Boğaziçi University CMPE451 Group 3 Milestone 2 Report

Table of Contents

Executive Summary

Project Description

Paperlayer is an online collaboration platform for academics, in which people can create new projects and write papers, while collaborating with other users and scholars. Moreover, in PaperLayer people can access public projects and keep track of the trends in their academic research area, search for co-authors for their research and follow upcoming events in academia such as conferences or journals. Projects can also be made public/private depending on the user's wishes. Users can invite others to projects in order to collaborate and search for an optimal collaborator for their project. Also users can rate other users, comment on their profiles and send each other messages. To sum up, it's a great platform for collaboration for academics.

Project Status

After Milestone 1, firstly we did some improvements on our project regarding feedbacks. Moreover, we made some changes in our project plan, since talking to our customer on our project led us to understand our customer more deeply and see what features are more valuable for our customer. So, after these improvements and changes in plans, we started to implement new features.

In this second Milestone, we have taken some lessons from our experiences in the first Milestone. Our teams made more inter-team meetings for planning formats of API calls to avoid needs for redesigning the request formats.

In this milestone, we managed to implement many important features of the project, such as collaboration requests, search and follow system. So far, we deployed our second MVP (Minimum Viable Product) for the Milestone 2.

- Our web project dockerized and deployed at https://paperlayer.azurewebsites.net/
- Our backend api documentation can be found at https://paperlayer-backend.azurewebsites.net/swagger/
- We demonstrated a demo of our Android Application at our representation.

Within the context of our second Milestone, we managed to keep our wiki up to date, define a convenient workflow and stick to our project plan.

Moving Forward

In this second Milestone, we were more experienced than the first Milestone. So, we did not face many problems in terms of the project management and we tried to practice the lessons we derived from our past experiences in Milestone 1.

On the other hand, we made a change in the teams, since we observed that our backend team was moving forward fastly and the frontend team was staying behind. In terms of this, one of our backend team members has moved to the frontend team.

With these changes, we believe that we will manage to finish our project as we have planned. In terms of the second milestone, we believe that we are in a good situation and followed our plans.

Deliverables

Deliverable	Update Frequency	Status
Wiki Page	Weekly	Complete
Issues/Pull Requests	Daily	In Progress
Project Plan	As needed	Complete
User scenarios	As needed	Complete
API Documentation	As we implement/update new endpoints	In Progress
Frontend Project	Daily	In Progress
Backend Project	Daily	In Progress
Android Project	Daily	In Progress

1- Wiki Page

We've updated our personal wiki pages, wiki home page and we share our meeting notes occasionally.

2- Issues/Pull Requests

We open issues and pull request for each requested feature implementation, bug fix, documentation. We also attach them with Projects so that we can track our progress on a board.

3- Project Plan

We've planned our tasks according to which milestone will include which requirement. It also includes due date and which person is responsible for that task.

4- User scenarios

We've prepared two user scenarios for the milestone 2 presentation. These scenarios show the current functionality of the application.

5- API Documentation

We're generating our API documentation automatically with swagger.

6- Frontend Project

Fronted features in Milestone 2: **Search**

Comment

Rating

Project Tags

Showing User's milestones

Showing User's Project

Profile Settings for public and private

Project Editing

Changing Profile Picture

Follow and follow requests

Showing Follower, following, follow request in the profile

7- Backend Project

Features that the Backend Team implemented for Milestone 2:

Rating & Comment: Adil Numan Celik - Done

Profile Picture CRUD: Adil Numan Celik - Done

Endpoint Usage Permissions: Adil Numan Celik - Done

Follow & Follow Request: Ahmet Emir - Done

Basic Search: Ali Furkan Budak - Done

Semantic Search Functionality: Ahmet Emir & Ali Furkan Budak - Done

Advanced Search with Filters: Ali Furkan Budak - Done

Email Verification: Ali Furkan Budak - In Progress

Collaboration Request & Invitation: Buse Gildereli - Done

Linking Google Scholar Profile: Buse Gildereli - Done

Notifiction System: Furkan Cansever - Mostly Done

Azure CD: Furkan Cansever - Done

Enhancing Follow & Milestone System: Furkan Cansever - Done

8- Android Project

We have implemented these features in milestone 2:

Home page is updated: recent projects, recent events and milestones.

Project create page is is updated: attaching an event to the project and adding tags.

Project edit page is implemented: editing fields of the projects, updating the event to the project and updating tags. Also we added new project states.

Project detail page is updated: Members list, tags are added. Also we've added authorization support to this page so that members and non members will see this page differently.

Collaboration request feature is implemented: sending colaboration requests to projects, displaying collaboration request of projects which are created by user. Accepting/rejecting collaboration requests.

Invite feature is implemented: project owners can send invite request to other users.

Profile page is updated: followers, followings and follow requests are displayed. Public/private profiles are displayed differently.

Profile edit page is updated: users can change their profile pictures.

Follow feature is implemented: users can follow public users and send follow request to private users. Also they can accept/reject incoming follow requests.

Search page is implemented: users can search projects users and events.

Work Done By Each Member

Name	Task
Mahir Efe KAYA	[Android]Homepage Event Subpage Implementation and UI [Android] Homepage Milestone Subpage Implementation and UI [Android] Homepage Recent Projects Subpage Implementation and UI [Android] Homepage Invite Fragment Implementation [Android]Collaboration Request Implementation [Android]Adding collaboration to Project Details Page [Android]Connecting some parts of the navigation Merging the individual parts of the milestone

Name	Task
Ramazan Koç	[Android] Implementing search user,event,project pages. Adding milestones,requirements,tags,events view in the project page Preparing the main scenario that used in android presentation. Adding view members page in project page. Added icons for different type of projects.
Ahmet Emir Kocağa	- Implemented "Follow" model and created CRUD endpoints - Implemented "Follow Request" model and created CRUD endpoints. - Implemented semantic search with Ali Furkan. - Helped deciding visual design of frontend. - Moved to Frontend team, so learned JavaScript & React. - Changed colours of the frontend. - Implemented Milestones sections project page and changed the design of project page in Frontend. - Prepared and made the frontend presentation. - Reviewed work done by other subteams. - Wrote the Executive Summary & Frontend Scenario of Milestone Report 2.
Yahya Bedirhan Pak	[Android] Profile page: public/private profile page support, follow information is added. Edit profile page: changing profile picture is added. Follow feature: followings, followers and follow request pages, accepting/rejecting follow requests are implemented. UI enhancements: login page layout is updated. UI layout of user lists are uptated in these pages: members, invite, collaboration requests. [Backend] Follow feature: follow-related fields added to user serializers such as is_follower", "is_following". Also a query filter is added to follow_request endpoint.
Furkan Cansever	Implemented Notification model and created CRUD endpoints Adding some properties to Follow and Follow Request Model. Adding an endpoint to Milestone for all milestones of the user. Handled Azure CD process Reviewed work is done by backend team members Tested the endpoints.
Buse Giledereli	Implemented Collaboration request mechanism Implemented Collaboration invite mechanism Implemented Goggle Scholar linking Fixed small bugs Thought about the recommendation system
Barış Başmak	Profile Page adjusted for private fields and private profiles. User's Milestones added to profile page User's Projects added to profile page. Comments and rating functionalities added to profile page. Profile Editing Page implemented. Profile Picture Displaying and Changing. Adding Deleting and Displaying Tags. Finding and implementing a layout format in Front-end. Adding (multiple) Milestones to projects functionalities. Fixing Layout bugs in Homepage & Projects Page. Fixing Bugs in Profile Page: Profile Picture, Project Display
Ali Furkan Budak	- Modified project endpoint ease of usage: endpoint accepts ID for linked objects(event, tags) and returns object data in GET requests (events, tags, members, milestones). - Changed project endpoint so that members members can only be added by collaboration requests. - Implemented basic search which accepts a keyword and case-insensitively checks if any field of project, event or profile objects contains the keyword (considers publicness and privateness of objects and checks only available fields). Returns the matched objects. - Added semantic search functionality to basic search w/Ahmet Emir. - Implemented 7 different search filters mentioned in requirements. (Details) - Implemented email verification. This adds is_active field to users. (Disabled email verification before demo because of Google's security measures.)
Adil Numan Çelik	-[Backend] Created rating system. -[Backend] Created commenting system. -[Backend] Created profile picture endpoints for updating, deleting and viewing them. -[Backend] Created the connection to AWS S3 storage system for storing user uploaded files and profile pictures. -[Backend] Worked on deciding who can access which endpoints and and how much information they can get from for User, Profile, Project, File, Rating, Comment, Profile_Picture endpoints. -[Backend] Overwrote functionalities of all endpoints from types mentioned above to make them accesible by only authorized parties and return different amount of information depending on who the requesting user is and what that user can access. -[Backend] Several bug fixes and small changes according to frontend and android teams’ feedback.
Ahmet Mert Tahran	Routing to event and search pages from an authorized page. Listing followers, followings, and follow requests on profile page. Listing milestones on homepage. Updating Profile bar. Updating Navigation bar. Enabling search feature on all authorized pages. Creating a page which lists all relevant results which comes from search endpoint in three different table by its type such as profile, project and event.
Emirhan Yasin Cetin	Edited the profile page code, so that editing will made user friendly Made the page routing settings Coding the user profile editing page and the functionalities Web security and using chrome for development settings Implementation of sending/receiving collaboration request to a project Implementation of sending/receiving collaboration invite to a project Started coding collaboration invite/request accepting and rejecting by user Made research on UX design and added to wiki page Reviewed the frontend project for bugs and reported

Name	Task
Yusuf Bayam	-Refactored project create request according to backend updates. -Fixed invite request body. -Implemented project edit page. -Refactored project model according to backend changes. -Refactor project by deleting unnecessary classes and renaming ambiguous named files. -Implement tab layout and scroll view in projects page. -Implement event and tag adding for project create page. -Implement event and tag adding for project edit page. -Add new project states for project according to backend. -Refactor project types according to backend. -Added authorization to projects page. -Fix duplicated tags on project detail page.
Yunus Kardaş	[Frontend] Designed project editing page to edit and change information for projects Adding project states in CreateProjectPage and EditProjectPage to control state of projects Added and designed Follow and Unfollow Button and these actions process in public profile's pages for other users Added and designed Sending Follow Request Button and action process in private profile's pages for other users Added and designed Accept and Reject Button and their actions for follow requests in profile page for users

Challenges Met During DevOps

After we started to implement our platform, we decided to use Amazon Services for Continous Deployment and we added all details about CI/CD progress to Milestone I. These services were Elastic Container Service and Elastic Container Registry, but these services were only one month free, therefore we closed these services. Then, we decided to Azure Services for deployment and we didn't have any issues with CI. We had a new account and created two Azure App services for the frontend and backend. We also created Container Registry for keeping our docker image in this registry. Then, We implemented two Github Action files that are triggered when a new version of the project is released. While implementing, we used the preset Github Action to create and push our images to the container registry. We had some problems with the usage of some environment variables and keeping them in a secure way. For this problem, we used the 'application settings' feature of Azure Web apps. Then, we created two pipelines that are triggered when images of the frontend and backend are deployed to the Azure Container Registry.

Tests

Member	Feature	Commit SH
Ahmet Emir Kocaaga	Profile	68948c92d52b9722f5fed4edb1c296741642dc9c
Ahmet Emir Kocaaga	Project	68948c92d52b9722f5fed4edb1c296741642dc9c
Ahmet Emir Kocaaga	Follow	dac992d12a6115b3a27e2a0a1720ff75adc376ff
Ahmet Emir Kocaaga	Follow Request	dac992d12a6115b3a27e2a0a1720ff75adc376ff
Furkan Cansever	Milestone	fa8277f4c662b4fb3ed927767c12cbfebebd5e60
Furkan Cansever	Notification	6531d0806a0885ab1f3502861ac5bd98d133ce1d
Buse Giledereli	Milestone test	8454270f6433180f3a13df6642d935e9150f86a3
Buse Giledereli	Collaboration request	27d467e1f489562426e4ebebc9d13333d60190d7
Buse Giledereli	Collaboration invite	a86b789be2170d8e362dcce176afc4111bb08316
Buse Giledereli	Collaboration permissions	7e3c6b791f67483088816c471499e240754cacc8
Buse Giledereli	Project creation test	8c54bbf2e0ba242a39639303a956e8c86e84df29
Buse Giledereli	Google Scholar Linking	69693fbed4a57207291250c0f4c0d783a96aa50b
Adil Numan Çelik	File Upload and View (Removed Temporarily)	bd242563f0d9644afe9a3d891b466518dcd0a3e4
Ali Furkan Budak	Updating File Tests to work with new features	faaba03e76f349b884156e59a51657e0b425d4ba

API documentation

PaperLayer API documentation v2

Project Plan

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
Registration/Sign in	1.1.1.1.	Registration	03.11.2020	10.11.2020	Barış	Ali / Adil	Mahir	

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
	1.1.1.4.	Profile pages			Mert	Ahmet Emir	Yahya	
	1.1.2.1.	Registration info			Barış	Ali / Adil	Mahir	
	1.1.2.3.	Sign in info			Barış	Ali / Adil	Ramazan	
	1.1.2.5	Password reset			-	Ali / Adil	Mahir	
Profile Page	1.1.5.1.1	Profile info	03.11.2020	10.11.2020	Barış	Ahmet Emir	Yahya	
	1.1.5.1.2				Emirhan	Ahmet Emir	Yahya	
	1.1.5.1.3				Barış	Ahmet Emir	Yahya	
	1.1.5.1.4				Barış	Ahmet Emir	Yahya	
	1.1.5.6	Hidden info			Barış	Ahmet Emir	Yahya	
Project Creation	1.1.3.1.	Posting a project	10.11.2020	17.11.2020	Barış	Buse / Ahmet Emir	Yusuf	
	1.1.3.2.	Seeking for collaborators state			Barış	Buse / Ahmet Emir	Yusuf	
	1.1.3.3.	Post project requirements			Barış	Buse / Ahmet Emir	Yusuf	
	1.1.3.4.	Set project public/private			Barış	Buse / Ahmet Emir	Yusuf	
	1.1.4.1.	Posting files			Mert	Adil	Yusuf	
	1.1.4.3.	Add milestone			Barış	Buse / Ahmet Emir	Yusuf	
	1.1.4.4.	Change state of post			Mert	Buse / Ahmet Emir	Yusuf	
	1.1.4.5.	Link event to project			Mert	Ali	Mahir	
Project Page	1.1.8.1	Setting project details	10.11.2020	17.11.2020	Mert	Buse / Ahmet Emir	Mahir	
Event Creation	1.1.4.2.	Adding an event	17.11.2020	24.11.2020	Barış	Ali	Mahir	
Milestone 1	-	Prepare for milestone 1	17.11.2020	24.11.2020	All team	All team	All team	
Report	-	Prepare milestone report	24.11.2020	29.11.2020	All team	All team	All team	
Project Page	1.1.8.1	Tag improvements	01.12.2020	08.12.2020	Barış	Ali	Yusuf	Earlier 1.1.8.1
Event Creation	1.1.4.2.	Event improvements	01.12.2020	08.12.2020	Yunus	Ali	Mahir	Earlier 1.1.4.2.
Editing Project	1.1.4.3.	Milestone improvements	01.12.2020	08.12.2020	Yunus	Ali	Ramazan	Earlier 1.1.4.3.
	1.1.4.4.	Change state of post			Yunus	Ahmet Emir	Yusuf	Earlier 1.1.4.4.
	1.1.4.5.	Link event to project			Yunus	Ali	Yusuf	Earlier 1.1.4.5.
Profile System	1.1.5.4	Set profile public/private	01.12.2020	08.12.2020	Barış	Ahmet Emir	Yahya	
	1.1.5.6	Hide hidden fields from profile			Barış	Ahmet Emir	Yahya	Earlier 1.1.5.6.
Project Structure	1.2.5.1	Add new stages	01.12.2020	08.12.2020	Mert	Buse	Yusuf	1.1.4.4.
	1.2.5.2	Add new stages			Mert	Buse	Yusuf	1.1.4.4.
	1.2.5.4	Add file to project			-	Adil	-	1.1.4.1
Project Page	1.1.8.2	View the project's public information	01.12.2020	08.12.2020	Barış	Adil	Ramazan	Earlier 1.1.8.1

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
Profile Page (Functional)	1.2.4.1	Private profile pages displayed to followers	01.12.2020	08.12.2020	-	Adil	-	1.1.5
Project Gathering (BACKEND)	1.1.3.5	Sending collaboration request	01.12.2020	08.12.2020	-	Buse / Furkan	-	1.1.3 - 1.1.5 - 1.1.6
	1.1.3.7	Members suggesting new users to project	01.12.2020	08.12.2020	-	Buse / Furkan	-	1.1.3 - 1.1.5 - 1.1.6
Project Page (BACKEND)	1.2.6.2	System shall ensure that project's private information isn't shown to those who are not collaborators.	01.12.2020	15.12.2020	-	Adil	-	
	1.2.6.3	System shall ensure that guest users can not collaborate or request collaboration.			-	Adil	-	
	1.2.6.4	System shall provide necessary mechanism for users to link their profiles with the project.			-	Adil	-	
Project Structure (BACKEND)	1.2.5.5	System shall make private projects to be visible by only the collaborators.			-	Adil	-	
Follow (BACKEND)	1.1.7.1	Follow other users with public profile pages	01.12.2020	15.12.2020	-	Ahmet Emir	-	1.1.5
	1.1.7.2	Send follow requests for private profile pages			-	Ahmet Emir	-	1.1.5
	1.1.7.3	Accept/decline following requests			-	Ahmet Emir	-	1.1.5
	1.1.7.5	Unfollow			-	Ahmet Emir	-	1.1.5
Project Gathering (BACKEND)	1.1.3.6	Owners sending invitation to users	08.12.2020	15.12.2020	Mert	Buse	Mahir	1.1.3 - 1.1.5 - 1.1.6
Search (BACKEND)	1.1.6.1	Search events, projects and other users	08.12.2020	15.12.2020	-	Ali & Ahmet Emir	-	
	1.1.6.2	Customize the search with filters			-	Ali & Ahmet Emir	-	
	1.1.6.3.1	Sort user-related search results with these criteria: alphabet order, connection with common teammates, rating			-	Ali & Ahmet Emir	-	1.1.5
	1.1.6.3.2	Sort project-related search results with these criteria: alphabet order, relation with users' interest			-	Ali & Ahmet Emir	-	1.1.3

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
	1.1.6.3.3	Sort event-related search results with these criteria: alphabet order, date, submission deadline			-	Ali & Ahmet Emir	-	1.1.4.2.
Profile pictures (BACKEND)		Endpoint for adding profile pictures	08.12.2020	15.12.2020	-	Adil	-	
Notifications (BACKEND)	1.2.3.1.1	Notify users in case of follows, follow requests and ratings on their profiles	08.12.2020	22.12.2020	-	Furkan	-	
	1.2.3.1.2	Notify users in case of project collaboration requests and updates on milestones, files and polls about collaborated projects.			-	Furkan	-	
	1.2.3.1.3	Notify users in case of stage changes about followed and collaborated projects.			-	Furkan	-	
Project Gathering (Frontend-Android)	1.1.3.5	Sending collaboration request	08.12.2020	22.12.2020	Mert	-	Mahir	1.1.3 - 1.1.5 - 1.1.6
	1.1.3.7	Members suggesting new users to project	08.12.2020	22.12.2020	Mert	-	Mahir	1.1.3 - 1.1.5 - 1.1.6
Follow (Frontend-Android)	1.1.7.1	Follow other users with public profile pages	08.12.2020	29.12.2020	Yunus	-	Yahya	1.1.5
	1.1.7.2	Send follow requests for private profile pages			Yunus	-	Yahya	1.1.5
	1.1.7.3	Accept/decline following requests			Yunus	-	Yahya	1.1.5
	1.1.7.5	Unfollow			Yunus	-	Yahya	1.1.5
Profile System (BACKEND)	1.1.5.2 - 1.2.4.2	Linking Google Scholar or ResearchGate accounts.	15.12.2020	22.12.2020	-	Buse	-	
	1.1.5.3	Add ratings and comments to teammates			-	Adil	-	1.1.3
Search Engine (BACKEND)	1.2.1.1.1	Basic search shall support searching with name and tag.	15.12.2020	22.12.2020	-	Ali & Ahmet Emir	-	
	1.2.1.1.2	Advanced search shall support searching with the institution, rating, and skills for the user; project stage, due date and linked event for projects; date, submission deadline, location and type for events.			-	Ali & Ahmet Emir	-	

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
	1.2.1.2.1	Search results shall include the public and followed profiles.			-	Ali & Ahmet Emir	-	
	1.2.1.2.2	Search results shall include public projects.			-	Ali & Ahmet Emir	-	
	1.2.1.3	Search engine shall support semantic search. Semantically related content about the search keywords shall be included in search results.			-	Ali & Ahmet Emir	-	
Search (Frontend-Android)	1.1.6.1	Search events, projects and other users	15.12.2020	29.12.2020	Mert	-	Ramazan	
	1.1.6.2	Customize the search with filters			Mert	-	Ramazan	
	1.1.6.3.1	Sort user-related search results with these criteria: alphabet order, connection with common teammates, rating			Barış	-	Ramazan	1.1.5
	1.1.6.3.2	Sort project-related search results with these criteria: alphabet order, relation with users' interest			Mert	-	Ramazan	1.1.3
	1.1.6.3.3	Sort event-related search results with these criteria: alphabet order, date, submission deadline			Mert	-	Ramazan	1.1.4.2.
Project Gathering (Frontend-Android)	1.1.3.6	Owners sending invitation to users	15.12.2020	29.12.2020	Mert	-	Mahir	1.1.3 - 1.1.5 - 1.1.6
Follow (BACKEND)	1.1.7	Changes in follow	22.12.2020	29.12.2020	-	Yahya	-	
Profile System (Frontend-Android)	1.1.5.3	Add ratings and comments to teammates	22.12.2020	29.12.2020		-	Yahya	1.1.3
Registration/Sign in (BACKEND)	1.1.2.6	Verification email	22.12.2020	29.12.2020	-	Ali	-	1.1.1
Milestone 2	-	Prepare for milestone 2	22.12.2020	29.12.2020	All	All	All	
Report	-	Prepare milestone report	29.12.2020	05.01.2021	All	All	All	

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
Profile System (Frontend-Android)	1.1.5.2 - 1.2.4.2	Linking Google Scholar or ResearchGate accounts.	05.01.2021	12.01.2021		-	Yahya	
Notifications (Frontend-Android)	1.2.3.1.1	Notify users in case of follows, follow requests and ratings on their profiles	08.12.2020	22.12.2020	Ahmet Emir	-	Ramazan	
	1.2.3.1.2	Notify users in case of project collaboration requests and updates on milestones, files and polls about collaborated projects.			Ahmet Emir	-	Mahir	
	1.2.3.1.3	Notify users in case of stage changes about followed and collaborated projects.			Ahmet Emir	-	Mahir	
Profile System	1.1.5.5	Report other user profiles for these reasons: Disturbing other users, Sharing unrelated or disturbing posts, Spam, Fake Profile, Stolen Account .	05.01.2021	12.01.2021	Barış	Furkan	Yahya	
Recommendation	1.2.2.1.1	Recommendation system shall be based on users' previous projects, interest areas, ratings, and skills.	05.01.2021	12.01.2021	Yunus	Ali / Buse	Yusuf	1.1.3
	1.2.2.1.2	System shall recommend possible collaborators to project creators.			Yunus	Ali / Buse	Yusuf	1.1.3
	1.2.2.1.3	System shall provide possible project recommendations to users.			Yunus	Ali / Buse	Yusuf	1.1.3
Project Structure	1.2.5.3	Text editor to edit project files	05.01.2021	12.01.2021	Barış / Mert	Adil	-	1.1.4.1
Follow	1.1.7.4	Activity page about followed users and collaborators	05.01.2021	12.01.2021	Barış	Adil	Mahir	1.1.7
Registration/Sign in	1.1.2.4	Terms of Service and Privacy Policy	05.01.2021	12.01.2021	Emirhan	Buse	Yusuf	
Registration and Sign In	1.1.2.2	Users should be able to sign up with their Google accounts.	12.01.2021	19.01.2021	Yunus	Adil	Yahya	
Guest User	1.1.1.2	Guests search	15.12.2020	22.12.2020	Barış / Yunus	Ali	-	
	1.1.1.3	Guests home page			Barış / Yunus	Ali	-	
Annotation	-		12.01.2021	19.01.2021	All	All	All	
Project Structure (Frontend-Android)	1.2.5.4	Add file to project	12.01.2021	19.01.2021	Emir	-	Ramazan	

Task	Requirement	Details	Start Date	Deadline	Assignee(Frontend)	Assignee(Backend)	Assignee(Android)	Predecessor
Milestone 3	-	Prepare for milestone 3	12.01.2021	19.01.2021	All	All	All	

User Scenarios

Web Scenario

Background

Özlem Türeci is a scientist. At the early stages of the pandemic, she developed a vaccine for covid19. Now, she wants to work with a group of skilled and hardworking scientists to examine if the new mutation seen in Great Britian will effect her vaccine research. For this purpose she can use PaperLayer again. Because on PaperLayer you can find other scientists to collaborate on your project or research. This way, she and other scientists can share their knowledge and work together for the same purpose. She searched for a platform to find other colleagues and found Paperlayer.

Preconditions

1. She is a registered user

Steps Shown in Demo

1. She logins to PaperLayer
2. She views her profile page, changes her profile picture
3. She creates a new project for Covid-19 mutation
4. She add a new milestone to her project
5. She add tags to the project
6. She goes back to home page and see her incoming milestone and tags of the project on her homepage
7. She searches for her husband on PaperLayer
8. She finds Ugur Sahin's profile
9. She gives rating to Ugur Sahin

Android Scenario

Background

Hakan Yılmaz is a professor that works in NeurotechEU. Hakan wants to carry his research projects to an online platform due to the fact that coronavirus causes people to stay at home. After some research, he finds the perfect platform to collaborate and research with other people and chooses to use Paperlayer. He talks with his teammates and invites them to Paperlayer. He and his team were already working on several projects.

Preconditions

1. He is a registered user
2. He already downloaded the PaperLayer app to his phone
3. He already has several projects in Paperlayer.

Steps Shown in Demo

1. He logins to PaperLayer app.
2. He navigates to his profile page.
3. He checks his follower requests.
4. He decides to change his profile picture.
5. He looks profiles of users who has sent follow request.
6. He sends follow request to users that he knows or wants to contact with.
7. He wants to search about his study area. He works in the neuroscience area, so he clicks search tab and search projects with keyword neuroscience.
8. He views couple of projects and this give him more insight of what is going on in neuroscience.
9. He chooses one of the projects and views details of it.
10. He searches for neuroscience events and views the details of one event.
11. He visits his projects page and wants to add new tags and change the event of his project.
12. He wants to invite new members to his project.So,he switches to members tab and invites new members.

Evaluation of tools

Backend Tools

Django: Django makes the most part of the backend development easier as it has numerous bulit-in functionalities, but this has side effects like steep learning curve so we are still trying to fully control it.

Poetry: It creates a virtual environment to include only the same versions of python modules accross the backend team. This way, we have less erorrs and backend development is streamlined.

Docker: It keeps all required dependecies for the project such as python, python modules, binary files(Postgresql, OpenSSL etc.) in one place. So each team member can have the same development environment in their machines. Without Docker, installing dependencies one by one and making sure that everyone has the same version would be hard to accomplish.

Swagger: Swagger is a great tool for us to create endpoint documentation automatically. The endpoints we create in Django end-up in Swagger documentation with its input and output.

Django REST Framework: Django REST framework is a powerful and flexible toolkit for building Web APIs. We utilized it for every endpoint and it made the development process easier.

VS Code: VS code is a great user friendly code editor. It allow us to work more efficiently.

Github Desktop: It makes everything about Github easier and understandable with its easy-to-use UI.

Discord: We heavily utilize Discord. It is convenient to use because we can have one app for one-to-one and group chats, meetings and screen sharing.

Whatsapp: Whatsapp is used for instant communication in our group.

Frontend Tools

Github: We used GitHub actions, issues, and projects to plan, automate test process and check the health of the development process. To use GitHub effectively, we found a branch creation system that for each issue, we create a branch that its name is the one of the directly related issues.

Docker: We used Docker to easily deploy the application into the server and run faster on our own computers due to the fact that it automates installations and environment configurations

Nginx: We used Nginx to dynamic HTTP content handling

ESLint: We used ESLint for static code analysis on health test and code quality check

Discord: We heavily utilize Discord. It is convenient to use because we can have one app for one-to-one and group chats, meetings and screen sharing to plan, report, and discuss the state of development.

Whatsapp: We used Whatsapp for fast or direct communication to plan, report, and discuss the state of development.

VS Code: It is a powerful and user friendly IDE and we generally used this because it has a lot of extensions to design and develop a webpage with React, and its git integration, keymap support and simultaneous coding support is very nice. Thus, it allows us to work more efficiently.

React: We used React as the web framework to develop each web page of the platform. It makes the frontend development easier because it has a great and nice community, and this property led us to find a solution for the bugs etc. easily. Since React is a simple framework that it uses the DOM like HTML, it leads learning period to become shorter. As two sides of a coin, React has some problems: there is no direct supports for some features such as state management and routing. Therefore, we need to use additional libraries

Material.ui: We used material.ui library in order to make development and design processes faster and easier. For example, we use Snackbar, Toolbar, MuiAlert, Grid and many other components with some minor changes in the pages and they were very useful overall.

WebStorm: It is a powerful and user friendly IDE and some members used this because it has a lot of code quality tools support in the editor, fast auto complete/correction and its git integration, keymap support and simultaneous coding support is very nice. Thus, it allows us to work more efficiently.

Axios: we used axios to handle promise-based HTTP request

Date-io: we used date-io for date management

Android Tools

Android Studio: Android Studio is the most powerful and most used IDE for Android development. It has Git, terminal integtrations. Also it has Gradle support.

Kotlin : We decided to use Kotlin for Android project of PaperLayer. There are several reasons behind these decision. One of them is Google announced Kotlin as official language for Android development in 2017. Some of the other reasons are there is lambda expressions, null safety and extension functions. These features are not supported on Java.

Gradle : Gradle is a build tool which comes with Android Studio IDE. Gradle handles the dependencies, minimum API version, target API version, third party libraries of an Android project. It makes it very easy to use libraries in a project.

MVP Pattern: Before we started to implement Android project we discussed about whether we should use some software architecture or not. Then, we decided on using MVP architecture. MVP is Model-View-Presenter and what it does is basically seperate the functionality in three parts. Model is for managing API requests, View is for UI related operations and Presenter is for logic operations. Following this architecture we created a project which is easier to implement, debug and test. Also, by following MVP pattern we tried to accomplish S.O.L.I.D principles as much as possible.

Retrofit : We used Retrofit for network operations. Retrofit is a network provider library used in almost every Android project. We created Retrofit instance in our project and we called that instance whenever we need to send a request and get a response. Currently we didn't add AuthenticationInterceptor in our Retrofit instance but we will add it for easier implementation later.

RxJava: RxJava is a library that enables us to use reactive programming. We used RxJava to handle request and response with combination of Retrofit instance. Basically when we create and send a request our RxJava instance waits for the response. Whenever the response comes(either success or error) our RxJava is notified and after that we can handle the response.

Moshi : Moshi is a library for converting request or response to Kotlin objects. Using Moshi we created our requests as Kotlin objects and when sending request it automatically converts it to JSON object. Also our response is in JSON format and by using Moshi it is converted to Kotlin object. Moshi is great for handling request and response in object oriented programming.

Dagger : Dagger is a Dependency Injection library for Android. We used Dagger for this purpose.

Material Components : We used Material Components library for customizing UI elements.

Git : We used Git as version control system. It allowed us to work as a team.

Management Evaluation

As a yet non-professional team, we have had some challenges throughout the period between the former and the current milestone. Our workload had to increase by quite a bit, so did our challenges. In the beginning we formed the teams in a skewed format. With more people in the back-end and less in front-end. But after a while the web (front-end) team started to suffer from lack of workforce and fell behind the other teams. To solve this issue, we decided to shift a member, Yunus Kardaş, from the back-end team to the front-end team after our first milestone. But this proved insufficient and we decided to put one more person to the front-end team and Ahmet Emir Kocaağa volunteered. He started working on the front-end team 2 weeks prior to this milestone. Also, after the milestone presentation we decided to shift one more person to the front-end team, also from backend. This was more because back-end tasks were becoming scarce while they are ahead of the other teams in their tasks and they are also doing more than decently when we look at the requirements they've fulfilled. Since people can't adapt to a different project right after changing teams, we are giving them a week or two to familiarize themselves with the Front-End project and ReactJS while assigning them smaller tasks. We hope to have come to an end with shifting members from one team to another. Since this results in confusion as well. But apart from this workload imbalance, we can say that we haven't met more difficulties after the first milestone.

Assessment of the customer presentation

For our second customer meeting, we have prepared two scenarios for showing the functionalities of web pages and android. In our scenarios, the main was on creating, changing, editing projects and adding tags, searching users, following users, collaboration to projects and inviting accepting users for them. By presenting those functionalities to our customers, we have collected so many useful insights about our project and it helped us see the way we are going in this project much more clearly. The reaction from our customers was generally positive as they were following the presentation, and at the end our customers came up with good questions and comments.

We have been suggested that there are no milestones in the project detail and it would be better if we add it to the project details. Second, there was a problem with the tags, which are being shown multiple times from the same tag at once, which was confusing for user, and now we have fixed it. We were showing the whole description of a milestone when we are setting it, bu the comment was that it would be more useful for a user to just showing its name, because the whole description is too long to read at the first glance. Lastly, we have received a comment on the follow requests: send a notification to a user only when the following request is accepted, not when rejected.

Among these constructive comments to make our project better, we have received some complements on the differentiation of tags by different colors and the functioning of our project. What helped us in this process was that we practiced the presentation earlier internally and made sure everything was working as planned.

What we have learned the most from this meeting was that from now on we should be more user-centric. It does not matter how good functioning the product is, if the user can't get it right and use it right, it becomes useless.

The code structure and group process

We kept our working style the same as we did before milestone 1. The biggest difference we have is we had some of our friends change their team and some of our team mates work in multiple teams. To summarize the way we worked, we have three different teams namely Backend, Android and Frontend all of which are working separately having their preferences on how to implement their part of the project. Group process might differ from team to team. Developers have their own branches for implementing, improving or debugging a feature. We do not have strict rules on how to use branches if no one violates another developer's branch or the main branch. Every branch is created with a prefix which tags the team it belongs to. After completing their coding developers need to create a pull request. Pull requests can have tags representing how critical they are, what they change, which team they belong to, their status and their type all of which helps reviewers. Every pull request is tested by various continuous integration tools such as Github Actions and Docker. If every test is passed, the code should be reviewed by other developers before being merged to the main branch. Number of reviewers depends on the size and priority of the pull request. After passing, tests and getting confirmed by reviewers, the newly requested code can be merged into the main branch. Backend team follows pep-8 coding standards for a more structured, readable and understandable code. They use a tool called flake8 to test and ensure our code is up to those standards. Poetry and pytest are the other tools they use. Frontend team uses Eslint for a cleaner code. They make sure every pull request has explanatory comments so that reviewers have an easier time to review. They used material-ui themes and components for minimizing disorders in CSS code. Android team uses MVP (model-view-presenter) pattern to organize the presentation layer in the application. They require at least a reviewer for a pull request like the other teams.

Appendix

PaperLayer API

Overview

API documentation of PaperLayer

Version information

Version : v2

URI scheme

Host : paperlayer-backend.azurewebsites.net
BasePath : /api
Schemes : HTTPS

Consumes

- `application/json`

Produces

- `application/json`

Security

Token

Please write "Token" before the key

Type : apiKey
Name : Authorization
In : HEADER

Paths

POST /auth/

Parameters

Type	Name	Schema
Body	**data** *required*	AuthToken

Responses

HTTP Code	Schema
201	AuthToken

Tags

- auth

POST /collaboration_invites/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	CollaborationInvitePOST

Responses

HTTP Code	Schema
201	CollaborationInvitePOST

Tags

- collaboration_invites

GET /collaboration_invites/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**from_user*id*** optional__	number
Query	**to_project*id*** optional__	number
Query	**to_user*id*** optional__	number

Responses

HTTP Code	Schema
200	< CollaborationInvite > array

Tags

- collaboration_invites

GET /collaboration_invites/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer

Responses

HTTP Code	Schema
200	CollaborationInvite

Tags

- collaboration_invites

PUT /collaboration_invites/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer
Body	**data** *required*		CollaborationInvite

Responses

HTTP Code	Schema
200	CollaborationInvite

Tags

- collaboration_invites

DELETE /collaboration_invites/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- collaboration_invites

PATCH /collaboration_invites/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer
Body	**data** *required*		CollaborationInvite

Responses

HTTP Code	Schema
200	CollaborationInvite

Tags

- collaboration_invites

POST /collaboration_invites/{id}/accept_collaboration_invite/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer

Type	Name	Description	Schema
Body	**data** *required*		CollaborationInvitePOST

Responses

HTTP Code	Schema
201	CollaborationInvitePOST

Tags

- collaboration_invites

POST /collaboration_invites/{id}/reject_collaboration/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration invite.	integer
Body	**data** *required*		CollaborationInvitePOST

Responses

HTTP Code	Schema
201	CollaborationInvitePOST

Tags

- collaboration_invites

POST /collaboration_requests/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	CollaborationRequest

Responses

HTTP Code	Schema
201	CollaborationRequest

Tags

- collaboration_requests

GET /collaboration_requests/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**from_user*id*** optional__	number
Query	**to_project*id*** optional__	number
Query	**to_user*id*** optional__	number

Responses

HTTP Code	Schema
200	< CollaborationRequest > array

Tags

- collaboration_requests

GET /collaboration_requests/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer

Responses

HTTP Code	Schema
200	CollaborationRequest

Tags

- collaboration_requests

PUT /collaboration_requests/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer
Body	**data** *required*		CollaborationRequest

Responses

HTTP Code	Schema
200	CollaborationRequest

Tags

- collaboration_requests

DELETE /collaboration_requests/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- collaboration_requests

PATCH /collaboration_requests/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer
Body	**data** *required*		CollaborationRequest

Responses

HTTP Code	Schema
200	CollaborationRequest

Tags

- collaboration_requests

POST /collaboration_requests/{id}/accept_collaboration/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer

Responses

HTTP Code	Schema
201	No Content

Tags

- collaboration_requests

POST /collaboration_requests/{id}/reject_collaboration/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this collaboration request.	integer

Responses

HTTP Code	Schema
201	No Content

Tags

- collaboration_requests

POST /comments/

Description

Users can only comment on other user that they colllaborate on a project.

Parameters

Type	Name	Schema
Body	**data** *required*	Comment

Responses

HTTP Code	Schema
201	Comment

Tags

- comments

GET /comments/

Description

Users can list comments that belong to users with public profiles, themselves, someone they follow or comments they created.

Parameters

Type	Name	Schema
Query	**from_user** *optional*	string
Query	**to_user** *optional*	string

Responses

HTTP Code	Schema
200	< Comment > array

Tags

- comments

GET /comments/{id}/

Description

Users can see comments of a user that has a public profile or that they follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this comment.	integer

Responses

HTTP Code	Schema
200	Comment

Tags

- comments

PUT /comments/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this comment.	integer
Body	**data** *required*		CommentUpdate

Responses

HTTP Code	Schema
200	CommentUpdate

Tags

- comments

DELETE /comments/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this comment.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- comments

PATCH /comments/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this comment.	integer
Body	**data** *required*		CommentUpdate

Responses

HTTP Code	Schema
200	CommentUpdate

Tags

- comments

POST /events/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	Event

Responses

HTTP Code	Schema
201	Event

Tags

- events

GET /events/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**event_type** *optional*	string

Responses

HTTP Code	Schema
200	< Event > array

Tags

- events

GET /events/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this event.	integer

Responses

HTTP Code	Schema
200	Event

Tags

- events

PUT /events/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this event.	integer
Body	**data** *required*		Event

Responses

HTTP Code	Schema
200	Event

Tags

- events

DELETE /events/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this event.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- events

PATCH /events/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this event.	integer
Body	**data** *required*		Event

Responses

HTTP Code	Schema
200	Event

Tags

- events

POST /example/

Parameters

Type	Name	Schema
Body	**data** *required*	ExampleModel

Responses

HTTP Code	Schema
201	ExampleModel

Tags

- example

GET /example/

Responses

HTTP Code	Schema
200	< ExampleModel > array

Tags

- example

GET /example/{id}

Parameters

Type	Name	Schema
Path	**id** *required*	string

Responses

HTTP Code	Schema
200	ExampleModel

Tags

- example

PUT /example/{id}

Parameters

Type	Name	Schema
Path	**id** *required*	string
Body	**data** *required*	ExampleModel

Responses

HTTP Code	Schema
200	ExampleModel

Tags

- example

DELETE /example/{id}

Parameters

Type	Name	Schema
Path	**id** *required*	string

Responses

HTTP Code	Schema
204	No Content

Tags

- example

POST /files/

Parameters

Type	Name	Schema
FormData	**file** *required*	file
FormData	**project** *required*	integer
FormData	**remark** *required*	string

Responses

HTTP Code	Schema
201	File

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

GET /files/

Parameters

Type	Name	Schema
Query	**project** *optional*	string

Responses

HTTP Code	Schema
200	< File > array

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

GET /files/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this file.	integer

Responses

HTTP Code	Schema
200	File

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

PUT /files/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this file.	integer
FormData	**file** *required*		file
FormData	**project** *required*		integer
FormData	**remark** *required*		string

Responses

HTTP Code	Schema
200	File

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

DELETE /files/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this file.	integer

Responses

HTTP Code	Schema
204	No Content

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

PATCH /files/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this file.	integer
FormData	**file** *required*		file
FormData	**project** *required*		integer
FormData	**remark** *required*		string

Responses

HTTP Code	Schema
200	File

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

GET /files/{id}/retrieve_file/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this file.	integer

Responses

HTTP Code	Schema
200	File

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- files

POST /follow/

Description

ViewSet for follow.

Parameters

Type	Name	Schema
Body	**data** *required*	FollowPost

Responses

HTTP Code	Schema
201	FollowPost

Tags

- follow

GET /follow/

Description

ViewSet for follow.

Parameters

Type	Name	Schema
Query	**from_user*id*** optional__	number

Type	Name	Schema
Query	**to_user*id*** optional__	number

Responses

HTTP Code	Schema
200	< Follow > array

Tags

- follow

GET /follow/{id}/

Description

ViewSet for follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this following.	integer

Responses

HTTP Code	Schema
200	Follow

Tags

- follow

PUT /follow/{id}/

Description

ViewSet for follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this following.	integer
Body	**data** *required*		Follow

Responses

HTTP Code	Schema
200	Follow

Tags

- follow

DELETE /follow/{id}/

Description

ViewSet for follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this following.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- follow

PATCH /follow/{id}/

Description

ViewSet for follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this following.	integer
Body	**data** *required*		Follow

Responses

HTTP Code	Schema
200	Follow

Tags

- follow

POST /follow/{id}/unfollow/

Description

ViewSet for follow.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this following.	integer
Body	**data** *required*		Follow

Responses

HTTP Code	Schema
201	Follow

Tags

- follow

POST /follow_request/

Description

ViewSet for follow request.

Parameters

Type	Name	Schema
Body	**data** *required*	FollowRequestPost

Responses

HTTP Code	Schema
201	FollowRequestPost

Tags

- follow_request

GET /follow_request/

Description

ViewSet for follow request.

Parameters

Type	Name	Schema
Query	**req_to_user** *optional*	string

Responses

HTTP Code	Schema
200	< FollowRequest > array

Tags

- follow_request

GET /follow_request/{id}/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer

Responses

HTTP Code	Schema
200	FollowRequest

Tags

- follow_request

PUT /follow_request/{id}/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer
Body	**data** *required*		FollowRequest

Responses

HTTP Code	Schema
200	FollowRequest

Tags

- follow_request

DELETE /follow_request/{id}/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- follow_request

PATCH /follow_request/{id}/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer
Body	**data** *required*		FollowRequest

Responses

HTTP Code	Schema
200	FollowRequest

Tags

- follow_request

POST /follow_request/{id}/accept_follow/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer
Body	**data** *required*		FollowRequest

Responses

HTTP Code	Schema
201	FollowRequest

Tags

- follow_request

POST /follow_request/{id}/reject_follow/

Description

ViewSet for follow request.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this follow request.	integer
Body	**data** *required*		FollowRequest

Responses

HTTP Code	Schema
201	FollowRequest

Tags

- follow_request

POST /logout/

Description

Logs the user out, has to be authenticated

Responses

HTTP Code	Schema
201	No Content

Tags

- logout

POST /milestones/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	Milestone

Responses

HTTP Code	Schema
201	Milestone

Tags

- milestones

GET /milestones/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**project*id*** optional__	number

Responses

HTTP Code	Schema
200	< Milestone > array

Tags

- milestones

GET /milestones/get_user_milestones/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**project*id*** optional__	number

Responses

HTTP Code	Schema
200	< Milestone > array

Tags

- milestones

GET /milestones/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this milestone.	integer

Responses

HTTP Code	Schema
200	Milestone

Tags

- milestones

PUT /milestones/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this milestone.	integer

Type	Name	Description	Schema
Body	**data** *required*		Milestone

Responses

HTTP Code	Schema
200	Milestone

Tags

- milestones

DELETE /milestones/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this milestone.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- milestones

PATCH /milestones/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this milestone.	integer
Body	**data** *required*		Milestone

Responses

HTTP Code	Schema
200	Milestone

Tags

- milestones

GET /profile_picture/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer

Responses

HTTP Code	Schema
200	ProfilePicture

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- profile_picture

PUT /profile_picture/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer
FormData	**profile_picture** *optional*		file

Responses

HTTP Code	Schema
200	ProfilePicture

Consumes

- `multipart/form-data`
- `application/x-www-form-urlencoded`

Tags

- profile_picture

DELETE /profile_picture/{id}/

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer

Responses

HTTP Code	Schema
204	No Content

Consumes

- `multipart/form-data`

- `application/x-www-form-urlencoded`

Tags

- profile_picture

POST /profiles/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	ProfileFull

Responses

HTTP Code	Schema
201	ProfileFull

Tags

- profiles

GET /profiles/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**owner*id*** optional__	number

Responses

HTTP Code	Schema
200	< ProfilePrivate > array

Tags

- profiles

GET /profiles/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer

Responses

HTTP Code	Schema
200	No Content

Tags

- profiles

PUT /profiles/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer
Body	**data** *required*		ProfileFull

Responses

HTTP Code	Schema
200	ProfileFull

Tags

- profiles

DELETE /profiles/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- profiles

PATCH /profiles/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this profile.	integer

Type	Name	Description	Schema
Body	**data** *required*		ProfileFull

Responses

HTTP Code	Schema
200	ProfileFull

Tags

- profiles

POST /projects/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	ProjectPublic

Responses

HTTP Code	Schema
201	ProjectPublic

Tags

- projects

GET /projects/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Query	**member*id*** optional__	number
Query	**owner*id*** optional__	number

Responses

HTTP Code	Schema
200	< ProjectPrivate > array

Tags

- projects

GET /projects/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this project.	integer

Responses

HTTP Code	Schema
200	No Content

Tags

- projects

PUT /projects/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this project.	integer
Body	**data** *required*		ProjectPublic

Responses

HTTP Code	Schema
200	ProjectPublic

Tags

- projects

DELETE /projects/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this project.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- projects

PATCH /projects/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this project.	integer
Body	**data** *required*		ProjectPublic

Responses

HTTP Code	Schema
200	ProjectPublic

Tags

- projects

POST /ratings/

Description

Rating can be an integer in [0,10]. The from_user field is automatically the requesting user.

Parameters

Type	Name	Schema
Body	**data** *required*	Rating

Responses

HTTP Code	Schema
201	Rating

Tags

- ratings

GET /ratings/

Description

Users can only see the ratings they created.

Parameters

Type	Name	Schema
Query	**to_user** *optional*	string

Responses

HTTP Code	Schema
200	< Rating > array

Tags

- ratings

GET /ratings/{id}/

Description

Users can read, update, partial_update or delete only the ratings that themselves created.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this rating.	integer

Responses

HTTP Code	Schema
200	Rating

Tags

- ratings

PUT /ratings/{id}/

Description

Users can read, update, partial_update or delete only the ratings that themselves created.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this rating.	integer
Body	**data** *required*		RatingUpdate

Responses

HTTP Code	Schema
200	RatingUpdate

Tags

- ratings

DELETE /ratings/{id}/

Description

Users can read, update, partial_update or delete only the ratings that themselves created.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this rating.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- ratings

PATCH /ratings/{id}/

Description

Users can read, update, partial_update or delete only the ratings that themselves created.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this rating.	integer
Body	**data** *required*		RatingUpdate

Responses

HTTP Code	Schema
200	RatingUpdate

Tags

- ratings

POST /register/

Description

Registers a new user

Parameters

Type	Name	Schema
Body	**data** *required*	Register

Responses

HTTP Code	Schema
201	Register

Tags

- register

POST /search/

Description

Execute a search with a string

Parameters

Type	Name	Schema
Body	**data** *required*	SearchRequest

Responses

HTTP Code	Schema
201	SearchRequest

Tags

- search

POST /tags/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Schema
Body	**data** *required*	Tag

Responses

HTTP Code	Schema
201	Tag

Tags

- tags

GET /tags/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Responses

HTTP Code	Schema
200	< Tag > array

Tags

- tags

GET /tags/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this tag.	integer

Responses

HTTP Code	Schema
200	Tag

Tags

- tags

PUT /tags/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this tag.	integer
Body	**data** *required*		Tag

Responses

HTTP Code	Schema
200	Tag

Tags

- tags

DELETE /tags/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this tag.	integer

Responses

HTTP Code	Schema
204	No Content

Tags

- tags

PATCH /tags/{id}/

Description

This viewset automatically provides `list`, `create`, `retrieve`, `update` and `destroy` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this tag.	integer
Body	**data** *required*		Tag

Responses

HTTP Code	Schema
200	Tag

Tags

- tags

GET /users/

Description

This viewset automatically provides `list` and `detail` actions.

Responses

HTTP Code	Schema
200	< UserPrivate > array

Tags

- users

GET /users/{id}/

Description

This viewset automatically provides `list` and `detail` actions.

Parameters

Type	Name	Description	Schema
Path	**id** *required*	A unique integer value identifying this user.	integer

Responses

HTTP Code	Schema
200	No Content

Tags

- users

Definitions

AuthToken

Name	Description	Schema
password *required*	**Minimum length** : `1`	string
token *optional* *read-only*	**Minimum length** : `1`	string
username *required*	**Minimum length** : `1`	string

CollaborationInvite

Name	Description	Schema
created *optional*		string (date-time)
from_user *optional* *read-only*		string
id *optional* *read-only*		integer
message *optional*	**Maximal length** : `200`	string
rejected *optional*		string (date-time)
to_project *required*		integer
to_user *optional* *read-only*		string

CollaborationInvitePOST

Name	Description	Schema
created *optional*		string (date-time)
from_user *optional* *read-only*		string
id *optional* *read-only*		integer
message *optional*	**Maximal length** : `200`	string

Name	Description	Schema
rejected *optional*		string (date-time)
to_project *required*		integer
to_user *required*		integer

CollaborationRequest

Name	Description	Schema
created *optional*		string (date-time)
from_user *optional* *read-only*		string
id *optional* *read-only*		integer
message *optional*	**Maximal length** : `200`	string
rejected *optional*		string (date-time)
to_project *required*		integer
to_user *optional* *read-only*		string

Comment

Name	Description	Schema
comment *required*	**Length** : `1 - 1000`	string

Name	Description	Schema
created *optional* *read-only*		string (date-time)
from_user *required*		integer
id *optional* *read-only*		integer
last_name *optional* *read-only*		string
middle_name *optional* *read-only*		string
name *optional* *read-only*		string
owner *optional* *read-only*		string
to_user *required*		integer

CommentUpdate

Name	Description	Schema
comment *required*	**Length** : `1 - 1000`	string
id *optional* *read-only*		integer

Event

Name	Description	Schema
date *required*		string (date)
deadline *required*		string (date)
description *optional*	**Maximal length** : `200`	string
event_type *required*		enum (journal submission, academic conference, funded project)
id *optional* *read-only*		integer
title *required*	**Length** : `1 - 100`	string
url *optional*	**Maximal length** : `200`	string (uri)

ExampleModel

Name	Description	Schema
created_date *optional* *read-only*		string (date-time)
description *required*	**Length** : `1 - 255`	string
id *optional* *read-only*		integer

Name	Description	Schema
name *required*	**Length** : `1 - 255`	string

File

Name	Description	Schema
file *optional* *read-only*		string (uri)
id *optional* *read-only*		integer
project *required*		integer
remark *required*	**Length** : `1 - 20`	string
timestamp *optional* *read-only*		string (date-time)

Follow

Name	Schema
created *optional* *read-only*	string (date-time)
from_user *optional*	UserPrivate
id *optional* *read-only*	integer
to_user *optional*	UserPrivate

FollowPost

Name	Schema
created *optional* *read-only*	string (date-time)
to_user *required*	integer

FollowRequest

Name	Schema
created *optional* *read-only*	string (date-time)
id *optional* *read-only*	integer
req_from_user *optional*	UserPrivate
req_to_user *optional*	UserPrivate

FollowRequestPost

Name	Schema
created *optional* *read-only*	string (date-time)
req_to_user *required*	integer

Milestone

Name	Description	Schema
date *required*		string (date)
description *optional*	**Minimum length** : `1`	string
id *optional* *read-only*		integer
project *required*		integer

ProfileFull

Name	Description	Schema
affiliations *optional*		string
bio *optional*	**Maximal length** : `1000`	string
birthday *optional*		string (date)
email *optional* *read-only*		string
expertise *optional*		string
gender *optional*		enum (male, female, do not want to share)
id *optional* *read-only*		integer
interests *optional*		string

Name	Description	Schema
is_commentable *optional* *read-only*		string
is_public *optional*		boolean
last_name *optional*	**Length** : `1 - 100`	string
middle_name *optional*	**Maximal length** : `100`	string
my_rating *optional* *read-only*		string
my_rating_id *optional* *read-only*		string
name *optional*	**Length** : `1 - 100`	string
owner *optional* *read-only*		string
owner_id *optional* *read-only*		string
profile_picture *optional* *read-only*		string (uri)
rating *optional* *read-only*		string

Name	Description	Schema
share_affiliations *optional*		boolean
share_bio *optional*		boolean
share_birthday *optional*		boolean
share_gender *optional*		boolean

ProfilePicture

Name	Schema
id *optional* *read-only*	integer
profile_picture *optional* *read-only*	string (uri)

ProfilePrivate

Name	Description	Schema
id *optional* *read-only*		integer
is_commentable *optional* *read-only*		string
is_public *optional*		boolean

Name	Description	Schema
last_name *optional*	**Length** : `1 - 100`	string
middle_name *optional*	**Maximal length** : `100`	string
my_rating *optional* *read-only*		string
my_rating_id *optional* *read-only*		string
name *optional*	**Length** : `1 - 100`	string
owner *optional* *read-only*		string
owner_id *optional* *read-only*		string
profile_picture *optional* *read-only*		string (uri)

ProjectPrivate

Name	Description	Schema
description *optional*	**Minimum length** : `1`	string
id *optional* *read-only*		integer
is_public *optional*		boolean

Name	Description	Schema
name *required*	**Length** : `1 - 500`	string
owner *optional* *read-only*		string
owner_id *optional* *read-only*		string
project_type *optional*		enum (conference, instutution, journal)
state *optional*		enum (draft, inviting collaborators, open for collaborators, in progress, submitted to event, published, cancelled, done, reopened)
tags *optional* *read-only*		< Tag > array

ProjectPublic

Name	Description	Schema
description *optional*	**Minimum length** : `1`	string
due_date *optional*		string (date)
event *optional*		integer
id *optional* *read-only*		integer

Name	Description	Schema
is_public *optional*		boolean
name *required*	**Length** : `1 - 500`	string
owner *optional* *read-only*		string
owner_id *optional* *read-only*		string
project_type *optional*		enum (conference, instutution, journal)
requirements *optional*	**Minimum length** : `1`	string
state *optional*		enum (draft, inviting collaborators, open for collaborators, in progress, submitted to event, published, cancelled, done, reopened)
tags *optional*		< integer > array

Rating

Name	Description	Schema
created *optional* *read-only*		string (date-time)
from_user *required*		integer

Name	Description	Schema
id *optional* *read-only*		integer
rating *required*	**Minimum value** : `0` **Maximum value** : `10`	integer
to_user *required*		integer

RatingUpdate

Name	Description	Schema
id *optional* *read-only*		integer
rating *required*	**Minimum value** : `0` **Maximum value** : `10`	integer

Register

Name	Description	Schema
email *required*	**Minimum length** : `1`	string (email)
first_name *required*	**Minimum length** : `1`	string
last_name *required*	**Minimum length** : `1`	string
middle_name *required*		string
password *required*	**Minimum length** : `1`	string
username *required*	**Minimum length** : `1`	string

SearchRequest

Name	Description	Schema
event_date_after *optional*		string (date)
event_date_before *optional*		string (date)
event_deadline_after *optional*		string (date)
event_deadline_before *optional*		string (date)
event_type *optional*		enum (journal submission, academic conference, funded project)
keyword *required*	**Minimum length** : 2	string
profile_affiliations *optional*	**Minimum length** : 1	string
profile_expertise *optional*	**Minimum length** : 1	string
project_due_date_after *optional*		string (date)
project_due_date_before *optional*		string (date)

Name	Description	Schema
project_event *optional*		integer
project_state *optional*		enum (draft, inviting collaborators, open for collaborators, in progress, submitted to event, published, cancelled, done, reopened)
search_type *optional*	**Default** : `"all"`	enum (all, project, profile, event)

Tag

Name	Description	Schema
color *optional*	**Minimum value** : `-2147483648` **Maximum value** : `2147483647`	integer
id *optional* *read-only*		integer
name *required*	**Length** : `1 - 500`	string

UserPrivate

Name	Description	Schema
count_of_follow_requests *optional* *read-only*		string
count_of_followers *optional* *read-only*		string

Name	Description	Schema
count_of_followings *optional* *read-only*		string
id *optional* *read-only*		integer
is_follow_request_received *optional* *read-only*		string
is_follow_request_sent *optional* *read-only*		string
is_follower *optional* *read-only*		string
is_following *optional* *read-only*		string
profile *optional* *read-only*		< ProfilePrivate > array
username *required*	Required. 150 characters or fewer. Letters, digits and @/./-/_ only. + **Length** : \`1 - 150\` + **Pattern** : \`"^[\\\\w.@-]+$"\`	string