CONTENTS

Executive Summary

Summary of project status

First, we took Project Plan as our guide, we separated each task between sub-teams. Each sub-team makes their task division on their own. For **the first milestone**, the predefined tasks were sign up, validation via email, login, gathering personal info, create project, basic search functionalities on both mobile(Android) and frontend, also related tables in the database created for these at the backend side. Necessary post and get requests implementation held by backend team for these functionalities. In addition to those, deployment of backend and frontend is done with dockerized fashion.

For **the second milestone**, we added several functionalities to our applications.Creation of a project is improved according to customers' comments at Milestone 1. Users are able to edit and delete their projects. Users are able to invite other users to collaborate their projects, also users can request to collaborate other users' projects, users are able to accept or reject the invitation and collaboration requests. Search is improved, it does not return duplicates anymore and users can search for projects and other users . Research tags and the Affiliation system are improved, mobile application and web application share the same database. Users can add new research areas and affiliations into this database, they are not restricted with the list provided by the applications. Following functionality is implemented and users are able to see who follows them and who they are following through their profile. Upvoting a user functionality became present. Notification for requests and invitations is implemented. Homepage (user feed) is implemented and the projects related to that user will be shown in their homepage. Relation will be decided via previously collaborated projects and followed users and research tags.Google Scholar integration is done to our system to be able to view more projects of the users.

Changes that are planned for moving forward

Search will support semantic search. Implementation of the events page will be done, and upcoming events and deadlines will be shown in this page. Notification for upvotes and upcoming events will be implemented. Homepage (user feed) will be improved and the events related to that user will be shown in their homepage, too. ResearchGate integration will be done to our system to be able to view more projects of the users. Users will be able to delete their accounts if they do not want to benefit from plentiful functionalities of Akademise.

Challenges About Dockerizing, Deployment, Ci/Cd

CI/CD

Milestone 1: First of all, we still don't have a CI/CD pipeline implemented to our github repository yet. We aimed to create a deployment pipeline before **milestone 1** but we had some challenges implementing it.

Firstly, since we were using AWS to deploy our code, we also tried to create a deployment pipeline with the CodePipeline service AWS provides. It took a lot of effort, a lot of rules, connections between services, creation of tags etc. In the end when it was about to be done, service didn't let us create a connection to github. At first we thought this was because of some needed configuration on Github repository and we talked with our teaching asistant to help us with that but still that didn't work. So, we decided to move onto the next CI/CD tool.

Travis was our next option. We connected our github repository with travis, travis.yaml was ready everything was fine. But then travis.yaml was asking for aws secret keys and ids to deploy our code. We had to create a user in AWS for that purpose and use its credentials in the travis.yaml but since we were using AWS Educate, AWS didn't let us create a user since we aren't the root user. We tried our existing student user's credentials but these credentials are not volatile. They were changing periodically with a session and also travis was not able to identify our credentials. Solution was using a non student AWS account but since we had our code deployed and every other configurations set up in the current account, that was the last solution if we didn't have any other.

We actually didn't have enough time to come up with a running CI/CD pipeline under those circumstances before the milestone 1. But we are currently planning to use github actions with either ssh or some http request method to deploy our code when a push or pull request is made on the repository.

Milestone 2:

In the previous milestone we were lastly planning to create a github action to connect to our EC2 instance via the ssh and run some commands to update our code on it. I actually found a way to do that in github actions. I created the github action and necessary commands to be run upon connecting to our instance in order to update our code successfully.

When we are connecting to our EC2 instance via ssh in github actions, we need our instance's host-ip, username and the secret key. We needed to add those as a secret into our github repository in order to reference them in the github action. We were not authorized to add secrets to our repository, so I asked our customer / assistant to add our secrets to the repository. In the end, we had a working pipeline which updates our code when we make a push to our branch. We implemented this pipeline on both frontend and backend.

Dockerizing

Milestone 1:

Docker was a complete new technology for every one of us. But we learned it through the online courses and walkthroughs etc. We found that technology amazing and it was actually pretty easy to set up and use. Without docker it would be actually way harder to

make a running served application in the EC2 instance. Docker helped that nicely where all we had to do was create the image and run it.

Milestone 2:

Since we already dockerized our project, we didn't really make any changes on docker part in the time span of milestone 2.

Deployment

Milestone 1:

Deployment to AWS EC2 is done by CodeDeploy service which is provided by AWS. We needed to create an appspec.yml file in the repository to configure the deployment. Connected the CodeDeploy and EC2 instances together, created an application, deployment group and a deployment for that process. After all those complicated document guided processes, all we had to do was give the github commit where we want to take the code from to the CodeDeploy service and it does the rest.

The only challenge was going through all the documents of AWS and other articles about this process where they explain this process. It actually took a considerable amount of time.

Milestone 2:

Deployment was successfully done in the previous milestone, but in some cases our frontend instance was out of memory and was failing the pipelines because of that. I had to fix those issues where docker was creating <none> id containers for our node_modules folder which includes all the dependency packages of npm, i.e. it is a big folder which takes around 1-2 GB. After deleting those unnecessary overhead containers and rebooting the instance, we were on the path again.

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/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.

/search?query&type
Request type: GET
Required parameters: query type of String, type type of Integer
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.
2-) code: 500, message: “Something went wrong” (String)

Description: /search endpoint is for the authorized users who want to search existing users or projects 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 which means that the user wants to search for users, and if the type is 1 which means that the user wants to search for projects. If users are searched, their first name and last name are concatenated and the execution of search is based on this query: **%query& isLike fullname** If projects are searched, the execution of search is based on this query: **%query& isLike title.** 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.

Project Plan

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

For Milestone 2 we decided to complete tasks up to task 50.

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
Auto Completion	Ömer Faruk Özdemir
create database V2	Ömer Faruk Özdemir

Mobile

inviting a user, 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

	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

Ethan Alpatian opens the mobile app and goes into his profile. He decides to add some interest areas. He then goes on to check if there are any join requests for his latest project. He sees Inés M. Bayern's request to join his project and decides to check her out. He goes into her profile and checks out her projects. He likes the idea used for a project and decides to apply to that project. He thinks Ms. Bayern is a respectable scholar so he decides to follow her. He goes back to his project's collaboration requests page and allows Ms. Bayern's request.

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.

The Code Structure and Group Process

Frontend

Team:

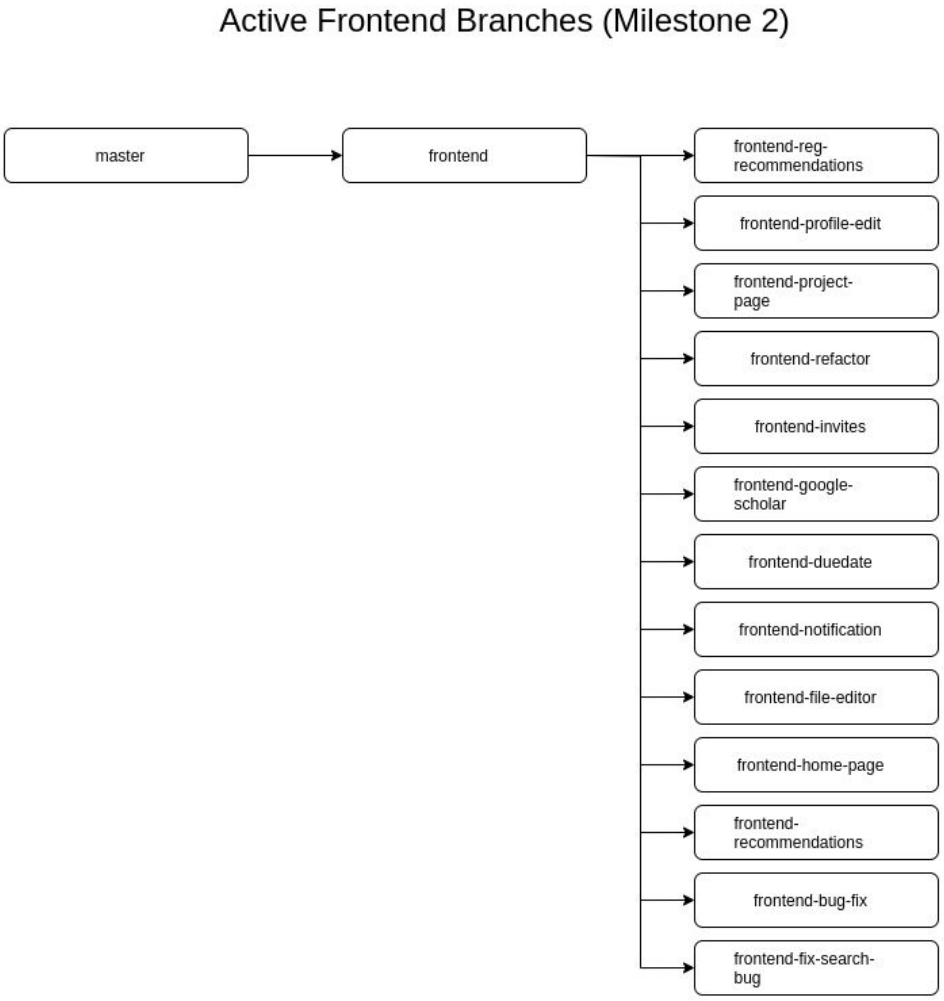
- Cemre Efe Karakaş
- Ali Ramazan Mert
- Emilcan Arıcan
- Ömer Faruk Doğru

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.

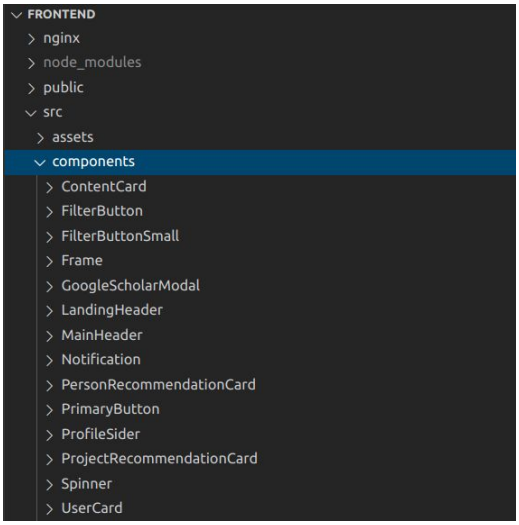
We used the existing branch under the name `frontend`. We made small feature additions and typo corrections on the frontend branch and created separate branches for specific tasks (like separate pages). We usually tried to only open a new branch if there are going to be a series of changes made for a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



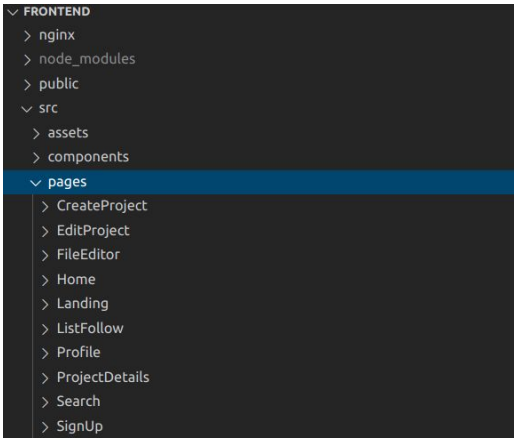
Each week we take the responsibilities given to the frontend team and divide them between the team members. We set a deadline before the team's internal deadline to have time for overcoming possible bugs and difficulties.

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.

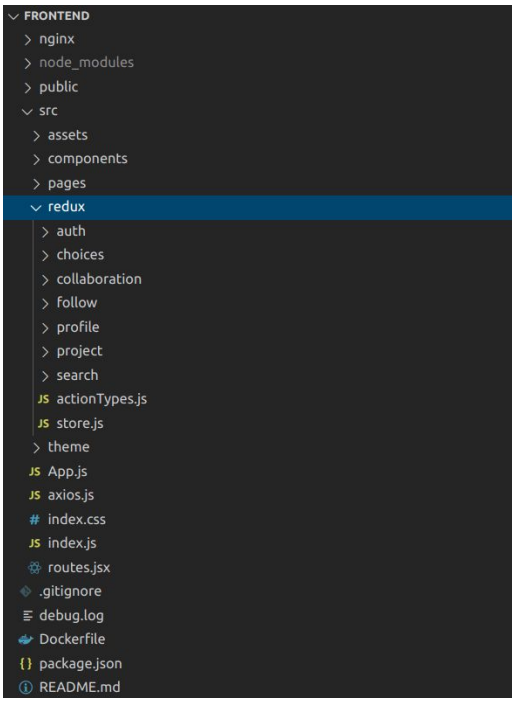
The directory structure is as follows:
(We decided to keep individual screenshots for individual subfolders to better capture everything)
Compontents:



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)

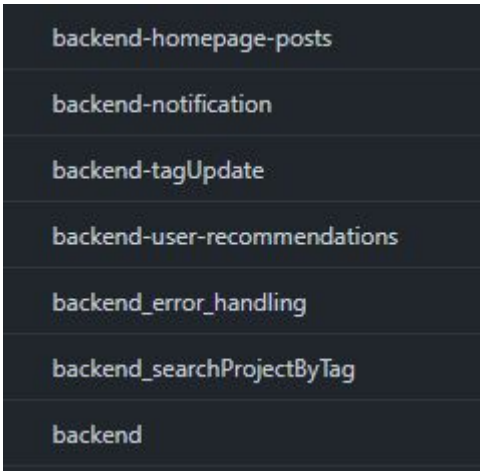
As the backend team, we kept meeting after our lab sessions and decided what to do there. We made use of WhatsApp and Slack frequently to talk about our coding struggles. We also hold some Zoom sessions and coded together in cases we needed more than one people's opinion.

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.

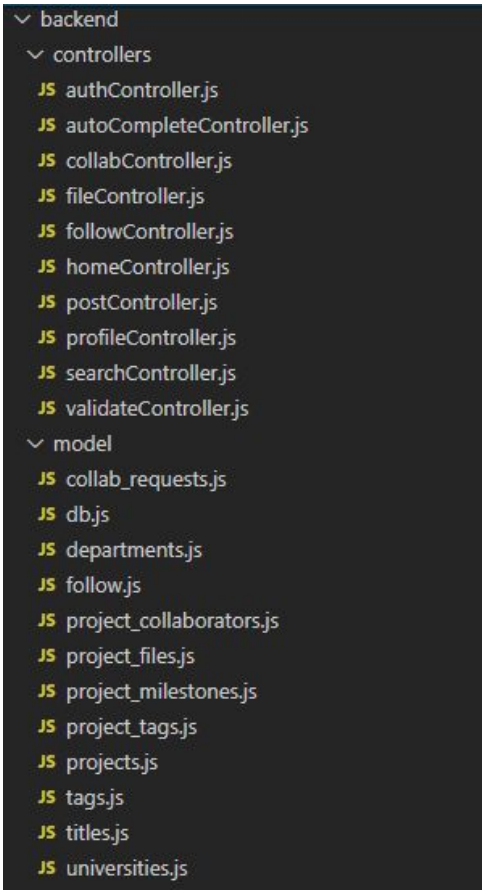
We kept using the branch “backend” as our main branch for backend. We set up CI/CD with Github actions also on this branch, and it redeployed every commit we pushed to this branch.

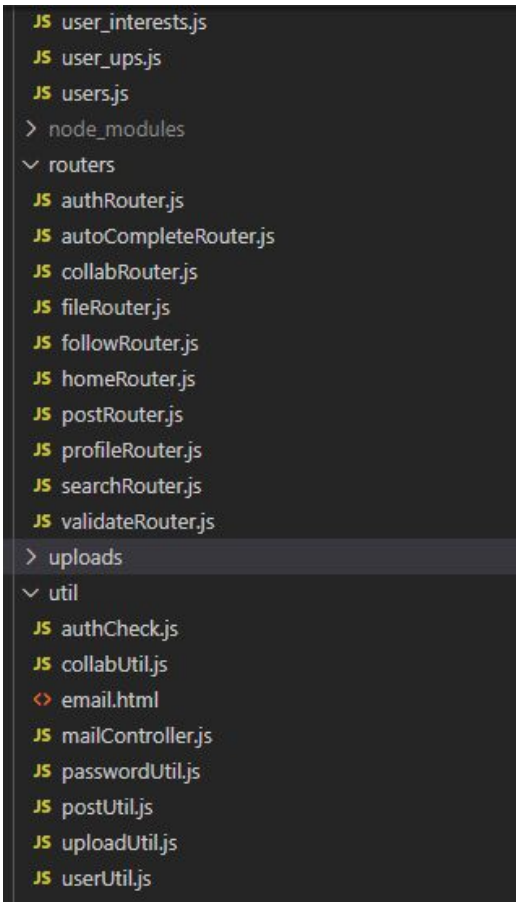
We tried to maximize our usages of issues, as they really helped on the process and prevented the minor things from going forgotten and lost.

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:

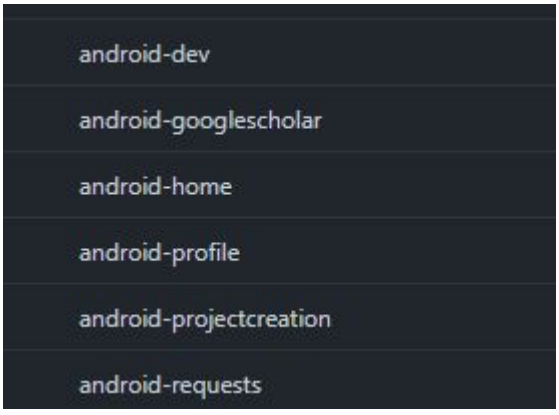
- Ezgi Gülperi Er
- Cihat Kapusuz
- Doğukan Kalkan
- Gülsüm Tuba Çibuk

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.

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:

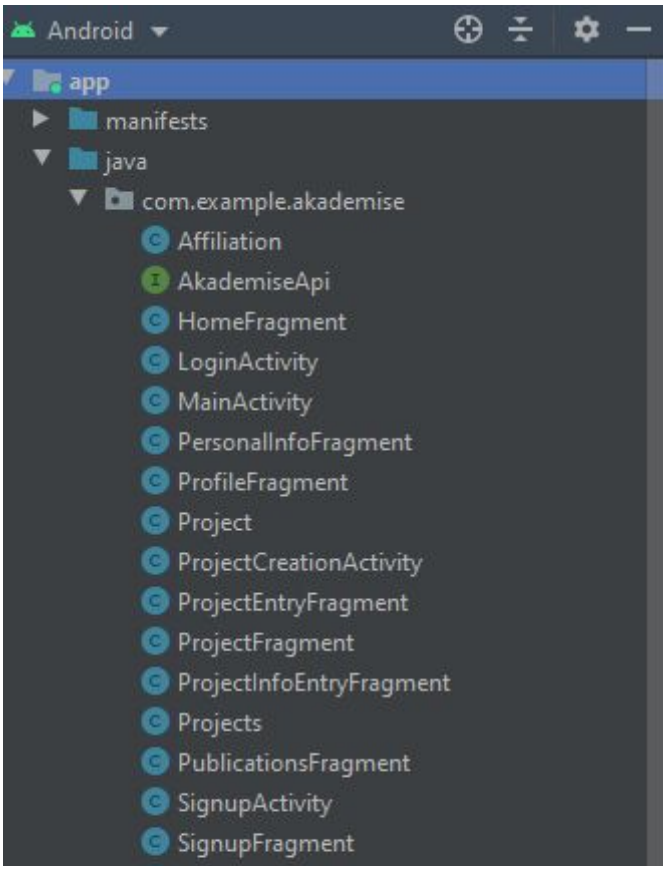


-android-dev is our master branch.
-android-googlescholar : page where we show details about user’s google scholar profile (stats, citations etc)
-android - home : home page, “news feed”
-android - profile : features related to profile page (editing / adding / deleting personal info)
-android -project 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.

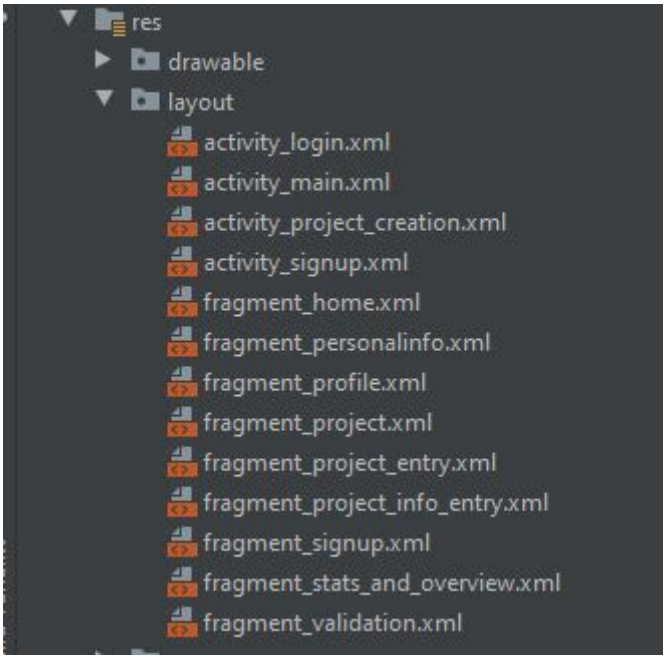
The directory structure is similar to Milestone1 and it is as follows (does not include all class,

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.



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

- We did a pretty good job keeping up with the dates that we have set for the tasks for Milestone-2. And we continued with the practice of setting an internal deadline before the actual milestone (set by the customer). On the day of the internal deadline every team comes together and talks about what is done, what has left to be done and what can be improved. This is useful practice to avoid last minute changes and to have a overall perspective on the product
  One point we can improve is inter-team communication. When a team made a modification in the codebase (changes that can affect other teams), we didn't realize it until an error occurred, this slowed down the development process. I think we should utilize the Slack channel more effectively.
  -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 excellent jobs about their task division and fulfilling their responsibilities. Backend provided necessary endpoints 2 weeks before the milestone. This enabled us to ask for additions and changes to endpoints to fulfill our desires. We completed our tasks in a given time according to the project plan deadlines. 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.
    - 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, Doğukan Kalkan and Cemre Efe Karakaş created a scenario in which all the new functionalities could be presented in an efficient and fun way. And they also did rehearsals before the actual presentation to warm up and get rid of complications that could occur during the presentation.

Our presentation consisted of two parts, namely the Android part and the Frontend part. The Android part was performed by Doğukan Kalkan and the Frontend part was performed by Cemre Efe Karakaş. 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 10 minutes.

There were almost no negative comments on our presentation. However our instructor warned us about one important concept. For the presentation, we decided to create profiles of our instructors at the university without their consent. Although we chose to have profiles like this to make our presentation more realistic and also more fun, we were told that it is not allowed to create profiles for other people with their information unless we display a disclaimer and take their consent. They also told us that one other option is to create profiles of team members since in this way there should be no consent issues regarding profiles.

As a team, we think that there are more functionalities that we can add to our application.

Deliverables

Frontend

- **Signup Page:**
  Complete as planned.
- **Home Page:**
  Requires future work but the planned portion is complete.
- **Landing Page:**
  Complete as planned.
- **Project Creation Page**
  As planned.
- **Search Page:**
  Requires future work but the planned portion is complete.
- **Logged Off Header Component**
  As planned.
- **Logged In Header Component**
  As planned.
- **Profile Sider:**
  As planned.
- **Profile Page:**
  As planned.
- **Project Details Page:**
  As planned.
- **File Edit Page:**
  As planned.
- **Followers and Followings Page:**
  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. Only adding new tags to the database remained.
- **Profile Page**
  Profile page is completed.
- **Project Page**
  Project page completed as planned.
- **Project Details Page**
  Completed as planned.
- **Project Creation Page**
  Requires future work, planned portion is complete.
- **Search:**
  Requires future work, the planned portion is complete.
- **Home Page:**
  Requires future work but the planned portion is complete.
- **Google Scholar Page**
  Completed as planned.
- **Request and Invitations Page**
- **Files Page**
  Requires future work but initial view and basic functionality is done

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

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
        - 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
- **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

- **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.
        - Cihat Kapusuz

    - **a**
    - **a**
- **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, it will have some improvements later on.
        - Cihat Kapusuz
    - **a**
- **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.
    - Users can send invitations to another user to invite her to the project but in Milestone 2 there was no search bar for invitation. We will implement this feature in the final milestone.
      -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

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
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
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
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 problem on the postman. -Tried to create the homepage endpoint, but failed to do so. -Reviewed some of the PRs.
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.
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
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.
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.
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.
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
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.