

Group Milestone 1 Report

GROUP 3 - BOXY

Prepared by:

- İrem Zeynep Alagöz
- Sevde Sarıkaya
- Kıymet Akdemir
- Kaan Dura
- Ufuk Arslan

Group name:

GROUP 3 - BOXY

Sub-group:

Backend:

- Sevde Sarıkaya
- Kaan Dura

Frontend:

- Ufuk Arslan
- Ufuk Karagöz
- Burak Onur Duman

Android:

- İrem Zeynep Alagöz
 - Kıymet Akdemir
 - Halil Baydar
-

Tag-id: <https://github.com/bounswe/2021SpringGroup3/releases/tag/v0.1>

Executive Summary

Project Description

This project is a social media platform which lets people benefit from cumulation of content while they are sharing experiences and knowledge. The main building block of the platform is data types (post types). Data types organize the posts on the platform. It makes communities more organized and allows people to find the content they are looking for faster. The communities allow people with similar interests and purposes to contact each other. Users can share and reach contents of common interests in the communities. There are different types of users with different privileges. Community moderators are the ones who created the community or are assigned by another community moderator. They can create data types so that the users post community specific contents. They are able to remove posts, comments or users from their communities if needed. They can also restrict access to the community via making it private. Private communities are only accessible to the community members and common users can join them only if the community moderators accept their requests. However, common users are able to join the public communities without a request. Users can post contents, comment and like the existing posts and report the inappropriate ones to the moderators. The things that are offered to the users are not limited to these. The platform provides search and filter mechanisms to the users so that they do not have difficulties with finding the communities and the users they want to interact with. They are able to follow other users, visit their profiles and message them privately. The platform, of course, does not disregard the privacy of its users. It allows users to change their privacy settings and let them decide who can follow, message and view their profiles. All in all, this application aims to reach the next level of what we know as social media by letting self-structured communities be formed by themselves.

Project Status

For the second milestone, we decided to complete the basic functionalities of a community and post based operations. We again implemented the mobile and frontend in parallel. First of all, we completed incomplete features from the M1 and improved the existing functionalities. The overall design of the application is improved. Create post and view post functionality is completed. Creating a community form is enhanced to fulfill the requirements.

Then we proceed with the new functionalities.

For the mobile application, we developed viewing communities (members, moderators, and all), viewing community detail, joining/leaving the community, applying to be moderator, viewing pending requests as a community moderator, accepting/rejecting joining and moderator requests, kicking member from community, updating community as moderator, homepage, view own/other profile, update profile, delete account.

For the web application, we developed viewing communities (members, moderators, and all), viewing community detail, view own/other profile, update profile, create post, view post detail.

Besides, we have added mapping functionality to our application using Google Maps API. Users are now able to specify locations as latitude longitude and view locations on the map. Details are explained in the "Implementing Core Functionality" section.

As a result, we have made a social media application that fulfills the requirements in general terms. Users can create communities, post types, posts. They can view and use created entities. They can view profiles and edit their profiles. Hence we have an organized social media platform where users can enjoy sharing and surfing.

The missing features and how we will complete them are explained in the road ahead section.

Changes we made after Milestone 1

After Milestone 1, we reviewed our plans to finish at least two days before the presentation. We tried to hold regular subteam meetings and review sessions. In this way, we aimed to accelerate the implementation process by supporting each other.

Prior to Milestone 1, swagger specification documents were co-created by the mobile and web teams. However, many changes had to be made to the specification when the backend started implementing it. For this reason,

We updated the sorting so that the backend gets the job first. As for front-end teams, we waited for the back-end team to complete the implementation before starting it.

Thanks to these improvements, we were able to implement more reliable and faster implementation.

Our Milestone 1 demo presentation was not very successful. Therefore, this time we designed a scenario in advance and created the necessary data. Before the presentation, we gained practice by practicing presentations. In this way, we were able to make a more fluent and effective presentation.

Road Ahead

Since the start of this course, each of our team members has gained a great deal of experience developing their assigned features, so this will definitely help us speed up building the next features of our projects. Our main goal is to complete most of our project requirements by the next milestone.

While completing our requirements our main focus will be the notification, search and tagging system. The notification system will be implemented using W3C Activity Stream Standard 2.0. The joining/leaving a community, accepting/rejecting requests will be updated based on our activity stream design. Also following a user and chatting functionalities will be implemented based on the design. The Wikidata will be used when the tagging post functionality is implemented.

According to our project plan, our Android and Web teams will complete the implementation of "Notifications", "Search Results", "Chat" pages by the end of the next milestone. Backend team will complete the implementation of "Notification", "Search",

“Filtering”, “Reporting”, and the “Service, Controller, and Data Transfer Classes” that frontend pages require.

Also, we are planning to improve our create post-type functionality based on the feedback we got during the milestone 2 presentations.

We will keep the communication within and between teams strong and continue having meetings to make sure that tasks are distributed evenly among all members, and make sure that everyone is on the same page. We are looking forward to implementing the next features that we have planned, and keeping up the good work.

Progress based on teamwork

Table of Work Done by Each Member

Member Name	Contributions
Kıymet Akdemir	<ul style="list-style-type: none">• Participating in the weekly meetings• Participating in the Frontend & Mobile meetings• Participating in the Mobile meetings• Designing and implementing Post Details page for mobile
İrem Zeynep Alagöz	<ul style="list-style-type: none">• Mobile: Create Post• Mobile: Community Page• Mobile: Create Community• Mobile: Community Detail Page• Mobile: Update Create Post Type Design• Mobile: Join Community• Mobile: Settings Page• Mobile: Page Header Component• Mobile: Community Detail - About & Members Tabs• Mobile: Accept-Reject Join Community Requests• Mobile: Leave Community• Mobile: Kick Member From Community• Mobile: Update Community• Mobile: Apply to be a moderator

	<ul style="list-style-type: none"> • Mobile: Accept-Reject Join Moderators Requests • Mobile: Delete Account Functionality • Mobile: Get Community Posts • Creating Scenario & Data for the M2 Demo Issue #390 • Reviewing Mobile: Post Detail, Backend: Location Field in Post and Profile is improved, Updating Swagger Doc, Backend: return-location fix, Backend: Community and Post Type Model Changes, Join Request, Backend: Kick from Community, Backend: Approve and Reject Endpoints for Join Requests, Backend: Endpoint to Become a Mod, Backend: Approve and Reject Endpoints for Moderator Requests, Mobile: Profile view screen and profile settings screen is done, Mobile: Home Page, Backend: Bug Fix • Milestone 2: Project Status Issue #404 • Milestone 2: Changes we made after Milestone 1 Issue #405 • Milestone 2: Road Ahead Issue #406 • Milestone 2: Implementation of Core Functionality Issue #407
Ufuk Arslan	<ul style="list-style-type: none"> • Participating in the Frontend Meetings • Taking notes and documenting frontend meetings • Implementation of the Create Post page in web • Implementation of the Post View page in web • Enhanceing of Navbar and community selection in web • Connecting post and user routes. • Updating swagger documentation. • Reviewing Profile settings and view feature in web • Reviewing enhancement on about community component in web
Halil Baydar	<ul style="list-style-type: none"> • Participating in the weekly meetings • Attending in frontend and mobile meeting • Building the apk file in release mode • Opening issues and pull requests • Reviewing pull requests • Documenting meeting note
Burak Onur Duman	<ul style="list-style-type: none"> • Participating in the Frontend Meetings • Implementation of the View Profile Page • Implementation of the Edit Profile Page • Implementation of the Enhancement of About Community Component • Deployed the React App to GitHub Pages at https://jcbod.github.io/my_app

	<ul style="list-style-type: none"> • Reviewing the Backend Implementation of Viewing Other Profiles • Reviewing the Backend Implementation of Leaving Communities
Kaan Dura	<ul style="list-style-type: none"> • Participating in weekly general meetings • Participating in backend meetings • Revising the project requirements • Updating Swagger documentation • Nodejs & MongoDB Research & Study • Implementing general base & boilerplate of the backend • Implementing Auth Controller • Implementing Community Controller • Implementing Post Type Controller • Implementing Post Controller • Reviewing pull requests • Configuring & deploying MongoDB database • Implementing a Github Action for auto deployment of backend
Ufuk Karagöz	<ul style="list-style-type: none"> • Participating in weekly general meetings • Participating in frontend and mobile meeting • Participating in the frontend meeting • Revising the project requirements • Revising and updating use case diagram • Reviewing and updating Mockup Scenario #3 • Reviewing and updating sequence diagram: Like Post • Learning to use React • Designing and implementing registration page for Web • Reviewing Swagger documentation. • Designing the post view component for Web • Designing the create post component for Web • Designing the create post form component for Web • Designing the create post page for Web • Implementing Redux statements for Web • Implementing Axios API handling for Web • Implementing Routing for pages • Implementing Private and Public Routing for pages • Designing and implementing register API handling • Designing and implementing login API handling • Designing and implementing logout API handling • Designing and implementing create community API handling • Designing and implementing create post type API handling • Designing and implementing get communities component

Sevde Sarıkaya	<ul style="list-style-type: none">• Participating in the weekly meetings• Attending the Backend Meetings• Implementing & Testing Viewing Other Profiles API• Implementing & Testing Leaving Community API• Implementing & Testing Like Post API• Adding Moderators and Members Fields to Community Detail & Testing• Updating Swagger documentation• Adding Like Count to Post Detail & Testing• Backend: Community Model Fix• Bug fix: Kick member• Formatter & validator fix for description of community• Adding location fields to post and profile models & Testing• Adding location fields and controls to the necessary API requests and responses & Testing• Community model fix: description and isPrivate added & Testing• Milestone 2: API Documentation
----------------	---

Kaan Dura	<ul style="list-style-type: none"> • Participating in the weekly meetings • Attending the Backend Meetings • Reviewing the requirements • Reviewing the use case diagram • Creating project plan for backend** • Research & Study NodeJs with tutorials and coding • Arrangements on Backend Spec on Swagger. • Implementing Profile delete endpoint • Implementing Community delete endpoint • Implementing Post delete endpoint • Implementing Post like endpoint • Implementing Post comment endpoint • Implementing Community update endpoint • Implementing functionality for creating image from base64 string user provides • Implementing Community join endpoint • Implementing Community moderator join endpoint • Implementing Community approve join request endpoint • Implementing Community reject join request endpoint • Implementing Community moderator reject join request endpoint • Implementing Community moderator reject join request endpoint • Implementing formatters for these endpoints • Implementing validations for these APIs. • Testing the Backend endpoints by sending requests from Postman • Testing the Swagger doc by sending requests from Swagger doc • Creating samples for the demo
-----------	---

The Status of Requirements

☒ Completed

☐ In progress

☐ Not started

Functional Requirements

• 1. User Requirements

• 1.1. Basics

○ 1.1.1. Registration

☒ ~~1.1.1.1. Users shall be able to register by providing a valid username, e-mail address, and password and accepting the terms of GDPR and KVKK.~~

☐ 1.1.1.2. After completing step 1.1.1.1, users shall validate their e-mail addresses to complete the registration process.

○ 1.1.2. Login/Logout

☒ ~~1.1.2.1. Users shall be able to log in via their registered username/e-mail address and password.~~

☒ ~~1.1.2.2. Users shall be able to log out after logging in.~~

○ 1.1.3. Account Management

☐ 1.1.3.1. Users shall be able to reset their passwords by using the authentication link in the verified email.

☒ ~~1.1.3.2. Users should be able to delete their accounts.~~

○ 1.1.4. Profile

☒ ~~1.1.4.1. Users shall have their usernames and default grey picture on their profile pages.~~

☒ ~~1.1.4.2. Users shall be able to add profile pictures to their profile pages.~~

☒ ~~1.1.4.3. Users shall be able to add/edit their personal information to their profile pages. Personal information shall include name, bio, birthday, location.~~

☒ ~~1.1.4.4. Users shall have public profiles as default.~~

☒ ~~1.1.4.5. Users shall be able to change the privacy setting (public or private) of their birthdays, profile pictures, location, and bio.~~

-

1.2. Community

○ 1.2.1 Search & Filter

- ☐ 1.2.1.1. Users shall be able to search for communities using a search bar.
- ☐ 1.2.1.2. Users shall be able to filter communities as implied in 2.4.3.
- ☐ 1.2.1.3. Users shall be able to search for posts using a search bar.
- ☐ 1.2.1.4. Users shall be able to filter posts as implied in 2.4.4.
- ☐ 1.2.1.5. Users shall be able to search for a data type using a search bar.
- ☐ 1.2.1.6. Users shall be able to filter data types as implied in 2.4.4.

○ 1.2.2 Joining

- ☒ ~~1.2.2.1. Users shall be able to join communities to be able to post content and receive notifications about updates.~~
- ☐ 1.2.2.2. Users shall be able to turn on or off notifications.
- ☐ 1.2.2.3. Users shall be able to invite their friends to the community.

○ 1.2.3 Posting Content

- ☒ ~~1.2.3.1. Users shall be able to share posts on communities.~~
 - ☒ ~~1.2.3.1.1. Users shall be able to select a data type for their posts.~~
 - ☒ ~~1.2.3.1.2. Users shall be able to include images, numbers, date, location and text as data type fields in posts.~~
- ☐ 1.2.3.2. Users shall be able to add tags to their posts.
- ☒ ~~1.2.3.3. Users shall be able to create new data types.~~
 - ☒ ~~1.2.3.3.1. Users shall be able to define multiple attributes for data types. Attributes shall be images, numbers, date, location or text.~~
- ☒ ~~1.2.3.4. Users shall be able to see data types that exist in a community as a list.~~

-
- 1.2.4 Liking & Commenting
 - ☐ 1.2.4.1. Users shall be able to comment on posts.
 - ☐ 1.2.4.2. Users shall be able to filter comments as implied in 2.4.5.
 - ☐ 1.2.4.3. Users shall be able to reply to comments on posts.
 - ☐ 1.2.4.4. Users shall be able to like posts on communities.
 - ☐ 1.2.4.5. Users shall be able to like comments on posts.
 - 1.2.5 Reporting
 - ☐ 1.2.5.1. Users shall be able to report offensive & inappropriate posts.
 - ☐ 1.2.5.2. Users shall be able to report offensive & inappropriate comments.
 - 1.2.6 Creating
 - ☒ ~~1.2.6.1. Users shall be able to create a community with a unique community name.~~
 - ☒ ~~1.2.6.2. Users shall be able to add a picture for the community.~~
 - ☒ ~~1.2.6.3. Users shall be able to add a description to the community.~~
 - ☒ ~~1.2.6.4. Users shall be able to set the privacy of the community as public or private.~~

•

1.3. User Interaction

- 1.3.1. Search & Filter
 - ☐ 1.3.1.1. Users shall be able to search for other users' profiles as 2.4.1 implies.
 - ☐ 1.3.1.2. Users shall be able to sort/filter their search results on users' profiles as 2.4.2 implies.
 - ☐ 1.3.1.3. Users shall be able to choose to view other users' profiles which are listed as search results. (they can view from community members list)
- 1.3.2. Profile View & Follow

☐ 1.3.2.1. Users shall be able to view other users' private profiles which they follow and other users' public profiles. *(finished except the follow part)*

☐ 1.3.2.2. Users shall be able to choose to follow other users' public profiles.

☐ 1.3.2.2.1 Users who have public profiles should get notifications when they are followed by other users.

☐ 1.3.2.3. Users shall be able to send a request to other users who have private profiles, to follow their profiles.

☐ 1.3.2.3.1 Users who have private profiles should get notifications when they are requested to get followed by other users.

☐ 1.3.2.4. Users shall be able to unfollow users which they follow already.

○ 1.3.3. Private Messaging

☐ 1.3.3.1 Users shall be able to message other users.

☐ 1.3.3.1.1 Users should get notifications when they got messages from other users as 2.2.1 implies.

○ 1.3.4. Reporting

☐ 1.3.4.1 Users shall be able to report other users who are behaving inappropriately.

☐ 1.3.4.2 Users shall be able to report community moderators.

•

1.4. User Roles

○ 1.4.1. Common Users

☒ ~~1.4.1.1 Common users shall complete the registration process as 1.1.1 implies.~~

☒ ~~1.4.1.2 Common users shall be able to view public user profiles and community content read-only.~~

-
- ☐ 1.4.1.3 Common users shall be able to use searching functionality.
 - ☒ ~~1.4.1.4 Common users shall be able to log in/log out as explained in 1.1.2.~~
 - ☐ 1.4.1.5 Common users shall be able to manage their accounts as 1.1.3 implies. (*Reset password is not implemented yet*)
 - ☒ ~~1.4.1.6 Common users shall have a profile page as explained in 1.1.4.~~
 - ☒ ~~1.4.1.7 Common users shall be able to join public communities directly.~~
 - ☒ ~~1.4.1.8 Common users shall be able to send a request to join private communities.~~
 - 1.4.2. Community Members
 - ☒ ~~1.4.2.1. Community members shall be able to have all the privileges of a common user.~~
 - ☒ ~~1.4.2.2. Community members shall be able to post content as explained in the 1.2.3 section except 1.2.3.3.~~
 - 1.4.3. Community Moderators
 - ☒ ~~1.4.3.1. Community moderators shall be able to have all privileges of community members.~~
 - ☒ ~~1.4.3.2. Community moderators shall be able to create data types as explained in 1.2.3.3.~~
 - ☐ 1.4.3.3. Community moderators shall be able to review reports.
 - ☐ 1.4.3.4. Community moderators shall be able to remove posts, comments, users from their communities. (*only removing users*)
 - ☒ ~~1.4.4.2. Community moderators shall be able to assign a new moderator.~~
 - ☒ ~~1.4.4.3. Community moderators shall be able to change the community between public & private.~~

-
- ☐ 1.4.4.4. Community moderators shall be able to close their communities.
 - 1.4.4 Admin Users
 - ☐ 1.4.5.1. Admin shall be able to have all privileges of community moderators.
 - ☐ 1.4.5.2. Admin shall be able to delete communities and also delete users

● 2. System Requirements

● 2.1. Recommendation

- ☐ 2.1.1. The system shall recommend communities and posts to users based on the communities they are member of and the posts they like.
- ☐ 2.1.2. The system shall provide trending posts and communities to users.
- ☐ 2.1.3. The system shall recommend the posts of followed users.
- ☐ 2.1.4. The system shall recommend the communities of followed users.

●

2.2. Notification

- ☐ 2.2.1. The system shall notify users when they get private messages.
- ☐ 2.2.2. The system shall notify users when their posts get a like/dislike.
- ☐ 2.2.3. The system shall notify users when someone comments on their posts.
- ☐ 2.2.4. The system shall notify community moderators when a user requests to join their private communities.
- ☐ 2.2.5. The system shall notify users when someone posts in a community they are a member of.

●

2.3. Community

- ☒ ~~2.3.1. The system shall disable users if they try to perform unauthorized actions.~~

-
- ☒ 2.3.2. ~~The system shall allow community moderators to create new data types.~~
 - ☒ 2.3.3. ~~The system shall not allow duplicate community names.~~
 - ☒ 2.3.4. ~~Users' accounts shall be deleted if they behave offensively.~~

•

2.4. Searching / Filtering / Sorting

- ☐ 2.4.1. The system shall have a semantic search mechanism.
- ☐ 2.4.2. The system shall allow users to sort other users by a number of followers and filter by their communities.
- ☐ 2.4.3. The system shall allow users to sort communities by number of members, number of posts, and filter by tags.
- ☐ 2.4.4. The system shall allow users to sort posts by the number of likes, dates, and filter by data types and tags.
- ☐ 2.4.5. The system shall allow users to sort comments by a number of likes and dates.
- ☐ 2.4.6. The system shall allow users to filter results as communities, users, and posts.
- ☐ 2.4.7. The semantic taggings should be supported with [Wikidata](#).

•

2.5. Security

- ☐ 2.5.1. In the case of a password change attempt, the system should send a verification code to the e-mail address of the account owner.

API Endpoints

Backend API Documentation:

During our Backend Implementation, we used Swagger as our API Documentation. You can see detailed requests and try it yourself on our Swagger.

Swagger Doc Link : <https://api.cmpegrouphthree.store/docs/>

Auth Controller:

Name	Req Type	URI	Header	Parameters	Req body:
Register	Post	/auth/register	X-Platform: Android or Web	-	Username: String, Email: String, Password: String
Login	Post	/auth/login	X-Platform: Android or Web	-	Username: String, Password: String
Logout	Get	/auth/logout	Token, X-Platform: Android or Web	-	-
Change Password	Post	/auth/changePassword	Token, X-Platform: Android or Web	-	Password: String

Register: Creates a new user. Email and username should be unique. Password should have at least 8 characters and it needs to include a number

Login: Checks if given username and password matches, creates a new session. If it doesn't match, throws an error

Logout: Deletes the current session of the user

Change Password: Changes the password of the user. Password should have at least 8 characters and it needs to include a number

Community Controller:

Name	Req Type	URI	Header	Parameters	Req body:
Create Community	Post	/communities	Token X-Platform: Android or Web	-	name: String, description: String, isPrivate: Boolean
Update Community	Post	/communities/update	Token X-Platform: Android or Web	communityId	name: String, description: String, isPrivate: Boolean
Delete Community	Delete	/communities	Token, X-Platform: Android or Web	communityId	-
Get Communities	Get	/communities	Token, X-Platform: Android or Web	isMember,isMod	-
Get Community Detail	Get	/communities/detail	Token, X-Platform: Android or Web	communityId	-
Leave Community	Post	/communities/leave	Token, X-Platform: Android or Web	communityId	-
Kick from Community	Post	/communities/kick	Token X-Platform: Android or Web	communityId, userId	-
Join community	Post	/communities/join	Token X-Platform: Android or Web	communityId	-
Join community moderators	Post	/communities/moderators/join	Token X-Platform: Android or	communityId	-

			Web		
Approve community join	Post	/communities/join/approve	Token X-Platform: Android or Web	communityId, userId	-
Reject community join	Post	/communities/join/reject	Token X-Platform: Android or Web	communityId, userId	-
Approve community Moderators join	Post	/communities/moderators/approve	Token X-Platform: Android or Web	communityId, userId	-
Reject community join	Post	/communities/moderators/reject	Token X-Platform: Android or Web	communityId, userId	-

Create Community: Creates a community. Name should be unique

Update Community: Updates a community. Name should be unique. If the community doesn't exist, it throws an ApiError.

Delete Community: Deletes given community with communityId parameter. If the community doesn't exist, it throws an ApiError.

Get Communities: Returns communities based on isMember and isMod fields. This fields are not required

Get Community Detail: It takes communityId as parameter. By finding the corresponding community, it returns the detailed information.

Leave Community: It takes communityId. If user is a member in that community, removes him from members list. If not, throws an error

Kick from Community: It takes communityId and userId. It enables a moderator of the community to kick a member from members list. If the given userId is not in the list it throws an error

Join Community: It takes communityId. If the community is private puts user into pendingMembers list. Otherwise adds him to members list. Mods should decide if the community is private.

Join Community Moderators: It takes communityId. Puts user into pendingModerators list. Mods should approve or reject this request

Approve Join Community: It takes communityId and userId. If given user is a pendingMember of that community removes him from that list and adds him to members. You should be a mod to use this endpoint

Reject Join Community: It takes communityId and userId. If given user is a pendingMember of that community removes him from that list. You should be a mod to use this endpoint

Approve Join Community Moderators: It takes communityId and userId. If given user is a pendingModerator of that community removes him from that list and adds him to moderators. You should be a mod to use this endpoint

Reject Join Community Moderators: It takes communityId and userId. If given user is a pendingModerator of that community removes him from that list. You should be a mod to use this endpoint

Post Controller:

Name	Req Type	URI	Header	Parameters	Req body:
Create Post	Post	/posts	Token X-Platform: Android or Web	-	communityId: String, postTypeId: String, textFields: list of name-value pairs numberFields: list of name-value pairs, dateFields: list of name-value pairs, linkFields: list of name-value pairs , LocationFields: list of location objects*,

					Tags: list of strings
Delete Post	Delete	/posts	Token, X-Platform: Android or Web	postId	-
Get Posts	Get	/posts	Token, X-Platform: Android or Web	communityId	-
Get Post Detail	Get	/posts/detail	Token, X-Platform: Android or Web	communityId, postId	-
Like Post	Post	/posts/like	Token, X-Platform: Android or Web	communityId, postId	-
Create Comment	Post	/posts/comment	Token X-Platform: Android or Web	-	Text, postId

Create Post:

Request Body Example:

```
{
  "communityId": "string",
  "postId": "string",
  "textFields": [
    {
      "name": "turnuva adi",
      "value": "İstanbul Satranç turnuvası"
    }
  ],
  "numberFields": [
    {
```

```
    "name": "turnuva sayısı",
    "value": 5
  },
  "dateFields": [
    {
      "name": "turnuva tarihi",
      "Value": 12.12.12
    }
  ],
  "linkFields": [
    {
      "name": "turnuva linki",
      "value": "turnuva.com"
    }
  ],
  "locationFields": [
    { "name": "turnuva yeri",
      "value": { "geo": {
        "latitude": 3.435345,
        "longitude": 3.435345
      }},
      "description": "text"
    }
  ]
}
```

Create Post API, creates a post with the given PostType and its fields. Each name in the fields objects corresponds the field name in the chosen post type. Each value given by the user gets mapped with the name. Eventually, the post is created.

Delete Post: Delete post API gets the postId as a parameter and removes the post object and its comments from the database .

Get Posts: Get Posts API takes communityId as a parameter and returns the posts belonging to the given community. If the community doesn't exist, it throws an ApiError.

Get Post Detail: It takes communityId and postId as parameters. By finding the corresponding post, it returns the detailed information.

Like Post: It takes communityId and postId. By finding the corresponding post and user, it adds the current user as a liker of the post.

Create Comment: It takes postId and text as a request body. It creates a Comment object with the text and the current user.

Post Type Controller:

Name	Req Type	URI	Header	Parameters	Req body:
Create Post-Type	Post	/post-types	Token X-Platform: Android or Web	-	communityId: String, name: String, textFieldNames: list of name-value pairs numberFieldNames: list of name-value pairs, dateFieldNames: list of name-value pairs, linkFieldNames: list of name-value pairs , LocationFieldNames: list of location objects*,

Get Post Types	Get	/post-types	Token, X-Platform: Android or Web	communityId	-
Get Post Type Detail	Get	/post-types/detail	Token, X-Platform: Android or Web	communityId, postTypeId	-

Create Post Type:

Request body example:

```
{
  "name": "turnuva",
  "communityId": "618fb28cd5da23b323f29dbe",
  "textFieldNames": [
    "Turnuva Adı"
  ],
  "numberFieldNames": [
    "Katılımcı Sayısı"
  ],
  "dateFieldNames": [
    "Turnuva Tarihi"
  ],
  "linkFieldNames": [
    "Kayıt linki"
  ],
  "locationFieldNames": [
    "Turnuva yeri"
  ]
}
```

Create Post type API creates a post type object with the given field names in the given community with the given name.

Get Post Types: Returns the post types in the current community

Get Post Type Detail: Returns the given post type object with the detailed information.

Example:{

```
"name": "turnuva",
"communityId": "618fb28cd5da23b323f29dbe",
"textFieldNames": [
  "Turnuva Adı"
],
"numberFieldNames": [
  "Katılımcı Sayısı"
],
"dateFieldNames": [
  "Turnuva Tarihi"
],
"linkFieldNames": [
  "Kayıt linki"
],
"locationFieldNames": [
  "Turnuva yeri"
]
}
```

Profile Controller:

Name	Req Type	URI	Header	Parameters	Req body:
------	----------	-----	--------	------------	-----------

Get Profile	Get	/profile	Token, X-Platform: Android or Web	-	-
Get Other Profile	Get	/profile/other	Token, X-Platform: Android or Web	userId	-
Get Profile Settings	Get	/profile/settings	Token, X-Platform: Android or Web	-	-
Set Profile	Post	/post-types/settin gs	Token, X-Platform: Android or Web		<pre>Example: { "profilePhotoUrl": { "value": "text", "isPublic": true }, "bio": { "value": "text", "isPublic": true }, "birthday": { "value": "text", "isPublic": true }, "location": { "value": { "latitude": 3.435345, "longitude": 3.435345 }, "isPublic": true, "description": "text" } }</pre>
Delete Profile	Delete	/profile/settings	Token, X-Platform: Android or Web	-	-

Get Profile: It returns the user's profile information.

Delete Profile: It deletes the profile and removes the user from the communities where she/he is a member or moderator. It deletes the user's posts, comments.

Get Profile Settings: It returns the profile settings of the user. Example:

```
{
  "profilePhotoUrl": {
    "value": "text",
    "isPublic": true
  },
  "bio": {
    "value": "text",
    "isPublic": true
  },
  "birthday": {
    "value": "text",
    "isPublic": true
  },
  "location": {
    "value": {
      "latitude": 3.435345,
      "longitude": 3.435345
    },
    "isPublic": true,
    "description": "text"
  }
}
```

SetProfile: It updates the user information with the given values.

Get Other Profile: It returns someone else's profile but it checks if the user's fields are public or not. If a field is not public, it doesn't return.

Example API Calls:

Postman Collection Link:

<https://www.getpostman.com/collections/a790711decec529588a9>

In the collection, you can find how to register, login, create community, create post-type, get post type details, create post and get posts API calls.

To be able to use these API calls, you need to login and put the token value in the response to the header Authorization value part. You can try the APIs with different values and fields.

Our core functionalities are creating post-types, creating posts, viewing post types and posts because post type is where our software differs from others. It makes it easier to search and create posts. It makes communities more organized.

Implementation of core functionality

Status of The Mapping Functionality

For many reasons, users may want to specify a location in their posts. For example, they may want to indicate the location of the tournament or the start and endpoints of the marathon. Therefore, as stated in our requirements, users can add address fields to their post types and thus to their posts. Specifying a location on the map in addition to an address description will improve the user experience.

Therefore, mapping functionality is implemented for the **create post** and **view post** features. While creating post users can choose a location from the map by long press and can search a location using the search bar. After choosing the location, the user can give an extra address description as text. To choose multiple locations, post type should include multiple location input areas. While viewing posts, users can view pre-chosen locations from the map and read the address description for detail. Hence mapping functionality is completed for the create post and the view post. We are planning to add this functionality to the user profile too. This feature is now in progress.

The Work that has been Completed Regarding The Mapping Functionality

MOBILE

The Google Maps API is used to display maps and choose a location on the map. First of all, we got the Google Maps API key. Then we configured the project to use the key for the google maps functionality. We updated the "create post" screen to display the map when the location icon is clicked. We used Google Geocoding API to search locations with text input. We get the first lat-long point from the search result and zoom to that location on the map. Then users can choose a specific location by long pressing on the map. The user can close the map and save the chosen point as a latitude-longitude pair by pressing the tick icon. Finally, the user is allowed to type additional address descriptions as text. Users can see the chosen location by re-clicking the location icon.

After a post with a location field is created the user is directed to the post detail page. In this page, a map in which the user selected the coordinates is viewed below the address description. Users can touch and drag to view around. Home page also displays the map view of a post similarly.

WEB

Most of the parts about mapping functionality for web is the same with mobile implementation. In our web application users can also use this mapping feature when updating their profile settings. However, in our web application, while viewing a post or checking out other users profiles a user can see the location fields only as strings of geo codes since Google Maps API has not been implemented for post and profile viewing components yet.

The Work that Remains for The Functionality

MOBILE

As a mobile team, we completed the functionality for the mapping on the creating post and viewing post. Now we are working on adding this functionality to the editing profile and viewing profile. Also, we are planning to improve our mapping functionality by enabling users to select locations from the list that appears while typing on the search bar.

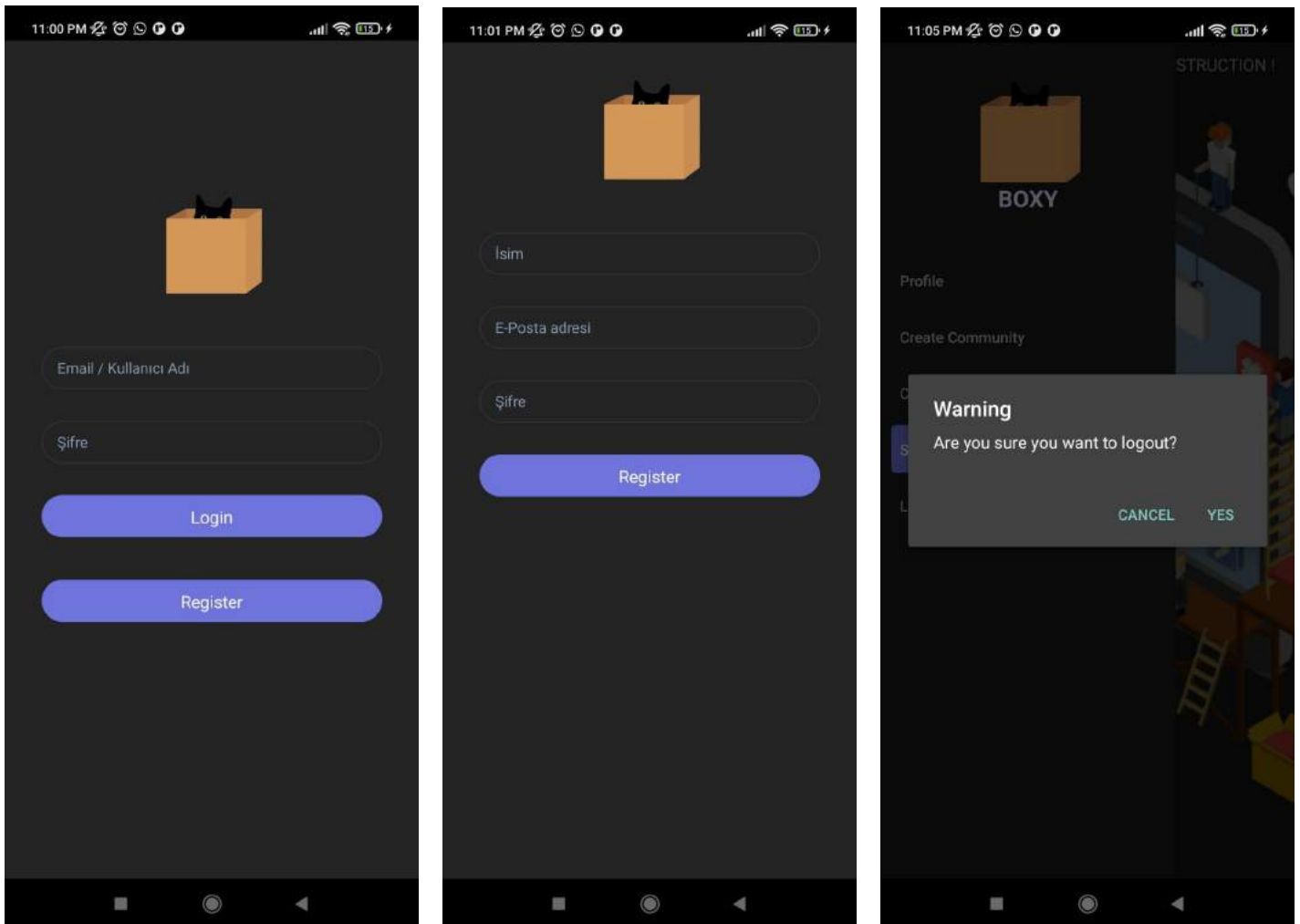
WEB

As explained above, we will be updating post and profile view components so that users can see the actual locations instead of strings of geo codes as locations. Also, we will be adding location search bar for post creation and profile settings pages so that users can find the locations they are looking for faster.

User Interface/User Experience

LOGIN, REGISTER AND LOGOUT

Mobile



WEB

Username
admin

Email
admin@mail.com

Password
.....

Password
.....

☒ Accepting the terms of GDPR and KVKK

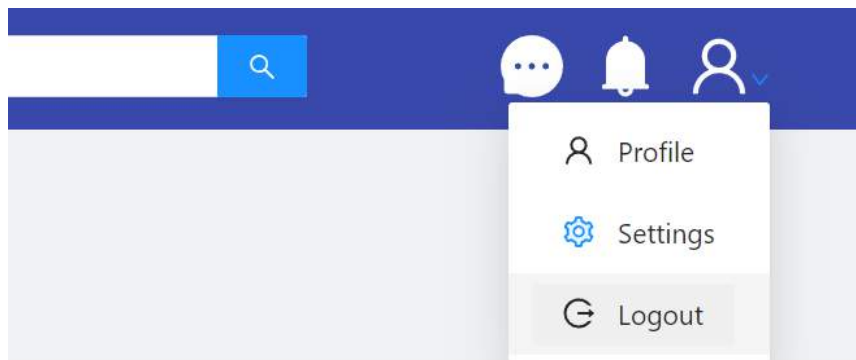
Submit Registration Form

Username or Email
Ahmet

Password
.....

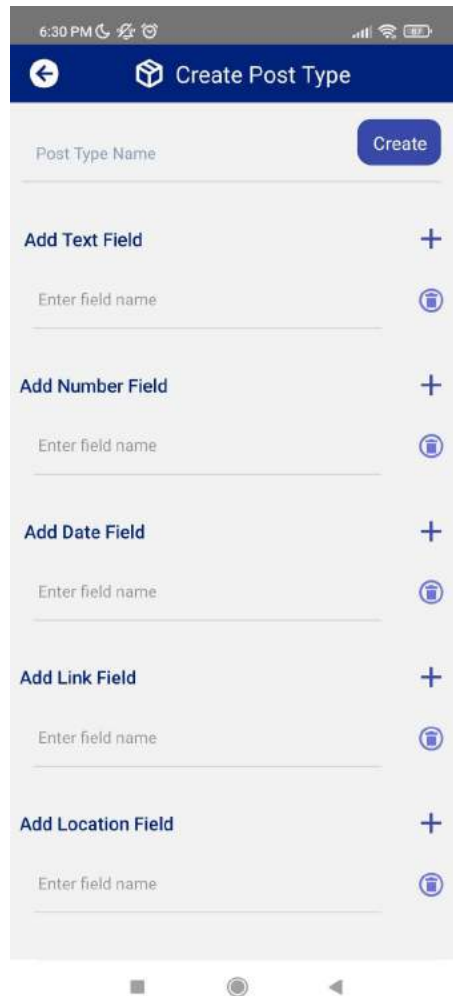
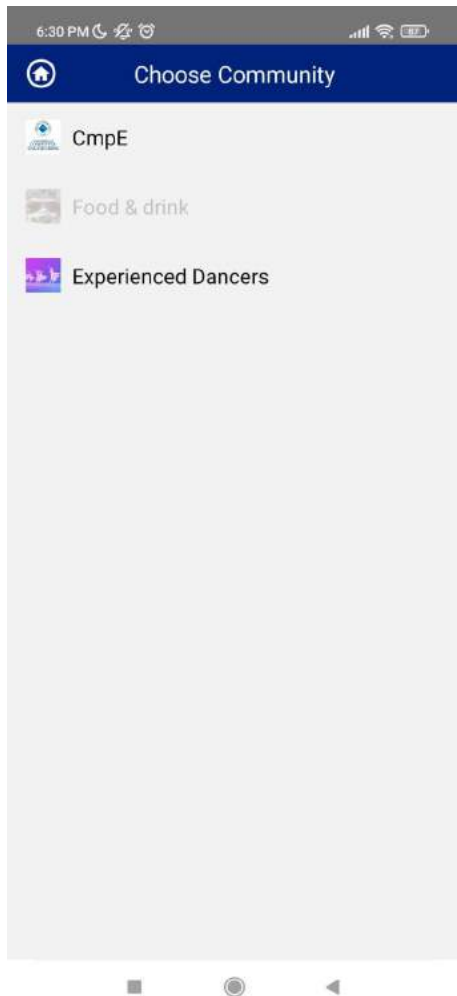
Login

[Forgot your password?](#) [Don't have an account?](#)



CREATE POST TYPE

Mobile



WEB

The image displays two screenshots of the BOXY web application. The top screenshot shows a community page for the 'DEPARTMENT OF COMPUTER ENGINEERING' at 'Boğaziçi University Computer Engineering Student Community'. The page features a sidebar with a 'Community Name' field, a 'Description' field, and a 'Members' section showing '6 people'. Two buttons, 'Create Post Type' and 'Create Post', are visible in the sidebar, with red arrows pointing to them. The main content area displays two posts. The first post is titled 'Event CMPE kayak eventii' and includes fields for 'Date', 'Zoom link', and 'Face to face location'. The second post is titled 'Event Erasmus info session' and includes similar fields. The bottom screenshot shows the 'Create Post Type' form. It has a 'Post Type Name' field and a list of field types: 'text', 'number', 'link', 'date', and 'location'. Each type has a corresponding 'Field name' input field. There is an '+ Add field' button and a 'Create Post Type' button at the bottom.

DEPARTMENT OF COMPUTER ENGINEERING
Boğaziçi University Computer Engineering Student Community

Community Name
CmpE
Description
Boğaziçi University Computer Engineering Student Community
Members
6 people
Create Post Type
Create Post

Event CMPE kayak eventii
Date 2021-12-14T03:41:35.036Z
Zoom link <https://googl.com.tr>
Face to face location 29.21455300336272 40.057351281502896
0 Likes
Comments
Report

Posted by Melis • Posted in CmpE • 12/13/2021 02:06 • Post Type

Event Erasmus info session
Date 2021-12-14T14:00:00.000Z
Zoom link <https://www.zoom.com>
Face to face location 29.05256185680628 41.08305077781619
0 Likes
Comments
Report

Posted by Melis • Posted in CmpE • 12/13/2021 01:19 • Post Type

Topic Community Moderator

BOXY Communities Create Community Create Post Input search text

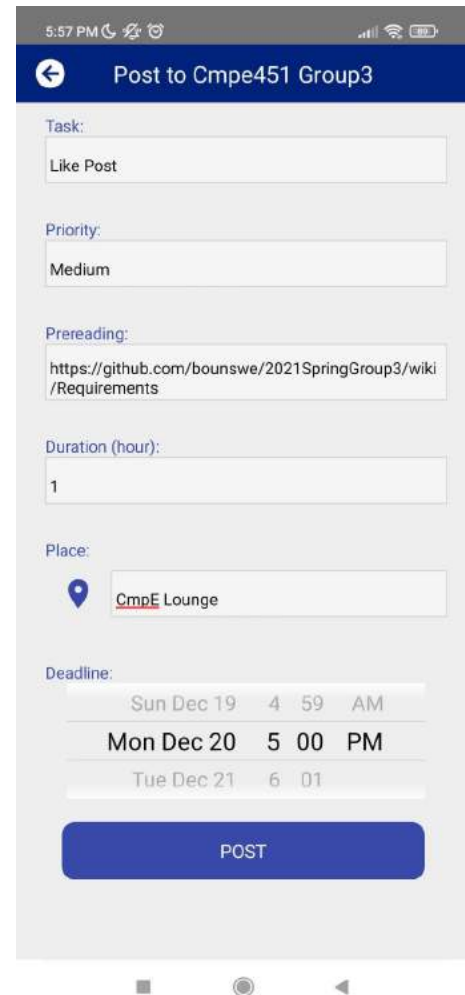
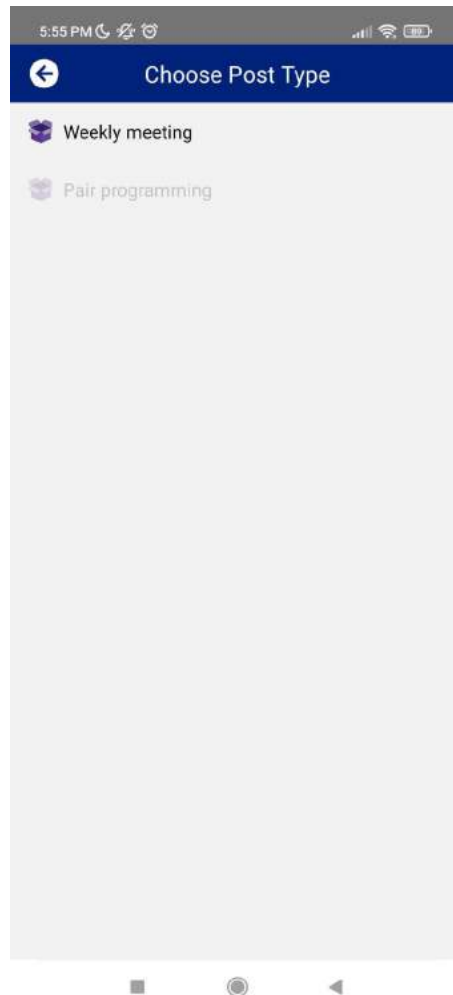
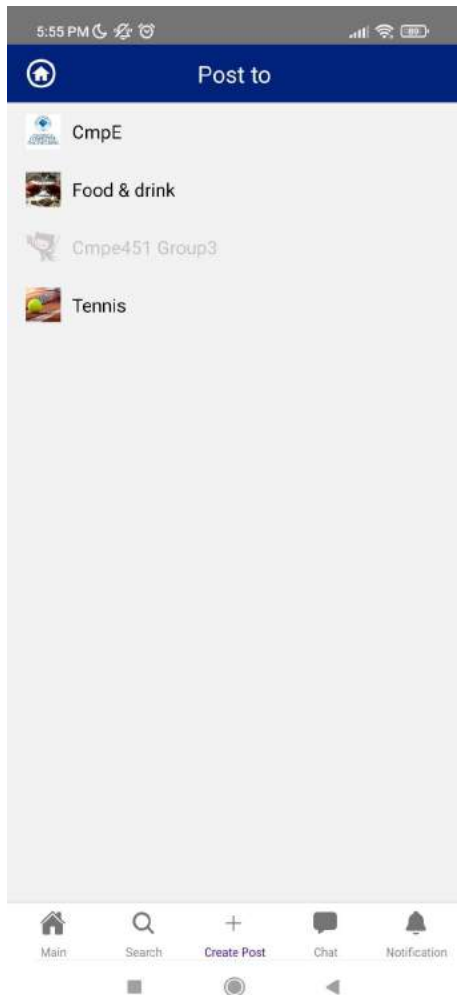
Post Type Name
Post type name

text Field name
number Field name
link Field name
date Field name
location Field name

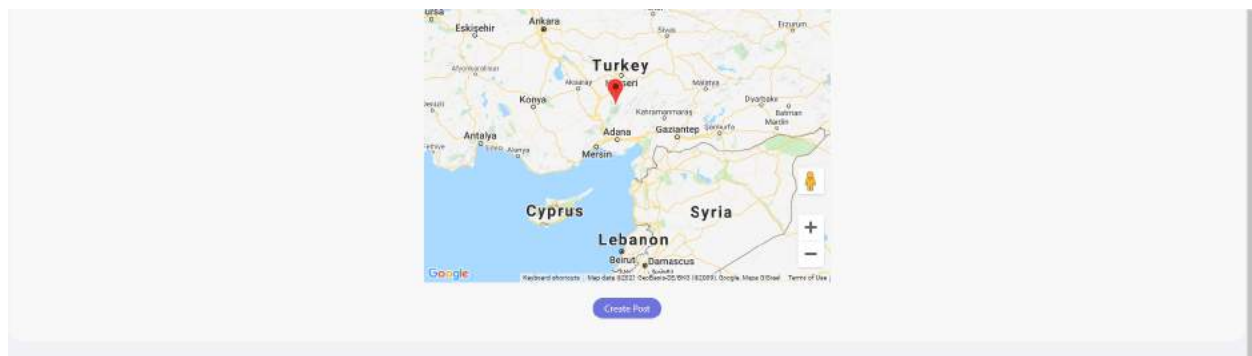
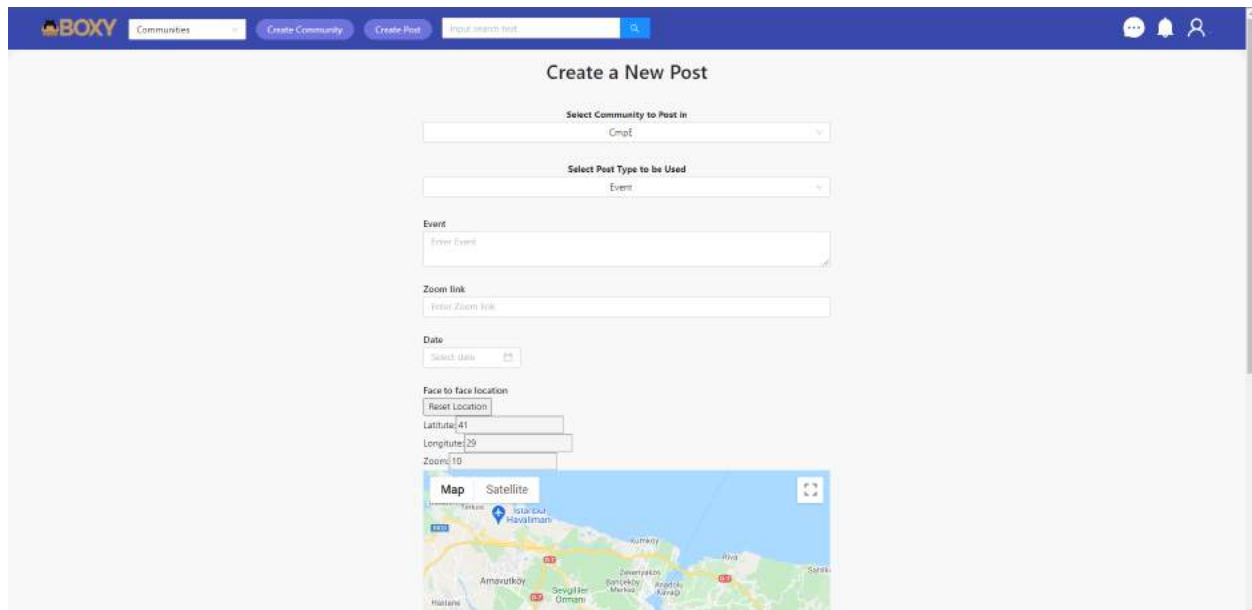
+ Add field
Create Post Type

CREATE POST

Mobile

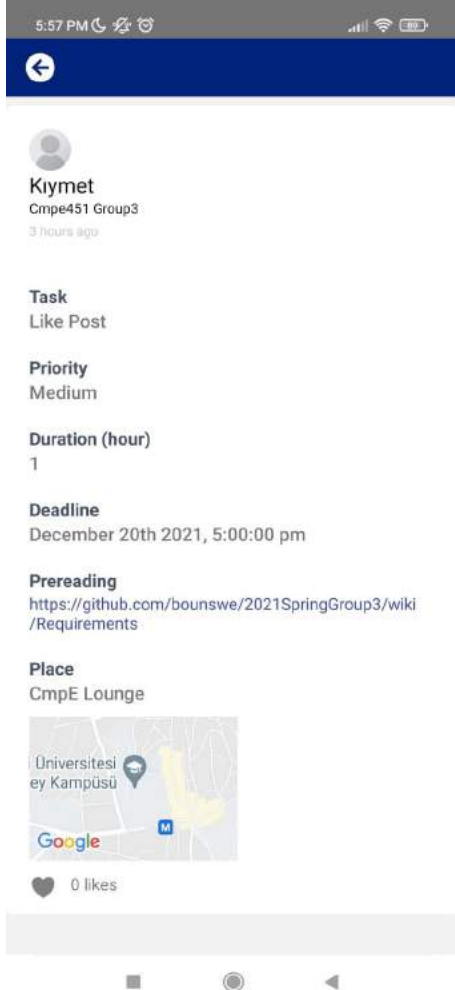


WEB

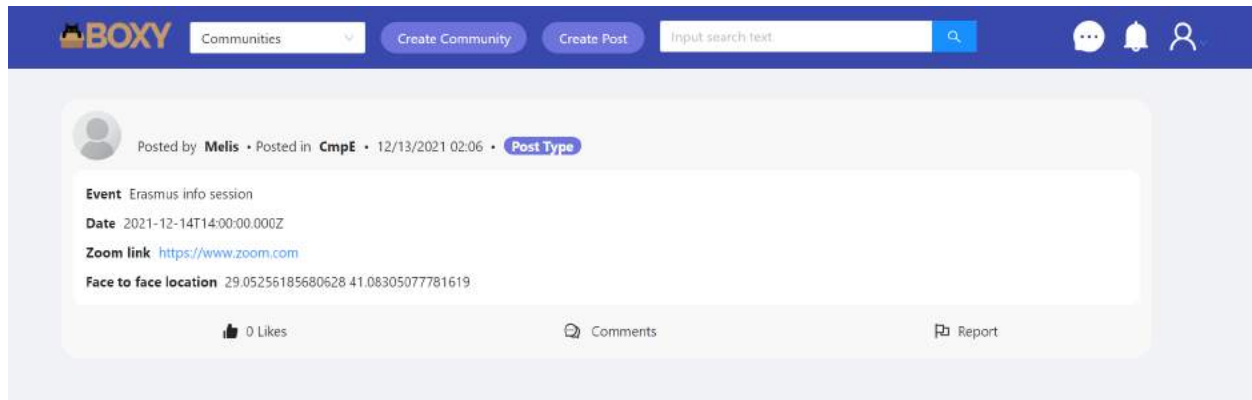


POST DETAIL

Mobile

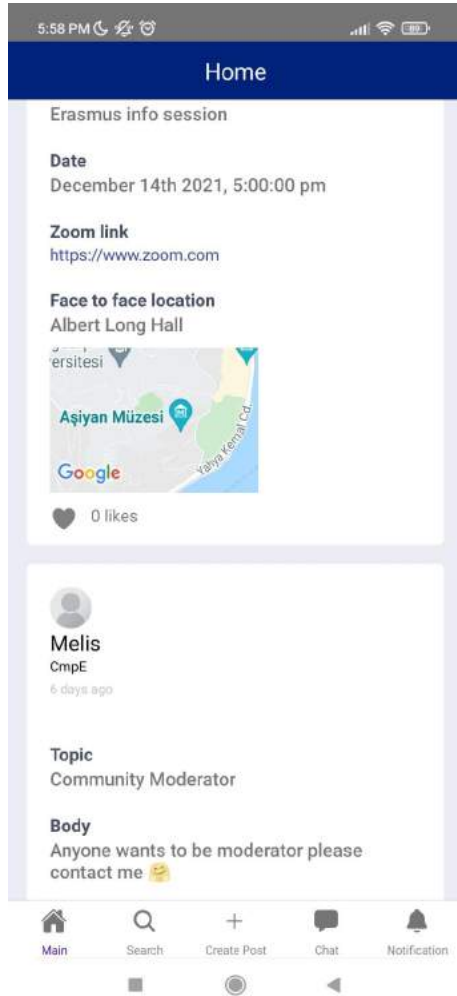


WEB




HOME PAGE

Mobile



WEB (Community Page)


Communities

Create Community

Create Post

Input search text

Follow Community


DEPARTMENT OF
COMPUTER
ENGINEERING

Community Name

CmpE

Description

Bogaziçi University Computer
Engineering Student Community

Members

6 people

Create Post Type

Create Post

Posted by Ahmet • Posted in CmpE • 12/14/2021 03:41 • Post Type

Event CMPE kayak event!
Date 2021-12-14T03:41:35.036Z
Zoom link <https://googl.com.tr>
Face to face location 29.21455300336272 40.057351281502896

0 Likes Comments Report

Posted by Melis • Posted in CmpE • 12/13/2021 02:06 • Post Type

Event Erasmus info session
Date 2021-12-14T14:00:00.000Z
Zoom link <https://www.zoom.com>
Face to face location 29.05256185680628 41.08305077781619

0 Likes Comments Report

Posted by Melis • Posted in CmpE • 12/13/2021 01:19 • Post Type

CREATE COMMUNITY

Mobile

5:58 PM

Create Community

Community Name:

Community Name

Community Photo:

Icon URL

Community Description:

Description

Private Community:
Only members can view posts of the private communities

CREATE

WEB

The screenshot shows the BOXY website's 'Create a New Community' form. The header is dark blue with the BOXY logo, a 'Communities' dropdown, 'Create Community' and 'Create Post' buttons, a search bar, and user icons. The form itself is on a light gray background with the title 'Create a New Community'. It contains three input fields: 'Name' (with placeholder 'Community Name'), 'Topics', and 'Description'. Below these is a 'Community Type' section with 'Public' (selected) and 'Private' radio buttons. A blue 'Submit' button is at the bottom.

BOXY Communities Create Community Create Post Input search text

Create a New Community

Name
Community Name

Topics

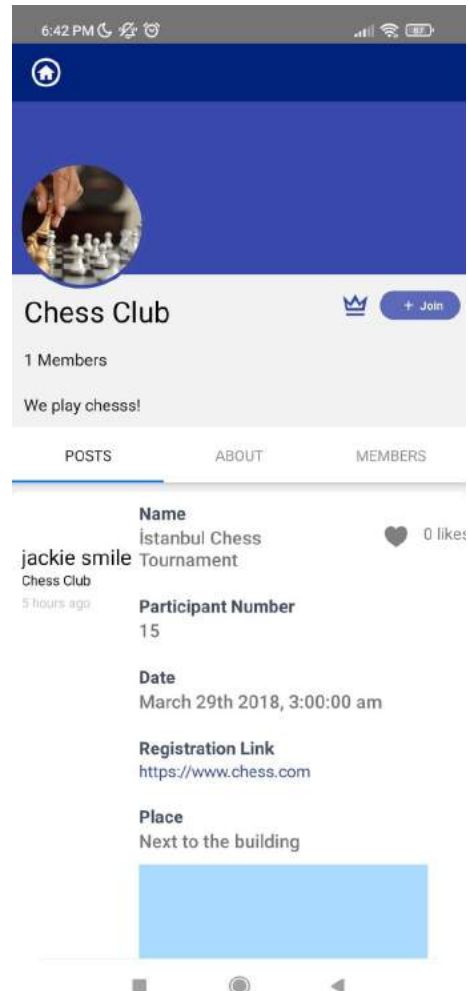
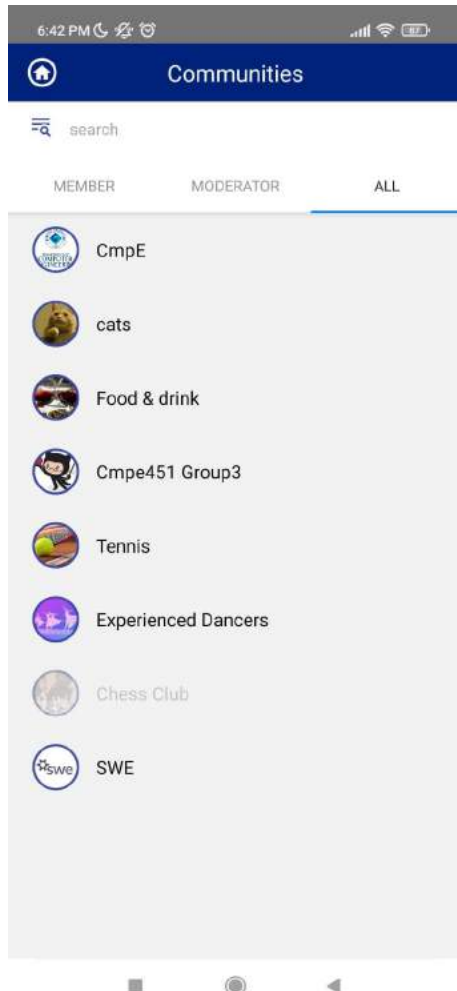
Description

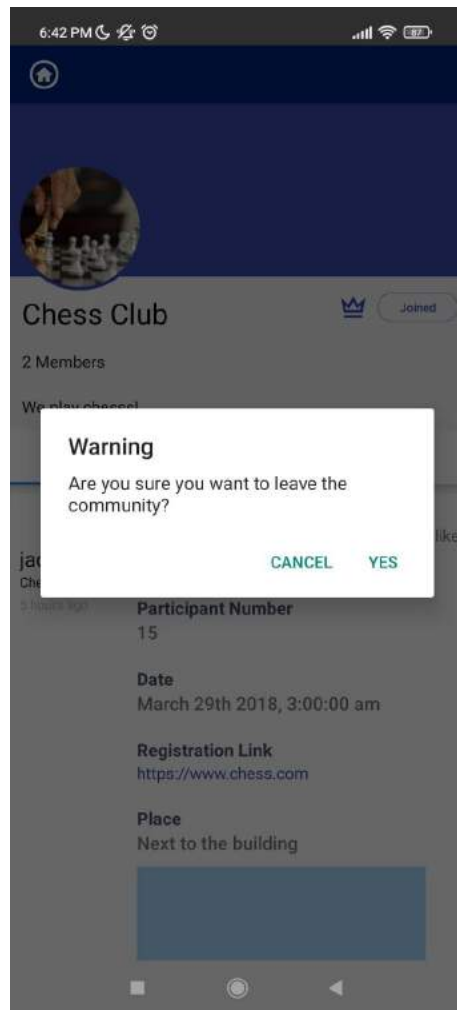
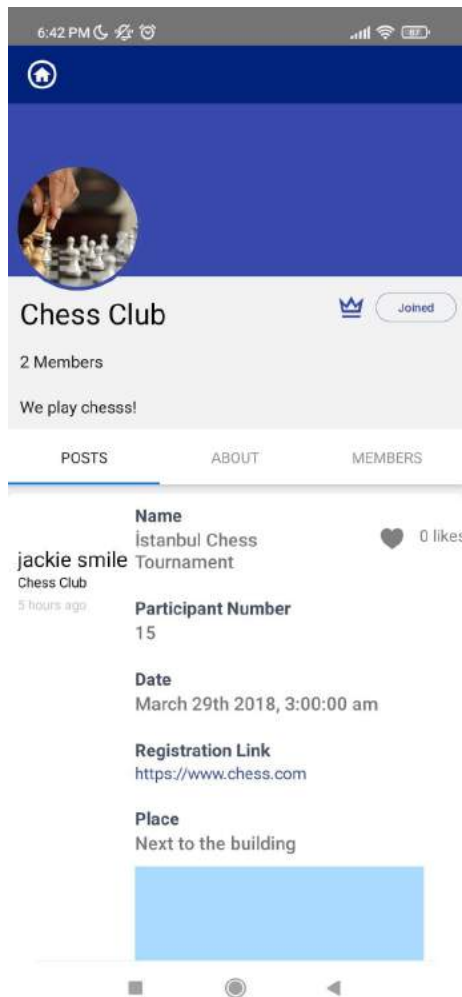
Community Type
☒ Public ☐ Private

Submit

COMMUNITY JOIN, PENDING, LEAVE

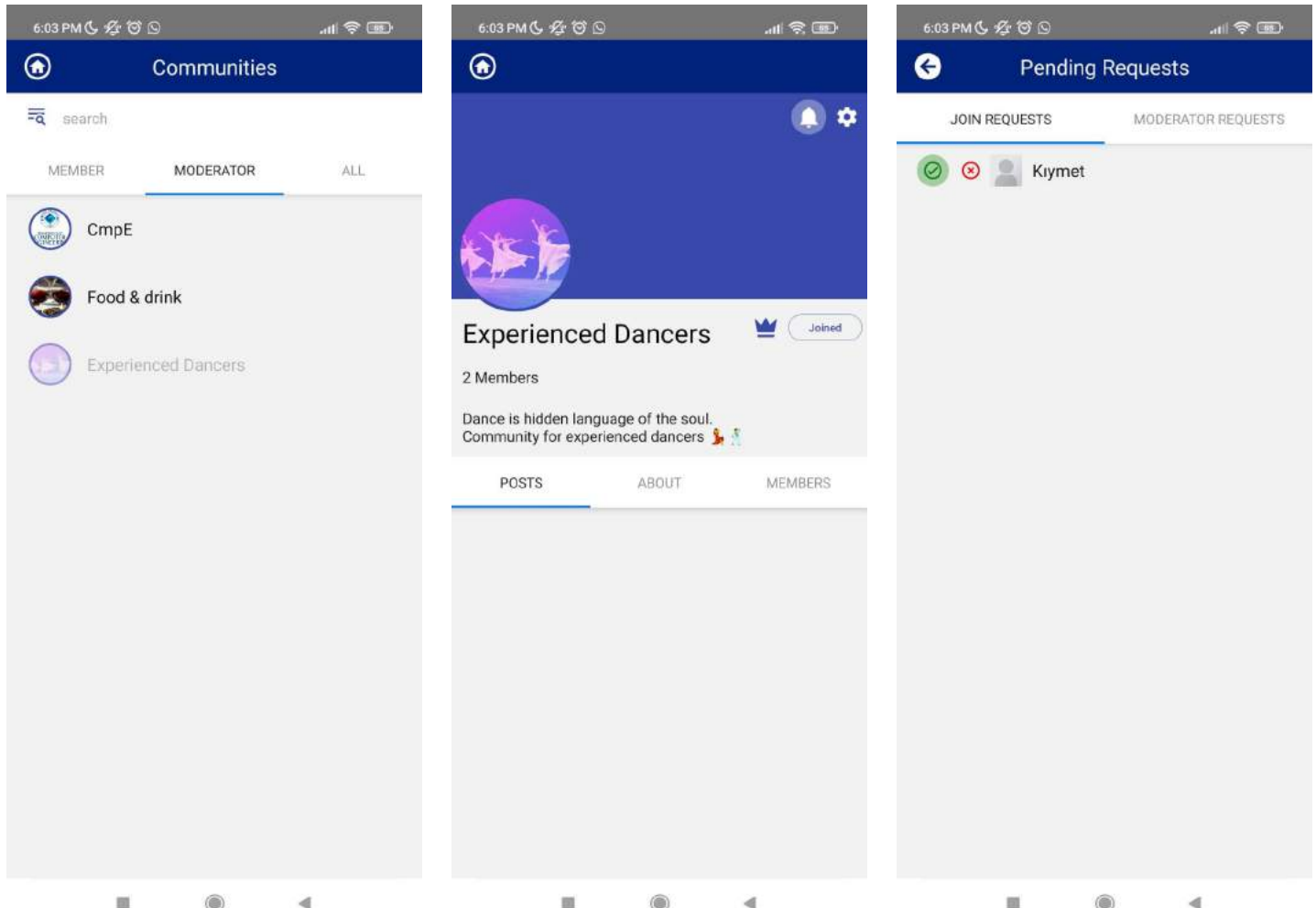
Mobile





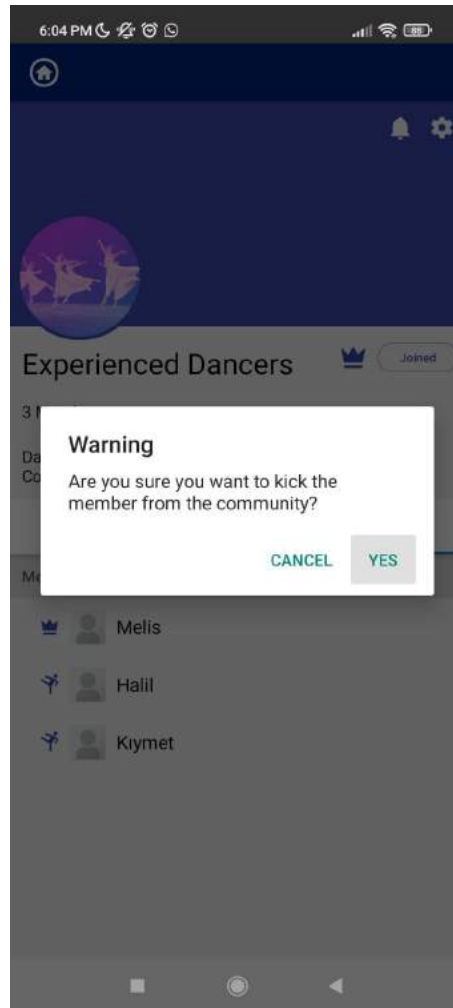
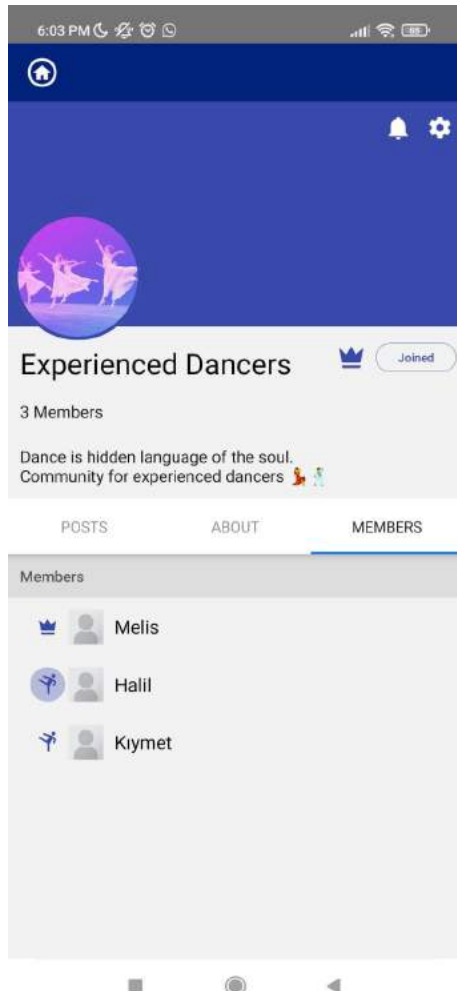
COMMUNITY PENDING JOIN & MODERATOR REQUESTS

Mobile



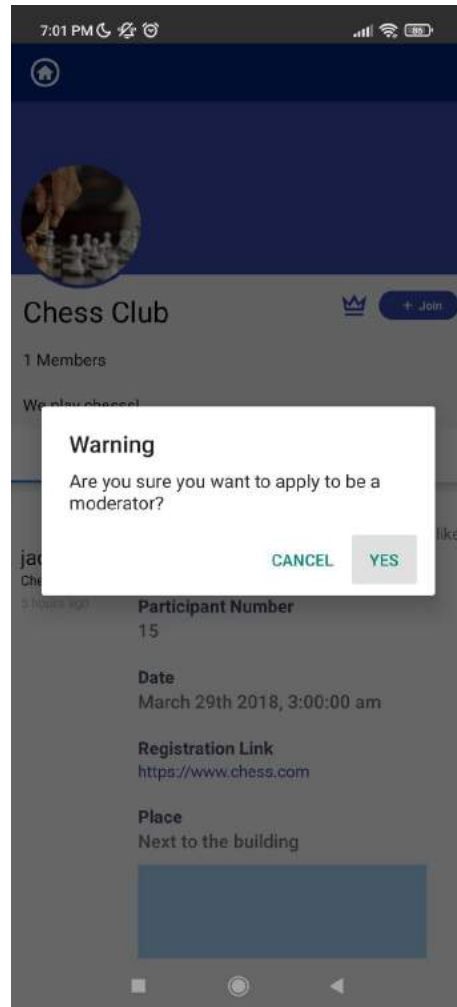
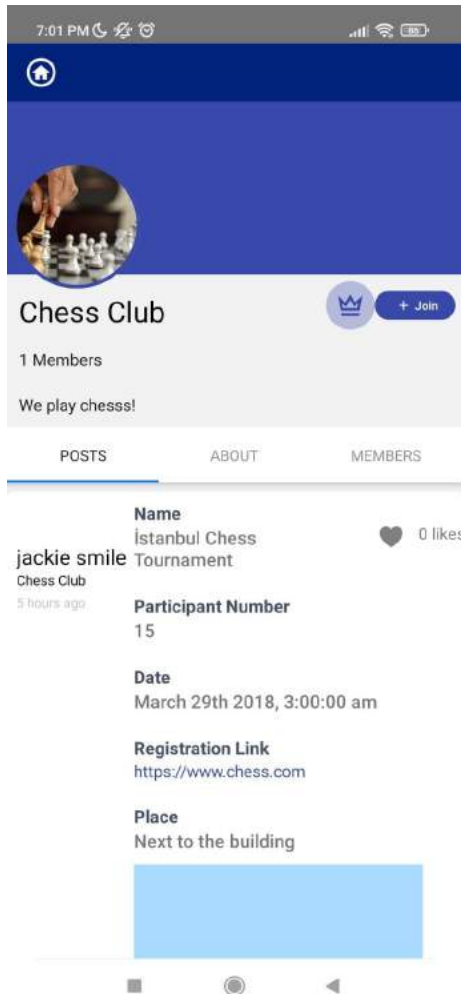
COMMUNITY KICK MEMBER

Mobile



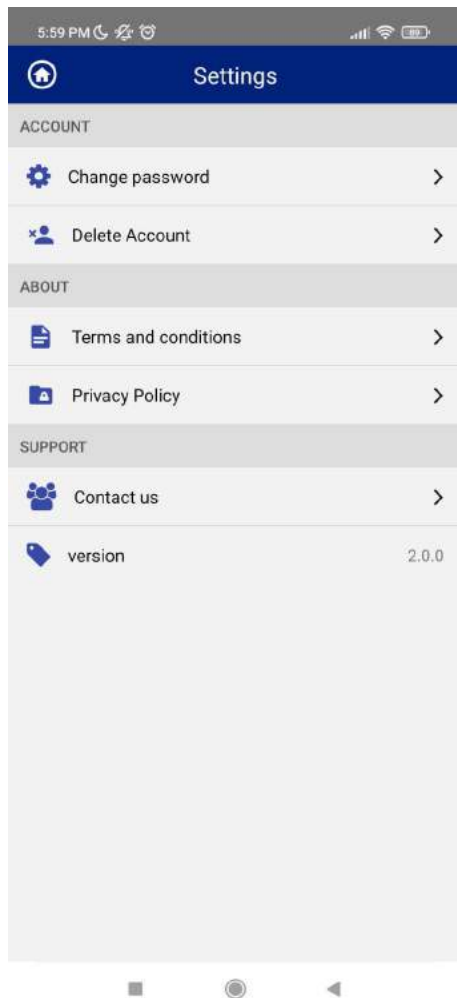
COMMUNITY APPLY TO BE MODERATOR

Mobile

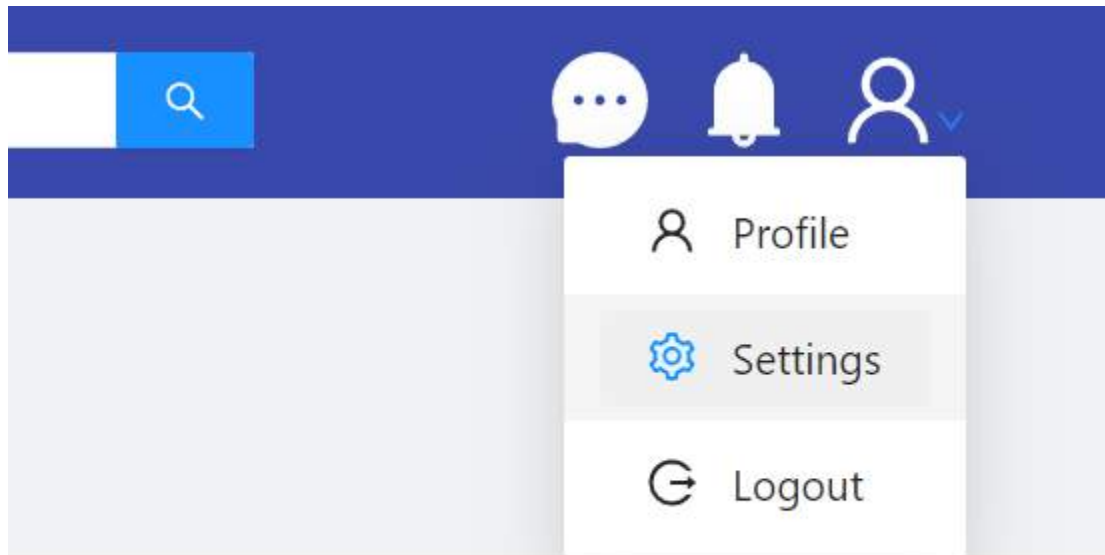


SETTINGS

Mobile

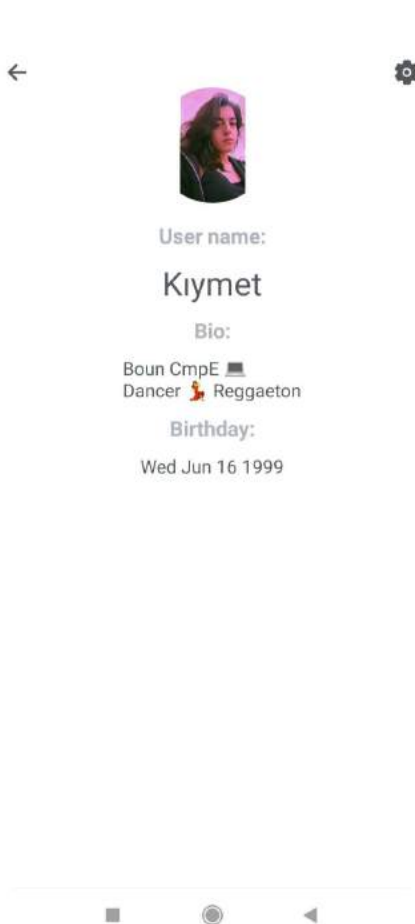


WEB

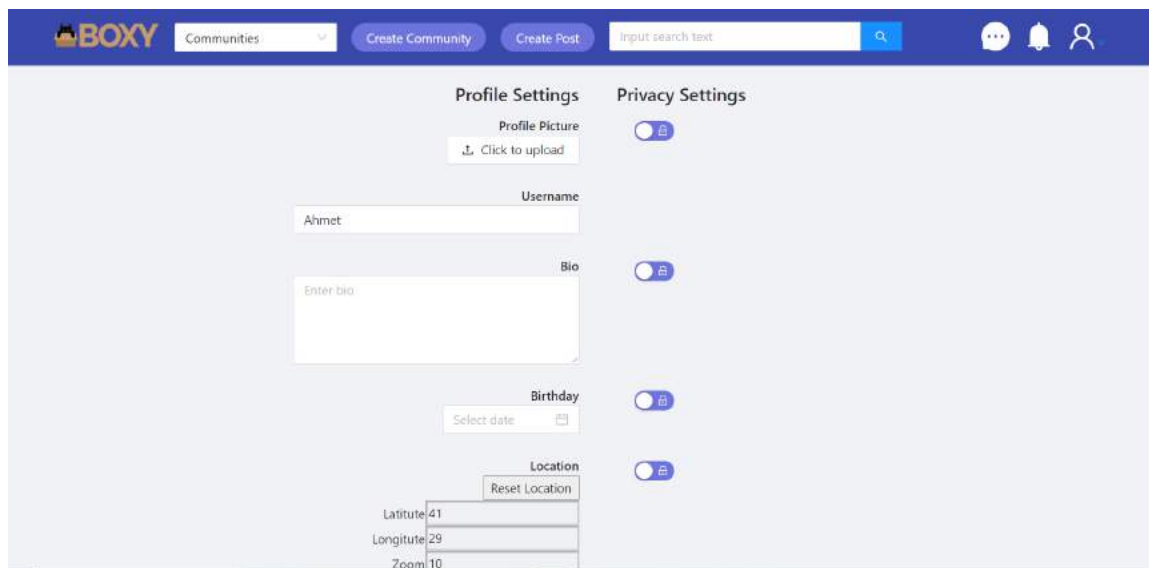
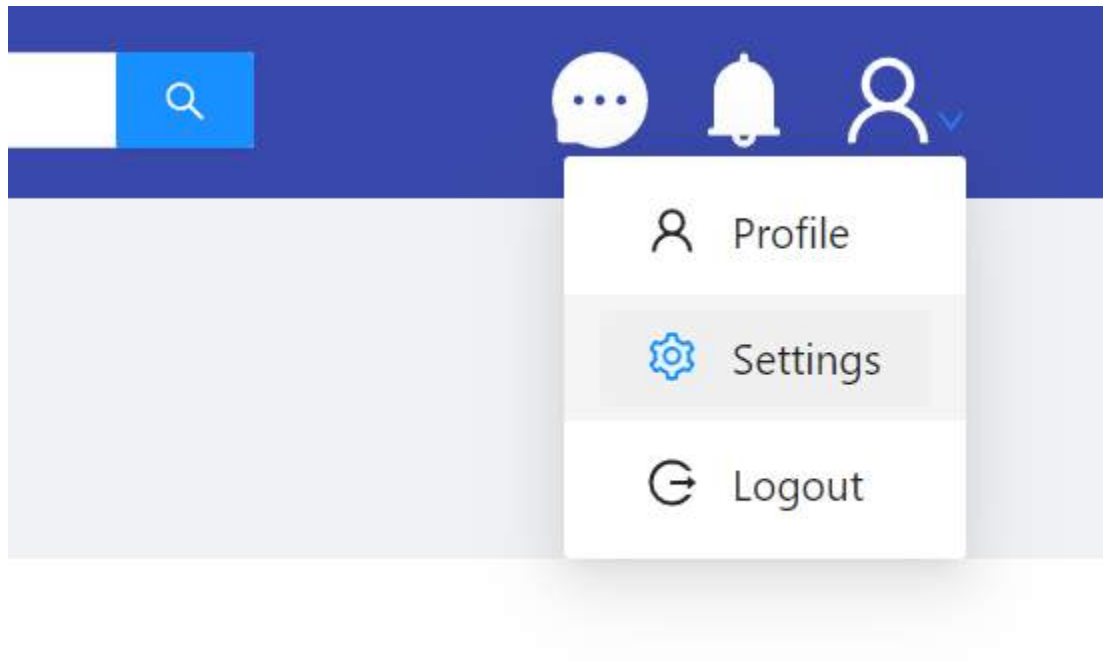


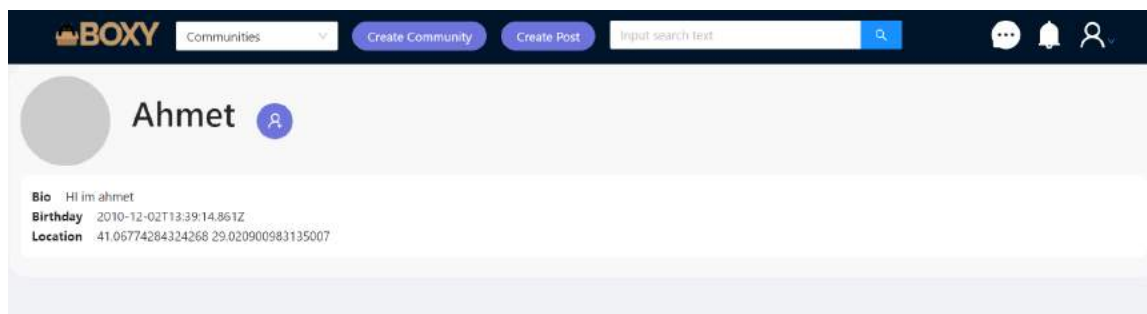
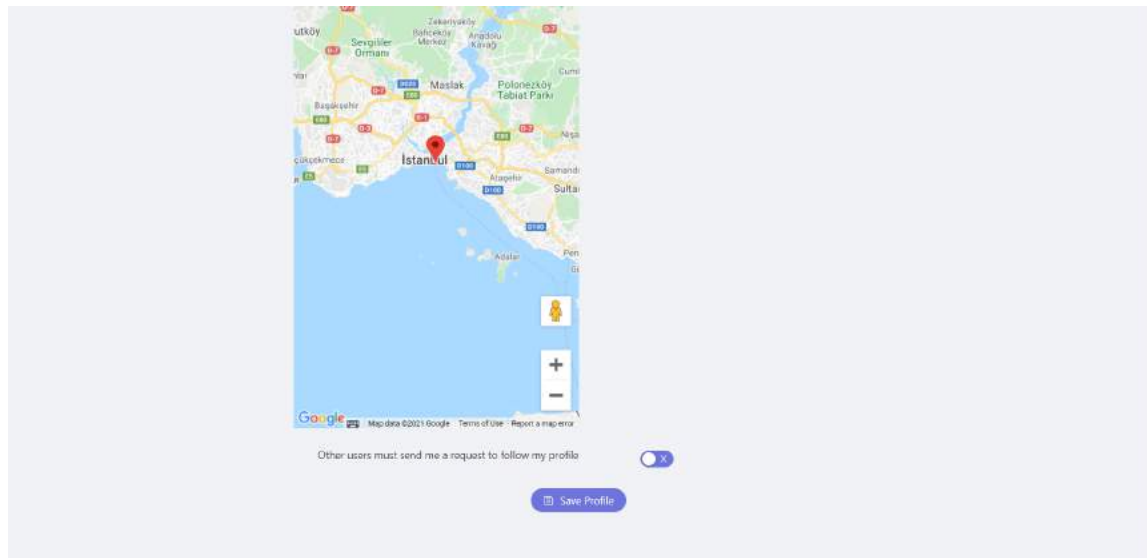
PROFILE / PROFILE SETTINGS

Mobile



WEB





Standards

While completing the requirements, we will focus on the search system which will be mostly based on taggings and post types. Post types make it easier to search in a community for us.

Our other update will be about notifications. The notification system will be implemented using W3C Activity Stream Standard 2.0. Necessary functionalities such as joining a community and accepting requests will be updated according to this Activity Stream Design as explained in the road ahead section.

The implementations of these standards are now in the design phase and will be completed in M3.

Backend Swagger Doc Link : <https://api.cmpegrouphree.store/docs/>

Postman Collection Link:

<https://www.getpostman.com/collections/a790711decec529588a9>

WEB Deployed URL:

https://jcbod.github.io/my_app/

MOBILE APK:

<https://drive.google.com/file/d/1Pk1fwGdHvugM3WFDIplnsl5yZzTYFBnL/view?usp=sharing>
