption (See A4)
4. The user sets the encryption secret.
5. The use case continues at position 7 of the main flow

A4: No encryption

1. The user does not set the encryption available.
2. The use case continues at position 7 of the main flow

Exceptional flow:

E1: Missing subject

1. The user did not provide the post subject
2. The application displays the error message about missing subject

E2: Missing text

1. The user did not provide the post text
2. The application displays the error message about missing text

Termination: The application publishes the post.

Post condition: The application loads the page with the posts category matching one that has been chosen by the user.

3.6 [USE-06] The user responds to the post [FREQ-04]

Description: Posts and their replies will be a main content of the application therefore allowing user to post his comment under the post is another key function.

Priority: High

Scope: The scope of this use case is to allow user to post a reply under the post.

Precondition: User must be logged and have a post page open.

Activation: The user starts typing the reply to the post.

Main flow:

1. The user starts typing a reply in provided for this purpose text input field.
2. An application makes button for submitting a reply available
3. The user submits the reply
4. An application validated the reply [E1]
5. An application saves the reply
6. An application refreshes the content of the page displaying newly added reply.

Exceptional flow

E1: Users reply is empty

1. An application displays warning message about the fact that the reply is empty
2. Termination

Termination: The user did not provide the reply message.

Post condition: Submitted reply is visible on the page.

3.7 [USE-07] The user responds to the reply [FREQ-05]

Description: The user wishes to reply to previously posted response to the main post.

Priority: Medium

Scope: The scope of this use case is to allow user to post a reply under other replies, but only under those replies that responds to the main post. Therefore, replying to second or higher-level replies should not be allowed.

Precondition: User must be logged and have a post page open.

Activation: The user starts typing the response to earlier reply.

Main flow:

1. The user starts typing a reply in provided for this purpose text input field.
2. The user submits the reply
3. An application validated the reply [E1]
4. An application saves the reply in the database
5. An application refreshes the content of the page displaying newly added reply.

Exceptional flow

E1: Users reply is empty

3. An application displays warning message about the fact that the reply is empty
4. Termination

Post condition: Submitted reply is visible on the page.

3.8 [USE-08] The user accessing closed group post [FREQ-06]

Description: Since some of the posts may be available only to closed group users but still listed on the search results (closed group – public) it is important to provide users with functionality to request access to those posts. The user will be only able to request access to the ‘closed group – open’ posts as ‘closed group – hidden’ posts should be hidden from user.

Priority: medium

Scope: The scope of this use case is to provide handling the user access to the closed group post.

Precondition: User must be logged in and accessing post with ‘closed group – open’ privacy setting.

Activation: The user opens the post.

Main flow

1. The user is accessing closed group post
2. An application validates if the user is on the list of users allowed to access this post [E1]
3. An application loads the post content [A1]

Alternative flow

A1: The post is encrypted: followed up in [USE-10]

Exceptional flow

E1: User is not authorised to access the post

1. An application displays the message about the fact that user is not authorised to access the post, together with the form to request an access
2. Flow continue in [USE-12]

Termination: User have no access to the post.

Post condition: The user can access the post.

3.9 [USE-09] The user accessing encrypted post [FREQ-07]

Description: One of the project USP's is ability to encrypt the post in the way that only users knowing the encryption phrase will be able to read it. User wanting to access such a post should be presented with the suitable message and a field to provide an encryption phrase.

Priority: High

Scope: The scope of this use case is to provide handling the user access to the encrypted post.

Precondition: User must be logged in and accessing post with 'closed group – open' privacy setting that is encrypted. The user must be authorised to access the post.

Activation: The user opens the post.

Main flow

1. The user is accessing closed group post that is encrypted.
2. An application displays the message that the post is encrypted and asks for the secret phase used for encryption
3. The user inputs the secret phrase and submits it
4. An application validates the encryption phrase [E1]
5. An application decrypts the post content
6. An application displays the post content to the user

Exceptional flow

E1: Provided phrase does not match

1. Encryption phrase provided by the user is not valid
2. An application displays the warning message about the fact that the provided phrase seems to be wrong
3. The use case continues at position 2 of the main flow

Post condition: The user can access the post.

3.10 [USE-10] Post author changing the privacy setting [FREQ-08]

Description: One of the USP of the application is possibility to adjust post privacy so it matches user's needs. As those needs may change overtime users should be able to change the privacy of their posts after they are already functional.

Priority: Medium

Scope: The scope of this use case is to provide user with functionality to change privacy settings on existing post.

Precondition: The user must be logged in and accessing the post which he has created.

Activation: The user opens post's settings.

Main flow

1. The user opens post's settings
2. An application presents user with option to change post's privacy settings as well as with detailed information on each setting
3. The user Picks a new privacy setting
4. [A1, A2, A3]
5. An application reloads the page

Alternative flow

A1: The post original setting was closed group and user is changing to another closed group post:

1. An application updates the privacy settings of the post

Safe Boards: Technical Report

2. An application displays a confirmation message that the change has been completed
3. The use case continues at position 5 of the main flow

A2: The post original setting was private, and user is changing to one of the closed groups post:

1. An application informs user about the fact that other users to access the post will have to be granted an access and asks for the confirmation.
2. The user confirms the prompt
3. An application updates the privacy settings of the post
4. An application displays a confirmation message that the change has been completed
5. The use case continues at position 5 of the main flow

A3: The post original setting was one of the closed groups, and user is changing to private:

1. An application informs user about the fact that going forward all users will be able to access and comment on the post and asks for the confirmation.
2. The user confirms the prompt
3. An application wipes out all existing access rights from the post
4. An application deletes all pending invites to the post
5. An application deletes all pending request to access the post.
6. An application updates the privacy settings of the post
7. An application displays a confirmation message that the change has been completed
8. The use case continues at position 5 of the main flow

Termination: The user cancels his choice when prompted for the confirmation

Post condition: The privacy settings of the post has been changed.

3.11 [USE-11] Post author enabling encryption [FREQ-09]

Description: One of the USP of the application is possibility to choose rather closed group post is encrypted or not as per author's requirements. As those requirements may change overtime users should be able to turn the encryption on or off whenever required.

This can also serve as utility to change the encryption phrase as user could remove encryption and apply it with a new phrase.

Priority: Medium

Scope: The scope of this use case is to provide user with functionality to enable an encryption on the existing post.

Precondition: The user must be logged in and accessing closed group post which he has created, and which is not using encryption.

Activation: The user opens post's settings.

Main flow

1. The user opens post's settings
2. An application presents user with option to enable encryption, detailed information about it, and the field to provide encryption phrase.
3. The user provides the encryption phrase.
4. The user enables the encryption [E1]
5. An application encrypts the post and all the replies
6. An application reloads the page

Exceptional flow

E1: the user did not provide an encryption phrase:

1. An application presents a warning about encryption phrase missing
2. The use case continues at position 2 of the main flow

Termination: The user closes post's settings window

Post condition: An encryption has been enabled.

3.12 [USE-12] Post author enabling encryption [FREQ-09]

Description: One of the USP of the application is possibility to choose rather closed group post is encrypted or not as per author's requirements. As those requirements may change overtime users should be able to turn the encryption on or off whenever required. This can also serve as utility to change the encryption phrase as user could remove encryption and apply it with a new phrase.

Priority: Medium

Scope: The scope of this use case is to provide user with functionality to disable an encryption on the existing post.

Precondition: The user must be logged in and accessing closed group post which he has created, and which is using encryption.

Activation: The user opens post's settings.

Main flow

1. The user opens post's settings
2. An application presents user with option to disable encryption, and detailed information about it.
3. The user disables the encryption
4. An application encrypts the post and all the replies
5. An application reloads the page

Termination: The user closes post's settings window

Post condition: An encryption has been disabled.

3.13 [USE-13] An author of the post inviting user [FREQ-10]

Description: Since all closed group posts will be available only to invited users, an author of the post must have a possibility to invite those users. This is a one of two ways users can gain the access to the closed group -open post, with second option being users requesting an access [FREQ-12], and the only way of inviting users to the closed group -hidden post

Priority: High

Scope: The scope of this use case is to allow an author of the post to invite other users to access the post.

Precondition: User must be logged in and user must be the owner of the post.

Activation: An author of the post searches for the user

Main flow

1. An author search for the username that interest him
2. An author picks the user from the search list
3. An author sends the invitation [E1, E2]
4. An application generates a new notification for the user
5. The flow continues in [USE-20]

Exceptional flow

E1: An author provides invalid user

1. An application presents an author with the warning message that username which he provided does not exist.
2. Termination

E2: An author picks the user which already have access to the post

1. An application presents an author with the warning message that user already has access to the post
2. Termination

E2: An author picks the user which already have been invited

1. An application presents an author with the warning message that user already has been already invited
2. Termination

Post condition: An invitation has been sent to the user

3.14 [USE-14] Removing user's access to the post [FREQ-11]

Description: Since all closed group posts will be available only to invited users, an author of the post must have a possibility to also revoke an access to the post at any stage, to any user currently allowed to access it.

Priority: High

Scope: The scope of this use case is to allow an author of the post to remove access rights to the post for chosen user.

Precondition: User must be logged in and user must be the owner of the post. The post must have at least one user allowed to access the post that is other than the owner.

Activation: Access post's settings

Main flow

1. An author of the post access post's settings
2. An application presents user with settings among which an author can review all users currently having access to the post.
3. An author deletes one of the users from the list.
4. An application presents an author with the confirmation prompt
5. An author confirms the prompt
6. An application removes user from the list of allowed users
7. An application creates a notification to the user that his access to the post has been removed.
8. An application refreshes the post view for an author.

Termination: An author closes the settings window or does not confirm the prompt in step 5.

Post condition: The user has been removed from the list of people allowed to access the post

3.15 [USE-15] The user requests an access to closed group [FREQ-12]

Description: Since all closed group posts will be available only to invited users, and invitations from an author to the user were covered in [FREQ-10] it is also required that users can communicate a will to access closed group – open posts. When requesting an access, user should be able to attach a short message to the request. However, providing such is not mandatory.

Priority: High

Scope: The scope of this use case is to allow the user to request an access from the author of the post.

Precondition: User must be logged in and there must be at least one closed group – open post to which use has no access to.

Activation: Accessing closed group – open type post

Main flow

1. The user finds closed group - open type post, the which he does not have an access
2. The user opens the post
3. An application validates users access and finds that he is not allowing to view the content of the post
4. An application displays a message that the user is not allowed to access content of this post.
5. An application opens a form where user can provide a message to the author of the post and request an access.
6. The user fills the message to the author (optional)
7. The user submits the request.
8. An application creates a new request notification for the author of the post.
9. An application displays the message to the user that the request has been created.

Termination: The user closes the page.

Post condition: A new notification to the author of the post with access request has been created.

3.16 [USE-16] An author edits the post [FREQ-13]

Description: The post content should be editable for the author but only until the first response is added, going forward an author can only update the post in the form of notes added to the post content.

Priority: Medium

Scope: The scope of this use case is to provide an author with functionality to edit the post.

Precondition: User must be logged in and be an author of the post.

Activation: An author enters the edit mode of the post.

Main flow

1. An author enters the edit mode of the post
2. An application replaces the text area with editable text box filled with the post original text. [A1]
3. An author makes amendments to the text
4. An author submits the changes [E1]

Safe Boards: Technical Report

5. An application updates the post content
6. An application replaces the editable text box with the updated text

Alternate flow

A1: Post already have comments added

1. The application displays information that post cannot be edited and only updated.
2. The application displays empty editable text field underneath the original post
3. The user types in the update for the post
4. The user clicks “Update” button (See E1)
5. The application appends the original post text with an update and the date on which the update has been added

Exceptional flow

E1: Content of update is empty

1. The application displays warning message that the update cannot be empty.

Termination: An author cancels the edition before submitting.

Post condition: The post’s new content is visible on the website

3.17 [USE-17] An author deletes the post [FREQ-14]

Description: An author should be able to delete the post but only until the first comment is posted. After that post deletion should not be possible. If the post has already any comments deletion option should not be possible, hence no alternative flow for the post with comments.

Priority: Medium

Scope: The scope of this use case is to provide an author with functionality to delete the post.

Precondition: User must be logged in and be an author of the post. The post should not have any replies.

Activation: An author process to delete the post.

Main flow

1. An author process to post deletion available on the post page
2. An application prompts an author for the confirmation of deletion
3. An author confirms the prompt
4. An application deletes all invitation and requests to access if any exists
5. An application deletes the post.
6. An application displays information box about post successful deletion
7. An application loads main page.

Termination: An author cancels the deletion in step 3.

Post condition: Chosen post has been removed

3.18 [USE-18] The user edits his reply [FREQ-15]

Description: The user should be able to edit his post content at any time. Each reply created by the user should have a functionality allowing user to edit a reply text.

Priority: Medium

Scope: The scope of this use case is to provide the user with functionality to edit the replies to the post as well as replies to other replies if they were created by the user itself.

Precondition: User must be logged in have at least one reply posted.

Activation: The user attempts to reply edition.

Main flow

1. The user access the post to which he earlier replied.
2. The user finds his reply
3. User attempts to reply edition.
4. An application replaces reply text with editable input pre-populated with an original text of the reply.
5. The user edits the text
6. The user submits the changes
7. An application saves new input
8. An application reloads the content of the post displaying newly edited reply

Alternate flow

A1: Empty text box

1. The user leaves empty text box
2. The application displays the warning that the textbox cannot be empty.

Termination: The user clicks “Save” button

Post condition: The comment’s new state is visible on the website

3.19 [USE-19] The user deletes his reply [FREQ-16]

Description: The user should be able to delete his comment at any time. Deleted comment should be removed from the post unless they were replied directly to the post and user has replied to it. In the situation that the reply has another child-replies under itself, a deletion should not remove the post as those child-replies will have to stay attached to something. Instead the comment that user is willing to remove should be replaced with the message informing that it has been removed.

Priority: Medium

Scope: The scope of this use case is to provide the user with functionality to delete his comment.

Precondition: The user must be logged in and must have reply to at least one post.

Activation: The user proceeds to reply deletion.

Main flow

1. The user proceeds to reply deletion
2. An application displays a prompt for confirmation
3. User confirms the prompt
4. An application deletes reply [A1]
5. An application refreshes the post with reply being removed.

Alternate flow

A1: Another user replied to the deleted response.

1. An application marks the reply as deleted

2. An application refreshes the post with an information that reply has been deleted in the place where the reply was.
3. The flow is terminated

Termination: Users deny confirmation in step 3

Post condition: The post is deleted or is marked as deleted

3.20 [USE-20] The user votes for the post or on replies [FREQ-17]

Description: An application should allow users to vote if the post or reply is useful by liking it or disliking it. User should be only able to vote once on each item, and it should be able to change his vote at any time.

Priority: Medium

Scope: The scope of this use case is to provide the user with functionality to vote on post items that is post itself and its replies.

Precondition: The user must be logged in.

Activation: The user votes on item.

Main flow

1. The user votes on the item
2. An application adds or subtracts 1 to/from the score of the item [E1] [A1]
3. An application refreshes the score [A2]

Alternate flow

A1: User already voted to this item but with opposite intend (previously downvoted when now upvoting or vice versa).

1. An application subtracts or adds 2 from/to the post score
2. An application refreshes the score

A2: Item is a reply item and its score reached negative number

1. An application refreshes the score
2. An application hides the reply

Exceptional flow

E1: User already voted to this item it the same way.

1. An application displays warning message that user already voted on this item
2. Termination

Termination: User votes on the item for the second time in the same way

Post condition: An item score has changed

3.21 [USE-21] The user viewing hidden messages [FREQ-18]

Description: If any reply will reach negative score (will receive more dislikes than likes) it will get hidden [USE-21 / A2] but with possibility to revel it by the user.

Priority: Medium

Scope: The scope of this use case is to provide the user with functionality to revel replies which has been hidden due negative score.

Precondition: The user must be logged in and he must approach reply which received more negative than positive scores.

Activation: The revels hidden reply.

Main flow:

1. The revels hidden reply.
2. An application replaces information about hidden message due low score with the message itself.

Post condition: Hidden message is now visible.

3.22 [USE-22] User search for post [FREQ-19]

Description: Post search is a key component that must be implemented to allow users to find interesting them information. The search box should be accessible from any page and its functionality is to return to the user a list of posts that are relevant to the search phrase.

Priority: High

Scope: The scope of this use case is to allow user to search through all the posts and filter them using a search phrase.

Precondition: No preconditions

Activation: Users attempt to search for the post

Main flow:

1. The user types in the search phrase into the search field
2. An application returns list of posts that matches searched phrase [A1]

Alternate flow

A1: No match was found

1. There was not match to the provided phrase
2. The application displays message that no posts were found for provided phrase

Post condition: The user is presented with list of posts meeting search criteria.

3.23 [USE-23] The user accessing their notifications [FREQ-20]

Description: A notification is a form of communication between the users. Notifications are generated by the system but as a result on user interaction with other users' elements such a post or interacting directly with the user. Users should be able to tell if they have a new notification as well as access to all their notifications. Users should be able to interact with the notifications. The user should be able to discard the notification or accept it if suitable to the notification type.

Priority: Medium

Scope: The scope of this use case is to provide user with an area where user can access and interact with his notifications.

Precondition: The user must be logged.

Activation: Users access the notifications area.

Main flow:

1. The user access the notification area

Safe Boards: Technical Report

2. The user picks one of the notifications
3. The system displays more detailed version of notification
4. The user discards the notification [A1, A2]
5. The system removes the notification

Alternate flow:

A1: The notification type allows user to approve notification

1. The user approves the notification
2. The system removes the notification
3. The system generates a new notification to user who sent approved notification, with an information that it has been approved.

A1: The notification type allows user to approve notification

1. The user discards the notification
2. The system removes the notification
3. The system generates a new notification to user who sent discarded notification, with an information that it has been discarded.

Termination: The user has not active notifications or the user leaves the page.

Post condition: The user interacted with notification.

3.24 [USE-24] The user accessing list his posts [FREQ-21]

Description: The user should be able to easily display a list of all the post that he has created. This will allow users to more easily maintain their posts.

Priority: Medium

Scope: The scope of this use case is to provide user area on the page that he can display all the post of which he is an author.

Precondition: The user must be logged in and must have created at least one post.

Activation: Users access their posts area.

Main flow:

1. The user access their posts area

2. The system displays a list of posts in the same manner as search result [FREQ-19]

Termination: The user leaves the page.

Post condition: The user is presented with a list of his posts.

3.25 [USE-25] The user changing his display name [FREQ-22]

Description: When creating a new account, the user receives unique generic display name to not reveal any personal information such as username or email address. The display name is used to identify user for others when publishing any content. The user should be able to change this their username at any time.

Priority: Medium

Scope: The scope of this use case is to provide user with functionality to change their display name.

Precondition: The user must be logged.

Activation: The user accesses his account settings.

Main flow:

1. The user accesses his account settings.
2. The user enters the display name change mode.
3. The user provides new username in designated field
4. The user submits new display name
5. The system validates new display name [E1]
6. The system updates user's display name

Exceptional flow:

E1: User's new display name is already being used by another user

1. The system displays a warning message that chosen display name is already taken.
2. The flow resumes at step 2.

Termination: The user leaves the page.

Post condition: User's display name has been updated.

3.26 [USE-26] The user changing his password [FREQ-23]

Description: When creating and using Safe Boards master account the user uses their private password to authenticate himself. The user should be able to change his master account password at any time.

Priority: Medium

Scope: The scope of this use case is to provide user with functionality to change their master account password.

Precondition: The user must be logged in with a master account.

Activation: The user accesses his account settings.

Main flow:

1. The user accesses his account settings.
2. The user enters the password change mode.
3. The user provides old password in designated field
4. The user provides new password in designated field
5. The user repeats new password in designated field
6. The user submits his new password
7. The system validates the passwords [E1, E2]
8. The system updates the password
9. The system logs out user's session.
10. The system loads login page

Exceptional flow:

E1: User's new password does not meet the minimum requirements

1. The system displays a warning message that new password does not meet the minimum requirements and display the requirements.
2. The system clears old and new password fields
3. The flow resumes at step 3.

E2: User's new password and new password repeat does not match

1. The system displays a warning message that new password and new password repeat does not match.
2. The system clears old and new password fields

3. The flow resumes at step 3.

Termination: The user leaves the page.

Post condition: User's password has been changed.

3.27 [USE-27] An administrator adding new category [FREQ-24]

Description: Each post must be assign to one of the categories. The category should be assigned to another category unless it a root category. An administrator should be able to create a new category as required.

Priority: High

Scope: The scope of this use case is to provide an administrator with functionality to create new categories.

Precondition: The user must be logged in and must be an administrator type.

Activation: An administrator enters the category management area

Main flow:

1. An administrator enters the category management area.
2. An administrator navigates to the category under the new one will be created
3. An administrator provides the name of new category
4. An administrator submits new category
5. The system validates new category [E1]
6. The system updates the category tree
7. The system refreshes the page view

Exceptional flow:

E1: The category with the exact same name already exists under that category.

1. The system displays a warning message that new category is the same as one that already exists
2. The system clears new category input
3. The flow resumes at step 3.

Termination: The user leaves the page.

Post condition: A new category has been created.

3.28 [USE-28] An administrator deletes a category [FREQ-25]

Description: Each post must be assign to one of the categories. An administrator should be able to create a new category as required.

Priority: Medium

Scope: An administrator should be able to remove the category as long as there are no posts using it at the time of deletion.

Precondition: The user must be logged in and must be an administrator type.

Activation: An administrator enters the category management area

Main flow:

1. An administrator enters the category management area.
2. An administrator navigates to the category which he would like to delete
3. An administrator deletes the category
4. The system checks if there are no posts using deleting category [E1]
5. The system deletes the category
6. The system updates the category tree
7. The system refreshes the page view

Exceptional flow:

E1: The category has posts attached to itself.

1. The system displays a warning message that the category user wants to delete has been used by some posts
2. The flow resumes at step 2.

Termination: The user leaves the page.

Post condition: The category has been deleted.

3.29 [USE-29] An administrator moves the post to different category [FREQ-26]

Description: The category of the post is decided on its creation by its author. However initial category may not always be the best pick, or possibly a better category was not available on the time of post creation. Therefore, an administrator should be able to move the post to different category.

Priority: Medium

Scope: The scope of this use case is to provide an administrator with functionality to move the post to different category.

Precondition: The user must be logged in and must be an administrator type.

Activation: An administrator enters the category management area

Main flow:

1. An administrator navigates to the post he wishes to move
2. An administration enters the post move mode
3. An administrator navigates to new category
4. An administrator moves the post
5. An application moves the post
6. An application generates information notification to an author of the post about the fact that his post has been moved
7. An application displays an information message to administrator about task completion

Termination: The user leaves the page.

Post condition: The post has been moved to new category.

3.30 [USE-30] The user logs out from the session [FREQ-27]

Description: As majority of the functionality is only available to logged in users, the users should be able to also finish his session and log out from the page so whoever is using a PC after him will not be able to perform actions limited to this user only.

Priority: High

Safe Boards: Technical Report

Scope: The scope of this use case is to provide the user with functionality log out from the session.

Precondition: The user must be logged in.

Activation: The user logs out from his session

Main flow:

1. The user logs out from his session
2. An application destroys the session
3. The application redirect user to login page.

Post condition: User's session has been destroyed.

4 Design and Architecture

4.1 Data models - overview

An application is build out of 5 data objects: category, notification, post, reply and users. Safe boards store its data in object related database, and even that MongoDB is non-relational database it has been implemented it the way that objects often relates to each other. For example, post object under settings arrays of references to user's object. This allows a great flexibility which is hard to achieve in relational databases where each object must be re-assemble after pulling data from the database. In MongoDB it is not only possible to easily pull a whole object, but it also allows to choose which elements of the object should not be loaded or most importantly it allows to populate references to other objects with the object itself. Below we have an example of loading post object and populating its references to user and replies objects:

```
post.findOne({ '_id': requestedPostId })
  .populate('settings.author', 'displayname')
  .populate({
    path: 'replies',
    model: 'Reply',
    populate: [{
      path: 'author',
      model: 'User',
      select: 'displayname'
    }],
    {
      path: 'rreplies',
      model: 'Reply',
      populate: {
        path: 'author',
        select: 'displayname'
      }
    }
  ]
})
```

Populates author with user object

but nulling only displayname
Populates 'replies' array

Populates author of reply

For each reply in replies

populate an array of rreplies

Populates author of reply

The post object contains a reference to an author's user object which is simple id of user who created the post. First .populate tells the Mongoose that instead of id we would like to receive an object of an author, but we do not need a whole object with sensitive information such as passwords, instead we only need a display name. The post object also contains an array of 'replies' which again is an array of ids referring to reply objects. Mongoose is asked again to populate and replace each id with reply object itself. Each reply may also contain an array of replies associated with which is being populated. In addition, each reply has an author which display name is being populated. What we are receiving from the database instead of simple post object with references to another

Safe Boards: Technical Report

object, is a super-object with all fields populated with requested information. This is incredibly sophisticated way of storing and receiving an object from the database.

The disadvantage of such a solution is maintaining arrays of IDs when removing an object. As for relational databases removing record is the end of the process, in our setup after removing an object such as reply we need to remove all references to it from any array. Therefore, it is handy to have double linked references for example each category has array of posts assigned to it but also each post has an ID of its category which makes it easier when deleting a post and removing reference to it from category. Thanks to that reference to category in post it is easy to find right category instead of looping through all of them in search for which one contains reference to our post:

```
post.remove({ _id: id }, function(err) {
  if (err) {
    res.status(404);
    res.send('Error while trying to remove post from the database');
  }
  else {
    var cid = p.settings.category;
    //search conditions
    var conditions = { _id: cid };
    //edit conditions
    var update = { $pull: { postsId: mongoose.Types.ObjectId(id) } };
    //edit options
    var options = { multi: true };
    //Execute query
    category.update(conditions, update, options, function(err) {
      if (err) {
        console.log("Error when removing from category:" + err);
      }
      else {
        r.status = "ok";
        r.m = "Your post has been removed";
        res.send(r);
      }
    });
    notification.remove({ post: id }, function(err) {
      if (err) {
        console.log(err);
      }
    });
  }
});
```

4.2 Data models

4.2.1 Category data model

```
var categorySchema = mongoose.Schema({
  name: { type: String, required: true },
  categoriesId: [{ type: Schema.Types.ObjectId, ref: 'Category' }],
  postsId: [{ type: Schema.Types.ObjectId, ref: 'Post' }],
  parent: { type: Schema.Types.ObjectId, ref: 'Category' }
});
```

4.2.2 Notification data model

```
var notificationSchema = mongoose.Schema({
  owner: { type: Schema.Types.ObjectId, ref: 'User' },
  creator: { type: Schema.Types.ObjectId, ref: 'User' },
  post: { type: Schema.Types.ObjectId, ref: 'Post' },
  type: {
    type: String,
    enum: ['info',
           'newRequest',
           'requestDen',
           'requestAcc',
           'newFollowedRep',
           'newMessage',
           'newInvite',
           'inviteDen',
           'inviteAcc'],
    default: 'info'
  },
  message: String,
  createdAt: { type: Date, default: Date.now},
  hasRead: { type: Boolean, default: false}
}, {
```

4.2.3 Post data model

```
var postSchema = mongoose.Schema({
  settings: {
    privacy: String,
    author: { type: Schema.Types.ObjectId, ref: 'User' },
    category: { type: Schema.Types.ObjectId, ref: 'Category' },
    encryption: { isEnabled: Boolean, checkword: String },
    isAdmin: Boolean,
    isRequested: Boolean,
    isAllowed: Boolean,
    access: {
      admin: [{ type: Schema.Types.ObjectId, ref: 'User'}],
      allowed:[{ type: Schema.Types.ObjectId, ref: 'User'}],
      requested:[{ type: Schema.Types.ObjectId, ref: 'User'}],
      invited:[{ type: Schema.Types.ObjectId, ref: 'User'}]
    }
  },
  header: {
    subject: String,
    votes: {
      num: Number,
      upVotes: [String],
      downVotes: [String]
    }
  },
  body: {
    text: String,
    updates:[String]
  },
  replies: [{ type: Schema.Types.ObjectId, ref: 'Reply' }],
  createdAt: { type: Date, default: Date.now}
}, {
```

4.2.4 Reply data model

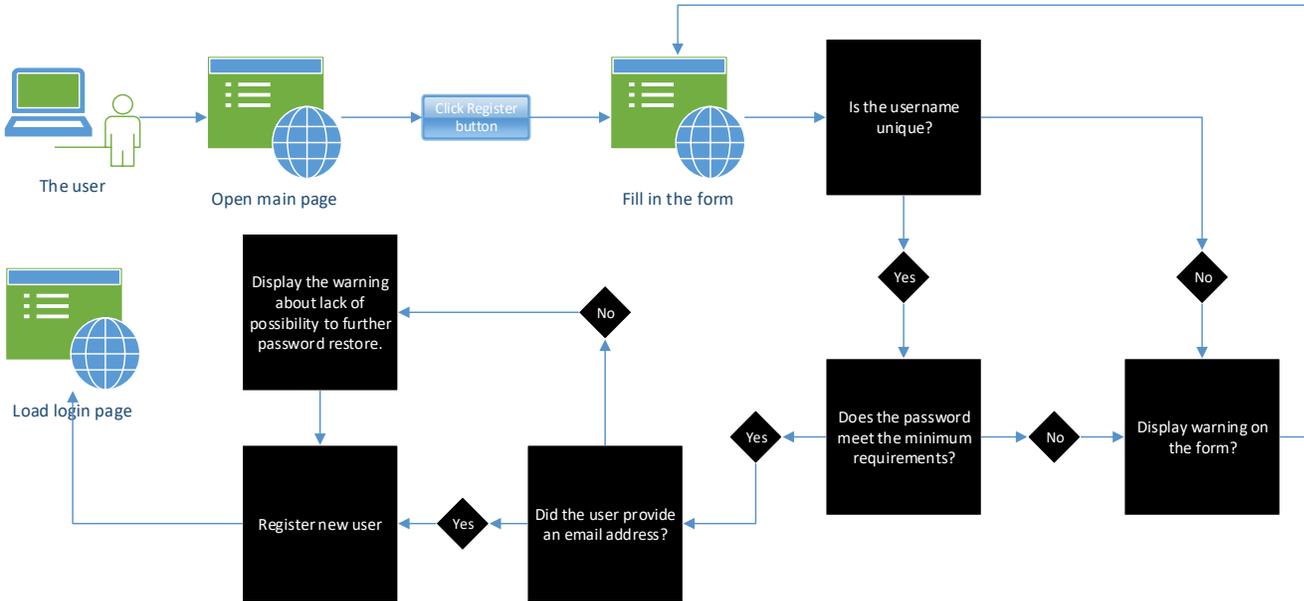
```
var replySchema = mongoose.Schema({
  isDeleted: Boolean,
  author: { type: Schema.Types.ObjectId, ref: 'User' },
  text: String,
  rreplies: [{ type: Schema.Types.ObjectId, ref: 'Reply' }],
  votes: {
    num: Number,
    upVotes: [String],
    downVotes: [String]
  },
  createdAt: {type: Date, default: Date.now}
}, {
```

4.2.5 User data model

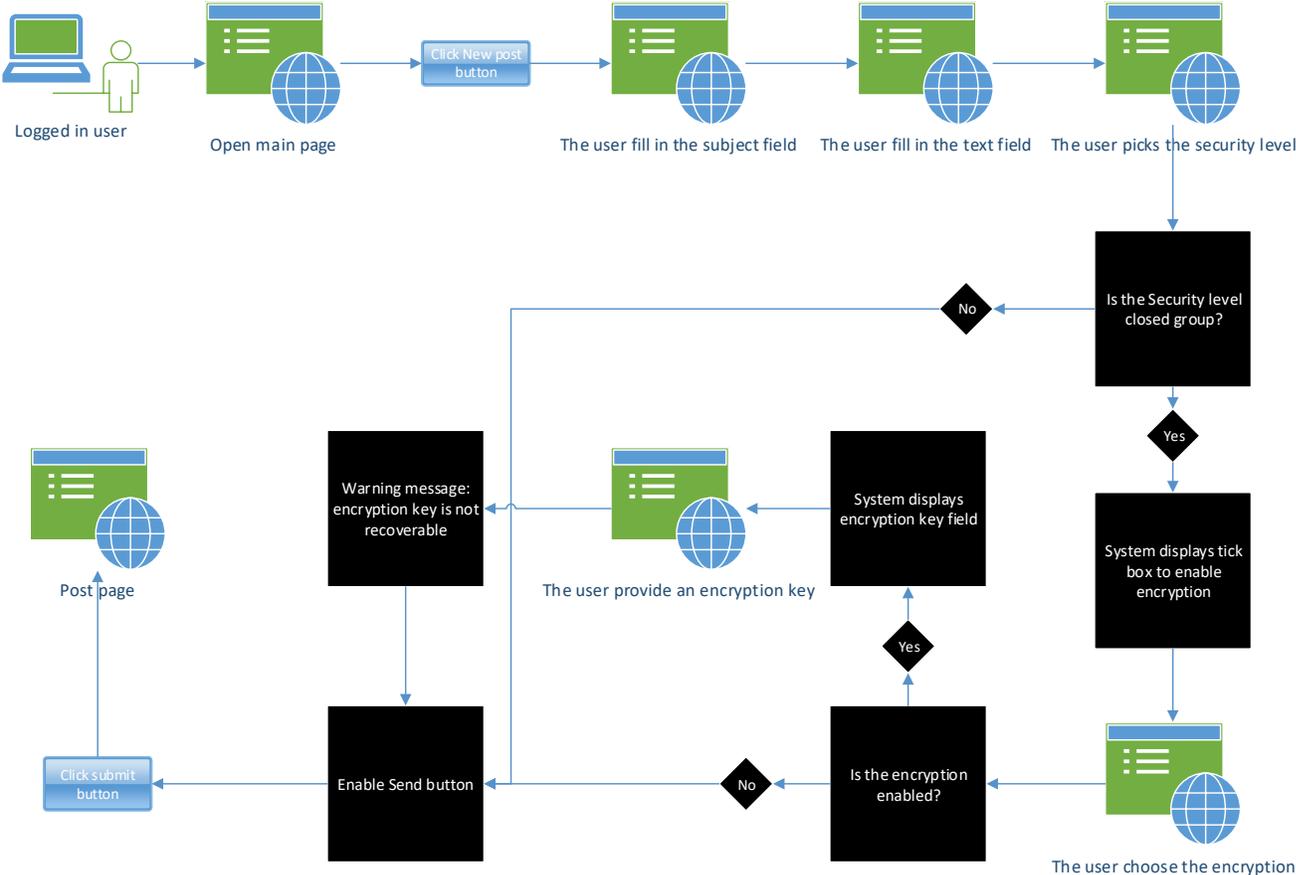
```
var userSchema = mongoose.Schema({
  displayname: String,
  local: {
    username: String,
    password: String,
    role: {
      type: String,
      enum: ['user', 'admin'],
      default: 'user'
    }
  },
  facebook: {
    id: String,
    token: String,
    email: String,
    name: String,
    role: {
      type: String,
      enum: ['user', 'admin'],
      default: 'user'
    }
  },
  google: {
    id: String,
    token: String,
    email: String,
    name: String
  }
}, {
```

4.3 Flow charts

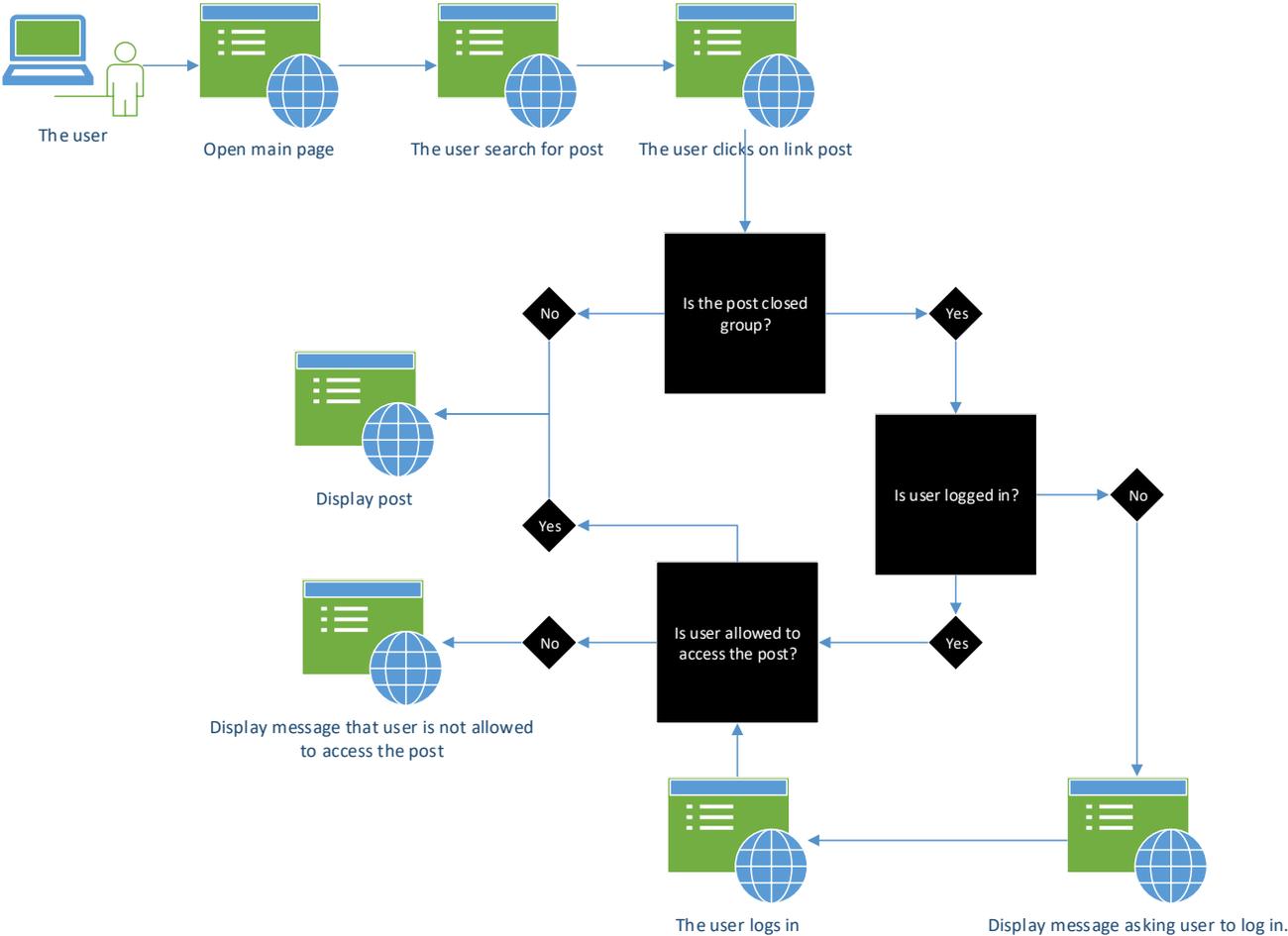
4.3.1 New user registration:



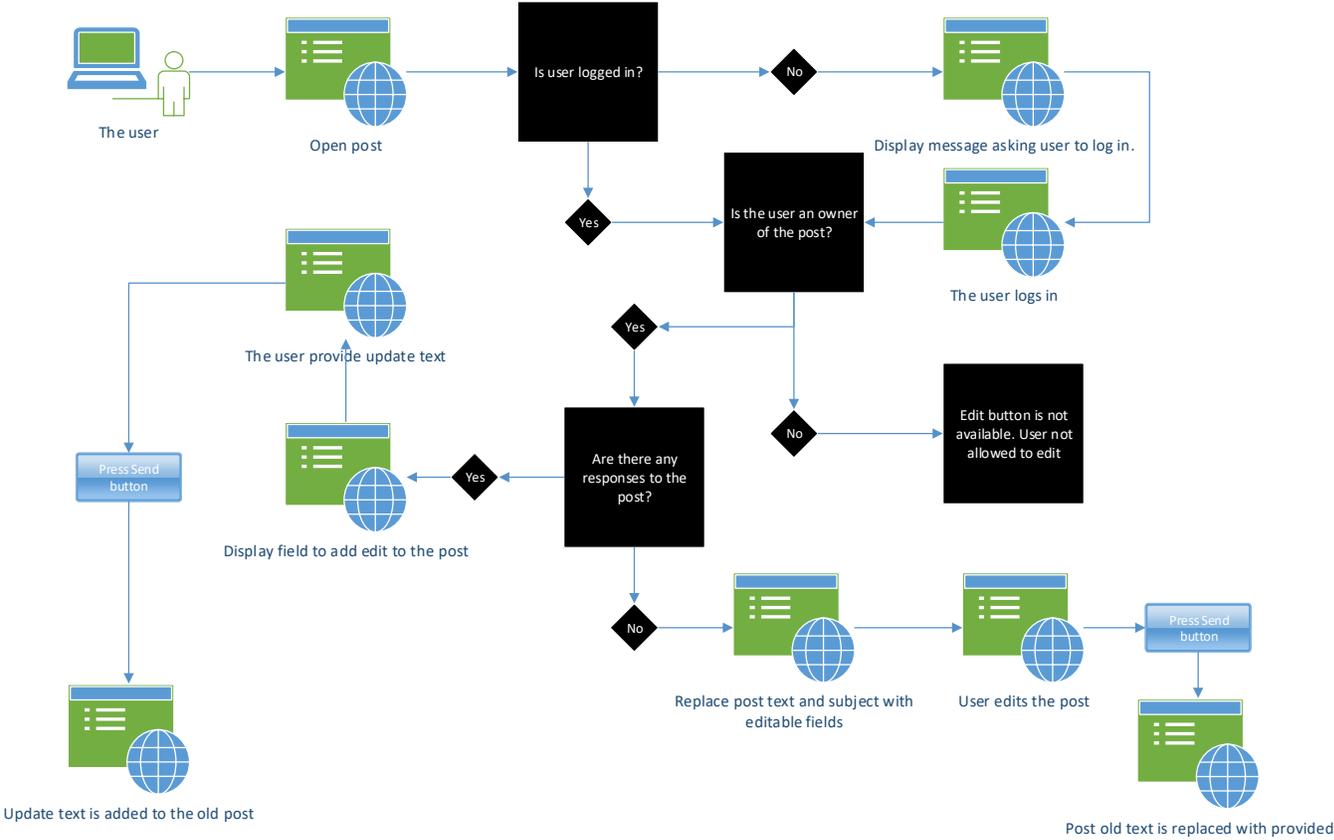
4.3.2 Creating new post:



4.3.3 Opening an existing post



4.3.4 Editing the post



5 Implementation

The project is built on Node.js server with an express serving users' request. High level implementation base on Node.js server listening on port 80, and user making a request to it via browser. Server validates then the request if it does match any of the available patterns consisted in routes.js file. Next if requested URL matches one of the patters it proceeds to execute code for this patter. If the URL require user to be authenticated, a function runs through middleware checking if user's session exists, and next it renders the view assigned to the URL. There are following views available:

v-category.ejs: containing all the functionality to manage the categories. It is available for admins only. Used in: [USE-27 and USE28]

v-list.ejs: this view displays a list of posts for picked category. Used in navigation to the post.

v-newpost.ejs: this view contains all the functionality used in new post creation. Used in [USE-05]

v-notifications.ejs: this contains all the functionality needed for viewing users' notifications as well as interacting with them. Used in [USE-23]

v-search.ejs: this view is used to display user's search results. Used in [USE-22]

v-settings.ejs: this view contains all the account settings. Used in [USE-03], [USE-04], [USE-25], [USE-26]

v-singlePost.ejs: this view is the main point of the application, containing majority of the functionality. This displays any post to the user and allows user to interact with it. Used in [USE-06] to [USE-21]

Once the view is rendered by Express it is being return to the user as HTML. The HTML is interpreted by the browser on the user side and AngularJS \$http modules making other requests to the server via its API. Those requests are triggered on page load but also during the user interactions with the website. This allows the content of the page to load and refresh dynamically without the need of reloading whole page. An API request are served by the same routes.js module on the server site but this time instead of rendering HTML they return the data, changing the state of the database, or both. Each API refers to one of five controllers that controls different parts of the application:

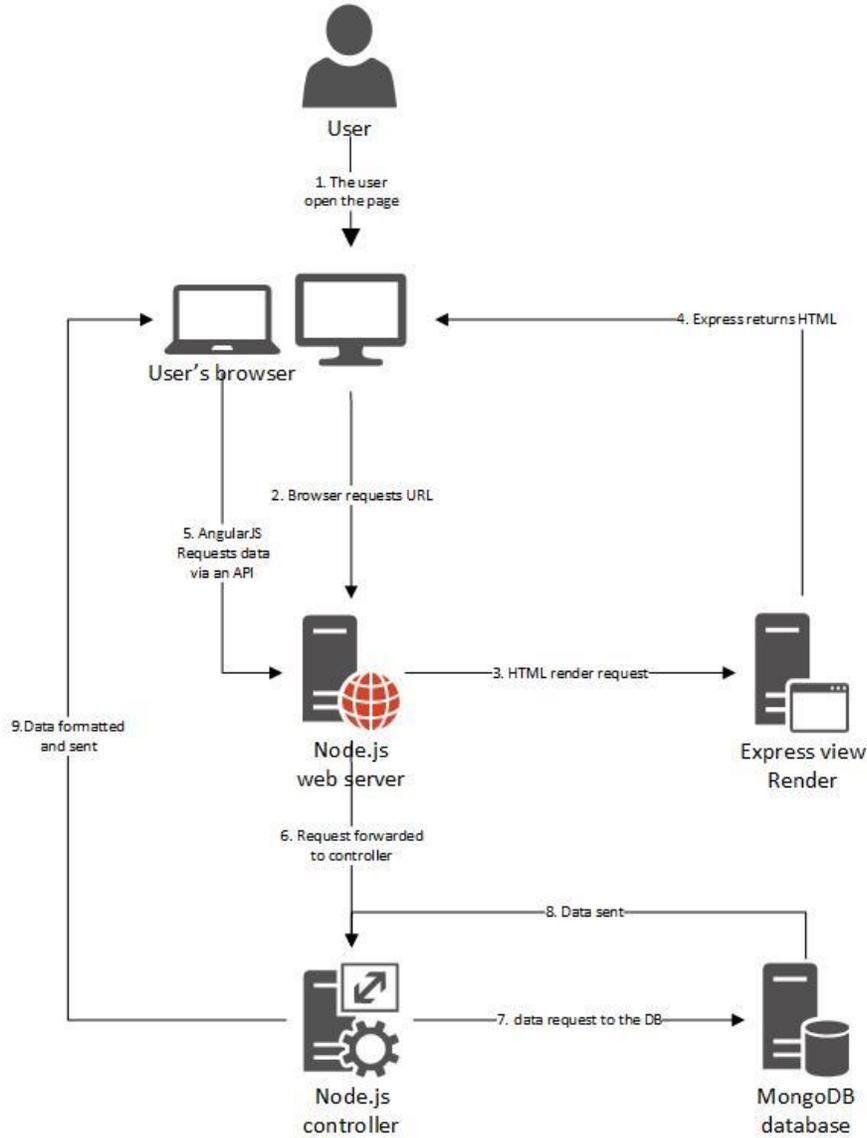
contr-category.js: controls all actions on post's category such as creating new category, removing category or getting a tree parents of chosen category.

contr-notifications.js: controls all the actions executed on the notification object such as creating new notification, marking notification as read, responding to the notification, deleting it, returning all notification for user.

contr-post.js: controls all the actions executed on the post and it's reply object. Server to over 20 different API requests such as: creating new post, deleting it, getting post by id, voting on post, editing post, responding to the post, deleting reply, editing reply, requesting an access, changing post settings etc.

contr-user.js: controls all the actions executed on user object such as getting list of all users for autofill, changing the password or display name.

Controllers responsibility is to take user request, process it, and return the data or change the state of the database. Controllers handle all the input validation, as well as user authorization, returning different responses if the request didn't pass either. Those responses are then handled back by AngularJS on the client side to visually communicate server response.



The flow above describes full run situation, but it is worth to notice that this is the case only when user is loading a new page. In the situation where the user is interacting with already loaded page steps 2 to 4 are irrelevant.

5.1.1 [IMP-01] Authentication [FREQ-01]

An authentication has been implemented using PassportJS library. This has been used to provide an authorization framework which allows users to authorize access with Facebook and Google OAuth, as well as with local username (Passport.js, 2018). Passport provides also a functionality to use it as a middleware gate, where each request

Safe Boards: Technical Report

which require user to be logged in can go through `isLoggedIn` function which will forward user to login screen if not authenticated or continue to execute the code if user's session has been found:

```
// route middleware to ensure user is logged in
function isLoggedIn(req, res, next) {
  if (req.isAuthenticated())
    return next();
  res.redirect('/login');
}
```

Use:

```
// NEW POST =====
app.get('/newpost', isLoggedIn, function(req, res) {
  req.user.getCount(function(u) {
    res.render('v-newpost.ejs', {
      user: req.user,
      nCount: u
    });
  });
});
```

5.1.2 [IMP-02] Linking social media account [FREQ-02]

Linking social media is handled by controller based on PassportJS. On the linking request the user is being forwarded to Fakebook's or Google's Auth service where he is asked to authenticate himself. Once authenticated user will get asked to share required information, and next the token is supplied to the application. Received token is saved to the database where later serves to authenticate user.

5.1.3 [IMP-03] Unlinking social media account [FREQ-02]

When user decide to unlink his Fakebook's or Google's account and API request is sent to the back end where is forwarded to correct controller. Next, the controller removes the token and email address of user form the right user object item, depending if user has disconnected a Facebook or Google account.

5.1.4 [IMP-04] Starting new post [FREQ-03]

New post has been implemented as a series of 3 or 4 small forms to fill in depending on privacy settings. This is to keep user's focus on information that is important on each stage. First stage is picking the category, then providing subject and text of post, third stage contains important information about privacy settings. User use a dropdown to pick the settings and as he is doing so the information on each setting is being displayed. If the user picks one of two closed group settings, on 4th step user can decide rather the post will be encrypted or not. Thanks to limiting last two items to separate displays user's attention can be focused on important information about the choices he must do. On submission the data acquired from the user is send to application's API in JSON format,

where is combined with authors data contained in the session. Because the data about an author is not being send with the request it makes it harder to spoof the user, and we can be sure that an author is always the user who was logged in. A double validation has been applied for each required field. JavaScript on client site checks if all necessary fields are populated before allowing user to send the data, but also additional check is done on the backend sending proper error message if any kind of data is missing or if the user is not logged in.

If the encryption is enabled the post is encrypted on the server side, before being send to the server, this is due the requirement [NFR-12]. An encryption is handled by Crypto.js library using 256bit AES. The front-end script encrypts the message as well as a known to the server phrase which later will be used in secret phrase decryption.

An encrypted message is built of salt, initialization vector and encrypted message itself. Next those elements are disassembled and together with user provided secret phrase user for decryption [IMP-06]

```
$scope.encrypt = function(input, password) {  
  
    var salt = CryptoJS.lib.WordArray.random(128 / 8);  
    var key = CryptoJS.PBKDF2(password, salt, {  
        keySize: keySize / 32,  
        iterations: iterations  
    });  
    var iv = CryptoJS.lib.WordArray.random(128 / 8);  
    var encrypted = CryptoJS.AES.encrypt(input, key, {  
        iv: iv,  
        padding: CryptoJS.pad.Pkcs7,  
        mode: CryptoJS.mode.CBC  
    });  
    var checkword = CryptoJS.AES.encrypt("decrypted", key, {  
        iv: iv,  
        padding: CryptoJS.pad.Pkcs7,  
        mode: CryptoJS.mode.CBC  
    });  
    $scope.encrypted = salt.toString() + iv.toString() + encrypted.toString();  
    $scope.checkword = salt.toString() + iv.toString() + checkword.toString();  
};
```

5.1.5 [IMP-05] Replying to the post [FREQ-04]

Replying to the post is served by AngularJS \$http.post request to the API with reply data attached in JSON format, reply author is pulled from the session and added to the data from JSON. All data validation is done on both front and back end. When back end being just fall over in case the request forgery. Once the reply object is created and its ID returned from the database it is next pushed to the post's array of replies.

Safe Boards: Technical Report

```
//create new instance of reply
var r = new reply();
//add author's id form the session
r.author = new mongoose.Types.ObjectId(req.user._id);
//populate rst of the object
r.text = req.body.m;
r.replies = [];
r.votes = {};
r.votes.num = 0;
r.votes.upVotes = [];
r.votes.downVotes = [];
//save the object to the DB
r.save(function(err, rep) {
  if (err) {
    throw(err);
  }
  //push an ID of newly created reply to the post's array
  p.replies.push(mongoose.Types.ObjectId(rep._id));
  p.save();
  res.end();
});
```

5.1.6 [IMP-06] Replying to the replies [FREQ-05]

This functionality has been implemented in very similar way as [IMP-05] with the only difference that in last part the ID of newly created reply is not being pushed to the post array but rather to the array of parent's reply.

```
p.rreplies.push(mongoose.Types.ObjectId(rrep._id));
p.save();
res.end();
```

5.1.7 [IMP-07] Accessing closed group post [FREQ-06]

On the post request the `v-singlePost.ejs` is rendered and returned to the user, the AngularJS script loaded with returned HTML is reading the post ID from the URL and send request for the post data to the API. Next, a controller function triggered by the API request checks for the post privacy settings. If the setting is 'public' it returns the post to the request, if the post is not public an application proceed to set following settings on the object:

- `isAdmin` - which indicates if requesting user is an admin of the post
- `isRequested` - which indicates if requesting user is on the list of users who request the access to the post
- `isAllowed` - indicates if the user is allowed to read the post

This is done by looping through the arrays of access setting:

Safe Boards: Technical Report

```
//check if user is an admin
for (var i = 0; i < doc.settings.access.admin.length; i++) {
    if (doc.settings.access.admin[i].toString() == req.user._id) {
        doc.settings.isAdmin = true;
    }
}

//check if user is allowed to access the post
for (var j = 0; j < doc.settings.access.allowed.length; j++) {
    if (doc.settings.access.allowed[j].toString() == req.user._id) {
        doc.settings.isAllowed = true;
    }
}

//check if user is awaiting to get access to the post
for (var k = 0; k < doc.settings.access.requested.length; k++) {
    if (doc.settings.access.requested[k].toString() == req.user._id) {
        doc.settings.isRequested = true;
    }
}
```

Next, using those indicators an application makes the decision on what kind of response it should send. If `isAllowed` is true it will send the post object, if `isRequested` is true it will send a message which then will be interpreted by Angular to display information to the user that he is not allowed to view this post and his request is awaiting approval. Finally, if both of them are false, an application will send the response which will display simple form on which user can request an access [FREQ-12].

Since the authorisation is done on the back end no post object data must be send to the browser until it is certain that the user is authorised to receive it.

5.1.8 [IMP-08] Accessing encrypted post [FREQ-07]

If the user is authorised to access the post [IPM-05] but the post is encrypted its encrypted form is passed to the front end (posts are not decrypted on the back end because as per [NFR-12] server side should never be allow to receive the encryption phrase. Once the post object with encrypted messages is received, Angular script will present user with the prompt to provide the secret phrase for decryption. Each post object contains a secret phrase known to the server which is also encrypted with the same key. This is used to validate the key provided by the user. First the front end will attempt to decrypt that phrase with provided key and if the encrypted phrase match the expected phrase it means that provided key is valid. If provided phrase is not valid JavaScript will present user with an error message.

If the phrase is correct, the front-end script will process to decrypting post, its updates if any and its replies. Once that is completed the script will display the content of the post:

Safe Boards: Technical Report

```
//post decryption
$scope.decr = function() {
  //check user's phrase if is valid
  var check = $scope.decrypt($scope.post.settings.encryption.checkword, $scope.phrase);
  if (check == "decrypted") {
    //decrypt the body
    $scope.post.body.text = $scope.decrypt($scope.post.body.text, $scope.phrase);
    //decrypt an updates
    if ($scope.post.body.updates.length > 0) {
      for (var k = 0; k < $scope.post.body.updates.length; k++) {
        $scope.post.body.updates[k] = $scope.decrypt($scope.post.body.updates[k], $scope.phrase);
      }
    }
    //decrypt replies
    if ($scope.post.replies.length > 0) {
      for (var i = 0; i < $scope.post.replies.length; i++) {
        $scope.post.replies[i].text = $scope.decrypt($scope.post.replies[i].text, $scope.phrase);
        //decrypt the replies of replies
        for (var j = 0; j < $scope.post.replies[i].rreplies.length; j++) {
          $scope.post.replies[i].rreplies[j].text = $scope.decrypt($scope.post.replies[i].rreplies[j].text, $scope.phrase);
        }
      }
    }
    $("#pass").delay(800).fadeOut(400, function() {
      $("#main").fadeIn(400);
    });
  }
}
```

Decryption process itself is done by stripping salt and initialization vector from the message and using them in combination with phrase provided by the user to generate the key, which then is used for encryption:

```
$scope.salt = CryptoJS.enc.Hex.parse(message.substr(0, 32));
$scope.iv = CryptoJS.enc.Hex.parse(message.substr(32, 32))
var encrypted = message.substr(64);

var key = CryptoJS.PBKDF2(pass, $scope.salt, {
  keySize: keySize / 32,
  iterations: iterations
});

var decrypted = CryptoJS.AES.decrypt(encrypted, key, {
  iv: $scope.iv,
  padding: CryptoJS.pad.Pkcs7,
  mode: CryptoJS.mode.CBC
})
try {
  return decrypted.toString(CryptoJS.enc.Utf8);
}
catch (ex) {
  if (decrypted.toString() != "") {
    return decrypted.toString();
  }
  else {
    return message;
  }
}
```

5.1.9 [IMP-09] Changing privacy settings [FREQ-08]

Changing privacy settings is fully handled on the back end with only role of the front end being: displaying to the user relevant options, information on each setting and sending requests to the API.

Safe Boards: Technical Report

Once the request to API is received the back end proceed to checking if the user is authorised to make the change (if he is an author of the post). If that is the case an application will process further. An application will check if the new status is different from previous, and then will handle one of the 3 possibilities:

Changing public post to one of the closed group: change the privacy type, create the empty arrays for access control, populate array of allowed users with an author.

```
p.settings.privacy = newStatus;
p.settings.access.admin = [p.settings.author];
p.settings.access.allowed = [p.settings.access.allowed];
p.settings.access.requested = [];
p.settings.access.invited = [];
p.save(function(err) {
```

Changing from one closed group to another: change the privacy type

Changing from closed group to public: change the privacy type, clear all the access control, and clear all notifications about access request and invitations to access.

```
//set new privacy
p.settings.privacy = newStatus;
//clear all the access controll
p.settings.access.admin = [];
p.settings.access.allowed = [];
p.settings.access.requested = [];
p.settings.access.invited = [];
p.save(function(err) {
  if (err) {
    a.status = "Problem";
    a.message = "Problem when saving new settings.";
    res.send(a);
  }
  else {
    //remove all notifications for this post
    notification.deleteMany({
      post: p._id, type:
        { $in: ['newRequest', 'requestDen', 'requestAcc', 'newInvite', 'inviteDen', 'inviteAcc'] } }, function(err) {
      if (err) {
        a.status = "Problem";
        a.message = "Problem when deleting notifications.";
        res.send(a);
      }
      else {
        a.status = "Completed";
        a.message = "Your privacy settings has been applied.";
        res.send(a);
      }
    });
  }
});
```

5.1.10 [IMP-10] Removing encryption from the post [FREQ-09]

As the encryption setting is only accessible from the post level, the user must access the post first and decrypt it. Therefore, to disabling encryption we will already had decrypted post object in the memory. The front end will take this object and send it in the PUT request to the API. An API then will strip the object and update the post, the

Safe Boards: Technical Report

post encryption settings, as well as it will fire off a separate update query to update each reply attached to the post.

```
p.settings.encryption.isEnabled = false;
p.settings.encryption.checkword = null;
//decrypt the post body with all the updates
p.body = encPost.body;
p.save(function(err) {
  if (err) {
    a.status = "Problem";
    a.message = "A problem occurred when saving encrypted post to the database. Please try again.";
    res.send(a);
  }
  else {
    //decrypt replies
    if (encPost.replies.length > 0) {
      for (var i = 0; i < encPost.replies.length; i++) {
        reply.update({
          _id: encPost.replies[i]._id
        }, {
          $set: { text: encPost.replies[i].text }
        },
        function(err) {
          if (err) {
            a.status = "Problem";
            a.message = "A problem occurred when saving encrypted post to the database. Please try again.";
            res.send(a);
          }
        });
      }
      //decrypt reply's replies
      for (var j = 0; j < encPost.replies[i].rreplies.length; j++) {
        reply.update({
          _id: encPost.replies[i].rreplies[j]._id
        }, {
          $set: { text: encPost.replies[i].rreplies[j].text }
        },
        function(err) {
          if (err) {
            a.status = "Problem";
            a.message = "A problem occurred when saving encrypted post to the database. Please try again.";
            res.send(a);
          }
        });
      }
    }
  }
});
}
```

5.1.11 [IMP-11] Adding an encryption to the post [FREQ-09]

Adding the encryption is done in the similar way as removing it with the only difference that the content of the post must be first encrypted on the client side. For this purpose, an application creates a deep copy of the post object:

```
var encPost = $.extend(true, {}, $scope.post);
```

And then encrypts the post content together with all the replies.

Safe Boards: Technical Report

```
//encrypt body
encPost.body.text = $scope.reen(encPost.body.text);
console.log(encPost.body.text)
//encrypt updates if any
if (encPost.body.updates.length > 0) {
    for (var k = 0; k < encPost.body.updates.length; k++) {
        encPost.body.updates[k] = $scope.reen(encPost.body.updates[k]);
    }
}
//encrypt replies
if (encPost.replies.length > 0) {
    for (var i = 0; i < encPost.replies.length; i++) {
        encPost.replies[i].text = $scope.reen(encPost.replies[i].text);
        for (var j = 0; j < encPost.replies[i].rreplies.length; j++) {
            encPost.replies[i].rreplies[j].text = $scope.reen(encPost.replies[i].rreplies[j].text);
        }
    }
}
```

Once this is done post object with encrypted text is send in PUT request to the API where is handled in the similar way as for encryption removal [IMP-08].

5.1.12 [IMP-12] Inviting users to closed group [FREQ-10]

The user invitation is handled by two modules, first is taking care of autocompleting users display name when searching for user [NFR-13], and second is generating an invitation.

The autocomplete has been applied with use of AngularJS Material framework which has its values populated via the API request. The request on the back end pulls all the user display names and returns it to the front end. Front end fetches the array into the map for the value and display username:

```
if (r.status == 200) {
    userList = r.data.map(function(user) {
        return {
            value: user.displayname.toLowerCase(),
            display: user.displayname
        };
    });
}
```

Next on every user input into the search a query against that list is executed filtering it out to only those values that contains user inputted phrase:

Safe Boards: Technical Report

```
//filter out the user display names on user input
function querySearch(query) {
  var results = query ? userList.filter(createFilterFor(query)) : userList;
  var deferred = $q.defer();
  deferred.resolve(results);
  return deferred.promise;
}

function createFilterFor(query) {
  var lowercaseQuery = angular.lowercase(query);

  return function filterFn(user) {
    return (user.value.indexOf(lowercaseQuery) === 0);
  };
}
}
```

When the user submits the invitation an API request is sent to back end to process. The back-end controller is then checking following: is the user authorised to make an invitation, is the post closed group, does invited user exist, is invited user already have an access to the post, was invited user already invited. If one of those is true system will send adequate message in response, else it will create the notification object and save it to the database, sending back to the user a response with confirmation.

```
//create new notification
var n = new notification();
n.owner = u;
n.creator = me;
n.post = new mongoose.Types.ObjectId(pId);
n.type = 'newInvite';
n.save(function(err) {
  if (err) {
    fn.status = "error";
    fn.message = "Error when adding new invitation.";
    res.send(fn);
  }
  else {
    fn.status = "ok";
    fn.message = "An invitation has been sent succesfully.";
    res.send(fn);
  }
});
```

5.1.13 [IMP-13] Requesting access to the post [FREQ-12]

Requesting an access is a follow up procedure from [IMP-05] when users is accessing closed group, but the application does not find him on the list of authenticated users. In this situation a form to request the access is made available to the user. Front end form is taking message entered by the user and send it in the POST request to the API. An IP then is forwarding request to the correct controller function which creates an instance of new notification which is built of the message user provided, post ID, user who requested an access and recipient of the notification being an author of the post. Next

the notification I saved to the database, and requestor's ID is pushed to the access control of the post to settings.access.requested array.

5.1.14 [IMP-14] Post edition [FREQ-13]

The post edition is only allowed until the first response is posted under the post. For this reason, an edit button will unhide edit form which will replace the original text only if number of replies = 0:

```
if ($scope.post.replies.length > 0) {  
    $("#postedit2").show();  
}  
else {  
    $scope.temp = $scope.post.body.text;  
    $("#posttext").hide();  
    $("#postedit").show();  
}
```

Next, on submission, the front end will validate the text field is not empty and send it to the back end via an API. The back end again validates if the user is authorised to make the change and push the change to the database. Next the back end sends the response to the front end which will react to it by refreshing the post view displaying updated text.

5.1.15 [IMP-15] Post update [FREQ-13]

Post update is a replacement for post edit in situation when someone responded to the post already. The update does not change the content of the post but rather adds a note to it. Updates are stored in separate array in the post object. Similarly to [IMP-12] different edit form is unhidden when number of replies > 0. When update is submitted Front end validates it for empty field and send to the back end via API. The back end then validates the user if it is authorised to add the update and push it to the array of updates. Next an application sends response about update completed to the front end which then refreshes the post view.

5.1.16 [IMP-16] Post deletion [FREQ-14]

Post deletion is first validated by the front end by making deletion option available base on 2 factors: if the user is an author and if the post does not have any replies:

```
<div class="col-md-2" ng-if="post.settings.author._id==user" style="text-align: right">  
  <md-icon md-svg-src="img/icons/settings.svg" style="width:30px;height:30px;" ng-click="showSettings($event)"></md-icon>  
  <md-icon md-svg-src="img/icons/edit.svg" style="width:30px;height:30px;" ng-click="pedit_show()"></md-icon>  
  <md-icon ng-if="post.replies.length==0" ng-click="pdel()" md-svg-src="img/icons/dustbin.svg" style="color: #f00;width:30px;height:30px;"></md-icon>  
</div>
```

Safe Boards: Technical Report

When the user deletes the post the API request is sent to the back end, where it is validated to check if the requestor is allowed to delete the post and once more for the number of updates. Then the post object is deleted from the database. Next the post category is acquired from the database using post's pointer `p.settings.category` and the reference to the post is deleted from it. Finally all the notifications such as invites or requests, which refer to the post are deleted:

```
//remove the post
post.remove({ _id: id }, function(err) {
  if (err) {
    res.status(404);
    res.send('Error while trying to remove post from the database');
  }
  else {
    //remove the refference to the post from the category
    var cid = p.settings.category;
    //search conditions
    var conditions = { _id: cid };
    //edit conditions
    var update = { $pull: { postsId: mongoose.Types.ObjectId(id) } };
    //edit options
    var options = { multi: true };
    //Execute query
    category.update(conditions, update, options, function(err) {
      if (err) {
        throw(err);
      }
      else {
        r.status = "ok";
        r.m = "Your post has been removed";
        res.send(r);
      }
    });

    //remove all notifications reffering to deleted post
    notification.remove({ post: id }, function(err) {
      if (err) {
        console.log(err);
      }
    });
  }
});
});
```

5.1.17 [IMP-17] Reply edit [REFQ-15]

Reply edition is only allowed if the user is the owner of the post which is first validated on the client side where it is not rendered unless the above is true. Once the user click the edit button reply text is being replaced with editable field which is pre populated with the original text. Once the text is edited and submitted it is validated for empty input and set via API to the back end. Then the application validates again if the user is allowed to edit such reply and if it is not empty. Next an application updates an object in the database and sent response to the front end which reloads the post content.

5.1.18 [IMP-18] Reply deletion {FREQ-16}

Reply deletion is first validated on the client side where it an button is only rendered if the user is an author of the post, then if triggered the request is send to back end via API. On the back end, the request is validated again for the authorization point, and

Safe Boards: Technical Report

checked if the referenced post exists. Then depends if the reply has any other replies attached to it or not it will be treated differently. If it does, then its text will get nulled and its isDeleted state will get changed to true. This state is then used by the front end to replace the reply with the message that it has been deleted. If reply has no other replies attached, then its instance gets removed from the database and the reference to it is removed from the post object. Finally, a confirmation response is sent to the front end where it is transformed into the information message to the user.

```
if (r.rreplies.length > 0) {
  r.text="";
  r.isDeleted = true;
  r.save();
  res.status(200);
  res.send("Your reply has been removed");
}
else {
  reply.remove({ _id: repId }, function(err) {
    if (err) {
      console.log('Error while trying to remove reply from the database');
      res.render('notfound.ejs');
    }
    else {
      post.update({ _id: posId }, { $pull: { replies: mongoose.Types.ObjectId(repId) } }, { multi: true },
        function(err) {
          if (err) {
            console.log("Error when removing reply ref from post category:" + err);
          }
          else {
            res.status(200);
            res.send("Your reply has been removed");
          }
        });
    }
  });
}
});
}
```

5.1.19 [IMP-19] Voting [FREQ-17]

The voting process is based on user pressing the button which then sends request via API containing user's id and post or reply id. There are 4 different API URLs for both post and reply, up and down vote. Information received from the client is then process by the controller which acquire the voted object from the database and checks if the user's id is already on the list of voters. Each item contains two arrays one for users down voting and second for users up voting. System first checks the array of the same type as received to find out if the user voted on the post already it the same way, if not it will check second array to find out if the user changed its mind. If not it will increase or decrease the vote number by one and add user to the correct array. If however, the user changed his mind and then an application deletes user from previously vote array, adds it to the new vote and decreasing or increasing vote number by 2:

Safe Boards: Technical Report

```
for (var i = 0; i < p.header.votes.upVotes.length; i++) {
  if (p.header.votes.upVotes[i] == req.user._id) {
    alreadyUpVoted = true;
  }
}
if (alreadyUpVoted) {
  console.log("Already upvoted by the user ... not upvoting");
  var r = {};
  r.n = null;
  r.m = "You have previously upvoted this post.";
  res.send(r);
}
else {
  console.log("Not yet voted, processing... upvote");
  for (var j = 0; j < p.header.votes.downVotes.length; j++) {
    if (p.header.votes.downVotes[j] == req.user._id) {
      alreadyDownVoted = j;
    }
  }
  if (alreadyDownVoted != -1) {
    console.log("downvoted prviously, processing... upvote");
    //remove user from downvote
    p.header.votes.downVotes.splice(alreadyDownVoted, 1);
    //add user to upvote
    p.header.votes.upVotes.push(req.user._id);
    //add 2 to vote number
    p.header.votes.num += 2;
    p.save();
    var r = {};
    r.n = p.header.votes.num;
    r.m = "";
    res.send(r);
  }
  else {
    console.log("not downvoted prviously, processing... upvote" + requestedPostId);
    //add user to upvote
    p.header.votes.upVotes.push(req.user._id);
    //add 1 to vote number
    p.header.votes.num += 1;
    p.save();
    var r = {};
    r.n = p.header.votes.num;
    r.m = "";
    res.send(r);
  }
}
```

Once all the changes are completed an application will send response to the front end to update the score, or to notify any validation errors.

5.1.20 [IMP-20] Hiding / un-hiding low score replies [FREQ-18]

The low score replies are being hidden on client side during the rendering of page. Each container displaying reply have its replacement with hidden post message and depending on the score one or another will load:

Safe Boards: Technical Report

```
<div ng-show="rep2.votes.num<0" id="e_{{rep2._id}}">
  <blockquote>
    <h5>Comment hidden due to poor quality</h5>
    <button type="button" class="btn btn-link" ng-click="show(rep2._id)">Show...</button>
  </blockquote>
</div>
```

Showing hidden reply hides the message and un-hides the reply.

5.1.21 [IMP-21] Post search [FREQ-19]

To support the search functionality the database had to be updated with new index, which allows to search through not only post subjects but also post's body. Giving subject double weight as body.

```
{
  "v" : 2,
  "key" : {
    "_fts" : "text",
    "_ftsx" : 1
  },
  "name" : "body.text_text_header.subject_text",
  "ns" : "forum.posts",
  "weights" : {
    "body.text" : 1,
    "header.subject" : 2
  },
  "default_language" : "english",
  "language_override" : "language",
  "textIndexVersion" : 3
}
```

When user is using the search utility, a request to for new page is sent to the back end. Once the view is rendered and HTML sent in response, the frontend's script is sending API request to back end to search the database using the index above. The search result contains an array of posts which subject or body matched the search phrase:

```
post.find({ $text: { $search: req.params.search } })
  .populate('settings.category')
  .select('-body')
  .exec(function(err, p) {
    if (err) {
      console.log(err);
    }
    else {
      if (err || !p) {
        res.status(400).end();
      }
      else {
        console.log(filterPosts(p, req.user));
        res.send(filterPosts(p, req.user));
      }
    }
  });
```

Since the array is going to be used only to render a list of links the body of post has been skipped in the request above. Once the array is send and received by frontend it loops through the array and renders a list of links to the posts.

5.1.22 [IMP-22] Notifications [FREQ-20]

Notifications are displayed on with use of v-notifications.ejs HTML template. The template generates the HTML on user's requests which then when loaded on client's site triggers a request via API to the server for notifications data. Result from the server contains an author of notification display name, and post subject of post if any populated:

```
notification.find({ owner: req.params.user })
  .populate('creator', 'displayname')
  .populate('post', 'header.subject')
  .exec(function(err, n) {
    if (!n || err) {
      res.status(400).end();
    }
    else {
      res.send(n);
    }
  });
```

Response from the API contains an array of all notifications for current user, this array is looped through to generate a user-friendly view. Notifications view contains different handlers for each of notification type: information, access request, request deny, and approval, followed post reply, invite, invite deny, and approval. For example

```
<span ng-if="notification.type=='newInvite'">Invitation to post</span>
<span ng-if="notification.type=='newRequest'">Post access request</span>
<span ng-if="notification.type=='requestDen'">Access revoked</span>
<span ng-if="notification.type=='requestAcc'">Access granted</span>
<span ng-if="notification.type=='info'">Information</span>
<span ng-if="notification.type=='newFollowedRep'">New reply on followed post</span>
<span ng-if="notification.type=='newMessage'">New message from user</span>
<span ng-if="notification.type=='inviteDen'">Invitation denied</span>
<span ng-if="notification.type=='inviteAcc'">invitation accepted</span>
```

Each message will also have a full, informative version of the message, which is reviled after user click on the high-level detail information.

Each notification also has a marker hasRead which is set to false on its creation and if it is true it will display a new badge next to the notification:

```
<span ng-if="!notification.hasRead" class="badge progress-bar-danger">NEW</span>
```

The state of hasRead is changed once user clicks the notification, and revel detailed message. hasRead marker is also used to display the new notifications bell in the navigation panel. Every time logged in user access any page a count request is sent to

the database and the bell with number of new notifications is displayed on the navigation panel.

5.1.23 [IMP-23] Accessing own posts list [RFEQ-21]

When user request the list of his own posts, the request to render page is sent to the back end. The page is then served to the front end and during loading process an API request for data is send. This request is served by back end controller which is taking user's ID from the session and pulls all the post where this ID is listed as an author. Once the data is received by the front end it is used to populate the page with use of AngularJS.

5.1.24 [IMP-24] Changing display name [RFEQ-22]

Display name change is first validated on the client side for minimum length. If the length is greater than 3 characters submit button become available and user can send request to the back end using API. Once the back end receives the request it checks it again for the length and attempt to find new display name in the database. If found it sends the information to the front end about username being already in use. Otherwise, it acquires user's object from the database and updates the display name value. Next the confirmation message is send to the front end.

5.1.25 [IMP-25] Changing master account password [RFEQ-23]

To change the master account password, the user is required to provide a new password and new password twice. First. New password is validated on client side for minimum length and if both new password are matching. Next, current password and new one is send in PUT request via an API to the back end. There the length of new password is validated again. Next, current password is validated against the database in the same way when during logging in process, that is an hash of the password is created and compared with hash stored in database. If the user provided correct current password a hash of new password is created and used to update the database. Once this is completed a confirmation is send to the front end which logs user out and loads the login page.

Safe Boards: Technical Report

```
var oldPass = req.body.oldPass;
var newPass = req.body.newPass;
//validate current password
if (!u.validPassword(oldPass)) {
    res.send("An old password seems to be incorrect.");
}
else {
    //generate a hash of new password
    u.local.password = u.generateHash(newPass);
    u.save(function(err) {
        if (err) {
            throw err;
        }
        else {
            res.send("Your password has been updated.");
        }
    });
}
```

5.1.26 [IMP-26] Adding new category [RFEQ-24]

When administrator creates a new category, a request with category name and category parent ID, is send via an API to the back end. Once received the controller checks if the user is an admin type and pulls the parent category object from the database. Next, parent category object is checked if it does not already contain the category with the same name. If it doesn't a new category object is created and uploaded to the database. Once the database assigns an _id to the object this id is pushed to the parent's array of children.

```
//get the parent category
category.findOne({'_id': req.body.parent }).populate('categoriesId').exec(function(err, doc) {
    if (!doc || err) {
        res.setHeader("Content-Type", "text/html");
        res.status(406);
        res.send("Wrong parrent category");
    }
    else {
        //Check if category already exists
        for (var i = 0; i < doc.categoriesId.length; i++) {
            if (doc.categoriesId[i].name == req.body.category) {
                alreadyExist = true;
            }
        }
        if (alreadyExist) {
            res.setHeader("Content-Type", "text/html");
            res.status(406);
            res.send("Category with this mane already exists");
        }
        else {
            var n = new category();
            n.name = req.body.category;
            n.categoriesId = [];
            n.postsId = [];
            n.parent = mongoose.Types.ObjectId(req.body.parent);
            n.save(function(err, newCategory) {
                doc.categoriesId.push(mongoose.Types.ObjectId(newCategory._id));
                doc.save();
                res.setHeader("Content-Type", "text/html");
                res.status(200);
                res.send("Item has been added successfully.");
            });
        }
    }
});
```

5.1.27 [IMP-27] Adding new category [RFEQ-25]

When administrator removes the category, a request containing removed category Id is sent to the back end. The back-end controller validates currently logged user if he is an admin type and then pulls the category from the database. Next, an application checks

if category has any post or any other categories attached to the it. This is to ensure that no posts or categories will become orphans. Only then an application removes the post and send a confirmation to the frontend.

5.1.28 [IMP-28] Logout [RFEQ-27]

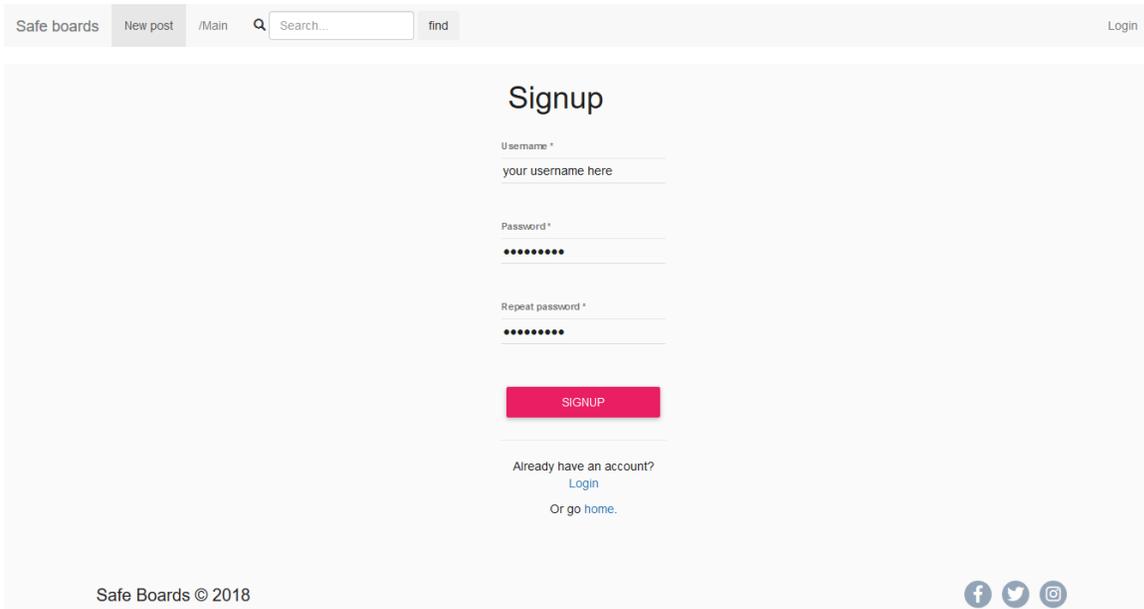
When the user is logging out a request to kill the session is sent to the back end. This request is full y handled by the PassportJS. Ad the user is being redirected to the main page.

```
app.get('/logout', function(req, res) {  
  req.logout();  
  res.redirect('/');  
});
```

5.2 Graphical User Interface (GUI) Layout

5.2.1 Authentication

Signup:



Safe Boards: Technical Report

Login:

The screenshot shows the login interface of the Safe Boards application. At the top, there is a navigation bar with 'Safe boards', 'New post', and '/Main'. A search bar is located to the right of the navigation bar. The main content area is titled 'Login' and contains the following elements:

- A 'Username *' input field.
- A 'Password *' input field.
- A red 'LOGIN' button.
- The text 'or login with:'.
- Two social login buttons: 'Facebook' (blue) and 'Google+' (red).
- Links for 'Need an account? Signup' and 'Or go home.'

At the bottom of the page, there is a copyright notice 'Safe Boards © 2018' and three social media icons: Facebook, Twitter, and Instagram.

5.2.2 Managing account

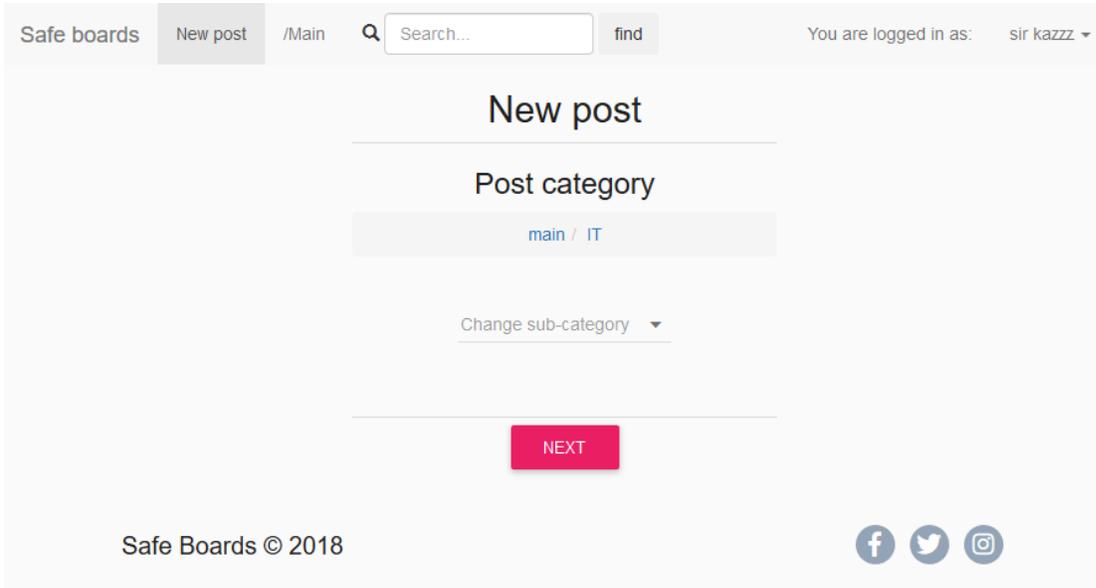
The screenshot shows the account management interface of the Safe Boards application. At the top, there is a navigation bar with 'Safe boards', 'New post', and '/Main'. A search bar is located to the right of the navigation bar. The main content area is titled 'You are logged in as: sir kazzz' and contains the following elements:

- A 'Local account' section showing the user's ID (5a77b98dab9edd9f8e49943b) and Username (kazzz).
- A 'Facebook' section with the text 'You do not have facebook account linked.' and a red 'CONNECT' button.
- A 'Google+' section with the text 'You do not have Google account linked.' and a red 'CONNECT' button.
- A 'Change display name' section with the text 'Your currently displayed name is: sir kazzz' and a 'Display name *' input field, followed by a 'SAVE' button.
- A 'Change the password' section with 'Password *', 'New password *', and 'Repeat new password *' input fields, followed by an 'UPDATE' button.

At the bottom of the page, there is a copyright notice 'Safe Boards © 2018' and three social media icons: Facebook, Twitter, and Instagram.

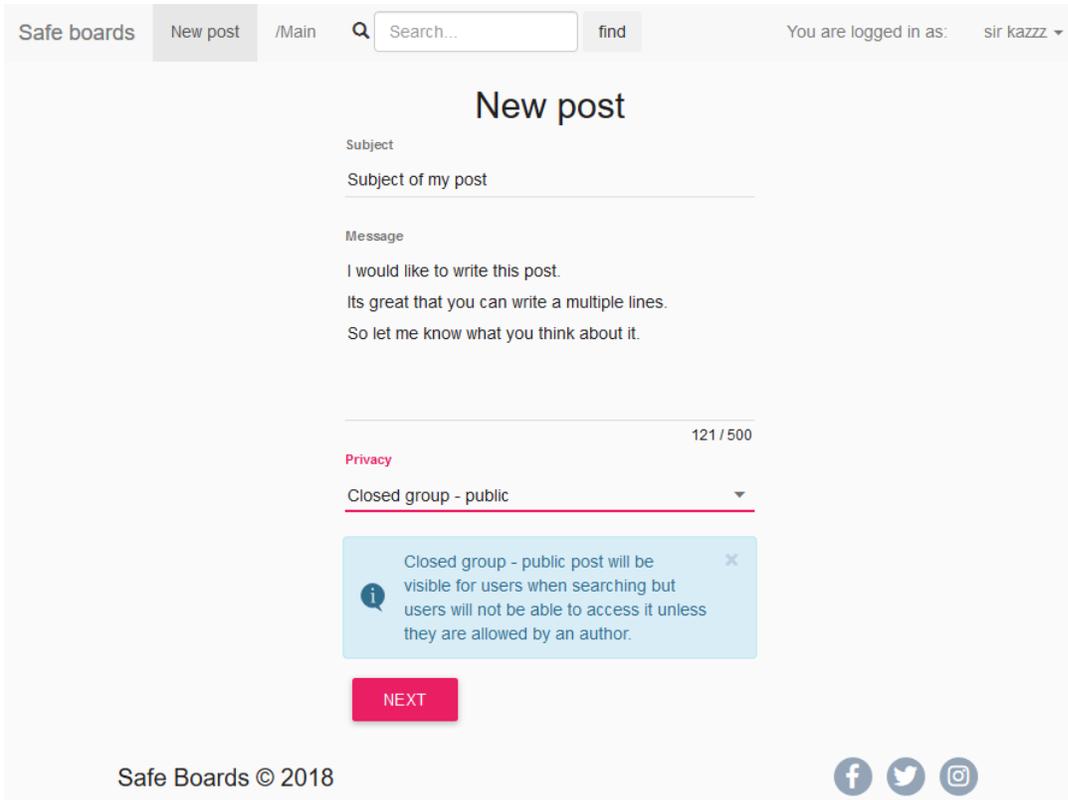
5.2.3 Create new post

Step 1: Category



The screenshot shows the 'New post' form in the 'Safe boards' application. The user is logged in as 'sir kazzz'. The form is titled 'New post' and is currently on the 'Post category' step. The category is set to 'main / IT'. There is a 'Change sub-category' dropdown menu. A red 'NEXT' button is visible at the bottom of the form. The footer includes 'Safe Boards © 2018' and social media icons for Facebook, Twitter, and Instagram.

Step 2: Post content and privacy level



The screenshot shows the 'New post' form in the 'Safe boards' application. The user is logged in as 'sir kazzz'. The form is titled 'New post' and is currently on the 'Post content and privacy level' step. The 'Subject' field contains 'Subject of my post'. The 'Message' field contains the text: 'I would like to write this post. Its great that you can write a multiple lines. So let me know what you think about it.' The character count is 121 / 500. The 'Privacy' level is set to 'Closed group - public'. A blue information box states: 'Closed group - public post will be visible for users when searching but users will not be able to access it unless they are allowed by an author.' A red 'NEXT' button is visible at the bottom of the form. The footer includes 'Safe Boards © 2018' and social media icons for Facebook, Twitter, and Instagram.

Safe Boards: Technical Report

Step 3: [optional] Encryption:

The screenshot shows the 'New post' form in the Safe Boards application. At the top, there are navigation links for 'Safe boards', 'New post', and '/Main', along with a search bar and a 'find' button. The user is logged in as 'sir kazzz'. The main heading is 'New post', followed by the 'Encryption' section. An orange 'ENABLED' button is visible. A warning box states: 'Warning! Your post will be encrypted. To read and reply to this post, users will need encryption phrase. The encryption phrase can be set below and should be distributed to the users using a secured communication channel. For security reason SafeBoards will not store the encryption phrase, and therefore if you forget your phrase we will not be able to assist you with restoring it.' Below the warning, there is a field for the 'Encryption secret...' with the placeholder text 'Secret phase so other cannot read it'. A red 'SEND' button is at the bottom of the form. The footer includes 'Safe Boards © 2018' and social media icons for Facebook, Twitter, and Instagram.

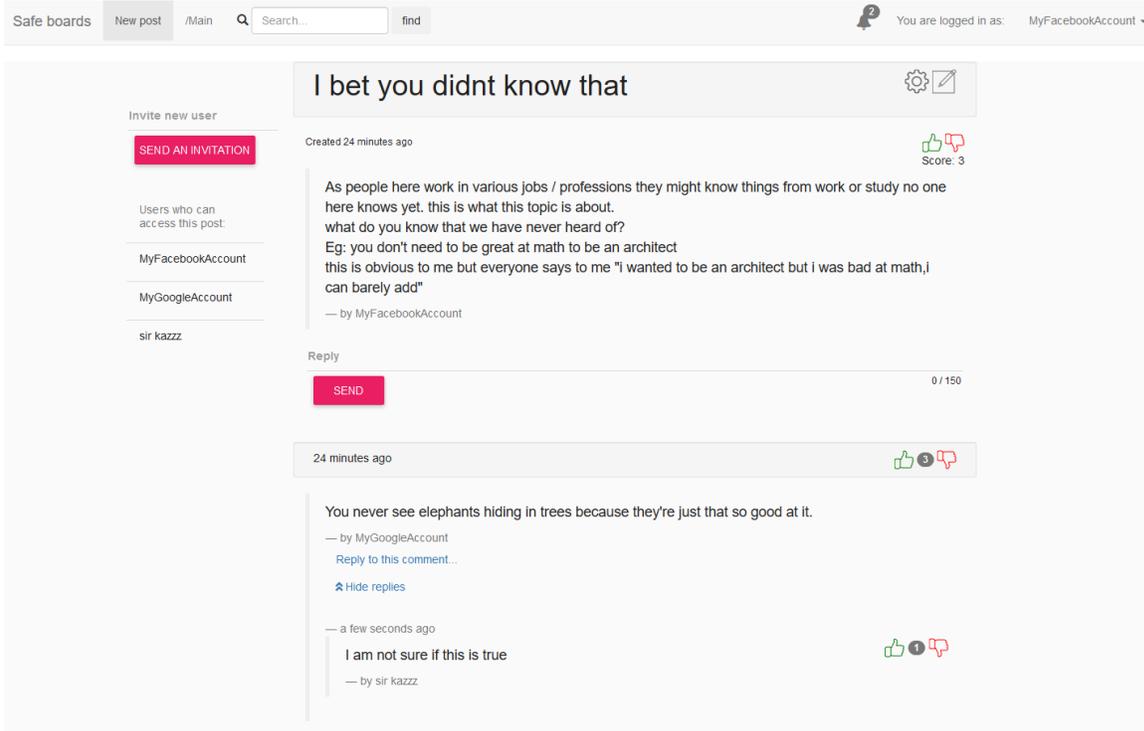
5.2.4 Single post view

Public post view

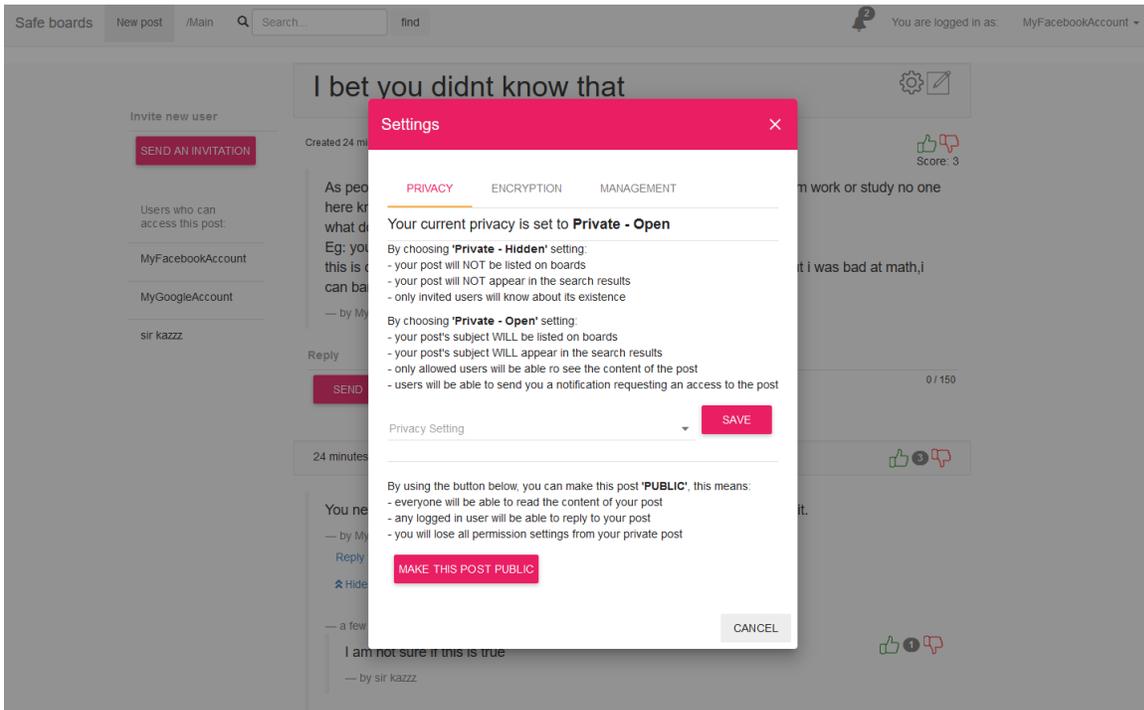
The screenshot shows a public post view in the Safe Boards application. The user is logged in as 'MyFacebookAccount'. The post title is 'I bet you didnt know that'. It was created 2 minutes ago and has a score of 1. The post content reads: 'As people here work in various jobs / professions they might know things from work or study no one here knows yet. this is what this topic is about. what do you know that we have never heard of? Eg: you don't need to be great at math to be an architect this is obvious to me but everyone says to me "i wanted to be an architect but i was bad at math,i can barely add"'. The post is by 'MyFacebookAccount'. There is a 'Reply' section with a 'SEND' button and a character count of '0 / 150'. Below the post, there is a comment from 'NewName' that says 'You never see elephants hiding in trees because they're just that so good at it.' and a link to 'Reply to this comment...'. The footer includes 'Safe Boards © 2018' and social media icons for Facebook, Twitter, and Instagram.

Safe Boards: Technical Report

Closed group view – as owner:

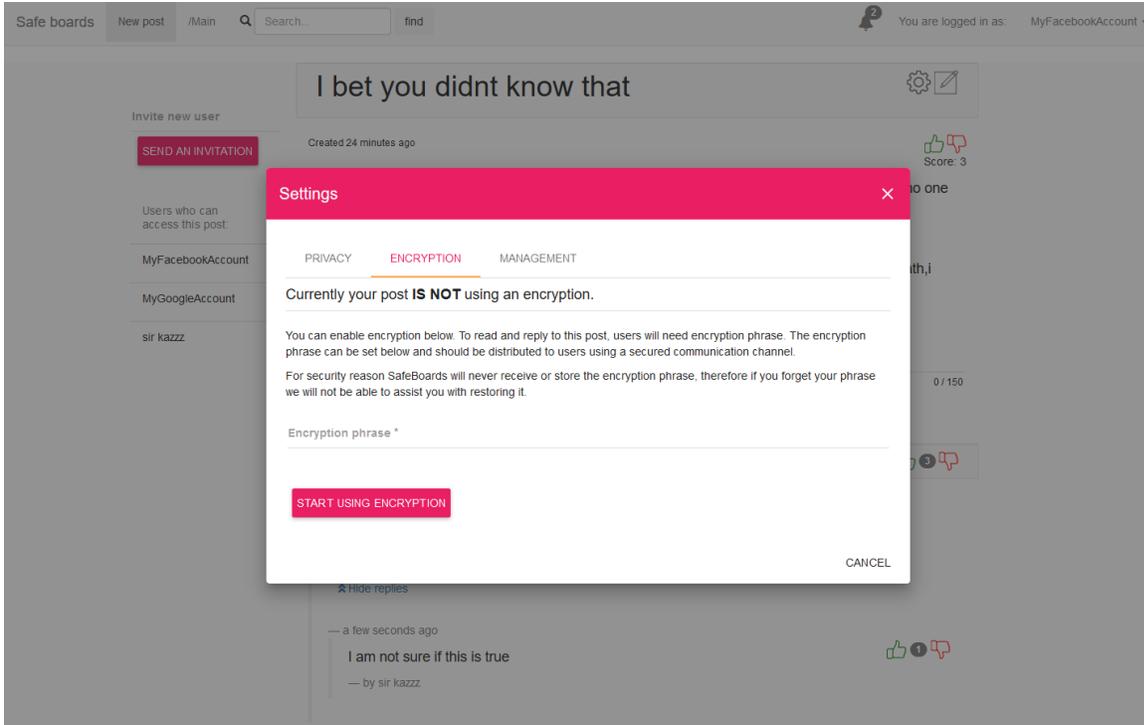


Posts settings - privacy:



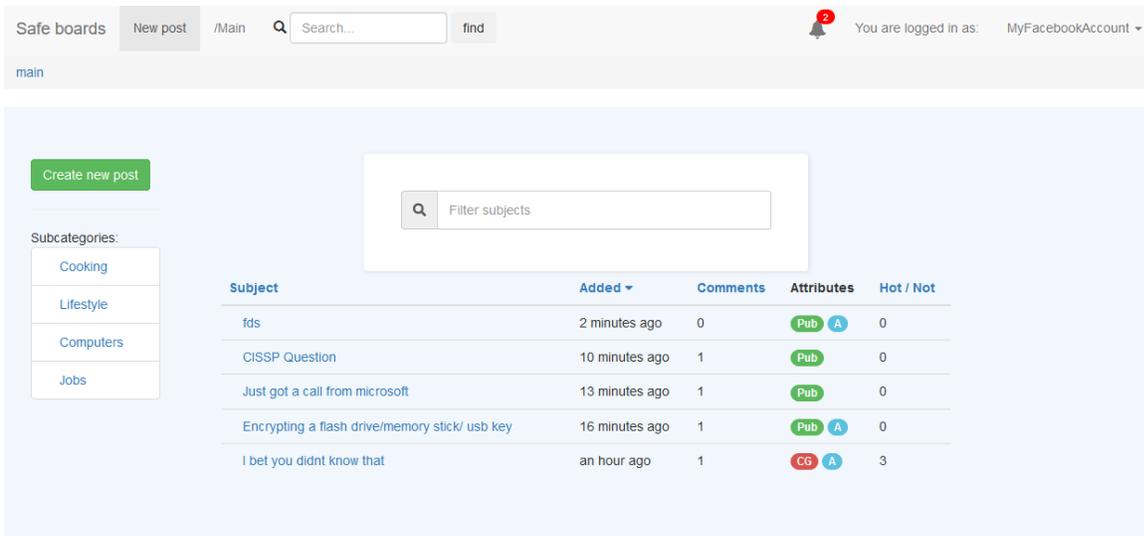
Safe Boards: Technical Report

Posts settings - encryption:



5.2.5 Viewing list of posts

Posts by category



Safe Boards: Technical Report

Users own posts or search results:

Subject	Added	Comments	Attributes	Hot / Not	Category
Encrypting a flash drive/memory stick/ usb key	18 minutes ago	1	Pub	0	main
I bet you didnt know that	an hour ago	1	CG	3	main

Safe Boards © 2018



5.2.6 Category management

Sub-categories

- Cooking
- Lifestyle
- Computers
- Jobs

Create new category

New category name...

CREATE

Safe Boards © 2018



6 Testing

6.1 *User acceptance testing*

In May 2018, the user acceptance testing (UAT) has been performed based on the list of use cases. The user has been asked to perform numerous tasks when an author of the application was noting the outcomes from the screen and from the user. Gathered information has been used to perform fixes and improvements, after which second session of UAT has been performed. After that just some minor issues have been found which were again fixed and reported to the user. After that user was satisfied with an outcome.

6.2 *Integration testing*

An integration testing has been performed in early January 2018 and all the front and back end packages were updated. This has been performed earlier than planned due to the fact that AUT would have to be performed after updates to ensure that all the functionality stayed intact.

6.3 *End-to-end testing*

End-to-end testing has been performed in May 2018. The test consisted of:

- inputting invalid values in the front end
- forging unauthorised API request using Postman
- posting cross site scripts to the forum
- non-SQL injections to the login

Multiple flaws were found during forging unauthorized API request. Those were patched by adding additional validation on server side, and successfully tested again using Postman.

An automated testing scripts for the back end were written using Mocha/Chai these were running regularly to ensure that any code changes does not affect the functionality of other elements.

6.4 *Customer testing*

A customer testing has been performed in May 2018 with use of forms (attached in appendix). The test has been performed on 15 users. The purpose of the test was to ask users to perform some tasks and report how user friendly was an interface. The feedback on each of 6 tasks was received in scale 1 to 6, but users were also able to provide their

Safe Boards: Technical Report

own feedback. Tasks were described only in high details to make sure that user will find a way around an application on their own.

Following results were received [in all cases lower score is better, range 1-5]:

User #	Task 1	Task 2		Task 3		Task 4	Task 5	Task 6
	Difficulty	Difficulty	Understanding	Difficulty	Understanding	Difficulty	Difficulty	Difficulty
User 1	1	1	1	1	1	1	2	2
User 2	1	1	2	1	2	3	1	1
User 3	2	2	2	2	2	2	2	2
User 4	1	1	1	1	1	1	1	1
User 5	1	1	1	1	1	1	1	1
User 6	1	1	1	1	1	1	2	1
User 7	1	1	1	1	1	2	1	1
User 8	2	2	2	2	2	2	3	3
User 9	1	1	1	1	1	1	1	1
User 10	2	2	2	2	1	1	2	1
User 11	3	3	4	3	4	4	4	3
User 12	1	1	1	1	1	1	2	2
User 13	3	3	3	3	3	3	2	2
User 14	1	1	1	1	1	2	1	2
User 15	1	1	1	1	1	1	1	1
Average	1.47	1.47	1.60	1.47	1.53	1.73	1.73	1.60

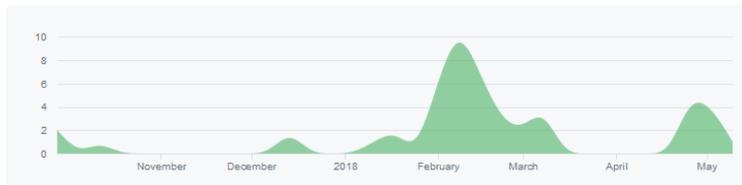
Above results indicates that in majority cases most of the users finds the application easy to navigate. Scores for understanding functionalities are a bit higher which may suggest that in the future it may be worth to put a bit more descriptive tooltips, information, and a section of frequently asked questions.

Further testing will be handled by creating a post where users can submit their feedbacks about the application, suggestions or ask questions about functionality.

6.5 Version control

A GitHub Repository has been used to keep the version control of the project. This has been helpful especially in testing process when some functionality has stopped working, and I was able to review what changes have been done since last upload.

Contributions to master, excluding merge commits

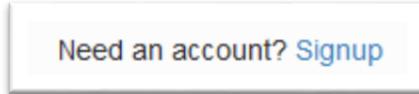


Total of 51 commits has been done over a period of 5 months with the main activity January 2018 to March 2018 and May 2018.

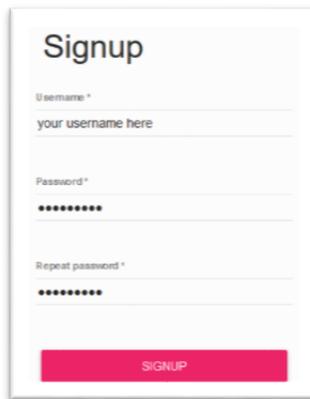
7 User manual

7.1 *Creating master account*

1. From the main page go to 'Login' and on the login page click 'Signup' link:

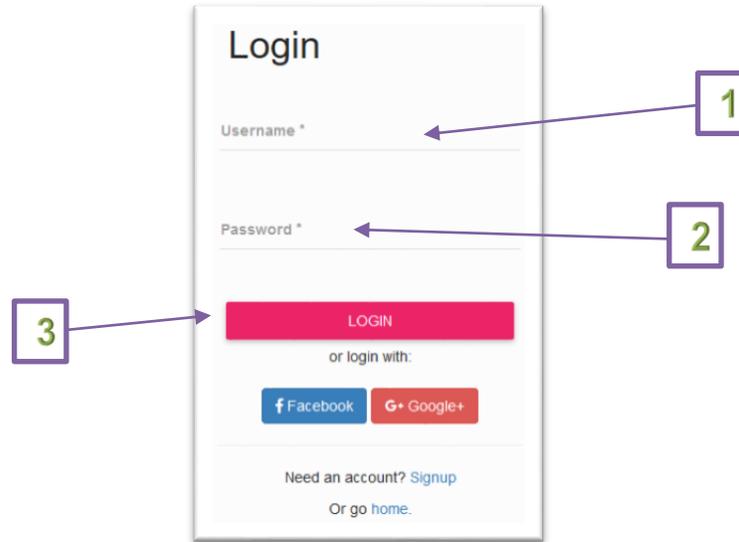


2. Provide your username and password and click SIGNUP button:

A white rectangular form titled "Signup" at the top. It contains three input fields: "Username *" with the placeholder text "your username here", "Password *" with a masked password of seven dots, and "Repeat password *" with a masked password of seven dots. At the bottom of the form is a red button with the text "SIGNUP" in white capital letters.

7.2 *Logging in with your created account*

- 1 On the main page click button
- 2 Fill in the username [1] and password [2] and click LOGIN button [3]

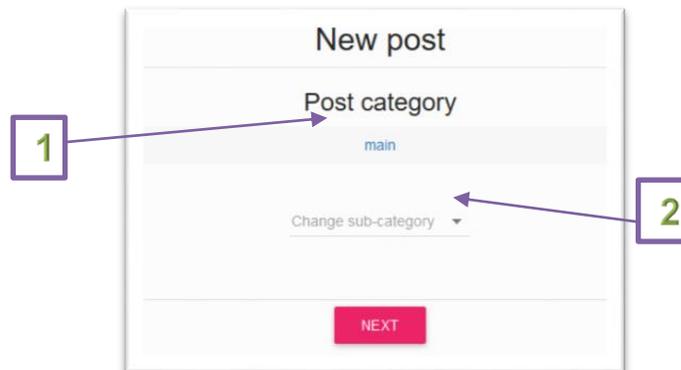


7.3 Login in with social media account

1. On the main page click  button
2. Click Facebook or Google button, and you will be taken to 3rd party login form.

7.4 Creating new post

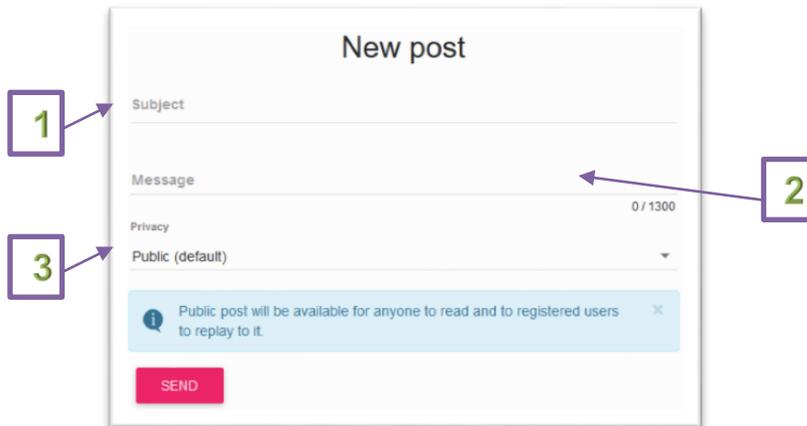
1. From any place on the website click  button located on the top of the screen.
2. Choose the category of the post you are going to create. You are starting from Main and you can choose sub categories. Once you chose subcategory breadcrumbs [1] and sub-categories in dropdown menu [2] should update



3. Once you pick your category click 'Next' button

Safe Boards: Technical Report

4. On next screen you can provide subject [1] of the post and a content of it [2]



The screenshot shows a 'New post' form with the following fields and elements:

- Subject:** A text input field with a callout box labeled '1' pointing to it.
- Message:** A larger text input field with a character count '0 / 1300' and a callout box labeled '2' pointing to it.
- Privacy:** A dropdown menu currently set to 'Public (default)' with a callout box labeled '3' pointing to it.
- Warning:** A light blue banner with an information icon and the text: 'Public post will be available for anyone to read and to registered users to replay to it.' with a close button 'x'.
- SEND:** A red button at the bottom of the form.

5. You should also pick your privacy. There are 3 privacy settings available:

Public post will be available for anyone to read and to registered users to replay to it.

Closed group - public post will be visible for users when searching but users will not be able to access it unless they are allowed by an author.

Closed group - hidden post will not appear on search results, therefore only allowed users will know about its existence, and will be able to access it.

6. If you choose 'Public' privacy, you should be able to submit the post straight away by clicking 'Send' button. Alternatively, if you choose one of the closed group settings you will have one more option which will allow you to enable encryption by clicking 'Disabled (default)' button [1].

7. If you wish to Submit your post without encryption you should leave the button as 'Disabled' and click send button

8. If you wish to enable encryption of your post change setting to 'Enabled'. This will display a warning message and input for secret phrase [2]. Please read all the information carefully and provide a secret. Entered secret will be used for encryption of the text.

Safe Boards: Technical Report

New post

Encryption

ENABLED

Warning!
Your post will be encrypted. To read and reply to this post, users will need encryption phrase. The encryption phrase can be set below and should be distributed to the users using a secured communication channel.
For security reason SafeBoards will not store the encryption phrase, and therefore if you forget your phrase we will not be able to assist you with restoring it.

Encryption secret...

SEND

7.5 Accessing posts

From the main page navigate to any categories of your choice [1] (on the left-hand side) and click the subject of the post that interest you [2]

Safe boards | New post | /Main | Search... | find | You are logged in as: MyGoogleAccount

main

Create new post

Filter subjects

Subcategories:

- Cooking
- Lifestyle
- Computers
- Jobs

Subject	Added	Comments	Attributes	Hot / Not
CISSP Question	13 hours ago	1	Pub A	0
Just got a call from microsoft	13 hours ago	1	Pub A	0
Encrypting a flash drive/memory stick/ usb key	14 hours ago	1	Pub	0
I bet you didnt know that	14 hours ago	1	CG	3

Safe Boards © 2018

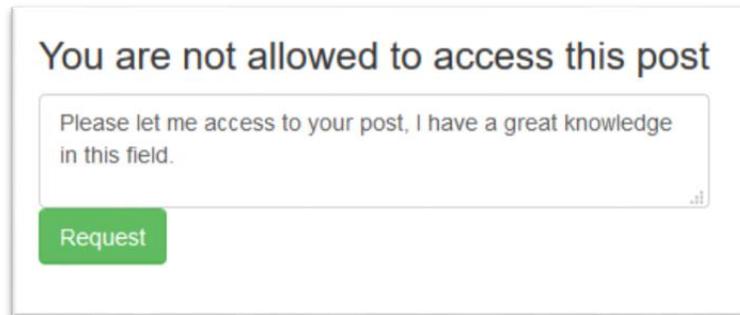
f t i

Attributes visible on the main page next to the posts will display the tooltips with their meanings:

- (Pub) – Public post
- (CG) – closed group visible to anyone
- (P) – closed group visible only to the members
- (E) – Post is encrypted, meaning you will need encryption phrase
- (A) – you are an author of the post

7.6 Accessing closed group when not a member

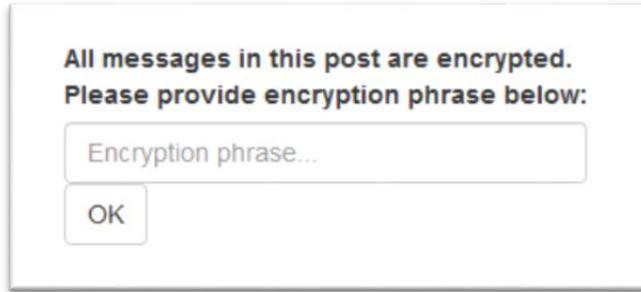
If you try to access a post which is accessible only to members you will be presented with the message below. This way you can send a message to the owner of the group and ask him to add you to the list of authorized users.



Your request will be revised by the owner of the post and whenever revoked or granted you will receive the appropriate notification.

7.7 Accessing encrypted post when member of the group

When you access closed group post which happens to be encrypted you will need a secret phrase to encrypt and access it. When accessing such post, you will be presented with following message:

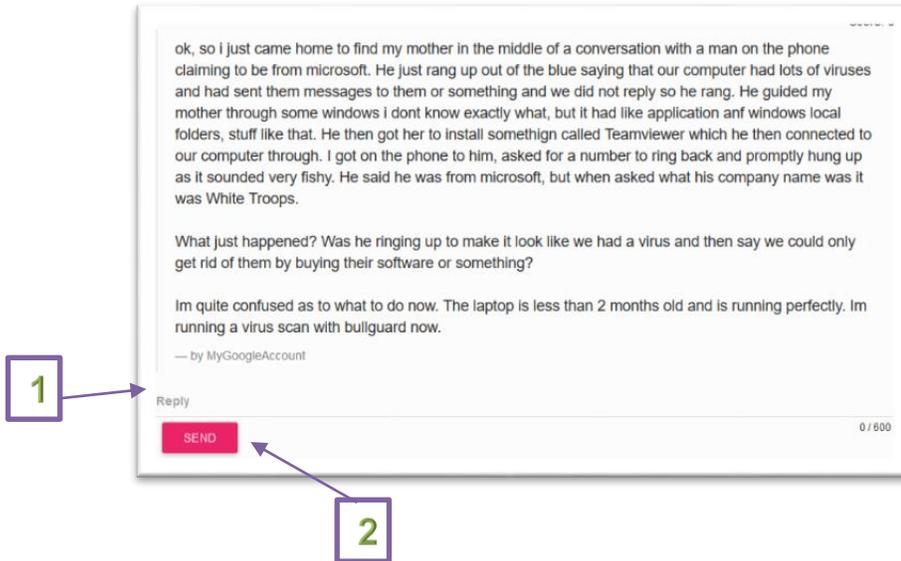


Entering a correct phrase and clicking 'OK' button will gain you an access to the post.

7.8 *Replying to the post or comment*

Once you access the post and you are logged in you can leave your comments to both a post itself and/or to the comments to the post left by other users.

To comment on post simply type your response to the 'reply' field [1] right under the post and click SEND button:

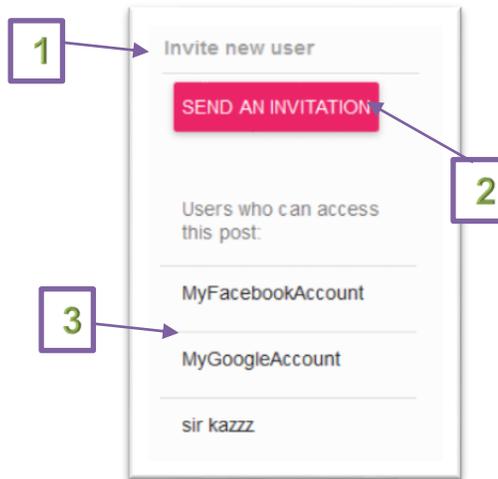


To reply on someone else's comment simply go to that comment and click '[reply to this comment...](#)' link below. This will display an input field like the one below, where you can type your response and slick SEND button



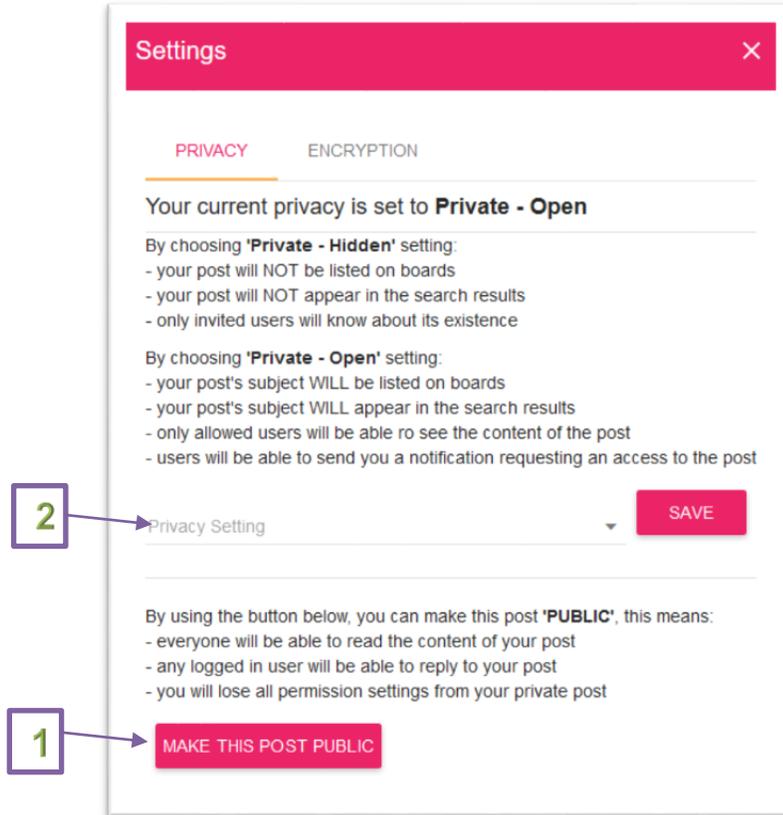
7.9 Inviting users to the closed group

When you are accessing your own closed group post, on the left-hand side you will be presented with field to invite users and a list of users who already have access to it [3]. If you wish to invite new user start typing in his username to the invite field [1] and you your field should start auto populating with the users matching your typed name. once you pick the right user click SEND AN INVITATION button [3]. This will create a notification to the invited user allowing him to accept or revoke your invitation. In both cases you will receive suitable notification.



7.10 Changing post privacy settings

If you are accessing your own post an additional gear icon will be displayed next to your post subject. Clicking this icon will bring a window with different post settings. First tab will allow you to change your privacy settings:



If your post is closed group you will get an option to both, change it to Public or changing the type of closed group. Changing post to public can be done by clicking MAKE THIS POST PUBLIC button [1] and changing from public to closed group or changing closed group type can be done from the 'Privacy settings' dropdown [2] and confirming it with SAVE button next to the dropdown.

7.11 Switching encryption on or off

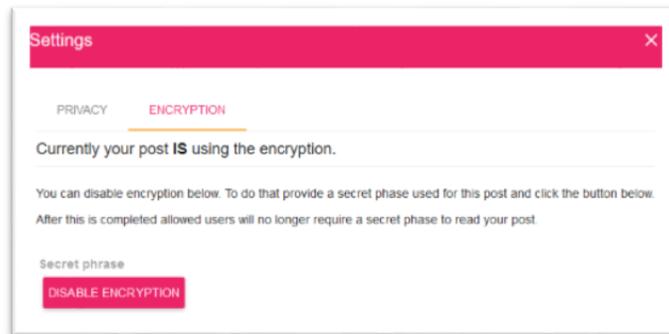
If you are accessing your own post an additional gear icon will be displayed next to your post subject. If your post is currently one of the closed group privacy type and you press the gear button a second tab called 'ENCRYPTION' will be available to you. This tab will allow you to turn the encryption on or off depending on your current state.

Turning encryption on will require you to provide a secret phrase [1] which you will have to distribute to your users for them to access the post, and clicking START USING ENCRYPTION button [2]:

Safe Boards: Technical Report



Turning encryption off will only require you to click the DISABLE ENCRYPTION button as the secret phrase has been provided already on loading the post.

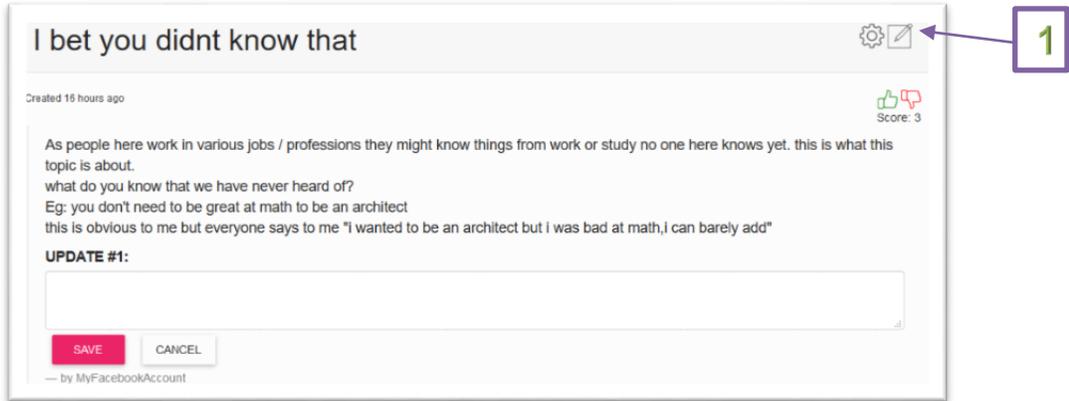


7.12 Editing the post

If you are accessing your own post an additional notepad icon [1] will be displayed next to your post subject. This icon will allow you to edit a content of your post for as long as no comments were added to the post. Clicking the icon will replace a content of your post with editable input pre-populated with current post content. From here you can edit your post and once done click save button.

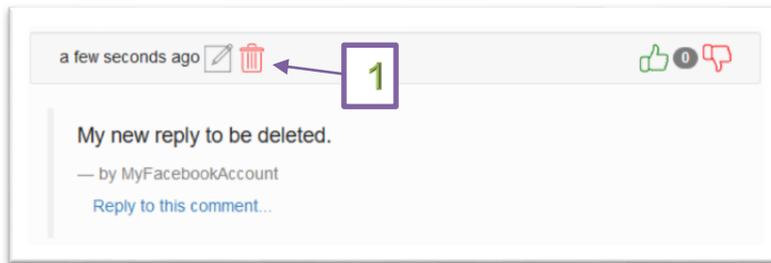
In the situation someone already replied to your comment you will not be able to change the content of your post as the response was to the original post therefore any altering of the post should not be allowed. However, you can still add an updated to your post which will be displayed under the original post.

Safe Boards: Technical Report



7.13 Deleting post or comment

You can delete your comment anytime by simply clicking a bin icon [1] next to your post



Deleting post is only allowed until there are no responses to the post, and it is done by the same type of icon as above but located next to the post subject.

7.14 Editing comment

You can edit your comment at any time by clicking a notepad icon next to your reply. This will replace current text of the reply with editable input field. Your changed then can be confirmed with SAVE button

7.15 Displaying your posts

You can access a list of all posts that you have created by clicking on your display name in the top right corner of the page and choosing 'My posts' position from the drop-down menu.

7.16 Notification

You can access your notifications from the drop-down menu under your display name [1] or if you have any new notifications you should be presented with a bell icon and the number of new notifications next to it [2]. This should be located on the top menu bar.



This will take you to the page displaying all your notifications. Please note that notifications that were not previously accessed will have a (NEW) icon next to them. Clicking any of the notification will display more details on it as well as interaction options such as approval for users' requests.



8 Conclusions

The Safe boards application main advantage and unique point is the possibility to secure the posts in the highly adjustable manner. In September 2017 Reddit forum had a 1.5billion unique visits, which shows the scale of the community gathered around the biggest boards. However, because it all public view of the posts administrators sometimes must ban some of the forums as it took place in November 2016 when Reddit banned the Pizzagate conspiracy board (Ohlheiser, 2016). Some users of the boards may have a political or religious view that are extremely different from most of community, and which may disturb those with opposite view. However Safe Boards will allow those groups off people to exchange their opinions without publicly reviling those conversation, by creating a closed group.

The biggest thread for the Safe Boards success may be a lack of initial interest in the functionality that it has to offer or the functionality not being enough handy to utilize. Without an interest there will be no content and without the content users will have no reason to visit. Therefore, it is very important to keep promoting Safe boards with its all unique selling points.

9 Further development or research

Further development of the Safe Boards may include but not to be limited to:

- Adding the functionality to follow the posts
- Adding a possibility to allow users other than post authors to invite to the closed groups
- Creating an area where users can see all their comments in one place
- Create a Frequently Asked Questions (FAQ) area
- Move the application to AWS C9 once they get all functionality from legacy C9 available (ticket with AWS open)
- Unify API responses. As the concept has changed overtime of the development some API responses are still have old response type. However, whatever is send by API is what is expected by front end, therefore a change to both ends will have to be performed.

10 References

Alexa.com. (2017). *Top Sites in United States - Alexa*. [online] Available at: <https://www.alexa.com/topsites/countries/US> [Accessed 21 Oct. 2017].

Cimpanu, C. (2017). *Cyberstalking Suspect Arrested After VPN Providers Shared Logs With the FBI*. [online] BleepingComputer. Available at: <https://www.bleepingcomputer.com/news/security/cyberstalking-suspect-arrested-after-vpn-providers-shared-logs-with-the-fbi/> [Accessed 21 Oct. 2017].

Ohlheiser, A. (2016). *Fearing yet another witch hunt, Reddit bans 'Pizzagate'*. [online] Washington Post. Available at: https://www.washingtonpost.com/news/the-intersect/wp/2016/11/23/fearing-yet-another-witch-hunt-reddit-bans-pizzagate/?utm_term=.901c817ee710 [Accessed 26 Nov. 2017].

Passport.js. (2018). *Documentation: OAuth 2.0*. [online] Available at: <http://www.passportjs.org/docs/oauth2-api/> [Accessed 7 May 2018]

11 Appendix

11.1 Project Proposal



11.2 Project Plan



11.3 Monthly Journals



11.4 Survey

