CMPE 451 – FALL'21

FINAL MILESTONE

Group 10 – PURBEE

TAG-ID

https://github.com/bounswe/2021SpringGroup10/releases/tag/v0.9

CONTRIBUTORS:

Gülşah Çilingiroğlu - Android

Berkay Demirtaş - Backend

Ekin Güngör - Backend

Onur Sefa Özçıbık - Backend

S. Gökberk Yılmaz - Frontend

Safa Burak Altunel - Frontend

Taha Kalkanlı – Frontend

CONTENTS

1. EXECUTIVE SUMMARY 4

STATUS OF DELIVERABLES 4
FİNAL RELEASE NOTES 7
CHANGES AFTER LAST MILESTONE 7
FINAL MILESTONE EVALUATIONS 8

2. SUMMARY OF CODING WORK 8

3. ANSWERS TO SCENARIO 10

4. API DOCUMENTATION 10

A) API DOCUMENTATION 10
B) EXAMPLE CALLS 35

5. USER INTERFACES 42

A. WEB 42
a) Login *42*
b) Signup *43*
c) Home *43*
d) Profile *44*
e) Update profile *44*
f) Create community *45*
g) Simple search *45*
h) Community page *46*
i) Create Post *46*
j) Create Post Type *47*
k) Advanced search *47*
B. ANDROID 48
a) Community Creation *48*
b) Community Section *48*
c) Community Post Types *49*
d) Create Post Type *49*
e) Sign In *50*
f) Sign Up *50*
g) Community Page *53*
h) Create Post *53*
i) Home Page *54*
j) Profile Page *55*
k) Update Profile *55*
l) Search *56*

6. USER MANUAL 57

A) WEB 57
i. Homepage *57*
ii. Create Community Page *57*
iii. Community Page *58*
iv. Create Post Type *58*
v. Create Post *59*
vi. Login page: *59*
vii. Sign up: *60*
viii. Edit Profile: *61*
ix. Search: *62*
x. Advanced search: *63*
xi. Profile Page: *64*
xii. Post Features *65*
B) ANDROID 68

i. TO SIGN IN ... *68*
ii. TO SIGN UP ... *68*
iii. TO SEARCH ... *68*
iv. TO SEE PROFILE PAGE ... *68*
v. TO UPDATE PROFILE ... *68*
vi. TO SEE SUBSCRIBED COMMUNITIES ... *68*
vii. TO CREATE A COMMUNITY ... *69*
viii. TO SEE POST TYPES OF A COMMUNITY ... *69*
ix. TO CREATE A POST TYPE FOR A COMMUNITY ... *69*
x. TO CREATE POST ... *69*

7. SYSTEM MANUAL ... 70

A) WEB ... 70
B) ANDROID ... 70

8. PROJECT PLAN ... 71

9. SOFTWARE REQUIREMENTS SPECIFICATION ... 73

GLOSSARY ... 73
1. FUNCTIONAL REQUIREMENTS ... 74
1.1. USER REQUIREMENTS ... *74*
1.2. SYSTEM REQUIREMENTS ... *76*
2. NON-FUNCTIONAL REQUIREMENTS ... 77

10. SOFTWARE DESIGN DOCUMENTS ... 78

A) CLASS DIAGRAM ... 78
B) USE CASE DIAGRAM ... 79
C) SEQUENCE DIAGRAMS ... 80
i. SEARCH ... *80*
ii. COMMENT ON A POST ... *81*
iii. ADVANCED SEARCH ... *81*
iv. CREATE POST TYPE ... *82*
v. BLOCK A MEMBER FROM COMMUNITY ... *82*
vi. CREATE A POST ACCORDING TO A POST TYPE ... *83*
vii. INTERACT WITH A POST ... *84*
viii. CREATE COMMUNITY ... *84*
D) MOCKUPS ... 85
i. SCENARIO 1 ... *85*
ii. SCENARIO 2 ... *92*
iii. SCENARIO 3 ... *99*
iv. SCENARIO 4 ... *104*

1. EXECUTIVE SUMMARY

STATUS OF DELIVERABLES

SRS(Software requirement specification)	Completed	08.05.2021
Software design documents	Completed	08.05.2021
Project plan	Completed	27.04.2021
Reports	Completed	07.01.2022
User manuel	Completed	12.01.2022
System manuel	Completed	07.01.2022
Manuels for UI designs	Completed	12.01.2022

1.1. User Requirements

1.1.1. Sign Up

1.1.1.1. Users shall be able to start the sign up process by providing an user-name, e-mail and password in the sign up page. [COMPLETED]
1.1.1.2. The user shall provide unique user-name and e-mail in the sign up process. [NON FUNCTIONAL] [COMPLETED]
1.1.1.3. Users shall be able to sign up using their google and facebook accounts.[NOT COMPLETED]

1.1.2. Sign In

1.1.2.1. Registered users shall be able to sign in via username/password or e-mail/password in the sign in page. [COMPLETED]
1.1.2.2. Registered users who signed in shall be able to sign out successfully when log out button clicked by registered user. [NOT COMPLETED]
1.1.2.3. Registered sers shall be able to sign in using their google and facebook accounts.[NOT COMPLETED]

1.1.3. Guest Users

1.1.3.1. Guest users shall be able to search for communities and profiles.[NOT COMPLETED]
1.1.3.2. Guest users shall be able to view the **public content** of communities and profiles. [NOT COMPLETED]

1.1.4. Registered Users

1.1.4.1. Common Features
1.1.4.1.1. Registered users shall inherit the Guest users' requirements specified in requirement 1.1.3. [COMPLETED]
1.1.4.1.2. Registered users' **profile page** shall have profile picture, biography, list of subscribed communities and timeline. [COMPLETED]
1.1.4.1.3. Registered users shall be able to change their profile picture from their **profile page**. [COMPLETED]
1.1.4.1.4. Registered users shall be able to subscribe to public communities from the **community page**. [COMPLETED]
1.1.4.1.5. Registered users shall be able to send request to the community admins for subscribing private communities from the **community page**. [COMPLETED]

1.1.4.1.6. Registered users shall be able to leave communities that they have subscribed before from the **community page**. [COMPLETED]
1.1.4.1.7. Registered users shall be able to report the content they find inappropriate or offensive from the **post**. [NOT COMPLETED]
1.1.4.1.8. Registered users shall be able to directly follow other registered users with public profile from their **profile page**. [COMPLETED]
1.1.4.1.9. Registered users shall be able to send request for following registered users with private profile from their **profile page**. [NOT COMPLETED]
1.1.4.1.10. Registered users shall be able to unfollow a registered user from their **profile page**. [NOT COMPLETED]
1.1.4.1.11. Registered users shall be able to accept or reject following requests from the **notification center**. [NOT COMPLETED]
1.1.4.1.12. Registered users should be able to contribute to content in a Community within the predetermined ranges that **Community Admin** has ruled. (You can check the privacy levels (1,2,3,4) from glossary part.)):
1.1.4.1.12.1. Creating Posts from the **community profile page** via using a post type from **community post types**. (2,4) [COMPLETED]
1.1.4.1.12.2. Interact with **interactive fields** of the post from post. (1,2,4) [COMPLETED]
1.1.4.1.12.3. Commenting posts from post. (1,2,4) [COMPLETED]
1.1.4.1.12.4. Hitting the like button whichin the post. (1,2,4) [COMPLETED]
1.1.4.1.12.5. Replying to comments from the post. (1,2,4) [COMPLETED]
1.1.4.1.12.6. View Community Posts from **timelines** and **main feed**. (1,2,4) [COMPLETED]
1.1.4.1.12.7. View Community's other members from the **community profile page**. (1,2,4) [NOT COMPLETED]
1.1.4.1.12.8. Registered users shall be able to turn on notifications of posts published in the community from the post. (1,2,4) [NOT COMPLETED]
1.1.4.1.13. Registered users shall be able to set privacy option of the Community they create in the community creation process from **community creation page**. [COMPLETED]
1.1.4.1.14. Registered users shall be able to set community profile photo in the community creation process from **community creation page**. [COMPLETED]
1.1.4.1.15. Registered users shall be able to set **manifest** in the community creation process from **community creation page**. [COMPLETED]
1.1.4.1.16. Registered users shall be able to search and filter posts according to the specific post type and its fields in the "main feed".[COMPLETED]
1.1.4.2. Community Admin
1.1.4.2.1. Community admins shall inherit the Registered User's requirements specified in requirement part 1.1.4.1. [COMPLETED]
1.1.4.2.2. Community admins shall be able to set another subscriber of the community as a community admin from the community's subscriber list. [COMPLETED]
1.1.4.2.3. Community admins shall be able to warn Community Members because of hazardous posts from the post. [NOT COMPLETED]
1.1.4.2.4. Community admins shall be able to remove Community Members from the Community from the community's subscriber list. [COMPLETED]
1.1.4.2.5. Community admins shall be able to edit the Community's community picture from the **community profile page**. [COMPLETED]

1.1.4.2.6. Community admins shall be able to edit the Community's community **manifest** from the **community profile page** [COMPLETED].
1.1.4.2.7. Community admins shall be able to create post types using these fields from the **post type creation page**: [COMPLETED]
Plain text field.
Photo field.
Date-time field.
Location field.
Price field.
Participation field.
Document field.
Poll field.

1.2. System Requirements

1.2.1. Recommendation

1.2.1.1. System shall be able to recommend content and persons of interest. [NOT COMPLETED]

1.2.2. Notification

1.2.2.1. Users who turned on their notifications to a post shall be notified with the developments on that post. [NOT COMPLETED]
1.2.2.2. App should provide notifications to the notification center of the mobile device. [NOT COMPLETED]

1.2.3. Authorization

1.2.3.1. The privacy levels of users who accept the invitations from community admins shall be set accordingly. [COMPLETED]
1.2.3.2. The privacy levels of users who removed from the community will be set accordingly. [COMPLETED]
1.2.3.3. The privacy levels of users will be set according to the privacy option of the community. [COMPLETED]

1.2.4. Community

1.2.4.1. Post histories of communities will be stored. [COMPLETED]
1.2.4.2. Community creator is the default **Community Admin**. [COMPLETED]
1.2.4.3. Communities' profile page shall have community picture, manifest, subscribed members list and timeline. [COMPLETED]
1.2.4.4. Communities have default post types named as **default post types.** [NOT COMPLETED]

1.2.5. Searching/Filtering

1.2.5.1. Upon searching related communities, people and posts shall be indexed under the **search bar**. [COMPLETED] ** NOT POSTS

1.2.6. Home Page

1.2.6.1. System shall provide search bar, tabs and community posts. [COMPLETED]
1.2.6.2. System should provide community posts that users are registered and publicly posted contents of followed registered users. [COMPLETED]

1.2.7. Registered User

1.2.7.1. Registered users' **profile page** should have default profile picture and empty biography when a user have signed up. [COMPLETED]

1.2.8. Tagging

1.2.8.1 The semantics tagging shall be supported by wikidata.org [NOT COMPLETED]

2. Non-Functional Requirements

2.1. Security

2.1.1 User passwords shall be encrypted or hashed [NOT COMPLETED]
2.1.2 The system shall use https Protocol [COMPLETED]

2.2. Performance

2.2.1. 2000 users shall be supported [COMPLETED]
2.2.2. 50 requests shall be responded in a second [COMPLETED]
2.2.3. 4 seconds shall be the maximum response time [COMPLETED]

2.3. Availability

2.3.1. The product shall work as Web application in Google Chrome [COMPLETED]
2.3.2. The product shall work as Android application in Android version 10 and later versions. [COMPLETED]
2.3.3. The system should be available 95% of the time. [NOT COMPLETED]
2.3.4. In the case of failure, the system should recover in at most 2 hours. [NOT COMPLETED]

2.4. Standarts

2.4.1. The system shall use the W3C activity stream standard 2.0 to implement subscriptions and notifications [NOT COMPLETED]

2.5. Privacy

2.5.1. User data shall be protected with compliance to KVKK [NOT COMPLETED]
2.5.2. Users shall accept privacy policy and user terms before they start using the product [NOT COMPLETED]

FİNAL RELEASE NOTES

Our final release link:
https://github.com/bounswe/2021SpringGroup10/releases/tag/v0.9

This link contains all the functionalities of our project. In this release, there 3 folders: one for backend, one for frontend and one for mobile. There are some missing requirements on frontend and mobile part. We should be able to provide all the necessary functionalities before final release but unfortunately, we couldn't cover all the requirements in frontend and android part. You can check the requirements completed and not completed from above part of the document.

CHANGES AFTER LAST MILESTONE

In the previous milestone, our project was in a bad state. Our backend, mobile, and frontend was not connected and there were only few pages implemented in mobile part. Backend team decided to document APIs earlier after milestone2 to make mobile and frontend team works in a more efficient way. After few days later, backend team created a endpoint documentation for other teams. This improved the speed of the implementation. We also communicated more to get rid of misunderstandings inside the backend, frontend, and mobile teams. We changed implementation of community and posts in backend part to make implementation of other features easier.

FINAL MILESTONE EVALUATIONS

We have done a big mistake in the final milestone. This mistake was last minute integration. Many functionalities in our frontend crashed due to this last-minute integration. Also, our frontend and android team was not able cover all the requirements. Therefore, it was a bad meeting for us. We had realized how important planning and communication inside a software development team is and how serious consequences can misunderstandings lead. We all hope that CMPE451 will help us in our future career.

2. SUMMARY OF CODING WORK

Safa Burak Altunel	• Login page in android app • Signup page in android app • Community creation page in android app • Basic android app layout(bottom navigation, navigation drawer etc.) • Community post type creation page(not completed) • Connected login signup page of web app with backend. • Profile information details page on web app. • Community home page on web app • Profile page on web app • Search page on web app • Advanced search page on web app
S. Gökberk Yılmaz	• All below tasks are done in frontend: • Implementing post: Like, dislike button, location pop up, liked user pop up • Discussion section • Established backend connection with overcoming CORS • Implementing Feed • Initial implementation of community page, sign in and sign up • Testing endpoints, detecting bugs in apis, giving feedback about them • Attending all labs and meetings • Reviewing Diagrams and requirements
Berkay Demirtaş	• Sign_in endpoint in backed • Sign_up endpoint in backend • Profile_page endpoint in backend • Follow and unfollow endpoints • User_search and community_search endpoints • Advanced_search endpoint • User_feed endpoint • Community_feed endpoint • AWS Gateway arrangement to prevent corse error • Hotfixes before meetings • Database creation • Attending all the meetings and the labs

	• Preparing necessary parts for group reports
Onur Sefa Özçıbık	• AWS API gateway search and initialization • MongoDB initialization for database operations • Community page related endpoints in backend • Comment related endpoints in backend • Discussion related endpoints • Attending to meetings and labs • Taking meeting notes in some of the meetings • Docker image creation and deployment • Preparation of necessary group related documents • Api documentations • Handling CORS error • Transfarring endpoints to API Gateway endpoints
Ekin Güngör	• Implemented all endpoints under api/post/. • Implemented all endpoints under api/post_type/. • Implemented the classes Post, PostType and Fields. • Implemented database functionalities for Post, PostType and Post Fields. • Created database collections for Post and Post Type • AWS API Gateway deployment to adhere to the CORS policy. • Prepared the API Documentation for all functionalities Post, Post Type, and Post Fields. • Prepared the documents for the API part in the group final milestone. • Dockerized our API. • Attending all the meetings.
Taha Kalkanlı	• Login and Signup Pages (Frontend) • Update Profile Info Page (Frontend) • Create Post (Frontend) • Create Post Type (Frontend) • Create Community (Frontend) • Homepage (Frontend) • Sidebar and header for the homepage providing functionalities to navigate, logout, and search. (Frontend)
Gülşah Çilingiroğlu	• Create post (Android) • Post Types Page (Android) • Search (Android) • Update Profile (Android) • Home page layout (Android) • Profile page layout (Android) • Android app navigation • Unit tests for android app • Finalized milestone reports • Created final version of repository on github • Provided User and System Manual of Android

3. ANSWERS TO SCENARIO

– Describe in detail the work that has been completed related to the features necessary to realize this scenario using your application.

First of all the users should be signed in or logged in to the application. We created login and sign in pages and implemented necessary API in the backend. Then, someone should create this bird enthusiasts community. On the sidebar, they can create new communities. Creator of this community should create a post type with 3 text fields(local name, scientific name, youtube link), location field, photo field and datetime field. All these fields are ready in our application. Then, other users should be able to find this community to subscribe. Search bar is great choice for this need. They could search the community name and then subscribe the community on the community page. Subscribed users can also share posts on the community page. They should first select the post type(which is defined by community creator) and then they can share their experience to other users. Users can also follow other bird enthusiast on their profile page. They need to reach profile page of other users. They can search names of the users on the search bar. They also can find the subscribers of the bird enthusiasts community on the community page and follow them.

– Provide the associated referenced to code in GitHub.

Frontend:
https://github.com/bounswe/2021SpringGroup10/tree/master/purbee_frontend/src/components
Backend:
https://github.com/bounswe/2021SpringGroup10/tree/master/purbee_backend/backend_source
Mobile:
https://github.com/bounswe/2021SpringGroup10/tree/master/purbee_mobile/app/src/main

4. API DOCUMENTATION

Link to API : https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev

a) API DOCUMENTATION

Community search:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_search
input:

```
{
  "search_text":"exa"
  }
```

Output:

```
{
  "community_names": [
    "example_id",
```

```
    "example_id2",
    "examplasasdasd_id",
    "example_idd",
    "example_isdd"
  ],
  "response_message": "search result successfully returned"
}
```


User search:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/user_search


```
{
  "search_text":"ber"
}

{
  "response_message": "search result successfully returned",
  "user_names": [
    "berkayyy",
    "berkayyy123",
    "berkSDFAayyy123",
    "berkaydyy123",
    "berkayD",
    "gkberk",
    "berkd",
    "berkddd",
    "berkdddd"
  ]
}
```


User feed:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/user_feed

input:


```
{
  "user_name":"berkdddd"
  }
```


Output:


```
{
  "response_message": "user feed post list successfully returned",
  "user_feed_post_list": [
    "96e56986-100b-413b-bed9-14154a5b8a22",
    "2eb4f558-0bb7-4527-b2dc-312ccbeab368",
    "3f95733b-6290-4578-90d1-15e3a19629aa",
    "2dfdde01-916a-450c-b896-b947168dc2e3",
    "73cc5f44-db4b-44d8-aebb-bc221b4651ab"
  ]
}
```

Community feed:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_feed

input:


```
{
  "user_name":"berkdddd",
  "community_id":"10"
  }
```


Output:


```
{
  "community_post_list": [
    "96e56986-100b-413b-bed9-14154a5b8a22",
    "2eb4f558-0bb7-4527-b2dc-312ccbeab368",
    "3f95733b-6290-4578-90d1-15e3a19629aa",
    "2dfdde01-916a-450c-b896-b947168dc2e3",
    "73cc5f44-db4b-44d8-aebb-bc221b4651ab"
  ],
  "response_message": "community post list successfully returned"
}
```


Follow:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/follow

input:

eking follows berkddd


```
{
  "follower":"eking",
  "following":"berkddd"
}
```


Output:


```
{
  "response_message": "succesfully followed."
}
```


Profile page - put:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/profile_page

input:


```
{
  "user_name":"eking"
}
```


Output:


```
{
  "data": {
    "birth_date": "03.05.1999",
    "first_name": "eekin",
    "followers": [],
```

```
        "following": [
            "berkddd",
            "berkddd",
            "berkddd",
            "berkddd",
            "berkdddd",
            "berkddd"
        ],
        "last_name": "güngör",
        "post_list": [
            17,
            18,
            32,
            39,
            40,
            42,
            44,
            46,
            "61ca3cc849342629a32914cd",
            "61ca3ef5de6df502fe0850b8",
            "61ca408861bdd2bad979be79",
            "36b502de-b8c6-4524-b9bd-7354fa55048f",
            "56e00a61-8454-4575-b9de-59430cc66617",
            "6a9ae4a5-e743-41e9-bc66-707c884f0e06",
            "04ac4399-e922-40c5-b40a-3f2285ab4ffb",
            "96e56986-100b-413b-bed9-14154a5b8a22",
            "2608fa97-9b37-4725-8c1b-eb678ac8c9e4"
        ],
        "profile_photo": [
            0,
            23,
            4,
            5,
            23
        ],
        "user_name": "eking"
    },
    "response_message": "Profile page is successfully returned. "
}
```


Profile page – post:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/profile_page

input:


```
{
    "user_name":"eking",
    "profile_info": {"last_name":"günggör","first_name":"ekin"}
}
```


Output:


```
{
    "response_message": "User page updated successfully."
```

}

Advanced Search:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/advanced_search

input:
**format of the dates and times is important. Below request show all the possible filtering mechanisms. You can use one or multiple of this fields. For example your search dictionary may only contains “PlainText” and “Price” fields.


```
{
  "user_name":"berkdddd",
  "community_id":"10",
  "search_dictionary": {"PlainText": {"search_text":"hiking"},"Price" : {"min_price" : 17, "max_price":18, "currency":"TL"
},"Location" : {"longitude":-
77.0364, "latitude":30, "radius":1000}, "DateTime" : {"starting_date": "10/03/2019" , "ending_date":"15/03/2021", "starting_
time": "12:00" , "ending_time":"13:00"},"Participation" : {"min_participation" : 1, "max_participation" : 3}}
  }
```


output:


```
{
  "post_ids": [
    "2eb4f558-0bb7-4527-b2dc-312ccbeab368"
  ],
  "response_message": "search result successfully returned"
}
```


Community Page

Creation of a community page. Needed input fields are unique community page id which is also the community name, privacy level of the community, and the creator id as a registered user id. This endpoint creates a community object in the backend with the given input. Because we want to enable users to create communities easily, we did not require all the fields inside the community page class. The other class will be empty until the update operation described below.

Example:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page POST


```
{
  "id": "example_isdd",
  "is_private": false,
  "community_creator_id": "example_community_creator"
}
```


Response:


```
{
  "response_message": "Community Page successfully created."
}
```


Updation of a community page. Needed input fields are the id of the community page and the other fields which will be updated inside the community page. All the fields should be given to successfully update the community page.

Example:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page PUT


```
{
  "id": "example_isdd",
  "admin_list": ["first_admin", "second_admin"],
  "subscriber_list": ["first_subscriber", "second_subscriber"],
```

```
  "post_type_id_list": [],
  "post_history_id_list": [],
  "description": "description",
  "photo": "some_url",
  "community_creator_id": "id",
  "created_at": "date",
  "banned_user_list": [],
  "is_private": true
}
```


Response:


```
{
  "community_instance": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "example_isdd",
    "is_private": true,
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber"
    ]
  },
  "response_message": "Community successfully updated"
}
```


Getting the information of the community page. Needed input field is the id of the community page. The input is given via the path parameter. The endpoint returns the all of the fields of the community page.
Example: (community_id=example_isdd)
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/:community_id GET
Response:


```
{
  "community_instance": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "example_isdd",
    "is_private": true,
```

```
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber"
    ]
  },
  "response_message": "Community successfully found"
}
```


Subscription to the community. Needed input fields are the community id and the registered user id. Below example is handled on a public community.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/subscribe PUT


```
{
  "user_id": "OnurSefa",
  "community_id": "first"
}
```


Response:


```
{
  "community": {
    "admin_list": [],
    "banned_user_list": [],
    "community_creator_id": "example_community_creator",
    "created_at": "2022-01-07 18:09:57.555220",
    "description": "",
    "id": "first",
    "is_private": false,
    "photo": null,
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "OnurSefa"
    ]
  },
  "response_message": "Registered user with the id OnurSefa successfully subscribed to Community with the id first"
}
```


Unsubscribe operation handled in a similar way.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/unsubscribe PUT


```
{
  "user_id": "OnurSefa",
  "community_id": "first"
}
```


Response:


```
{
  "community": {
    "admin_list": [],
```

```
    "banned_user_list": [],
    "community_creator_id": "example_community_creator",
    "created_at": "2022-01-07 18:09:57.555220",
    "description": "",
    "id": "first",
    "is_private": false,
    "photo": null,
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": []
  },
  "response_message": "Registered user with the id OnurSefa successfully unsubscribed"
}
```


Changing the privacy level of the community page. The community creator or admins can access this operation. After the operation handled, the privacy changes from private to public or vice versa.

Example:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/change_privacy PUT


```
{
  "community_id": "onursefaCommunity",
  "admin_id": "OnurSefa"
}
```


Response:


```
{
  "community": {
    "admin_list": [],
    "banned_user_list": [],
    "community_creator_id": "OnurSefa",
    "created_at": "2022-01-07 18:16:11.875721",
    "description": "",
    "id": "onursefaCommunity",
    "is_private": true,
    "photo": null,
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": []
  },
  "response_message": "Community privacy set to private"
}
```


Example:


```
{
  "community_id": "onursefaCommunity",
  "admin_id": "OnurSefa"
}
```


Response:


```
{
  "community": {
```

```
    "admin_list": [],
    "banned_user_list": [],
    "community_creator_id": "OnurSefa",
    "created_at": "2022-01-07 18:16:11.875721",
    "description": "",
    "id": "onursefaCommunity",
    "is_private": false,
    "photo": null,
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": []
  },
  "response_message": "Community privacy set to public"
}
```


Handling the subscription requesters in the community page. If the community page is private and a registered user wants to be a subscriber, s/he become requester. The community creator or the community admins can handle this process.

Example: (accept)

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/request PUT


```
{
  "admin_id": "first_admin",
  "community_id": "first",
  "user_id": "OnurSefa",
  "action": "accept"
}
```


Response:


```
{
  "community": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "first",
    "is_private": true,
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber",
      "OnurSefa"
    ]
  },
  "response_message": "Requester successfully accepted"
}
```

Example: (reject)

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/request PUT


```
{
  "admin_id": "first_admin",
  "community_id": "first",
  "user_id": "OnurSefa",
  "action": "reject"
}
```


Response:


```
{
  "community": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "first",
    "is_private": true,
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber"
    ]
  },
  "response_message": "Requester successfully rejected"
}
```


Making or removing admin. The community creator or the community admins have access to add new community admins or remove them. This operation handled as showed in the example.

Example: (make)

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/admin PUT


```
{
  "admin_id": "first_admin",
  "user_id": "OnurSefa",
  "community_id": "first",
  "action": "make"
}
```


Response:


```
{
  "community": {
    "admin_list": [
      "first_admin",
      "second_admin",
      "OnurSefa"
```

```
        ],
        "banned_user_list": [],
        "community_creator_id": "id",
        "created_at": "date",
        "description": "description",
        "id": "first",
        "is_private": true,
        "photo": "some_url",
        "post_history_id_list": [],
        "post_type_id_list": [],
        "requesters": [],
        "subscriber_list": [
            "first_subscriber",
            "second_subscriber"
        ]
    },
    "response_message": "Registered user successfully changed"
}
```


Example: (remove)

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/admin PUT


```
{
    "admin_id": "first_admin",
    "user_id": "OnurSefa",
    "community_id": "first",
    "action": "remove"
}
```


Response:


```
{
    "community": {
        "admin_list": [
            "first_admin",
            "second_admin"
        ],
        "banned_user_list": [],
        "community_creator_id": "id",
        "created_at": "date",
        "description": "description",
        "id": "first",
        "is_private": true,
        "photo": "some_url",
        "post_history_id_list": [],
        "post_type_id_list": [],
        "requesters": [],
        "subscriber_list": [
            "first_subscriber",
            "second_subscriber"
        ]
    },
    "response_message": "Registered user successfully changed"
}
```

Banning a registered user from the community page. A community creator or the community admins can ban a user from their community. If the user was previously subscriber, then they become unsubscriber.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/ban PUT


```
{
  "admin_id": "first_admin",
  "community_id": "first",
  "user_id": "OnurSefa"
}
```


Response:


```
{
  "community": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [
      "OnurSefa"
    ],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "first",
    "is_private": true,
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber"
    ]
  },
  "response_message": "Given user successfully banned"
}
```


Unban operation of the registered user from the community page. It is similar with the ban operation but in opposite direction.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/community_page/unban PUT


```
{
  "admin_id": "first_admin",
  "community_id": "first",
  "user_id": "OnurSefa"
}
```


Response:


```
{
```

```
  "community": {
    "admin_list": [
      "first_admin",
      "second_admin"
    ],
    "banned_user_list": [],
    "community_creator_id": "id",
    "created_at": "date",
    "description": "description",
    "id": "first",
    "is_private": true,
    "photo": "some_url",
    "post_history_id_list": [],
    "post_type_id_list": [],
    "requesters": [],
    "subscriber_list": [
      "first_subscriber",
      "second_subscriber"
    ]
  },
  "response_message": "Given user successfully unbanned"
}
```


Discussion

Creation of a discussion. There is no needed input parameter. This endpoint is not designed for the frontend operations. It is designed for the backend operations because when a post created, a discussion should be created and appended to the post. Nevertheless, we share this information.

Example:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/discussion POST

Response:


```
{
  "discussion": {
    "comment_list": [],
    "id": "84a6396f-3a14-4b60-8576-3dc89098f80f"
  }
}
```


Getting the discussion object. Needed input parameter is id of the discussion and it is given as path parameter.

Example:

https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/discussion/:discussion_id GET

Response:


```
{
  "discussion": {
    "comment_list": [],
    "comment_object_list": [],
    "id": "84a6396f-3a14-4b60-8576-3dc89098f80f"
  },
  "response_message": "Successfully discussion found"
}
```


Comment

Creation of a comment. The needed input parameter is the id of the discussion which will be connected. Also, the text and the registered user id will be given inside the comment.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/comment POST


```
{
  "user_id": "OnurSefa",
  "parent_discussion_id": "84a6396f-3a14-4b60-8576-3dc89098f80f",
  "text": "example text 1"
}
```


Response:


```
{
  "comment": {
    "creator_id": "OnurSefa",
    "discussion_id": "ce3ec85a-b3e2-4981-9436-234f87158794",
    "id": "9c019313-a13c-4500-85dc-822355635a7f",
    "text": "example text 1"
  },
  "response_message": "Successfully a comment created"
}
```


Getting the information of the comment. Needed parameter is the id of the comment and it will given via the path parameter.
Example:
https://cz2qlmf16e.execute-api.us-east-2.amazonaws.com/dev/api/comment/:comment_id GET
Response:


```
{
  "comment": {
    "creator_id": "OnurSefa",
    "discussion_id": "ce3ec85a-b3e2-4981-9436-234f87158794",
    "id": "9c019313-a13c-4500-85dc-822355635a7f",
    "text": "example text 1"
  },
  "response_message": "Successfully comment found"
}
```


Example:
After some comments are made inside the discussion, we get the discussion:


```
{
  "discussion": {
    "comment_list": [
      "9c019313-a13c-4500-85dc-822355635a7f",
      "9308eaef-fac1-4734-96a4-44de30c35d82",
      "124cdf4c-fe9a-4699-81c8-7d646b44592b"
    ],
    "comment_object_list": [
      {
        "creator_id": "OnurSefa",
        "discussion_id": "ce3ec85a-b3e2-4981-9436-234f87158794",
        "id": "9c019313-a13c-4500-85dc-822355635a7f",
        "text": "example text 1"
      },
      {
```

```
            "creator_id": "OnurSefa",
            "discussion_id": "792a93a9-b9b5-43ed-943b-40c05b6f7aad",
            "id": "9308eaef-fac1-4734-96a4-44de30c35d82",
            "text": "example text 1"
        },
        {
            "creator_id": "OnurSefa",
            "discussion_id": "8ca35247-3694-42f9-a98f-1f69a641c7d1",
            "id": "124cdf4c-fe9a-4699-81c8-7d646b44592b",
            "text": "example text 1"
        }
    ],
    "id": "84a6396f-3a14-4b60-8576-3dc89098f80f"
  },
  "response_message": "Successfully discussion found"
}
```


The comment and discussion relation is recursive. When a comment made inside the discussion, the discussion holds the id of the comment. And newly generated comment creates a new discussion which will be commented afterwards. This continues indefinitely. The raltion can be represented via the graph below:

POST_0
|Disucssion_0
||Comment_0
|||Discussion_1
||Comment_5
|||Discussion_6
||||Comment_6
|||||Discussion_7
||||Comment_7
|||||Discussion_7
||Comment_1
|||Discussion_2
||||Comment_3
|||||Discussion_4
||||Comment_4
|||||Discussion_5
||Comment_2
|||Discussion_3

/api/post_type/

POST:

Request JSON body:

"post_type_name" (str): is the name of the post type to be created for the community.

"parent_community_id" (str): is the unique id of the community (can also be a unique name), to which the post type will belong to.

“post_field_info_dictionaries_list” (list of dictionaries): Each dictionary in the list is for one requested field.

In all post_field_dictionaries there are exactly 2 keys: “header” and “field_type”
[“header”] (str): must be unique inside a single post_type, meaning no single post_type can contain 2 fields with the same header. Also notice that all types of fields have a header.
[“field_type”] (str): must be one of the following strings:
“DateTime”, “Location”, “PlainText”, “Price”,”Participation”, “Document”, “Photo”, “Poll”.

Example


```
{
  "post_type_name": "Tennis Event”,
   "parent_community_id": "example_id2",
  "post_field_info_dictionaries_list": [
    {
      "header": "Event Description",
      "field_type": "PlainText"
    },
    {
      "header": "Event Cost",
      "field_type": "Price"
    },
    {
      "header": "Event Location",
      "field_type": "Location"
    },
    {
      "header": "Event Participation",
      "field_type": "Participation"
    }]}
```


API JSON Return:

“response_message” (str): stating what has happened.
“data” (dict): containing the created post_type_id

Example


```
{
```

```
  "data": {
    "post_type_id": "2d97d016-07ac-416e-8dfd-a0d8d0eb3e3c"
  },
  "response_message": "PostType is successfully created."
}
```


PUT:

Request JSON body:

“post_type_id“ (str): the post_type_id of the requested post_type

Example


```
{
  "post_type_id": "2d97d016-07ac-416e-8dfd-a0d8d0eb3e3c"
}
```


API JSON Return:

“data”: contains the same fields as in the post request, but the post_type_id is now in the key “_id”

Example


```
{
  "data": {
    "_id": “2d97d016-07ac-416e-8dfd-a0d8d0eb3e3c",
"post_field_info_dictionaries_list": [
        {
          "field_type": "PlainText",
          "header": "Event Description"
        },
        {
          "field_type": "Price",
          "header": "Event Cost"
        },
        {
          "field_type": "Location",
          "header": "Event Location"
        },
        {
          "field_type": "Participation",
```

```
            "header": "Event Participation"
        }
    ],
    "post_type_name": "Tennis Event"
  },
  "response_message": "PostType is successfully created."
}
```


/api/post/

POST:

Request JSON Body:

“post_type_id" (str): the id of the post_type that this post will be belong to.
“post_owner_user_name” (str): user_name of the user who posts the post.
“”post_entries_dictionary_list” (list of dictionaries): different than the post_field_info_dictionaries_list in the post request of post_type endpoint, in the dictionaries of “post_entries_dictionary_list”, there will not be any specification of the field type (headers are unique, so they define the post type). In the dictionaries there are:

“header”s corresponding to the header of the fields (designed when the utilized post_type is created), which means the headers must match with the previously defined headers of the utilized post_type.

Other than the header, there are also other necessary fields depending on which post field it is. The necessary arguments for all field types are listed as follows:

P.S: ADT stands for (Any Data Type you want)
DateTime: header (str), date (ADT), time (ADT)
Location: header (str), latitude (ADT), longitude (ADT), text (str)
PlainText: header (str), text (str)
Price: header (str), amount (float), currency (str)
Participation: header (str)
Document: header (str), url (str)
Photo: header (str), image (ADT)
Poll: header (str), options (list of strings), can_vote_for_n_many_options (int)

Example


```
{

  "post_type_id": “7f621d31-3fe9-459e-ac46-7aad77ab52a1",
  “post_title”: “Some Title",
  "post_owner_user_name": "eking",
```

```
  "post_entries_dictionary_list": [
    {
      "header": "Event Description",
      "text": "The tournament will host The top 100 ATP players."
    },
    {
      "header": "Event Cost",
      "amount": 17,
      "currency": "tl"
    },
    {
      "header": "Event Location",
      "location": "<some_kind_of_location_data_representation>",
      "text": "Court number 4."
    },
    {
      "header": "Event Participation"
    }
  ]
}
```


API JSON Return:

"data" (dict): including the _id of the created post.

Example


```
{
  "data": {
    "_id": "fa8da556-68b5-4d4b-9ff8-040a07e93a7a"
  },
  "response_message": "Post is successfully created. "
}
```


PUT:

Request JSON Body:

“post_id" (str): id of the post.

Example


```
{
  "post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20"
}
```


API JSON Return:

In the “data” key it returns following information similar to the post version:

“_id”, “post_entries_dictionary_list”, post_liked_user_list" ,”post_owner_user_name”, “post_type_id”

Example


```
{
  "data": {
    "_id": “fa8da556-68b5-4d4b-9ff8-040a07e93a7a",
      “post_title": “Some Title”,
  “post_creation_time”: ""2022-01-03 21:20:04.816271”",
      “post_tag_list”: [],
    "parent_community_id": “example_id2”,
    "post_entries_dictionary_list": [
      {
        "header": "Event Description",
        "text": "The torunament will host The top 100 ATP players."
      },
      {
        "amount": 17,
        "currency": "tl",
        "header": "Event Cost"
      },
      {
        "header": "Event Location",
        "latidude": “<some_kind_of_location_data_representation>",
      “longitude”: “<some_kind_of_location_data_representation>”,
        "text": "Court number 4."
      },
      {
        "header": "Event Participation",
        "list_of_participants": []
```

```
        }
      ],
      "post_liked_user_list": [],
      "post_owner_user_name": "eking",
      "post_type_id": "7f621d31-3fe9-459e-ac46-7aad77ab52a1"
    },
    "response_message": "Post is successfully returned. "
}
```


/api/post/like/

PUT

Request JSON Body:

"post_id" (str): id of the post to be liked

"user_name" (str): user_name of the user who likes the post.

Example


```
{
    "post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
    "user_name": "eking2"
}
```


API Return Json Body:

["post_liked_user_list"] (list of strings): user_name of the users that liked that post, after executing last like request.

Example


```
{
    "data": {
      "post_liked_user_list": [
        "eking2"
      ]
    },
    "response_message": "Post is successfully liked. "
}
```


/api/post/unlike/

PUT

Request JSON Body:

"post_id" (str): id of the post to be unliked

"user_name" (str): user_name of the user who unlikes the post.

Example

```
{
   "post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
   "user_name": "eking2"
}
```

API Return Json Body:

["post_liked_user_list"] (list of strings): user_name of the users that liked that post, after executing last like request.

Example:

```
{
   "data": {
      "post_liked_user_list": [ ]
   },
   "response_message": "Post is successfully unliked. "
}
```

/api/post/participate/

PUT

Request JSON Body:

"post_id" (str): id of the post where participation field belongs to

"user_name" (str): user_name of the user who will participate

"header_of_participation_field" (str): header of the participation field (there could be more than one participation fields of a post, since headers are unique inside a post_type this locates which participation field it is)

Example:

```
{
   "post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
```

"user_name": “eking2”,
"header_of_participation_field": “Event Participation”
}

API Return Json Body:

[“list_of_participants”] (list of strings): user_name of the users that will participate to that specific Participation field of the post, after executing last participate request

Example:

{
"data": {
"list_of_participants": [eking2]
},
"response_message": “ User with user_name “eking2” has successfully marked as participating to the Participation field with header “Event Participation” of the post with id “a7bae279-5ffa-4686-8d2a-e20406297e20”.}

/api/post/cancel_participation/

PUT

Request JSON Body:

“post_id" (str): id of the post where participation field belongs to
“user_name” (str): user_name of the user who will participate
“header_of_participation_field” (str): header of the participation field (there could be more than one participation fields of a post, since headers are unique inside a post_type this locates which participation field it is)

Example:

{
"post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
"user_name": “eking2”,
"header_of_participation_field": “Event Participation”
}

API Return Json Body:

[“list_of_participants”] (list of strings): user_name of the users that will participate to that specific Participation field of the post, after executing last cancel participation request

Example:

{
"data": {
"list_of_participants": []
},
"response_message": " User with user_name "eking2" has successfully unmarked as participating to the Participation field with header "Event Participation" of the post with id "a7bae279-5ffa-4686-8d2a-e20406297e20".}

/api/post/vote/

PUT

Request JSON Body:
"post_id" (str): id of the post where Poll field belongs to
"user_name" (str): user_name of the user who will vote
"header_of_poll_field" (str): header of the Poll field (there could be more than one Poll fields of a post, since headers are unique inside a post_type, this locates which Poll field it is)
"option" (str): the option to vote for.

Example:

{
"post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
"user_name": "eking2",
"header_of_poll_field": "Event Poll",
"option" : "This seemed a little bit wrong to me. It shouldn't stay as it is."
}

API Return Json Body:

["options"] (dictionary): where the keys in this dictionary are single options of type string, and the corresponding values of these strings are list of strings, that holds the user_name of the users that vote for that string option.

Example:

{
"data": {
"options":{
"This seemed a little bit wrong to me. It shouldn't stay as it is.": ["eking2", "Mert"],
"I think it's completely fine. It can stay like this.": ["Safa", "Onur"]}

},
"response_message": " User with user_name "eking2" has successfully voted in the Participation field with header "Event Poll" of the post with id "a7bae279-5ffa-4686-8d2a-e20406297e20", for the option "This seemed a little bit wrong to me. It shouldn't stay as it is.".
}

/api/post/cancel_vote/

PUT

Request JSON Body:

"post_id" (str): id of the post where Poll field belongs to
"user_name" (str): user_name of the user who will cancel vote
"header_of_poll_field" (str): header of the Poll field (there could be more than one Poll fields of a post, since headers are unique inside a post_type, this locates which Poll field it is)
"option" (str): the option to cancel vote for.

Example:

{
"post_id" : "a7bae279-5ffa-4686-8d2a-e20406297e20",
"user_name": "eking2",
"header_of_poll_field": "Event Poll",
"option" : "This seemed a little bit wrong to me. It shouldn't stay as it is."
}

API Return Json Body:

["options"] (dictionary): where the keys in this dictionary are single options of type string, and the corresponding values of these strings are list of strings, that holds the user_name of the users that vote for that string option.

Example:

{
"data": {
"options":{
"This seemed a little bit wrong to me. It shouldn't stay as it is.": ["Mert"],

“I think it’s completely fine. It can stay like this.”: [“Safa”, “Onur”]}

},

"response_message": “ User with user_name “eking2” has successfully cancelled their vote in the Participation field with header “Event Poll” of the post with id “a7bae279-5ffa-4686-8d2a-e20406297e20”, for the option “This seemed a little bit wrong to me. It shouldn’t stay as it is.”.}

b) EXAMPLE CALLS

In the shared postman folder, core functionalities are the ones starting with [1], [2] and [3]. [-2], [-1] and [0] are the creation of users and communities, necessary for the following requests. The ones starting with [optional] are mainly get requests, to show that POST requests work properly.


```
{
            "name": "[-2] create example_user2",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"user_name\": \"example_user2\", \n
\"mail_address\": \"example_user2@gmail.com\",\n    \"password\":
\"AAaa..1234578\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/sign_up/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "sign_up",
                        ""
                    ]
                }
```

```
            },
            "response": []
        },
        {
            "name": "[-1] create example_user1",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"user_name\": \"example_user1\", \n
\"mail_address\": \"example_user2@gmail.com\",\n    \"password\":
\"AAaa..1234578\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/sign_up/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "sign_up",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[0] user1 creates community",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\r\n    \"id\": \"onursefaCommunity\",\r\n
\"is_private\": false,\r\n    \"community_creator_id\": \"OnurSefa\"\r\n}",
                    "options": {
                        "raw": {
```

```
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/community_page",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "community_page"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[optional] get created community",
            "protocolProfileBehavior": {
                "disableBodyPruning": true
            },
            "request": {
                "method": "GET",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "",
                    "options": {
                        "raw": {
                            "language": "html"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/community_page/:community_id",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
```

```
                    ],
                    "path": [
                        "dev",
                        "api",
                        "community_page",
                        ":community_id"
                    ],
                    "variable": [
                        {
                            "key": "community_id",
                            "value": "example_isdd"
                        }
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[1] user2 subscribes to community",
            "request": {
                "method": "PUT",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\r\n    \"user_id\": \"ex_user\",\r\n
\"community_id\": \"first\"\r\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/community_page/subscribe",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "community_page",
                        "subscribe"
                    ]
                }
            },
```

```
            "response": []
        },
        {
            "name": "[2] community create post type",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"post_type_name\": \"Birds Post Type\",\n
\"parent_community_id\": \"Bird Enthusiasts\",\n
\"post_field_info_dictionaries_list\": [\n        {\n            \"header\":
\"Local Name of Bird\",\n            \"field_type\": \"PlainText\"\n        },\n
{\n            \"header\": \"Scientific Name\",\n            \"field_type\":
\"PlainText\"\n        },\n        {\n            \"header\": \"Photo of the
Bird\",\n            \"field_type\": \"Photo\"\n        },\n        {\n
\"header\": \"Location of the Photo\",\n            \"field_type\": \"Location\"\n
},\n        {\n            \"header\": \"Audio/Video of the Bird Singing\",\n
\"field_type\": \"Document\"\n        } \n    ]\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/post_type/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "post_type",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[optional] get created post type",
            "request": {
                "method": "PUT",
                "header": [],
```

```
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"post_type_id\": \"4d63ee89-b651-4706-b2b1-
0351b9b78ae9\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/post_type/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "post_type",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[3] user2 creates post",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"post_type_id\": \"4d63ee89-b651-4706-b2b1-
0351b9b78ae9\",\n    \"post_title\": \"The beautiful bird from my Los Angels
trip.\",\n    \"post_owner_user_name\": \"example_user\",\n
\"post_entries_dictionary_list\": [\n        {\n            \"header\": \"Local
Name of Bird\",\n            \"text\": \"Saka\"\n        },\n        {\n
\"header\": \"Scientific Name\",\n            \"text\": \"Carduelis carduelis\"\n
},\n        {\n            \"header\": \"Photo of the Bird\",\n
\"image\":
\"https://i2.milimaj.com/i/milliyet/75/0x410/61749ffc86b24704903f6132.jpg\"\n
},\n        {\n            \"header\": \"Location of the Photo\",\n
\"latitude\": \"34.0522 N\",\n            \"longitude\": \"118.2437 W\",\n
\"text\": \"Near the beautiful Mount San Antonio\"\n        },\n        {\n
```

```
\"header\": \"Audio/Video of the Bird Singing\",\n            \"url\":
\"https://www.youtube.com/watch?v=bCLA9MZ7tO4\"\n        } \n    ]\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/post_type/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
                        "us-east-2",
                        "amazonaws",
                        "com"
                    ],
                    "path": [
                        "dev",
                        "api",
                        "post_type",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "[optional] get post",
            "request": {
                "method": "PUT",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"post_id\" : \"7f4f2b46-721d-4293-be96-
5c2ffcfb2d24\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "https://cz2qlmf16e.execute-api.us-east-
2.amazonaws.com/dev/api/post/",
                    "protocol": "https",
                    "host": [
                        "cz2qlmf16e",
                        "execute-api",
```

```
                    "us-east-2",
                    "amazonaws",
                    "com"
                ],
                "path": [
                    "dev",
                    "api",
                    "post",
                    ""
                ]
            }
        },
        "response": []
    }
```


5. USER INTERFACES

URI of the Code

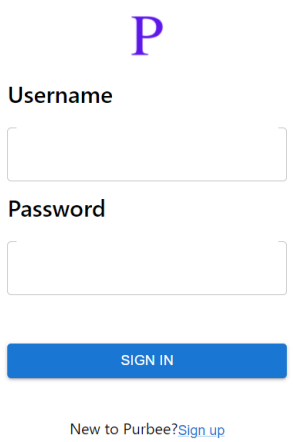
Frontend:
https://github.com/bounswe/2021SpringGroup10/tree/master/purbee_frontend/src/components

Mobile:
https://github.com/bounswe/2021SpringGroup10/tree/master/purbee_mobile/app/src/main

A. WEB

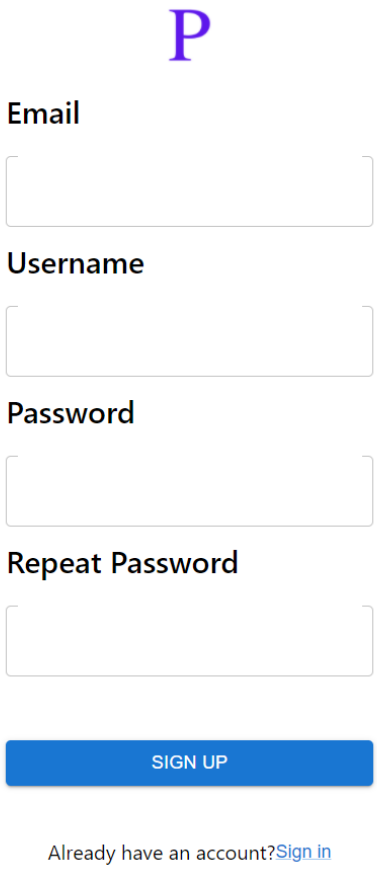
a) Login

Root directory



b) Signup

Root directory

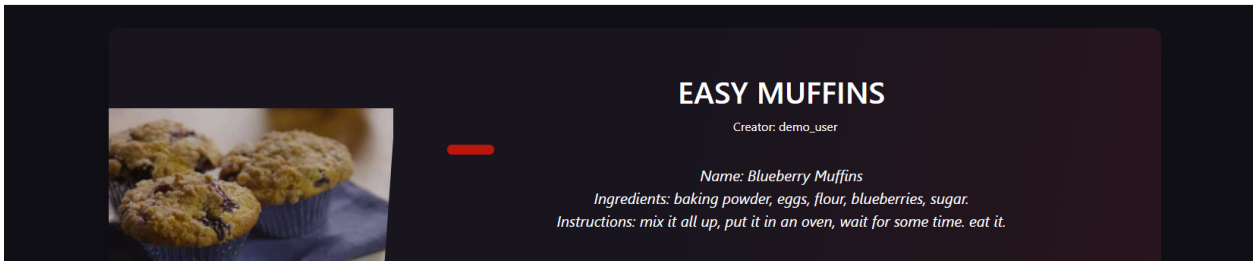


c) Home

/home

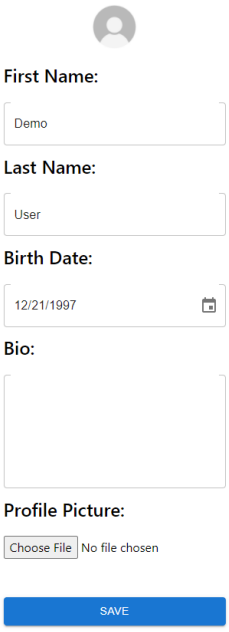
d) Profile

/profile-page/<user_id>



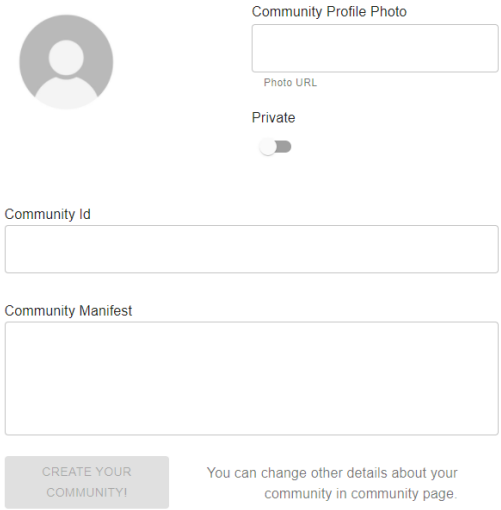
e) Update profile

/profile-info



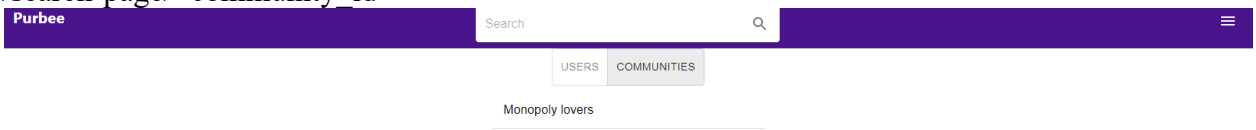
f) Create community

/create-community



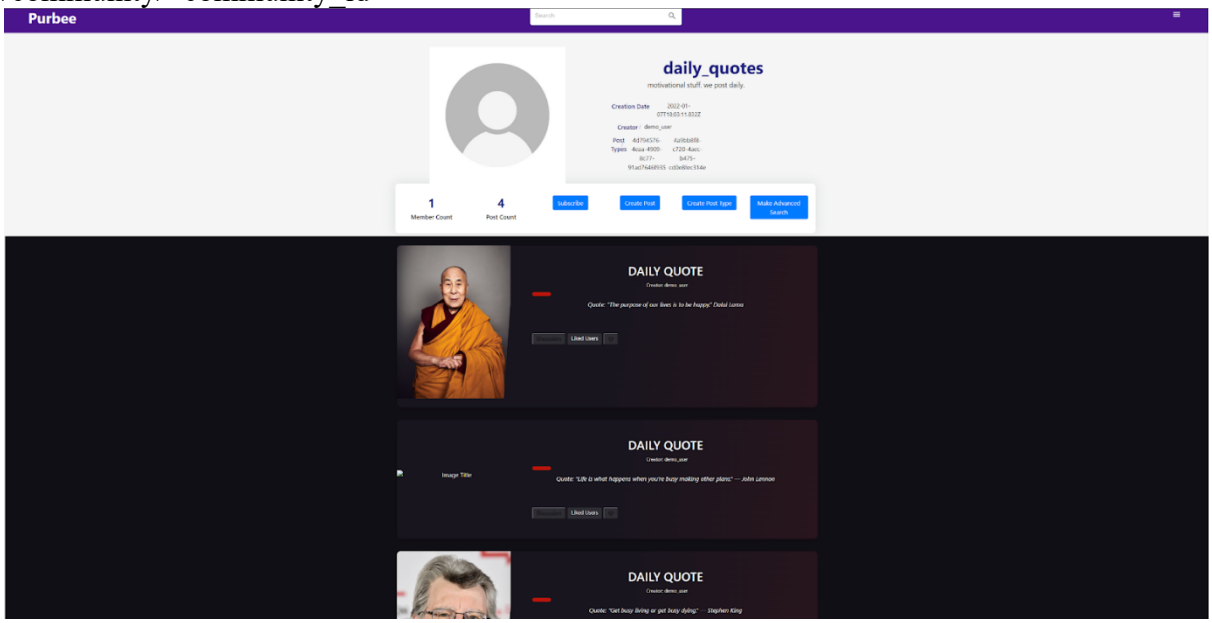
g) Simple search

/search-page/<community_id>



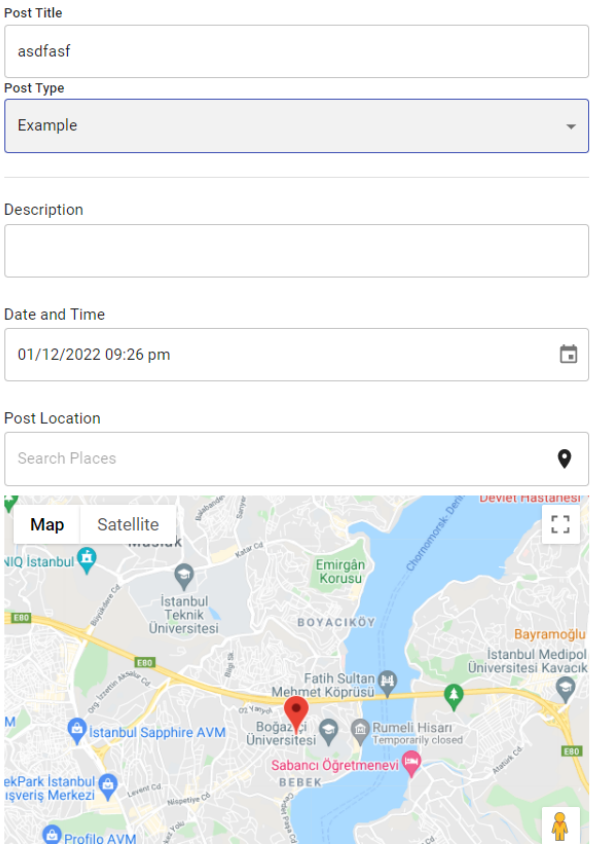
h) Community page

/community/<community_id>



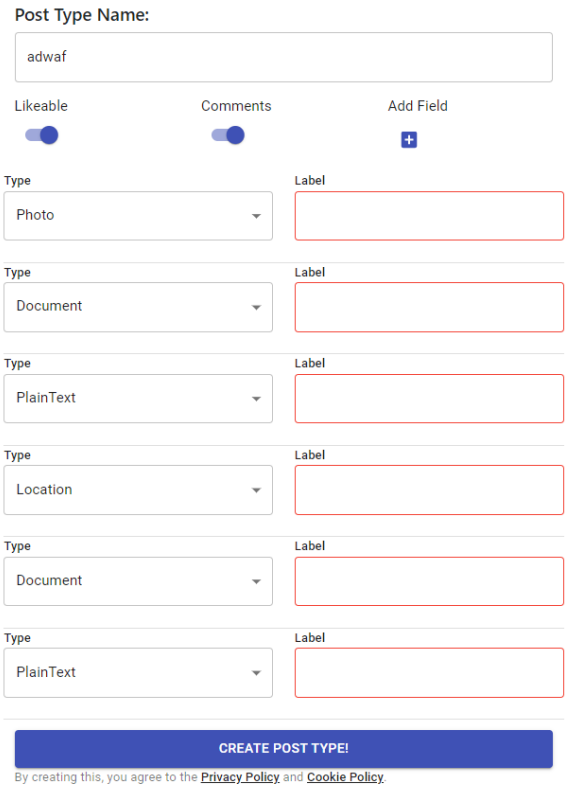
i) Create Post

/create-post/<community_id>



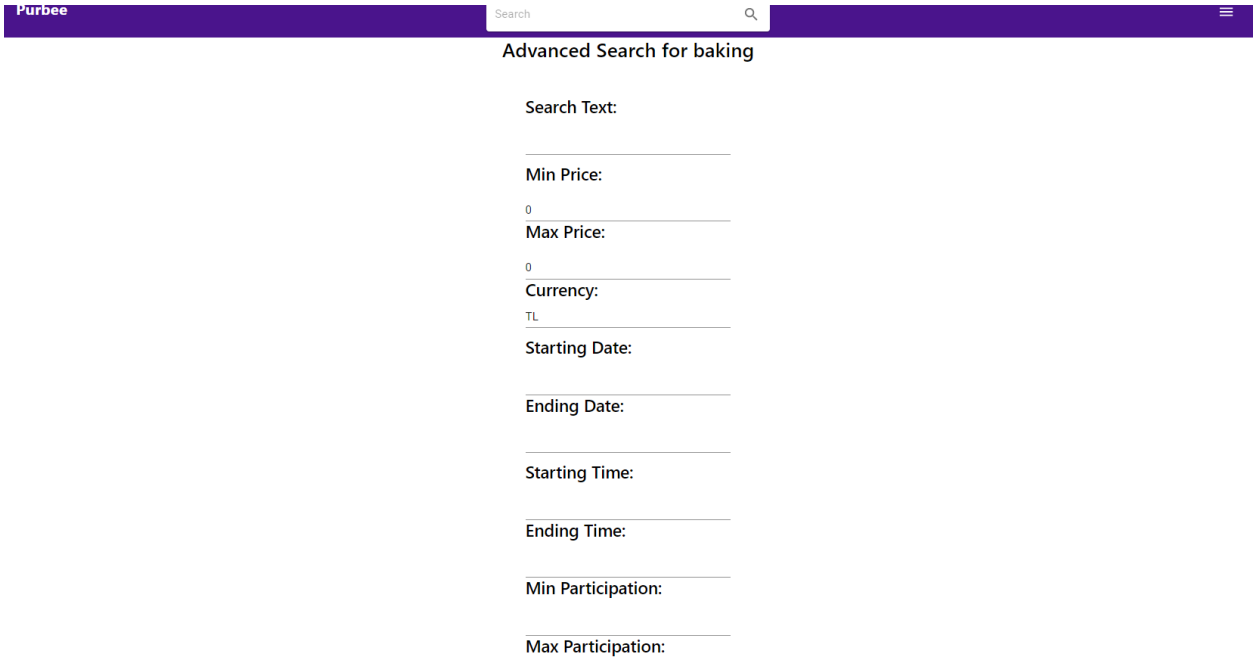
j) Create Post Type

/create-post-type/<community_id>



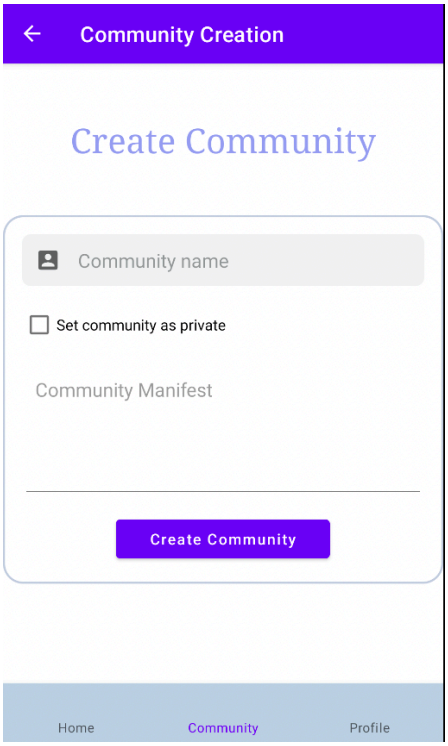
k) Advanced search

/advanced-search/<community_id>

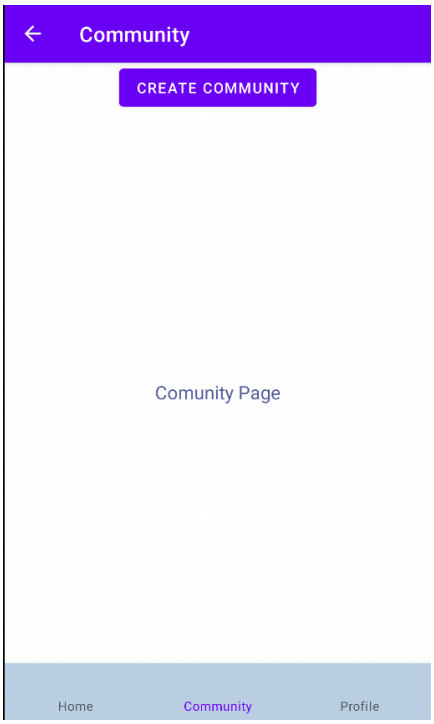


B. ANDROID

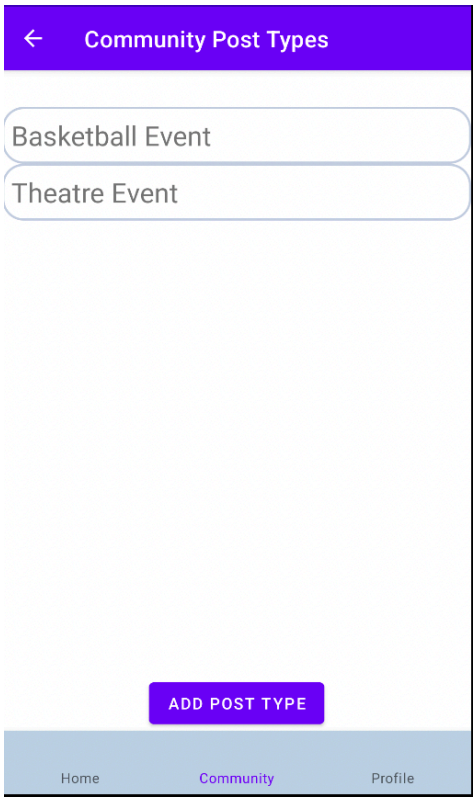
a) Community Creation



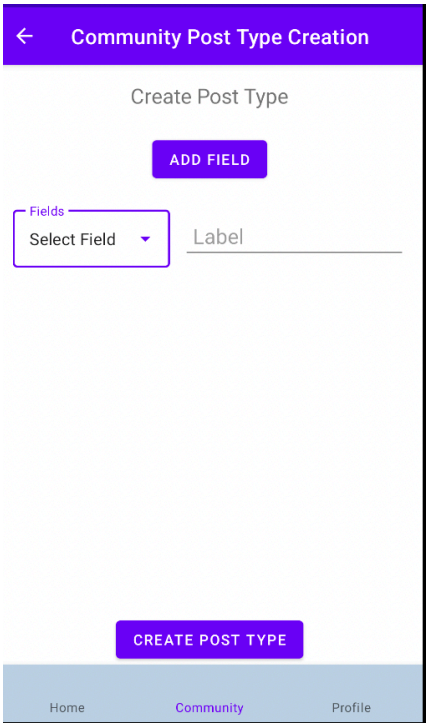
b) Community Section



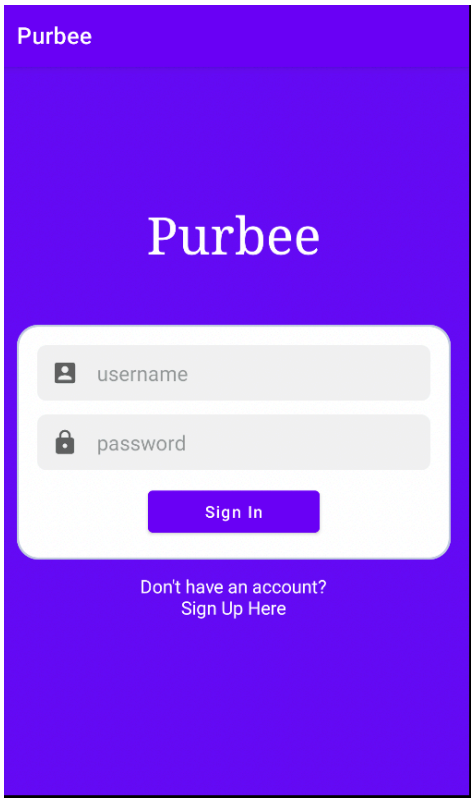
c) Community Post Types



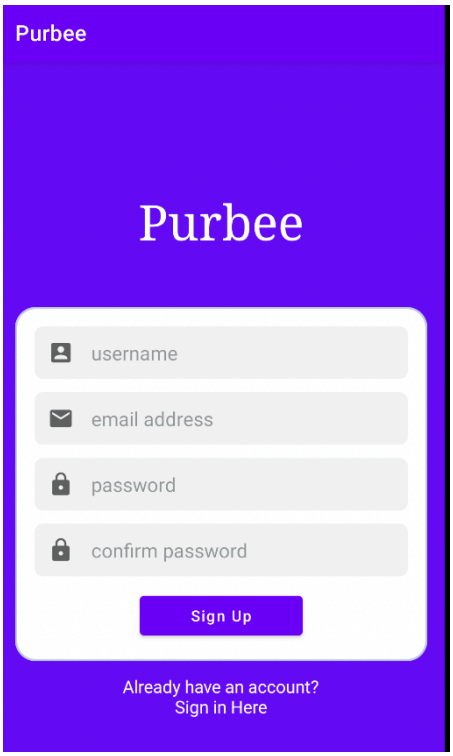
d) Create Post Type

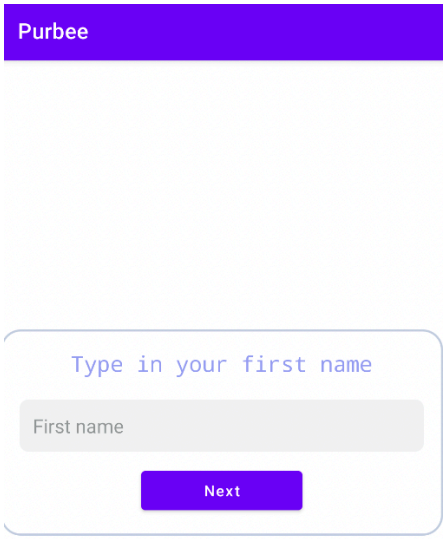


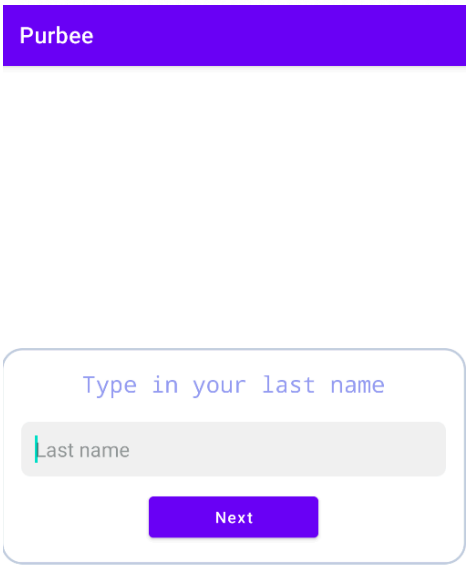
e) Sign In

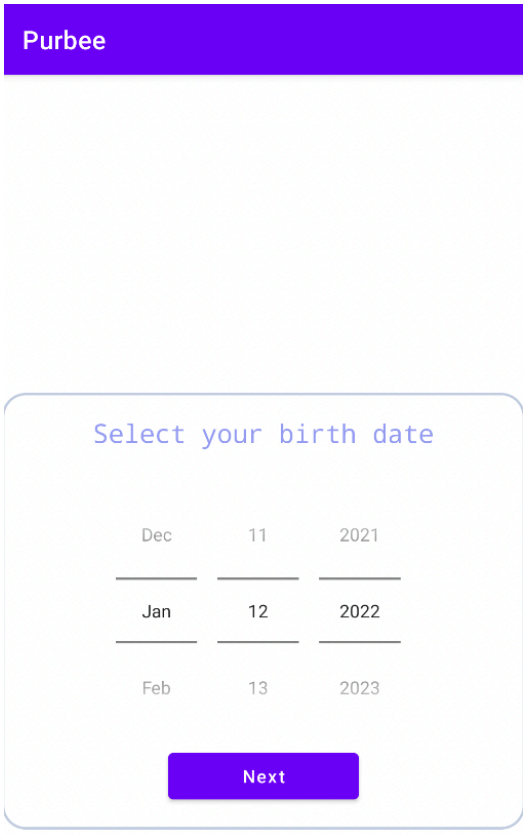


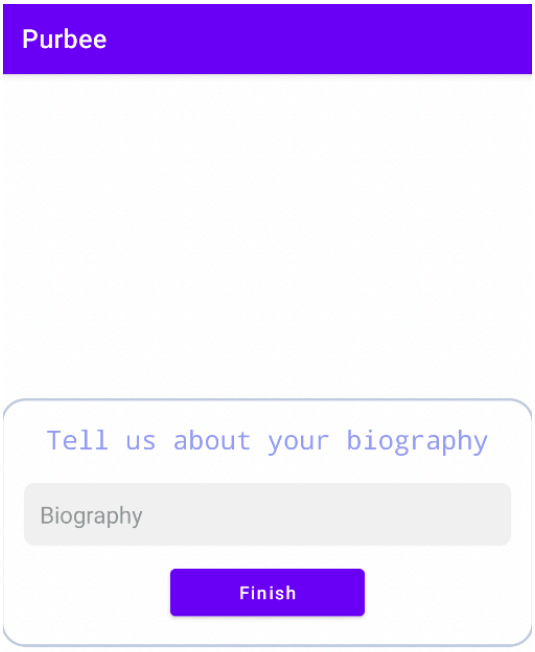
f) Sign Up




Purbee
Type in your first name
First name
Next

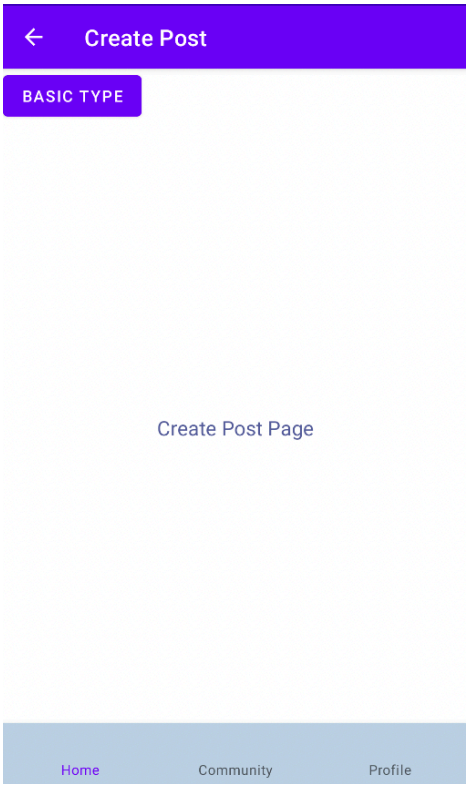

Purbee
Type in your last name
Last name
Next

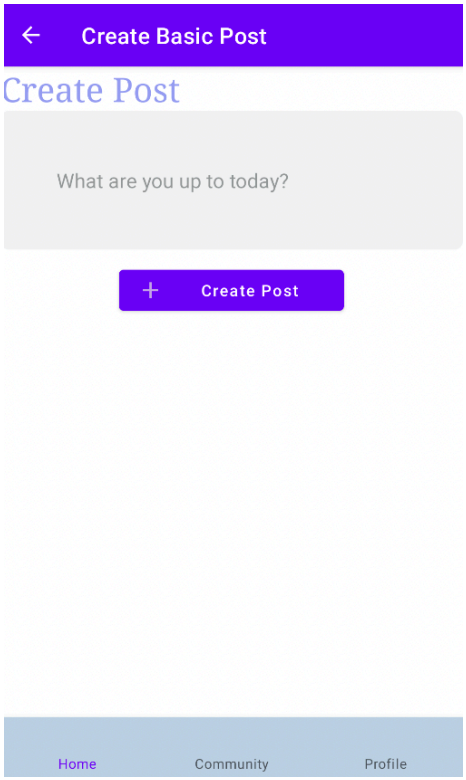

Purbee
Select your birth date
Dec 11 2021
Jan 12 2022
Feb 13 2023
Next


Purbee
Tell us about your biography
Biography
Finish

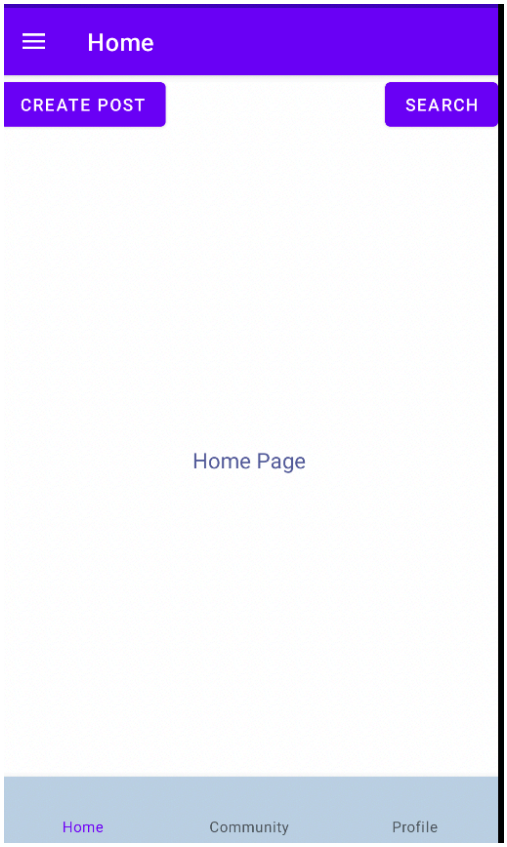
g) *Community Page*

h) *Create Post*

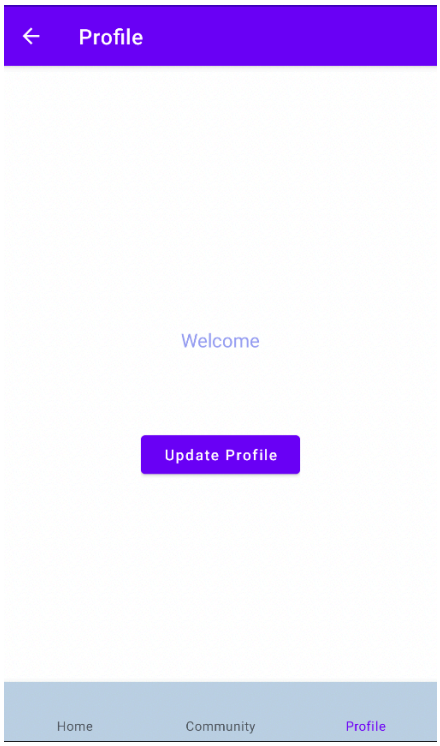



Create Basic Post
Create Post
What are you up to today?
Create Post
Home
Community
Profile

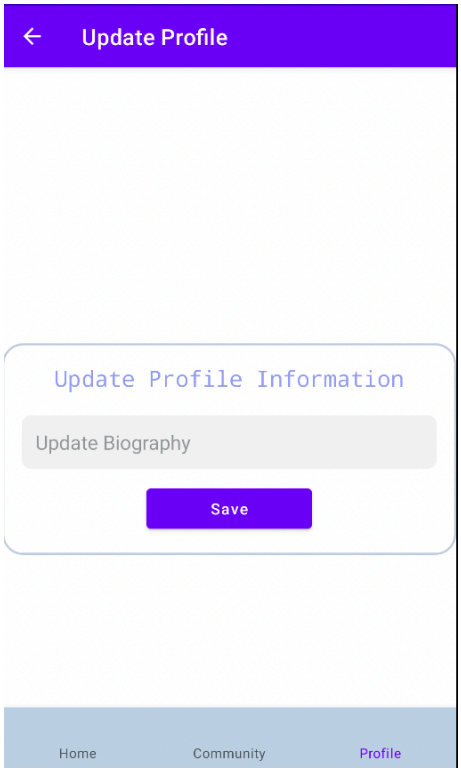
i) Home Page

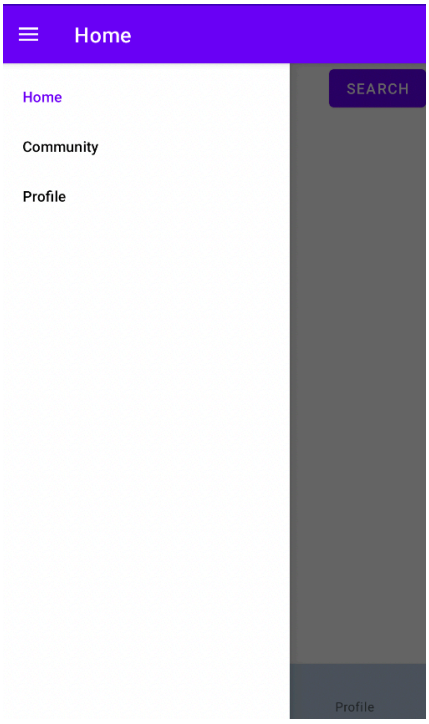

Home
CREATE POST
SEARCH
Home Page
Home
Community
Profile

j) Profile Page

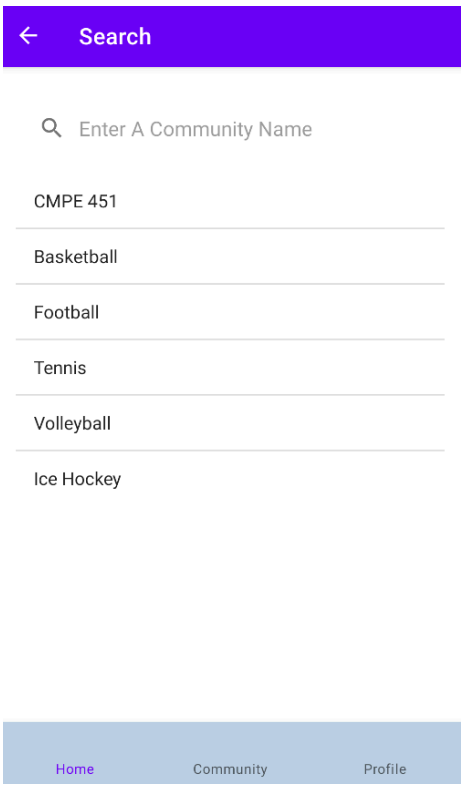


k) Update Profile




Home
Home
Community
Profile
SEARCH
Profile

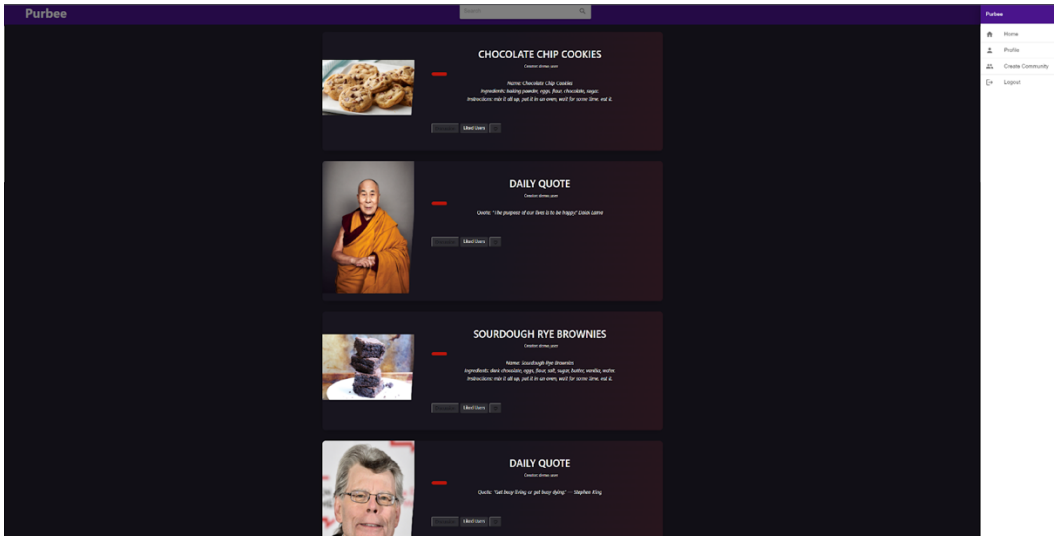
l) Search


Search
Enter A Community Name
CMPE 451
Basketball
Football
Tennis
Volleyball
Ice Hockey
Home
Community
Profile

6. USER MANUAL

a) WEB

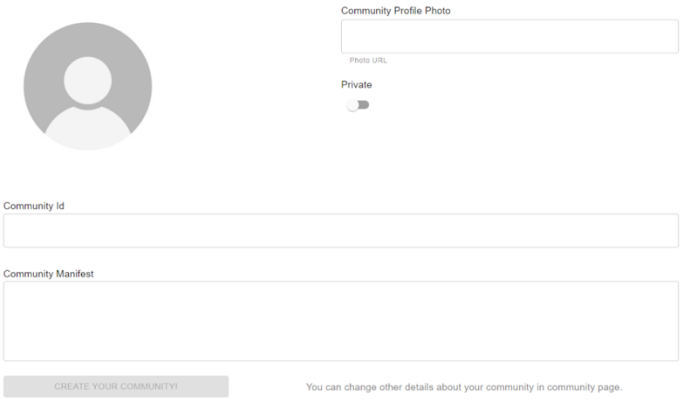
i. Homepage



In the homepage you can see the posts shared by people you follow or communities you are subscribed to. You can see the search bar for searching for the profiles and communities in the Purbee platform. In order to navigate through the app you can use the sidebar on the right. You can go to your profile page, home page, log out or create a new community of your own interest.

ii. Create Community Page

For creating your own community you must provide the following:



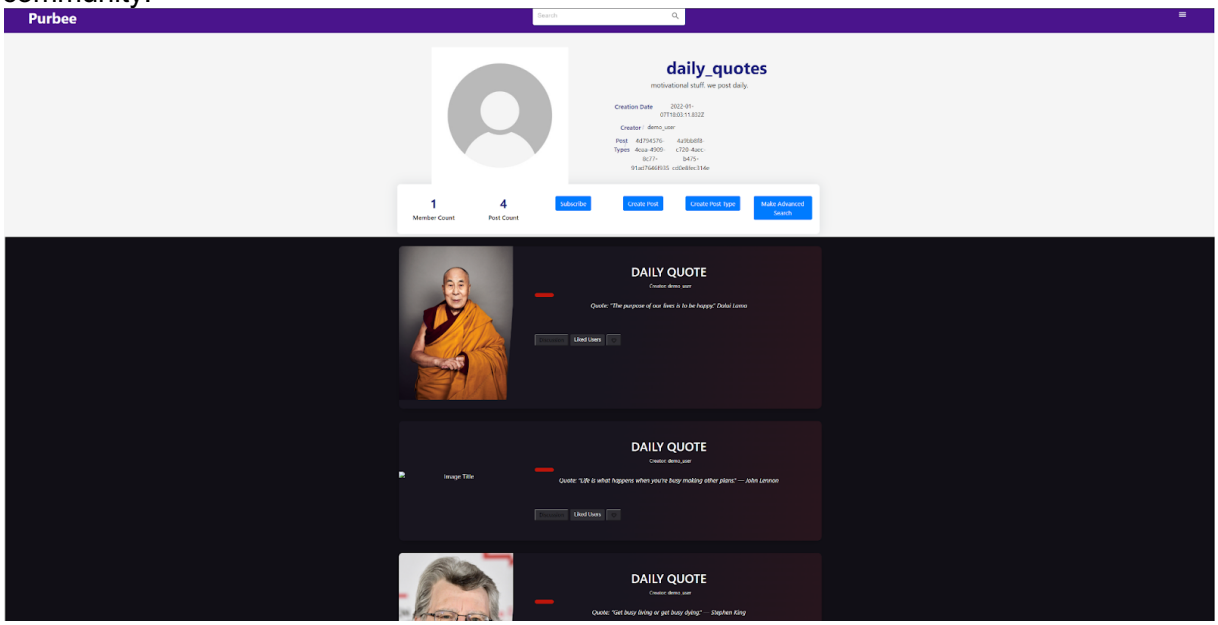
- Url to the community profile photo.
- Community Id

- Whether your community will be a private one.
- Manifest (Description about your community.)

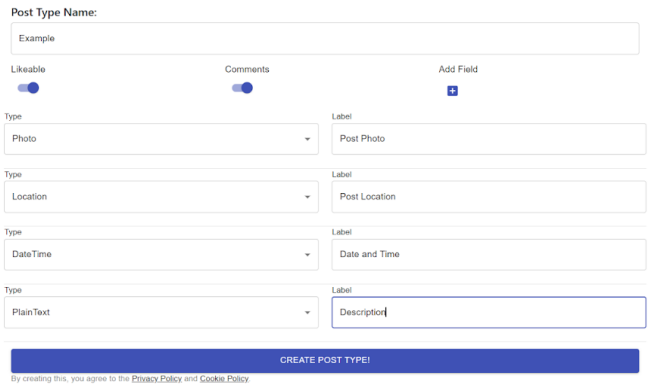
You can edit other details from the community page after clicking to the “Create Your Community” button.

iii. Community Page

In the community page, you can see the most recent posts shared in that community. You can also make advanced search of the posts according to their post types and fields of the community post types. You can also create a post, or a post type which will be serving as a template for users who would like to share in that community.



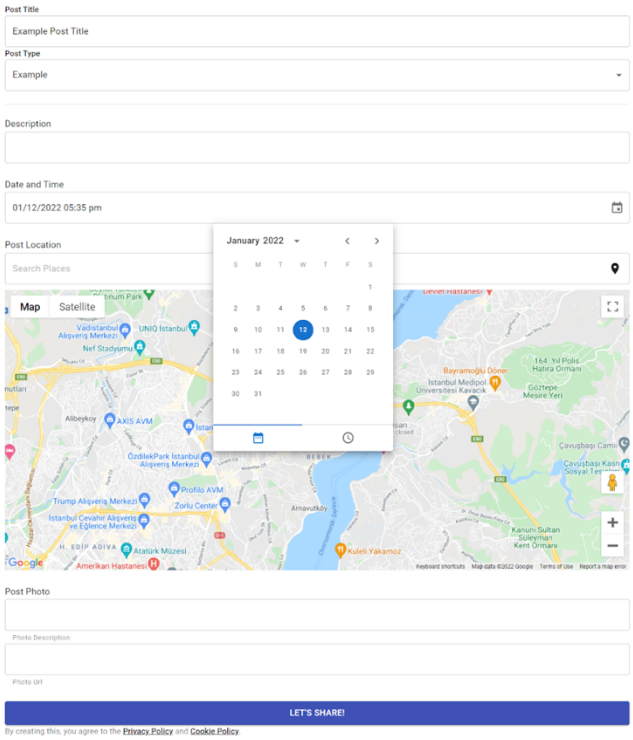
iv. Create Post Type



In order to create a post type you must be authorised as admin. For creating a post type you have to provide a post type name and if that post type will be able to be liked or commented

on. Later by providing your selection of the post type fields (PlainText, Photo, DateTime, Document, Price, Location) and their labels you can define your new post type.

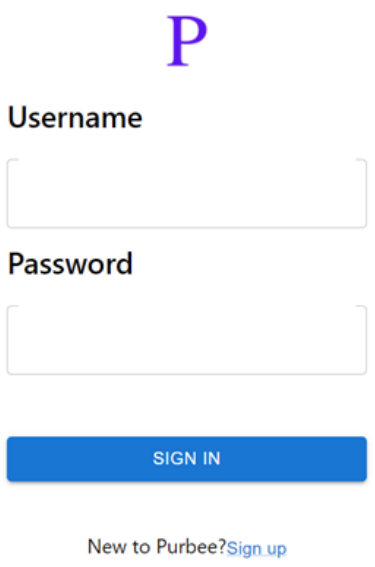
v. Create Post

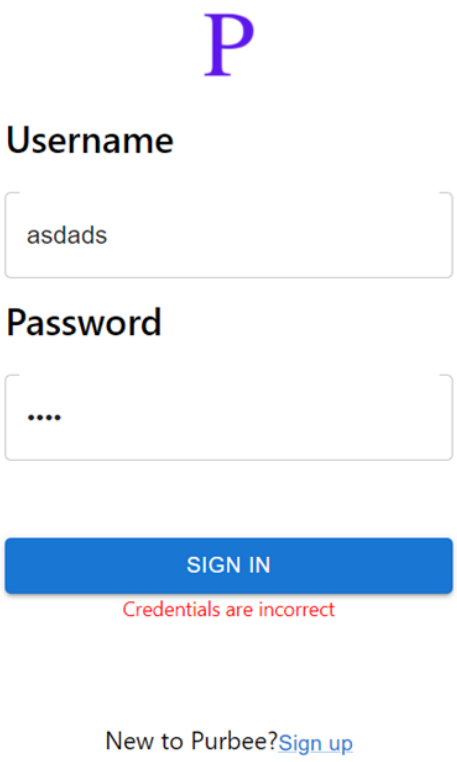


After creating the post type, you can see how your selection of post fields will show up for post creation. Every user will have to provide values of that post fields you picked in order to share their posts in the community.

vi. Login page:

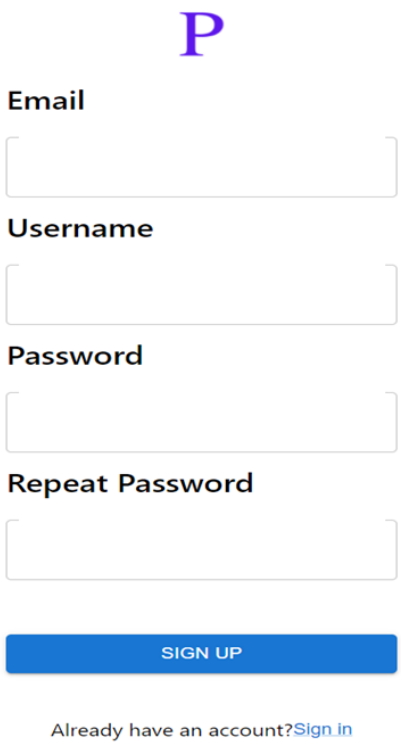
If you have signed up before, you can enter your credentials and log in to purbee. If credentials are wrong, you receive credentials are incorrect warning.





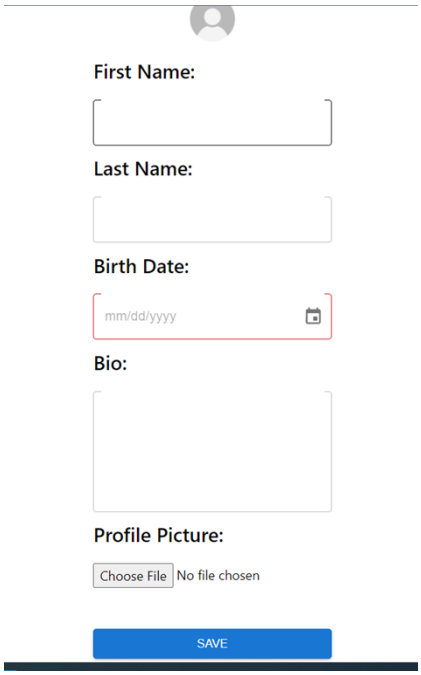
vii. Sign up:

A valid email, untaken username and a password consisting of at least one uppercase and one lowercase letter and one special character. Passwords have to match, otherwise passwords are not matching warning will be raised.

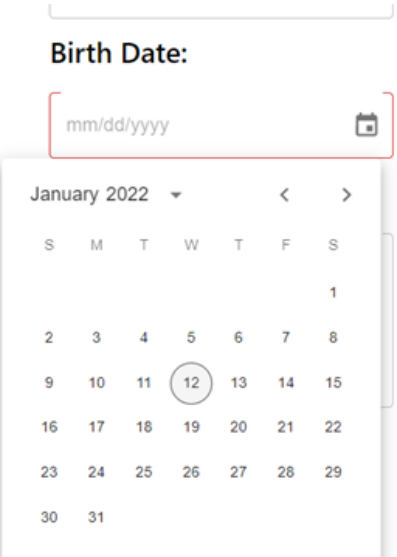


viii. Edit Profile:

After sign up, edit profile screen will show up.

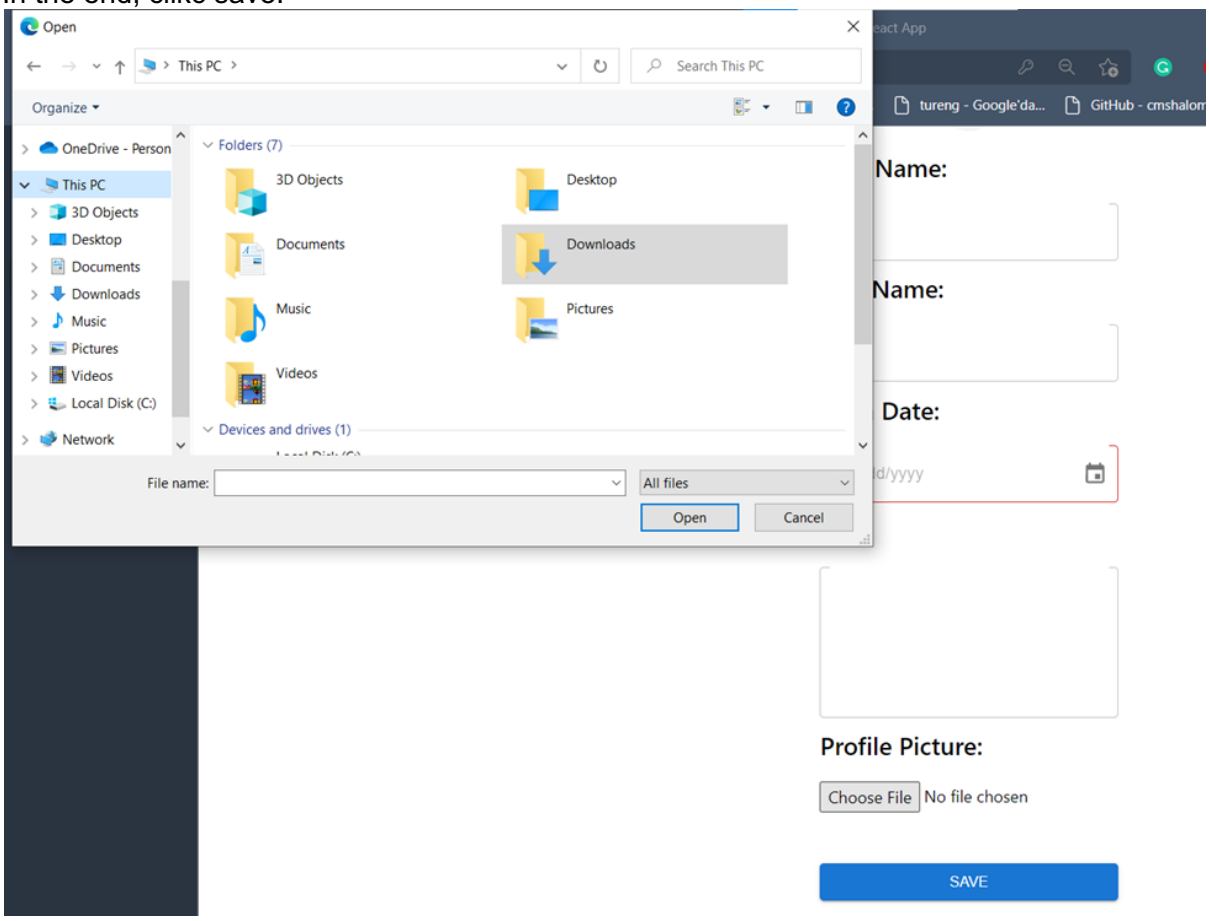


First name, Last name, birth date, bio, and profile pictures can be uploaded by the relevant fields.
Birth date is a date picker:



Profile picture:

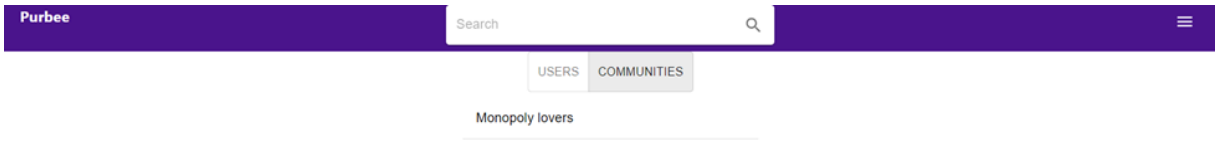
Clicking choose file will open file explorer of the operating system.
In the end, clikc save.



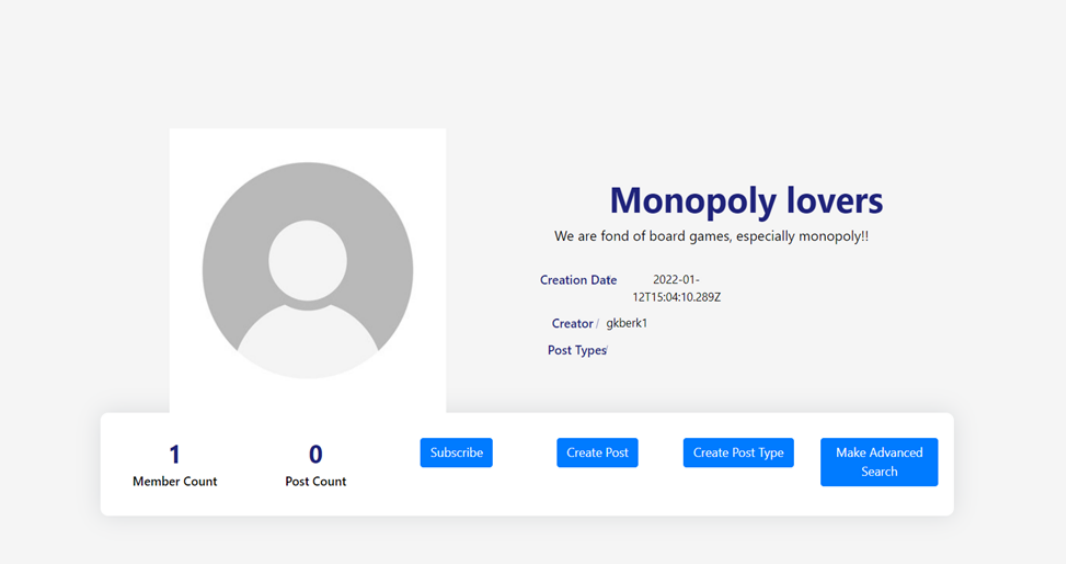
ix. Search:

Search bar is on top of all pages. A text is given as input, then magnify glass icon is clicked,

You should pick either users or communities.
Click on the result of search. You will be directed to the page.

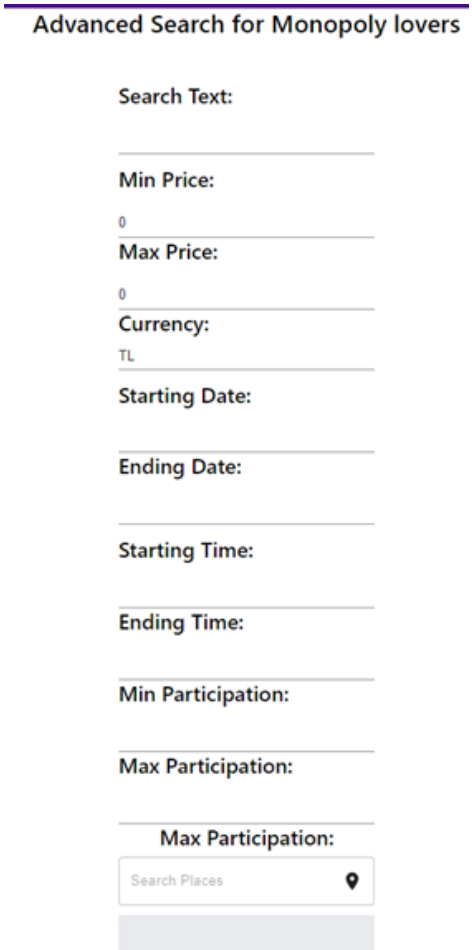


An example result, community page is returned from search.



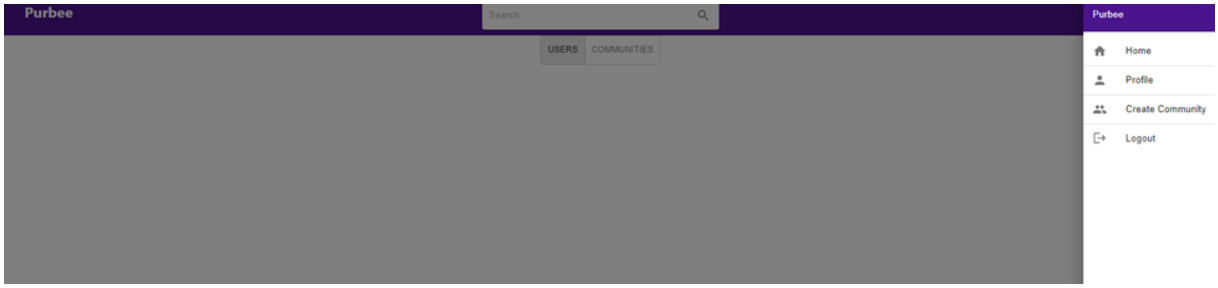
x. Advanced search:

Advanced search is like a filtering, price limits, currency, date interval, participation boundaries, location are given as input and satisfying posts shared in the community are returned.

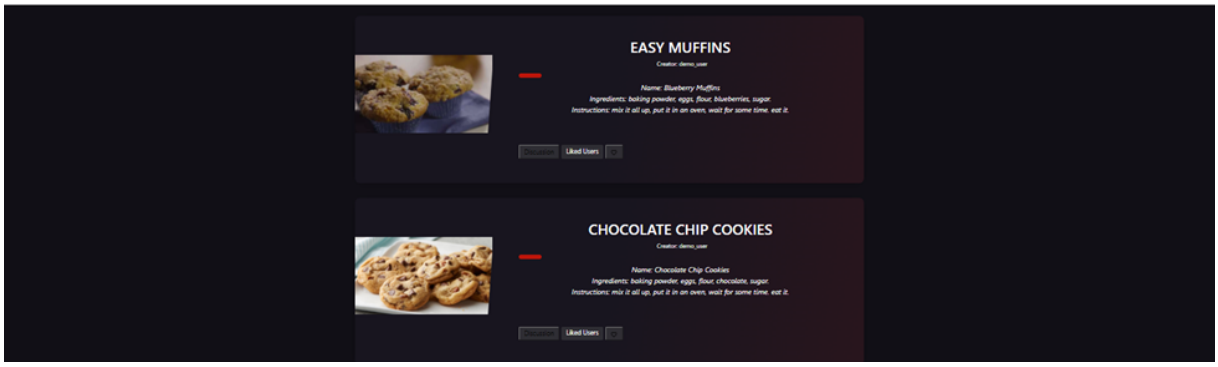


xi. Profile Page:

From side bar that is present in all pages, you can navigate to profile
Profile:

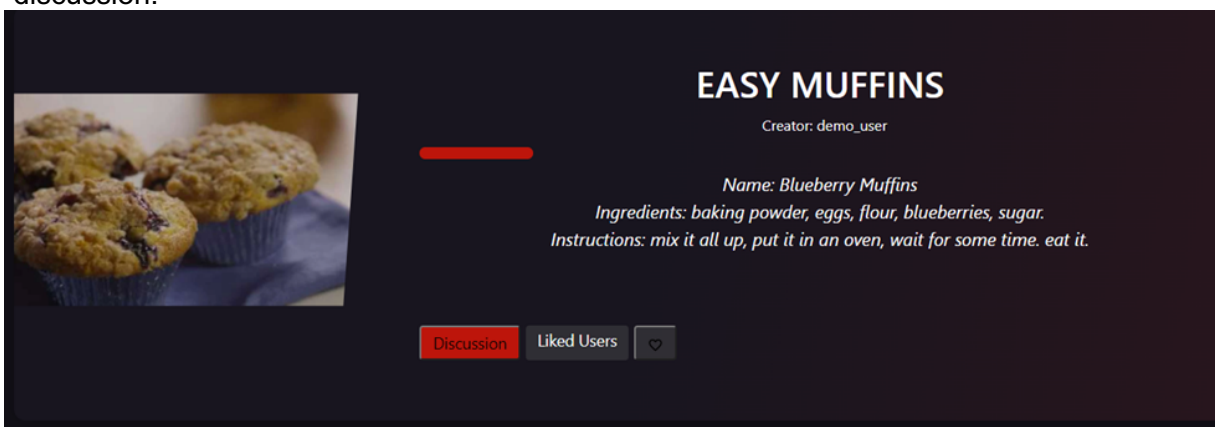


User's previous posts are all displayed here.

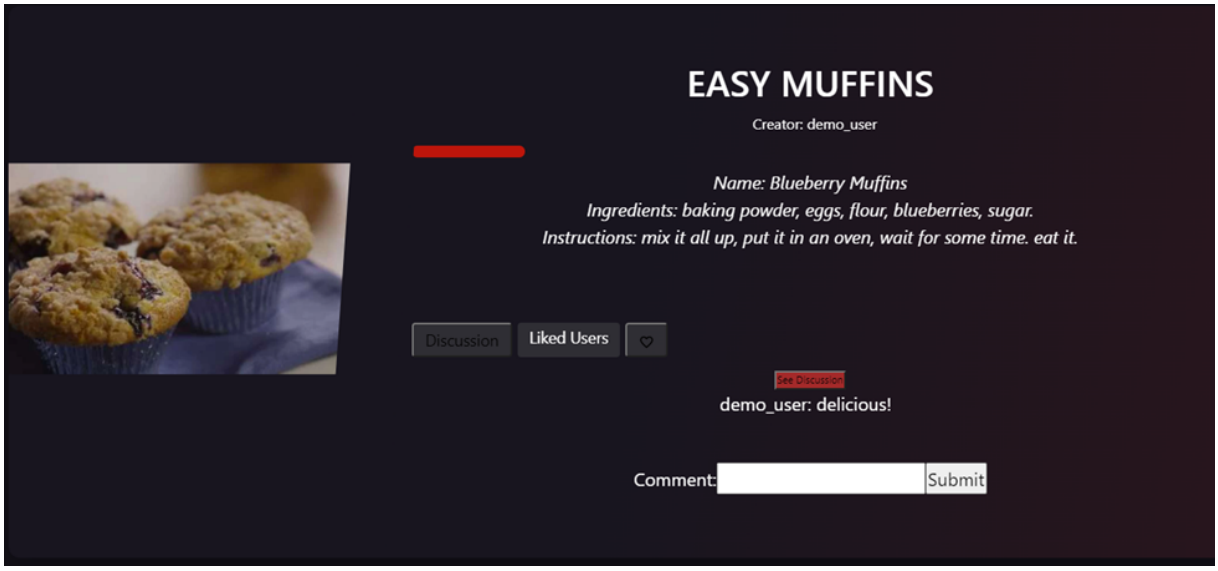


xii. Post Features

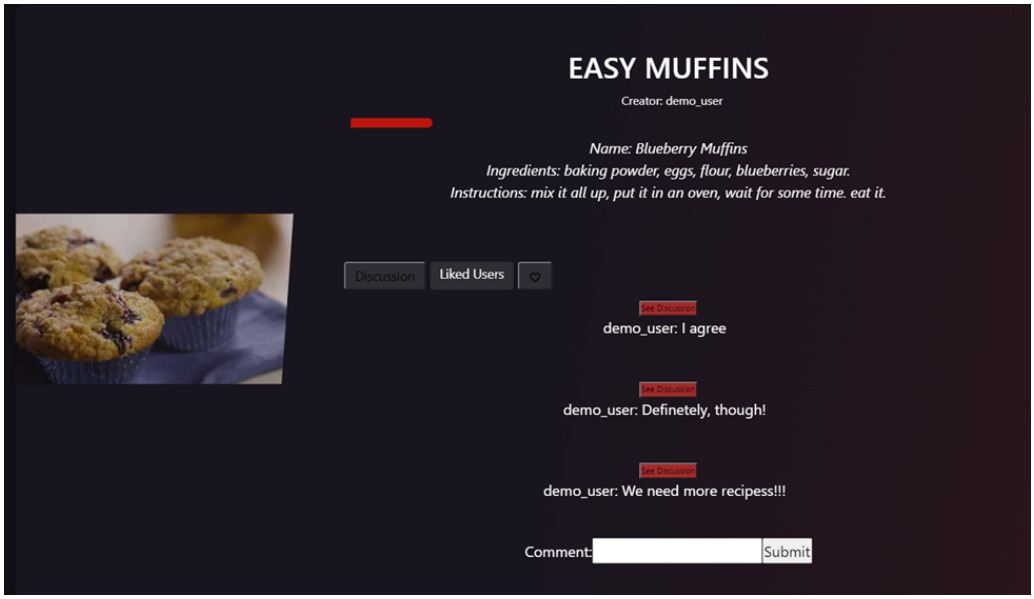
*discussion:



Click on the discussion, posts discussion will be displayed. Comments can be added.

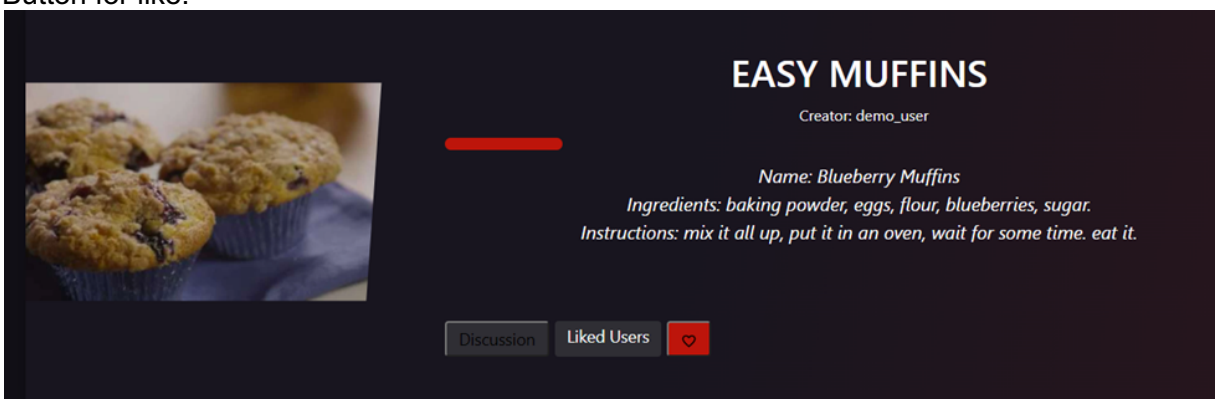


Each comment has its own discussion, click on See Discussion above a comment to see discussion below it.

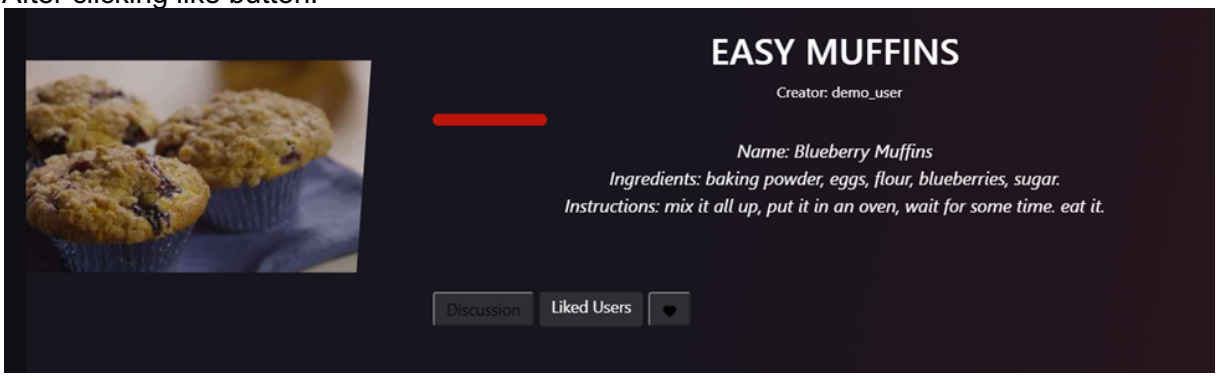


Discussion below the first comment.

*Like:
Button for like:

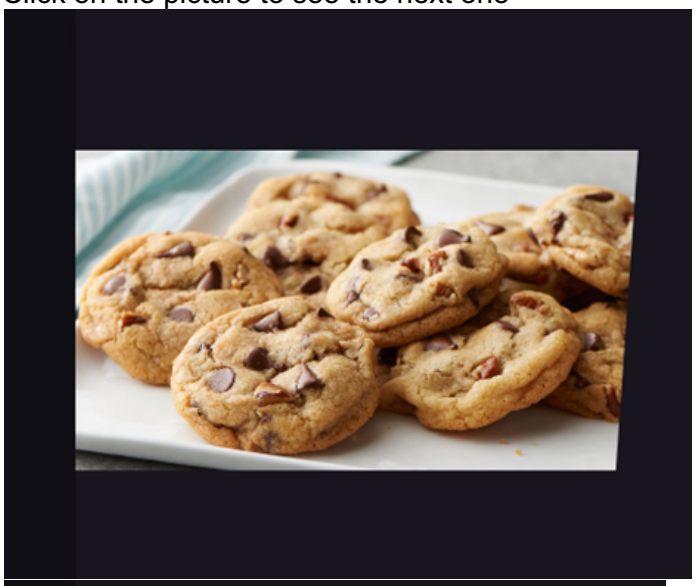


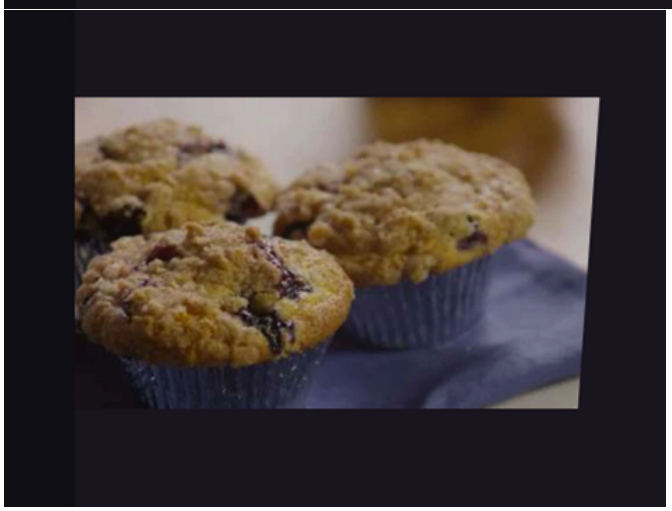
After clicking like button:



Liked user list

*Next picuture
Click on the picture to see the next one





b) ANDROID

i. TO SIGN IN

- Open the application and fill out requested information, e-mail and password.
- Then click Sign In.
- Sign in is succesfull, you will be navigated to home page.

ii. TO SIGN UP

- Click 'Sign In Here'.
- Fill out requested information, username, e-mail, password, password confirmation.
- Then click Sıgn Up.
- Type first name in the opened page and click next.
- Type last name in the opened page and click next.
- Select birth date in the opened page and click next.
- Type biography in the opened page and click next.
- Sign up is succesfull, you will be navigated to home page.

iii. TO SEARCH

- Click 'Search' at the rigth top corner of home page.
- Enter a community name to the field on the top.
- Communities including the typed string will appear as you type.
- You can click to one of the results in order to navigate through community page.

iv. TO SEE PROFILE PAGE

- Click 'Profile' at the navigation bar at the bottom.
- You will be navigated to your profile page.

v. TO UPDATE PROFILE

- In the profile page, click update profile button.
- Type biography in the opened page and click Save.
- Biography update is successfull.

vi. TO SEE SUBSCRIBED COMMUNITIES

- Click 'Community' at the navigation bar at the bottom.

- You will be navigated to community page which includes your subscribed communities.

vii. TO CREATE A COMMUNITY

- Click ‘Community’ at the navigation bar at the bottom.
- Click ‘Create Community’.
- Fill out requested information, community name, privacy option and community manifest.
- Click ‘Create Community’ to end this process.

viii. TO SEE POST TYPES OF A COMMUNITY

- Click ‘Community’ at the navigation bar at the bottom.
- Click on a community and navigate to its page.
- Click ‘Community Post Types’.
- You will be navigated to community post types page, there you can see the list of communitiy’s post types.

ix. TO CREATE A POST TYPE FOR A COMMUNITY

- Click ‘Community’ at the navigation bar at the bottom.
- Click on a community and navigate to its page.
- Click ‘Community Post Types’.
- Click ‘Add Post Type’
- Select fields you want in this post types from the dropdown list ‘Fields’.
- You should add a label for each field you add to this post type.
- You can add more field by clicking ‘Add Field’.
- Click ‘Create Post Type’ to end this process.

x. TO CREATE POST

- Click ‘Create Post’ at the left top corner of home page.
- Click on a post type on the opened page.
- On the navigated page, fill out the fields of selected post type.
- Click ‘Create Post’ to end this process.

7. SYSTEM MANUAL

a) WEB

Docker must be installed on the computer.
Run:
$ docker pull 2017400207/purbee_frontend
for pulling the docker image.
Then run:
$ docker run -p 3000:3000 --env
GOOGLE_MAPS_KEY=AIzaSyDqatrPkZebWGnKiLybd4nNFzERwmQCSCI
2017400207/purbee_frontend

To run the application. The application will be running on localhost:3000

b) ANDROID

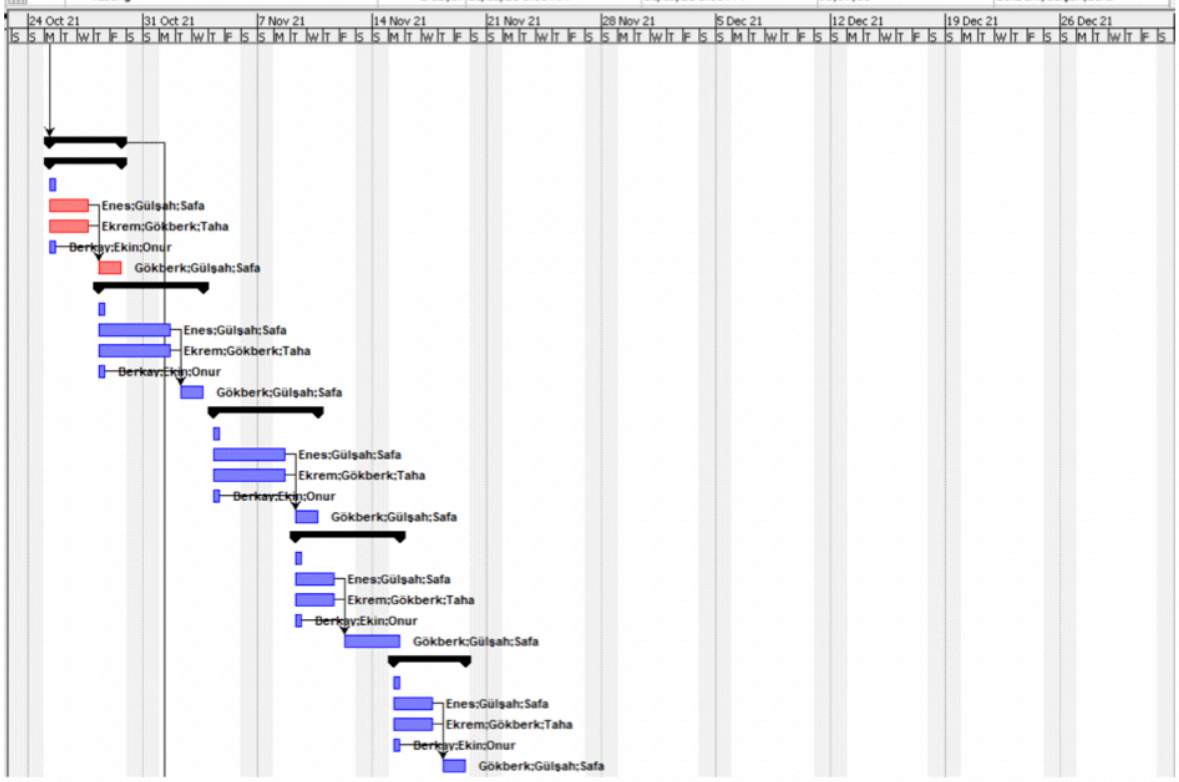
Install android emulator with the configurations:

Pixel 2 /with orientation portrait
Release name —> API 32
Size —> 5.0’’

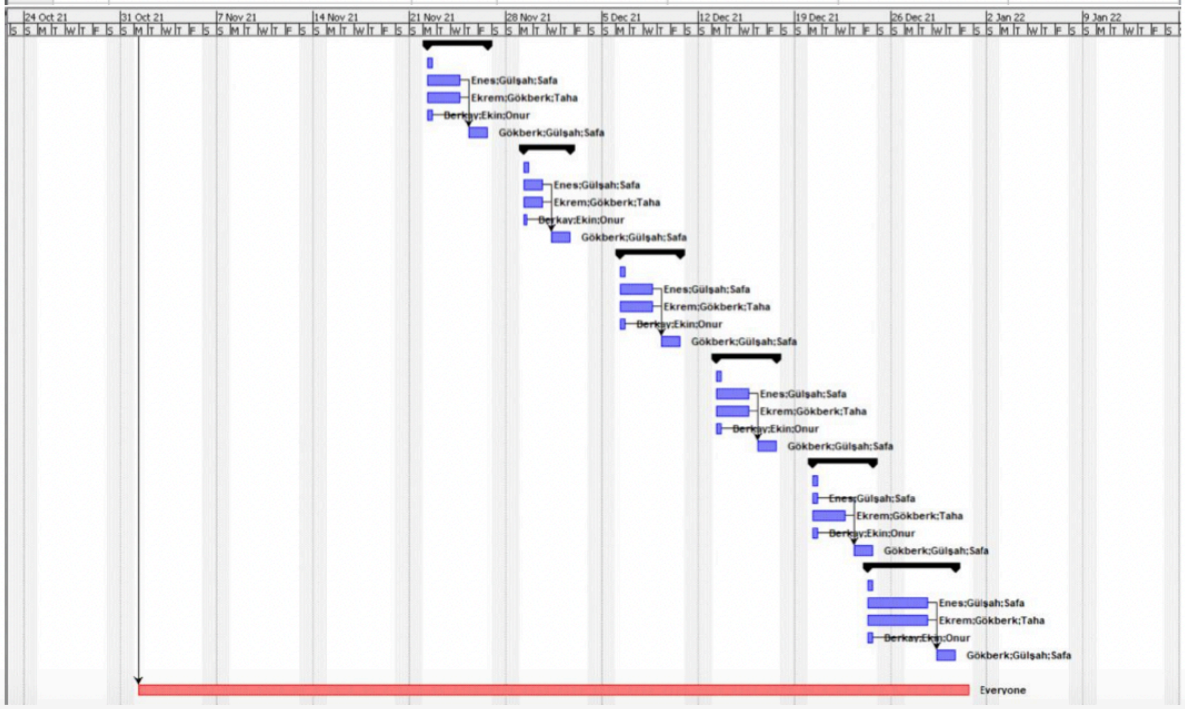
Install apk in the directory ‘android/release/debug’ to the emulator

8. PROJECT PLAN

Preparing Deliverables for Final Milestone	5 days?	**10/25/21 8:00 AM**	**10/29/21 5:00 PM**	1	
Implementation of Login Page	5 days?	**10/25/21 8:00 AM**	**10/29/21 5:00 PM**		
Deciding Login Page Features	1 day?	10/25/21 8:00 AM	10/25/21 5:00 PM		
Android Implementation	3 days?	10/25/21 8:00 AM	10/27/21 5:00 PM		Enes;Gülşah;Safa
Front-End Implementation	3 days?	10/25/21 8:00 AM	10/27/21 5:00 PM		Ekrem;Gökberk;Taha
Back-End Implementation	1 day?	10/25/21 8:00 AM	10/25/21 5:00 PM		Berkay;Ekin;Onur
Testing	2 days?	10/28/21 8:00 AM	10/29/21 5:00 PM	73;74;75	Gökberk;Gülşah;Safa
Implementation of Feed	5 days?	**10/28/21 8:00 AM**	**11/3/21 5:00 PM**		
Deciding Feed Features	1 day?	10/28/21 8:00 AM	10/28/21 5:00 PM		
Android Implementation	3 days?	10/28/21 8:00 AM	11/1/21 5:00 PM		Enes;Gülşah;Safa
Front-End Implementation	3 days?	10/28/21 8:00 AM	11/1/21 5:00 PM		Ekrem;Gökberk;Taha
Back-End Implementation	1 day?	10/28/21 8:00 AM	10/28/21 5:00 PM		Berkay;Ekin;Onur
Testing	2 days?	11/2/21 8:00 AM	11/3/21 5:00 PM	80;79;81	Gökberk;Gülşah;Safa
Implementation of Community Page	5 days?	**11/4/21 8:00 AM**	**11/10/21 5:00 PM**		
Deciding Community Page Features	1 day?	11/4/21 8:00 AM	11/4/21 5:00 PM		
Android Implementation	3 days?	11/4/21 8:00 AM	11/8/21 5:00 PM		Enes;Gülşah;Safa
Front-End Implementation	3 days?	11/4/21 8:00 AM	11/8/21 5:00 PM		Ekrem;Gökberk;Taha
Back-End Implementation	1 day?	11/4/21 8:00 AM	11/4/21 5:00 PM		Berkay;Ekin;Onur
Testing	2 days?	11/9/21 8:00 AM	11/10/21 5:00 PM	86;87;85	Gökberk;Gülşah;Safa
Implementation of User Profile Page	5 days?	**11/9/21 8:00 AM**	**11/15/21 5:00 PM**		
Deciding User Profile Page Features	1 day?	11/9/21 8:00 AM	11/9/21 5:00 PM		
Android Implementation	3 days?	11/9/21 8:00 AM	11/11/21 5:00 PM		Enes;Gülşah;Safa
Front-End Implementation	3 days?	11/9/21 8:00 AM	11/11/21 5:00 PM		Ekrem;Gökberk;Taha
Back-End Implementation	1 day?	11/9/21 8:00 AM	11/9/21 5:00 PM		Berkay;Ekin;Onur
Testing	2 days?	11/12/21 8:00 AM	11/15/21 5:00 PM	93;91;92	Gökberk;Gülşah;Safa
Implementation of Community Creation Page	5 days?	**11/15/21 8:00 AM**	**11/19/21 5:00 PM**		
Deciding Community Creation Page Features	1 day?	11/15/21 8:00 AM	11/15/21 5:00 PM		
Android Implementation	3 days?	11/15/21 8:00 AM	11/17/21 5:00 PM		Enes;Gülşah;Safa
Front-End Implementation	3 days?	11/15/21 8:00 AM	11/17/21 5:00 PM		Ekrem;Gökberk;Taha
Back-End Implementation	1 day?	11/15/21 8:00 AM	11/15/21 5:00 PM		Berkay;Ekin;Onur
Testing	2 days?	11/18/21 8:00 AM	11/19/21 5:00 PM	99;97;98	Gökberk;Gülşah;Safa



101	**Implementation of Post Type Creation Page**	5 days?	**11/22/21 8:00 AM**	**11/26/21 5:00 PM**		
102	Implementation of Post Type Creation Page	1 day?	11/22/21 8:00 AM	11/22/21 5:00 PM		
103	Android Implementation	3 days?	11/22/21 8:00 AM	11/24/21 5:00 PM		Enes;Gülşah;Safa
104	Front-End Implementation	3 days?	11/22/21 8:00 AM	11/24/21 5:00 PM		Ekrem;Gökberk;Taha
105	Back-End Implementation	1 day?	11/22/21 8:00 AM	11/22/21 5:00 PM		Berkay;Ekin;Onur
106	Testing	2 days?	11/25/21 8:00 AM	11/26/21 5:00 PM	105;103;104	Gökberk;Gülşah;Safa
107	**Implementation of Search Page**	4 days?	**11/29/21 8:00 AM**	**12/2/21 5:00 PM**		
108	Deciding Search Page Features	1 day?	11/29/21 8:00 AM	11/29/21 5:00 PM		
109	Android Implementation	2 days?	11/29/21 8:00 AM	11/30/21 5:00 PM		Enes;Gülşah;Safa
110	Front-End Implementation	2 days?	11/29/21 8:00 AM	11/30/21 5:00 PM		Ekrem;Gökberk;Taha
111	Back-End Implementation	0.667 days?	11/29/21 8:00 AM	11/29/21 2:20 PM		Berkay;Ekin;Onur
112	Testing	2 days?	12/1/21 8:00 AM	12/2/21 5:00 PM	111;110;109	Gökberk;Gülşah;Safa
113	**Implementation of Message Page**	5 days?	**12/6/21 8:00 AM**	**12/10/21 5:00 PM**		
114	Deciding Message Page Features	1 day?	12/6/21 8:00 AM	12/6/21 5:00 PM		
115	Android Implementation	3 days?	12/6/21 8:00 AM	12/8/21 5:00 PM		Enes;Gülşah;Safa
116	Front-End Implementation	3 days?	12/6/21 8:00 AM	12/8/21 5:00 PM		Ekrem;Gökberk;Taha
117	Back-End Implementation	1 day?	12/6/21 8:00 AM	12/6/21 5:00 PM		Berkay;Ekin;Onur
118	Testing	2 days?	12/9/21 8:00 AM	12/10/21 5:00 PM	115;116;117	Gökberk;Gülşah;Safa
119	**Implementation of Post Details**	5 days?	**12/13/21 8:00 AM**	**12/17/21 5:00 PM**		
120	Deciding Post DetailsFeatures	1 day?	12/13/21 8:00 AM	12/13/21 5:00 PM		
121	Android Implementation	3 days?	12/13/21 8:00 AM	12/15/21 5:00 PM		Enes;Gülşah;Safa
122	Front-End Implementation	3 days?	12/13/21 8:00 AM	12/15/21 5:00 PM		Ekrem;Gökberk;Taha
123	Back-End Implementation	1 day?	12/13/21 8:00 AM	12/13/21 5:00 PM		Berkay;Ekin;Onur
124	Testing	2 days?	12/16/21 8:00 AM	12/17/21 5:00 PM	122;123;121	Gökberk;Gülşah;Safa
125	**Implementation of Admin Panel**	5 days?	**12/20/21 8:00 AM**	**12/24/21 5:00 PM**		
126	Deciding Admin Panel Features	1 day?	12/20/21 8:00 AM	12/20/21 5:00 PM		
127	Android Implementation	1 day?	12/20/21 8:00 AM	12/20/21 5:00 PM		Enes;Gülşah;Safa
128	Front-End Implementation	3 days?	12/20/21 8:00 AM	12/22/21 5:00 PM		Ekrem;Gökberk;Taha
129	Back-End Implementation	1 day?	12/20/21 8:00 AM	12/20/21 5:00 PM		Berkay;Ekin;Onur
130	Testing	2 days?	12/23/21 8:00 AM	12/24/21 5:00 PM	129;127;128	Gökberk;Gülşah;Safa
131	**Implementation of Notification Page**	5 days?	**12/24/21 8:00 AM**	**12/30/21 5:00 PM**		
132	Deciding Notification Page Features	1 day?	12/24/21 8:00 AM	12/24/21 5:00 PM		
133	Android Implementation	3 days?	12/24/21 8:00 AM	12/28/21 5:00 PM		Enes;Gülşah;Safa
134	Front-End Implementation	3 days?	12/24/21 8:00 AM	12/28/21 5:00 PM		Ekrem;Gökberk;Taha
135	Back-End Implementation	1 day?	12/24/21 8:00 AM	12/24/21 5:00 PM		Berkay;Ekin;Onur
136	Testing	2 days?	12/29/21 8:00 AM	12/30/21 5:00 PM	135;134;133	Gökberk;Gülşah;Safa
137	End-to-end Testing	5 days?	7/22/21 8:00 AM	7/28/21 5:00 PM		
138	Preparing Deliverables for Milestone 3	45 days?	11/1/21 8:00 AM	12/31/21 5:00 PM	70	Everyone



9. SOFTWARE REQUIREMENTS SPECIFICATION

Glossary

Admin: The main rulers of the application. This position is only for application developer team.
Community: A platform with some ideology around particular subject. It is the main communication channel to share created purposeful content.
Community Admin: Superior manager of a community. This person manages all the data of the community and the community members.
Community Creation Page:
Community Imperatives: Compulsory fields that must take place in any of the community post types.
Community Post Types: Varios post types in a community, either created and customized by community creator or given default(text and event).
Community Profile Page: A dedicated page for every other community to show profile picture, manifest, list of subscribers, and timeline to users.
Content: Anything published in a Community.
Default Post Types: "Community Post Types" that exist without the effort of community creator, upon a creation of a community. Text and Event.
Engaging discussion: Replying a comment.
Event: A content which is an invitation and announcement for some activities.
Field: A single piece of information that is part of a post or a template.
Follower: A user who wants to see some other users content. To be a Follower, the user should be approved by the aimed user.
Guest User: Person who is not registered to the application. This person still can use the application with some restrictions.
Interactive fields: Poll and Participation fields.
Main Feed: A Registered User Spesific page which shows posts of followed users and subscribed communities.
Manifest: Introduction text of a user or a community to give brief explanation about themselves to other users.
Non-Follower: A user who does not follow the specified user.
Notification Center: A page which accumulates recieved notifications regarding to a registered user.
Post: A content which has predefined structures to bring out some experience a user acquired or cumulate some knowledge.
Post Type Creation Page: A page which is used to create new post types from given fields as a part of community profile page.
Privacy Level 1: Community is public and user is not a member of community.
Privacy Level 2: Community is public and user is a member of community.
Privacy Level 3: Community is private and user is not a member of community.
Privacy Level 4: Community is private and user is a member of community.
Profile Page: A personal page for every other user to show personal data(profile picture, biography, subscribed communities and timeline) to other users.
PSP (Purpose Spesific Post) Template: Basic post template that could be configured to made a community default by community admin for members to use. Provided by the system.
Public Content: Content which is visible to not only subscribers, but all users and guests.

Search Bar: The bar that enables users to search for communities and registered users.
Subscription: Action that a user can take. After the subscription, a user can reach all the content from subscribed community.
Timeline: Posts that a user post to a public community.
Turning on notifications of a post: The action of receiving notifications about any interactions related to the post.
User: Person who registered to the application. This person can use all benefits of the application. For example he/she can create or join to a community while being able to keep contact with other users.

1. FUNCTIONAL REQUIREMENTS

1.1. USER REQUIREMENTS

1.1.1. Sign Up

1.1.1.1. Users shall be able to start the sign up process by providing an user-name, e-mail and password in the sign up page.
1.1.1.2. The user shall provide unique user-name and e-mail in the sign up process. [NON FUNCTIONAL]
1.1.1.3. Users shall be able to sign up using their google and facebook accounts.

1.1.2. Sign In

1.1.2.1. Registered users shall be able to sign in via username/password or e-mail/password in the sign in page.
1.1.2.2. Registered users who signed in shall be able to sign out successfully when log out button clicked by registered user.
1.1.2.3. Registered sers shall be able to sign in using their google and facebook accounts.

1.1.3. Guest Users

1.1.3.1. Guest users shall be able to search for communities and profiles.
1.1.3.2. Guest users shall be able to view the public content of communities and profiles.

1.1.4. Registered Users

1.1.4.1. Common Features

1.1.4.1.1. Registered users shall inherit the Guest users' requirements specified in requirement 1.1.3.
1.1.4.1.2. Registered users' profile page shall have profile picture, biography, list of subscribed communities and timeline.
1.1.4.1.3. Registered users shall be able to change their profile picture from their profile page.
1.1.4.1.4. Registered users shall be able to subscribe to public communities from the community page.
1.1.4.1.5. Registered users shall be able to send request to the community admins for subscribing private communities from the community page.

1.1.4.1.6. Registered users shall be able to leave communities that they have subscribed before from the community page.
1.1.4.1.7. Registered users shall be able to report the content they find inappropriate or offensive from the post.
1.1.4.1.8. Registered users shall be able to directly follow other registered users with public profile from their profile page.
1.1.4.1.9. Registered users shall be able to send request for following registered users with private profile from their profile page.
1.1.4.1.10. Registered users shall be able to unfollow a registered user from their profile page.
1.1.4.1.11. Registered users shall be able to accept or reject following requests from the notification center.
1.1.4.1.12. Registered users should be able to contribute to content in a Community within the predetermined ranges that Community Admin has ruled. (You can check the privacy levels (1,2,3,4) from glossary part.)):
1.1.4.1.12.1. Creating Posts from the community profile page via using a post type from community post types. (2,4)
1.1.4.1.12.2. Interact with interactive fields of the post from post. (1,2,4)
1.1.4.1.12.3. Commenting posts from post. (1,2,4)
1.1.4.1.12.4. Hitting the like button whichin the post. (1,2,4)
1.1.4.1.12.5. Replying to comments from the post. (1,2,4)
1.1.4.1.12.6. View Community Posts from timelines and main feed. (1,2,4)
1.1.4.1.12.7. View Community's other members from the community profile page. (1,2,4)
1.1.4.1.12.8. Registered users shall be able to turn on notifications of posts published in the community from the post. (1,2,4)
1.1.4.1.13. Registered users shall be able to set privacy option of the Community they create in the community creation process from community creation page.
1.1.4.1.14. Registered users shall be able to set community profile photo in the community creation process from community creation page.
1.1.4.1.15. Registered users shall be able to set manifest in the community creation process from community creation page.
1.1.4.1.16. Registered users shall be able to search and filter posts according to the specific post type and its fields in the "main feed".

1.1.4.2. Community Admin

1.1.4.2.1. Community admins shall inherit the Registered User's requirements specified in requirement part 1.1.4.1.
1.1.4.2.2. Community admins shall be able to set another subscriber of the community as a community admin from the community's subscriber list.
1.1.4.2.3. Community admins shall be able to warn Community Members because of hazardous posts from the post.
1.1.4.2.4. Community admins shall be able to remove Community Members from the Community from the community's subscriber list.
1.1.4.2.5. Community admins shall be able to edit the Community's community picture from the community profile page.
1.1.4.2.6. Community admins shall be able to edit the Community's community manifest from the community profile page.
1.1.4.2.7. Community admins shall be able to create post types using these fields from the post type creation page:
Plain text field.

Photo field.
Date-time field.
Location field.
Price field.
Participation field.
Document field.
Poll field.

1.2. SYSTEM REQUIREMENTS

1.2.1. Recommendation

1.2.1.1. System shall be able to recommend content and persons of interest.

1.2.2. Notification

1.2.2.1. Users who turned on their notifications to a post shall be notified with the developments on that post.
1.2.2.2. App should provide notifications to the notification center of the mobile device.

1.2.3. Authorization

1.2.3.1. The privacy levels of users who accept the invitations from community admins shall be set accordingly.
1.2.3.2. The privacy levels of users who removed from the community will be set accordingly.
1.2.3.3. The privacy levels of users will be set according to the privacy option of the community.

1.2.4. Community

1.2.4.1. Post histories of communities will be stored.
1.2.4.2. Community creator is the default Community Admin.
1.2.4.3. Communities' profile page shall have community picture, manifest, subscribed members list and timeline.
1.2.4.4. Communities have default post types named as default post types

1.2.5. Searching/Filtering

1.2.5.1. Upon searching related communities, people and posts shall be indexed under the search bar.

1.2.6. Home Page

1.2.6.1. System shall provide search bar, tabs and community posts.
1.2.6.2. System should provide community posts that users are registered and publicly posted contents of followed registered users.

1.2.7. Registered User

1.2.7.1. Registered users' profile page should have default profile picture and empty biography when a user have signed up.

1.2.8. Tagging

1.2.8.1 The semantics tagging shall be supported by wikidata.org

2. NON-FUNCTIONAL REQUIREMENTS

2.1. Security

2.1.1 User passwords shall be encrypted or hashed
2.1.2 The system shall use https Protocol

2.2. Performance

2.2.1. 2000 users shall be supported
2.2.2. 50 requests shall be responded in a second
2.2.3. 4 seconds shall be the maximum response time

2.3. Availability

2.3.1. The product shall work as Web application in Google Chrome
2.3.2. The product shall work as Android application in Android version 10 and later versions.
2.3.3. The system should be available 95% of the time.
2.3.4. In the case of failure, the system should recover in at most 2 hours.

2.4. Standarts

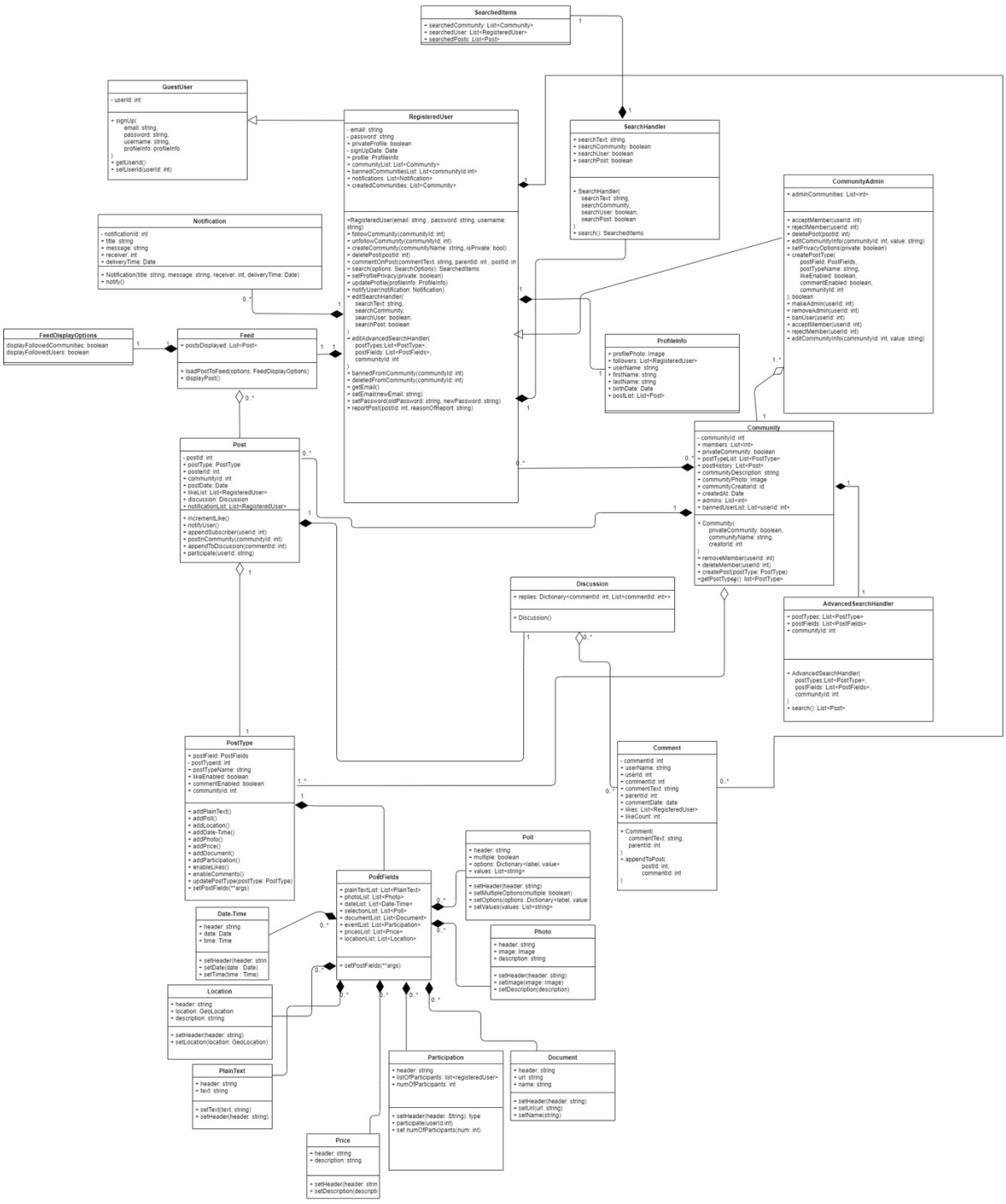
2.4.1. The system shall use the W3C activity stream standard 2.0 to implement subscriptions and notifications

2.5. Privacy

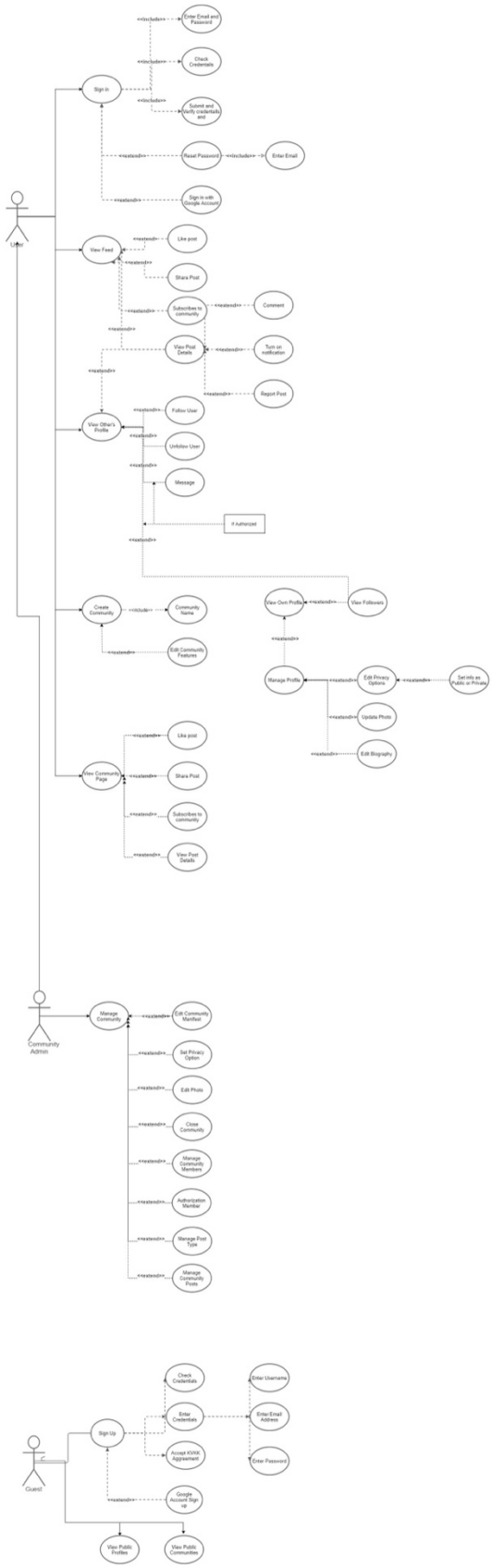
2.5.1. User data shall be protected with compliance to KVKK
2.5.2. Users shall accept privacy policy and user terms before they start using the product

10. SOFTWARE DESIGN DOCUMENTS

a) CLASS DIAGRAM

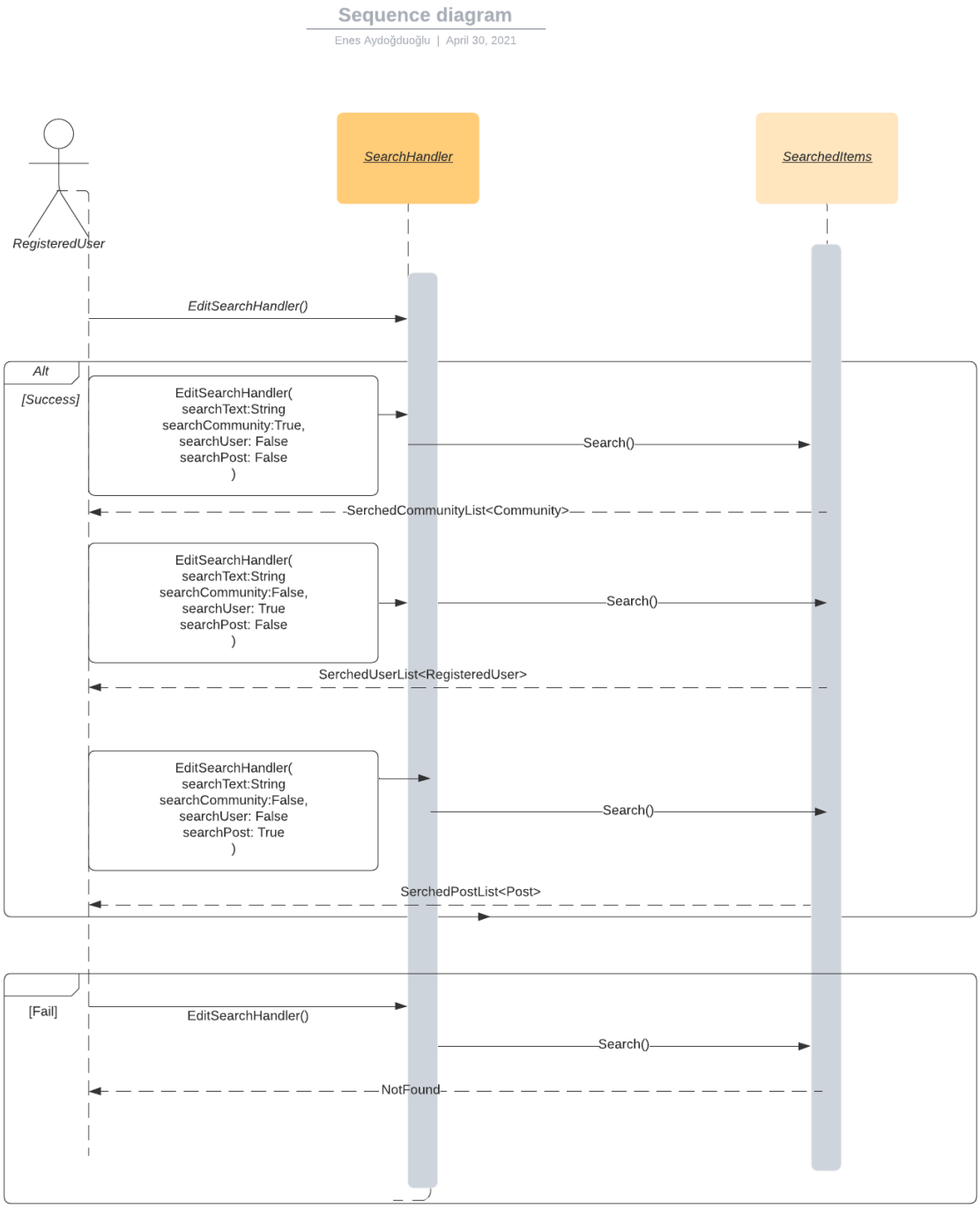


b) USE CASE DIAGRAM

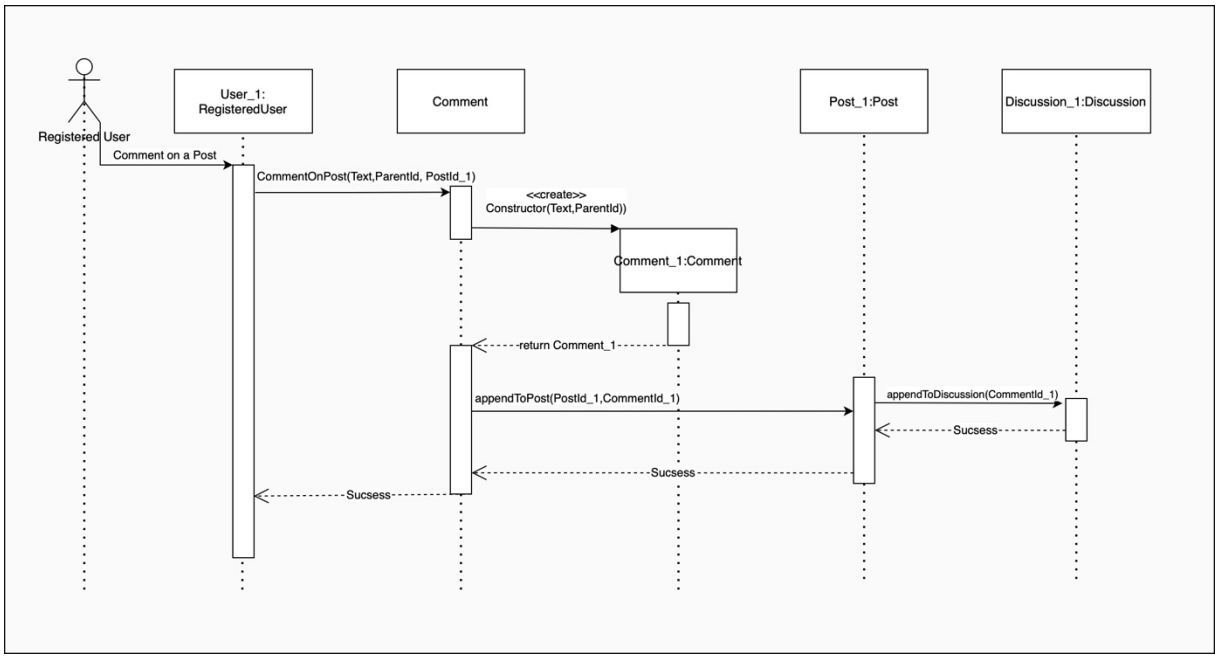


c) SEQUENCE DIAGRAMS

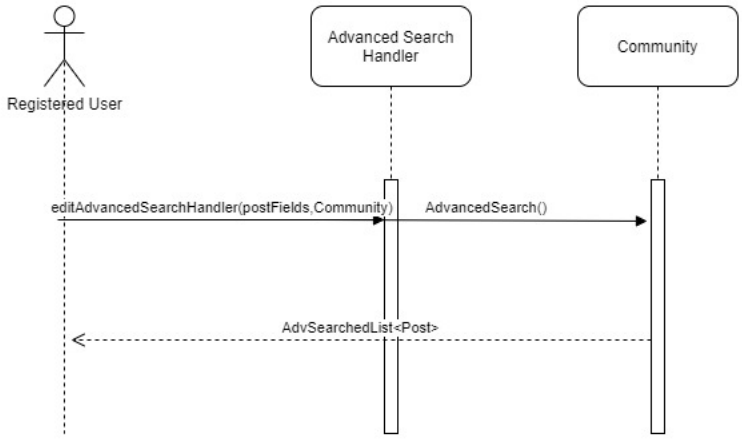
i. SEARCH



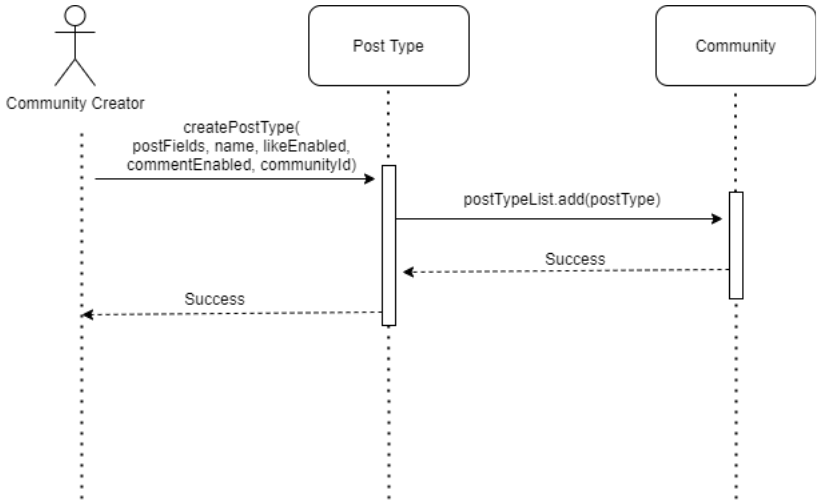
ii. COMMENT ON A POST



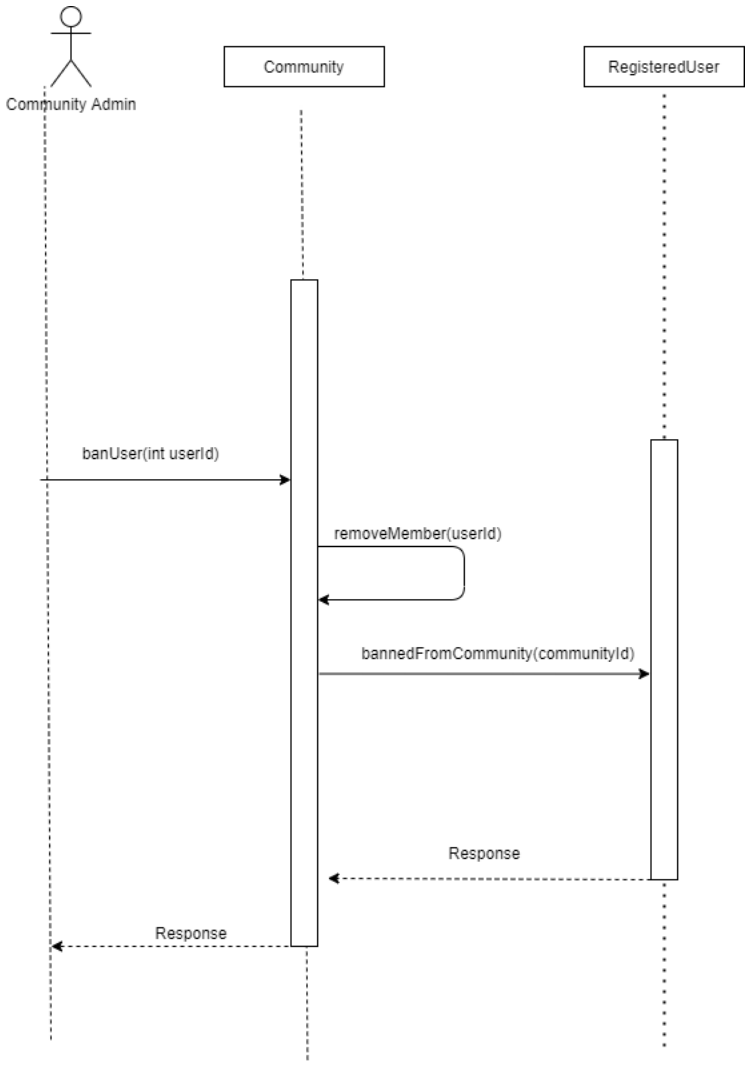
iii. ADVANCED SEARCH



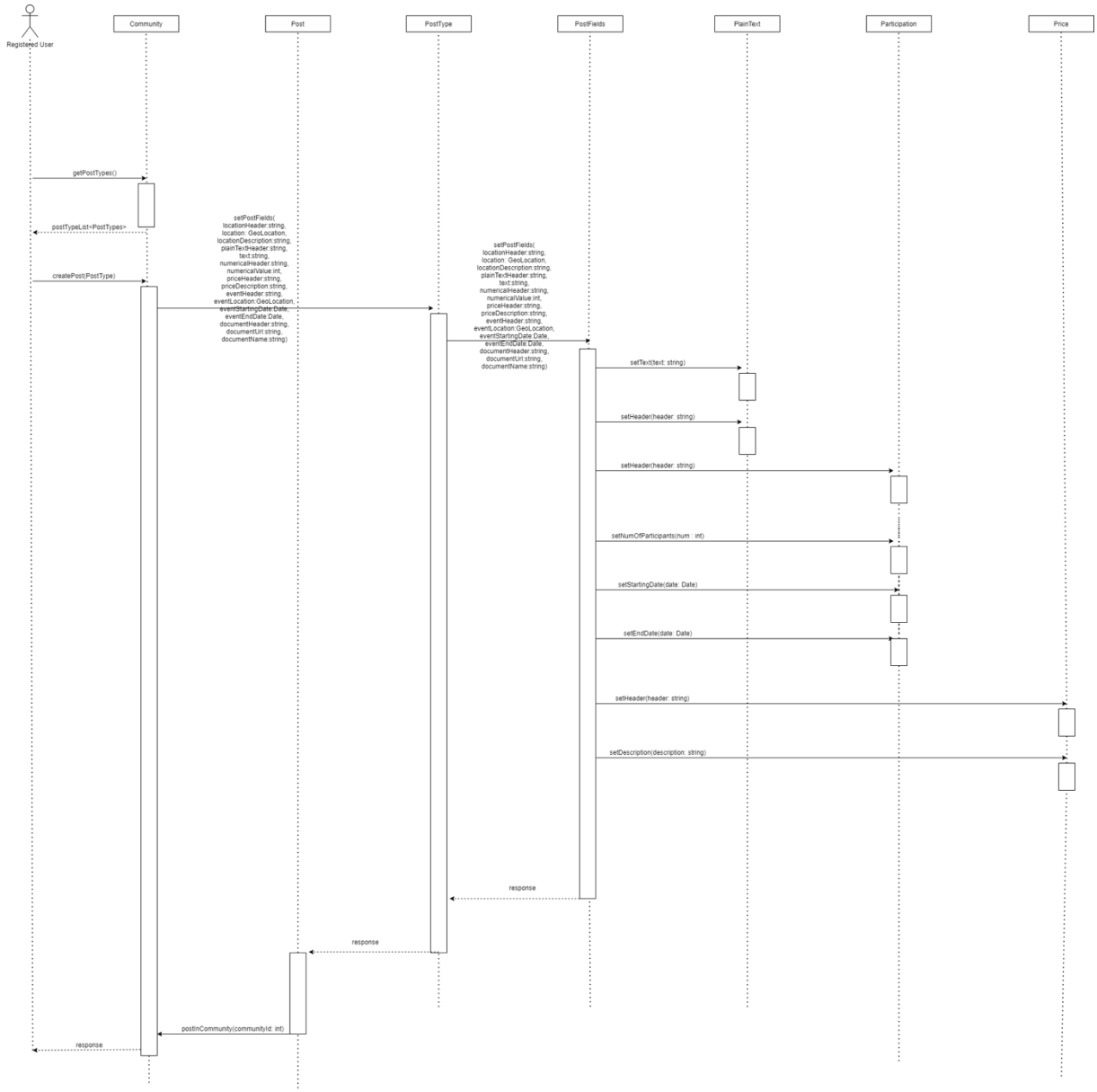
iv. CREATE POST TYPE



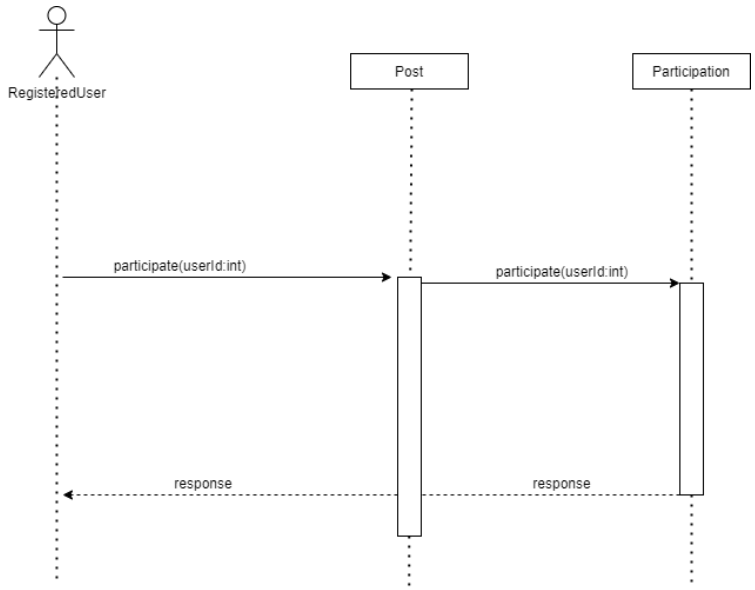
v. BLOCK A MEMBER FROM COMMUNITY



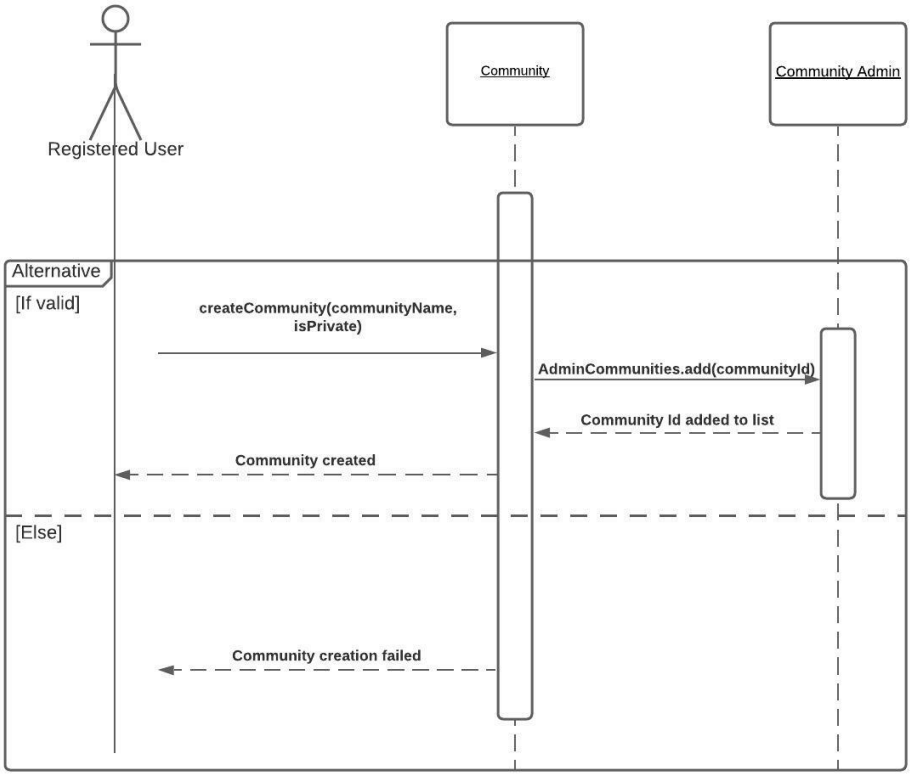
vi. CREATE A POST ACCORDING TO A POST TYPE



vii. INTERACT WITH A POST



viii. CREATE COMMUNITY



d) MOCKUPS

i. SCENARIO 1

- Persona

Ahmet Yıldız
22 years old
Student
Currently pursuing bachelor's degree
Part-time software developer
Interested in artificial intelligence
Crazy about hiking and camping
He loves listening to music while coding

- Story

Ahmet is a junior CMPE student at Bogazici University
He is a very hard-working and determined student.
Ahmet loves the department that he is studying and does everything he can to be a good computer engineer.
In addition to being a hard-working student, Ahmet is a very social person.
Ahmet constantly follows the activities of school clubs and loves participating in camping activities.
He uses an application that can follow college club's events.

- Preconditions

Ahmet has already signed up to the application.
Ahmet previously had subscribed to some communities.

- Acceptance Criteria

1.1.2.1. Registered users shall be able to sign in via username/password or e-mail/password.

1.1.4.2.1. Community members should be able to contribute to content in a Community within the predetermined ranges that Community Creator has ruled.

1.1.4.2.2. Community members shall be able to specify participation status to Events.

1.1.4.2.3. Community members shall be able to view Community Posts and Events.

1.1.4.2.4. Community members shall be able to view Community's other members.

1.1.4.2.5. Community members shall be able to turn on notifications of posts published in the community.

1.2.2.1. Users subscribe to the post shall be notified with the developments on that post.

1.2.4.1. Posts, shared by authorized members shall be displayed on the community timeline.

1.2.4.3. Posts shall have comment section.

1.2.6.1. System shall provide search bar, tabs and community posts.

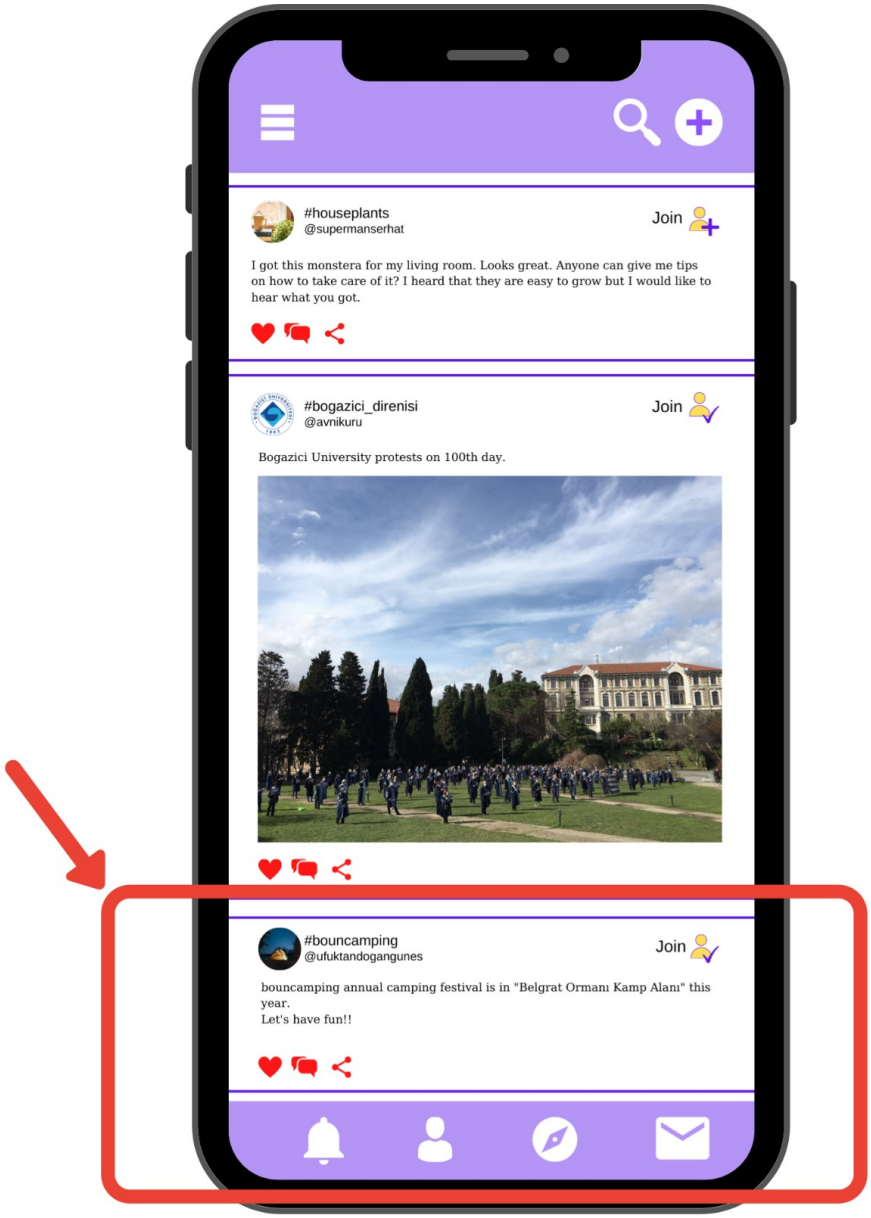
1.2.6.2. System should provide community posts that users are registered.

- Scenario with Mockups

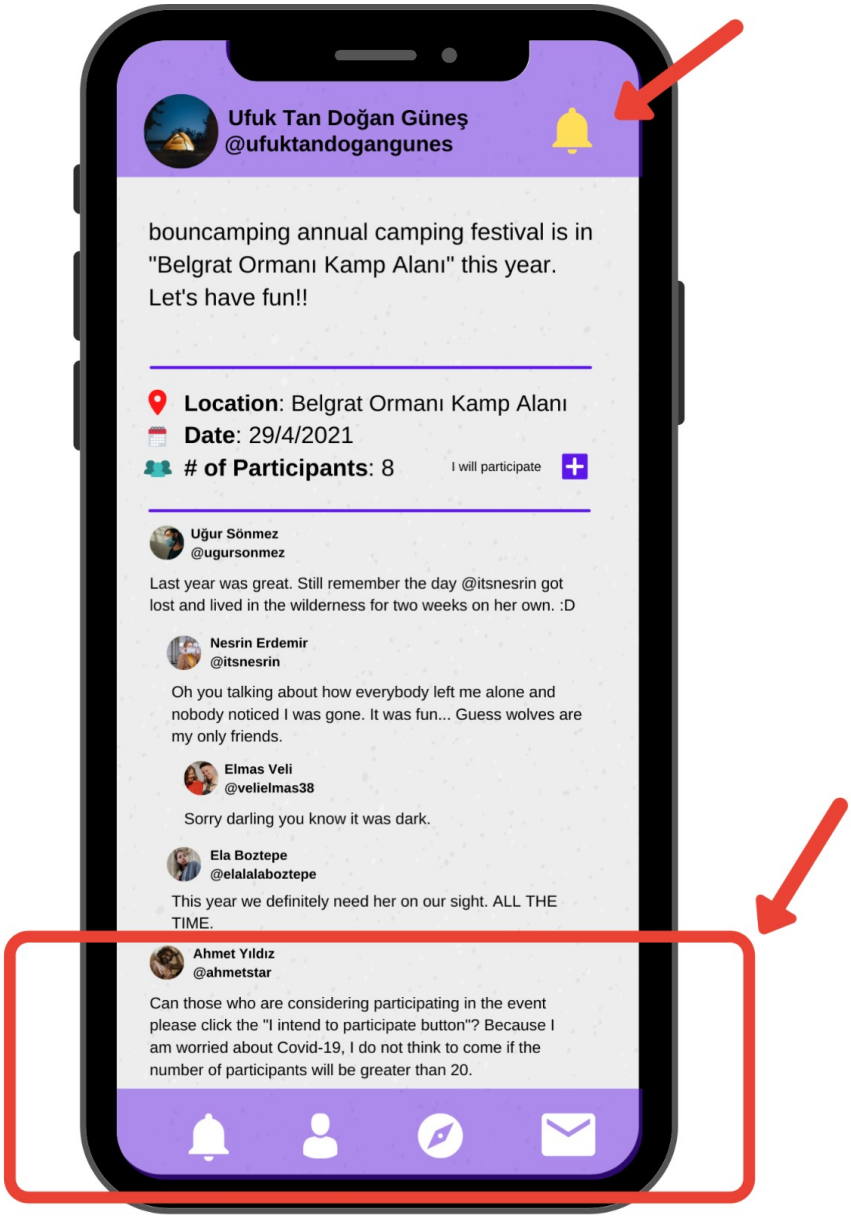
Ahmet got tired while studying and took a break.
He took his phone and opened the application.
He previously had registered to the application so, he entered his e-mail address and password and he signed into the application.
He saw #bouncamping's post on feed page. This post was about camping event and attracted his attention.
This post, whose type is determined by the creator, contains the button "I will participate".
Also this post has a counter showing the number of people who clicked the "I will participate button".
Ahmet clicked the "I will participate button".
"Can those who are considering participating in the event please click the "I will participate button"? Because I am worried about Covid-19, I do not think to come if the number of participants will be greater than 20." Ahmet left this comment under the post.
He turned on notifications of the post because he wanted to be notified about the number of participants.
When he re-entered the app after a while, he saw that there were some notifications.
When he clicked on the notification bar, he saw that 30 people click the "I will participate button".
Because he hesitates over covid-19, he gave up on participating in camping event.

1- Ahmet signed into app.

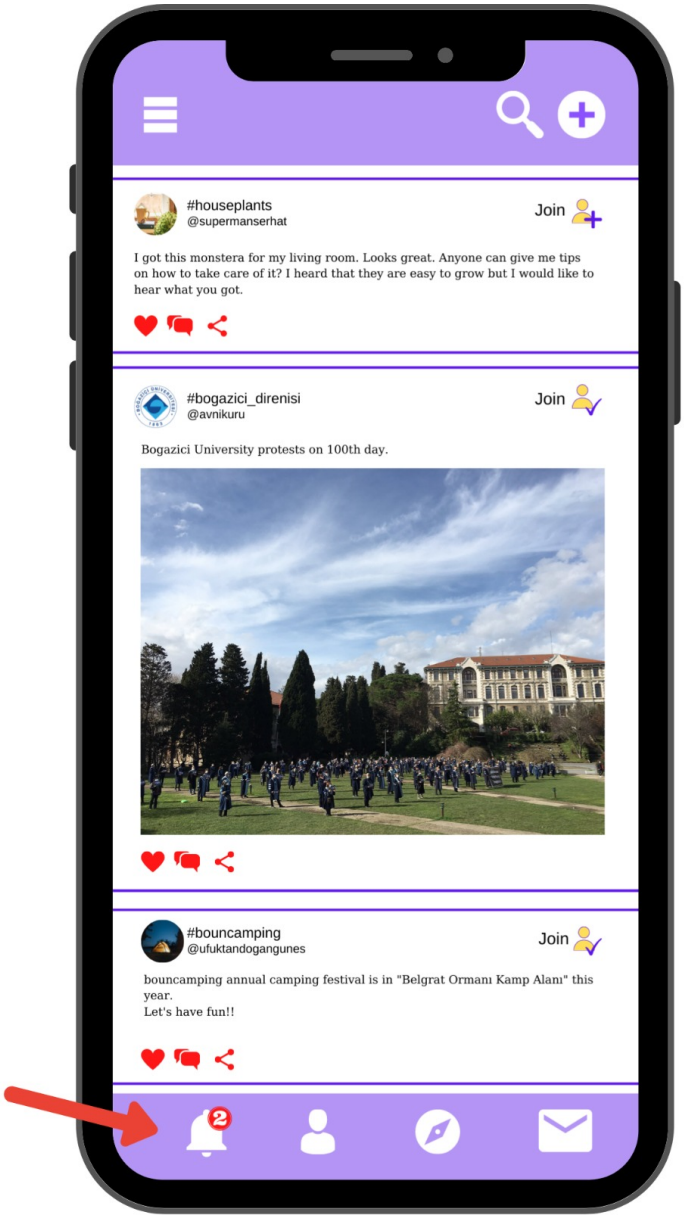
2- When signed into the application, the feed page appeared at first. #bouncamping's post took his attention.



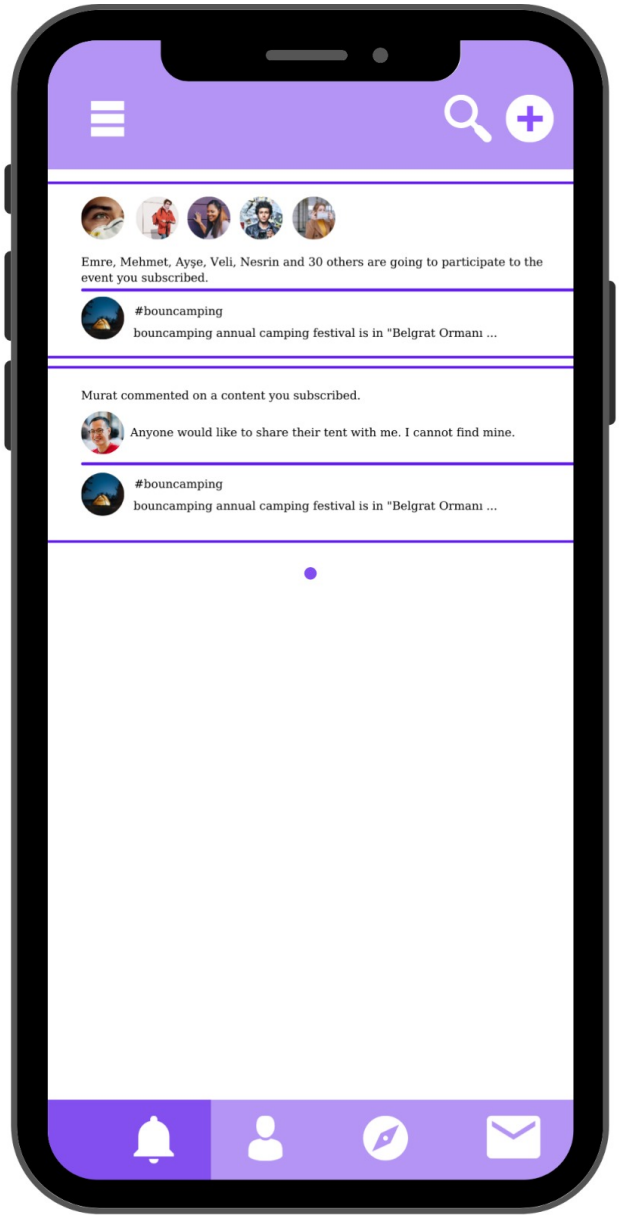
3- He clicked on the post to see the detailed content of the post. He clicked the participation button, left a comment under post and he subscribed to the post(clicked to the bell button).



4- When he re-entered the app after a while, he saw that there were some notifications (there was a mark on notification bar).



5- He clicked on the notification bar and saw app's notifications.


Emre, Mehmet, Ayşe, Veli, Nesrin and 30 others are going to participate to the event you subscribed.
#bouncamping
bouncamping annual camping festival is in "Belgrat Ormanı ...
Murat commented on a content you subscribed.
Anyone would like to share their tent with me. I cannot find mine.
#bouncamping
bouncamping annual camping festival is in "Belgrat Ormanı ...

ii. SCENARIO 2

- Persona

  Gökberk Yılmaz
  21 years old
  Event Organizer
  Currently pursuing bachelor's degree
  Leader of Compec in Bogazici university

- Story

  Gokberk Yılmaz is a computer engineering student in Bogazici University. Because he is interested in IT, he want to be a part of Compec. After 2 years of membership, he was elected as leader of the club. However many people are unaware of the club's meetings and events. Therefore he wants to solve this problem.

- Goals

  He wants to promote his club.
  Therefore he wants to create a public community for his club.

- Preconditions

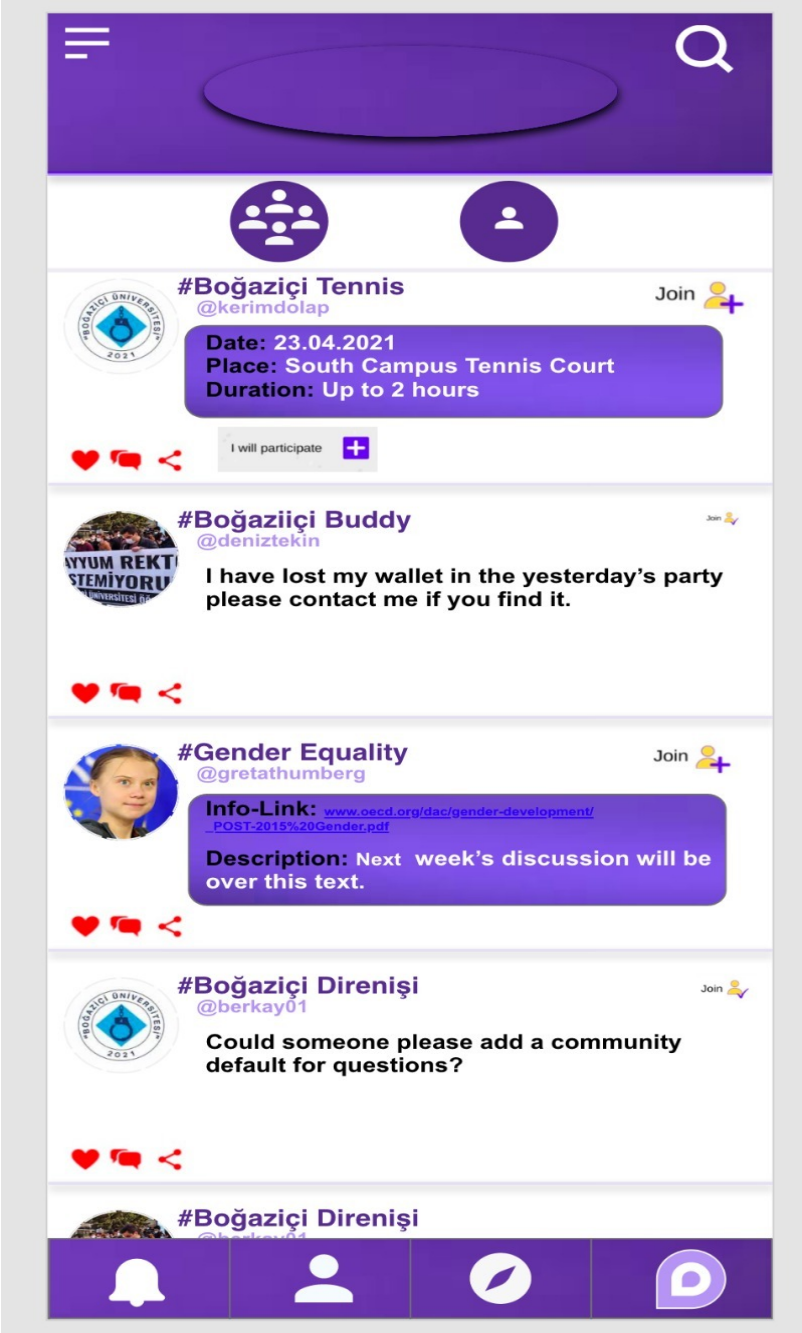
  Gökberk has already signed up to the application.
  Gökberk has already signed in to the application.
  Gökberk previously had subscribed to some communities.

- Acceptance Criteria

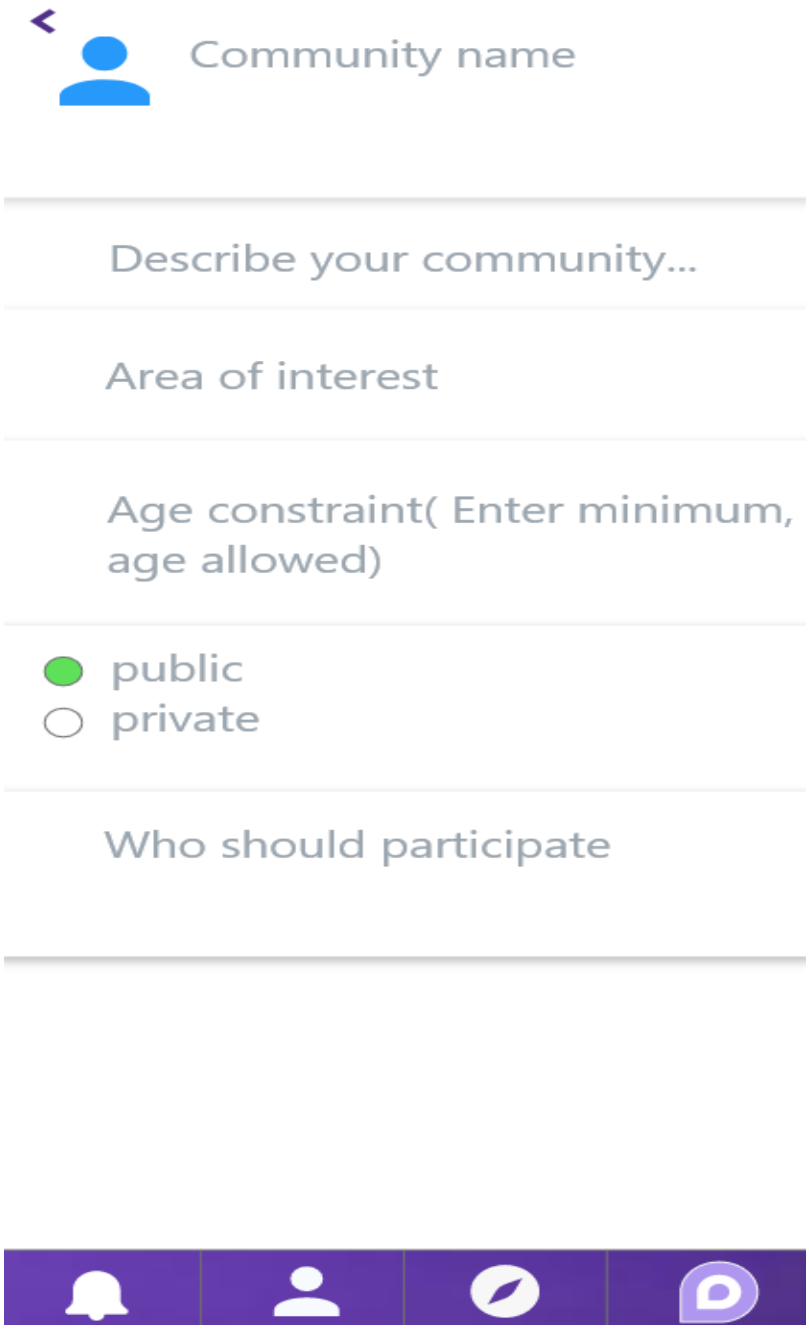
  1.1.2.1. Registered users shall be able to sign in via username/password or e-mail/password.
  1.1.4.3.4. Community admins shall be able to set privacy options of the Community.
  1.1.4.3.3. Community admins shall be able to give authorization to the Community members.
  1.1.4.3.2. Community Creator is the default Community Admin.
  1.1.4.3.8. Community admins shall be able to publish/edit the Community Manifest
  1.2.3.1. Users who accept the invitations shall re-authorized.
  1.2.4.2. Users shall be authorized according to the community creators decisions.

- Scenario with Mockups

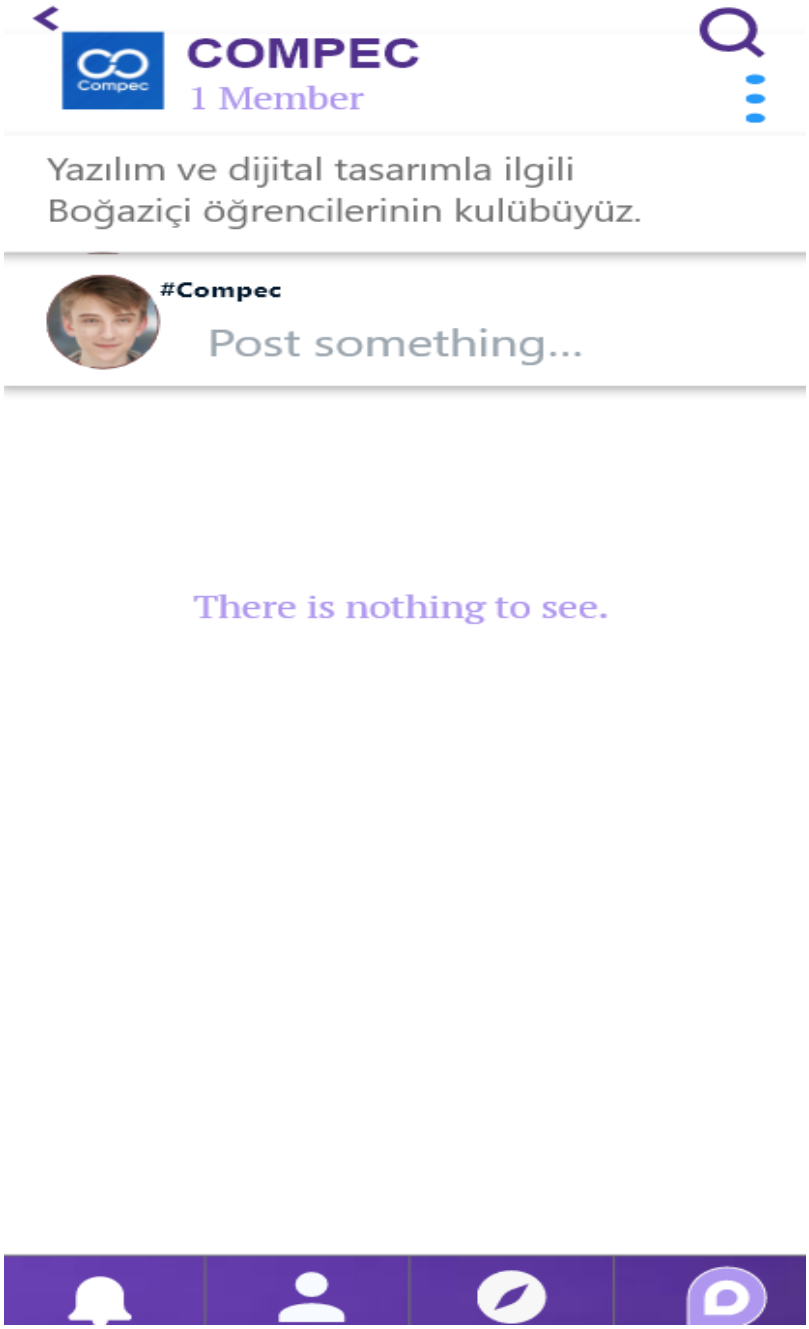
  1. He logs into his account and main page is opened.



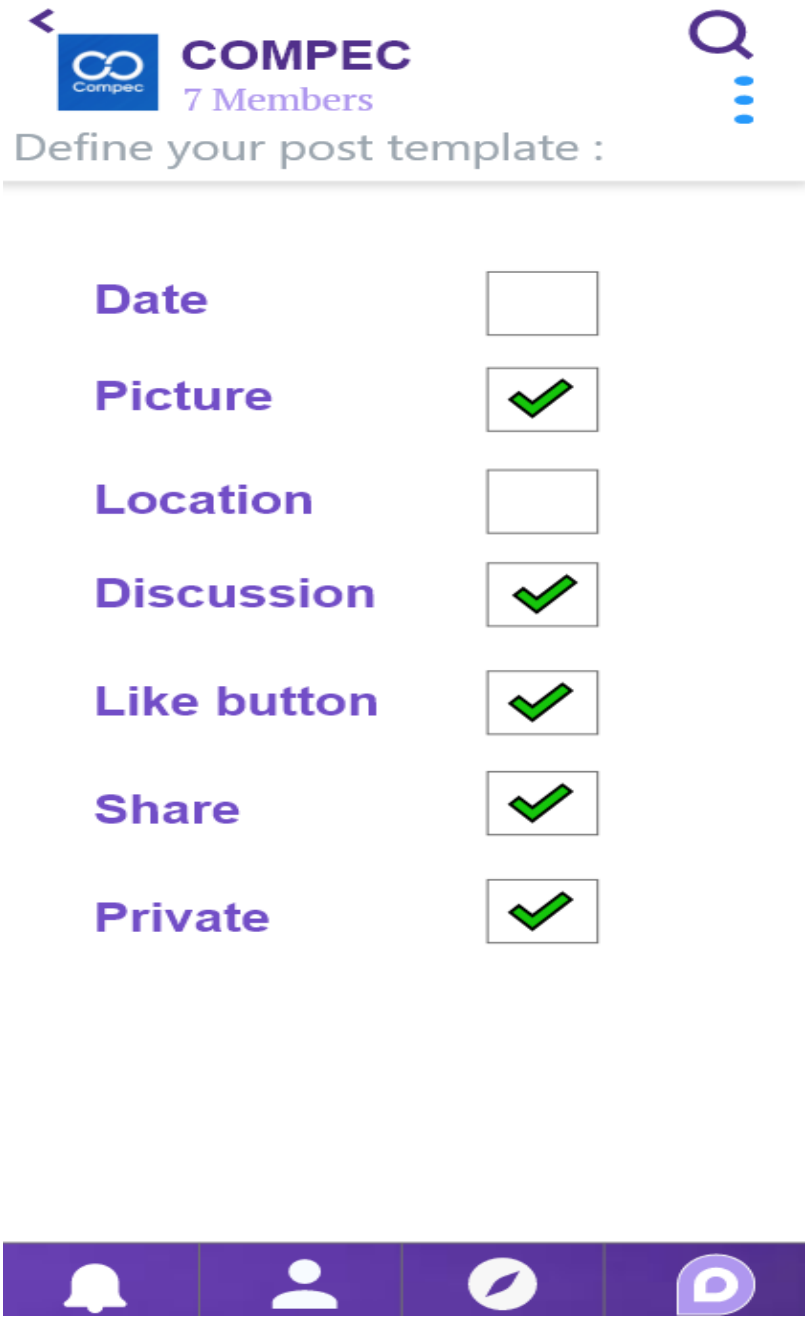
2. He clicks create a new community button and writes some necessary informations about community



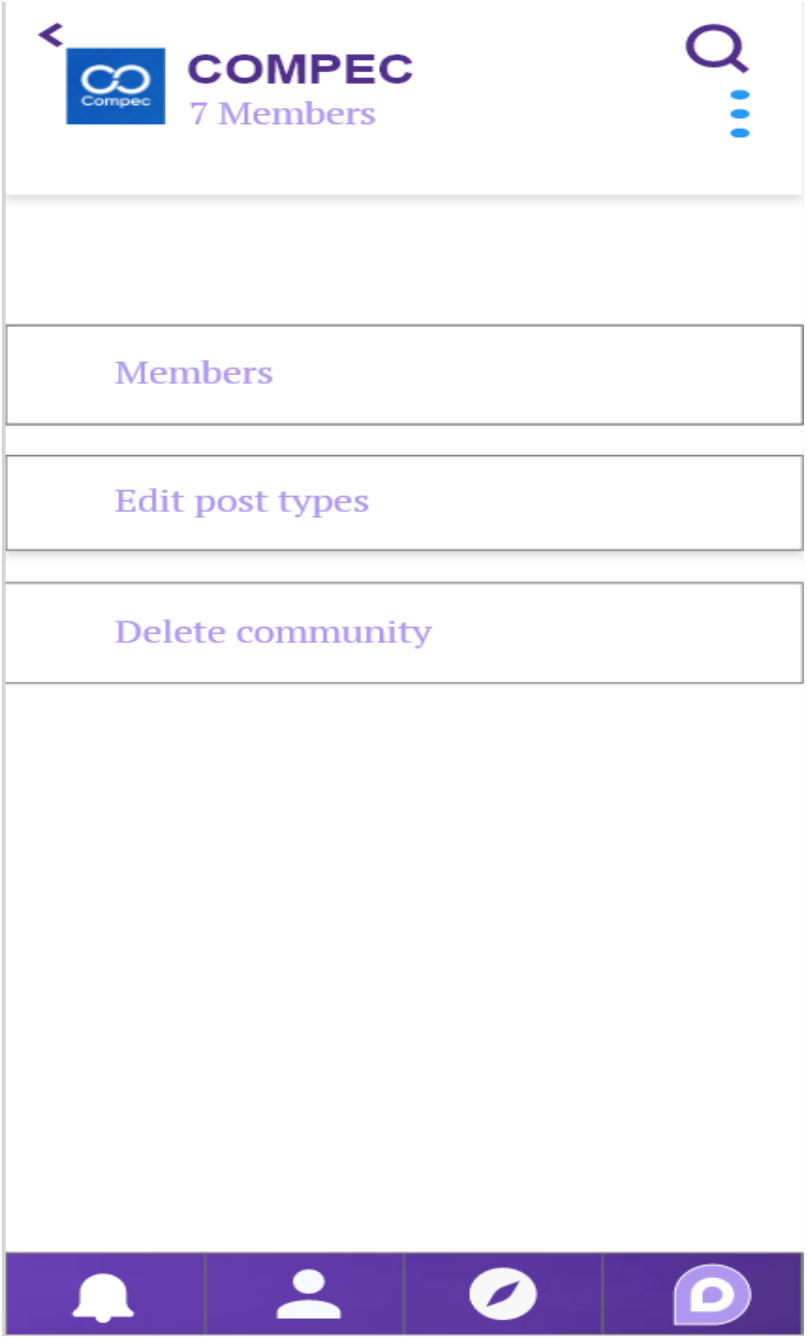
3. App navigates him to community page



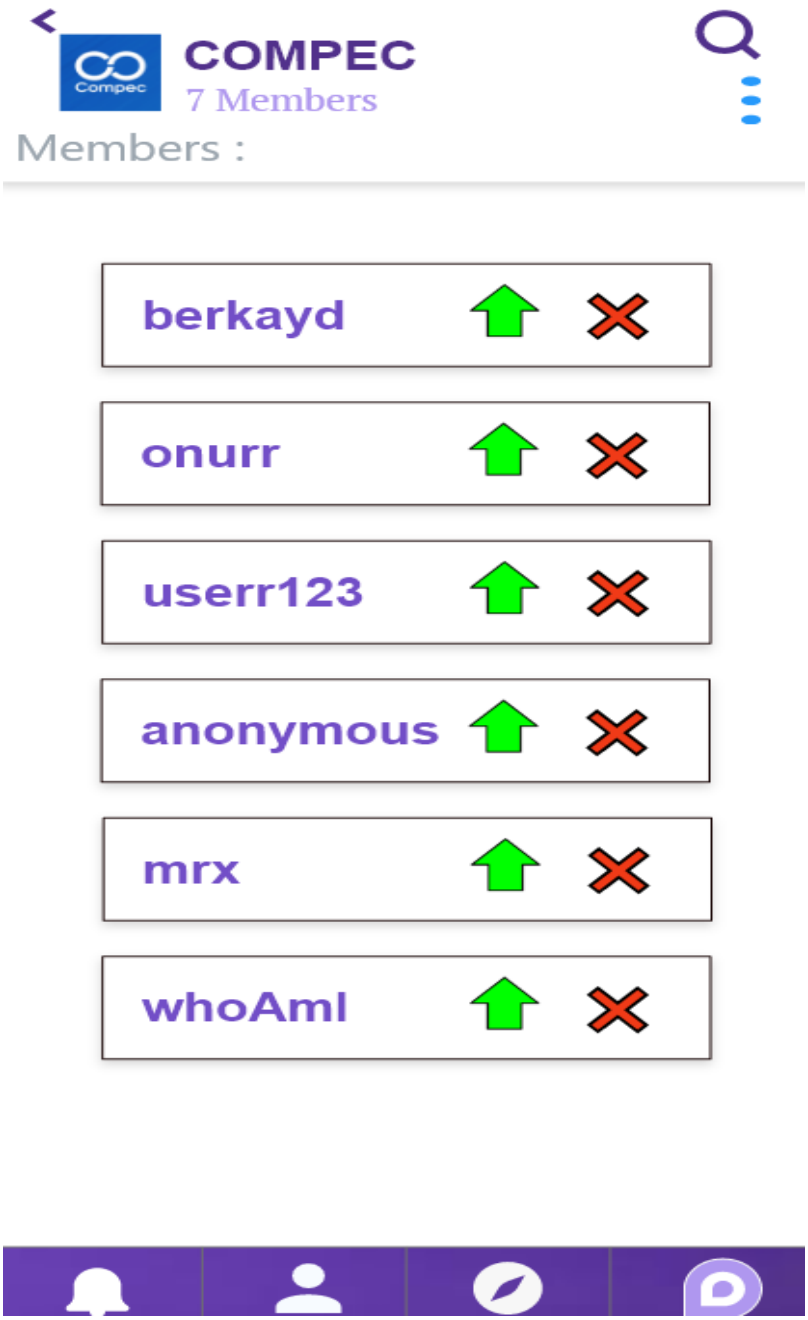
4. He decides that some features and information fields of the posts should always be filled. Therefore he opens "edit post type" section and marks the necessary fields.



5. After a few days, some new people join the community and Gökberk decides to promote " onurr " as moderator of the community. For that, he clicks " three dot ".



6. He opens " members " section and clicks the green arrow of " onurr ".


COMPEC
7 Members
Members :
berkayd
onurr
userr123
anonymous
mrx
whoAml

iii. SCENARIO 3

- Persona

Mabel Brooks
21 years old
Currently pursuing bachelor's degree in Birmingham
She is going to be in Bogazici University next semester as an Erasmus Student

- Story

Mabel Brooks is a Computer Science student in University of Birmingham.
She is very excited about her next semester in Bogazici.
In her last days in England, she wants to learn more about Bogazici, its courses and atmosphere.
She even became a member of Bogazici Community in the app in order to meet some people and follow some student clubs before coming here.

- Goals

She wants to get to know Bogazici and discover new student clubs.
She wants to find her department's community so that she can find answers to any question she may have about the department and courses.

- Preconditions

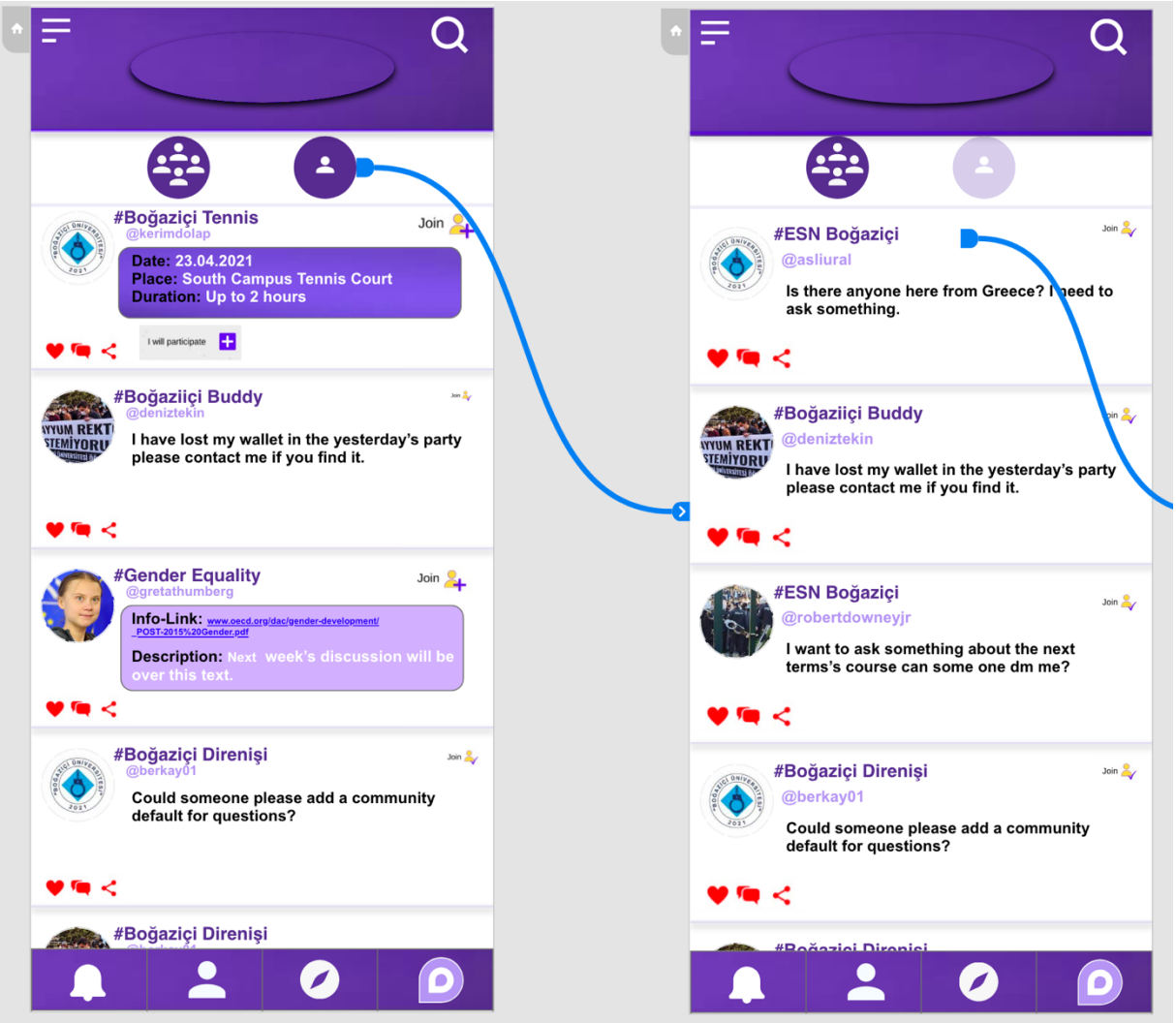
Mabel has already signed up to the application.
Mabel has already logged in to the application.
Mabel previously had subscribed to Bogazici ESN, Boğaziçi Buddy and Boğaziçi Direnişi Communities.

- Acceptance Criteria

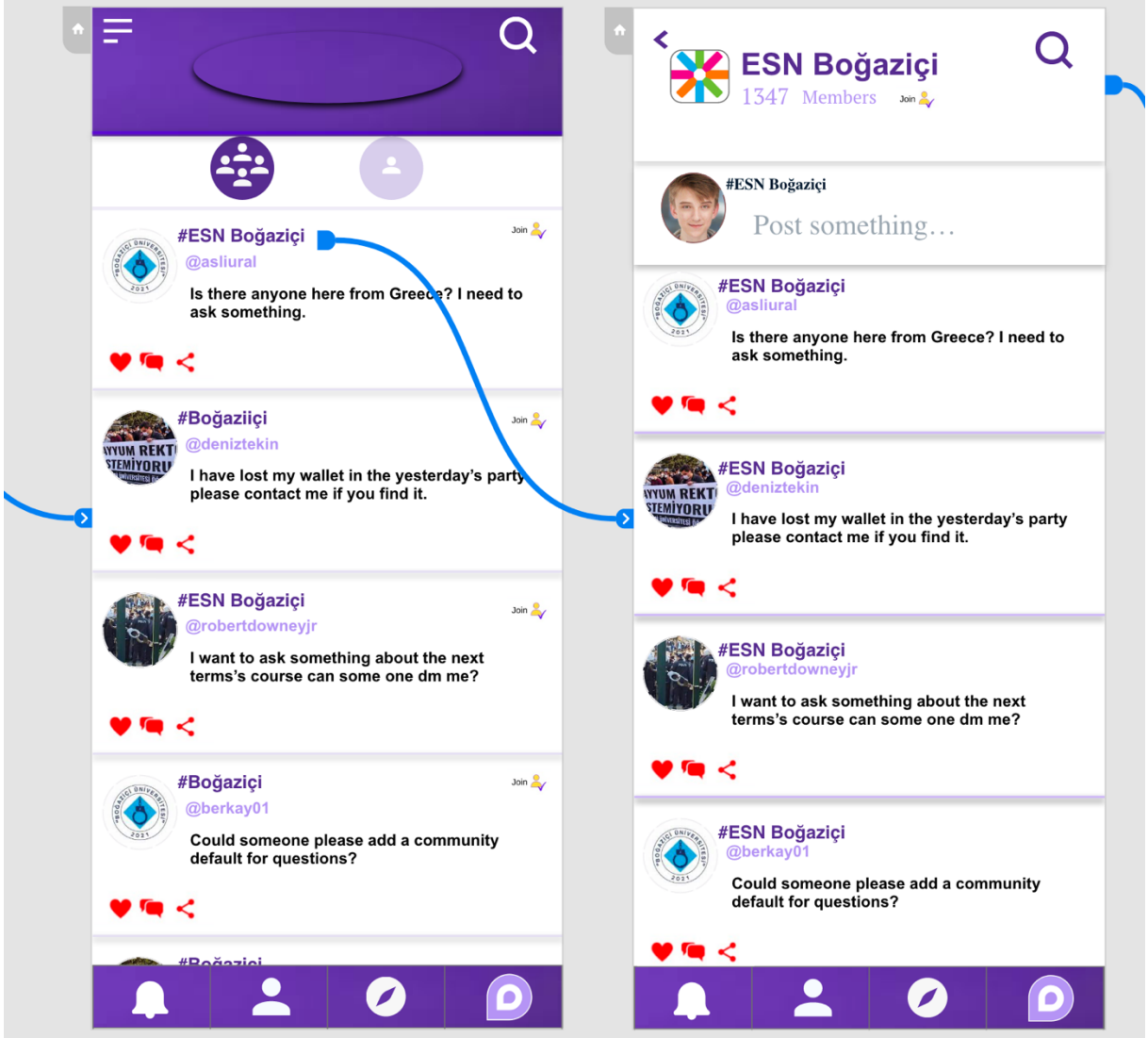
1.1.4.2.1. Community members should be able to contribute to content in a Community within the predetermined ranges that Community Creator has ruled.
1.1.4.2.3. Community members shall be able to view Community Posts and Events.
1.2.1.1. System shall be able to recommend content and persons of interest.
1.2.3.3. According to the community privacy options users having the right to see the posts shall be authorized accordingly.
1.2.4.1. Posts, shared by authorized members shall be displayed on the community timeline.
1.2.4.3. Posts shall have comment section.
1.2.5.1. Upon searching related people and posts shall be indexed.
1.2.5.2. Upon searching, search history will be displayed.
1.2.6.1. System shall provide search bar, tabs and community posts.
1.2.6.2. System should provide community posts that users are registered.

- <u>Scenario with Mockups</u>

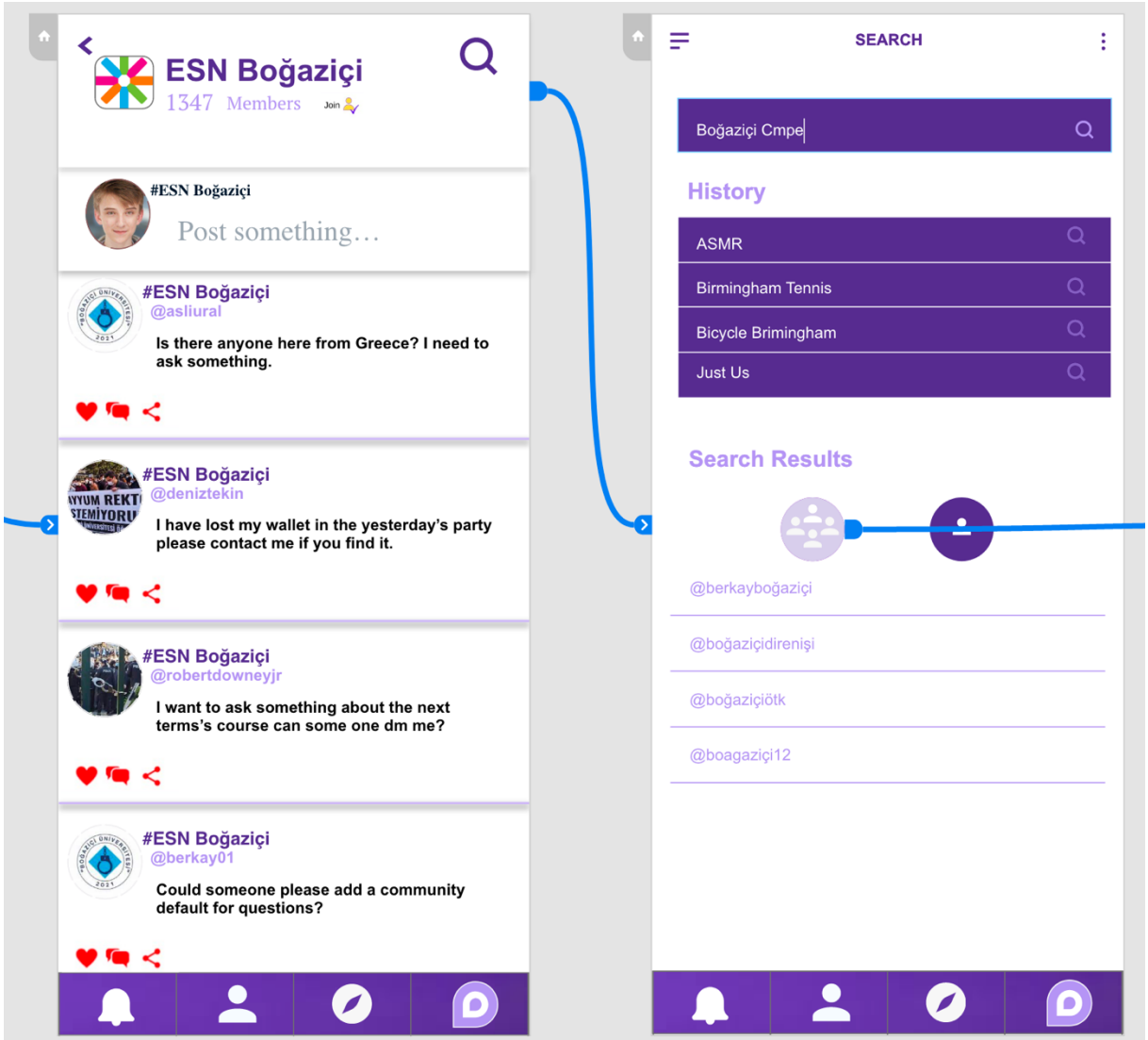
1. She opened the app and scrolled down on the feed. She only wanted to see community posts, so she chose community filtering option.



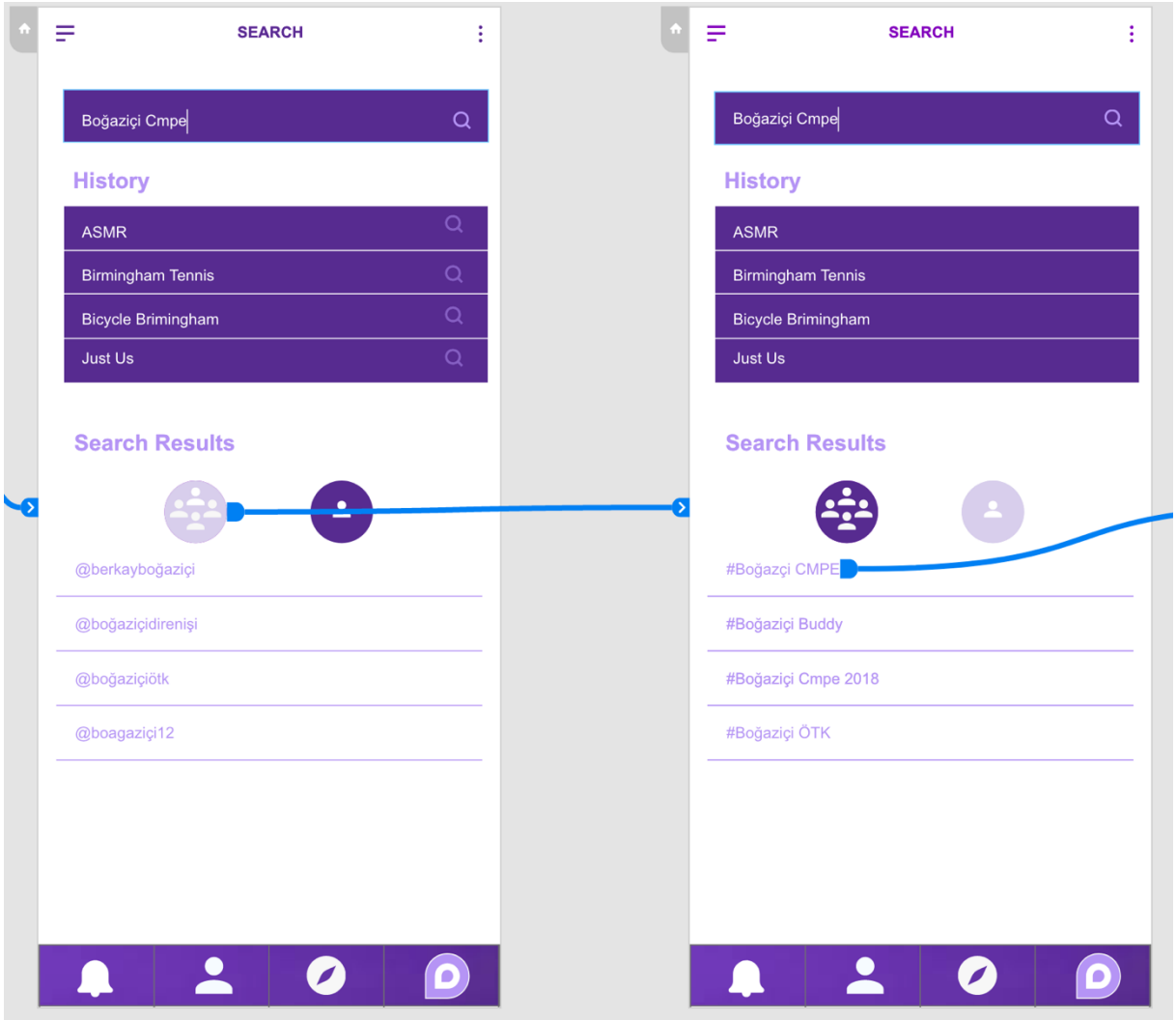
2. She saw #ESN Bogazici post on the feed and went to community's page by clicking on the community name above the post.



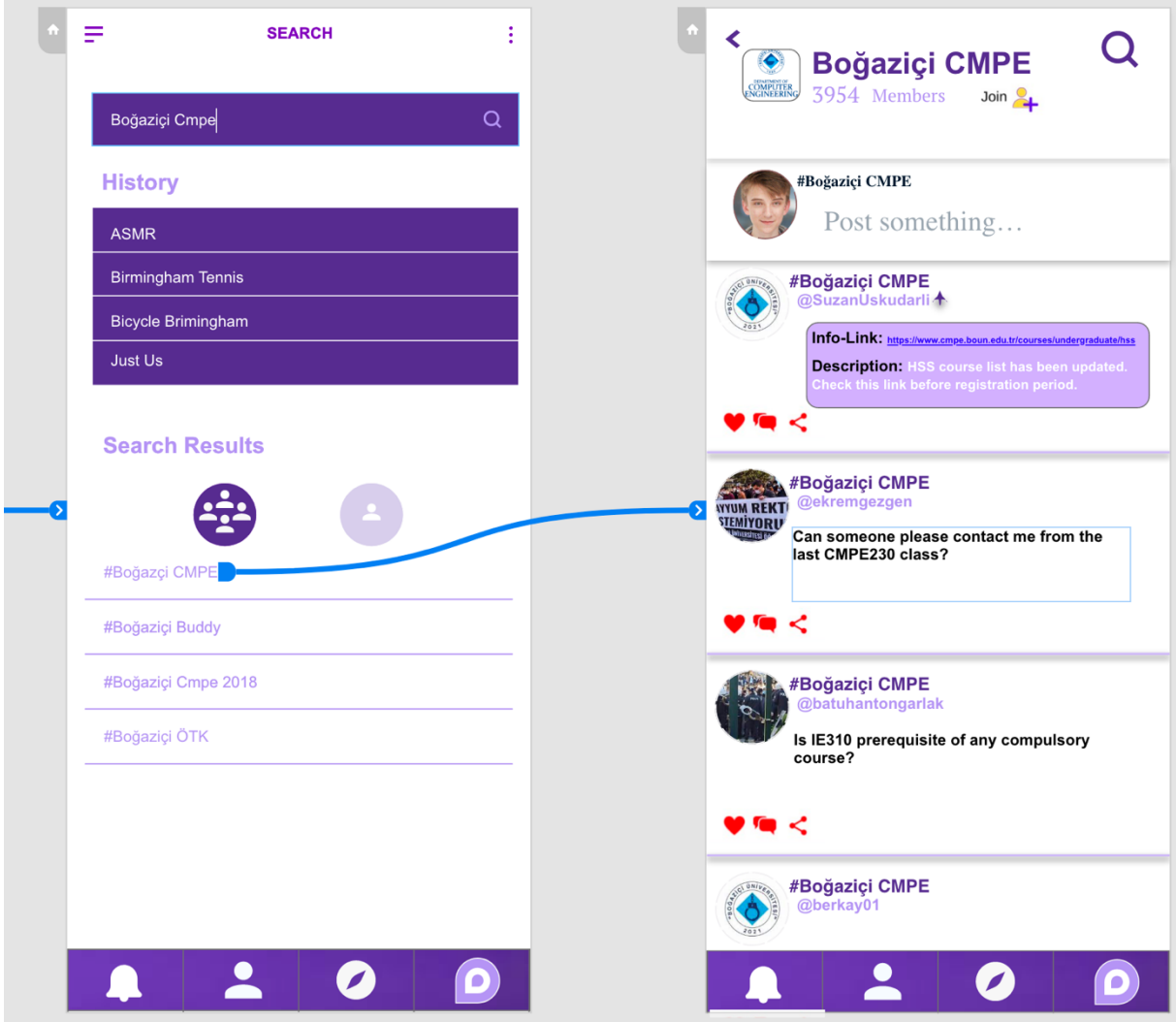
3. She wanted to find someone in her department but there was not any post related to Cmpe or posted by someone in the department in ESN Community. In order to find the department's community, she clicked on the search button and typed 'Bogazici Cmpe' in the search box.



4. At first, user profiles appeared in the search page as default. Below the search history, she saw the filtering option just as on the feed for community and user profiles.



5. She chose community filter and found Bogazici Cmpe Community.



iv. SCENARIO 4

- <u>Persona</u>

  Maria Sharapova
  She is 34 years old.
  She is a professional tennis player.

- <u>Story</u>

  She came to Antalya for a tournament.
  After the tournament, she wants to stay there more and organize tennis events with talented young people who are interested in tennis.
  Since she dreams of settling in this city and become a tennis coach after her retirement, she wants to meet new people through these events.

- Goals
  She wants to find new people to play tennis with by creating new community post type, tennis event.

- Preconditions

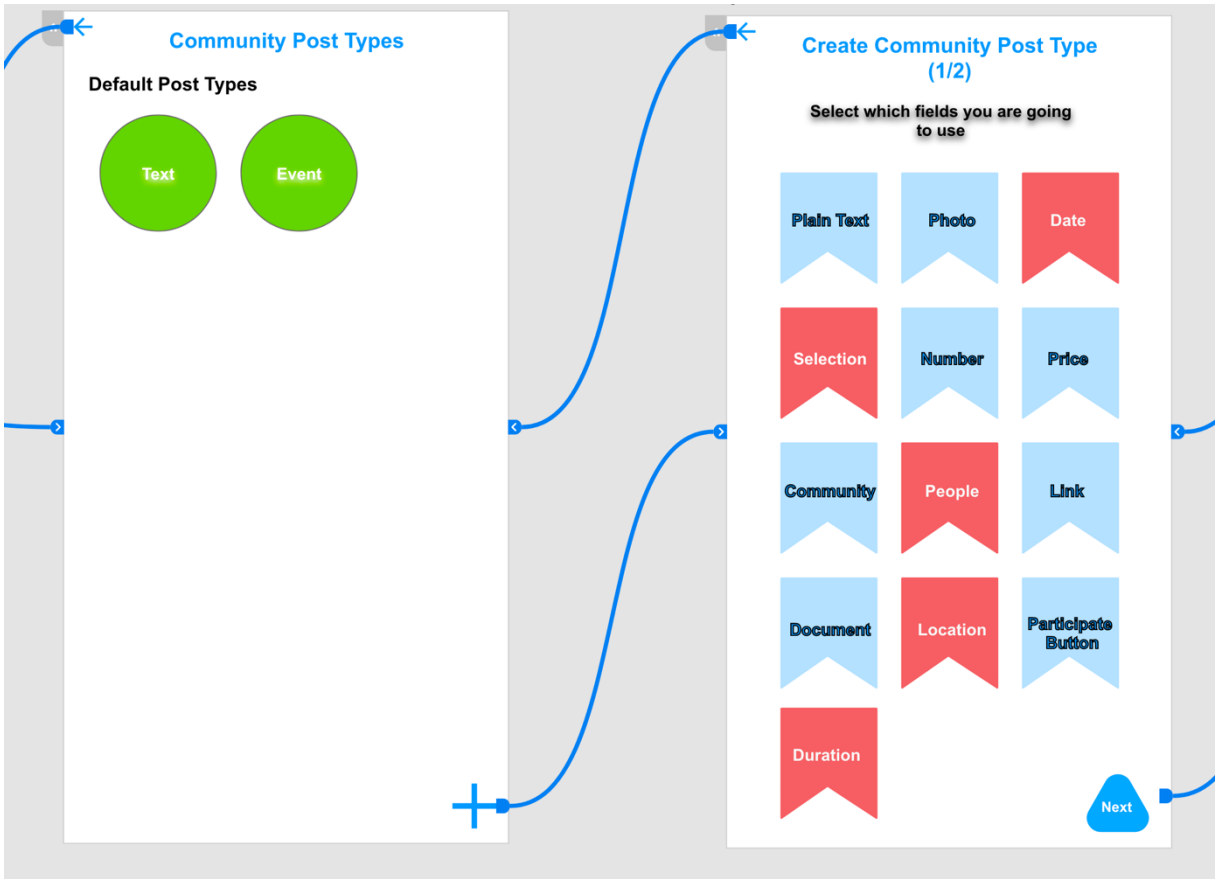
  She has already signed up to the application.
  She has already logged in to the application.
  She is the admin of Antalya Tennis Community.
  She has clicked settings of post types of Antalya Tennis Community.

- Acceptance Criteria

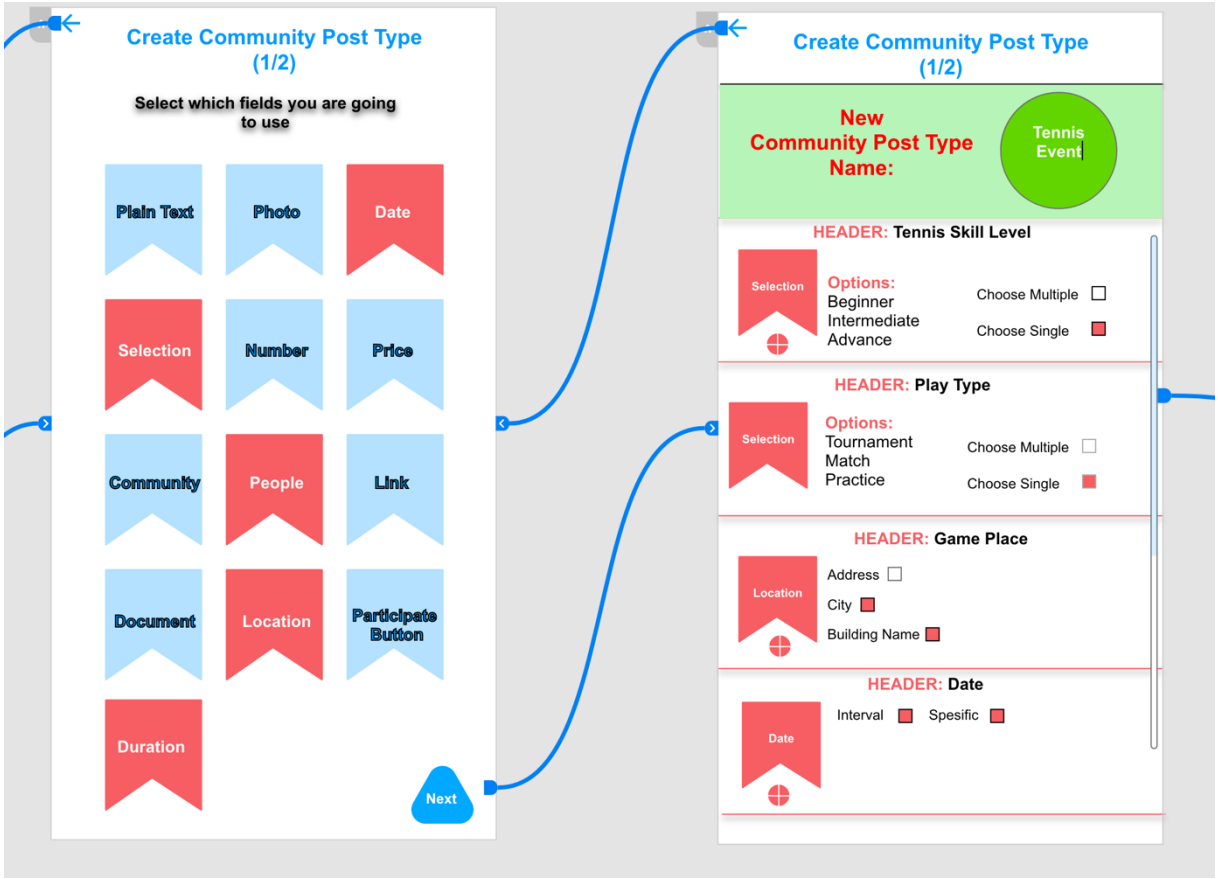
  1.1.2.1. Registered users shall be able to sign in via username/password or e-mail/password.
  1.1.4.3.10. Community admins shall be able to create Community Post Types for the purpose of the Community.

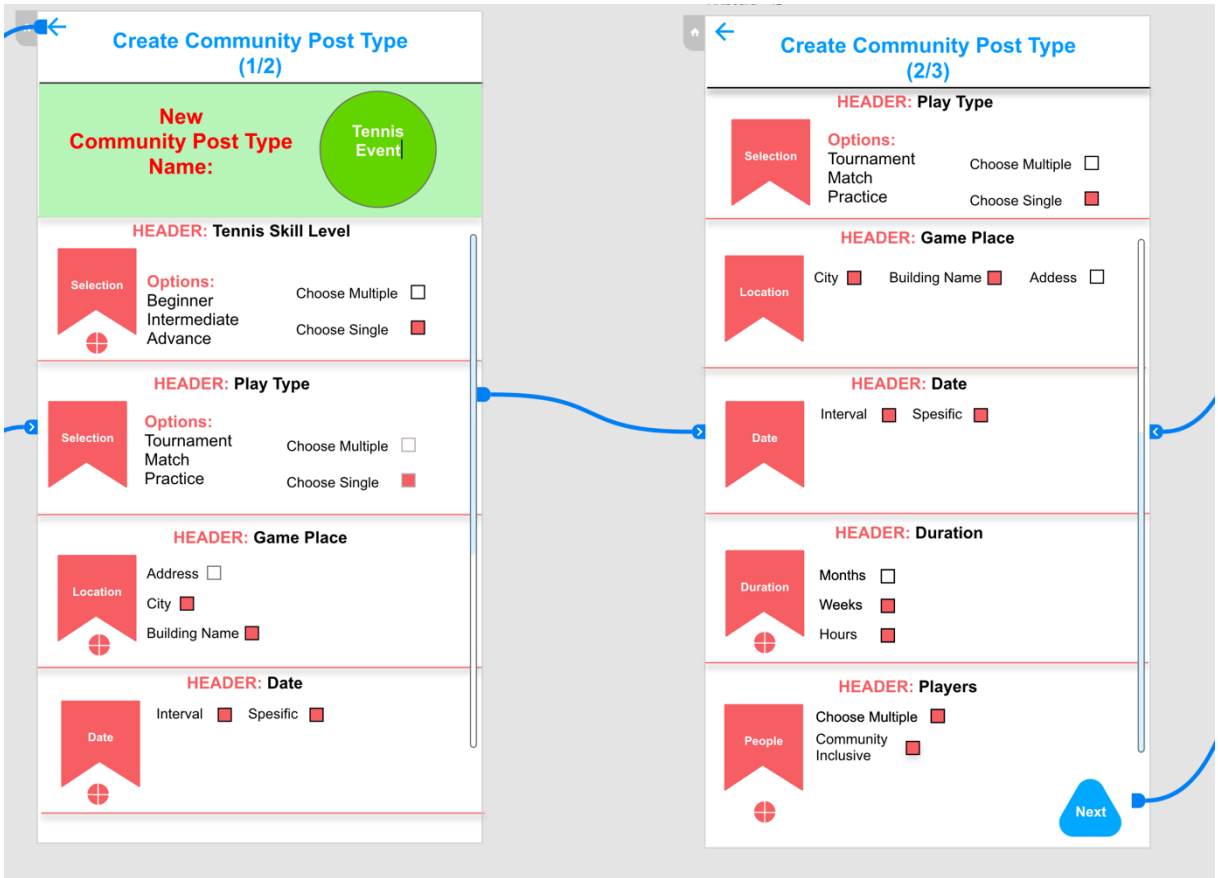
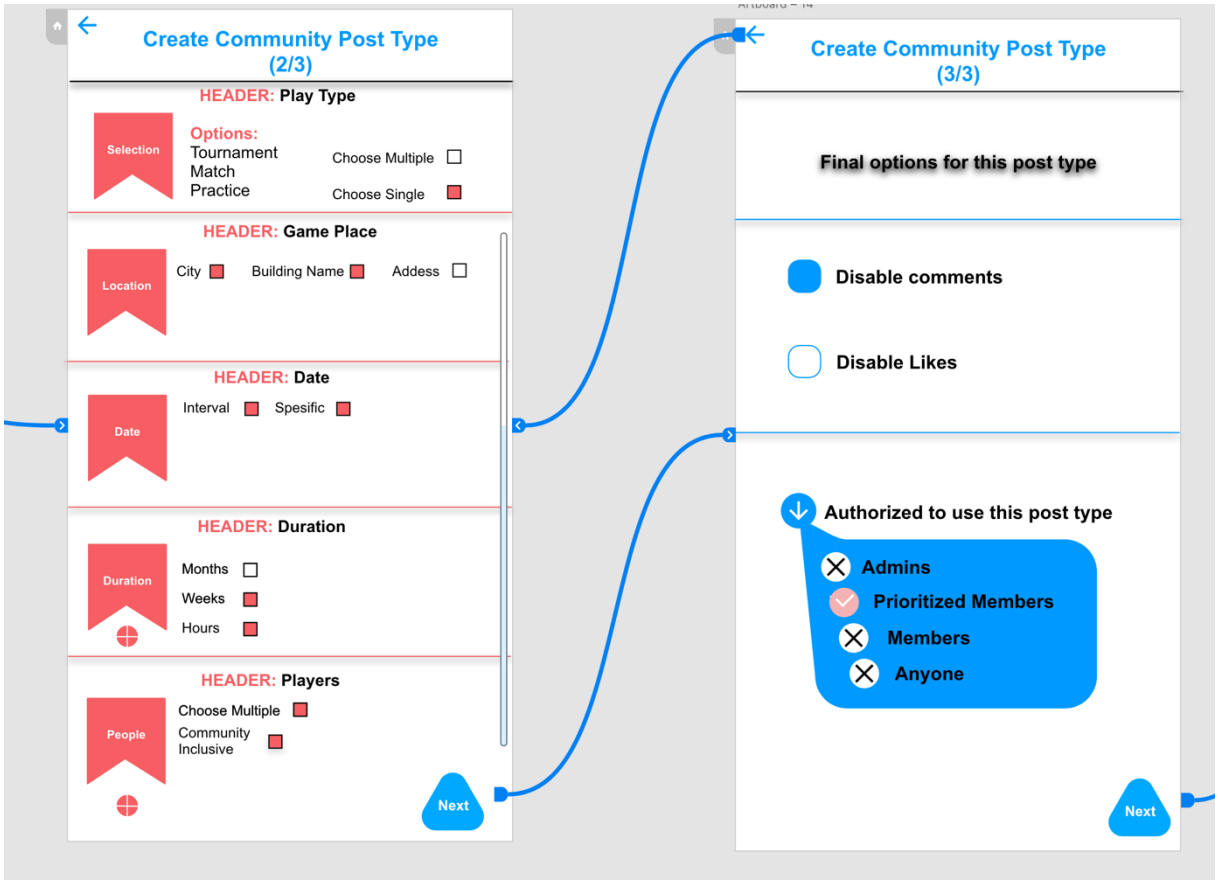
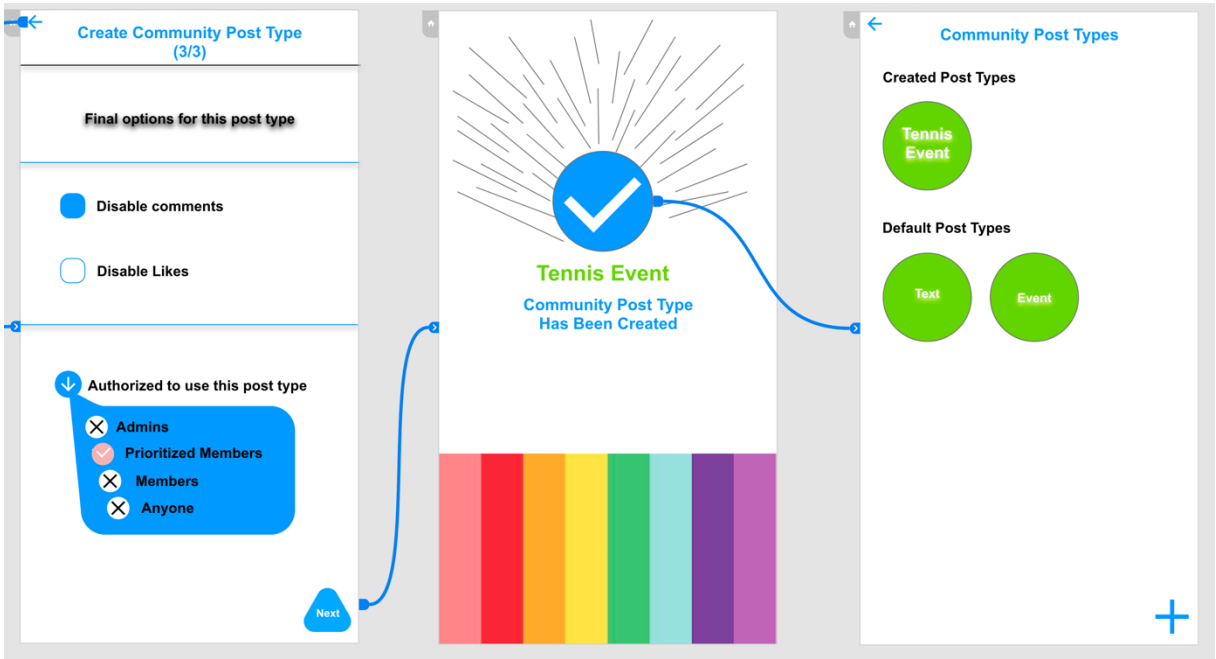
- Scenario with Mockups

  1. She is on the community post types page in the Antalya Tennis Community. She clicks "+" button on the bottom right and the button navigates her to create post type page (1/2). On this page, there are default options to select. She selects the features that she wants it to be in the post type she will create.



2. On the page (2/2), she customizes the features she has selected.


Create Community Post Type
(1/2)
Select which fields you are going to use
Plain Text
Photo
Date
Selection
Number
Price
Community
People
Link
Document
Location
Participate Button
Duration
Next
Create Community Post Type
(1/2)
New Community Post Type Name:
Tennis Event
HEADER: Tennis Skill Level
Selection
Options:
Beginner
Intermediate
Advance
Choose Multiple
Choose Single
HEADER: Play Type
Selection
Options:
Tournament
Match
Practice
Choose Multiple
Choose Single
HEADER: Game Place
Location
Address
City
Building Name
HEADER: Date
Date
Interval
Spesific


Create Community Post Type
(1/2)
New Community Post Type Name:
Tennis Event
HEADER: Tennis Skill Level
Selection
Options:
Beginner
Intermediate
Advance
Choose Multiple
Choose Single
HEADER: Play Type
Selection
Options:
Tournament
Match
Practice
Choose Multiple
Choose Single
HEADER: Game Place
Location
Address
City
Building Name
HEADER: Date
Date
Interval
Spesific
Create Community Post Type
(2/3)
HEADER: Play Type
Selection
Options:
Tournament
Match
Practice
Choose Multiple
Choose Single
HEADER: Game Place
Location
City
Building Name
Addess
HEADER: Date
Date
Interval
Spesific
HEADER: Duration
Duration
Months
Weeks
Hours
HEADER: Players
People
Choose Multiple
Community Inclusive
Next

3. She clicks "OK" button on the bottom right and the button navigates her to community post types page. There are 3 different post types, 2 are under default post types, and the one she has created, "tennis event", is under created post types.