Cmpe 451 - Milestone 2

Group 10

Berkay Demirtaş

Ekin Güngör

Enes Aydoğduoğlu

Gülşah Çilingiroğlu

Muhammet Ekrem Gezgen

Onur Sefa Özçıbık

S. Gökberk Yılmaz

Safa Burak Altunel

Taha Kalkanlı

CONTENTS

EXECUTIVE SUMMARY

At the second customer meeting, we presented some improvements we made in our project based on the feedback we received in the first customer meeting and our project plan. Addition to sign in, sign up, editing profile page functionalities, we presented community page, post creation, post type creation, and post page functionalities at the demo.

During the presentation, since there is an unresolved problem about the connection of frontend to backend, we used mock data. Future plan is to resolve this problem and fully represent the functionalities we have implemented and requirements that we have planned to implement at the end of the semester.

During the preperation of second milestone, we achieved our goal to work with API documentation provided by our backend team and front-end and mobile team to connect the features they developed based on this documentation to the backend system except the bugs occured during this connection.

As a next step of implementation of Purbee, we are planning to implement comment section which will be present on the posts, direct message section to create a communication environment between users, and search section to provide the functionality of searching a community and an user by their names. Addition to these functionalities, feed page implementation will be completed and posts that we have created will be visible at the last customer meeting. After the implementation of these functionalities, we will achieve our aim to provide the necessary functionalities described our project plan.

SUMMARY OF WORK

Name	Work done
Safa Burak Altunel	Implemented community creation page, comminity post type creation page in android Added bottom navigation bar and navigation drawer in android
Ekin Güngör	Implemented the post and post_type endpoints. Implemented the classes Post, PostType, PostFields and Fields. Implemented the database utilities of the classes above. Tested and debugged the implementations of the classes above. Presented in the second customer meeting. Contributed to the second milestone by creating the requirements part. Created the initial structure of the class Community.

Gülşah Çilingiroğlu	Attended lessons, Zoom and Discord meetings. Contributed to milestone 2 report with executive summary. Created and finalized the design of milestone 2 report. Opened some issues based on planning and tasks. Reviewed pull request.
Taha Kalkanli	Implemented a React component where community admins can create new templates for posts that will be shared by users. Implemented another React component where users can share post according to the post types created by community admins. Deployed the React project to AWS. Reviewed code. Kept learning React.js while implementing the project. Fixed a bug that allowed users to view and fill the necessary fields only when the page is zoomed out. Implemented the map functionality with Google Maps API.
Onur Sefa Ozcibik	Implemented the community_page endpoints. Created Community class. Created a proper database collection. Wrote code to handle database operations regarding Community class and its endpoints. Fixed CORS policy error. Created and updated documentations related to the endpoints. Contributed to the second group milestone report via publishing the latest version of the API documentation. Prepared postman collection. Tested the endpoints to ensure giving correct functionality. If there were any problems, debugged them.
Berkay Demirtas	Implementation of post endpoint, Implementation of post_type endpoint. Implementation of database methods for post and post creation process. Implementation of post,postType, postFields and all the classes under fields.py .Creating initial structure of community class.Updating database and creating necessary collections for implementation.Tested the endpoints and debugged the classes implemented for post and post type creation.Writing mapping functionality part of second milestone.
Gokberk Yilmaz	-Implementing Post component -Implementing Community Page component -Working with backend to solve CORS policy error -Implementing API call function using Axios -Implementing a Post type creation component, then deciding on a different approach with Taha, he re-implemented it -Reviewing all PRs of frontend: Post creation, post type creation, scroll bug fix. -Attending all lab sessions -Keeping in touch with backend team while solving CORS policy error -Testing login and register endpoints via frontend page

	-Connecting login and register pages with backend -Opening issues and PRs related to frontend, they can be seen in my personal report in details.
Ekrem Gezgen	
Enes Aydogduoglu	Attended all lessons, Zoom and Discord meetings. Uploaded Android team meeting notes to Github. Reviewed my teammates' works. Opened some issues. Reviewed pull requests.

PROGRESS ACCORDING TO REQUIREMENTS

1. Functional Requirements

1.1. User Requirements

1.1.1. Sign Up

- [Completed] 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.

1.1.2. Sign In

- [Completed] 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 started] 1.1.2.3. Registered users shall be able to sign in using their google and facebook accounts.

1.1.3. Guest Users

- [Not started] 1.1.3.1. Guest users shall be able to search for communities and profiles.
- [Not started] 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

◦ [In progress] 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.
◦ [In progress] 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.
◦ [In progress] 1.1.4.1.6. Registered users shall be able to leave communities that they have subscribed before from the community page.
◦ [In progress] 1.1.4.1.7. Registered users shall be able to report the content they find inappropriate or offensive from the post.
◦ [Not started] 1.1.4.1.8. Registered users shall be able to directly follow other registered users with public profile from their profile page.
◦ [In progress] 1.1.4.1.9. Registered users shall be able to send request for following registered users with private profile from their profile page.
◦ [In progress] 1.1.4.1.10. Registered users shall be able to unfollow a registered user from their profile page.
◦ [Not started]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.)):
▪ [Completed] 1.1.4.1.12.1. Creating Posts from the community profile page via using a post type from community post types. (2,4)
▪ [In progress] 1.1.4.1.12.2. Interact with interactive fields of the post from post. (1,2,4)
▪ [In progress] 1.1.4.1.12.3. Commenting posts from post. (1,2,4)
▪ [In progress] 1.1.4.1.12.4. Hitting the like button on the post. (1,2,4)
▪ [In progress] 1.1.4.1.12.5. Replying to comments from the post. (1,2,4)
▪ [In progress] 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 started] 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)
◦ [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.

◦ [Not started] 1.1.4.1.16. Registered users shall be able to filter posts according to the specific post type and its fields in the "main feed".

• 1.1.4.2. Community Admin

◦ [Not started] 1.1.4.2.1. Community admins shall inherit the Registered User's requirements specified in requirement part 1.1.4.1.

◦ [Not started] 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.

◦ [Not started] 1.1.4.2.3. Community admins shall be able to warn Community Members because of hazardous posts from the post.

◦ [Not started] 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:

▪ 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

• [Not started] 1.2.1.1. System shall be able to recommend content and persons of interest.

1.2.2. Notification

• [Not started] 1.2.2.1. Users who turned on their notifications to a post shall be notified with the developments on that post.

• [Not started] 1.2.2.2. App should provide notifications to the notification center of the mobile device.

1.2.3. Authorization

• [In progress] 1.2.3.1. The privacy levels of users who accept the invitations from community admins shall be set accordingly.

• [In progress] 1.2.3.2. The privacy levels of users who removed from the community will be set accordingly.

• [In progress] 1.2.3.3. The privacy levels of users will be set according to the privacy option of the community.

1.2.4. Community
- [Completed] 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

1.2.5. Searching/Filtering
- [In progress] 1.2.5.1. Upon searching related communities, people and posts shall be indexed under the search bar.

1.2.6. Home Page
- [In progress] 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.

1.2.7. Registered User
- [Completed] 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
- [Not started] 1.2.8.1 The semantics tagging shall be supported by wikidata.org

2. Non-Functional Requirements

2.1. Security
- [Not started] 2.1.1 User passwords shall be encrypted or hashed
- [Not started] 2.1.2 The system shall use https Protocol

2.2. Performance
- [Completed] 2.2.1. 2000 users shall be supported
- [Not started] 2.2.2. 50 requests shall be responded in a second
- [Not started] 2.2.3. 4 seconds shall be the maximum response time

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

2.4. Standarts

- [Not started] 2.4.1. The system shall use the W3C activity stream standard 2.0 to implement subscriptions and notifications

2.5. Privacy

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

API ENDPOINTS

Postman link for all our endpoints:
https://app.getpostman.com/join-team?invite_code=760a4f338193055fce219bf9ac507dd3&target_code=625d200645d9379b53bbe9e06d96a5d3

Our all endpoints accepts json format input. Other types of input should not be given to our endpoints.

Sign Up:
http://3.134.93.99:8080/api/sign_up/ method="POST"

To successfully sing up, a new user_name and a mail_adress should be provided. And password should include number, punctuation, upper letter, lower latter with at least 8 characters long. After sign up process handled, a new profile page will be created according to the given information. Other information related to profile page will be null for the first time. But when user comes to their page, they shall update their profile page. This process handled by another endpoint declared in the below.

```
input:
   {
      'user_name': <user_name>,   //string
      'mail_address': <mail>,     //string
      'password': <password>      //string
   }

outputs:
   if <password> is not secure:
      {
         'response_message': 'Password is not secure enough.'
      },
      status code = 403
   if <mail> is already in use:
      {
         'response_message': 'E-mail address already exists.'
      },
      status code = 403
   if <user_name> is already exists:
      {
         'response_message': 'User name already exists.'
      },
      status code = 403
   if input json is not provided in the defined way:
      {
         'response_message': 'Incorrect json content. (necessary fields are
mail_address,user_name,password)'
      },
      status code = 400
   if successful:
      {
         'response_message': 'User successfully signed up.'
      },
      status code = 201
```

Sign In:
http://3.134.93.99:8080/api/sign_in/   method="POST"

To successfully sign in, already existing user_name should be given. Also the given password should match with the given user_name information.

```
input:
    {
        'user_name': <user_name>,   // string
        'password': <password>      // string
    }

outputs:
    if <user_name>, <password> pair does not exist in the database:
        {
           'response_message': 'Credentials are incorrect',
           'user_name': None
        },
        status code = 401
    if input json is not provided in the defined way:
        {
           'response_message': 'Incorrect json content. (necessary fields are
mail_address,user_name,password)',
            'user_name': None
        },
        status code = 400
    if successful:
        {
            'response_message': 'Successfully signed in.',
            'user_name': <user_name>
        },
        status code = 201
```

Update Profile Page:
http://3.134.93.99:8080/api/profile_page/ method="POST"

To successfully update a profile page, all the necessary fields declared in the below (but if the wanted update fields less than the given template, it also handles to update given parts). Also, user name should exist before hand. But this process does not includes the user itself. Because giving the correct information handled via the front end part and users only should describe the other fields described below.

```
input:
    {
        'user_name': <user_name>,   //String
        'profile_photo': <image>,   //String url
        'bio': <information>,       //String
        'first_name': <name>,       //String
        'last_name': <surname>,     //String
        'birth_date': <date>        //String: '13.07.1997'
    }
outputs:
    if successful:
        {
            'response_message': 'User page updated successfully.'
        }
        status code = 200
    if there is no user with <user_name>:
        {
            "response_message": "No such user."
        }
        status code = 400
```

Get Profile Page Infromation
http://3.134.93.99:8080/api/profile_page/ method="GET"

This endpoint is used when a user wisits their profile page or other user's profile page. Resulting information is used to prepare a profile page corrrectly and fully in the front end. To successfully get profile page information, existing user_name should be provided.

```
input:
   {
      'user_name': <user_name>    //String
   }
outputs:
   if successful:
      {
         "data": {
            'profile_photo': <image>,       //String (base64)
            'followers': <follower_list>,   //List of String holding <user_name>s
            'following': <following_list>,  //List of String holding <user_name>s
            'user_name': <user_name>,       //String
            'first_name': <name>,           //String
            'last_name': <surname>,         //String
            'birth_date': <date>,           //String: '13.07.1997'
            'post_list': <post_list>        //List of String holding <post_id>s
         },
         "response_message": "Profile page successfully returned. "
      }
      status code = 200
   if there is no user with <user_name>:
      {
         "response_message": "No such user."
      }
      status code = 400
```

Create Post Type:
http://3.134.93.99:8080/api/post_type/ method="POST"

<field_name>s are "PlainText", "Photo", "DateTime", "Document", "Price", "Location", "Poll", "Participation". All field types hold information in an array and this structure should be followed by the endpoint user.

```
input:
   {
      "fields_dictionary": {
         <field_name_0>: [                       //String
            {
               "header": <header_name_0>             //String
            },
            {
               "header": <header_name_1>              //String
            }
            .
            .
            .
         ],
         <field_name_1>: [                            //String
            {
               "header": <header_name_2>              //String
            }
            .
            .
            .
            ]
         .
         .
         .
      },
      "user_name": <user_name>,                       //String
      "post_type_name": <post_type_name>              //String
      "parent_community_id": <community_id>           //integer
   }
output:
   if successful:
      {
         "data": {
            "post_type_id": <post_type_id>            //integer
         },
         "response_message": "PostType is successfully created."
      }
      status code = 200
```

Get Post Type
http://3.134.93.99:8080/api/post_type/ method="GET"

To correctly get post type, <post_type_id> should be provided correctly. Output shows all the information about the existing post type. For instance, even if the post type does not include any "DateTime" object, we show this field with empty array.

```
input:
   {
      "post_type_id": <post_type_id>                  //integer
   }
output:
```

```
{
   "data": {
      "fields_dictionary": {
         "DateTime": <date_time_array>,              //List of DateTime object
         "Document": <document_array>,               //List of Documentation object
         "Location": <location_array>,               //List of Location object
         "Participation": <participation_array>,     //List of Participation object
         "Photo": <photo_array>,                     //List of Photo object
         "PlainText": <plain_text_array>,            //List of PlainText object
         "Poll": <poll_array>,                       //List of Poll object
         "Price": <price_array>                      //List of Price object
      },
      "parent_community_id": <community_id>,         //Integer
      "post_type_id": <post_type_id>,                //Integer
      "post_type_name": <post_type_name>,            //String
   },
   "response_message": "PostType is successfully returned."
}
status code = 200
```

Create Post
http://3.134.93.99:8080/api/post/ method="POST"

To successfully create a post <post_type_id> should be exist. "fields_dictionary" holds information about related type field objects. Every <field_name> holds a list which should be correctly defined. Examples for some field types:

```
"PlainText": [
      {
         "text": "example_text_0"
      },
      {
         "text": "example_text_1"
      }
   ]

"Location": [
      {
         "text": "description_text_about_location",
         "location": "0001.12.13..13.1.31"
      }
   ]
```

input:

```
{
        "fields_dictionary": {
            <field_name_0>: <field_object_array_0>,            //List of related object
            <field_name_1>: <field_object_array_1>,            //List of related object
            .
            .
            .
        }
        "user_name": <user_name>,                          //String
        "post_type_id": <post_type_id>                     //Integer
    }
output:
    if successful:
        {
            "data": {
                "post_id": <post_id>                       //Integer
            },
            "response_message": "Post is successfully created. "
        }
        status code = 200
```

Get Post
http://3.134.93.99:8080/api/post/ method="GET"

To get correct data correct <post_id> should be provided. This get mechanism aimed for showing posts related to user with <user_name>. It will be used in profile page to show user related posts.

```
input:
    {
        "post_id": <post_id>,                              //Integer
        "user_name": <user_name>                           //String
    }

output:
    if successful:
        {
            "data": {
                "base_post_type": {
                    "fields_dictionary": {
                        <field_name_0>: <field_array_0>,            //List of object
                        <field_name_1>: <field_array_1>,            //List of object
                        <field_name_2>: <field_array_2>,            //List of object
                        <field_name_3>: <field_array_3>,            //List of object
                        <field_name_4>: <field_array_4>,            //List of object
                        .
                        .
                        .
                    },
                    "parent_community_id": <community_id>,             //Integer
                    "post_type_id": <post_type_id>,                   //Integer
                    "post_type_name": <post_type_name>                //String
                },
                "post_id": <post_id>,                                 //Integer
                "post_owner_user_name": <user_name>                   //String
            },
            "response_message": "Post is successfully created. "
        }
        status code = 200
```

Create Community Page
http://3.134.93.99:8080/api/community_page/ method="POST"

Creation of community page needs a new <community_page_id>. <is_private> information holds the privacy of the community page and shows whether it is open to be viewed from non subscribers or not. <community_creator_id> is directly taken from the front side and there is no need to be specified from the application user.

```
input:
   {
      'id': <community_page_id>,                 //String
      'is_private': <community_privacy_info>,    //Boolean
      'community_creator_id': <user_name>,       //String
   }
output:
   if successful:
      {
         'response_message': "Community Page successfully created."
      }
      status code = 201
   if input json is not provided in the defined way:
      {
         'response_message': "Incorrect json content. (necessary fields are id, is_private,
community_creator_id)"
      },
      status code = 400
   if already a community exists with given <community_page_id>:
      {
         'response_message': "Community ID is already in use"
      }
      status code = 403
   if some unexpected error occurs in the database:
      {
         'response_message': 'Internal Error'
      }
      status code = 500
```

Get Community Page Information
http://3.134.93.99:8080/api/community_page/ method="GET"

Only need in this endpoint is the <community_page_id>. This information will be given by the front side when a user wants to visit a community page.

```
input:
    {
        'id': <community_page_id>           //String
    }
output:
    if successful:
        {
           'response_message': "Community successfully found",
           'community_instance':
              {
                 'id': id,                                  //String
                 'admin_list': admin_list,                      //List of Strings
                 'subscriber_list': subscriber_list,            //List of Strings
                 'post_type_id_list': post_type_id_list,         //List of Strings
                 'post_history_id_list': post_history_id_list,   //List of Strings
                 'description': description,                   //String
                 'photo': photo,                            //String (base64)
                 'community_creator_id': community_creator_id,   //String
                 'created_at': created_at,                   //String: '13.07.1997'
                 'banned_user_list': banned_user_list,           //List of Strings
                 'is_private': is_private                     //Boolean
              }
        }
        status code = 200
    if input json is not provided in the defined way:
        {
           'response_message': "Incorrect json content. (necessary field is id)"
        },
        status code = 400
    if there is no community with the specified <community_page_id>:
        {
           'response_message': "Specified community with the id not found"
        },
        status code = 403
```

Update Community Page information
http://3.134.93.99:8080/api/community_page/ method="PUT"

All the necessary information is given below. Additional information or lack of information will result an error message. The system takes whole information related to a page and updates all the given fields.

```
input:
   {
      'id': id,                                    //String
      'admin_list': admin_list,                    //List of Strings
      'subscriber_list': subscriber_list,          //List of Strings
      'post_type_id_list': post_type_id_list,      //List of Strings
      'post_history_id_list': post_history_id_list,   //List of Strings
      'description': description,                  //String
      'photo': photo,                              //String (base64)
      'community_creator_id': community_creator_id,   //String
      'created_at': created_at,                    //String: '13.07.1997'
      'banned_user_list': banned_user_list,        //List of Strings
      'is_private': is_private                     //Boolean
   }
output:
   if successful:
      {
         'response_message': "Community successfully updated",
         'community_instance':
            {
               'id': id,                                    //String
               'admin_list': admin_list,                    //List of Strings
               'subscriber_list': subscriber_list,          //List of Strings
               'post_type_id_list': post_type_id_list,      //List of Strings
               'post_history_id_list': post_history_id_list,   //List of Strings
               'description': description,                  //String
               'photo': photo,                              //String (base64)
               'community_creator_id': community_creator_id,   //String
               'created_at': created_at,                    //String: '13.07.1997'
               'banned_user_list': banned_user_list,        //List of Strings
               'is_private': is_private                     //Boolean
            }
      }
      status code = 201
   if invalid input error:
      {
        'response_message': "Incorrect json content. (necessary field are the
community class fields)"
    }
    status code = 400
  if there is no community with the specified <community_page_id>:
    {
        'response_message': "Specified community with the id not found"
    },
    status code = 403
  if some unexpected error occurs in the database:
    {
        'response_message': 'Internal Error'
    }
    status code = 500
```

IMPLEMENTATION OF CORE FUNCTIONALITY

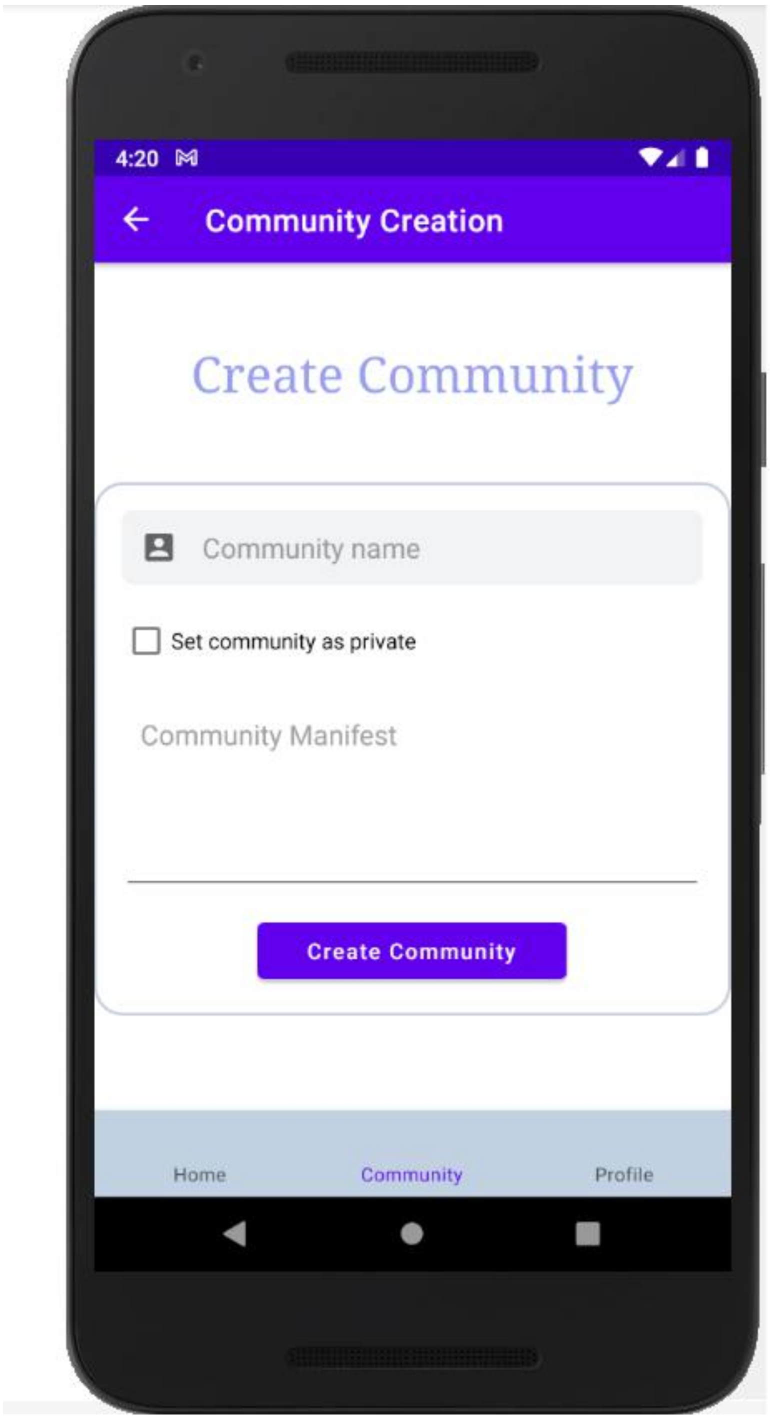
Mapping functionality:

Mapping functionality implemented in frontend ,but there is no mapping functionality in android part. Posts which have a location field are displayed on the map. We have used Google maps api to implement this feature. This feature is not integrated with backend yet.

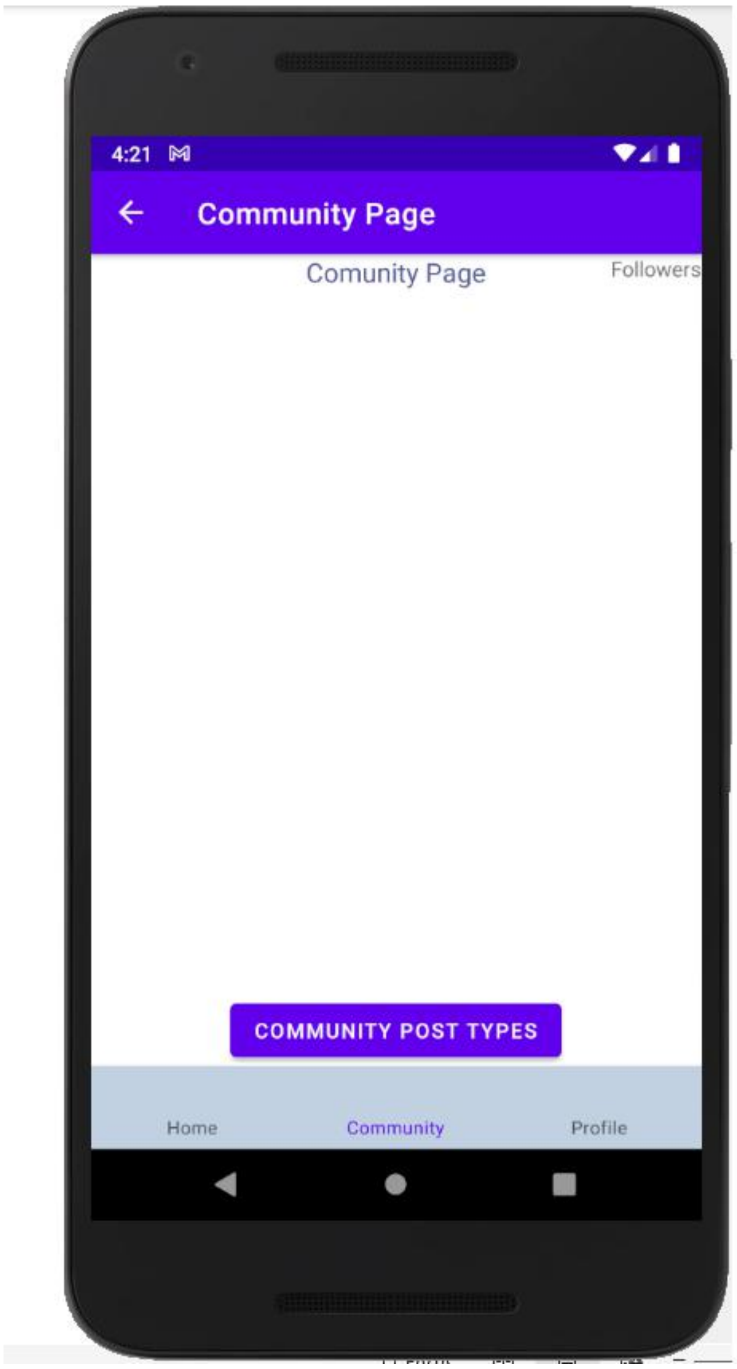
USER INTERFACE

Mobile Interfaces

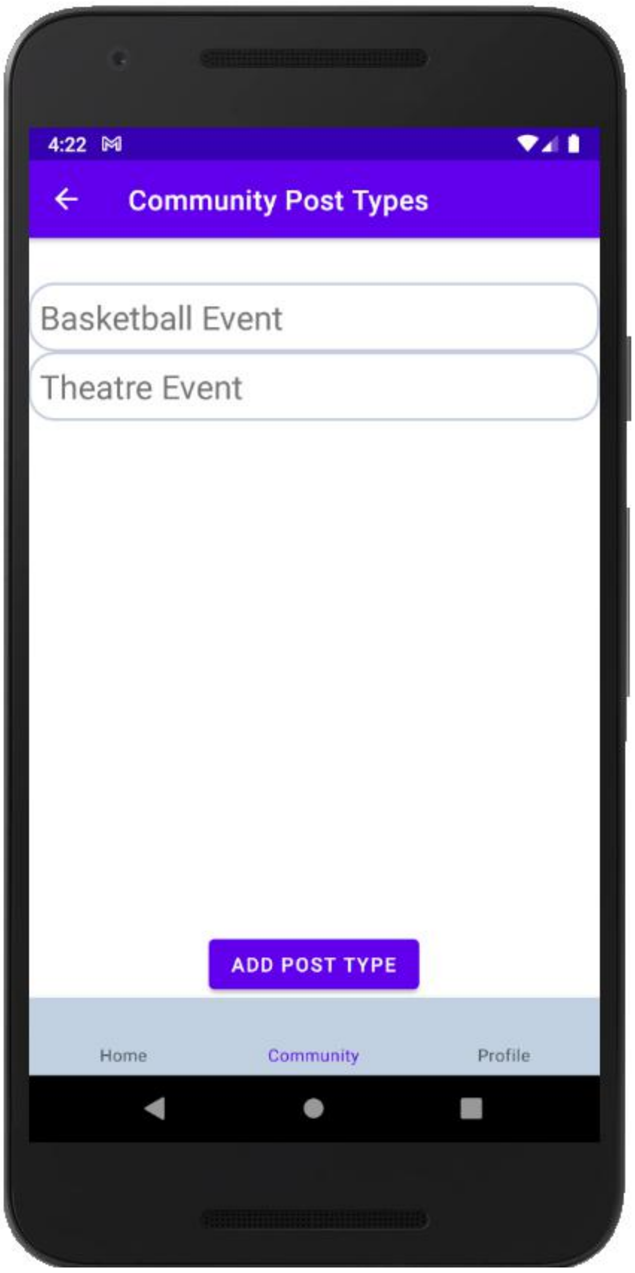
a) Community Creation



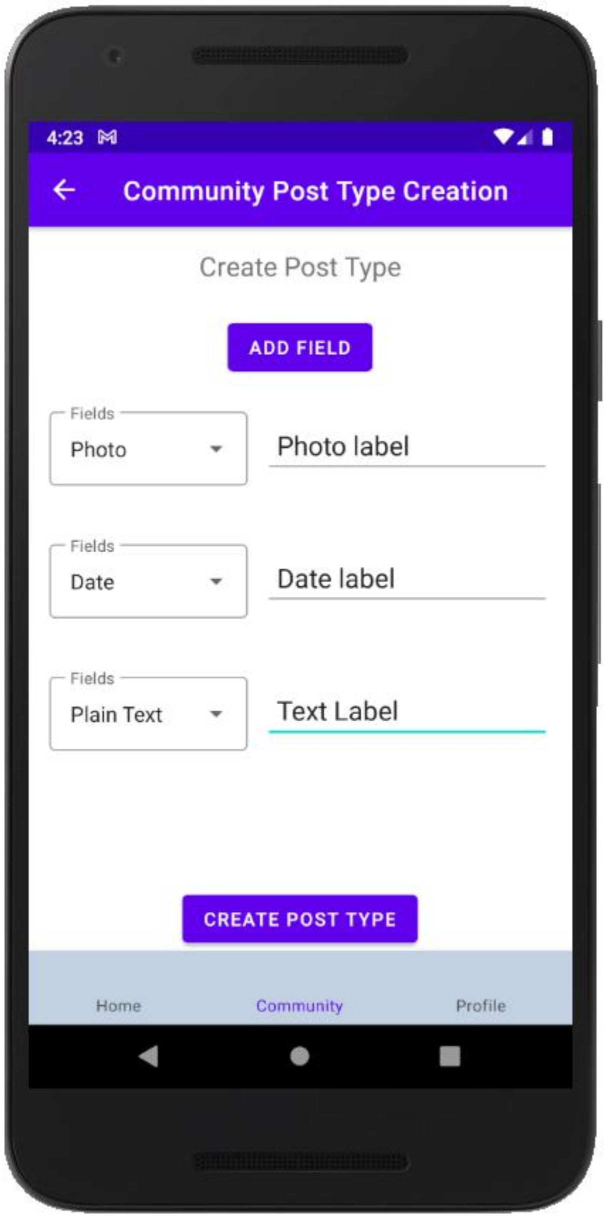
b) Community



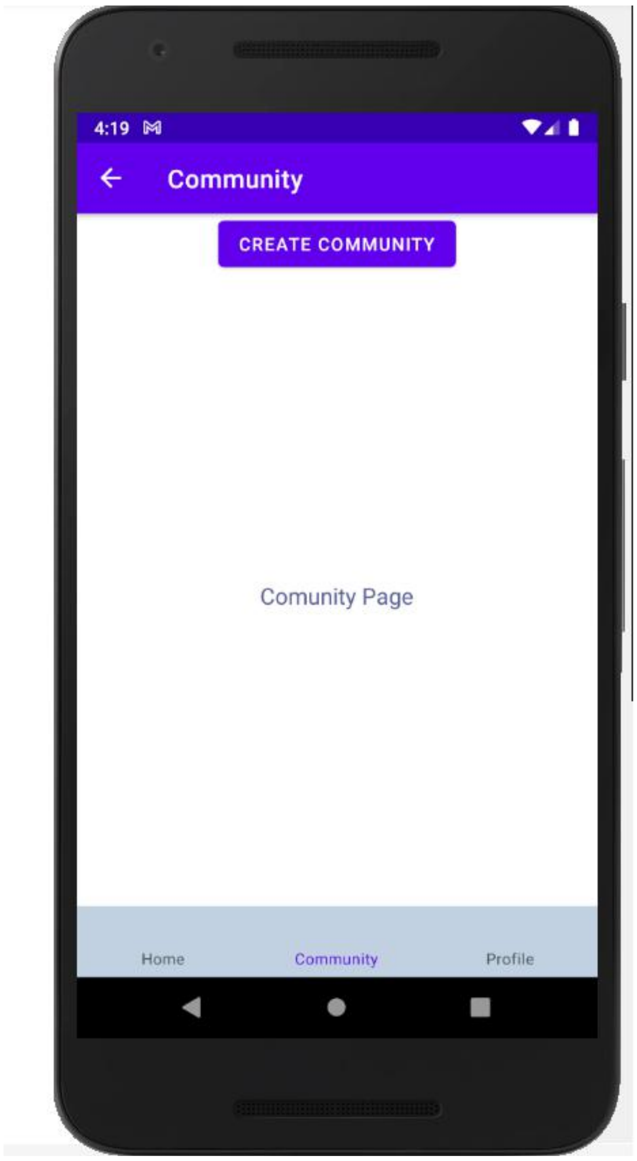
c) Community Post Types



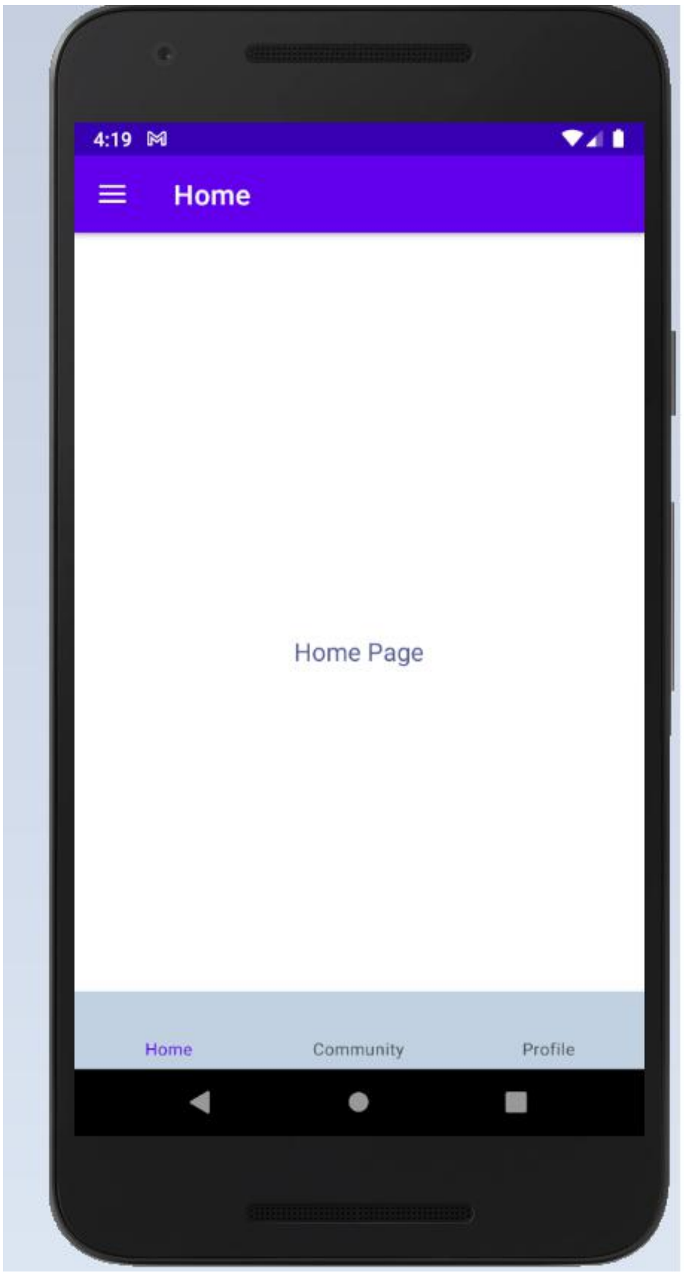
d) Create Post Type



e) Subscribed Communities

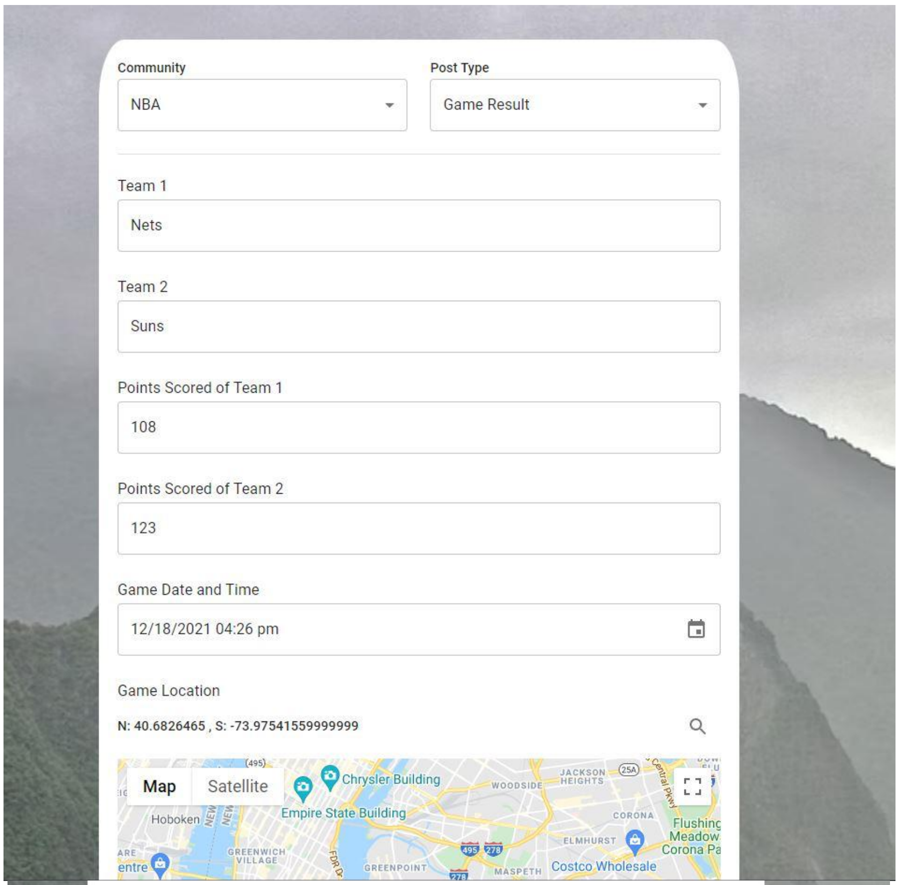

4:19
Community
CREATE COMMUNITY
Comunity Page
Home
Community
Profile

f) Home Page



Web Interfaces

a) Create Post



Community

NBA

Post Type

Game Result

Team 1

Nets

Team 2

Suns

Points Scored of Team 1

108

Points Scored of Team 2

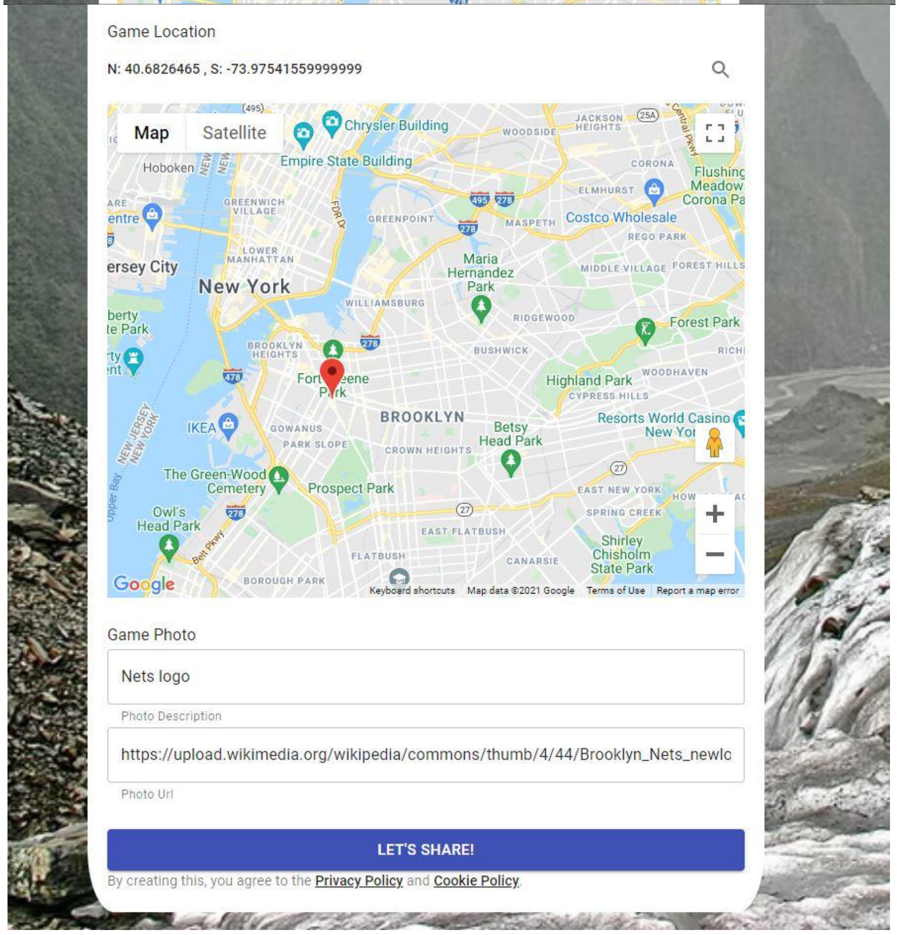
123

Game Date and Time

12/18/2021 04:26 pm

Game Location

N: 40.6826465 , S: -73.97541559999999



Game Location

N: 40.6826465 , S: -73.97541559999999

Game Photo

Nets logo

Photo Description

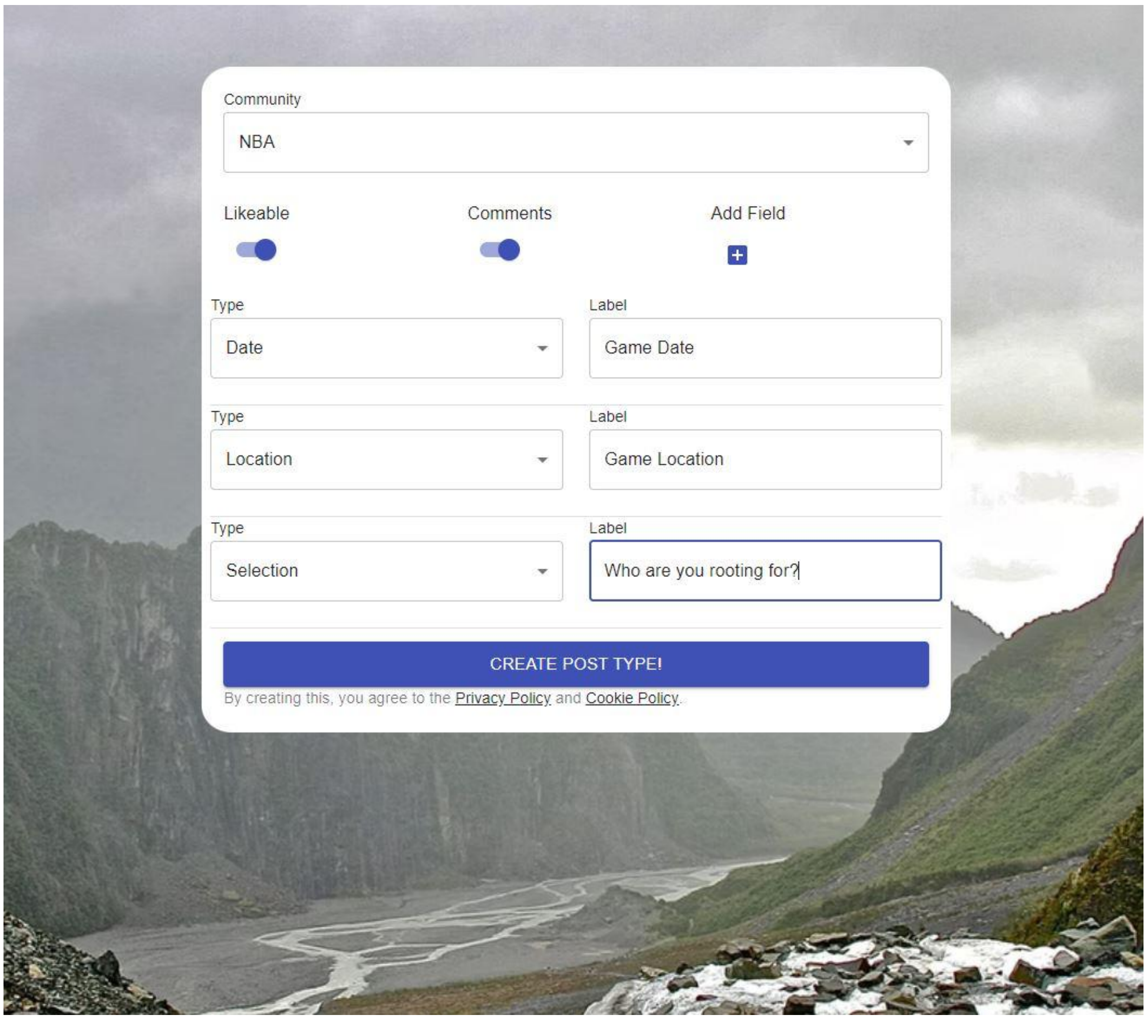
https://upload.wikimedia.org/wikipedia/commons/thumb/4/44/Brooklyn_Nets_newlc

Photo Url

LET'S SHARE!

By creating this, you agree to the **Privacy Policy** and **Cookie Policy**.

b) Create Post Type



Community

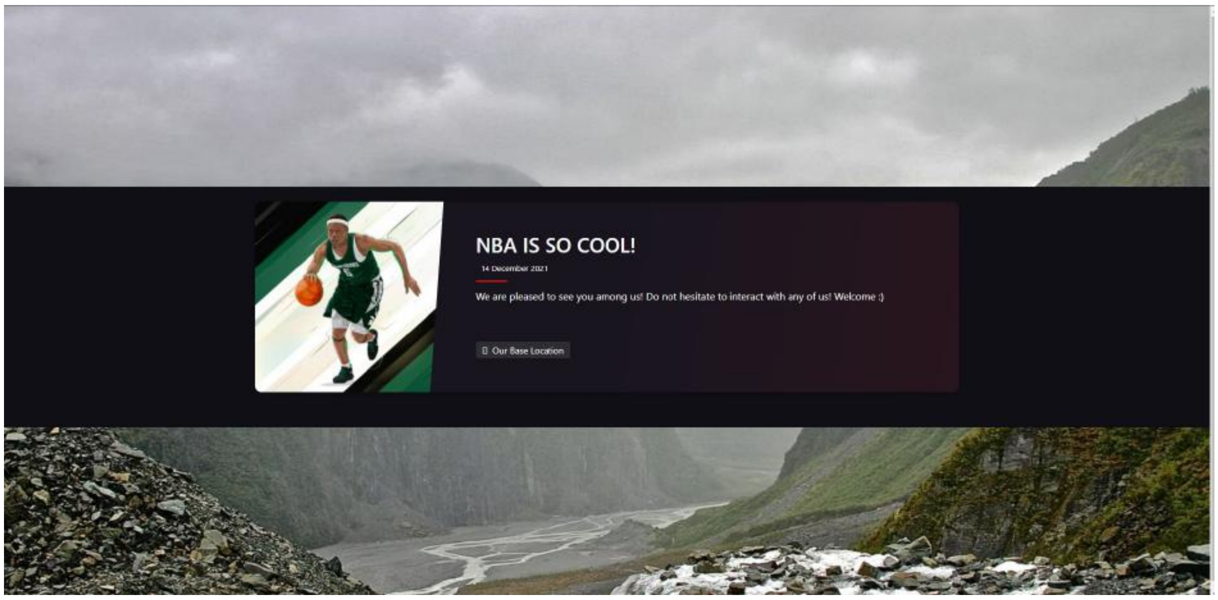
NBA

Likeable Comments Add Field

Type	Label
Date	Game Date
Type	Label
Location	Game Location
Type	Label
Selection	Who are you rooting for?

By creating this, you agree to the Privacy Policy and Cookie Policy.

a) Post Page

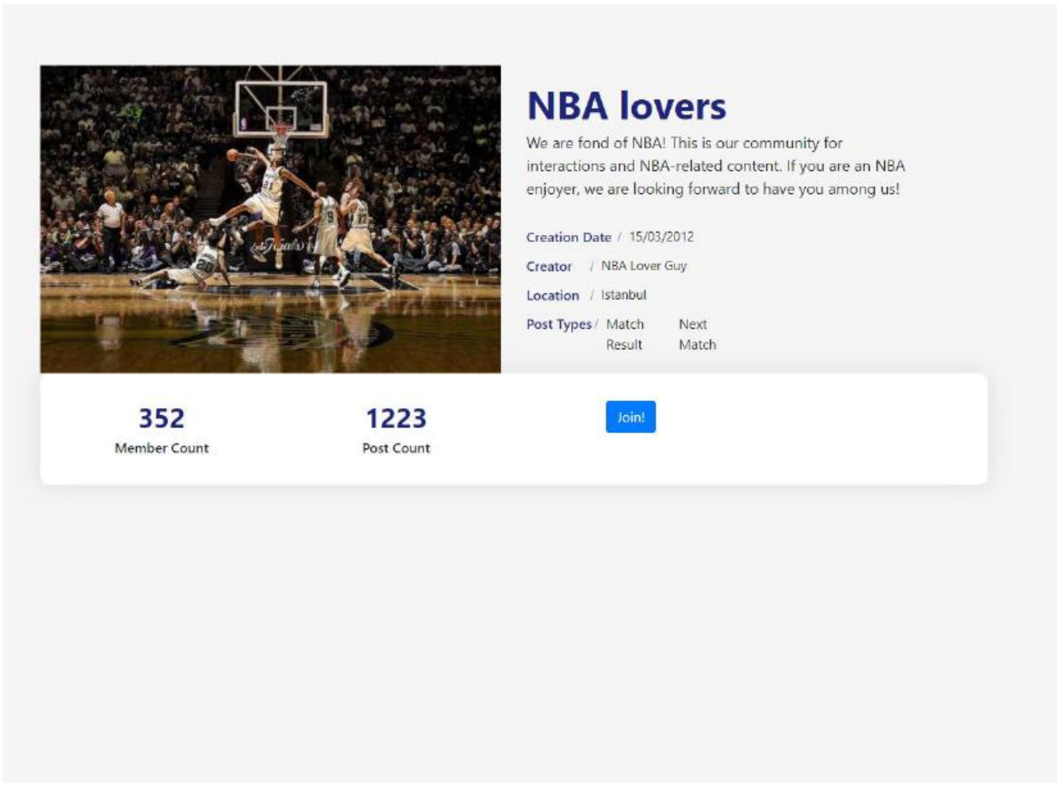


NBA IS SO COOL!

14 December 2021

We are pleased to see you among us! Do not hesitate to interact with any of us! Welcome :)

c) Community Page



NBA lovers

We are fond of NBA! This is our community for interactions and NBA-related content. If you are an NBA enjoyer, we are looking forward to have you among us!

Creation Date / 15/03/2012

Creator / NBA Lover Guy

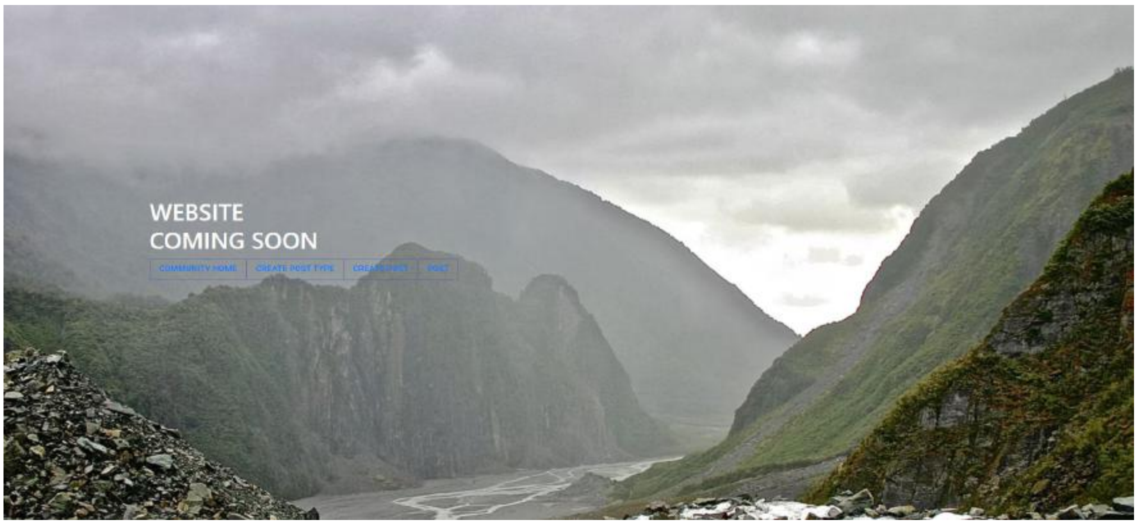
Location / Istanbul

Post Types / Match Result   Next Match

352
Member Count

1223
Post Count

d) Home Page


WEBSITE
COMING SOON
COMMUNITY HOME
CREATE POST TYPE
CREATE POST
POST