CONTENTS

Requirements

Requirement Changes Report:

1-There was an unnecessary corporate user registered user duality, it is removed.
2-There was a duality of Project and Paper. It was creating confusion and redundancy. After talking with the customer it is removed.
3-Password reset feature has been added.
4-Text editor feature has been added to the system.
5-Users can add private information to their profiles.
6-Comment related elements are removed
7-Rating is replaced with upvote
8-Research area, topic, scope and difficulty are removed from filtering system
9-”Conferences or journals” terms in requirements are replaced with events.
10-Description for events is added in glossary

Glossary

1. **User**:A person capable of contributing to or collaborating with others Projects, and has their own profile page. Authentication is required.
2. **Collaborator**: A user that participates in a project.
3. **Profile Page**: Page that users shall be able to share information about themselves.
4. **Events**: The news about upcoming conferences, meetings and important dates.
5. **Profile**: A visual display of personal data associated with a specific user.

6. **Search**: Find semantically similar users and Projects on the context information provided in the semantic tags.
7. **Password**: Consists of at least six characters and at most sixteen characters that are a combination of letters, numbers and special characters.
8. **Projects**: Long-term works that involve milestones, business packages, deadlines.
9. **Basic Search**: Only text matching based search.
10. **Tag**: Small labels that indicate related aspects of a project.
11. **Notification**: Message that informs the user about a specific event.
12. **Recommendation System**: System that recommends contents and projects that users might be interested in.
13. **Academician**: Users that teach or do research in university.
14. **Public**: Projects that can be seen by all users and can be requested to join by all users.
15. **Private**: Projects that can be seen by only collaborators and the authors of them can invite other users.
16. **Author**: Owner of the project.

Project Requirements

1. Functional Requirements

1.1 *User Requirements*

- 1.1.1 *Registration*
    - 1.1.1.1 Users shall be able to register by providing name, surname, unique email address and password.
    - 1.1.1.2 Users shall validate the email address for completing the registration phase.
    - 1.1.1.3 Users shall specify the research interests by tags.
    - 1.1.1.4 Users shall provide the information about affiliation.
- 1.1.2 Users shall login by email and password provided.
- 1.1.3 Users shall be able to reset their passwords through their associated email.
- 1.1.4 Users shall be able to search and view users, public projects.
- 1.1.5 Users shall be able to view profile pages of authors, and collaborators of projects.
- 1.1.6 Users shall be able to search related information about the upcoming events.
- 1.1.7 Users shall be able to send request to join the public projects as collaborators.
- 1.1.8 Users, as the owner of the projects, shall be able to accept or reject requests.
- 1.1.9 Users shall be able to invite any user to their projects.
- 1.1.10 Users shall be able to accept or reject invitations.
- 1.1.11 User shall be able to disable join requests to end call for collaboration phase.
- 1.1.12 Users shall be able to upvote other users.
- 1.1.13 Users shall be able to collaborate on more than one project.
- 1.1.14 Users shall be able to have more than one project topic posted.
- 1.1.15 Users shall be able to specify a deadline for the project.
- 1.1.16 Users shall not be able to delete a project that is in progress.
- 1.1.17 Users shall be able to log out from his account.

- 1.1.18 Users shall be able to delete his account.
- 1.1.19 Users shall be able to revert an upvote they gave before.
- 1.1.20 Users shall be able to add additional files to the collaborated projects.
- 1.1.21 Users shall be able to follow other users on the platform.

-1.1.23 Profile Page

- 1.1.23.1 Users shall be able to provide information about the research area, recent publications and affiliation.
- 1.1.23.2 Users shall be able to link their Google Scholar or ResearchGate accounts.
- 1.1.23.3 Users shall be able to edit their own profile page and make some personal information private.
- 1.1.23.4 Users shall be able to see invitations that are sent from the other users.
- 1.1.23.5 Users shall be able to see upvotes that are made by other users.
- 1.1.23.6 Users shall be able to see their followers.
- 1.1.23.7 Users shall be able to see people who they follow.

-1.1.24 Creating / Editing Page

- 1.1.24.1 Users shall be able to provide information about: topic of the research, deadline of submission, milestones, codes, documents, result plots / figures, required skills in order to apply and if the project is funded or not.
- 1.1.24.2 Users shall be able to add a summary(abstract) part.
- 1.1.24.3 Users shall be able to state the type of the content (Project).
- 1.1.24.4 Users shall be able to state whether project is private or public.
- 1.1.24.5 Users shall be able to add collaborators during creation of a project.
- 1.1.24.6 Users shall be able to add tags related to project.
- 1.1.24.7 Collaborators shall be able to track the information shared by other collaborating users.
- 1.1.24.8 Collaborators shall be able to prepare document or article for submission on the platform.

- 1.1.25 Project Page

- 1.1.25.1 Users shall be able to create a new project.
- 1.1.25.2 Users shall be able to see the status of their project(s).
- 1.1.25.3 Users shall be able to edit the specifications of their project(s).
- 1.1.25.4 Users shall be able to see their project(s) in progress.
- 1.1.25.5 Users shall be able to edit the specifications of their project(s) in progress.
- 1.1.25.6 Users shall be able to invite another user to collaborate on their own project(s) in progress.
- 1.1.25.7 Users shall be able to accept or reject other users' request to collaborate on their own project

- 1.1.26 Search Page

- 1.1.26.1 Users shall be able to search; other users, projects, and events within the system.
- 1.1.26.2 Users shall not be able to see private projects unless being a contributor of that project.
- 1.1.26.3 Users shall be able to filter search results with regards to the tag.

- 1.1.26.4 Users shall be able to find related posts searching related keywords.
- 1.1.26.5 Users shall be able to view abstract and description of projects/events related to their activities, profile and interests.

- 1.1.27 Events Page

- 1.1.27.1 Users shall be able to see upcoming events.
- 1.1.27.2 Users shall be able to see closing deadlines of joined projects.
- 1.1.27.3 Users shall be able to travel in the timeline to see past and future events.
- 1.1.27.4 Users shall be able to view particular event more detailed when clicked/tapped.

1.2 System Requirements

- 1.2.1 Search & Recommendation

- 1.2.1.1 System shall support basic search of the available content.
- 1.2.1.2 System shall support semantic search.
- 1.2.1.3 System shall provide a tagging system.
- 1.2.1.4 System shall provide a filtering system. This system filters the content such as projects/profiles shown to the user.
- 1.2.1.5 System shall provide a recommendation mechanism that recommends projects/events to the user based on the user activities and profile.

- 1.2.2 Homepage

- 1.2.2.1 System shall provide a homepage for each user, showcasing recent and followed users' project(s).
- 1.2.2.2 System shall provide ongoing activities of the collaborators for each user.

- 1.2.3 Profile Page

- 1.2.3.1 System shall provide a profile page for each user, showcasing their interests, past contributions to projects, and their related links (example: Google Scholar).
- 1.2.3.2 System shall retrieve information of a user from their Google Scholar/ResearchGate pages.

- 1.2.4 Notifications

- 1.2.4.1 System shall provide a notification mechanism for informing users about the incoming invitations.
- 1.2.4.2 System shall provide a notification mechanism for informing users about whether their requests to join a project accepted or rejected.
- 1.2.4.3 System shall provide a notification mechanism for informing users about whether someone has upvoted on their profile.

- 1.2.5 Events

- 1.2.5.1 System shall provide a page of events for every user.
- 1.2.5.2 System shall fill this page with the events related to user.

- 1.2.6 System shall provide a text editor for publishing documents and articles.

2. Nonfunctional Requirements

- 2.1 Activity Streams

- 2.1.1 System shall obey [W3C Standards]
- 2.1.2 System shall follow [W3C Activity Streams Model]

- 2.2 Availability & Accessibility

- 2.2.1 The system shall be available as a web application and an Android application.
- 2.2.2 Applications shall be available in English.
- 2.2.3 The system shall notify users in case of a failure.
- 2.2.4 The system shall support Safari, Chrome and Firefox as a browser and shall support Android 6.0 and later versions.

- 2.3 Performance & Scalability

- 2.3.1 The system shall use caching mechanism to reduce response time.
- 2.3.2 The system shall respond to a request at most 15 seconds.
- 2.3.3 The system shall be able to respond up to 100 requests per second.

- 2.4 Security

- 2.4.1 The system shall be protected from unauthorized accesses and attacks to the system and its stored data according to [KVKK], [GDPR].
- 2.4.2 The system shall force users to use strong passwords that contain a lowercase letter, an uppercase letter, a number and a special character and contain 6-16 characters in total.
- 2.4.3 The system shall store only the hashed version of passwords.
- 2.4.4 The system shall provide a validation code for users during registration.
- 2.4.5 The system shall be protected against SQL injection and DDOS attacks.

Unit Tests

- *Cihat Kapusuz*

  commit SHA: "**23308efd832abc0e64258f3a5aab652b18767070**"

  Tested functionality: Checked the inserted date is in the expected format or not.
  What have done: Implemented 3 unit tests for validity of the date format. Expected date format is DD/MM/YY. First unit test is in the correct format and the other 2 are not in the correct format.

- *Ezgi Gülperi Er*

  commit SHA: "**8e28cfad1971a243c9a281ce8749d02ff64599cb**"

Tested functionality: Checked if the registered email address has a valid format or not .
What have done: Implemented unit tests to check the functionality, checked 5 test cases.

API Documentation

Allowed HTTPs requests:

POST: To create a resource
PATCH: To update a resource
GET: To get a resource or list of resources
DELETE: To delete a resource

Description of Usual Server Responses:

200: OK => the request successfully handled
201: CREATED => the request successfully handled and a resource created
204: NO CONTENT => the request successfully handled and there is nothing to return
400: BAD REQUEST => the request could not be understood. (missing parameters)
401: UNAUTHORIZED => authentication failed. user does not have permissions.
404: NOT FOUND => resource was not found.
500: INTERNAL SERVER ERROR => something went wrong in the server.

Endpoints:

/auth/login
Request type: POST
Required parameters: email type of String, password type of String.
Responses:
1-) code: 200, message: “Success” (String), body: {authorization token type of String, userId type of Integer}.
2-) code: 401, message: “Wrong password” (String)
3-) code: 404, message: “User not found” (String)

Description: /auth/login endpoint is for registered users who want to gain authorization using their credentials. The user sends their credentials in the body of this post request. First, their email is checked for validity, then the hash of their password is compared with the one with the database. If the password matches, response 1 is returned. If the email address is valid but the password does not match, response 2 returned. If the email address has no user registered via itself, response 3 returned.

/auth/signup
Request type: POST
Required parameters: email type of String, password type of String, name type of String, surname type of String.
Responses:

1-) code: 201, message type of String, body: {authorization token type of String, userId type of Integer}.
2-) code: 400, message: “This email address is already used” (String)
3-) code: 400, message: “Something went wrong” (String)

Description:/auth /signup endpoint is for those who want to register to the application. The user sends their email address, password, name and surname in the body of this post request. First, their email is checked to know if the email address is in use or not, then their password is hashed and an authorization token is created. After that the user is saved to the database and a validation code is sent to their email address. If no errors are thrown in the processes above, response 1 is returned. If the given email address is in use, response 2 is returned. For any other errors that can be thrown at the server side, response 3 is returned.

/auth/forgot
Request type: POST
Required parameters: email type of String
Responses:
1-) code: 200, message: “success” (String)
2-) code: 400, message: “Something went wrong” (String)
3-) code: 404, message: “No such email registered” (String)

Description:/auth/forgot endpoint is for those who have forgotten their passwords. The user sends their email address in the body of this post request. First, their email is checked to know if there is any user registered with that email. If not, response 3 is returned. Then a validation code is sent to their email for password change and response 1 is returned. For any other errors that can be thrown at the server side, response 3 is returned.
/auth/code
Request type: POST
Required parameters: email type of String, code type of String
Responses:
1-) code: 200, message: “success”, accessToken type of String, id type of Integer
2-) code: 400, message: “Something went wrong” (String)
3-) code: 404, message: “No such email registered” (String)
4-) code: 404, message: “Wrong code” (String)

Description:/auth/code endpoint is for those who have forgotten their passwords and got a rescue email. The user sends their email address and the code sent to them in the body of this post request. First, their email is checked to know if there is any user registered with that email. If not, response 3 is returned. Then the validation code is checked to know if it is true or not. If not, response 4 is returned. For any other errors that can be thrown at the server side, response 3 is returned.

/auth/jwt
Request type: POST
Required parameters: token type of String
Responses:
1-) code: 200, message: “Token is valid” (String), body: authorization token type of String.
2-) code: 400, message: “Token is invalid” (String)

Description: /auth/jwt endpoint is for the users who have an authorization token in their hand and want to check if they can continue being authorized with that token. The user sends their current authorization token in the body of this post request. The given token is tried to be verified. If the token has not expired and is still the valid token for the user, response 1 is returned. If the token has expired or does not match with the valid one, response 2 is returned.

/validate
Request type: POST
Required parameters: validation code type of String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Validation is successful” (String), accessToken type of String.
2-) code: 400, message: “Validation code is wrong”(String)

Description: /validate endpoint is for the users who have registered to the app but are yet to validate their email addresses. The user sends the validation code sent to their email address in the body of this post request. The given code is compared with the one in the database. If the code matches, response 1 is returned. If the code does not match, response 2 is returned.

/validate/fpassword
Request type: POST
Required parameters: password type of String
Responses:
1-) code: 200, message: “Password successfully changed” (String)
2-) code: 400, message: “Something went wrong” (String)
3-) code: 401, message: “Invalid token, you need a token with code parameter” (String)

Description:/validate/fpassword endpoint is for those who have forgotten their passwords and got a rescue email and entered the code correctly. The user sends their new passwords in the body of this post request. First, their token is checked to know if the user has done the steps above correctly. If not, response 3 is returned. Then their password is hashed and saved to the database and response 1 is returned. For any other errors that can be thrown at the server side, response 2 is returned.

/validate/npassword
Request type: POST
Required parameters: oldPassword type of String, newPassword type of String
Responses:
1-) code: 200, message: “Password successfully changed” (String)
2-) code: 400, message: “Something went wrong” (String)
3-) code: 403, message: “Old password is incorrect” (String)

Description:/validate/fpassword endpoint is for those who want to change their password. The user sends their new password and old password in the body of this post request. First, their old password is checked to know if it is correct. If not, response 3 is returned. Then

their new password is hashed and saved to the database and response 1 is returned. For any other errors that can be thrown at the server side, response 2 is returned.

/search?query&type&tags
Request type: GET
Required parameters: query type of String, type type of Integer, tags type of list of String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Users are fetched” (String), body: list of users type of User Model.
2-) code: 200, message: “Projects are fetched”(String), body: list of projects type of Project Model.
3-) code: 200, message: “Events are fetched”(String), body: list of events type of Event Model.
4-) code: 500, message: “Something went wrong” (String)

Description: /search endpoint is for the authorized users who want to search existing users, or projects, or events with the keyword they provide. The user sends the search keyword and search type the user wants to execute in the url parameters of this get request. If the type is 0, the user wants to search for users, if the type is 1, the user wants to search for projects, and if the type is 2, the user wants to search for events. All results come from the concatenation of the elastic-search results which provides semantic search and non-semantic search. If no errors are thrown in the process above, response 1 or 2 is returned based on the search type. If any error is thrown, response 3 is returned.

/profile/update
Request type: PATCH
Required parameters: interests type of list of strings, or affiliations type of json object containing title type of String, university type of String, and department type of String, or both.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /profile/update endpoint is for the authorized users who want to update a portion of their profile information. The portion is not necessarily the entire profile information. The user sends a new set of their interests or affiliations or both in the body of this patch request. The old data is removed from the database and new data is saved to the database. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/profile/update
Request type: POST
Required parameters: interests type of list of strings, or affiliations type of json object containing title type of String, university type of String, and department type of String, or both.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String)

2-) code: 500, message: “Something went wrong” (String)

Description: /profile/update endpoint is for the authorized users who want to add information to their profile. The user sends a new set of their interests and affiliations in the body of this post request. The data is saved to the database. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/profile/{userId}
Request type: GET
Required parameters: userId type of String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String), body: user profile type of Profile Model.
2-) code: 500, message: “User not found” (String)

Description: /profile/{userId} endpoint is for the authorized users who want to fetch the profile information of the user whose id is given by themselves. The user sends the userId of the user whose profile the user wants to fetch in the url parameters of this get request. First check is to find whether the userId matches with any user. If there is a match, response 1 with the profile information of the matched user in its body is returned. If there is no match, response 2 is returned.

/profile/biography
Request type: POST
Required parameters: bio of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String).
2-) code: 500, message: “Something went wrong” (String)

Description: /profile/biography endpoint is for the authorized users who want to add a short summary of their lives to their profile. The user sends the biography they want to add to their profile in the body of this post request. If successful, response 1, if not response 2 is returned.
/profile/avatar
Request type: POST
Required parameters: file type of jpg, jpeg, or png.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “File is uploaded” (String), name of type String, mimetype type of String, size type of Integer, path type of String(url).
2-) code: 500, message: “Something went wrong” (String)

Description: /profile/avatar endpoint is for the authorized users who want to add a profile picture to their profile. The user sends the file they want to add to their profile as the profile picture in the body of this post request. If successful, response 1, if not response 2 is returned.

/profile/googlescholar

Request type: POST
Required parameters: url type of String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” type of String
2-) code: 400, message: “Url is not valid” (String)
3-) code: 500, message: “Something went wrong” (String)

Description: /profile/googlescholar endpoint is for the authorized users who want to link their google scholar account to their akademise profile. The user sends the url of their google scholar account in the body of this post request. If successful, response 1 is returned. If the url is not valid, response 2 is returned. Otherwise, response 3 is returned.

/profile/up
Request type: POST
Required parameters: userId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” type of String
2-) code: 400, message: “User is not found” (String)
3-) code: 500, message: “Something went wrong” (String)

Description: /profile/up endpoint is for the authorized users who want to upvote another user. The user sends the user id of the user they want to upvote in the body of this post request. If successful, response 1 is returned. If there are no users with the given id, response 2 is returned. Otherwise, response 3 is returned.

/profile/disup
Request type: POST
Required parameters: userId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” type of String
2-) code: 400, message: “User is not found” (String)
2-) code: 400, message: “You have not upped this user” (String)
3-) code: 500, message: “Something went wrong” (String)

Description: /profile/disup endpoint is for the authorized users who want to remove upvote given for another user. The user sends the user id of the user for whom they want to remove upvote in the body of this post request. If successful, response 1 is returned. If there are no users with the given id, response 2 is returned. If there is a user with the given id, but the requesting user has not upvoted it, response 3 is returned. Otherwise, response 3 is returned.

/post/add
Request type: POST
Required parameters: title type of String, summary type of String, description type of String, privacy type of Boolean, requirements type of String, tags type of list of strings, status type of Boolean, uploaded files.

Header: Bearer token for authorization
Responses:
1-) code: 201, message: “Post is created” body: postId type of Integer.
2-) code: 500, message: “Something went wrong” (String)

Description: /post/add endpoint is for the authorized users who want to create a new project with the fields given by themselves. The user sends all required parameters in the body of this post request. All information is saved to the relevant database tables. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/update/{postId}
Request type: PATCH
Required parameters: postId type of Integer, or title type of String, or summary type of String, description type of String, or privacy type of Boolean, or requirements type of String, or any combination of previous fields.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Post is updated” (String),
2-) code: 500, message: “Something went wrong” (String)

Description: /post/update/{postId} endpoint is for the authorized users who want to update their projects with the fields given by themselves. The user sends all fields they want to update in the body of this post request. The id of the post to be updated is sent in the url parameter. All information is saved to the relevant database tables. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/delete/{postId}
Request type: DELETE
Required parameters: postId type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Post is deleted”.
2-) code: 500, message: “Something went wrong”

Description: /post/delete/{postId} endpoint is for the authorized users who want to delete their own project whose id is given by themselves. The user sends the id of the post to be deleted in the url parameter of this delete request. The post is found and deleted from the database. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/get/{id}/{type}
Request type: GET
Required parameters: id type of Integer, type type of Boolean
Header: Bearer token for authorization
Responses:
1-) code: 200, body: list of projects type of Json array of Project Model.
2-) code: 200, body: project type of Project Model.
3-) code: 500, message: “Something went wrong”

Description: /post/get/{id}/{type} endpoint is for the authorized users who want to fetch all posts of a specific user whose id is given by themselves or to fetch the project whose id is given by themselves. The distinction is via type parameter. 0 means get the projects of the users whose id is given, 1 means get the project whose id is given. The posts are found fetched from the database. If type is 0 and no error occurs, response 1 is returned. If type is 1 and no error occurs, response 2 is returned. If any error is thrown, response 3 is returned.

/post/add_tag
Request type: POST
Required parameters: tags type of list of string, projectId type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 201, message: “Available tags are added” tagsAddedBefore type of list of strings.
2-) code: 500, message: “Something went wrong” (String)

Description: /post/add_tag endpoint is for the authorized users who want to add tags to their projects. The user sends all tags they want to add and the id of the project to which the tags are going to be added to in the body of this post request. All information is saved to the relevant database tables. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/delete_tag
Request type: DELETE
Required parameters: tags type of list of string, projectId type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 204
2-) code: 500, message: “Something went wrong” (String)

Description: /post/delete_tag endpoint is for the authorized users who want to remove tags from their projects. The user sends all tags they want to remove and the id of the project to which the tags are going to be added to in the body of this post request. All information is synced with the relevant database tables. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/add_milestone
Request type: POST
Required parameters: project_id type of Integer, date type of Date, title type of String, description type of String, type type of String
Header: Bearer token for authorization
Responses:
1-) code: 201, message: “Milestone is added” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /post/add_milestone endpoint is for the authorized users who want to add milestones to their projects. The user sends all the required information about the milestone in the body of this post request. All information is saved into the relevant database tables. If

no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/update_milestone
Request type: PATCH
Required parameters: milestone_id type of Integer, project_id type of Integer, or date type of Date, or title type of String, or description type of String, or type type of String, or any combination of the fields above.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Milestone is updated” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /post/update_milestone endpoint is for the authorized users who want to update the milestones they previously added to their projects. The user sends all the fields they want to update about the milestone in the body of this post request. All information is synced with the relevant database tables. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/post/delete_milestone/{id}
Request type: DELETE
Required parameters: id of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 204, message: “Milestone is deleted” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /post/delete_milestone/{id} endpoint is for the authorized users who want to delete milestones they previously added to their projects. The user sends the id of the milestone they want to delete in the url parameters of this post request. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/getTitles
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, result type of list of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/getTitles endpoint is for the authorized users who want to fetch predefined titles. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/getDepartments
Request type: GET
Required parameters:
Header: Bearer token for authorization

Responses:
1-) code: 200, result type of list of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/getDepartments endpoint is for the authorized users who want to fetch predefined departments. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/getTags
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, result type of list of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/getTags endpoint is for the authorized users who want to fetch predefined tags. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/getUniversities
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, result type of list of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/getUniversities endpoint is for the authorized users who want to fetch predefined universities. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/addTitle
Request type: POST
Required parameters: title of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Title is created” (String), title type of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/addTitle endpoint is for the authorized users who want to add a new title to the predefined titles. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/addDepartment
Request type: POST
Required parameters: department of type String
Header: Bearer token for authorization
Responses:

1-) code: 200, message: “Department is created” (String), department type of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/addDepartment endpoint is for the authorized users who want to add a new department to the predefined departments. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/addTag
Request type: POST
Required parameters: tag of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “tag is created” (String), tag type of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/addTag endpoint is for the authorized users who want to add a new tag to the predefined tag . If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/autoComplete/addUniversity
Request type: POST
Required parameters: university of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “university is created” (String), university type of String
2-) code: 500, message: “Something went wrong” (String)

Description: /autoComplete/addUniversity endpoint is for the authorized users who want to add a new university to the predefined university. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/follow/add
Request type: POST
Required parameters: userId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String)
2-) code: 400, message: “User not found ” (String)
3-) code: 400, message: “You are already following this user” (String)
4-) code: 500, message: “Something went wrong” (String)

Description: /follow/add endpoint is for the authorized users who want to follow another user. The user sends the user id of the user who they want to follow in the body of this request. If success, response 1 is returned. If there are no users with the given id, response 2 is returned. If there is a user with the given id, but the requesting user already followed him, response 3 is returned. If any error is thrown in the process above, response 4 is returned.

/follow/remove
Request type: POST
Required parameters: userId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String)
2-) code: 400, message: “User not found ” (String)
3-) code: 400, message: “You are not following this user” (String)
4-) code: 500, message: “Something went wrong” (String)

Description: /follow/remove endpoint is for the authorized users who want to unfollow another user. The user sends the user id of the user who they want to unfollow in the body of this request. If success, response 1 is returned. If there are no users with the given id, response 2 is returned. If there is a user with the given id, but the requesting user does not follow him, response 3 is returned. If any error is thrown in the process above, response 4 is returned.

/follow/followers
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String), data type of list of User Model
2-) code: 500, message: “Something went wrong” (String)

Description: /follow/followers endpoint is for the authorized users who want to fetch all the users who are following them. If success, response 1 is returned. If any error is thrown in the process above, response 4 is returned.
/follow/followings
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Successful” (String), data type of list of User Model
2-) code: 500, message: “Something went wrong” (String)

Description: /follow/followings endpoint is for the authorized users who want to fetch all the users they are following. If success, response 1 is returned. If any error is thrown in the process above, response 4 is returned.

/file/add
Request type: POST
Required parameters: projectId of type Integer, uploaded files.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “File is added” (String), id of type Integer
2-) code: 500, message: “Something went wrong” (String)

Description: /file/add endpoint is for the authorized users who want to add a new file to their projects. The user sends the related project id and files to be uploaded in the body of this post request. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/file/delete/{project_id}/{filename}
Request type: DELETE
Required parameters: projectId of type Integer, filename of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “File is deleted” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /file/delete/{project_id}/{filename} endpoint is for the authorized users who want to remove previously added files from their projects. The user sends the related project id and filename to be deleted in the query parameter of this post request. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/file/get/{project_id}/{filename}
Request type: GET
Required parameters: projectId of type Integer, filename of type String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “File is fetched ” (String), file of type File
2-) code: 500, message: “Something went wrong” (String)

Description: /file//get/{project_id}/{filename} endpoint is for the authorized users who want to fetch a file belonging to their projects. The user sends the related project id and the filename to be fetched in the url parameters of this get request. If no errors are thrown in the process above, response 1 is returned. If any error is thrown, response 2 is returned.

/collab/add_request
Request type: POST
Required parameters: array of CollabRequest model. the first element is requesterId, the second is requestedId, the third is projectId, the fourth is request Type
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Available collaborations are requested” (String)
2-) code: 400, message: “Some users are not found ” (String)
3-) code: 400, message: “But, you requested collaboration from these users before” (String)
4-) code: 500, message: “Something went wrong” (String)

Description: /collab/add_request endpoint is for the authorized users who want to send collaboration requests to other users. The user sends all required fields in the body of this request. If success, response 1 is returned. If there are no users with at least one of the given ids, response 2 is returned. If there is a user with at least one of the given ids who the requesting user has already collaborated with, response 3 is returned. If any error is thrown in the process above, response 4 is returned.

/collab/delete_request/{id}
Request type: DELETE
Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Operation is completed” (String)
2-) code: 500, message: “ooop” (String)

Description: /collab/delete_request/{id} endpoint is for the authorized users who want to undo their collaboration requests to other users. The user sends all required fields in the query parameters of this request. If success, response 1 is returned. If any error is thrown in the process above, response 4 is returned.

/collab/get_requests
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Operation is completed” (String), requests type of list of Request Model
2-) code: 500, message: “Something went wrong” (String)

Description: /collab/get_requests endpoint is for the authorized users who want to see their collaboration requests. If success, response 1 is returned. If any error is thrown in the process above, response 4 is returned.

/collab/add_collaborator
Request type: POST
Required parameters: projectId of type Integer, userId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 204
2-) code: 500, message: “Request is no longer available” (String)
3-) code: 500, message: “Project does not exist” (String)
4-) code: 500, message: “Something went wrong” (String)

Description: /collab/add_collaborator endpoint is for the authorized users who want to add other users to their projects as collaborators. If success, response 1 is returned. If the project does not exist, response 2 is returned. If the request has been withdrawn, response 2 is returned. If any error is thrown in the process above, response 4 is returned.

/collab/delete_collaborator/{projectId}/{collaboratorId}
Request type: DELETE
Required parameters: projectId of type Integer, collaboratorId of type Integer
Header: Bearer token for authorization
Responses:
1-) code: 201

2-) code: 500, message: “Something went wrong” (String)

Description: /collab/delete_collaborator/{projectId}/{collaboratorId} endpoint is for the authorized users who want to remove other users from collaborating with their projects. If success, response 1 is returned. If any error is thrown in the process above, response 4 is returned.

/home/posts
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, byFollowings of type list of Project Model, byUserTags of type list of Project Model
2-) code: 500, message: “Something went wrong” (String)

Description: /home/posts endpoint is for the authorized users who want to fetch the posts recommended for themselves. If success, response 1 is returned. If any error is thrown in the process above, response 2 is returned.

/home/users
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, similarInterests of type list of User Model, sameUniversity of type list of User Model, sameDepartment of type list of User Model
2-) code: 500, message: “Something went wrong” (String)

Description: /home/users endpoint is for the authorized users who want to fetch the users recommended for themselves. If success, response 1 is returned. If any error is thrown in the process above, response 2 is returned.

/home/delete
Request type: POST
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “User deleted successfully” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /home/delete endpoint is for the authorized users who want to delete their account in an irreversible way. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/add
Request type: POST
Required parameters: type type of String, isPublic type of Boolean, title type of String, body type of String, link type of String, location type of String, date type of Date, other type of String, tags type of list of String
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Event is created” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /event/add endpoint is for the authorized users who want to create a new event with the specifications given in this post request’s body. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/all
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, result: list of Events type of Event Model
2-) code: 500, message: “Something went wrong” (String)

Description: /event/all endpoint is for the authorized users who want to fetch all the events that they can see. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/{id}
Request type: GET
Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, result: event type of Event Model
2-) code: 500, message: “Something went wrong” (String)

Description: /event/{id} endpoint is for the authorized users who want to fetch specifications of the event whose id is given in the url parameters of this GET request.. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/search
Request type: GET
Required parameters: a list of filters which consist of field-query pairs.
Header: Bearer token for authorization
Responses:
1-) code: 200, result: list of Events type of Event Model
2-) code: 500, message: “Something went wrong” (String)

Description: /event/search endpoint is for the authorized users who want to fetch all the events that they can see and filtered by the parameters given in the body of this get request. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/update/{id}
Request type: PATCH
Required parameters: type type of String, isPublic type of Boolean, title type of String, body type of String, link type of String, location type of String, date type of Date, other type of String, tags type of list of String, or any combination of these.
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /event/update/{id} endpoint is for the authorized users who want to update some specifications of their events. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/unfav
Request type: POST
Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /event/unfav endpoint is for the authorized users who want to remove the event whose id is given in the body of this post request from their favorites. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/fav
Request type: POST
Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /event/fav endpoint is for the authorized users who want to add the event whose id is given in the body of this post request to their favorites. If success, response 1 is returned. If any error occurs, response 2 is returned.

/event/delete
Request type: POST

Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /event/delete endpoint is for the authorized users who want to delete their event permanently. If success, response 1 is returned. If any error occurs, response 2 is returned.

/notification/add_rejection
Request type: POST
Required parameters: projectId type of Integer, rejectedId type of Integer, requestId type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /notification/add_rejection endpoint is for the authorized users who want to reject the collaboration request whose fields are given in the body of this post request. If success, response 1 is returned. If any error occurs, response 2 is returned.

/notification/get
Request type: GET
Required parameters:
Header: Bearer token for authorization
Responses:
1-) code: 200, result: a list of notifications type of Notification Model
2-) code: 500, message: “Something went wrong” (String)

Description: /notification/get endpoint is for the authorized users who want to fetch all notifications issued for them. If success, response 1 is returned. If any error occurs, response 2 is returned.
/notification/delete/{id}
Request type: DELETE
Required parameters: id type of Integer
Header: Bearer token for authorization
Responses:
1-) code: 200, message: “Success” (String)
2-) code: 500, message: “Something went wrong” (String)

Description: /notification/delete/{id} endpoint is for the authorized users who want to delete the notification whose id is given in the url parameters of this delete request.. If success, response 1 is returned. If any error occurs, response 2 is returned.

Project Plan

Milestone 1 Date: 24.11.2020
Milestone 2 Date: 29.12.2020
Final Milestone Date: 19.01.2021

For the Final Milestone we decided to complete all the tasks.

Frontend

inviting a user, accepting and rejecting requests	Ali Ramazan Mert
seeing , accepting, and rejecting requests	Emilcan Arıcan, Ali Ramazan Mert
Logout	Ali Ramazan Mert
status of papers/projects	Emilcan Arıcan
google scholar linking	Ali Ramazan Mert
file related pages and file edit / update / delete functionalities	Cemre Efe Karakaş
filtering system (home page recommendation)	Emilcan Arıcan, Cemre Efe Karakaş
follow and up functionality	Ali Ramazan Mert
notification system	Emilcan Arıcan, Ali Ramazan Mert
Updating Profile Page	Ali Ramazan Mert
Project Details Page	Emilcan Arıcan
Project edit page	Cemre Efe Karakaş
Followers and followings page	Cemre Efe Karakaş

Backend

inviting a user, accepting and rejecting requests	Hazer Babur
seeing , accepting, and rejecting requests	Hazer Babur

status of papers/projects	Hazer Babur
google scholar linking	Muhammed Enes Toptaş
adding profile picture	Muhammed Enes Toptaş
document preparation	Hazer Babur
filtering system (home page recommendation)	Hazer Babur, Hamza Işıktaş, M. Enes Toptaş
follow and up functionality	Hamza Işıktaş
notification system	Hazer Babur
create database V2	Ömer Faruk Özdemir
Auto Completion	Ömer Faruk Özdemir
create database V3	Ömer Faruk Özdemir
create event get&add endpoints	Ömer Faruk Özdemir
create notification get&add endpoints	Ömer Faruk Özdemir

Mobile

accepting and rejecting requests	Ezgi Gülperi Er, Doğukan Kalkan
seeing , accepting, and rejecting requests	Ezgi Gülperi Er, Doğukan Kalkan, Cihat Kapusuz
Logout	Ezgi Gülperi Er
status of papers/projects	Gülsüm Tuba Çibuk, Cihat Kapusuz
google scholar linking	Gülsüm Tuba Çibuk
document preparation	Cihat Kapusuz
filtering system (home page recommendation)	Gülsüm Tuba Çibuk
follow and up functionality	Doğukan Kalkan
link to the project owner's profile when project details viewed from search	Cihat Kapusuz
edit profile & change profile picture	Doğukan Kalkan
Final Milestone	

edit project page	Cihat Kapusuz
events page	Doğukan Kalkan
notifications	Gülsüm Tuba Çibuk
invitations page	Ezgi Gülperi Er

	Name	Resource Na...	Durat...	Start
1	Backend implementation of login with name, surname, unique email address and password	Backend Team	5 days	11/3/20 8:00 AM
2	Frontend implementation of login with name, surname, unique email address and password	Frontend Team	5 days	11/3/20 8:00 AM
3	Mobile implementation of login with name, surname, unique email address and password	Mobile Team	5 days	11/3/20 8:00 AM
4	Backend implementation of tagging system regarding research interests	Backend Team	5 days	11/3/20 8:00 AM
5	Frontend implementation of tagging system regarding research interests	Frontend Team	5 days	11/3/20 8:00 AM
6	Mobile implementation of tagging system regarding research interests	Mobile Team	5 days	11/3/20 8:00 AM
7	Backend implementation of the part about the research area, recent publications and affiliation.	Backend Team	5 days	11/3/20 8:00 AM
8	Frontend implementation of the part about the research area, recent publications and affiliation.	Frontend Team	5 days	11/3/20 8:00 AM
9	Mobile implementation of the part about the research area, recent publications and affiliation.	Mobile Team	5 days	11/3/20 8:00 AM
10	Deployment of backend	Backend Team	5 days	11/3/20 8:00 AM
11	Backend implementation of entering information of the project being created	Backend Team	5 days	11/10/20 8:00 AM
12	Frontend implementation of entering information of the project being created	Frontend Team	5 days	11/10/20 8:00 AM
13	Mobile implementation of entering information of the project being created	Mobile Team	5 days	11/10/20 8:00 AM
14	Backend implementation of creating and editing paper/project	Backend Team	5 days	11/10/20 8:00 AM
15	Frontend implementation of creating and editing paper/project	Frontend Team	5 days	11/10/20 8:00 AM
16	Mobile implementation of creating and editing paper/project	Mobile Team	5 days	11/10/20 8:00 AM
17	Backend implemention of search	Backend Team	5 days	11/10/20 8:00 AM
18	Frontend implemention of search	Frontend Team	5 days	11/10/20 8:00 AM
19	Mobile implemention of search	Mobile Team	5 days	11/10/20 8:00 AM
20	Backend implementation of inviting a user, accepting and rejecting requests	Backend Team	5 days	11/17/20 8:00 AM
21	Frontend implementation of inviting a user, accepting and rejecting requests	Frontend Team	5 days	11/17/20 8:00 AM
22	Mobile implementation of inviting a user, accepting and rejecting requests	Mobile Team	5 days	11/17/20 8:00 AM
23	Backend implementation of seeing, accepting and rejecting invitations	Backend Team	5 days	11/17/20 8:00 AM
24	Frontend iimplementation of seeing, accepting and rejecting invitations	Frontend Team	5 days	11/17/20 8:00 AM
25	Mobile implementation of seeing, accepting and rejecting invitations	Mobile Team	5 days	11/17/20 8:00 AM
26	Backend implemention of entering affiliation	Backend Team	5 days	11/17/20 8:00 AM
27	Frontend implemention of entering affiliation	Frontend Team	5 days	11/17/20 8:00 AM
28	Mobile implemention of entering affiliation	Mobile Team	5 days	11/17/20 8:00 AM
29	Backend implementation of log out	Backend Team	5 days	11/24/20 8:00 AM
30	Frontend implementation of log out	Frontend Team	5 days	11/24/20 8:00 AM
31	Mobile implementation of log out	Mobile Team	5 days	11/24/20 8:00 AM
32	Backend implementation of seeing the status of paper/project	Backend Team	5 days	11/24/20 8:00 AM
33	Frontend implementation of seeing the status of paper/project	Frontend Team	5 days	11/24/20 8:00 AM
34	Mobile implementation of seeing the status of paper/project	Mobile Team	5 days	11/24/20 8:00 AM
35	Backend implementation of home page	Backend Team	5 days	11/24/20 8:00 AM
36	Frontend implemention of home page	Frontend Team	5 days	11/24/20 8:00 AM
37	Mobile implemention of home page	Mobile Team	5 days	11/24/20 8:00 AM
38	Preparation of Milestone document	Backend Team...	5 days	12/1/20 8:00 AM
39	Backend implementation of e-mail validation	Backend Team	5 days	12/8/20 8:00 AM
40	Frontend implementation of e-mail validation	Frontend Team	5 days	12/8/20 8:00 AM
41	Mobile implementation of e-mail validation	Mobile Team	5 days	12/8/20 8:00 AM
42	Backend implementation of linking Google Scholar and ResearchGate accounts.	Backend Team	5 days	12/8/20 8:00 AM

43	Frontend implementation of linking Google Scholar and ResearchGate accounts.	Frontend Team	5 days	12/8/20 8:00 AM
44	Mobile implementation of linking Google Scholar and ResearchGate accounts.	Mobile Team	5 days	12/8/20 8:00 AM
45	Backend implementation of simultaneous document preparation	Backend Team	5 days	12/8/20 8:00 AM
46	Frontend implementation of simultaneous document preparation	Frontend Team	5 days	12/8/20 8:00 AM
47	Mobile implementation of simultaneous document preparation	Mobile Team	5 days	12/8/20 8:00 AM
48	Backend impementation of filtering system	Backend Team	5 days	12/15/20 8:00 AM
49	Frontend impementation of filtering system	Frontend Team	5 days	12/15/20 8:00 AM
50	Mobile impementation of filtering system	Mobile Team	5 days	12/15/20 8:00 AM
51	Backend Implementation of events page	Backend Team	5 days	12/15/20 8:00 AM
52	Frontend Implementation of events page	Frontend Team	5 days	12/15/20 8:00 AM
53	Mobile Implementation of events page	Mobile Team	5 days	12/15/20 8:00 AM
54	Backend implementation of seeing events detailed	Backend Team	5 days	12/22/20 8:00 AM
55	Frontend implementation of seeing events detailed	Frontend Team	5 days	12/22/20 8:00 AM
56	Mobile implementation of seeing events detailed	Mobile Team	5 days	12/22/20 8:00 AM
57	Backend implementation of notification system	Backend Team	5 days	12/22/20 8:00 AM
58	Frontend implementation of notification system	Frontend Team	5 days	12/22/20 8:00 AM
59	Mobile implementation of notification system	Mobile Team	5 days	12/22/20 8:00 AM
60	Preparation of Milestone 2 document	Backend Team...	5 days	12/29/20 8:00 AM
61	Backend implementation of editing all information entered by the user	Backend Team	5 days	1/5/21 8:00 AM
62	Frontend implementation of editing all information entered by the user	Frontend Team	5 days	1/5/21 8:00 AM
63	Mobile implementation of editing all information entered by the user	Mobile Team	5 days	1/5/21 8:00 AM
64	Backend implementation of tracking information shared by other collaborating users	Backend Team	5 days	1/5/21 8:00 AM
65	Frontend implementation of tracking information shared by other collaborating users	Frontend Team	5 days	1/5/21 8:00 AM
66	Mobile implementation of tracking information shared by other collaborating users	Mobile Team	5 days	1/5/21 8:00 AM

67	Backend implementation semantic search	Backend Team	5 days	1/5/21 8:00 AM
68	Frontend implementation semantic search	Frontend Team	5 days	1/5/21 8:00 AM
69	Mobile implementation semantic search	Mobile Team	5 days	1/5/21 8:00 AM
70	Backend implementation of account delete	Backend Team	5 days	1/12/21 8:00 AM
71	Frontend implementation of account delete	Frontend Team	5 days	1/12/21 8:00 AM
72	Mobile implementation of account delete	Mobile Team	5 days	1/12/21 8:00 AM

User Scenarios

Two collaborators on Akademise

1. Frontend

Inés M. Bayern logs into akademise, of which she is already a member. Upon logging in she notices she doesn't have anything on her homepage and notices the prompt telling her to add interest areas or follow people to get content. So she goes to her profile. She adds an avatar, she adds some interest areas, and she edits her bio.

She returns to the homepage to see new content. She sees a project by another professor (Ethan Alpatian). She clicks on the project to see more details. She likes the project and decides to check out the project owner. She clicks on the owner's name to go into his profile. She sees that they are a respectable scholar (when she sees that they have many citations on google scholars) and decides to follow them. She returns to the project and sends a join request. She thinks it would be beneficial for her to link her own google scholars profile to appeal to possible collaborators.

2. Mobile

Hi, I am Tugba Çabuk. I am a researcher graduated from Boğaziçi University. I am looking for collaborators for my projects. I check my notifications to see if there is any collaboration request. I see that “Emilcan Arican” followed me and requested a collaboration for my project “Generative Art”. Ali Mert accepted my invitation to my “Generative Art” project. I do not know who is “Emilcan Arican” so I go to his profile by clicking on his name. I see that he is a “Ph.D. student” from “Boğaziçi University”. His research area is “Mathematics”. I want to see his Google Scholar. I turn back to his profile and I upvote him because of his efforts. I turn back to notifications. I click on the following notification so that I can delete it. I learned that “Emilcan Arican” requests collaboration for my “Generative Art”. I go to my “Generative Art” project and click on the requests and invitations button. I accept the request. I invite Tuba Çibuk to my project. She is a student that I know well. Then I turn back to my project. I want to add a new tag to my project. I look at the existing tags and can not find the tag I want. So I write it. Then I delete a tag that I find unnecessary. Then I want to add a new milestone. I add a new one and update the date of the existing one. Because I think that the project will not be ready at that time. Then I search for “Communication” because I am interested in that topic. However, I do not have enough knowledge to start a new project so I want to join a project. Then I click on a project. I see that owner of this project is also Emilcan Arican. It is a nice coincidence. Then I request a collaboration. Then I go to the events page. I find some events and I want to attend a conference”. It has a website link so that I can get further information. I add it to my favs. Then I look at my events and create a new one. Then I close my app and start to play chess.

3. Frontend Again

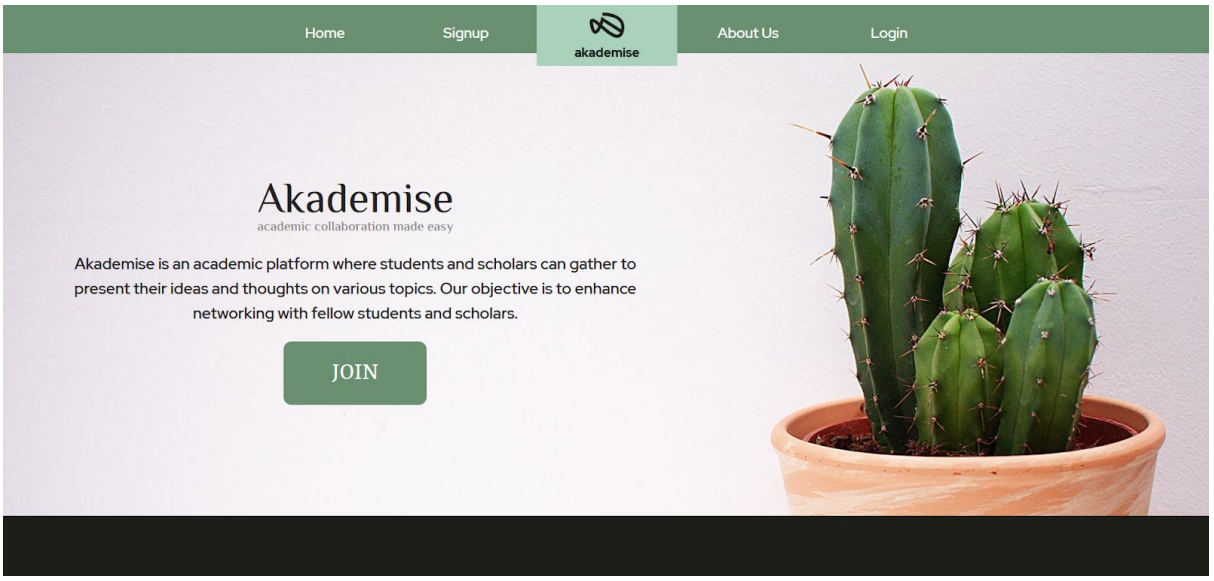
Inés sees a notification indicating that Ethan wants to join one of her projects. She sees that he accepted his collaboration request too. When she realizes he also followed her, she decides to give him an "up" on his profile as a sign of gratitude. She opens up the project to which she applied to be a collaborator and sees the project files. She notices that there is a file called "ContactInformation.txt" and clicks on it to see that it has the phone numbers of the collaborators. She edits the file and hits save changes. Now her information is on that file in the server too.

She goes into her profile and remembers that she has a project live on Akademise that she no longer plans to complete. She goes into the project's details page and then into the project edit page. She clicks delete, but is asked with a prompt if she is sure. At this point she decides that it would be better to mark the project as cancelled. She does just that and saves changes.

User Manuals

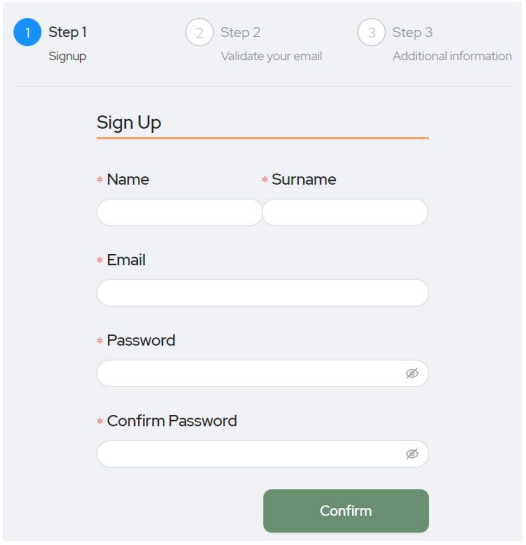
Frontend User Manual

Landing Page



This page is here to welcome users. Users can register by clicking the **Join** button. By clicking on the **Login** button at the top right, users can display a login modal and can login the platform by providing their email address and password. There is also a **forgot password** link that redirects users to the change password page. Users can see a brief explanation about the site and the developers by clicking on **About Us** link.

Sign Up Page



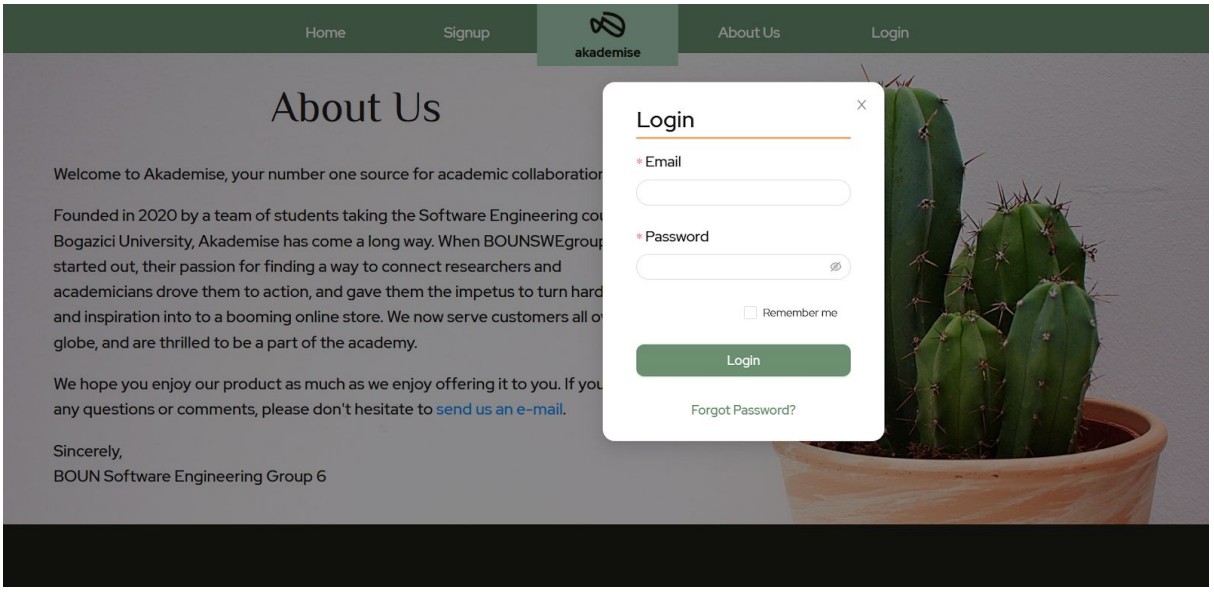
In the Sign Up page, firstly user should fill in the necessary personal information to create a new profile. These are **email address**, **password**, **name** and **surname**. If email is not in the correct format, “Given email is not valid” error will be seen and user can not

proceed without providing a valid email address. After filling these up, user clicks **Confirm** button, which will navigate the user to the validation step.

In the validation page, user enters the validation code that has been sent to his/her registered email address. And clicks next to proceed to next steps of signing up.

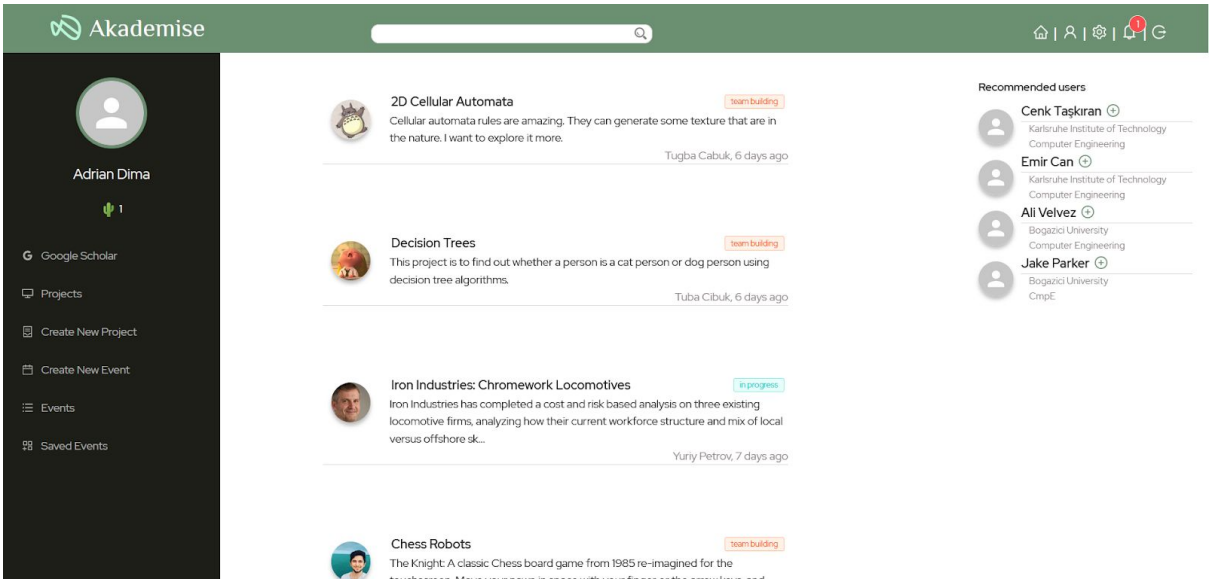
In the Additional Information part, user fills up information about his/her academic background and research interests . She/he can choose the from the dropdown list or if it is not listed, she can add her own through typing her information in the text spaces.

Login



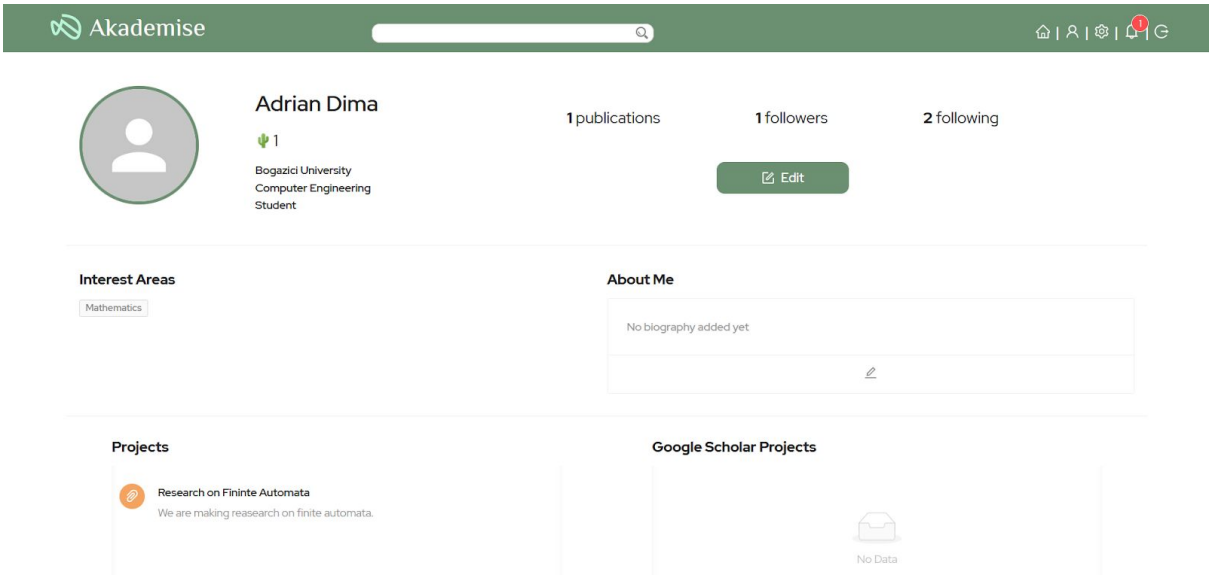
The login window consists of an email entry part, a password entry part, a login button, and a forgot password choice. Email and password must be provided correctly to enter the platform. If they are entered wrongly “Wrong email or password. Try again.” shows up. After filling in the email and password part, the login button is pressed and redirects to the home page.

Home Page

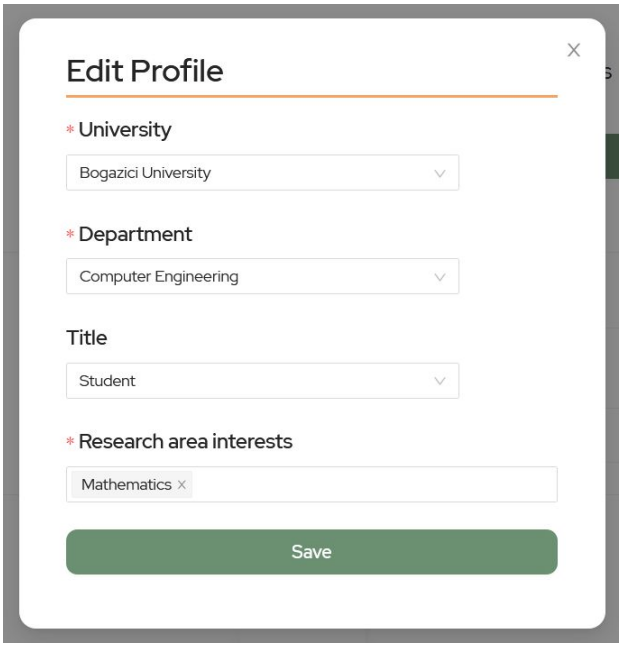


When users are logged in, the first page they land on is their home page. First thing to notice here is the feed. The content on the home page is determined by the users' interests and projects of users that are followed by them. There are also some user recommendations at the right side, users can follow those users by clicking on follow button or click on the name of the recommended user's name and navigate to the profile of them.

Profile Page

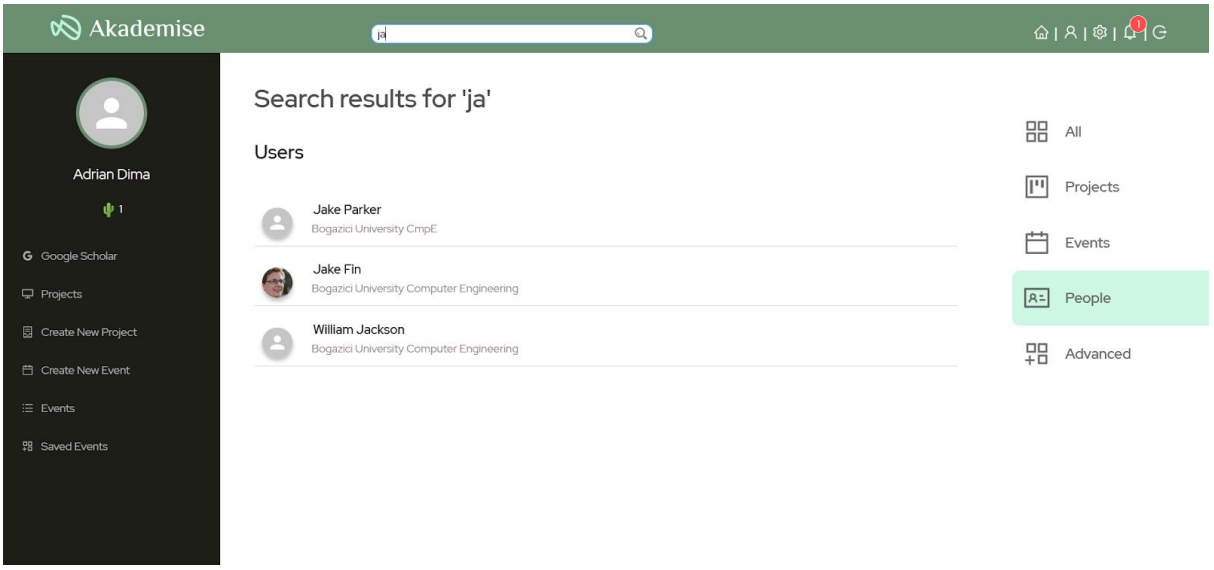


Here is the profile page. At this page, there is quite a lot of information about the user. Let's cover them one bite at a time. First of all, at the top left there is a name, profile picture, cactus score (number of ups), and affiliation. To upload a profile picture to the system user can click on to the profile picture. Then we see publication, follower and following numbers. Users' both Akademise projects and scholar projects are displayed. There is also a biography part, users can edit their own biography by simply clicking the edit button below the "About Me" text box. There are also interest areas of the user is displayed as well, by clicking those tags users can see related projects at the Akademise platform.



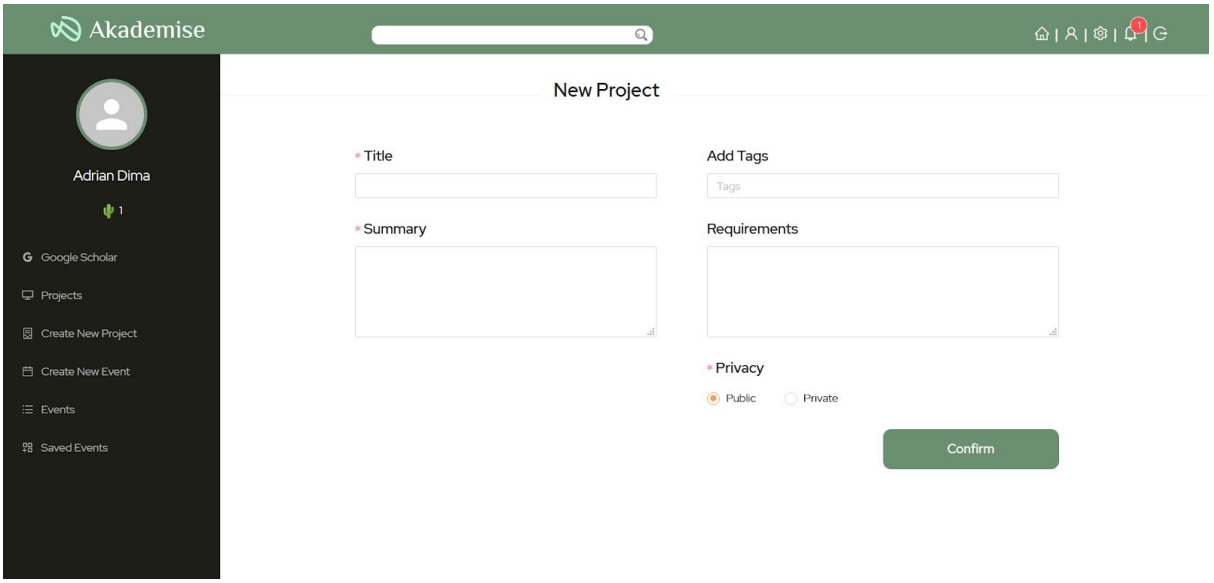
By clicking the edit button users can edit their profile in more detail. They can add or remove interests and change their university, department, and title.

Search



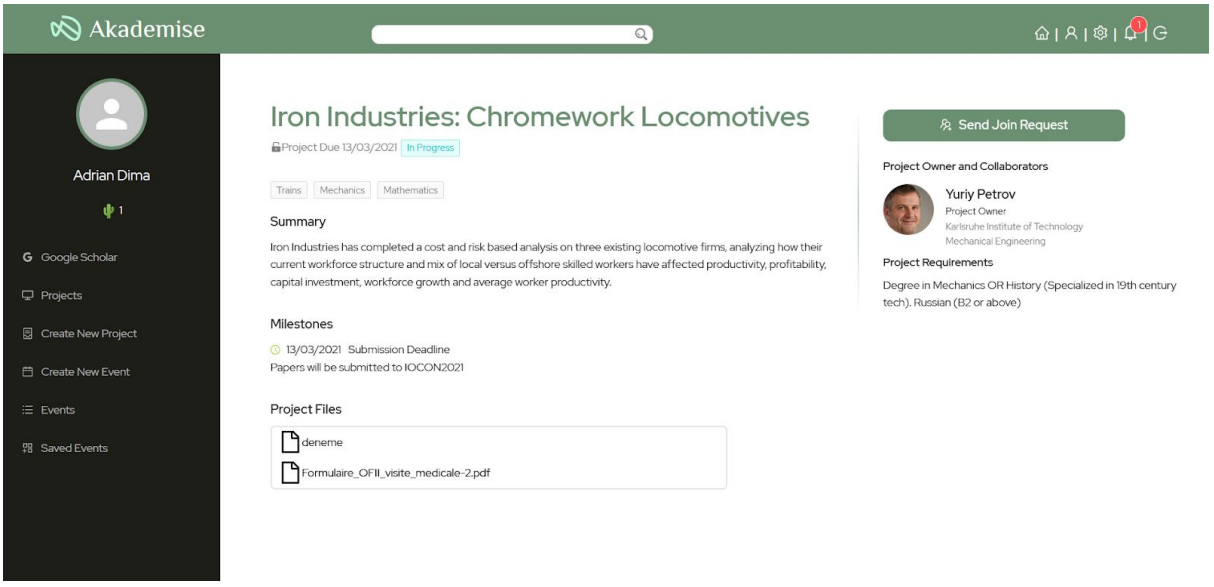
When users perform a search by using the search bar at the header, they are redirected to this page to show search results. There are several search types (all, people, project, event, and advanced) and for each of those there are related buttons at the right side of the screen. When users select a search type, results are filtered accordingly. If users want to make a more advanced search, they can choose the advanced search type. So that they can filter results by interest tags as well.

Project Create



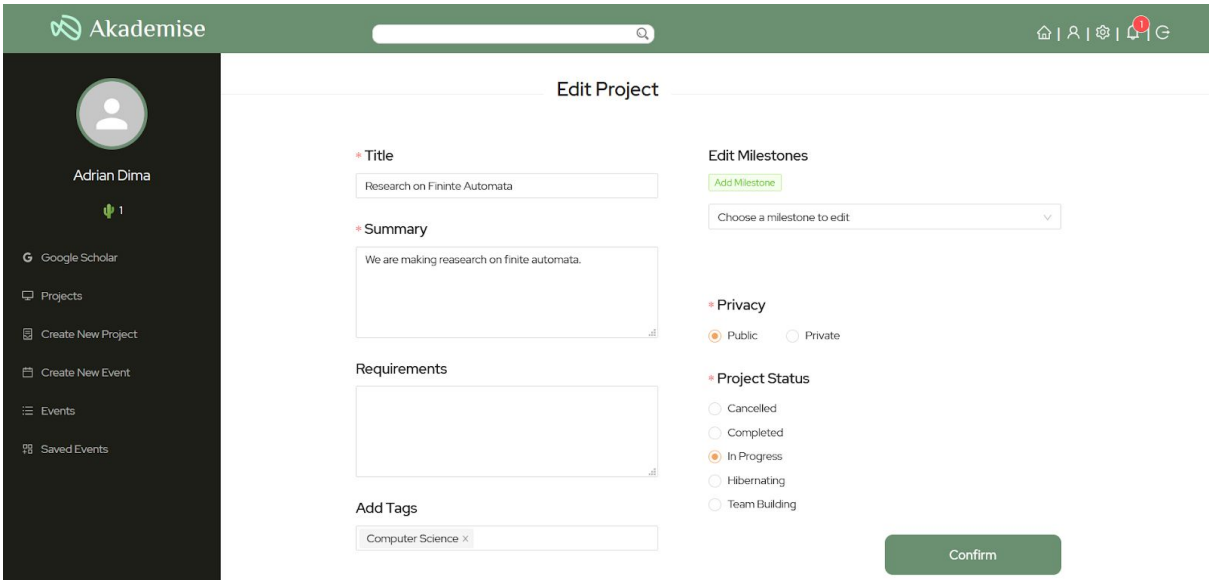
To create a new project user must enter the necessary fields for creating a project. They can navigate to this page from the sidebar menu by clicking the **Create New Project** button. In this part, the user must enter the project title, summary and the privacy of the project. Optionally, users also can enter tags and requirements. Then, click the **Confirm** button to proceed to the next page.

Project Details



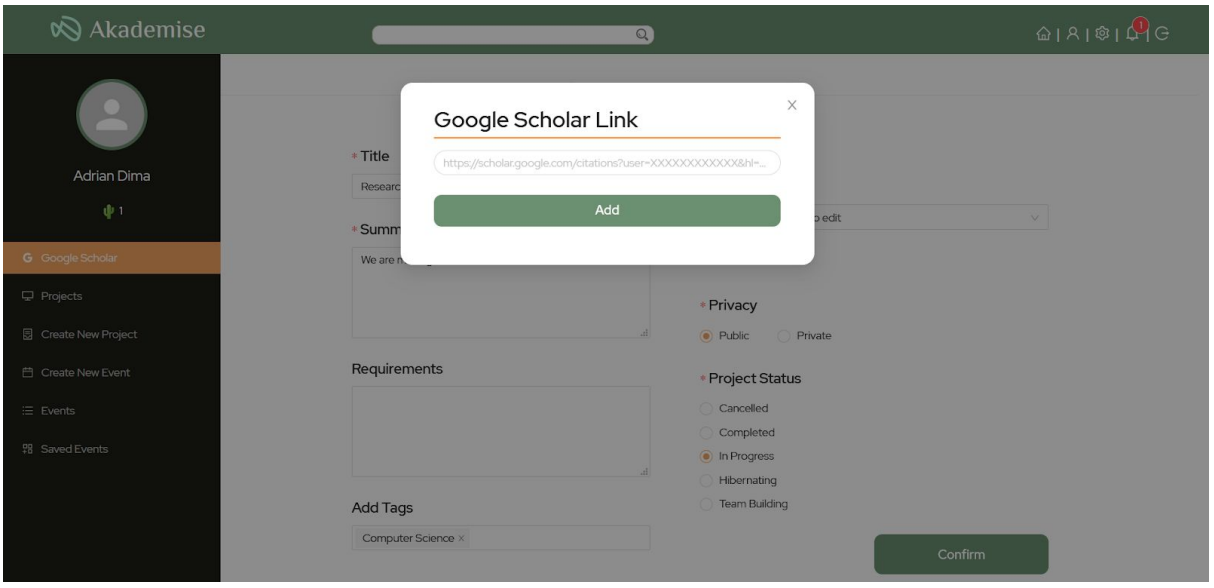
At this page users can display details of the projects such as summary, requirements, collaborators, milestones, and due date of the project. If they are not already a collaborator in the project, they can send a join request by clicking the top right button. If they are already a collaborator in this project, there are edit and invite buttons at the top right side of the screen. By using those, users can either navigate to the edit project page or open a batch invite dialogue where they can select multiple users to invite to their project.

Edit Project



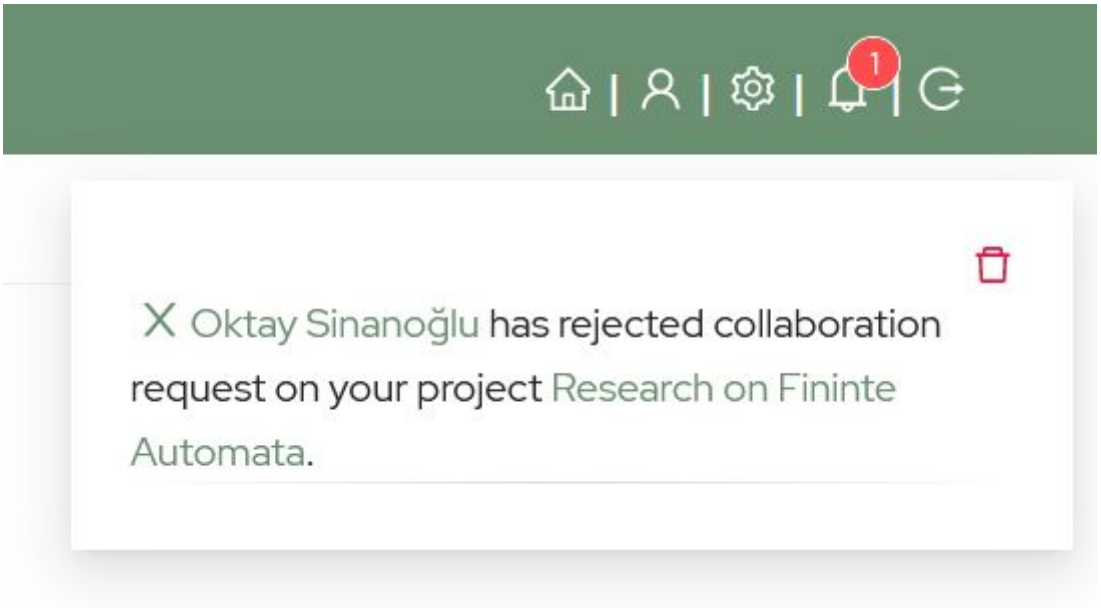
Users can change fields that are filled in the creation page in their own project. In addition to that, users can also Add milestones and change project status. By clicking the **Confirm** button all changes saved and redirects to the project page.

Google Scholar Link



Users can provide their google scholar link in order to display their google scholar projects on their profile page as well. To open this add google scholar link dialogue, users can click on the **Google Scholar** button on the profile sider that is at the left side of the page.

Notifications



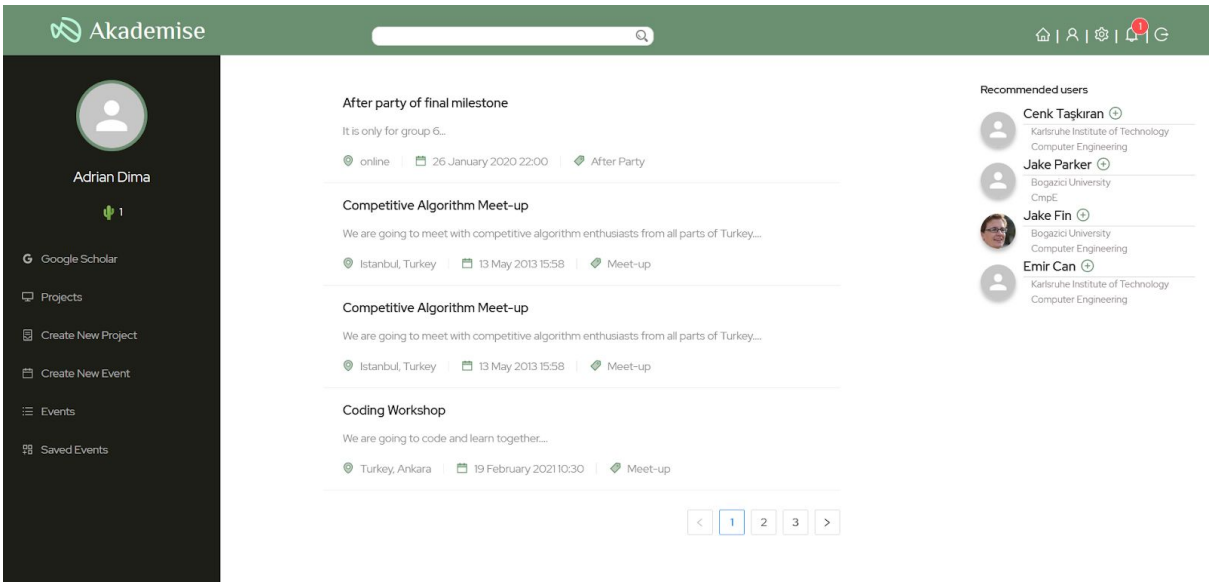
Notifications could be about:

-Being followed by a user
-being accepted or rejected to a project
-being invited to a project
-acceptance or rejection of an invitation
-being removed from a project
-collaboration requests

-collaboration invitations are shown on the notifications dialog.

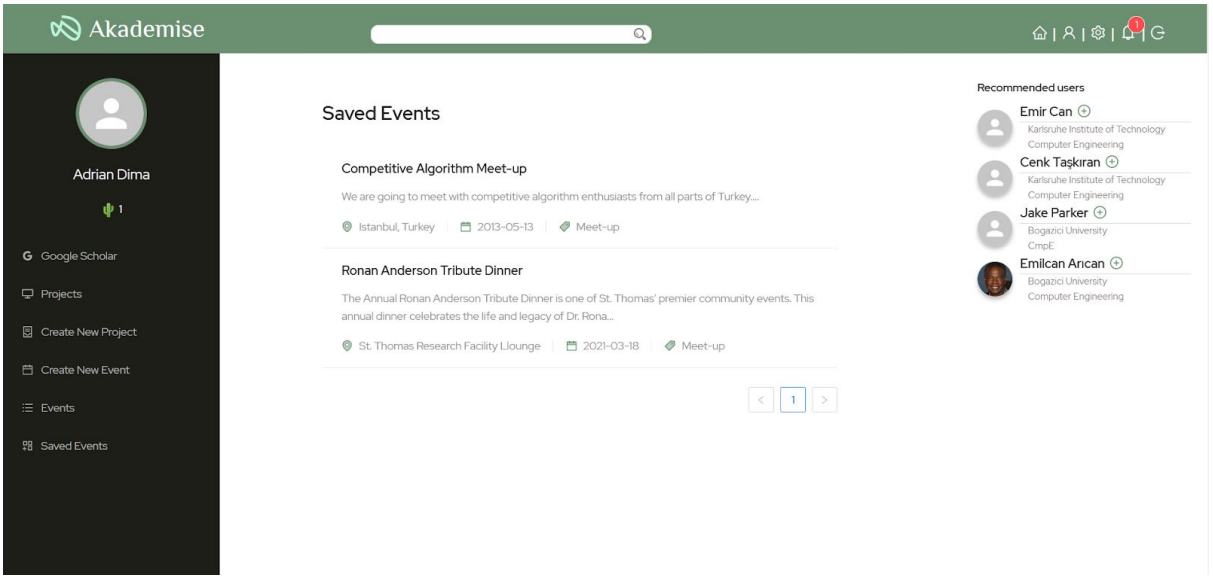
Users can click the user or project to see details and remove the notification by clicking red icon.

Events



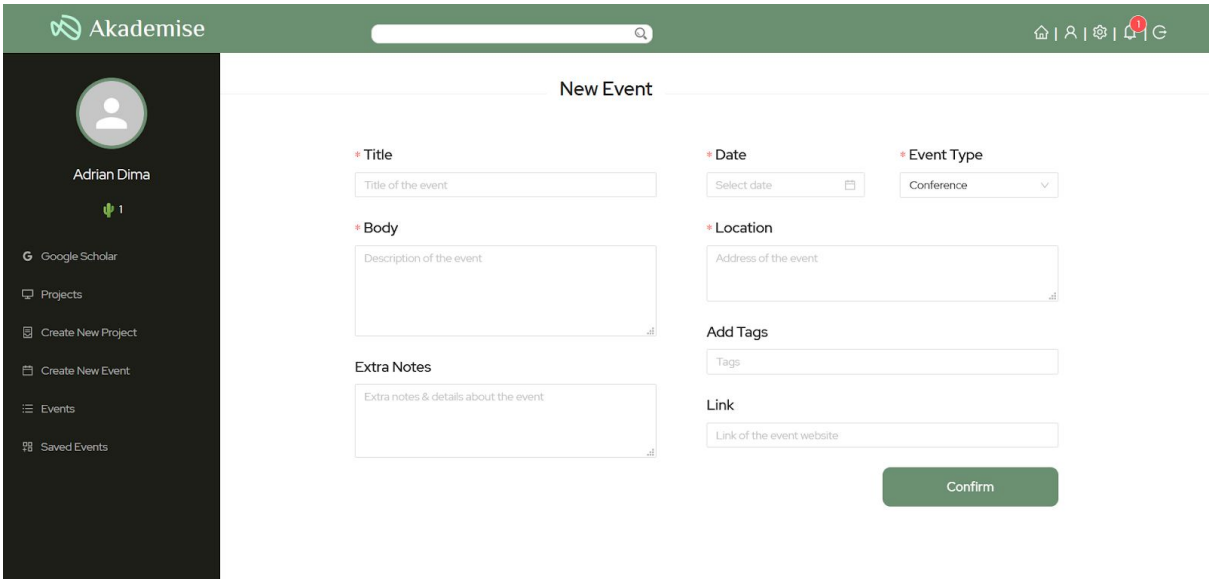
A user can see all events by clicking **Events** in the sidebar menu. In the events page, All events are listed and each event entry has its title, brief description, location and date and type. Pagination is used and divides multiple events to pages. Users can click page number bottom of the events and see the other events.

Saved Events



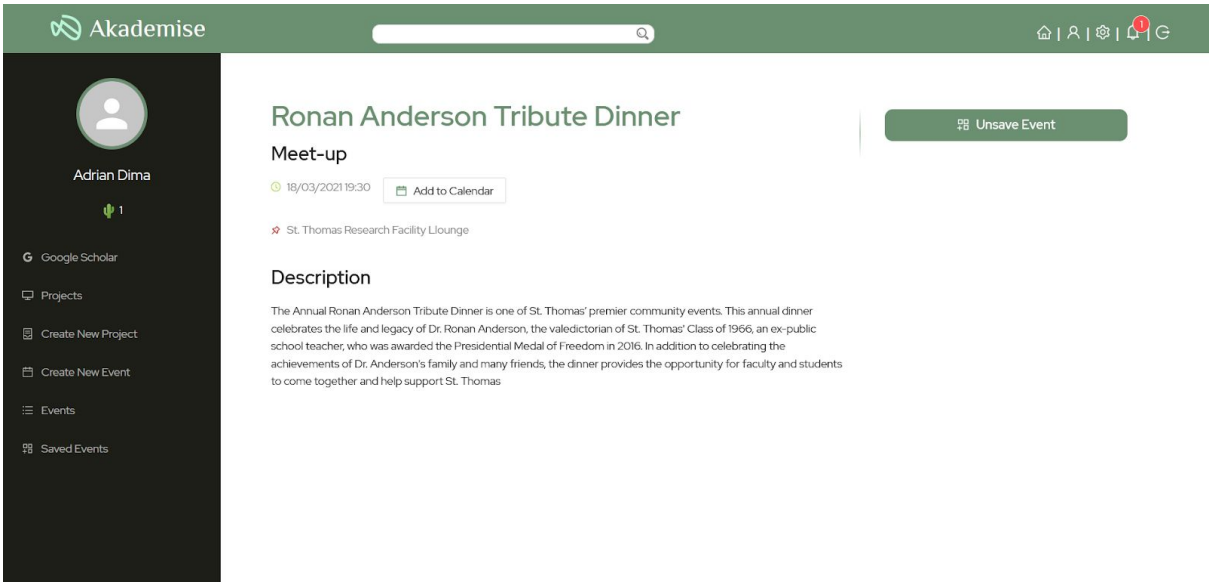
Users can navigate to this page by using the sidebar menu. As in the events page, there are brief descriptions of the events. However in this page displayed events are only the ones that are saved by the user.

Event Create



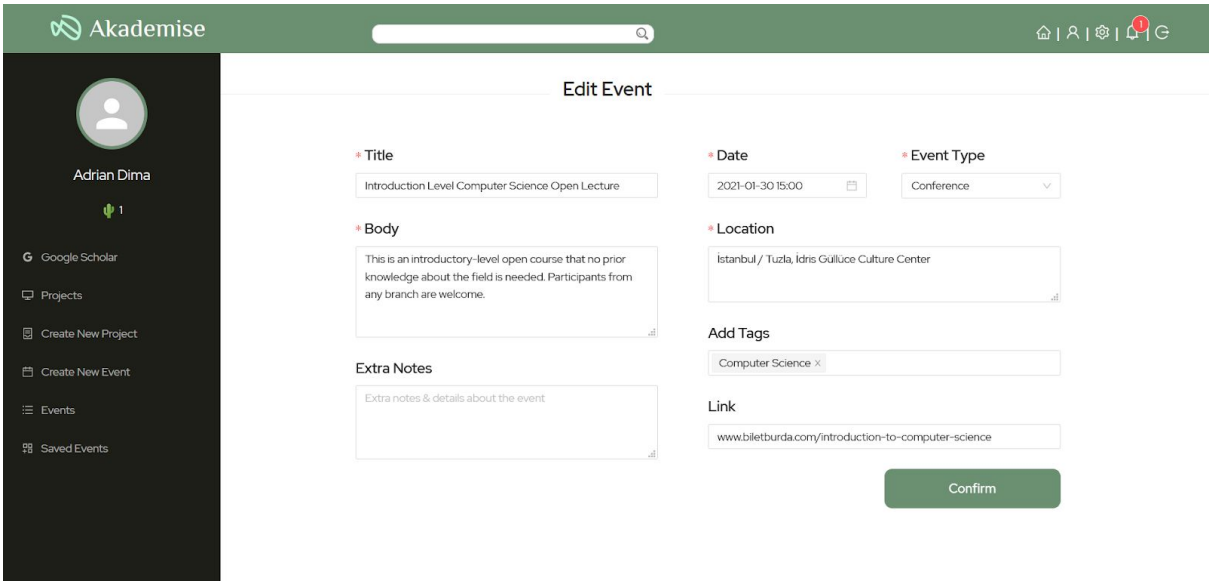
When creating an event, all the necessary information can be written in the corresponding fields, and the user can also set the privacy of the events as well as the tags s/he desires. Note that the user must enter the date with a specific time.

Event Details



When users need to see more detailed information about the event, they can check this event's details page. To navigate this page users can simply click on the event name. At this page, all the information of the events are displayed namely title, description, extra notes, date, type, location, and the link of the event. If users want to add this event to the saved events, there is a **Save Event** button top right corner of the screen. If the event is already a saved one there will be a **Unsave Event** button instead.

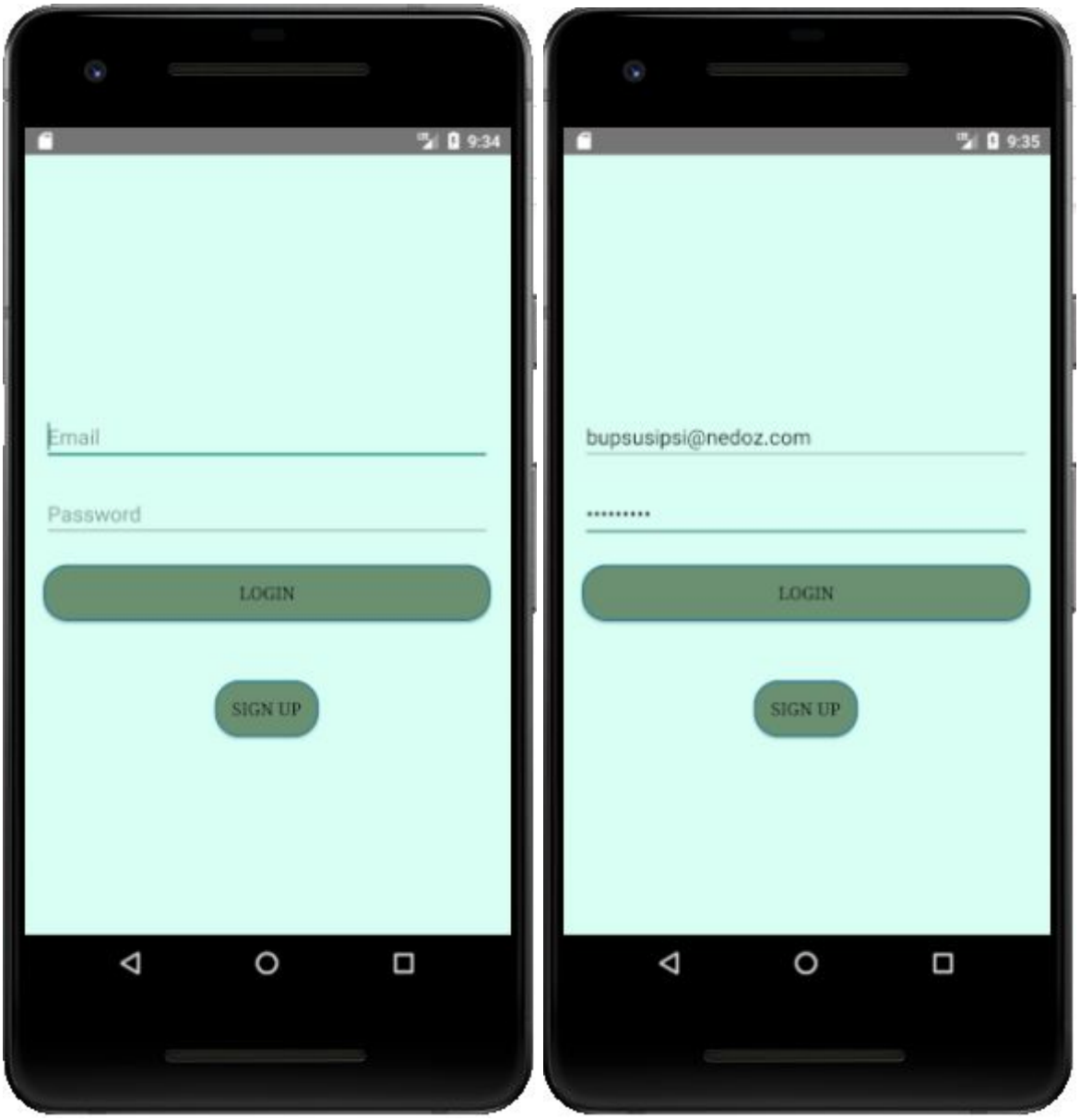
Edit Event



The edit event page is almost identical with the event creation page. It comes in a prefilled form by the former data of the event. Users can change the fields and confirm the changes. After they hit confirm changes fields are replaced by the former ones.

Android User Manual

Login Page

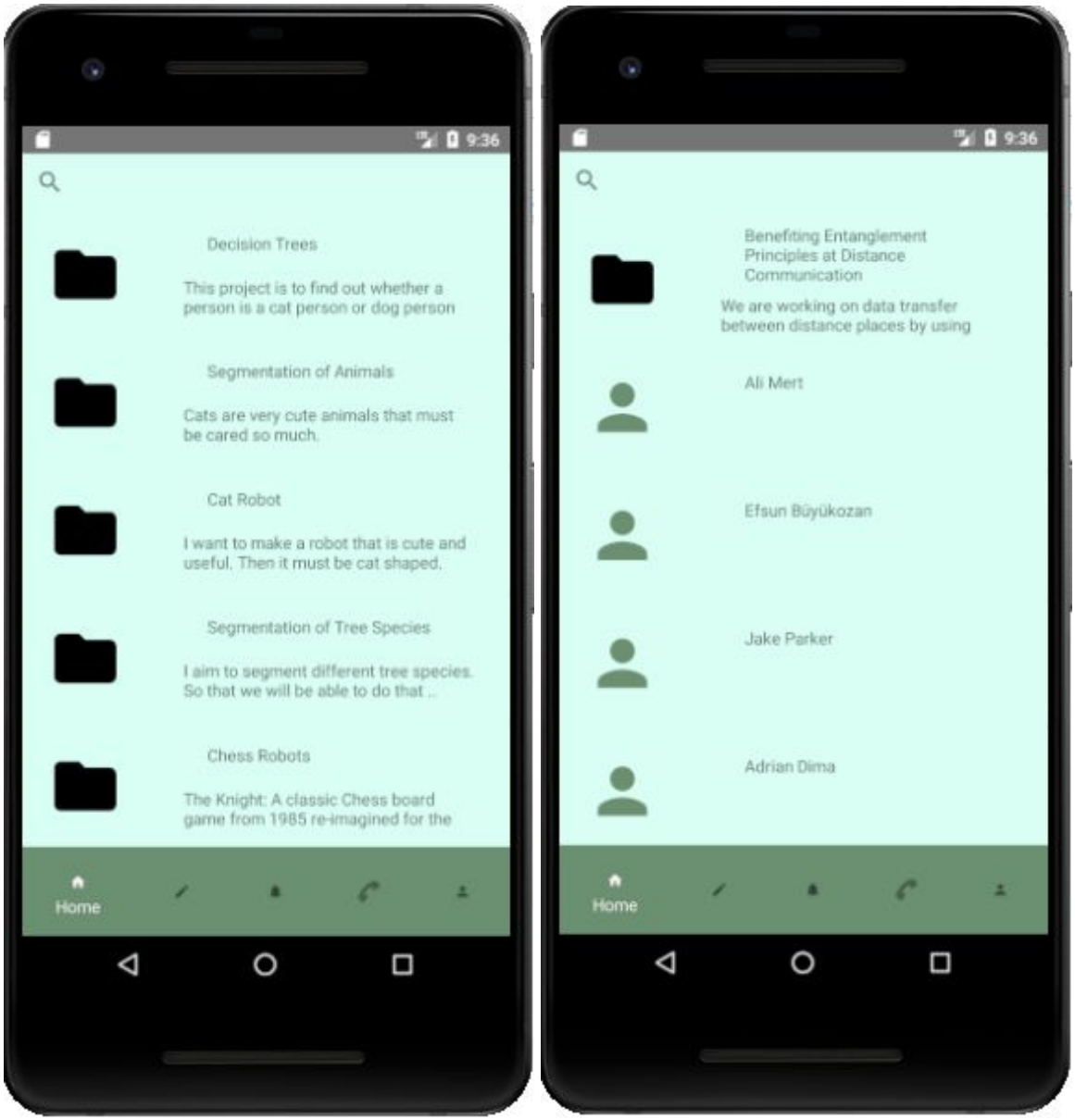


If it is the first login to the Akademise application, Login Page is opened. If a user logged in before, then the home page is opened without login. The login page consists of an email entry part, a password entry part, a login button, and a signup button. Email and password must be provided correctly to enter the platform. If they are entered wrongly “Wrong email or password. Try again.” shows up. After filling in the email and password part, the login button is pressed.

Home Page

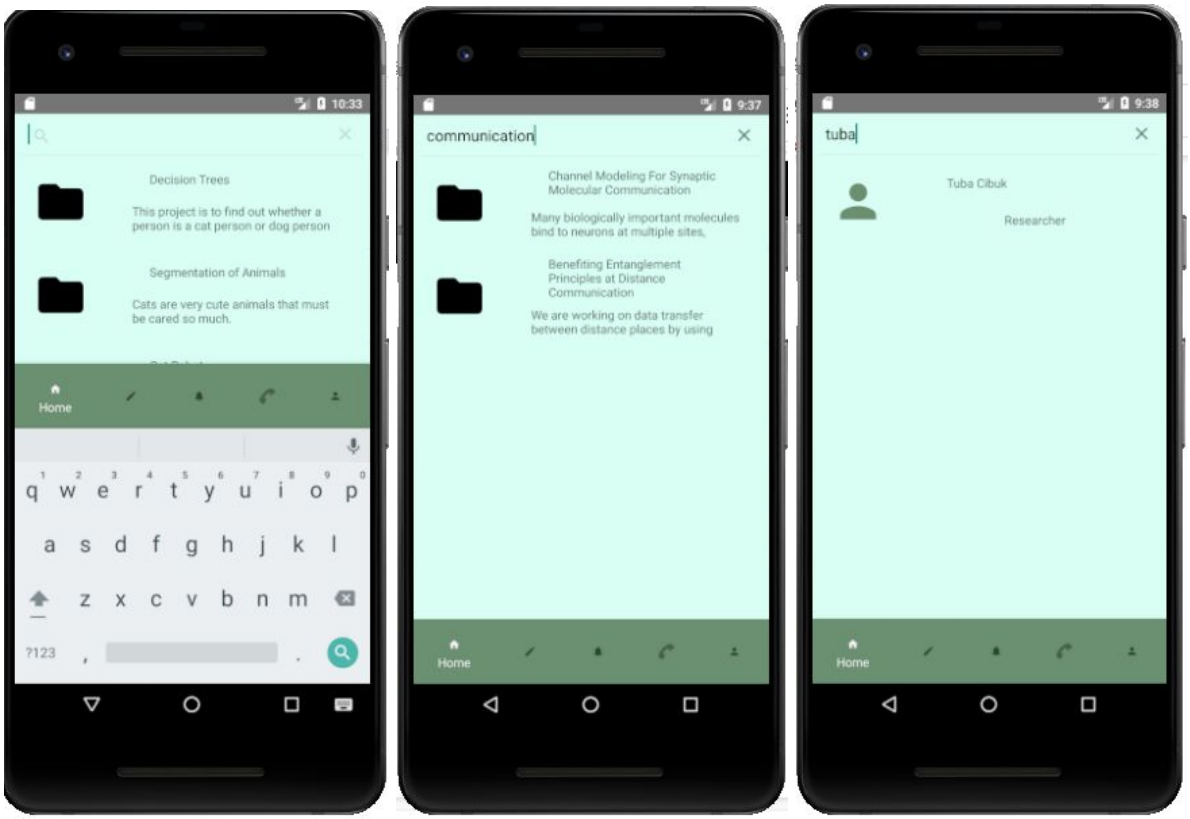
The home page includes two functionalities. One of them is the recommendation. The other one is searching.

Recommendation



The recommendation system recommends users and projects to the user. Users are recommended according to their research interests, universities, and departments. Projects are recommended according to the followed users and project tags. If a project is pressed, then the project details page is opened. If a user is pressed, then the profile page of that user is opened.

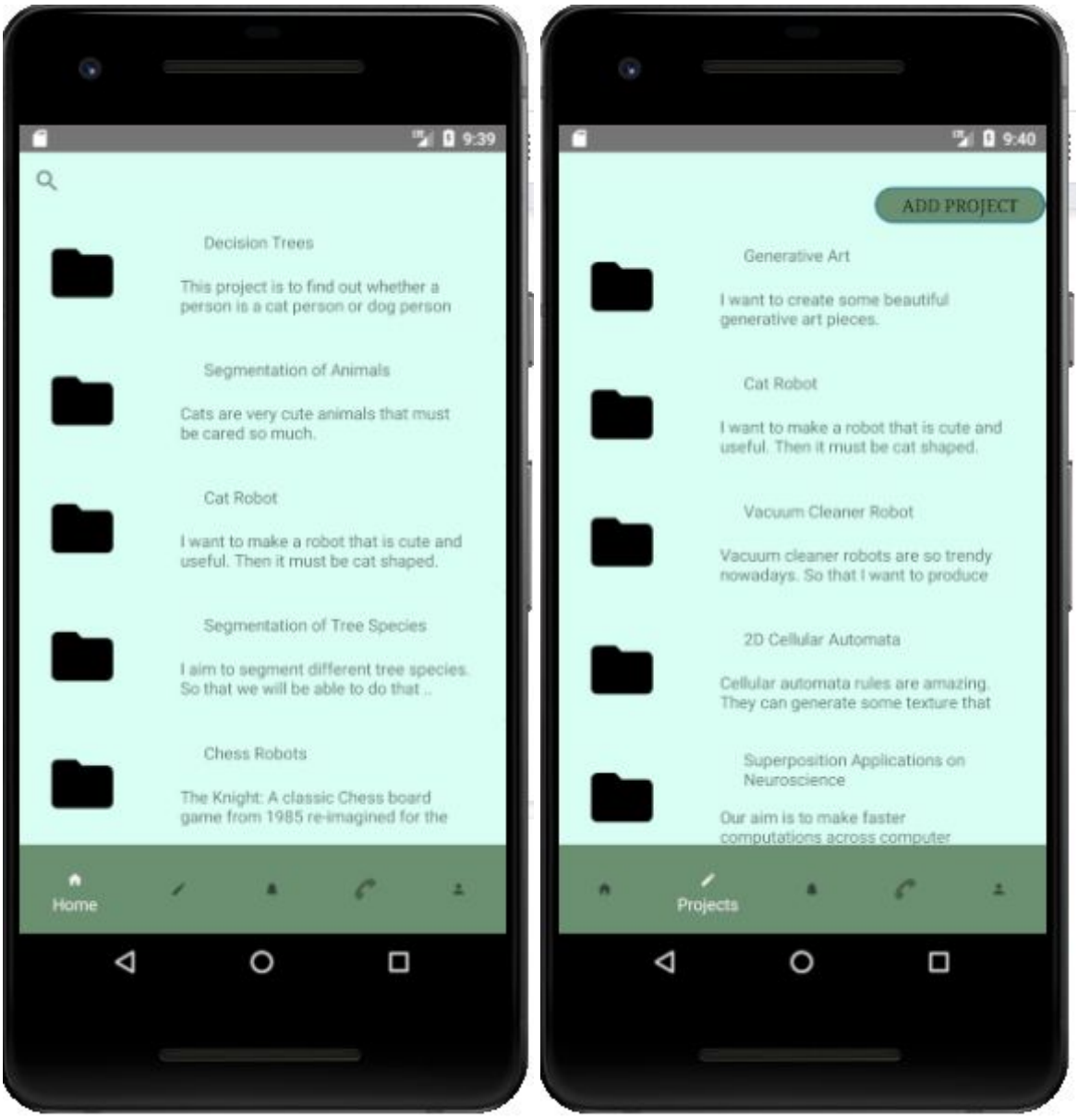
Search



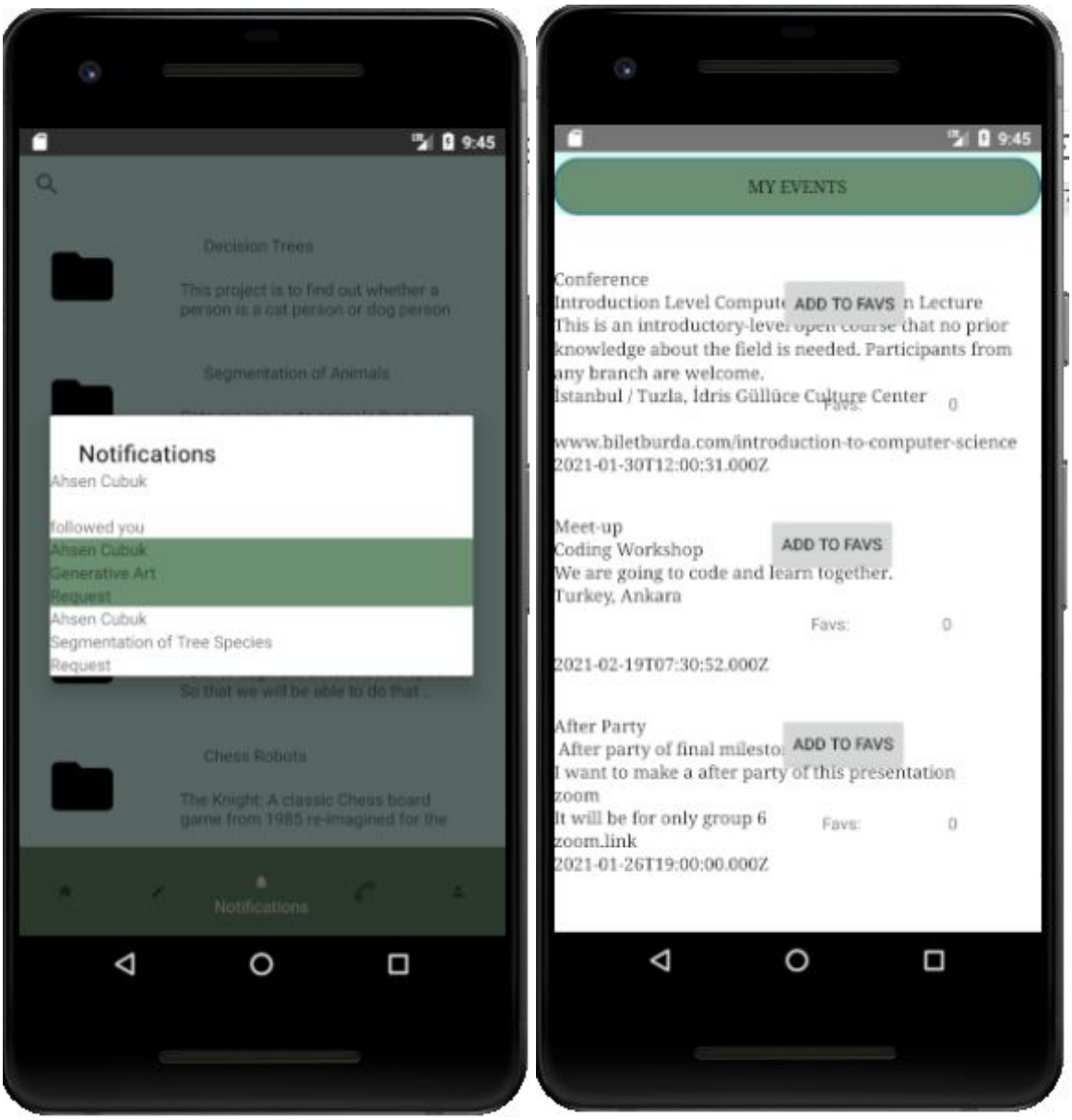
The search bar is located above the home page. Users and projects can be searched. After entering the query, the search button on the keyboard is pressed. The search button of the keyboard can be seen in the first picture of this section. If a searched project is pressed, then the project details page is opened. If a searched user is pressed, then the profile page of that user is opened.

Navigation

The navigation menu is located below the home page and the projects page. When a menu item is pressed, the name of that page is shown. If the home page item is pressed, it navigates to the home page. If the projects page item is pressed, it navigates to the home page. If the notifications item is pressed, the notification dialog is opened. If the events item is pressed, the events page is opened. If the profile item is pressed, the profile page is opened. The navigation menu can not be seen on the events and the profile page.



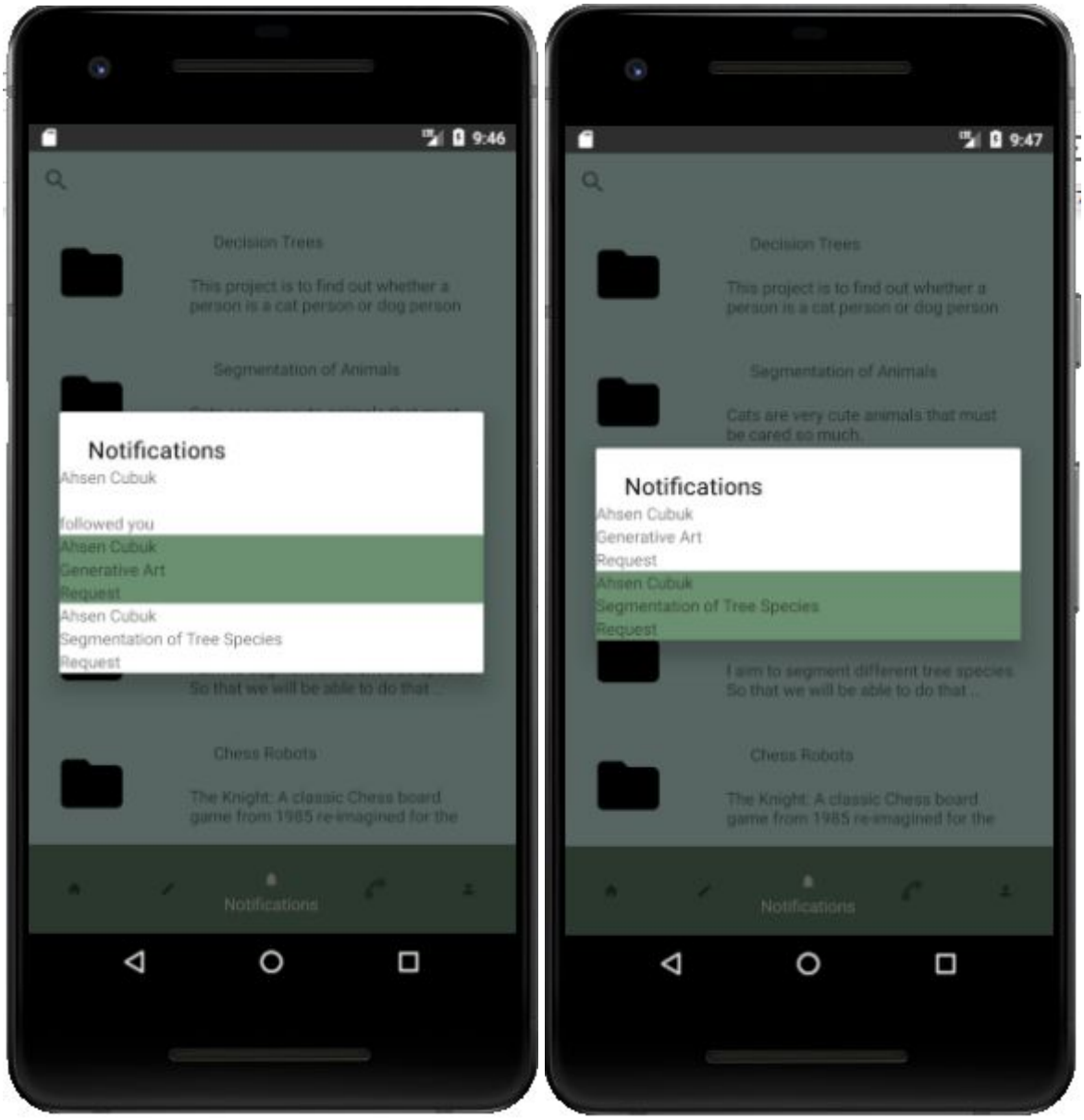
The figure on the left side shows the home page. The figure on the right side shows the projects page.



The figure on the left side shows the notifications. The figure on the right side shows the events page.

The figure shows the profile page.

Notifications



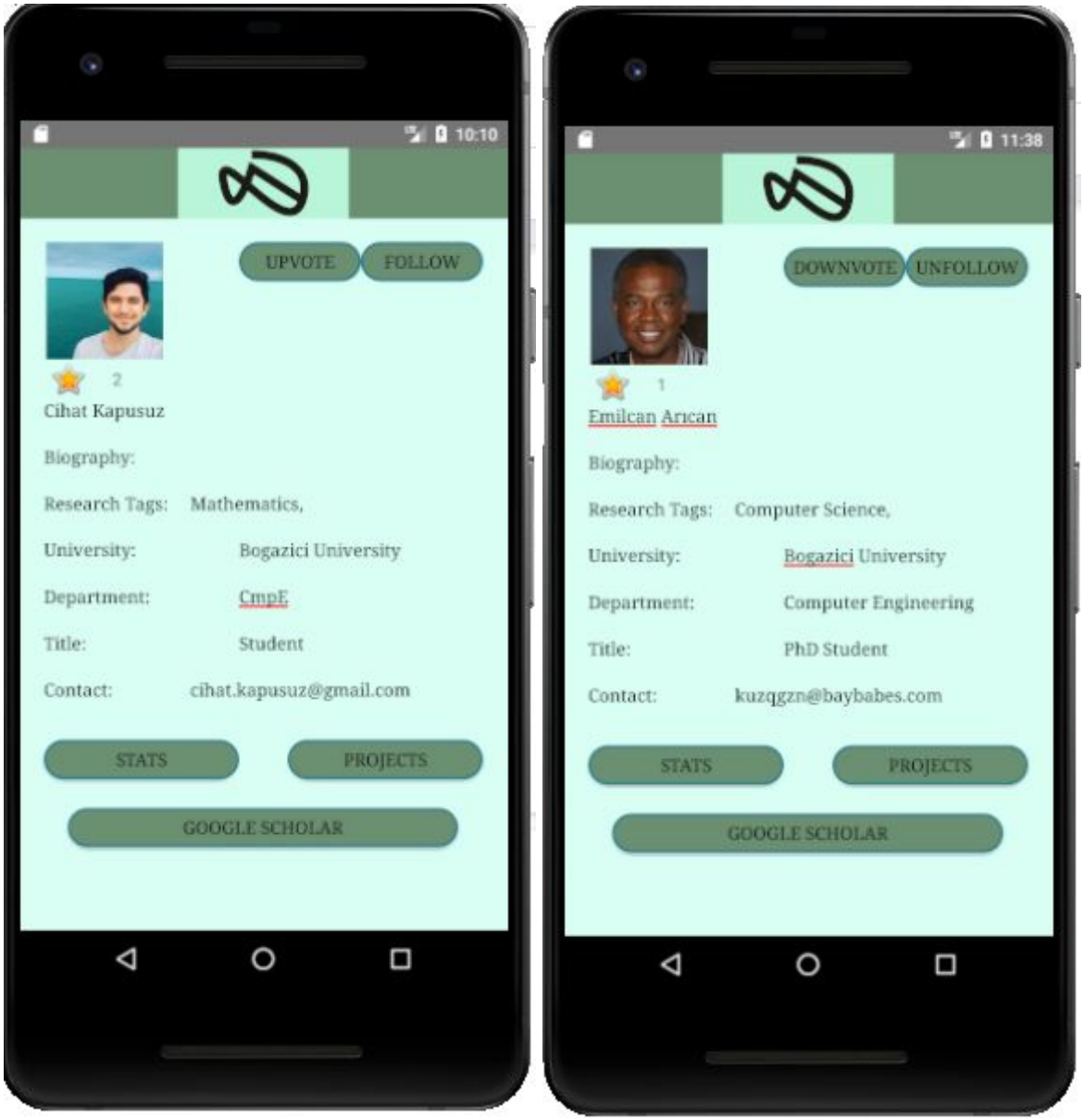
-Being followed by a user
-being accepted or rejected to a project
-being invited to a project
-acceptation or rejection of an invitation
-being removed from a project
-collaboration requests
-collaboration invitations
are shown on the notifications dialog.
If the name part of a notification (first line) is clicked, then the profile of that user is opened.
If the type part of a notification is clicked (third line) is clicked, then that notification is deleted. (If it is not a collaboration request or invitation notification)

Profile Page

Own Perspective

The profile page includes a profile photo, an edit button, a logout button, an upvote count part, a name part, a biography part, a research tags part, a university part, a department part, a title part, a contact part, a stats button, a projects button, a Google Scholar button, and an Invitations button. By pressing the edit button, the profile page can be edited. This part will be explained under the “Edit Profile” title. If the logout button is pressed, the user is directed to the login page. If the stats button is pressed, the Stats and overviews page is opened. If the projects button is pressed, the user is directed to the projects page. If the Google Scholar button is pressed, the Google Scholar page is opened. If the invitations is pressed, then the invitations page is opened.

Not Own Perspective

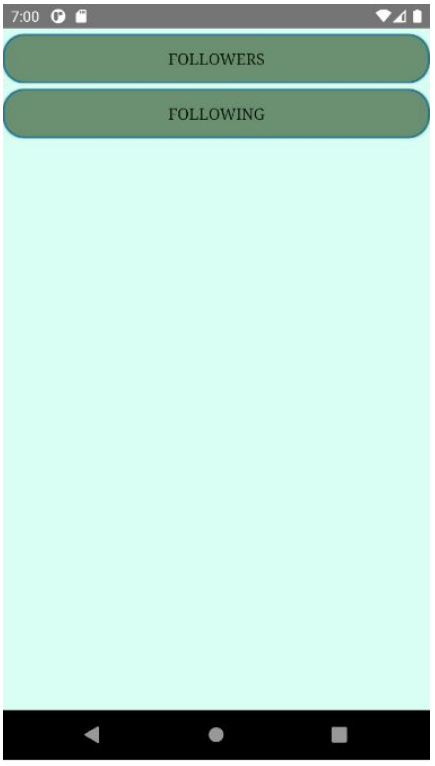


If a user looks at another user's profile a profile photo, an upvote button, a follow button, an upvote count part, a name part, a biography part, a research tags part, a university part, a department part, a title part, a contact part, a stats button, a projects button, and a Google Scholar button can be seen. If the upvote button is pressed, the user being displayed is upvoted and the upvote count increases by 1. Then the name of the button becomes downvote. If the button is pressed when its name is downvote, the user being displayed is downvoted and the up count decreases by 1. If the follow button is pressed, the user being displayed is followed. Then the name of the button becomes unfollow. If the button is pressed when its name is unfollow, the user being displayed is unfollowed. If the projects button is clicked, the projects page of that user is opened. If the Google Scholar is clicked, then the Google Scholar page of that user is opened.

Edit Profile

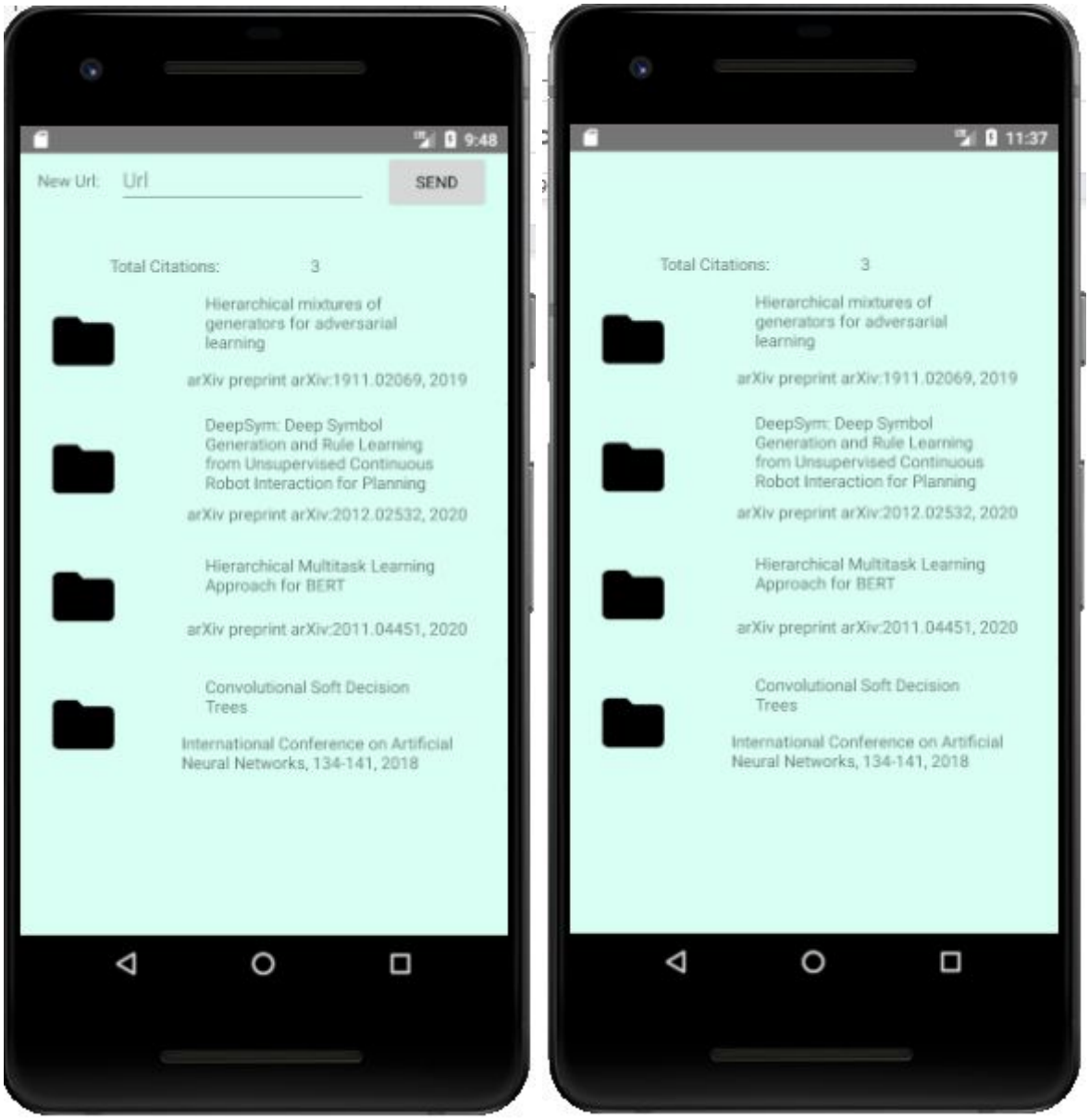
A user can see his/her profile via the bottom navigation menu and edit his/her profile by tapping the edit button. After tapping it, the user can edit the fields(except for contact) to his/her liking. After completing the editing process, the user can complete it by tapping on the update button.

Stats and Overview



A user can get to the Stats and Overview page via his/her profile. By tapping on Followers or Following buttons, the user will see the users who follow him/her or the users whom s/he follows, respectively. The user can also see the stats and overview of other users in the same manner.

Google Scholar Page



If a user looks at her/his own Google Scholar page, then it looks like the figure left. A user can see his/her total citations, publication titles and venues. A user can change the Google Scholar url. When a new url is typed the send button is pressed. When the Google Scholar page is opened again, it displays new information. It is important that A url must be typed in the correct format.

Correct Format Example:
https://scholar.google.com/citations?user=B23iqYwAAAAJ&oi=ao

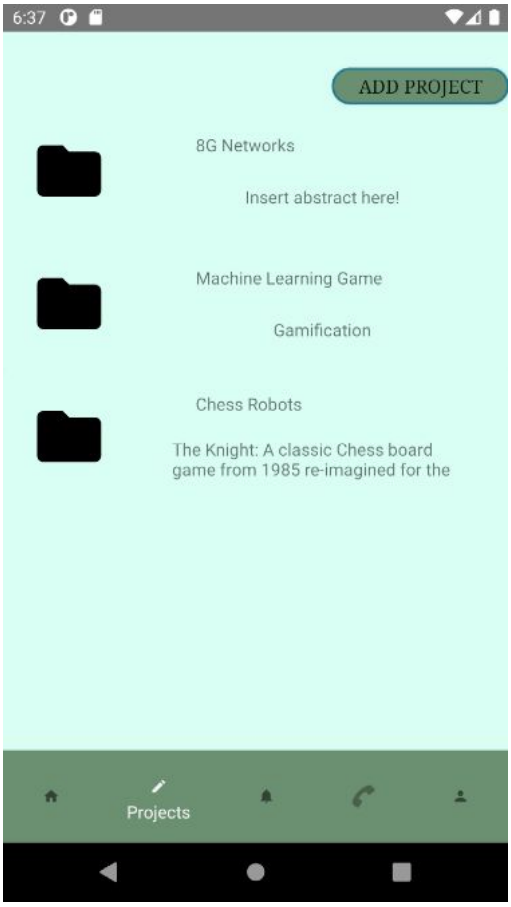
Incorrect Format Example:
https://scholar.google.com/citations?user=B23iqYwAAAAJ&hl=tr&oi=ao

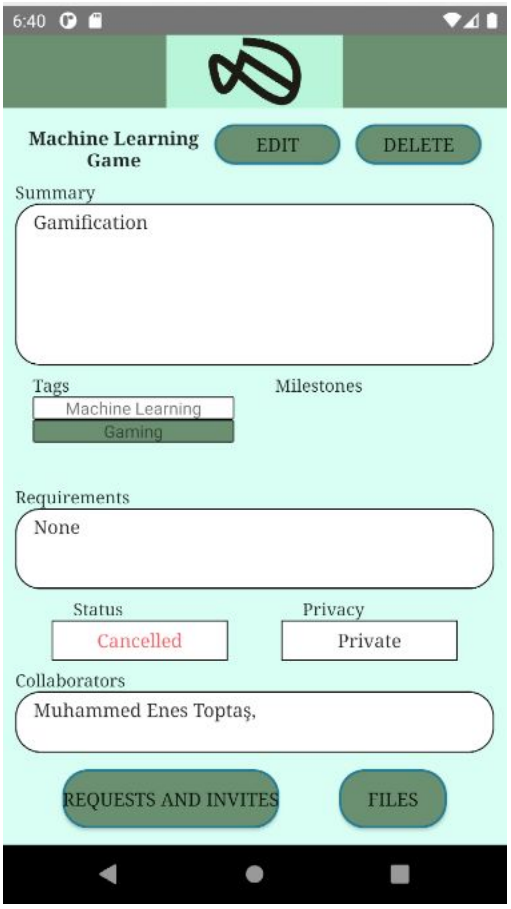
“&hl=tr” part of the url must not be typed.

If a user looks at another user's Google Scholar page, then it looks like the figure right. A user can see another user's total citations, publication titles and venues.

Projects Page

Owner Perspective



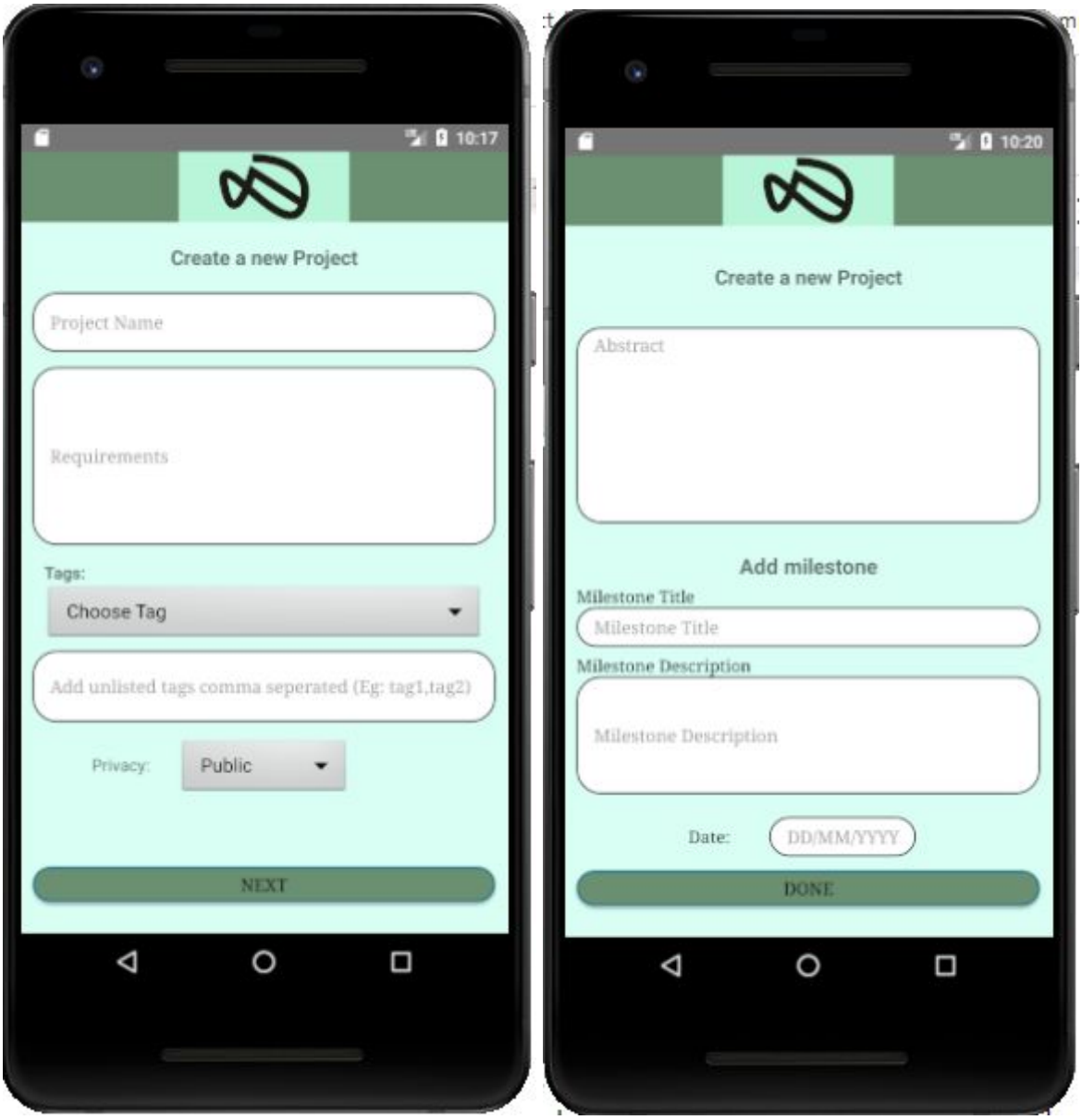


A user can get to the projects page via the bottom navigation menu or via his/her profile. The user can see all the projects s/he created. The user can see the details of a project by tapping on the project.

Non-Owner Perspective

A user can see other users' projects via their profile.

Project Creation



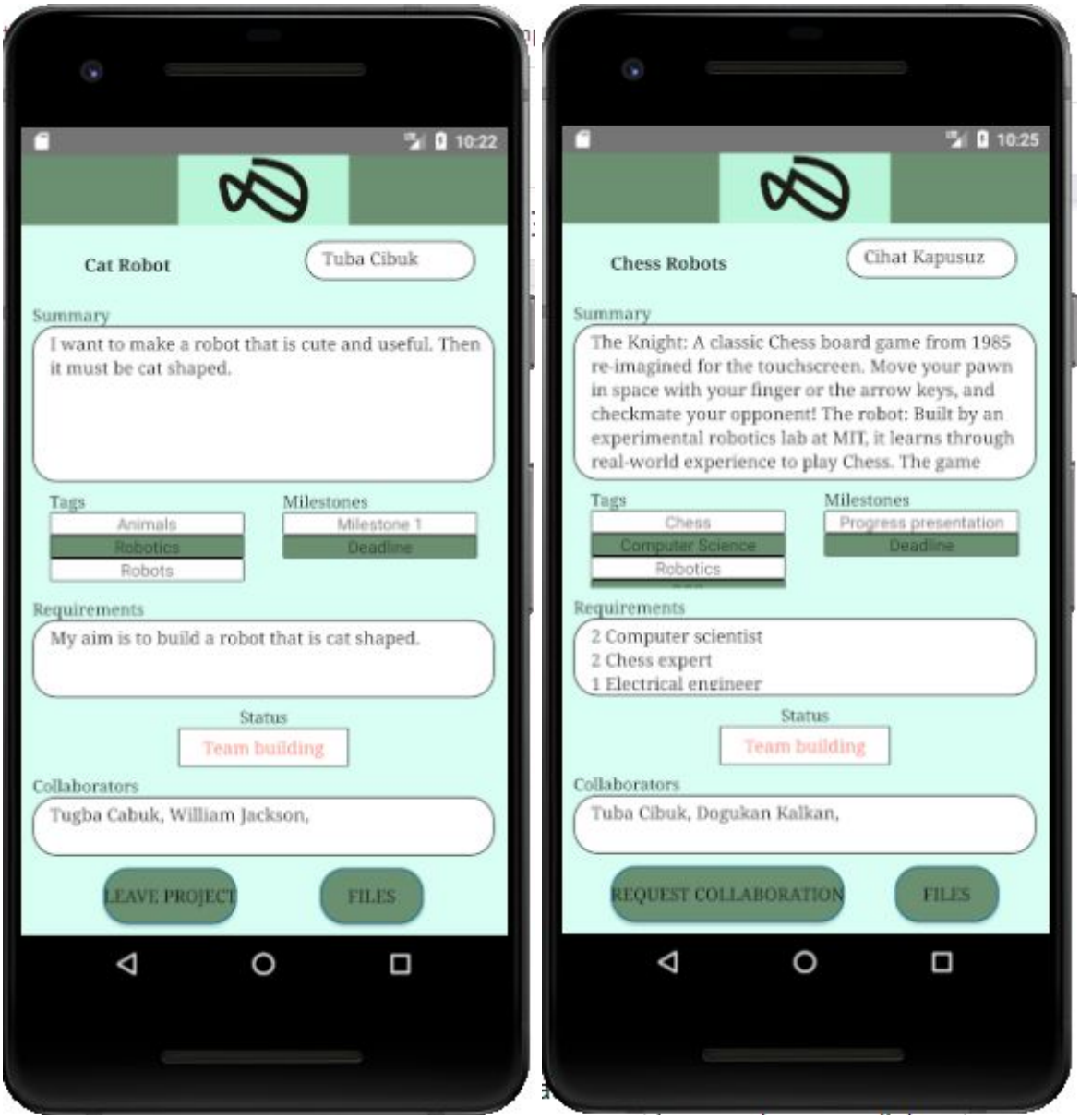
To create a new project user must enter the necessary fields for creating a project. They can navigate to this page from the projects page by clicking the ‘ADD PROJECT’ button. In this part, the user must enter the project title, requirements of the project, tags of the project and the privacy of the project, then click the ‘NEXT’ button to proceed to the next page. On the next page, the project's abstract and milestone must be added to be able to fill all the necessary parts of the project. After filling in the abstract and milestone, the user must click the ‘DONE’ button to complete project creation. The project’s status is ‘Team-building’ by default while creating a new project, the user can change it on the edit project page.

Project Details

Owner perspective

On the project details page, the project's title, abstract, requirements, tags, milestones, privacy, status, and collaborators are shown. Abstract and requirements text boxes can be scrollable if the content in those boxes does not fit in the defined area. Tags and milestones are shown in recycler views, they are scrollable, too. Status of the project can be ‘Team-building’, ‘Hibernating’, ‘In progress’, ‘Completed’, and ‘Cancelled’. The status of the project is shown with a colored text and there exists a different color for each status. Privacy of the project can be private or public, private projects can not be seen by non-collaborator users. Collaborators of the projects are shown by their names in a comma-separated form.
If the user is the owner of the project, s/he can edit or delete the project. S/he can edit the project by clicking the ‘EDIT’ button and delete the project by clicking the ‘DELETE’ button. Also, there is the ‘FILES’ button to be able to see files of the project. There is a ‘REQUESTS AND INVITES’ button to link the project details page to the requests and invites page when the user is the owner of the project.

Non-Owner perspective

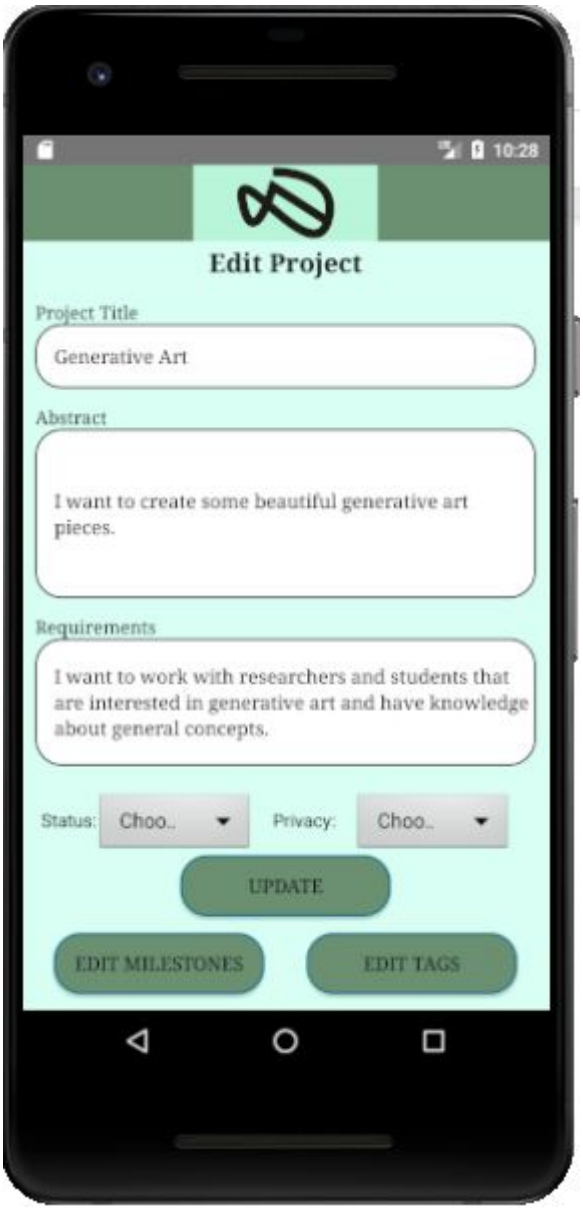


Different from the owner's perspective, the user is not able to edit or delete the project. Instead of the 'EDIT' and 'DELETE' buttons, the name of the project's owner is shown there, the user can see the profile of the project owner by clicking the name of the user. Also, the privacy of the project is not visible from the non-owner perspective. If the user is the collaborator of the project, s/he can decide to leave the project by clicking the 'LEAVE PROJECT' button, otherwise s/he can request to collaborate the project by clicking the 'REQUEST COLLABORATION' button. Other parts are the same with the owner perspective.

Project Files

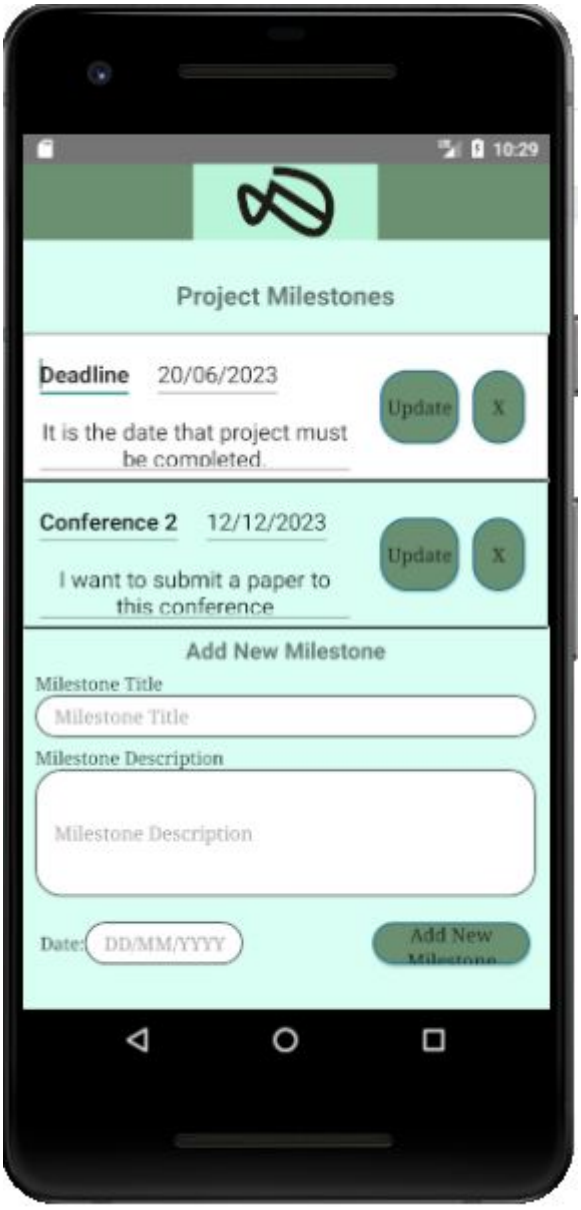
In the project files page, the user is able to see what are the files of the project.

Edit Project



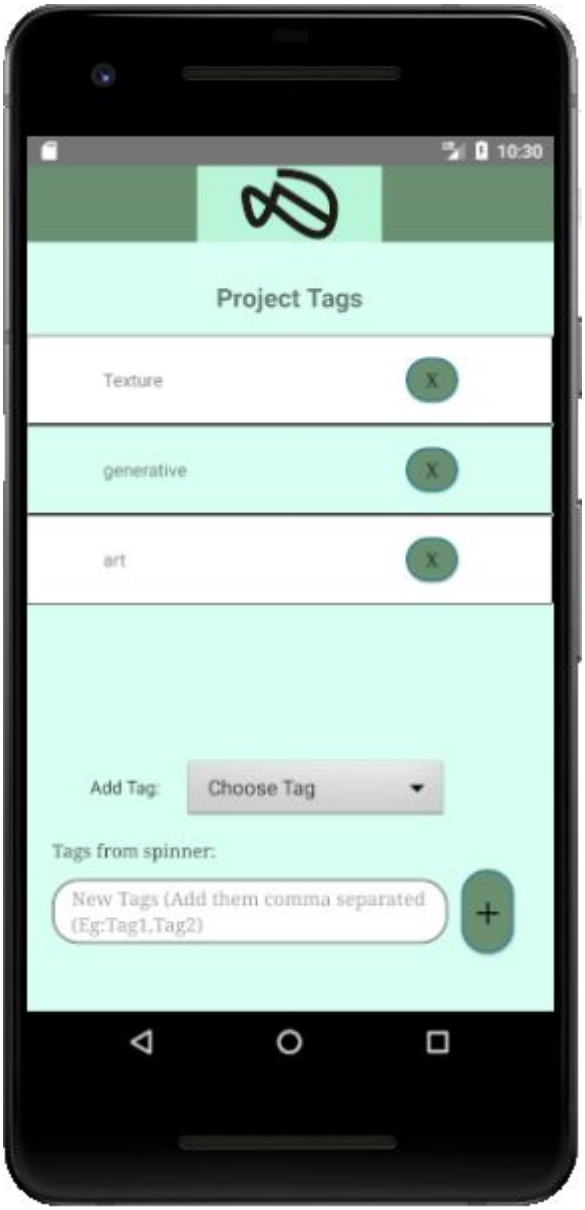
The user can come into this page from the project details page if the user is the owner of the project. In the edit project page, the user is able to update project title, abstract and the requirements of the project. Current versions of these fields are shown in editable format. Also, the user is able to set new states for the status and privacy from a spinner. If s/he does not select a value to set new status or privacy, previous values will be kept. The changes done while updating the project are applied when the user presses the ‘UPDATE’ button. Also, the user is able to edit the project’s milestones and tags by pressing corresponding buttons. To edit milestones, the user should press the ‘EDIT MILESTONES’ button. To edit tags, the user should press the ‘EDIT TAGS’ button.

Edit Milestones



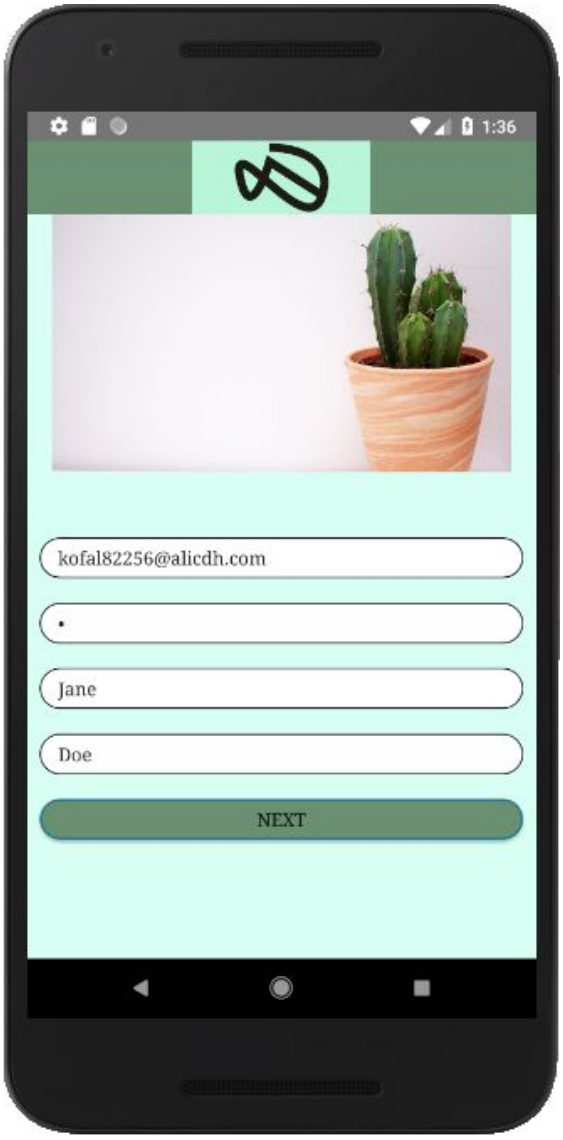
In the edit milestones page, the user is able to edit current milestones by updating their contents or removing them. S/he can add a new milestone to the project too. Current milestones are shown in a recycler view. Title, description and date fields of the milestone can be edited. Date field must preserve DD/MM/YYYY format in order to be able to perform changes correctly. After changing values for these fields, the user must press the corresponding ‘UPDATE’ button for updating the milestone. If the user wants to delete a milestone, s/he must press the ‘X’ button near the information of the milestone. To add a new milestone to the project, the user must enter a title, a description and a date in the correct format (DD/MM/YYYY) in related text boxes. After adding necessary information, the user must press the ‘Add New Milestone’ button to add a new milestone to the project.

Edit Tags



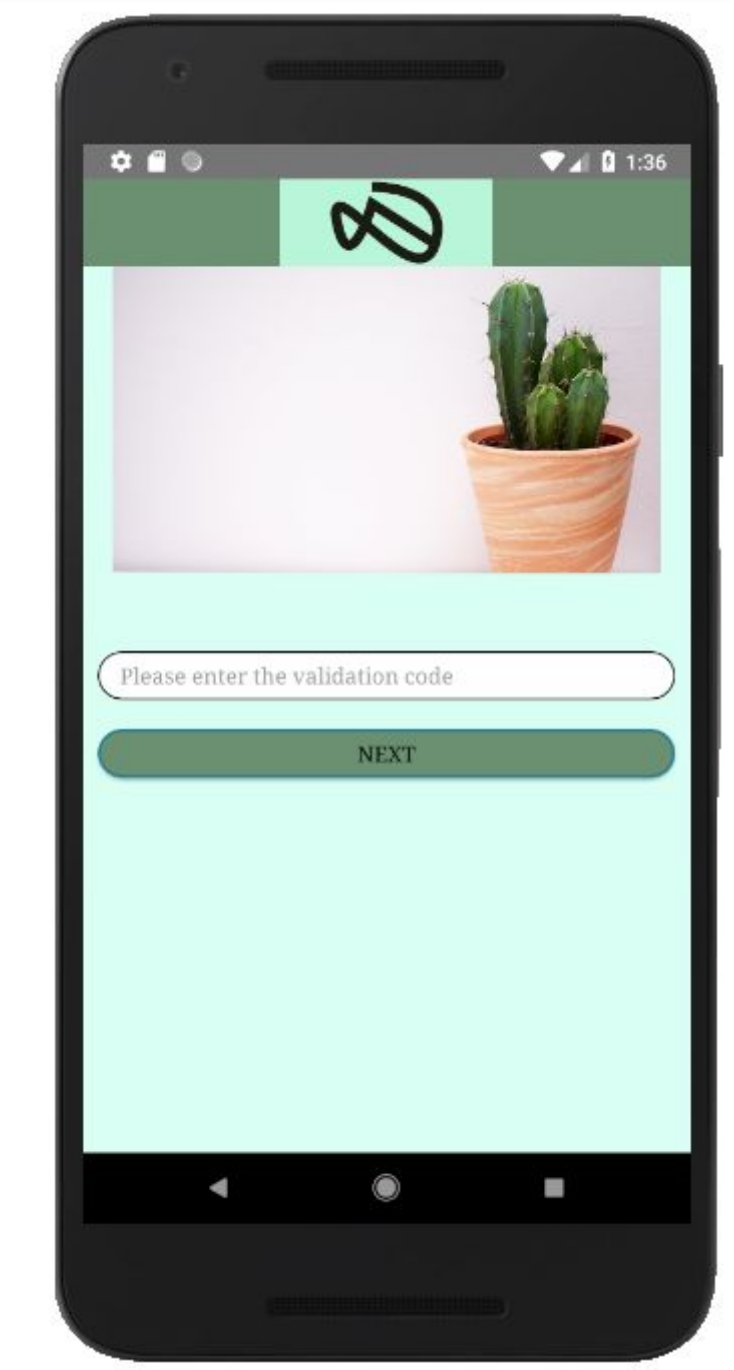
In the edit tags page, the user can delete current tags and add new tags to the project. Current tags of the project are displayed in a recycler view, s/he can scroll to see all tags if the project has more tags. The user must press the ‘X’ button near to a tag to delete that tag from the project. On the other hand, if the user wants to add new tags to the project, s/he can select new tags from the spinner, the tags already in the project are not shown as selected new tags to avoid duplicate tag addition. Also, if the desired tag is not listed in the spinner, users can enter new tags by adding them comma separated. After selecting all new tags the user wants to add, s/he should press the ‘+’ button to add new tags to the project.

Signup page



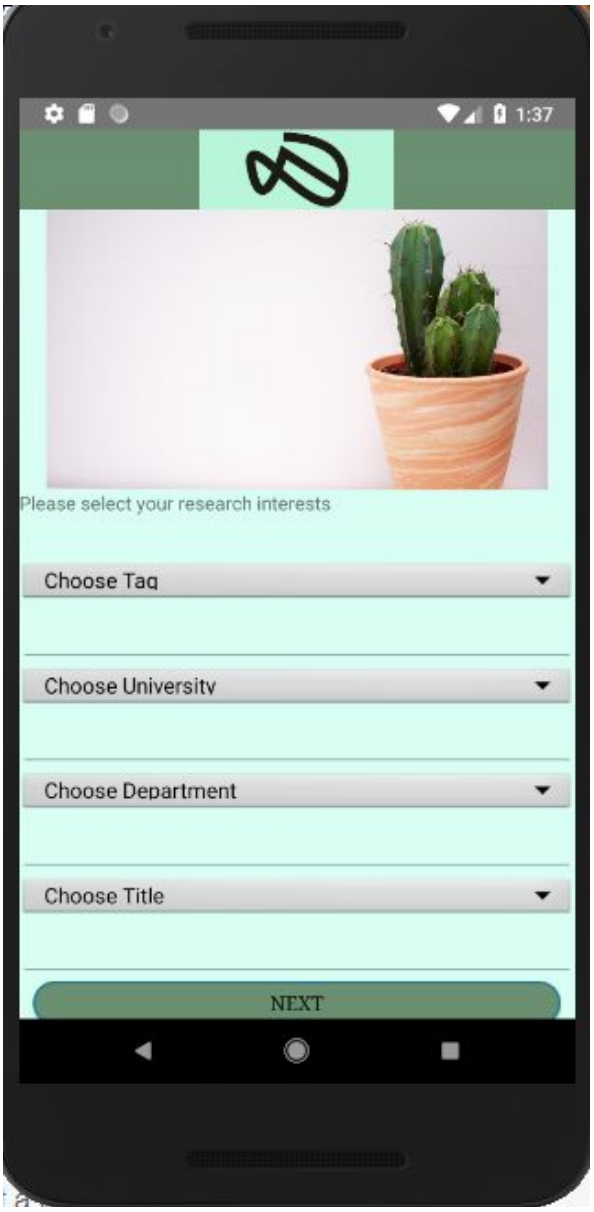
In the signup page , firstly user should fill in the necessary personal information to create a new profile. These are : email address, password, name and surname.
If email is not in the correct format, “Given email is not valid” error will be seen and user can!t proceed without providing a valid email address.
After filling these up, user clicks Next button, which will navigate the user to the validation step.

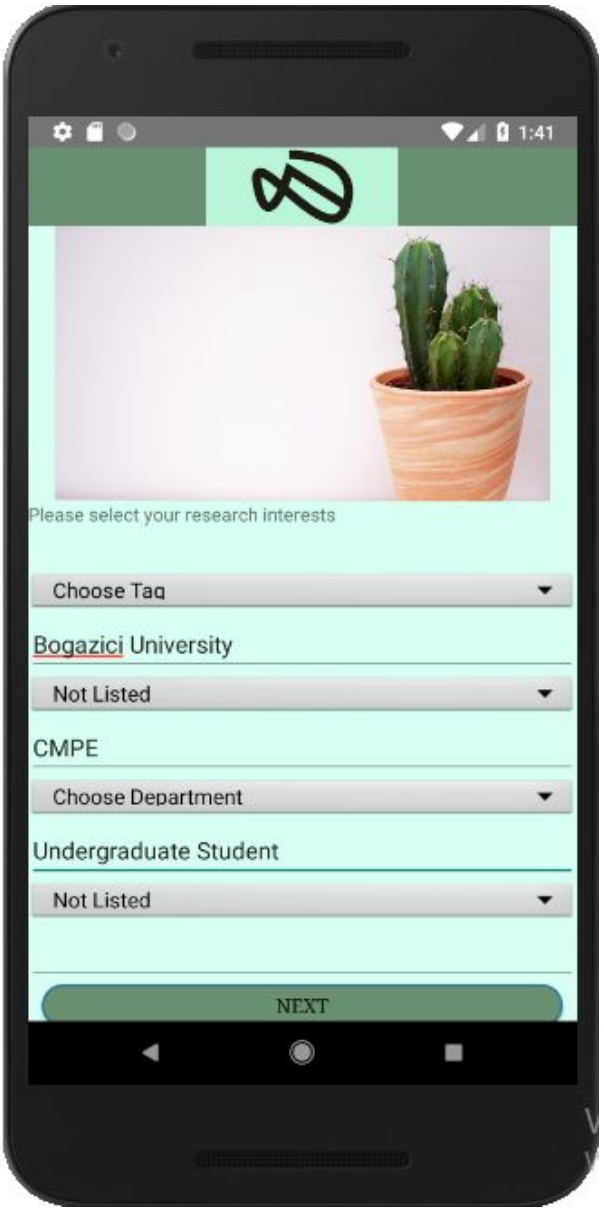
Validation Page



In the validation page, user enters the validation code that has been sent to his/her registered email address. And clicks next to proceed to next steps of signing up.

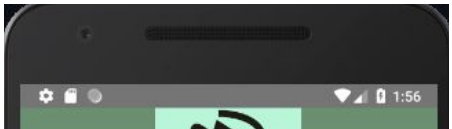
Affiliation Page

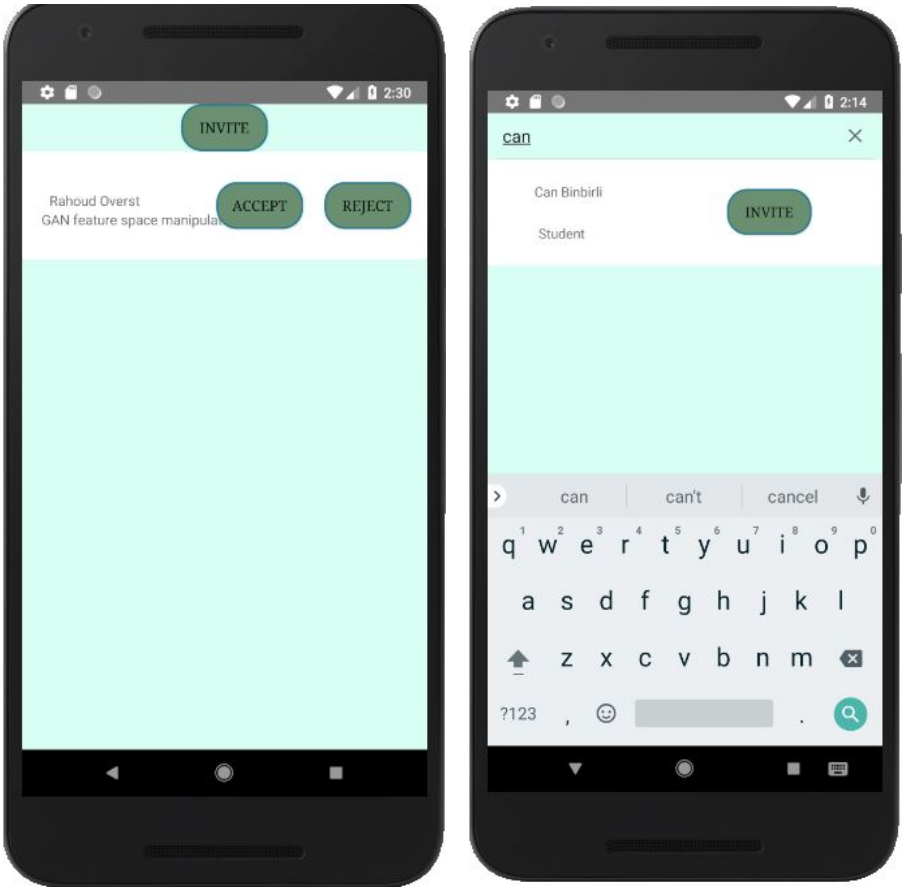




In this page, user fills up information about his/her academic background and research interests . She/he can choose the from the dropdown list or if it is not listed, she can choose “not listed” option and add her own through typing her information in the text spaces (shown in second figure)

Seeing Collaboration Requests and Inviting Users





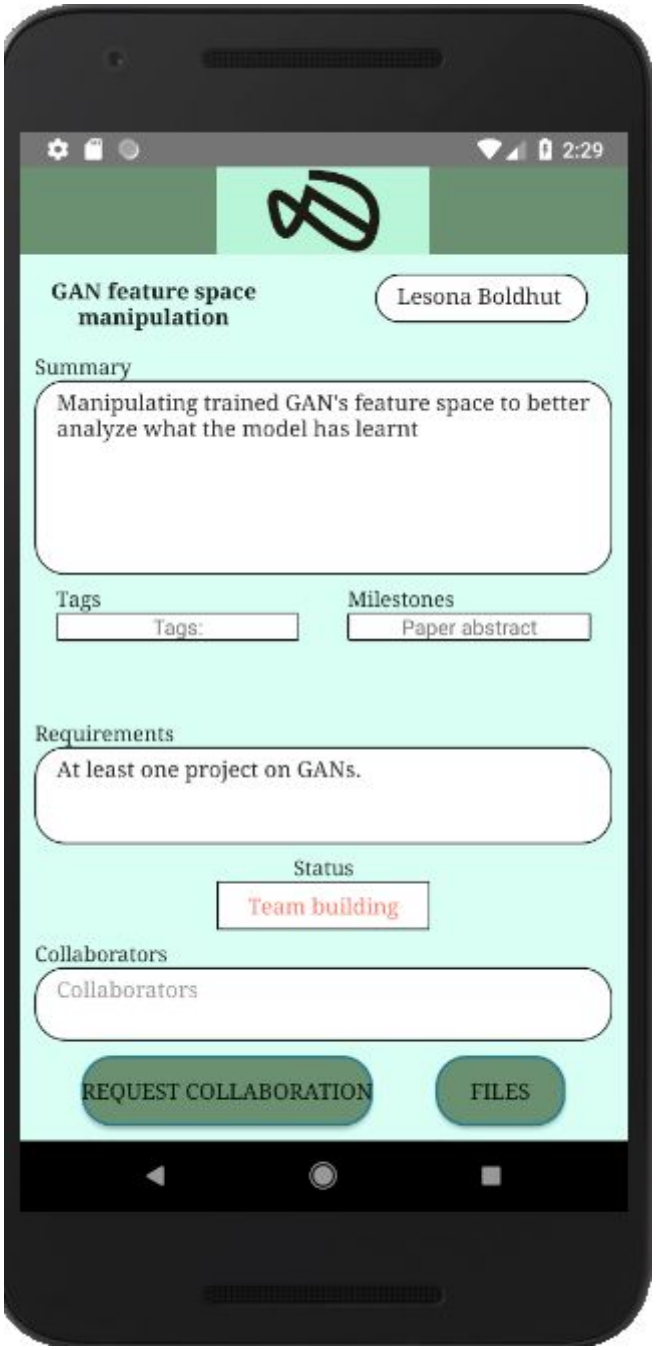
When the user is the owner of the project, when she goes to the project details page in the left bottom part “Requests and Invite” button.

This button leads to (second figure) received collaboration requests for this project and another button “Invite” to search and invite new users to the project. For received requests, the project owner can either accept or reject the collaboration request by clicking the corresponding buttons.

When a user clicks the “Invite” button she will be navigated to another page where she can search for other users to send collaboration invitations for this project. User clicks the “Invite” button next to the other user’s name to send an invitation.

Users can see received invitations by going to her profile page and clicking the “Invitations” button.

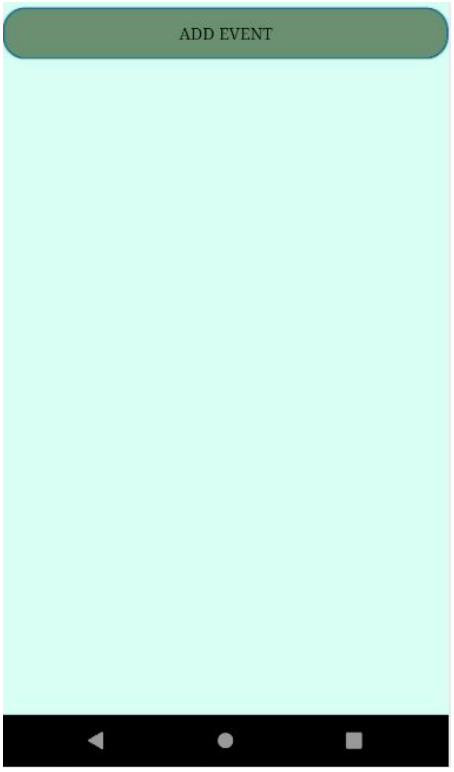
Sending Collaboration Requests for Other Projects



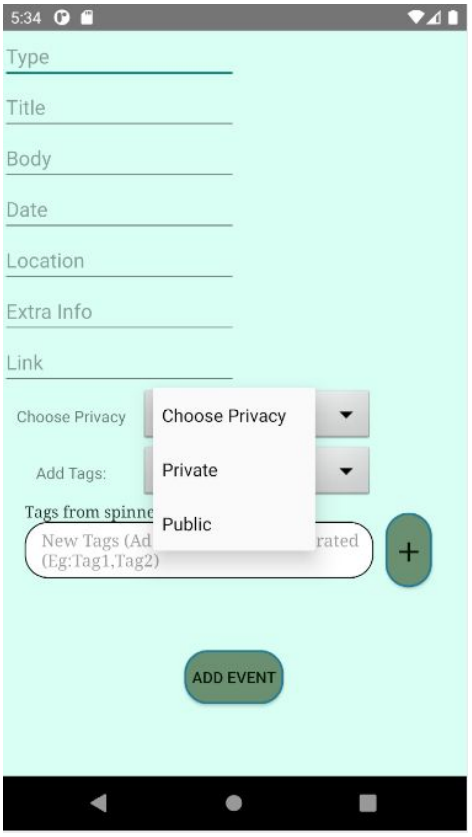
When user is in the project details page of a project (that she is not the owner / collaborator of) in the left bottom part “Request Collaboration” button is visible. When the user clicks on that button she will send a request to work on this project.

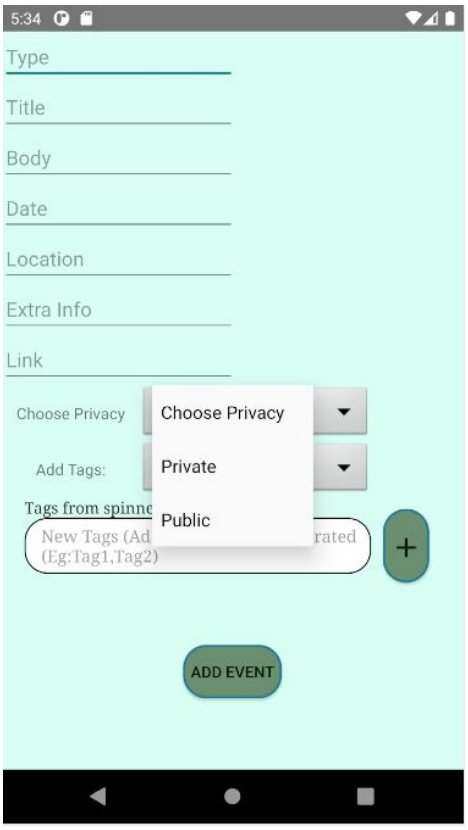
EVENTS

Add Event:



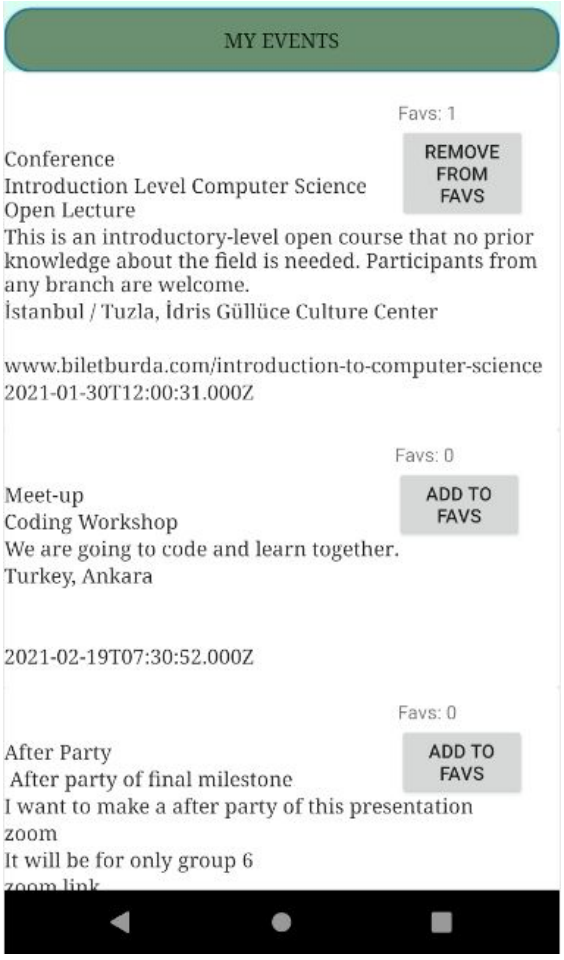
A user can get to the events page(first figure) from the bottom navigation menu. After that, the user can see his/her events and add new ones.





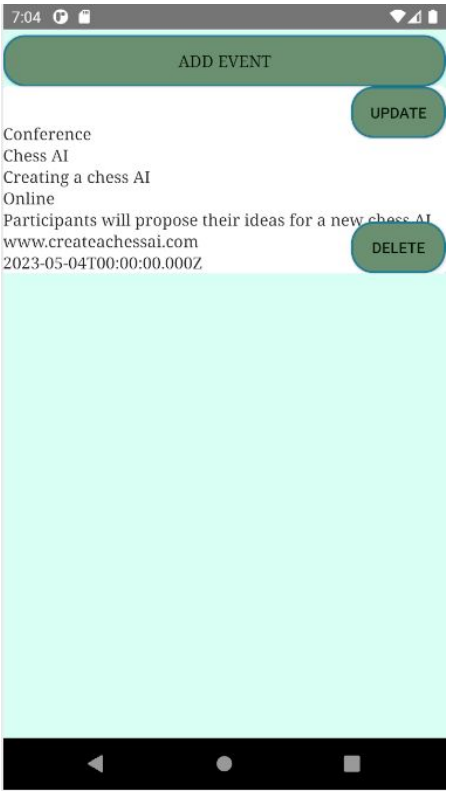
As you can see in the figures above, when creating an event, all the necessary information can be written in the corresponding fields, and the user can also set the privacy of the events as well as the tags s/he desires. Note that the user must enter the date as the following: YYYY/MM/DD 00:00:00 (e.g. 2020-05-10 00:00:00).

Fav Event:



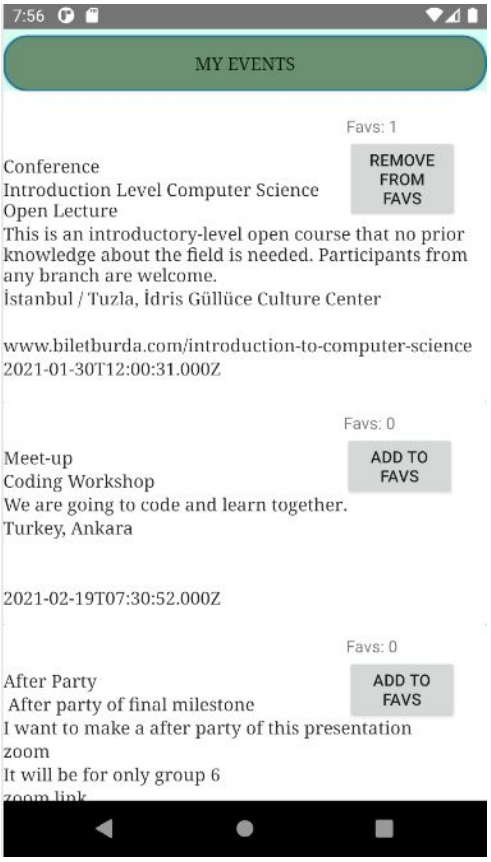
In the events page, the user can see all the events in Akademise and add any of them to his/her favorites or remove them from his/her favorites.

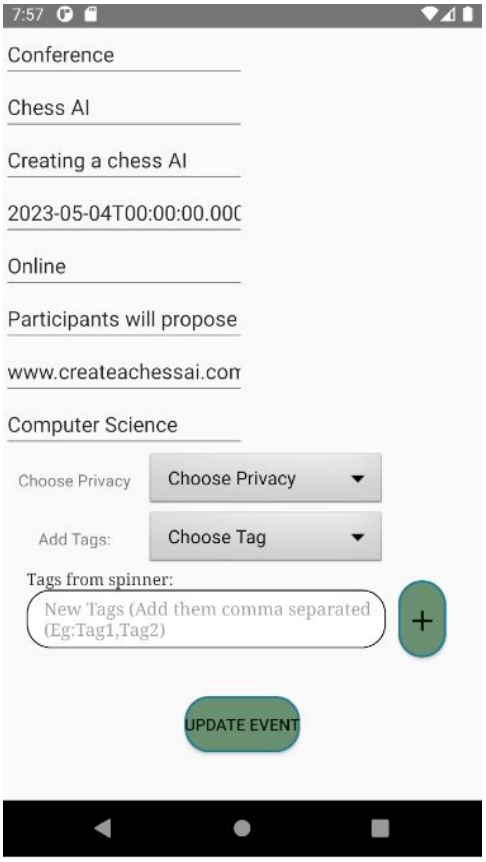
Delete Event



A user can see his/her events via tapping the My Events Button in the Events page. After that the user will see his/her events. The user can see an event via tapping the Delete button next to the corresponding event.

Update Event





A user can get to his/her events via My Events Button in the Events Page Then s/he can update any of his/her events by tapping the update button next to the event. Note that the

user must enter the date as the following: YYYY/MM/DD 00:00:00 (e.g. 2020-05-10 00:00:00).

The Code Structure and Group Process

Frontend

Team:

- Cemre Efe Karakaş
- Ali Ramazan Mert
- Emilcan Arıcan
- Ömer Faruk Doğru

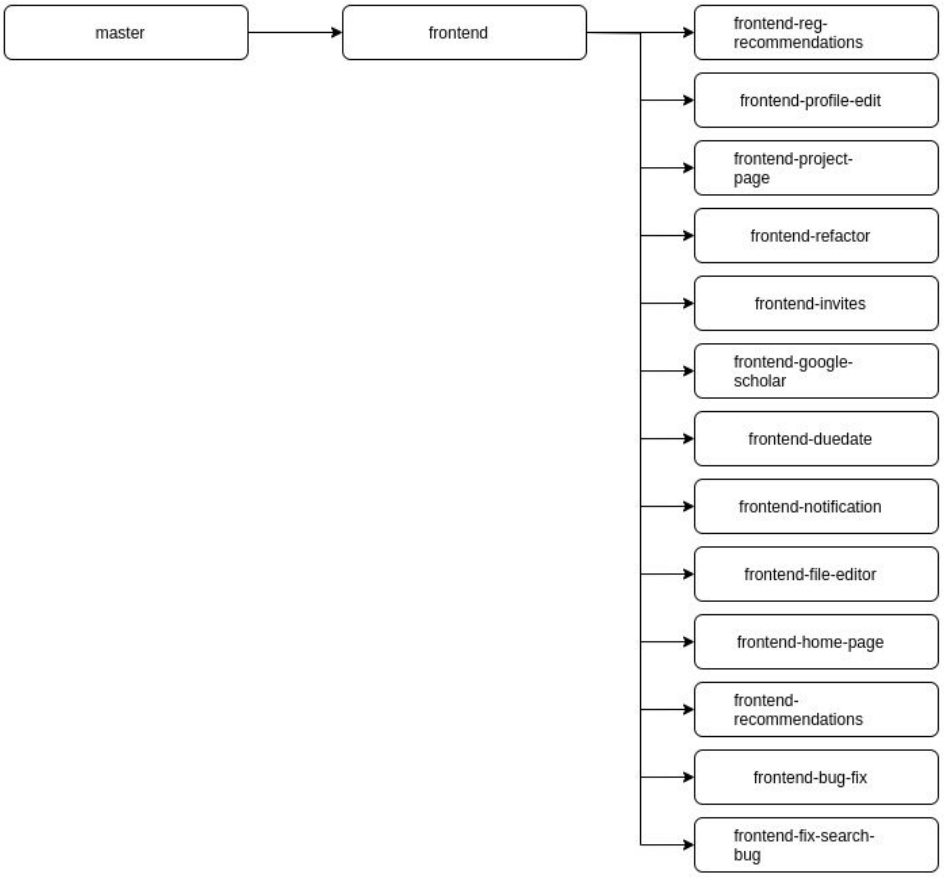
Tools Used:

- Figma (collaborative design tool)
- Yarn (package manager)
- React (frontend library)
- VSCode (ide)
- Ant Design (react UI library)

As the frontend team, we opted for using React framework and making use of Antd's readily available components. As our friend Ali had quite a bit of experience with both, we organized three workshops. One for react, one for antd and one for redux and async calls.

We used the existing branch under the name `frontend`. We made small feature additions and typo corrections on the frontend branch and created separate branches for specific tasks (like separate pages). We usually tried to only open a new branch if there are going to be a series of changes made for a specific task.

Active Frontend Branches (Milestone 2)

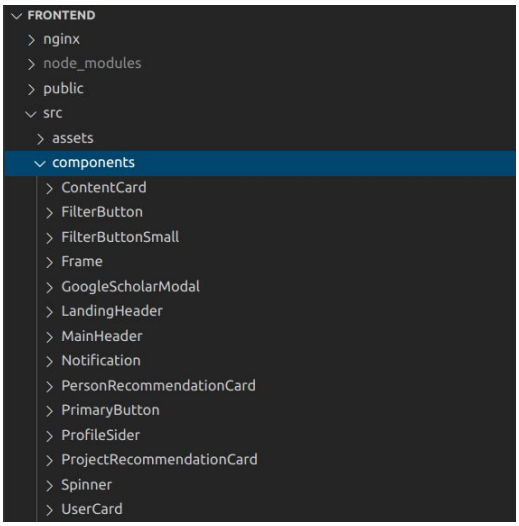


Each week we take the responsibilities given to the frontend team and divide them between the team members. We set a deadline before the team's internal deadline to have time for overcoming possible bugs and difficulties.

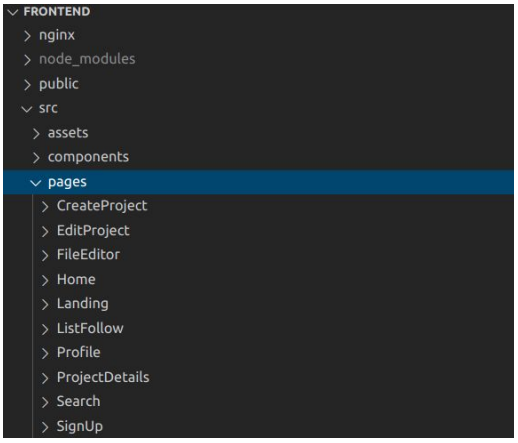
After the feature being developed in a branch is complete, a pull request is sent. The other frontend members are assigned. At least one team member reviews the changes and makes comments for possible improvements. The user responsible for the request reviews the code and re-commits. After this point, the pull request is approved by a reviewer via comment. And a team member merges the branch. If there is no planned future work for a branch, the branch is deleted.

The directory structure is as follows:
(We decided to keep individual screenshots for individual subfolders to better capture everything)

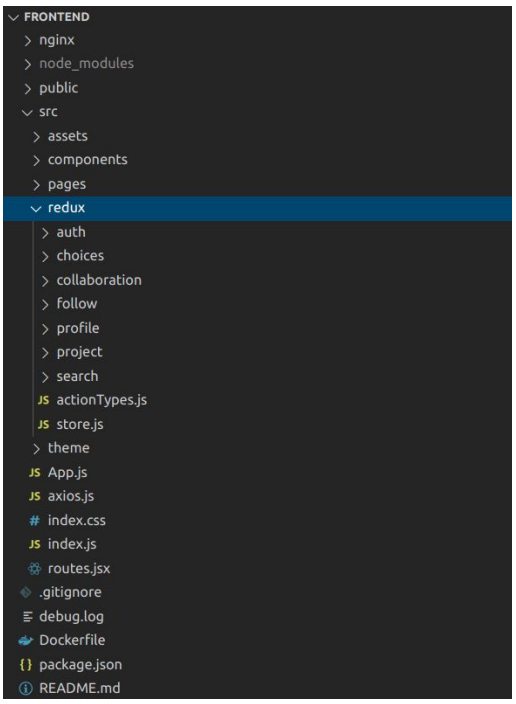
Compontents:



Pages:



Reducers & others:



Each folder in pages and components contains a jsx file for the page/component and a js file for css styles.

The assets folder contains files that are used in pages. The public folder keeps public files like images for the website.

Backend

Team:

- Muhammed Enes Toptaş
- Hamza Işıktaş
- Hazer Babür
- Ömer Faruk Özdemir

Tools Used:

- ExpressJS (framework)
- PostMAN (api testing)
- VSCode (text editor)
- PostgreSQL (database)
- Docker (containerization)
- Sequelize (orm)

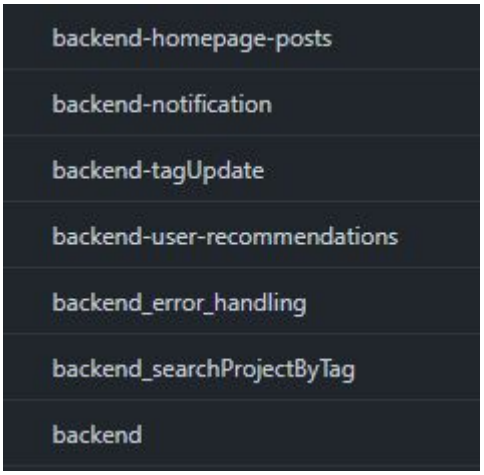
As the backend team, we kept meeting after our lab sessions and decided what to do there. We made use of WhatsApp and Slack frequently to talk about our coding struggles. We also hold some Zoom sessions and coded together in cases we needed more than one people's opinion.

We used Visual Studio Code as our code editor. And utilized Git branches with Visual Studio's integrated Git system to make version control simpler.

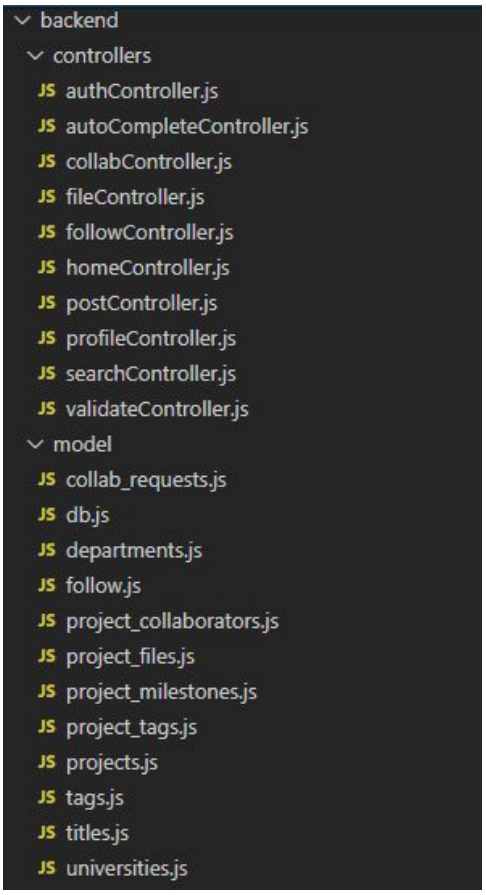
We kept using the branch "backend" as our main branch for backend. We set up CI/CD with Github actions also on this branch, and it redeployed every commit we pushed to this branch.

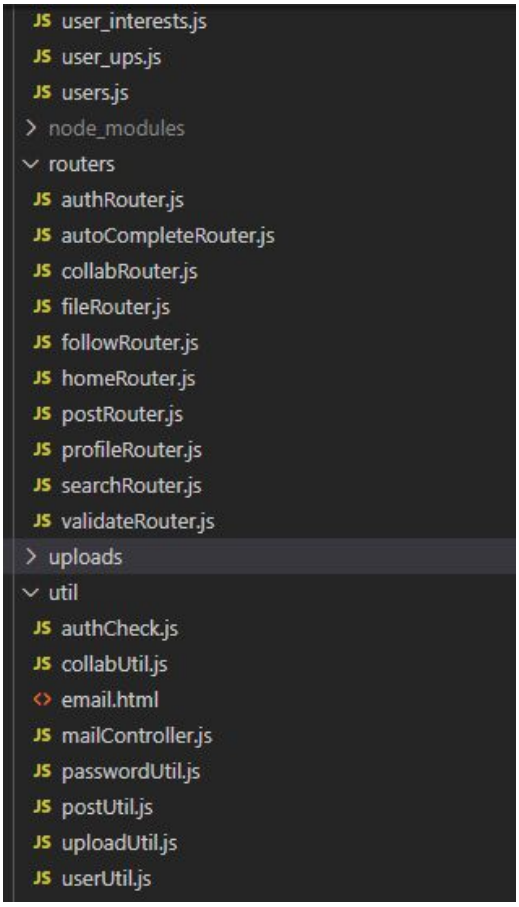
We tried to maximize our usages of issues, as they really helped on the process and prevented the minor things from going forgotten and lost.

Other branches we had were as in the picture



The code directory looks as belows:





model folder contains the database tables and initialization of the database, routers are about the paths of endpoints, and controllers is where we implement the logic of our server. util is the folder where we implement auxiliary functions to be used throughout the project. uploads folder is where we store the static files such as profile pictures or project files. The name is pretty self-explanatory for the Dockerfile. And finally app.js is where we initialize our server.

Mobile

Team:

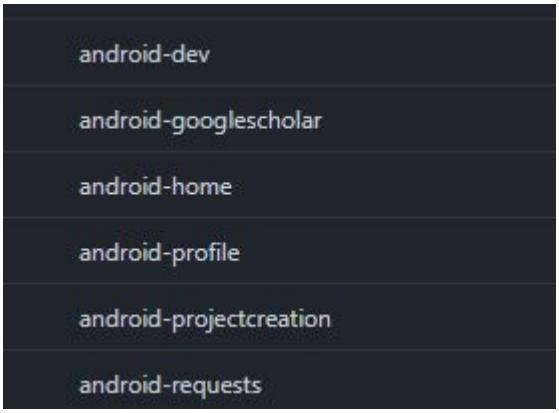
- Ezgi Gülperi Er
- Cihat Kapusuz
- Doğukan Kalkan
- Gülsüm Tuba Çibuk

Tools Used:

- Android Studio (IDE)

After every lab session we hold a subgroup meeting as the mobile team where we discuss what we have done so far , whether there are any issues we can't solve and lastly we decide on the following week’s tasks.

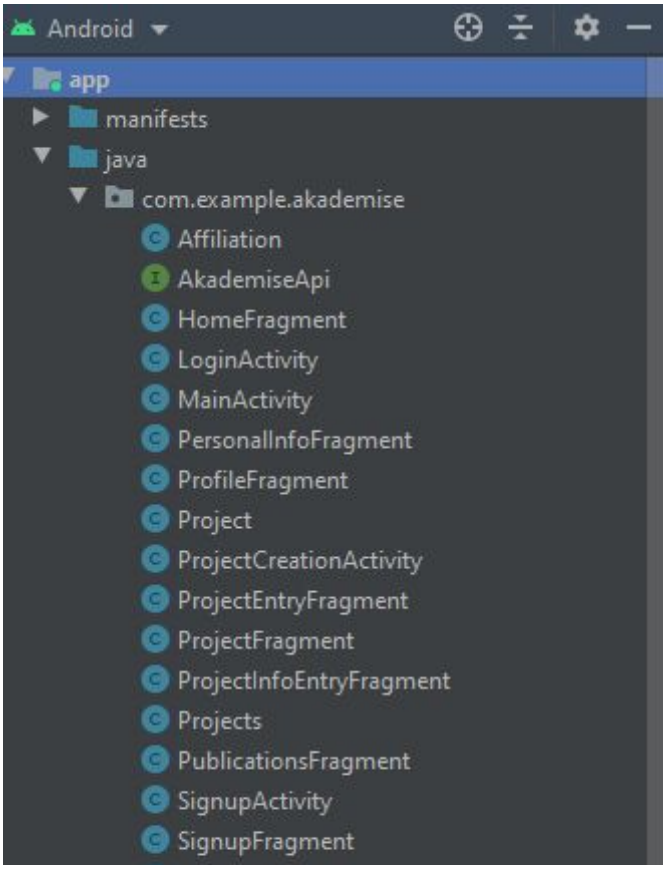
Second Milestone
After the first milestone, we decided to change our workflow to improve our productivity and efficiency. Instead of working the same branch for every feature we are developing, we opted for creating a new branch for new features.
Below, you can see the branches we have worked on:



-android-dev is our master branch.
-android-googlescholar : page where we show details about user's google scholar profile (stats, citations etc)
-android - home : home page, "news feed"
-android - profile : features related to profile page (editing / adding / deleting personal info)
-android -project creation : features related to project creation
-android - requests : request and invitation feature

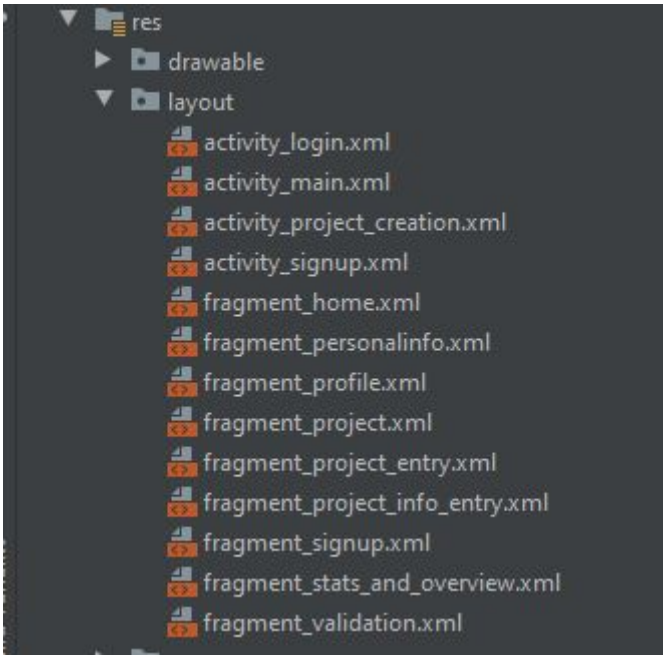
When the task assignee finishes the corresponding task, s/he creates a pull request and mostly assigns another team member (preferably the one whose work is related to / affected by this pull request) to review her / his changes.
After they resolve conflicts (if exists), new feature is merged into the master branch.

The directory structure is similar to Milestone1 and it is as follows (does not include all class, please visit repository to see full class list) :



We have activities as our main components and fragments for the features we want to implement for that activity.
We also used classes to represent objects (such as User,Project) to make the code structure more organised and to make it easier to communicate with the backend.

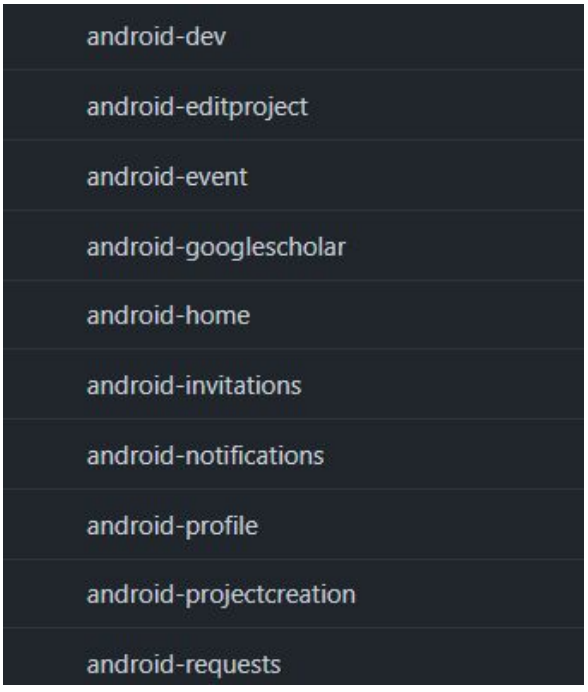
Each Activity and Fragment has an xml file as a layout.



Final Milestone:

Team structure ,job division and code structure were similar to Milestone-2 . We also added a new folder for unit-tests

Final list of branches:



Evaluation of Tools and Managing the Project

Project Management

- We had an already established project management process in the first milestone and we kept at it. We had some internal delays but I believe in the end we managed to keep up with our project plan and delivered all planned deliverables for the 2nd milestone. Although that speaks for itself I would like to add that we had a decent communication inside and between subgroups. The process was smooth and well-defined.
  -Cemre Efe Karakaş

- Overall, we were good at following our plan and delivering our promises to the customer. In the final milestone as well, we implemented all of the functionalities we have set as a goal. We again set an internal milestone as android group and finalized our code before the milestone deadline.
  Team communication and task division was good. Everyone was responsible and asked for help without jeopardizing the flow of the project plan. We did some zoom sessions to solve problems together.
  -Ezgi Gülperi Er

- I think our project management was spectacular. We arranged our project plan according to certain needs and talked about how to shape our plan efficiently. While doing this, we set ourselves realistic goals. We used our weekly meetings effectively and checked where we were on the plan, this helped us keep pace with the project plan. In addition, by putting internal milestones within the team, we had the chance to control what we did just before the real milestone. Thus, we reduced the errors that may occur. Subgroups worked in harmony with each other. All of these took place within the framework of our experiences we gained from last semester's Cmpe 352 course. In general, the whole team did a great job.
  -Hazer Babür

- We did a good job while managing the project. In the first week, we continued with our subgroups and talked about new features that we plan to add to our project. After that we let the teams decide their own task division to catch up the Milestone functionalities. Sub Teams performed good jobs about their task division and fulfilling their responsibilities. Even if some of the group members worked on the project in the last days, we completed most of our tasks in a given time according to the project plan deadlines. Changing to PostgreSQL from MySql affected some of the working endpoints. It created some bugs that we need to fix first instead of implementing new features. It slowed our development process. I appreciate the work done by my group members.
  -Cihat Kapusuz

- We had nice and effective project management. We made necessary discussions in sub teams. And we also decided task division, necessary functions, and the deadline beforehand. Furthermore we coordinated subteams weekly and had necessary discussions for unclear parts where there was a need for communication. Thus the

team did a good and effective job in the project management. Good work!
-Ömer Faruk Özdemir

- As frontend team, in the time span of milestone 2, at first we all had a delay due to our heavy work load. But that didn't last long. We arranged meetings and kept up the communication between us via the possible channels (slack, whatsapp, github issues). We started planning the issues/tasks we had by listing them in detail, after that we started working and separated the tasks between us one-by-one. It was a well managed process in my opinion, since we are in touch with each other all the time and everyone was being responsible about the process. Eventually, we succeeded to deliver our expected features for milestone 2.
  -Ali Ramazan Mert

- In general, as a team, we have done a great job in managing our project. Every single team member tries to attend the classes and lab sessions after the classes. In lab sessions, we discuss what we have done so far and what we will be doing in the upcoming weeks. Everyone does their best to be active and energetic in the discussions. We share our problems with one another and always try to come up with a solution that works for everyone in the team.
  -Doğukan Kalkan

- We did pretty good as the backend team, we had implemented our endpoints in the earlier weeks of the milestone 2 and then we just did the bug fixing and small changes after that. However there were some problems we faced, and those were pretty costly. Our server crashed twice and our URL changed once, this affected not only the backend team but the whole group. Also our database got hacked twice and the data inside it got stolen, because we were using relatively easy passwords. Now we switched to a stronger password and it seems to be holding up.

Frontend

Tools Used:

- Figma (collaborative design tool)
  - I think figma is a great tool that facilitates the design process. With a ready design we can build our website by following the blueprint provided by the design. Figma gives us a lot of useful information about our design such as readily available CSS for different components, component hierarchy etc.
    -Cemre Efe Karakaş
  - Figma is an easy to learn design tool. It enhanced our design process and allowed us to consider some design ideas before we dive in the code. Having the visual interpretation of the product beforehand helped a lot while

implementing the related pages and components.
-Emilcan Arıcan
    - Figma is one of the most used online collaborative design tools in the current environment. Using it with the frontend team members was a good experience and helped developing UI/UX for our frontend pages considerably.
-Ali Ramazan Mert
- Yarn (package manager)
    - Compared to npm, yarn is more secure, consistent and resource-friendly. I think it is a great tool with no down sides compared to npm.
-Cemre Efe Karakaş
    - yarn is a package manager alternative to npm. I have used both of those according to my experience; both of those tools are equally powerful and reliable.
-Emilcan Arıcan
    - yarn is starting to be used more compared to npm due to its reliability nowadays. I was the one suggesting using yarn and I'm pretty happy with it.
-Ali Ramazan Mert
- React (frontend library)
    - I am indifferent to react. It takes care of many functionalities which otherwise require a lot of manual work but still react requires a lot of manual work in other cases. Personally I'd prefer native javascript if it wasn't for Antd's components.
-Cemre Efe Karakaş
    - React is an excellent tool with a huge community. It eases many cumbersome processes and if there is a problem at any point, you can get help very quickly due to the online community. I think one of the best features of the react is to be able to create reusable components, so that we do not have to write repetitive codes.
-Emilcan Arıcan
    - React is one of the biggest frontend development frameworks in the current tech industry. It actually started to overpower its rivals. I am the one with the most experience with React, so it is a good experience while practising my knowledge and helping my teammates in some cases. It is an easy-to-start hard-to-expert framework and provides lots of space and freedom to developers as well as millions of packages that are implemented by others and ready to be used in any react project.
-Ali Ramazan Mert
- VSCode (ide)
    - Great ide.
-Cemre Efe Karakaş
    - VS Code is a great IDE. Users can customize it with extensions. So it can meet many specific needs.
-Emilcan Arıcan
    - VS Code is almost like the greatest and the most used IDE, at least in the frontend community. It provides a lot of extensions and packages which makes the development's overhead parts really fade away. It is an alive ide where it still keeps growing rapidly and has a big community.
-Ali Ramazan Mert
- Ant Design (react UI library)

    - A really nice ui library. Provides many readily-available components for a lot of cases. I think it is a good addition to the project.
      -Cemre Efe Karakaş
    - Ant Design is one of the most popular UI libraries that is used with react. It has many available components. It eases the process of the creating web interfaces.
      -Emilcan Arıcan
    - Antd is a UI library provided for react(also for other frameworks). It has the most github stars among the other ui libraries and I have lots of experience on it. So, I'm really glad of it. Although there are some cases where antd doesn't satisfy your needs by itself.
      -Ali Ramazan Mert

Backend

- Express.js
  Express.js is an open source and minimalist node.js framework. It is simple and easy to use, and takes care of many functionalities with the help of npm packages.
  -Ömer Faruk Özdemir

- Express.js makes a lot of things easier than it is. Last semester, we used Flask as a framework, I can definitely say that express is better. It has lots of useful modules and usage of those modules makes backend development very efficient and on point.
  -Hazer Babür

- Sequelize ORM
  Sequelize is a promise-based Node.js ORM for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server. It makes database access and usage easier, and with its abstraction it makes DB closer to human thinking.
  -Ömer Faruk Özdemir

- At first I wasn't sure if it would be good to use Sequelize. I was used to MySQL and idea of using direct SQL commands for interacting with database was appealing. However, as I used Sequelize ORM I saw that Sequelize makes code more readable and it is as good as using direct SQL commands
  -Hazer Babür

Mobile

- Android Studio (IDE)
    - Android Studio is complex and huge. It is dedicated to developing android applications. So that it has an emulator to help us test the application without installing apk to our phones. It has an enormous amount of tools and features. It is also integrated with Git. However, there are some downsides of

this IDE. One of them is it consumes a lot of RAM, and it is so slow that I waited more than 20 minutes for the emulator to open. Sometimes the IDE and emulator did not respond and I had to close it and open again. Despite all these discouragements, we decided to use this IDE because other methods to implement an android application are way harder.
    - Gülsüm Tuba Çibuk.
- I can not work in an emulator in Android Studio. It never opens well-functioning and I know times that I have waited half an hour to see it open. Most of the time, I connect my Android device with the Android Studio to study on the project. My computer is a little old and freezes from time to time while I was working on Android Studio. It might be my computer's problem but I think Android Studio has some optimization problems. Also, You can edit layout through design instead of code. That is very good functionality but it has plenty of bugs. You may spend an hour to put a textbox to the desired location in the layout. Instead of negative feedback, inspecting the code feature of the Android Studio is amazing. It shows each place when the error occurred and it helped a lot while solving lint error problems.
    - Cihat Kapusuz
- Android Studio is the perfect tool for building and developing Android applications. It has a great directory concept in which we can find and manage our files. It also has what is called Android Virtual Device Manager with which we can install different emulators in which we can test our code. Similar to other IDEs, Android Studio has a built-in Version Control System for its users to make the commit and push processes faster and easier.
    - Doğukan Kalkan

Assessment of the Customer Presentation

Before the presentation, Tuba Çibuk and Emilcan Arıcan created a scenario in which not only the new functionalities but also some of the old functionalities could be presented in an efficient way. The reason for presenting old functionalities was to show updates. And they also did several rehearsals before the actual presentation to warm up and get rid of complications that could occur during the presentation.

Our presentation consisted of three parts, namely the Frontend part-1, Android part, and the Frontend part-2. The Android part was performed by Tuba Çibuk and the Frontend parts were performed by Emilcan Arıcan. During the presentation, the other team members were given the task to take notes about the comments on the presentation made by the instructor and the teaching assistants and how it could be further improved. We managed to complete our presentation in 15 minutes.

There were almost no negative comments on our presentation. However, the event concept in our implementation should be more clarified. Our instructor suggested that there should also be an ending date of the event since some events may take more than one day. Our instructor said that some events may include smaller multiple events and the date section should allow entering multiple dates. And also it can be argued that it would be a more precise approach to implement the events. One suggestion from the previous presentation was to create profiles of team members since in this way there should be no consent issues regarding profiles. Therefore we used the names of our team members.

We also considered the comments on the first presentation. Deadlines of the projects were not clarified much, but we updated it as the suggestions and showed the final version of the deadlines and milestones in the final presentation.

As a team, we think that we made great work.

Deliverables

Frontend

- **Signup Page:**
  Complete as planned.
- **Home Page:**
  Complete as planned.
- **Landing Page:**
  Complete as planned.
- **Project Creation Page**
  Complete as planned.
- **Search Page:**
  Complete as planned.
- **Logged Off Header Component**
  Complete as planned.
- **Logged In Header Component**
  Complete as planned.
- **Profile Sider:**
  Complete as planned.
- **Profile Page:**
  Complete as planned.
- **Project Details Page:**
  Complete as planned.
- **File Edit Page:**
  Complete as planned.
- **Followers and Followings Page:**
  Complete as planned.
- **About Us Page:**
  Complete as planned.
- **Create Event Page:**
  Complete as planned.
- **Edit Event Page:**
  Complete as planned.
- **Event Details Page:**
  Complete as planned.
- **Events Page:**
  Complete as planned.

- **Forgot Password Page:**
  Complete as planned.
- **Saved Events Page:**
  Complete as planned.
- **Delete Account Modal:**
  Complete as planned.

Backend

- **Login:**
  Complete as planned.
- **Sign in:**
  Complete as planned.
- **Homepage:**
  Complete as planned.
- **Project Creation**
  Complete as planned
- **Search Page:**
  Completed the basic functionality. Will be developed.
- **Validation Code**
  Complete as planned.
- **Profile page**
  Complete as planned.
- **Auto Complete Lists**
  There are updates in mind, completed the basic functionality.
- **Requests and Invitations**
  Complete as planned.
- **Profile page improvements**
  Complete as planned.
- **Follow system and ups**
  Complete as planned.
- **Home page**
  Completed as planned. Might be improved.

Mobile

- **Login Page:**
  As planned.
- **Signup Page:**
  As planned.
- **Validation Page:**
  Completed as planned.
- **Affiliation Page:**
  Completed as planned.
- **Profile Page**
  Profile page is completed.

- **Project Page**
  Project page completed as planned.
- **Project Details Page**
  Completed as planned.
- **Project Creation Page**
  Completed as planned.
- **Edit Project Page**
  Completed as planned.
- **Search:**
  Needs future work. (There is semantic search but no filtering)
- **Home Page:**
  Completed as planned.
- **Google Scholar Page**
  Completed as planned.
- **Request Page**
  Completed as planned.
- **Notifications**
  Completed as planned.
- **Files Page**
  Requires future work but initial view and basic functionality is done
- **Invitations Page**
  Completed as planned.
- **Events Page**
  Completed as planned but the Update part needs some more work.

Evaluation of the Deliverables

Frontend

- **Signup Page:**
  - It is implemented as three tabs which makes it aesthetically pleasing. Functionalities of this page is working properly as well.
    -Emilcan Arıcan
  - Signup page was implemented nicely in a modular way, in the last section there was a tag choosing choice we had to make and now it is done nicely.
    -Ali Ramazan Mert
- **Home Page:**
  - Implemented as it is designed. User and project recommendations added at milestone 2.
    -Emilcan Arıcan
  - Homepage design is changed a bit and I think it looks really nice like in a real social media.
    Ali Ramazan Mert
- **Landing Page:**

    - The landing page looks complete, and it has a professional feel, especially with the login modal.
      -Cemre Efe Karakaş
    - Landing page design is responsive and easy on the eyes and login modal is a nice touch at this implementation.
      -Emilcan Arıcan
    - Landing page is really catchy and it adapts the trending UI/UX points such as Call-to-action (CTA) button. (JOIN)
      -Ali Ramazan Mert
- **Project Creation Page**
    - It has been reworked and it is now reliable, responsive and versatile.
      -Cemre Efe Karakaş
    - Revisions that are planned to be made after the milestone 1 are implemented. Now, it is more reliable and spot on implementation.
      -Emilcan Arıcan
    - Smooth and responsive page.
      -Ali Ramazan Mert
- **Project Edit Page:**
    - The project edit page is fully implemented and works without known bugs. It allows the user to alter a project to their liking. I believe we have done a good job at implementing this page.
      -Cemre Efe Karakaş
    - In the previous version there were some issues like responsiveness and input limits. They are fixed now and the edition is well defined and enjoyable.
      -Ali Ramazan Mert
- **Search Page:**
    - It works as intended. Returns results precisely and in an easily viewable manner. It only lacks semantic search which will be connected with the backend when ready.
      -Cemre Efe Karakaş
    - It functions well, and displays results as intended.
      -Emilcan Arıcan
    - Search results are viewed and filtered nicely.
      -Ali Ramazan Mert
- **Logged Off Header Component**
    - It is a good and stylistic implementation. It seems overall responsive but in the range of 500-600 pixels, the drawer button acts strange. It should be fixed.
      -Emilcan Arıcan
    - It is responsive and satisfies our needs but could be improved further.
      -Ali Ramazan Mert
- **Logged In Header Component**
    - Simple, effective and faithful to the original design on figma.
      -Cemre Efe Karakaş
    - Responsive and nice implementation as it is designed.
      -Emilcan Arıcan
    - New design is so nice and the responsiveness issue is solved.
      -Ali Ramazan Mert
- **Profile Sider:**

    - Looks good. It is fully implemented, changed according to needs and it works as intended.
      -Cemre Efe Karakaş
    - In the latest version it has all the functionalities working and it gives a really nice UI element to the homepage.
      -Ali Ramazan Mert
- **Profile Page:**
    - It is implemented well and usage is very intuitive. Users can edit many fields related to their own profile.
      - Emilcan Arıcan
    - The profile page is exquisitely implemented, works well and fits the user's needs. It is completed as of milestone 2.
      -Cemre Efe Karakaş
    - Profile page is looking and working perfectly in my opinion. More than expected is implemented on it.
      -Ali Ramazan Mert
- **Project Details Page:**
    - Users can display many aspects of a project, and it is easy on the eyes. Colors are used to communicate with the users more efficiently especially at milestone bullets and the labels. There is a known bug related to download of non-text based files, but it will be solved as soon as possible.
      -Emilcan Arıcan
    - Project detail page has a really good design and satisfies the needs nicely.
      -Ali Ramazan Mert
- **File Edit Page:**
    - Clean and elegant implementation far from features that might distract users.
      - Emilcan Arıcan
    - Clearly implemented. It is easy to use and intuitional. It unfortunately incorporates a known bug that will be fixed until the final milestone.
      -Cemre Efe Karakaş
    - There is a bug for updating the file but other than that it has some unique UI.
      -Ali Ramazan Mert
- **Followers and Followings Page:**
    - Does what it is intended to do without bugs. We plan to add additional features to make it more convenient.
      -Cemre Efe Karakaş
    - A clean interface that lists followers / followings. It achives the desired functionalities.
      - Emilcan Arıcan
    - It was an extra implementation and it is a really nice functionality
      -Ali Ramazan Mert
- **About Us Page:**
    - It has a brief explanation about our group. I think it captures the desired aspects.
      - Emilcan Arıcan
    - Describes our work and purpose well enough.
      - Ali Ramazan Mert

- **Create Event Page:**

    - Users are able to create the events with ease. It satisfies the necessary functionality.
      - Emilcan Arıcan
    - Satisfies our need of creating an event with a good UI/UX.
      - Ali Ramazan Mert

- **Edit Event Page:**
    - Users can edit any field of an event. It has a quite similar design with the event creation page. It is clean and easy to use.
      - Emilcan Arıcan
    - Edit page is basic and does what it needs to do.
      - Ali Ramazan Mert
- **Event Details Page:**
    - Event details page has a clean design and is easy on the eyes. I think our team made great work on this one.
      - Emilcan Arıcan
    - Event is displayed with its details with a clean UI.
      - Ali Ramazan Mert
- **Events Page:**
    - This page has a nice implementation that lists the event results.
      - Emilcan Arıcan
    - It was needed to show current event list at somewhere and this page solved that problem.
      - Ali Ramazan Mert
- **Forgot Password Page:**
    - This is a simple page that allows users to change their password. It is working very well.
      - Emilcan Arıcan
    - Forgot password was a necessary function for the final product and it is implemented nicely..
      - Ali Ramazan Mert
- **Saved Events Page:**
    - This page is almost the same page with the events page. As the events page, this page has a nice implementation that lists the event results.
      - Emilcan Arıcan
    - Listing the saved events is a good functionality for “saving events” to be actually meaningful. So it is nice to have that page.
      - Ali Ramazan Mert
- **Delete Account Modal:**
    - User can delete his account with a simple modal and it provides a popconfirm to prevent such unwanted misclick actions.
      - Ali Ramazan Mert

Backend

- **Login:**

    - It is fully implemented and works as intended. You need to provide your email and password, then the requester will be granted of the userId of loginer and the JWT Access Token
      -Muhammed Enes Toptaş
- **Sign in:**
    - It is fully implemented and works as intended. You provide email, password, name and surname, then if email is unique, the user will be created. Then the requester will get the JWT Access Token and userId of the user. Also a validation code will be sent to their emails.
      -Muhammed Enes Toptaş
- **Homepage:**
    - It is implemented but changes will be made. Right now, homepage includes recommended projects that have tags which user interested in, also includes recommended users to follow according to user interests. It functions well but recommendation functionality can be improved further up to milestone 3
    - -Hazer Babur
- **Project Creation**
    - It is implemented but changes will be made according to what the professor asked us to do. Right now, the requester needs to provide userId, title, abstract, privacy, status, deadline and the requirements of the project. Files can also be included in the request.
      -Muhammed Enes Toptaş
- **Search Page:**
    - It is like a placeholder right now, and will definitely be developed more. Will be a semantic search and will work on multiple fields of the data. Right now when provided a search query, it looks up the database to find similar strings to the given query, also received a type variable to search in the users or in the projects
      -Muhammed Enes Toptaş
- **Validation Code**
    - It is fully implemented and works as intended. Accepts a validation code and userId and checks if it belongs to that user, if it does, then the user gets validated.
      -Muhammed Enes Toptaş

- **Profile page**
    - It is implemented and works as intended but definitely will be developed more. In this endpoint, we return everything related to the user to the requester. This includes emails, their names and surnames, their affiliations and research areas. In future, we plan to also return their projects.
      -Muhammed Enes Toptaş

- **Requests and Invitations**
      Collaboration requests (participation requests and invitations) are implemented and functions very well. Requester user id, requested user id, project id and request type should be stated in the post request in order to add this collaboration request into database. And when these requests are accepted by the requested user, the requester user is added as a collaborator to project.

-Hazer Babur

- **Auto Complete Lists**
    - It is implemented and works just as we wanted. When a user selects something from a dropdown menu, there are some suggestions, and if they are not satisfied with these suggestions, they can add new things to that list. We plan to implement a filter to ban bad entries from being added there..
      -Muhammed Enes Toptaş
- **Home Page**
    - There are developments in mind, but currently is working just fine and can be left as it is now. We now suggest projects based on the user's followed accounts and their interest areas. We also provide users based on 3 things: common interest areas, common universities and common departments.
      -Muhammed Enes Toptaş
- **Profile page improvements**
    - Implemented as intended. We added biography, google scholar related fields and profile pictures to the profile page.
      -Muhammed Enes Toptaş
- **Follow system**
    - It is implemented and works as intended. Users can follow each other, every profile is public so there is no need to accept or reject a follow request, or even a follow request. One can directly follow.
      -Muhammed Enes Toptaş

Mobile

- **Login Page:**
    - Login page is simple. A user can easily understand what to do. There are simple edit views with explanatory hints. If a user does not have an account, s/he can easily find the sign up button on the screen.
        - Gülsüm Tuba Çibuk
- **Signup Page:**
    - Sign up page is also simple. It requires only 4 pieces of information. Email, password, name, and surname. Easy! We did not fill the page with the full procedure of sign up. It is step by step. Then the user is directed to the validation procedure.
        - Gülsüm Tuba Çibuk
- **Validation Page:**
    - This page requires only one thing: Validation code! After the user provides his/her email, s/he gets a code as a mail. Then the user should only enter the code. This step is done easily, also.
        - Gülsüm Tuba Çibuk
- **Affiliation Page:**
    - This page comes after validation and it is the final step of the registration. It consists of affiliation and research tags. Affiliation represents the university of the user, department of the user and degree of the user. Research tag represents the users' interests in a tagged fashion to ease the searching

process. All of this information can be selected via dropdown combobox, also user can add new research tag and affiliation if her/his affiliation is not listed.
    - Cihat Kapusuz

    - User interests (tags), university, department, and title lists are taken from the database by Milestone 2. A User can add a new university, department, or a title if it is not listed.
        - Gülsüm Tuba Çibuk

- **Profile Page:**
    - This page displays the profile of a user, in which all the information about a user is shown, namely a brief 'About Me' part, an affiliation part, a contact part, the user's name and a profile picture. There are also two buttons that take us to statistics and overview page and projects page. For now, we only display a hardcoded profile. We will be modifying this page so that it could display the profile of the user that is logged in.
        - Doğukan Kalkan
    - By Milestone 2, profile page is functional. It gets the information from the backend and displays profile picture, bio, research interests, university, department, and title. Google Scholar information, projects can be reached from profile page clicking related buttons.
        - Gülsüm Tuba Çibuk
    - With the completion of Milestone 2, there are now 2 kinds of profile pages. One is for the profile page of the user logged in and the other one is for the profile pages of the other users seen by the user logged in. A user sees edit and logout options in his/her profile and sees follow/unfollow and upvote/downvote options in other users' profiles.
        - Doğukan Kalkan

- **Project Page:**
    - Users' own and collaborated projects are shown here, we acquire the whole projects from the get requests but we show the titles and the abstracts of the projects. Users are able to click projects and go to their details page to see all features of the project.
        - Cihat Kapusuz

- **Project Creation Page**
    - There is a button on the project page with a plus sign. It redirects us to the project creation page. This page is the first prototype of the project creation. It is not fully functional yet, but it shows the general aspects of creating a project. New functionalities like uploading pdf files will be added soon. It has fields for title, abstract, tags, requirements, and deadline. It also has a check view to specify it as public or not. It is a simple version of the project creation but it will be more specific.
        - Gülsüm Tuba Çibuk
    - Project creation consists of 2 pages. User enters the title, requirements of the project and selects tags and privacy of the project at first page. At the second

page, the user enters a summary for the project and adds a milestone to the project.
    - Cihat Kapusuz
- **Search:**
    - Search functionality is detailed by Milestone 2. Users can be searched and seen detailly. Projects also can be searched and seen detailly. From a searched project its creator profile page can be reached. Searched users projects and its details can be seen.
        - Gülsüm Tuba Çibuk

- **Home Page:**

    - On the homepage, we show the recommended projects. Recommended projects are shown with respect to user's research tags, followed users and collaborated projects. Projects' titles and abstracts are shown to define the projects. To see recommended projects in detail, users can click on them to see it in detail.
        - Cihat Kapusuz
    - By Milestone 2, the homepage has a recommendation system. It recommends projects according to user interests and followings. Project details can be seen by clicking on them.
        - Gülsüm Tuba Çibuk
    - By the Final Milestone, the homepage recommendation system also recommends users. Profile pages can be seen by clicking on them.
        - Gülsüm Tuba Çibuk

- **Project Details Page:**
    - It has 2 versions, one for the owner view and the other one is the visitor view. Owner of the project can see the details of the project and s/he is able to edit or delete the project. Visitor view is shown when a user visits a project's detail page while not contributing to the project. S/he can see the details of the project and request for collaboration but is not able to edit or delete the project. Also, s/he can go to the project owner's profile by clicking the name of the owner of the project. Also, collaborators can leave project from this page.
        - Cihat Kapusuz

- **Edit Project Page:**
    - In the edit project page, the user is able to edit the title, summary and requirements of the project. Also, s/he is able to change status and the privacy of the project. In addition to those, users are able to edit tags and milestones of the project with clicking edit tags and edit milestones buttons. In these pages users are able to add or delete tags and add, update and delete milestones of the project.
        - Cihat Kapusuz

- **Files Page:**
    - Users can see files of a project on the files page. Users can come to this page through the project details page. Users are only able to see the names of the files of the project on mobile for now.
        - Cihat Kapusuz

- **Requests and Invitations**
    - Users can see who has sent request to collaborate on user's project. She can access the "Requests" page from the project details page.
    - Users can also accept or reject the request from the same page.
    - In Milestone 2 there was no search bar for invitation. For final milestone search functionality is added to invitations page
    - Users can see received invitations and from the same page, user can accept or reject the received invitation for another project.
      -Ezgi Gülperi Er
- **Google Scholar Page:**
    - By Milestone 2, Google Scholar page can be reached from the profile page by clicking the related button. Users can see their google scholar information and also other users Google Scholar information from their profiles. New url for Google Scholar account can be typed in their own Profile. Total citations and publications can be seen on this page.
        - Gülsüm Tuba Çibuk

- **Events Page**
    - Users can see the events in the system as well as their own events.
    - Users can add an event to the favorites.
    - Users can remove an event from the favorites.
    - Users can add/delete/update an event.
    - Users can see the events page via the bottom navigation menu. In the events page, they can see the events in the system, and they can get to the My Events page via tapping the My Events button.
        - Doğukan Kalkan

- **Notifications**
    - Being followed by a user, being accepted or rejected to a project, being invited to a project, acceptance or rejection of an invitation, being removed from a project, collaboration requests, and collaboration invitations are shown on the notifications dialog.
        - Gülsüm Tuba Çibuk

Summary of Coding Work

Name Surname	Summary
Cihat Kapusuz	**Milestone 1** -I implemented the general flow of the sign up on Android and the first draft of the sign up page. -I implemented and tested the view of the projects page on Android. -I followed the Retrofit tutorial which is suggested by Tuba. -I implemented search functionality with respect to title and abstract with the usage of the get request. I used the Retrofit library for this purpose. -I attended all of the Android team meetings and took an active role during the task division. -Tuba checked the functionalities of the Android app before the demo and I assisted her through Zoom. Also,we added final small changes to the code. -In addition to code related work, I prepared the Android demo's scenario with the help of the Android team and attended rehearsal sessions, and also presented the demo of the Android with the help of Tuba. ---------- **Milestone 2** -Applied the design that we created on figma into project creation related pages. -Modified project creation with respect to the feedback from the Milestone 1. -On request and invitations page added link to the requester's profile for enabling the project owner to check the qualifications of the user before accepting or rejecting a request. -Added link to the project owners profile from clicking her/his name when a user searches for a project and wants to see project details. -Added files page into project details page, initial list of files view added. -Worked on how to add a file into a project through android but failed on it after several tries -Created pull request for android-projectcreation branch after completing the related adjustments -Merged pull requests for team members and resolved conflicts if they exist before merging -Explored web application of the Akademise for bugs and reported bugs to Frontend team

	-Attended rehearsal session and commented on the scenario will be presented on the Milestone 2 presentation -Fixed some minor bugs that we faced during development process (can be seen on the commit messages) -Attended sub-team meeting to discuss project status and helped my teammates when they faced a difficulty ---------- **Final Milestone:** -Implemented edit project page on Android, also implemented edit milestones and edit tags pages to be able to edit all fields of the project -Implemented delete project functionality -Implemented refreshing contents when an update is occured about project details or list of projects. -Implemented above features in android-editproject branch and created pull request -Fixed project's privacy bug when it changed into boolean from integer and adjusted related api calls and object fields according to this change -Prettified project details page for both owner perspective and non-owner perspective, showed tags and milestones of the project in recycler views, also adjusted color of the status with the frontend -Fixed collaborator of the project can request collaboration bug, requesting collaboration replaced with leave project if the user is already collaborator of the project -Added milestone adding feature while project creation -Implemented above features in android-projectcreation branch and created pull request -Transferred backend's base url to the string.xml file to be able to change it easily when deployment url is changed -Commented java files that are assigned to me -Implemented unit test for entering valid date format for the milestones -Fixed lint error while creating APK with inspecting code and prepared APK for the mobile
Hazer Babur	**Milestone 1:** -searched and found a way to connect

	Expressjs to database without using Sequelize -implemented project creation, updation, deletion related endpoints and methods, also implemented a method that gathers projects from database according to given parameters -implemented Project related database models and relationships between those models with Sequelize -contributed to general design of backend architecture **Milestone 2:** -implemented project tags related endpoints & functions for adding, updating, deleting project tags -implemented project milestones related endpoints & functions for adding, updating, deleting project milestones -implemented project files related endpoints & functions for uploading, updating , deleting, getting project files -implemented collaboration related endpoints & functions such as adding collaboration(invitation and participation) requests to database, deleting collaboration requests, adding collaborators to projects, and deleting them from projects -implemented project file, project collaborators , collaboration requests, project tags, project milestones database models -implemented homepage post recommendations with Hamza Işıktaş and Enes Toptaş -solved server issues with backend team -informed frontend team about bugs and created related issues **Final Milestone:** -constructed the logic behind notification mechanism -implemented notification related endpoints and methods that adds, deletes notifications when collaboration occurs or collaborator removed from collaboration -updated functions that are about adding and deleting collaborators of projects -implemented notification model -implemented routers for notifications -detected some bugs helped team to solve those bugs. -reviewed pull requests

Ali Ramazan Mert	**Milestone 1:** -Created the React project from scratch and implemented the homepage design and some components on it. -I was the one with the most experience with React, so during the frontend process, I actually prepared some kind of workshops for the team, where each took 1 hour on average. -In general I designed the project structure and chose the necessary npm packages we needed and told the team how to use them properly. -Created a theme file to keep our static theme like colors etc. -Integrated redux to the project from scratch -Created a components folder and put my landing header there in order to make others use it easily -I created a template axios object to make our api calls with it. -I handled async operations on the homepage, like logining. -I implemented the authentication/ authorization logic where we use the local storage to keep our token received from the login request. -I helped Cemre on sign up page in general. -I helped Ömer in the project creation page. -I made the whole deployment/dockerization part on the frontend. -I did a lot of research and testing on CI/CD. -I set up the domain redirection with Cemre. **Milestone 2:** -I built a github CI/CD action to update and rerun our project in the ec2 instance when we make a push or pr to the frontend branch -I also helped backend team to add this CI/CD action in their own configured version -Improved the UI/UX in the homepage on frontend(all features will be about frontend from now on) -I created a notification red alert (1) when user has a new notification it shows the number of pending notifications in a red small indicator -I added google scholar linking functionality, you can enter your google

	scholar link and your profile is updated with respect to the link you provided -I added a router link to the “projects” menu item in the homepage left sider menu -I redesigned the UI/UX of the entire profile page(all below will be about profile page from now on) -Added buttons to follow or unfollow a user -Added indicators for followers/followings/publications -Added invite button for a project creator to invite that user to a project he will choose after clicking the invite button -Added an edit button (visible only on own profile) where you can edit your profile’s general info -Added changing profile picture functionality -Added about me section and user can edit it on place if it is user’s own profile -Below added a section for google scholar projects where it is updated when you provide a google scholar link in the homepage left sider menu -Both in the signup (section 3) and the profile editing pages, I recreated the select boxes where user can either choose a tag/university etc. from the database or user can add a new tag/university etc. to the database and choose the newly created of his own -Solved many of bugs and unwanted cases in my frontend team member’s works -Analysed,reviewed, commented on pull requests of frontend members -In some cases our EC2 instance was failing to update the project code and was out of memory, i.e broken completely. I had to fix the instance and solve the occuring problems and reverted it to its working state -During the process of implementing the frontend, we were actively in communication within the frontend team and shared the work, helped each other and assigned necessary leftover work to the members without tasks successfully **Final Milestone:** -Implemented change password functionality and needed pages -Implemented forgot password functionality and necessary pages -Implemented delete account functionality and necessary modal -Implemented new activity stream

	notifications and the necessary logic for it
Cemre Efe Karakaş	**Milestone 1:** -I created the signup page from scratch. -I implemented a pagination system for the sign-up page. -I implemented three forms for the signup process. -I used redux to manage the data on the client side. -I prepared async action handlers and dispatchers for signup, validation and addinfo actions (called by forms). -I connected the forms to the backend using axios calls to the existing API endpoints. -I used the tokens and other information returned by the endpoints to save them in client-side local storage to implement a real 'signup' process. -I obtained the domain 'akademise.ml'. Ali and I routed it to the frontend amazon instance using nameservers and elastic IP. -I made design changes on the landing page, the header component, the profile sider component, and the recommendation card component. -I made small contributions to the project creation page. -I used history for managing redirects **Milestone 2:** -I reimplemented several features of the signup page to fit with the new endpoint designs of the backend team. -I reworked the signup page to integrate field recommendations with the backend (The recommendations are directly pulled from the server and the user can add new ones). -I redesigned and simplified the project creation page. I made the page responsible, I reconnected it to the new backend endpoints and I removed some parts that were no longer relevant after the new database design approaches. -I implemented a project edit page that had more fields. This is a page that lets the user better define their project. This includes editing the title and description as well as adding & removing tags, milestones and requirements. -I have implemented a dynamically created form component structure for the project edit page. This component lets the user add an arbitrary number of milestones to

	the project. -I implemented the pages in which a user's followers and followings are listed. -I reworked the homepage with Emilcan. We prettified the page, introduced limitations and rules as to what kinds of information related to the project is displayed on the homepage. We also connected it to the backend's new homepage content endpoints, from which we filter and sort information to better fit the concept of a homepage. -I implemented a page that allows users to upload, view, download, delete and edit (in the browser) a project's files. -Together with my friends from the frontend team, I made sure the user can navigate easily through the website with properly placed hyperlinks. -Worked hard to make renderable react components error prone with necessary null checks. -Implemented the mobile navigation menu. -I changed the design of the project details page to provide a distinction between the project owner and other collaborators.
Ömer Faruk Özdemir	**Milestone 1:** -After communicating with the team, receiving information needed by DB, I created the DB design. -I updated the DB design in the following weeks, with respect to changes in the project discussed with the customer and the changes discussed with the team. -I wrote SQL tables according to DB design. -I wrote the Sequelize ORM models according to SQL tables and DB design. -I prepared Backend's presentation, and presented it. **Milestone 2:** -Create Version 2 of database design and models according to discussion with the team after Milestone 1. -Create autocomplete endpoints. -Help Enes to fix database server after the database attack. -Help Hamza to solve problems on the postman. -Tried to create the homepage endpoint, but failed to do so. -Reviewed some of the PRs. **Milestone 3:** -Create Version 3 of database design and models according to additions necessary

	for Milestone 3. -Create notification get&add endpoints. -Create event get&add endpoints. -Reviewed some of the PRs. -Write requirements for the milestone report.
Gülsüm Tuba Çibuk	**Milestone 1:** Firstly, I created the first template for our Android Project. The first template includes templates for Login, Home, Projects, and Profile Pages. After we distributed the jobs among the android team, I implemented ResearchTagFragment. After all team members did their jobs, we again specified new jobs, and discussed the required fields to communicate with the backend. Then I implemented the first prototype of project creation activity according to these fields. After we discussed with our customer, we decided to remove “papers”. We decided that there will be only “projects” that include some papers. Therefore I adjusted the names according to the customer meeting. Then we started to connect our project with the backend. I created the first template for API interface, get and post functions using Retrofit library. Then I implemented an example get function for search. After the backend deployment is done, we again specified and distributed the jobs among the android team. Then I implemented the post function for the email validation part. Then I implemented the post function to create a new project. After that, I implemented a get method for existing projects of the user. Then I implemented the jwt validation part and I changed the user class to be able to get user id. Finally, I made some changes on the profile page for the presentation. ---------- -**Milestone 2:** -I added recycler view scripts to use on related pages -I changed the scroll view to recycler view on the project page. -I added project details page

	-I adjusted getting token on validation page according to the changes on backend endpoints -getting university, tag, title,department from backend and adding new ones to backend added at research interests page. -personal info and profile page getting and posting methods corrected. -seeing tags added to profile page. -getting projects at project page corrected according to endpoints. -button names corrected -getting user id at login added. -searching users added. -new project class created to correctly get the projects. project details page detailed. -profile page for other users connected with the recycler view. - profile page activity created and related profile fragments connected. -I changed the background and icon colors. -I resolved the conflicts caused by the merges. -I created pull requests and approved other pull requests. -I added google scholar functionality -I added home page functionality -I added seeing another user's projects. ---------- - **Final Milestone:** -I added notification mechanism -I updated the home recommendation mechanism. It also recommends users. -Url change is adjusted -I updated the notification system so that following, joining etc. notifications can be seen. -I added notification deletion functionality. -search bug is fixed -adding events to favorites and removing from favorites added. -I opened pull requests #396, #379, #358 -Merged pull requests #415 #424 #426 -Design of the notifications improved -I added comments
Ezgi Gülperi Er	**Milestone 1:** -In the first part before Milestone 1, I have implemented the PersonalInfoPage and corresponding layout where the user enters his/her personal information and academic affiliation information during sign up. - After talking with the backend team we

	decided to change the structure so I merged two fragments and updated the PersonalInfoFragment and xml file accordingly. - I implemented the post function for sending personal information. -I created a new class “Affiliation” to make it easier to communicate with Retrofit. --- **Milestone 2:** -I have implemented the logout functionality - I have implemented the necessary functionality and xml files for a user to be able to see other user’s profile details. -As mobile team we created a figma document to change the design of our layout. As a part of this task I have modified the project details page (user’s own page and also other user’s profile details pages) - I have implemented the following functionalities : *Sending a collaboration request to a project *Reviewing collaboration requests *Accepting/rejecting collaboration request *Sending invitations to other users - I have created the layout files for the following pages: *seeing the collaboration requests for the project *sending collaboration invitation to another user *accepting / rejecting request - I have reviewed pull request - 253 - I have reviewed pull request - 305 **Final Milestone:** - I have fixed the critical bug which causes sending all invitations to the user with id 1. - I have added search functionality to invitations page. - I have implemented a page for user to see her received invitations, where user can also accept and reject the listed invitations. - Fixed a bug that causes the app to crash when run on a device with API <=26 - Minor fixes on Events page’s design to fix overlapping of texts and button and content wrapping. - I have commented my implementations

	- Design of the pages improved - Implemented a function to check email formatting to avoid non-valid email registration. I have opened pull requests #427 , #424 I have review pull request #403
Hamza Işıktaş	**Milestone 1** -I created the architecture of the backend. -I contributed to creating database -I contributed to connecting the backend to the database. -I deployed the backend to AWS with Enes. -I implemented register and validation related endpoints. -I implemented search endpoint **Milestone 2** - I implemented all follow related endpoints - I implemented all up related endpoints - I built the CICD action in github with Enes Toptaş - I implemented all home page related endpoints with Enes Toptaş and Hazer Babür - I created and managed postman collection - I created the API documentation for the team. **Milestone 3 (Final)** -I implemented home/delete endpoint for user account removal -I implemented semantic search using elastic-search. For this purpose, I created elasticsearch client. I implemented add, update, delete, search elastic-functions for projects and events. -I re-implemented search endpoint from scratch using elastic search. Now the endpoint provides user-search, project-search and event-search with both semantic and non-semantic search functionality. -I wrote searchUtil.js file -I and Enes migrated our database from mysql to postgresql -I and Enes deployed the elastic-search client to the aws. -I and Enes migrated database in the AWS from mysql to postgresql -I and Enes solved the memory problem in AWS by changing our instance type to t2.xlarge. -I and Enes added the tag filtering

	mechanism to search. -I helped to solve db model related issues. Some relations between db models have been linked wrong. -I contributed to closure of the consistent error messaging issue. -I commented on the files created mostly by me. -I contributed to solving many bugs.
Emilcan Arıcan	**Milestone 1** -I contributed to the design of the landing, search, and feed pages. -I created the feed page from scratch. -I created the search page from scratch. -I created a profile info sider component that displays a summary of the user information in the sider. -I created a navigation bar component (logged in version). -I created a content card component to display project or user contents. -I created recommendation card components to suggest projects and users to follow. -I made backend connections of the search and feed pages. -I used useHistory hook to redirect pages. -I send requests to the backend using Axios to get the search results and feed content. **Milestone 2** - I made backend connections to send batch requests. - I created a modal to let user be able to search and select multiple users to invite their teams at the same batch. - I implemented the setting and display of status of a project - I revised the home page with Cemre. We made some cosmetic changes in project recommendation components. We also connected the home page to the backend and sorted and filtered the data in order to make it ready to be displayed. - I made several bug fixes for the home pageimplementation. - I made backend connections of user recommendations - I created notification list and made related backend connections. - I created project details page and made related backend connections. - I implemented the file download logic at the project details page. - I have implemented milestone bullets that

	changes color according to whether milestone date is already passed or not. **Final Milestone:** - I implemented the user agreement modal on the registration page. - I implemented the event creation page and connected it to the backend. - I implemented the event edit page and connected it to the backend. - I implemented the saved events page (by using the events page created by Ömer Faruk Doğru as a base code) and connected it to the backend. - I revised the backend connection of the user selection modal at the batch invite functionality (at the project details page). - I wrote comments on the following page implementations: Create Event, Edit Event, Home, Project Details, Search, Saved Events. - I wrote comments on the following component implementations: Profile Sider, Content Card, Filter Button, Filter Button Small, Main Header, Notification, Person Recommendation. - I have written frontend user manual with Ömer Faruk Doğru.
Doğukan Kalkan	**Milestone 1:** -During the process of implementing the requirements of the Milestone 1, I attended all the meetings with the whole team as well as the meetings with the Android Team. -I updated the LoginActivity, and updated the layout for that Login page. -I created the ValidationFragment page and created the layout for the page. -Then I implemented the ProfilePage and created the layout for the page from scratch. -I created the StatisticsAndOverview Page and created the layout for the page. -I updated the ProfilePage so that the buttons would take us to the correct pages. -I created the User class, and updated the SignupFragment to implement the sign up request. As decided earlier, I used retrofit to make requests to the backend server. -I updated the LoginActivity to implement the sign up request. -I added the functionality to save the tokens that help us make the requests for a user. **Milestone 2:**

	-During the process of implementing the requirements of the Milestone 2, I attended all the meetings except 1. I missed a class and couldn't attend that one. - I created the new designs for our application in figma. - Upon creating the new designs, I changed the following pages according to the new designs: Log in, Sign up, Validation, Personal Info, Profile, Stats and Overview. I updated the layouts of these pages while maintaining the functionality. -For the changes that were to be done, I created a new branch called "android-profile", and at the end of my work, I made a pull request. - I added the edit functionality to the profile page. -I added the upvote functionality to the profile page. -I added the following functionality to the page. -Before the actual demo, I fixed a couple of bugs. **Final Milestone:** **-** For this milestone, I was responsible for the Events-related pages. -I created the following classes, fragments, and activities: AddEvent, AddEventFragment, Event(along with EventFav, EventResult, EventTag, EventUser, MyEvent), EventActivity, EventAdapter, EventFilters, EventFragment, EventOperation, MyEventsAdapter, MyEventsFragment, UpdateEvent, UpdateEventActivity, UpdateEventFragment. -I also created the corresponding xml files for the pages. -Functionalities I added to the application: Listing all the events, listing all the events of the user, adding/deleting/updating an event.
Omer Faruk Dogru	-Contributed to design of pages in the beginning -Created project creation page from scratch -Implemented input form for project creation process -designed page with other components such as profile sider and navigation bar which are implemented by Emilcan -connected form inputs to backend using axios post call to the existing API

	endpoints -inputs changed json to form-data format before post call to handle uploading a file **Final Milestone:** -I've implemented Events page that show all events and connected it to backend. - Some bugs(redirecting to event's website) in the event details page are fixed. -Add to calendar button is implemented in the event details page.(Redirects to google calendar) - Some small design changes in the different parts of the website. -I've written User Manual of frontend with Emilcan Arican.
Muhammed Enes Toptaş	-I contributed to the creation of the architecture of the backend. -I created initial users tables on the database. -I suggested Sequelize ORM to be used on our project. -I implemented JWT Token validation and creation functionalities. -I created and deployed the database on AWS EC2 instance, as Dockerized -I created the AWS EC2 instance. -I implemented login related endpoints. -I deployed the backend to AWS with Hamza. -I implemented profile related endpoints, that are about creating, updating and viewing. **Milestone 2:** - I implemented all new profile related endpoints - I implemented linking with Google Scholar - I built the CICD action in github with Hamza Işıktaş - I implemented all home page related endpoints with Hamza Işıktaş and Hazer Babür - I maintained the server, rebooted it after crashes and restarted the SQL server. - I connected the docker with host system to maintain static files - I implemented some utility functions to serve better recommendations on homepage with Hamza Işıktaş - I created a better email template to send the validation code with helps of Cemre Efe Karakaş.

	Milestone 3: -Attended weekly meetings of the group and backend team. -Hamza and I together migrated our database from mysql to postgresql -Hamza and I deployed the elasticsearch client on AWS. -Hamza and I solved the memory problems caused by elasticsearch by moving into a new EC2 instance with instance type of t2.xlarge -Together with Hamza added the tag filtering mechanism to search. -I have written the System Manual for Backend and Database deployment. -I have written the docker-compose.yml to be used for the final milestone. -I have fixed the bug of search where it was case sensitive -I have implemented the cron job to remove unvalidated users once every month -I have implemented password recovery endpoints and resetting password endpoints -I have added new logic to JWT tokens to provide a secure and safe conditions for password recovery. -I have added a new model to the database, representing event favorites. Added ORM code of the model, and it automatically created the table for it. -I have implemented 7 new endpoints (getEvent, searchEvents, updateEvent, favEvent, unfavEvent, deleteEvent, listFavEvents) for the event model while fixing and generally improving insufficient add & get endpoints for events. -I have created a nice email template to be sent to users in cases of sign up and password recovery -I have reviewed every PR request about the backend branch. -Hamza and I deleted test entries in the database that would look ugly during the presentation. -I have commented on the parts of code I was assigned to. I tried to follow JSDocs convention when writing those comments. -I have changed incompatible data types for the postgresql to compatible ones. -Together with Cihat, fixed a problem with creating a new project after changing the data types, this bug was not very easy to find and we spent around 2 3 hours.

System Manual For Backend and Database

In this file, we'll walk you through on how you can get the backend and backend related services such as Postgresql Database and Elasticsearch running.

Requirements

- **Docker**
- **Docker-Compose**
- **Preferably a Linux distribution as the OS**

Files required

The docker image of the backend application is stored in the file akademise.tar. Also since we will have 3 containters running in parallel, we will also need a docker-compose.yml. This YAML file can be found in the main directory of this repository. Database dump akademise.db is also required, if you want to restore the database.

How to deploy/run

1. **First change directory to the directory where you keep the file akademise.tar**
2. **Run below command to import the image into your docker system**

```
docker image load -i akademise.tar
```

3. **Now head in to the directory where you keep the file docker-compose.yml**
4. **Open up this file and have a look at line 26**
5. **This url should be the url that other people will use to connect to this computer. So change this url accordingly. If you are not planning to expose the backend to outerworld, you might simply write localhost. We use this url to create the links to the static files in the project, such as profile pictures. For example if you provide www.example.com, a profile**

picture will reside in the url www.example.com/static/avatars/filename.png.

6. Under database-service tab, you can see the username, password and the database name the project uses. DO NOT change these variables, these are preconfigured in the akademise image, and changing these settings here will cause the backend app not being able to connect to the database.
7. After saving the file after changes, again head in to the directory of the file docker-compose.yml
8. Now run below command to run the docker-compose

```
docker-compose up -d
```

9. After docker does it's thing, we are done. You can reach the rest API on port 3000, the database on port 5432 and the elasticsearch client on port 9200

Restoring the database

The database is initially empty, to restore the database follow directions below:

1. Run the below command to copy the dump into Postgresql container

```
docker cp akademise.db bounswe2020group6_database-service_1:/
```

2. Then run the below command to enter the Postgresql container

```
docker exec -it bounswe2020group6_database-service_1 bash
```

3. Make sure you are inside of the container
4. Finally run below command inside the container to import the dump

```
psql akademise < akademise.db
```