Milestone II Report

Group 8

Doğukan Türksoy
Elif Bayraktar
Furkan Keskin
Karahan Sarıtaş
Metehan Dündar
Mustafa Cihan
Sena Mumcu
Serdar Akol
Serhat Hebun Şimşek
Sinem Koçoğlu

1 Executive Summary

1.1 Summary of project and overall status

We are a group of Bogazici University students who are taking the CMPE352 course in the Spring 2022 semester. For the second milestone in this course, we have extensively worked and studied and developed an Art Community Platform where users can interact with other users and display different art works.

We built an application management system which is available on the Web. After signing up and logging in, a user can share new art items and also view all of the art items in the system. Admin can add comments on art items for a specific user, delete his/hers existing art item. To make the tag functionality more interactive we added functionalities like creating new tags, deleting an existing tag, listing all of the available tags in the system and also searching art items by their tags.

1.2 URL of our deployed application

http://54.209.217.195:8000/

1.3 API Documentation URL

https://github.com/bounswe/bounswe2022group8/wiki/Practice-App---API-Documentation

1.4 Tag URI:

https://github.com/bounswe/bounswe2022group8/releases/tag/v1.1.0

1.5 Basic Functionality of Our Project

This Web application accomplishes various tasks, which are mostly the primitive form of the functionalities of the big project “Art Community Platform”. When one clicks the url of the deployed application, they would see most of the solid functionalities of our application. With the application, users are navigated through the web pages easily. There we present our api endpoints basically. Firstly, they can sign up and login the application by giving their personal information. By clicking the right links, users can search for a specific user by providing username or user id, or if they wish they can get the list of all users. Apart from that, they can create a user, after that, delete the user by providing the id. They can upload art items by providing necessary information, or list the all art items. They can search for the specific art item by giving the id of the art item, or they can list all of the art items of a user by giving the username or id of the user. They can even search for art items by giving tag or tag - keyword. Apart from these operations they can delete an art item, or make a comment on art items. After that, they can get the list of the comments of an art item by giving the id of the art item or if they wish they can delete the comments by giving the comment and ar item id. Users of this practice application can add, delete or list tags. Furthermore they can follow a user by a http request using Postman, this functionality is not presented on the web application home page. Unfortunately it does not have a html page, but they can list the follower or following users of a user, if they wish, by using the undermost links on the home page. Apart from these, users can search the episodes of tv series by giving the name of it, or they can list the unanswered questions on stackexchange. For developing these last two functionalities, we have used the tvmaze.com and stackexchange APIs.

1.6 Challenges we met as a group

- None of us was familiar with web development before. Therefore, we had difficulty understanding the whole concept of interacting with the end-user

on Frontend and consuming our REST APIs. However, eventually we understood the whole concept and implemented our project accordingly.

- Specifications were too ambiguous and therefore we had headaches trying to understand what we were requested.
- Looking at a small piece of code that uses abstractions in django (like models and forms), it is easy to underestimate the effort that goes into it. Admittedly once you get the hang of it, it does make life a whole lot easier, but there is a steep learning curve (especially as a newbie) when you're trying to learn structures in web development and the abstractions that come on top with tools like the framework, distinctions between different layers, which one you're supposed to use where and how different components interact.
- Another difficulty (again caused by inexperience) is that once you get an error it is more difficult and time consuming to debug. As you get more experienced it gets easier to understand and solve various bugs, but as a rookie it might take a lot of time and research even to solve a very trivial error.

1.7 List of status of deliverables

Deliverable	Status	Update Frequency	Description
Github Issues	In Progress	After each meeting & as new tasks are assigned	A bug or progress tracking system built into the GitHub service.
Github Wiki	In Progress	Daily	Documentation of our project and work.
Meeting Notes	In Progress	Weekly (generally two times per week)	Agenda, discussion and action items of our meetings
Practice App Requirements	Completed Delivered on 20/05/22.	As improvement needed.	Features, functions, and tasks that need to be completed for

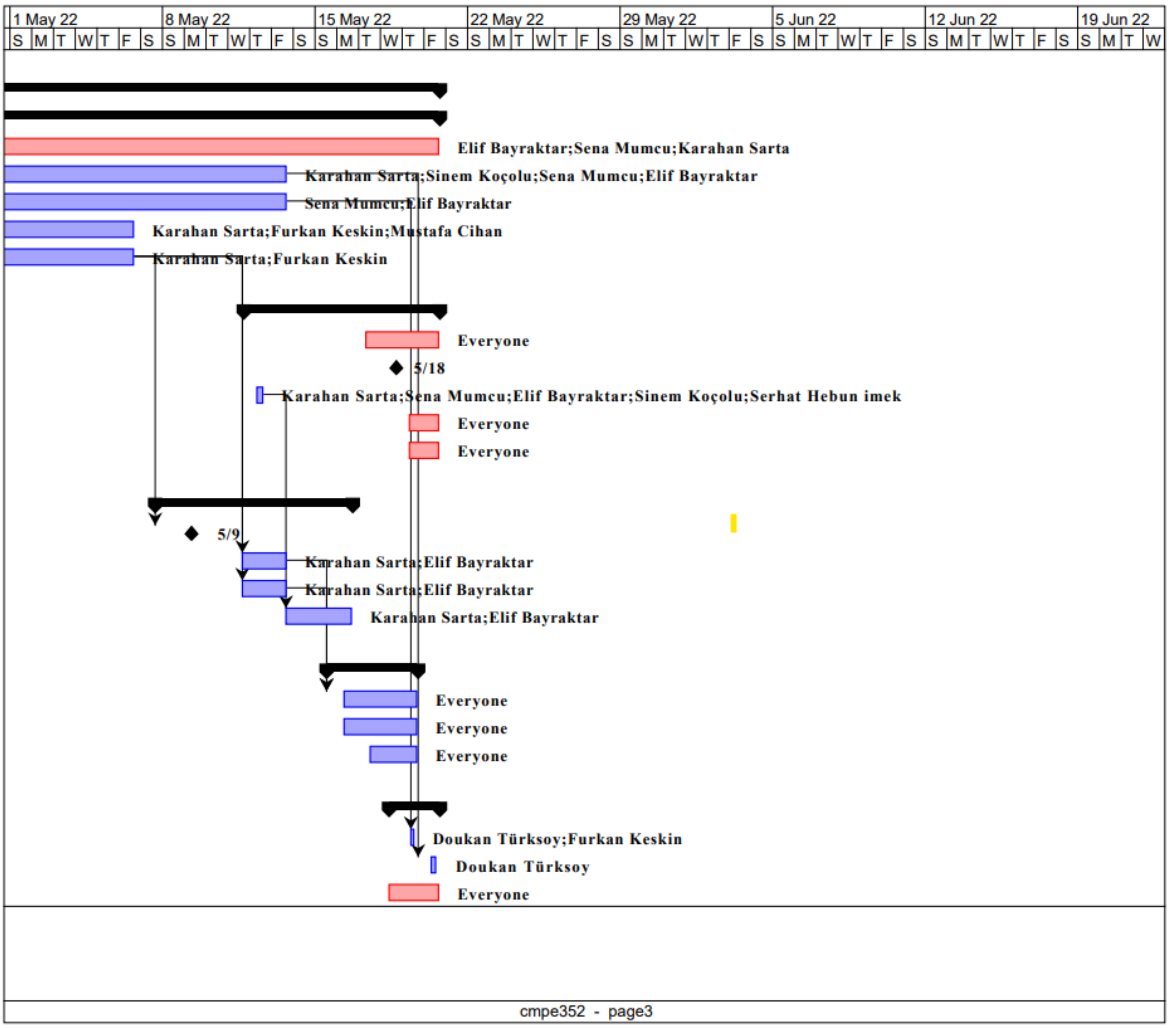
			our practice app to be deemed successful.
Design Diagrams (Use-case Diagram)	Completed Delivered on 20/05/22	As improvement needed.	A use case diagram is a graphical depiction of a user's possible interactions with a system.
Design Diagrams (Class Diagram)	Completed Delivered on 20/05/22	As improvement needed in our database design.	A class diagram describes the attributes and operations of a class and also the constraints imposed on the system.
Project Plan	Completed Delivered on 20/05/22	As our plans change - improvement needed	Project plan: Identification of tasks, estimations and task allocation
API Documentation	Completed Delivered on 20/05/22	As each team member completed implementation of their API endpoints.	API Documentation is a technical document that describes the nature of the APIs, how to use them and expected responses.
Code	Completed Delivered on 20/05/22	As the development goes on.	The code of the project along with a brief guide that describes how to build and run it using **docker**.
URI of the Tag of our project	Completed Delivered on 20/05/22	-	Github release to bundle and deliver iterations of a project to users.

Milestone II Report	Completed Delivered on 20/05/22	-	Milestone group deliverable that contains all other deliverables as well (you are reading it right now)

1.8 Project Plan

		Name	Duration	Start	Finish	Predecessors
1		Milestone 1 Report	3 days	4/13/22 8:00 AM	4/15/22 5:00 PM	
2		**Practice Application**	**24.125 day...**	**4/26/22 4:00 PM**	**5/20/22 5:00 PM**	
3		**Research**	**24.125 days**	**4/26/22 4:00 PM**	**5/20/22 5:00 PM**	
4		Research on Git	24.125 days	4/26/22 4:00 PM	5/20/22 5:00 PM	
5		Research on AWS EC2	17.125 days	4/26/22 4:00 PM	5/13/22 5:00 PM	
6		Research on Docker	17.125 days	4/26/22 4:00 PM	5/13/22 5:00 PM	
7		Research on Postman	10.125 days	4/26/22 4:00 PM	5/6/22 5:00 PM	
8		Research on Django	10.125 days	4/26/22 4:00 PM	5/6/22 5:00 PM	
10		**Design for Practice App**	**9 days**	**5/11/22 5:00 PM**	**5/20/22 5:00 PM**	
11		Requirements	4 days	5/17/22 8:00 AM	5/20/22 5:00 PM	
12		Class Diagram	0 days	5/18/22 5:00 PM	5/18/22 5:00 PM	
13		ER Diagram	0.8 days	5/11/22 5:00 PM	5/12/22 3:24 PM	
14		Use Case Diagram	2 days	5/18/22 5:00 PM	5/20/22 5:00 PM	
15		Sequence Diagram	2 days	5/18/22 5:00 PM	5/20/22 5:00 PM	
17		**Initial Code Implementation**	**9.125 days**	**5/7/22 4:00 PM**	**5/16/22 5:00 PM**	
18		Initial configuration of practice app	0 days	5/7/22 4:00 PM	5/9/22 5:00 PM	8
19		Implementation of models	2.125 days	5/11/22 4:00 PM	5/13/22 5:00 PM	8
20		Implementation of serializers	2.125 days	5/11/22 4:00 PM	5/13/22 5:00 PM	8
21		Creating database	3.125 days	5/13/22 4:00 PM	5/16/22 5:00 PM	13
23		**Implementation**	**4.5 days?**	**5/15/22 1:00 PM**	**5/19/22 5:00 PM**	
24		Implementation of API endpoints	4 days?	5/15/22 1:00 PM	5/19/22 5:00 PM	19;20
25		Implementation of frontend templates	4 days?	5/15/22 1:00 PM	5/19/22 5:00 PM	
26		Implementation of unit tests for API endpoints	2.5 days?	5/17/22 1:00 PM	5/19/22 5:00 PM	
28		**Project Configurations**	**2.75 days?**	**5/18/22 10:00 AM**	**5/20/22 5:00 PM**	
29		Docker Configuration	0.375 days?	5/19/22 10:00 AM	5/19/22 2:00 PM	6
30		AWS Deployment Configuration	0.75 days?	5/19/22 5:00 PM	5/20/22 3:00 PM	5
31		API Documentation	2.75 days?	5/18/22 10:00 AM	5/20/22 5:00 PM	


Resource Names
6 Mar 22
13 Mar 22
20 Mar 22
27 Mar 22
3 Apr 22
10 Apr 22
17 Apr 22
24 Apr 22
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S
Everyone
Everyone
Elif Bayraktar;Sena Mumcu;...
Karahan Sarta;Sinem Koço...
Sena Mumcu;Elif Bayraktar
Karahan Sarta;Furkan Kes...
Karahan Sarta;Furkan Keskin
Everyone
Karahan Sarta
Karahan Sarta;Sena Mumc...
Everyone
Everyone
Karahan Sarta
Karahan Sarta;Elif Bayraktar
Karahan Sarta;Elif Bayraktar
Karahan Sarta;Elif Bayraktar
Everyone
Everyone
Everyone
Doukan Türksoy;Furkan Ke...
Doukan Türksoy
Everyone



2. Evaluation of the Status of Deliverables

- Github Issues

  After each meeting, everyone is responsible for creating the related issue in accordance with the action items of our discussion. Issues are used to communicate the progress of the tasks and inform each other in an organized way. Upon completion of a task, the owner of the issue is responsible for commenting on the current status of the task and label the issue as *Review needed* for the reviewer to make the necessary checks. We have also used a task *list* feature for tasks that requires everyone's individual contribution.

- Meeting Notes

  After each meeting, we have prepared meeting notes including the agenda of the week, important points from the discussion and action items. We tracked the list of

participating team members. Keeping the meeting notes helped us to catch up with the progress and discussions.

- Github Wiki
  Used our Wiki very frequently in order to reflect and document our progress. Updated our wiki page as needed and added necessary comments (issue numbers etc.) to the revision descriptions.

- Practice App Requirements
  During our meetings we discussed the functionalities we are going to provide in our application. Then we distributed the features among the team members. Each team member implemented their own feature(s) and documented related requirements in the wiki page. At the second phase, we validated and revised the requirements by testing our application via frontend, tests and API calls using third-party testing tools such as Postman.

- Design Diagrams (Class Diagram - Use Case Diagram - Sequence Diagram)
  We started creating an ER diagram for our database. Then related assignees have prepared models and serializers in accordance with our ER diagram. We have generated our class diagram using Graphviz and pyparsing on Django. Then we have created use case diagrams and sequence diagrams according to the features we have implemented.

- Project Plan
  Project plan includes the tasks, expected amount of time each task will take, and team members assigned for that specific task.

- API Documentation
  After implementing our features, we have created an API documentation that includes instructions on how to effectively use the API, response and example API calls.

- Code
  We have implemented our practice application on Python using Django and Django REST Framework. Initially, we have splitted into three branches to implement models and serializers. Then, each team member started working on their own branch in order to implement their features. After the working is done, pull requests are sent and branches are merged to the main branch named as **practice_app**. We tried to keep our working space as clean as possible and complied with the format provided in the issues. Each team member has written their APIs, HTML templates and tests in specific folders. Eventually, we merged the **practice_app** into **master** branch to finalize our work.

- URI of the Tag of our project
  After finalizing our work, we have generated a tag by releasing our project on Github. We have provided a brief description about the application.

- Milestone II Report
  We have prepared our second milestone report including all the deliverables listed above.

3. Brief Summary of Work Done by Team Members

- Karahan Sarıtaş

Task	Issues	PRs (if related)
Individual Postman Research (I was familiar with Postman and occasionally I showed others how to use it to test their APIs.)	#89	
AWS EC2 Research - Watched some videos on the topic - created an AWS Account and filled up the Research page.	#87	
Made the initials for Django project and API application. Documented each step, prepared a README.	#92	
Worked together on ER Diagram Design.	#94	
Implemented Models and Serializers	#96, #100	#97, #101
Made extensive research on Django to inform others and make starting procedure as easy as possible for others.	#103	
Implemented art-item related endpoints. (4 GET - 1 POST API)	#105	#118, #120
Implemented TV Series related endpoint (external API).	#106	#118
Generated the class diagram using Django Extensions.	#127	
Created HTML templates that includes some CSS, input forms and Javascript to consume our REST API. I informed the team about which parts they have to modify to create their own pages, and even helped them frequently.	#133	#118
Created a tag for the application	#145	
Reviewed codes of others officially (PRs) and		#93, #117, #119,

unofficially (on Discord). I resolved so many merge conflicts that I started feeling connected to them.		#128, #130, #135
Created use case diagram for art item related actions and fetching episodes of a TV series from external API.	#125	
Created sequential diagram that depicts the flow when the end user attempts to get information about a TV series and list of episodes.	#126	
Prepared API documentation for the API points I've implemented.	#115	
Prepared requirements for the APIs I've implemented.	#121	

- Doğukan Türksoy

Task	Issues	PRs (if related)
AWS EC2 Deployment	#144	
Dockerization	#143	
Implemented API Get Users by Credentials		
Implemented Sign Up and Login HTML's		
Approved Karahan's pull request		App/cleanup

- Sinem Koçoğlu

Task	Issues	PRs (if related)
Research on AWS EC2	#87	

Meeting notes #14 documentation	#91	
Worked together on ER diagram in meeting #15	#94	
Created sequence diagram for comment feature	#126	
Created use case diagram for comment feature	#125	
Created the project plan file and added tasks and resource names	#132, #137	
Documented my API endpoints	#104, #116	
Documented comment endpoint requirements for practice app	#121	
Implemented comment feature and unit test of it for practice app	#102	#119
Implemented html files for comment on practice app	#102	#130
Reviewed pull&request feature/artitem		#120
Reviewed pull&request feature/tags.		#136

- Serhat Hebun Şimşek

Task	Issues	PRs(if related)
Worked together on ER	#94	

Diagram Design.		
Created Sequence Diagram document and diagram #5	#126	
Created Search User and implemented unit tests and html for Search User	#134	#135
Reviewed and approved Feature/questions PR		#141
Created Use Case Diagram document	#125	
Documented my API endpoints	#104,#116	
Did research on Docker	#88,#98	

- Sena Mumcu

Task	Issues	PRs(if related)
Did research on Docker	#88, #98	
Did research on Postman	#89	
Meeting notes #15 documentation	#95	
Worked together on ER diagram in meeting #15	#94	
Created sequence diagram for tag feature	#126	
Created use case diagram for tag feature	#125	

Documented my API endpoints	#104, #116	
Created Requirements page for practice app and added non-functional and functional requirements	#121	
Wrote project summary for Milestone 2	#123	
Added date entries to Project plan for Milestone 2	#132	
Implemented tag feature, unit test, html files for practice app	#131	#136
Reviewed pull request		#142, #130

- Serdar Akol

Task	Issues	PRs (if related)
Writing the Basic Functionality part of the Milestone Report II	#147, #137	
Writing the api endpoints regarding the follow functionality of the application	#146	
Playing with the git commands	#86	
Implementing the html pages of the get_followers and get_followings functionalities		
Documenting the API endpoints	#116	
Writing the meeting notes #11		

Attending the meetings: #12, #13, #14, #16		
Did research on postman	#89	
Did research on AWS	#87	
Did research on Django and learned it	#103	

- Elif Bayraktar

Task	Issues/Related links	PRs(if related)
Researched dockerization and documented on repo	issue , #88 , wiki page comment	
Updated .gitignore	commit	
Designed ER diagram with @mumcusena @simsekhebun @sinemKocoglu @KarahanS	issue	
I created the (Django) models to represent the database part of our app together with *Karahan Sarıtaş*. Also created the corresponding serializers.		Model1 PR , model2 PR
I implemented search_by_tag API	issue , issue#107	Pull request
I created a user interface for search_by_tag API, via using Django forms and templates	Issue#111	
Created unittests for search_by_tag API. Later added more tests	issue#107 , commit	
Created template for homepage with links to relevant pages	Issue#109 commit	

Created base template	Issue#110 commit	
Modified existing templates to extend the base.	commit	
Suffix-formatted url patterns.	commit	
Added questions API. Functionality explained in detail here #139	here #139	PR
Created frontend for questions	commit	
Created unit-tests for questions API	commit	
Set aside a few hours to help teammates with their code & git related problems.		
Spent countless hours researching django, REST APIs, testing, dockerization, deployment.		
Made contributions to requirements.	wiki issue#121	
Made contributions to the Milestone-2 Group report.		
Made contributions to use case diagram	issue#125	
Created 3 sequence diagrams	issue#126	
Created documentations form my API endpoints	Issue#116 wiki	
Revised teammates code, solved conflicts , merged pull requests.		model2 PR users_frontend

4. Evaluation of Tools and Processes

Github: Github is a platform where we can communicate through issues, assign tasks, document our weekly meetings and show what we have produced as a team. It is simply the main platform on which we work and mature our project. It also helps us stay organized and up to date.

Discord: Discord is the platform on which we hold our weekly meetings. We have voice and text channels separated by topic, allowing us to communicate in a very organized, clear and practical way. We also use this platform for sharing files, distributing tasks and working out ideas that come to us during the week.

Whatsapp: We use Whatsapp as it allows practical, fast and easy communication. It is usually used to decide meeting time and to discuss urgent matters.

Trello & Monday: At the beginning of the semester, we thought of using an application specifically for project management, in addition to other communication tools we had already decided to use. We gave up on this idea because free versions of these applications offered limited access and we felt the communication tools we used were sufficient.

Uizard: Uizard is the tool that we used for creating mock-up scenarios. Although it was not very practical and did not offer much variety for the scenarios we wanted to design, we were able to achieve a good result with it. Besides, it was a big plus for us that it allowed collaborative work.

Lucidchart: We used Lucidchart to design UML diagrams. As a team, we were quite satisfied with Lucidchart because it lends itself to collaborative work and we could find everything we needed for the diagrams we wanted to design.

ProjectLibre: ProjectLibre was used as the project planning tool for both Milestone 1 and Milestone 2. Although we did not find projectlibre very useful when planning the project for the Milestone 1 report, we decided to continue with this application for Milestone 2 as well. Because we learned how to use it and there is a built-in feature where we can get the PDF version of our project plan.

VS Code: Microsoft Visual Studio Code is a lightweight code editor. We chose this app as the code editor because everyone in the group was already familiar with the app. It was indeed very useful and the built-in git management tools provided a lot of convenience during the project. Also, thanks to its active community, it was very easy to find a solution when we faced any problem with regard to the app.

Django: We built our API using the Django web framework. Although many of us had not been familiar with web development, we were able to get the basics of web development thanks to Django. It really is easy to learn thanks to hundreds of great tutorials out there. Django's philosophy of providing everything needed and leaving everything else to the user was also highly appreciated by our group. Thus, we were able to develop our project in a very simple and understandable developing environment.

Django Rest Framework: As we needed to develop a RESTful API, we chose this framework to provide the necessary tools. Understanding and using this framework was even easier than understanding Django. The fact that this framework built in *response* method has a debug screen really worked for us and made it very easy for us to develop the project.

Postman: Postman is a wonderful API testing tool and is probably one of the most useful tools we have ever used. Also, it was also very easy to learn how to use. Using Postman, we were able to test the required parts directly via JSON objects without the need for an interface, which greatly accelerated the debugging process.

SQLite: SQLite is a software library that provides a relational database management system. In one of our meetings, we talked about whether to use mysql or not, but then we continued with SQlite, which Django provides as built-in. So everyone could make changes in their local database as they wished.

Faker: Faker is a Python package that allows us to create mock data for our unit tests. With Faker, we were able to test our API endpoints on a wide variety of data.

Base64 Encode & Decode Flow: The biggest problem we encountered while writing our APIs was probably how we could send images within JSON objects. The solution we found, in short, is: extract the image from the media file, encode the image with Base 64, embed it in

the JSON object, execute the required queries, decode the Base 64 encoded image from the incoming response object.

Docker: Docker is the tool we used to automate the deployment of our application in a lightweight container so that our application can work efficiently in different environments**.** Honestly, dockerizing our app was a lot harder than we expected. Although we understand what we need to do in general through some tutorials on the internet, we dealt with an unexpected bug for hours. The problem was that the tkinter Python package was called by some Docker generated Python files, but Docker did not automatically take care of it. Also, we couldn't find the solution we wanted on the internet for a long time. Fortunately, after dozens of unsuccessful attempts we were able to fix the problem. In short, a process that should have taken a very short time took a long time, but we achieved the result we wanted.

Note: At least, thanks to this error, we had the opportunity to use and better understand the WSL.

AWS Deployment: AWS is a widely used service for initializing virtual machines. It was hard to initialize EC2 instance at the first place, because we need to fill so much information before registering. After creating the instance, we connected the instance via SSH and deployed our application into the Amazon Linux machine.

Use case diagram

Hebun Simsek | May 20, 2022

Name of the TV Series doesn't have to be the exact match. It uses fuzzy string searching algorithm.

Miscellaneous

- Enter the name of the TV Series
- Get information about a TV Series and list of the episodes. <<includes>> Enter the name of the TV Series
- Get a list of all unanswered stackexchange questions
- Get a list of all unanswered stackexchange questions with the given tag <<includes>> Text Enter the name of the tag

Comment Management

- Enter the user ID who posts the comment
- Enter the art item ID
- Enter the body of new comment
- Enter the user ID whom the comment belongs to
- Enter the comment ID
- View comments of an art item <<includes>> Enter the art item ID
- Delete comment of a user <<includes>> Enter the user ID whom the comment belongs to <<includes>> Enter the comment ID
- Enter the art item ID the new comment will be posted on
- Post comment on an art item <<includes>> Enter the art item ID the new comment will be posted on <<includes>> Enter the body of new comment <<includes>> Enter the user ID who posts the comment

User Management

- View all users in the system
- View users <<extends>> View all users in the system <extends> View a user by user ID <<includes>> Enter user ID <extends> View a user by username <<includes>> Enter username
- Create a User <<includes>> Set unique username <<includes>> Set password <<includes>> Enter name <<includes>> Enter surname <<includes>> Enter email <<includes>> Provide a profile image for the user or set default profile image.
- Delete user <<extends>> Delete a user by user ID

Tag Management

- Click on the create button
- Delete an exiting tag <<includes>> Click on the create button <<includes>> Enter the id of the tag
- Create a new tag <<includes>> Enter the name of the tag <<includes>> Enter the description of the tag <<includes>> List all tags
- Click on the delet tag button
- Enter the name of the tag
- Enter the description of the tag
- List all tags <<includes>> Click on the list tags button
- Get a list of all art items that have the given tag <<includes>> Enter the name of the tag
- Click on the list tags button
- Enter the name of the tag

Art Item Management

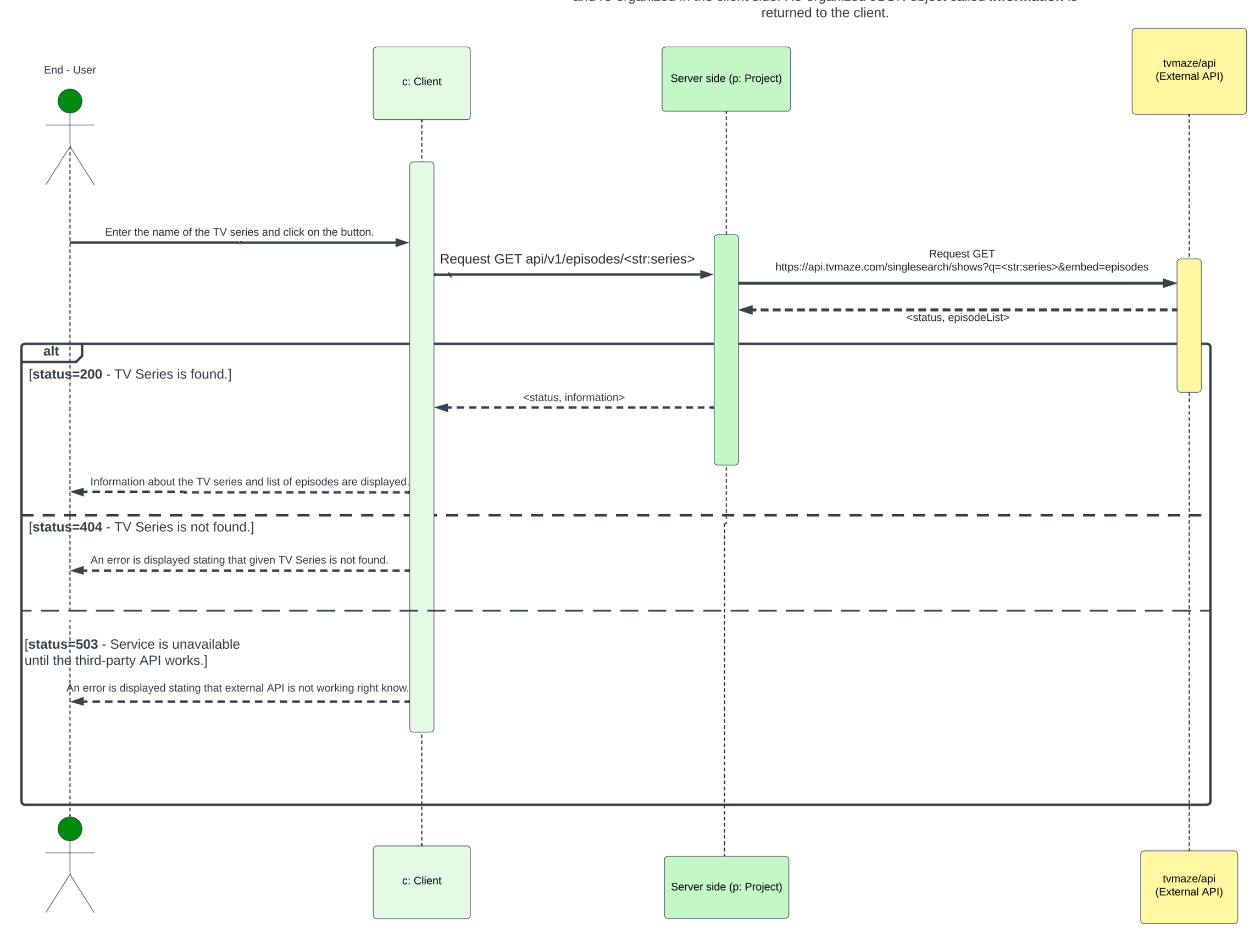
- View All art items in the System
- View art items <<extends>> View All art items in the System <extends> View an art item by ID <<includes>> Enter the art item ID <extends> View art items of a user by user ID <<includes>> Enter the ID of the user <extends> View art items of a User by Username <<includes>> Enter the username of the User
- Create an Art Item (for any user) <<includes>> Enter the ID of the owner of the art item <<includes>> Enter title and description for the art item <<includes>> Provide an image for the art item or set a default image.
- Delete art item <<extends>> Delete an art item by art item ID

End-User

Sequence Diagram #1: Getting list of episodes using an external API

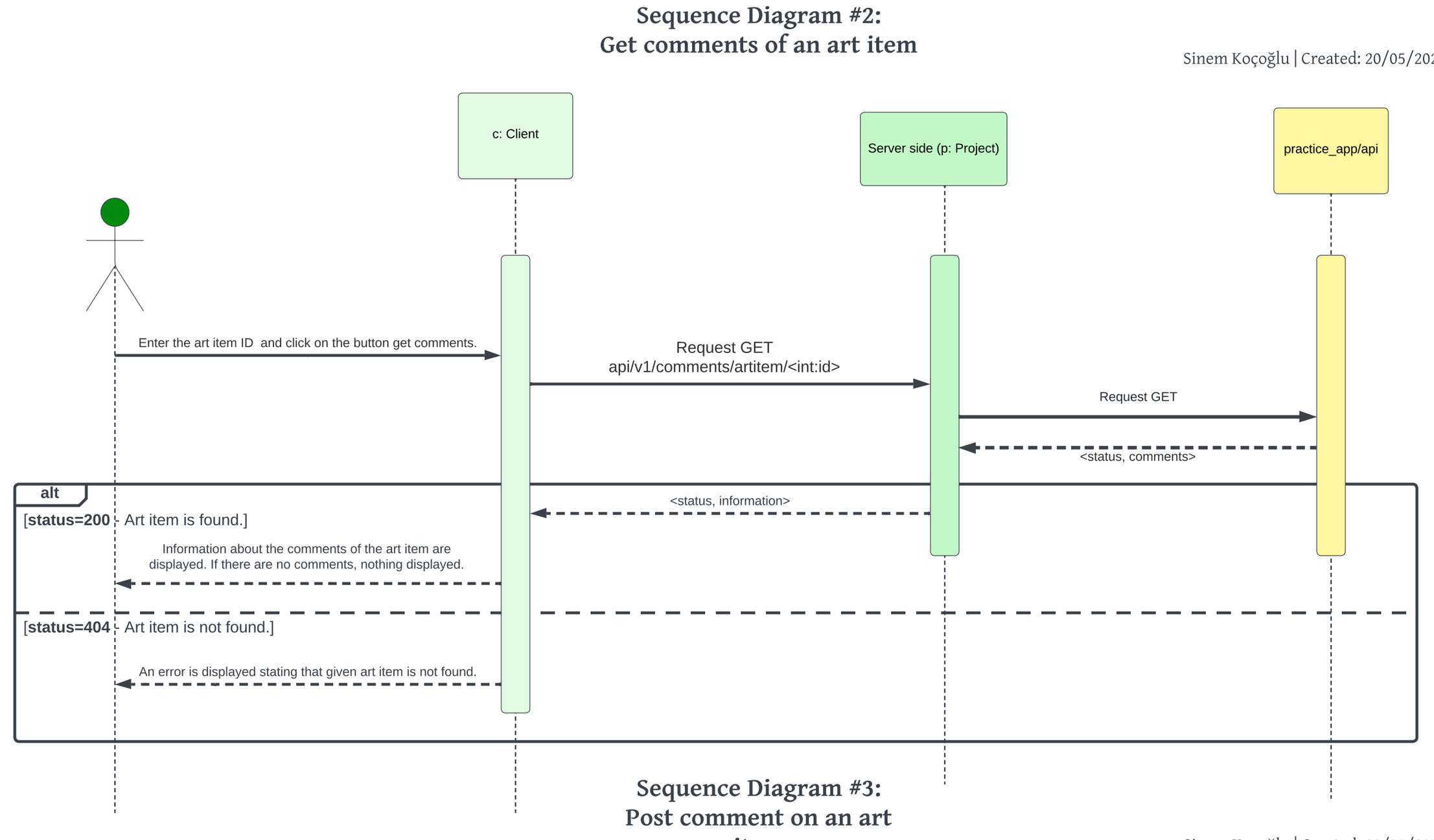
Karahan Sarıtaş | Created: 20/05/2022

Note: **episodeList** is a JSON object that is returned from the external API. It includes unnecessary details for our purpose of use. Necessary data is extracted from this object and re-organized in the client side. Re-organized JSON object called **information** is returned to the client.



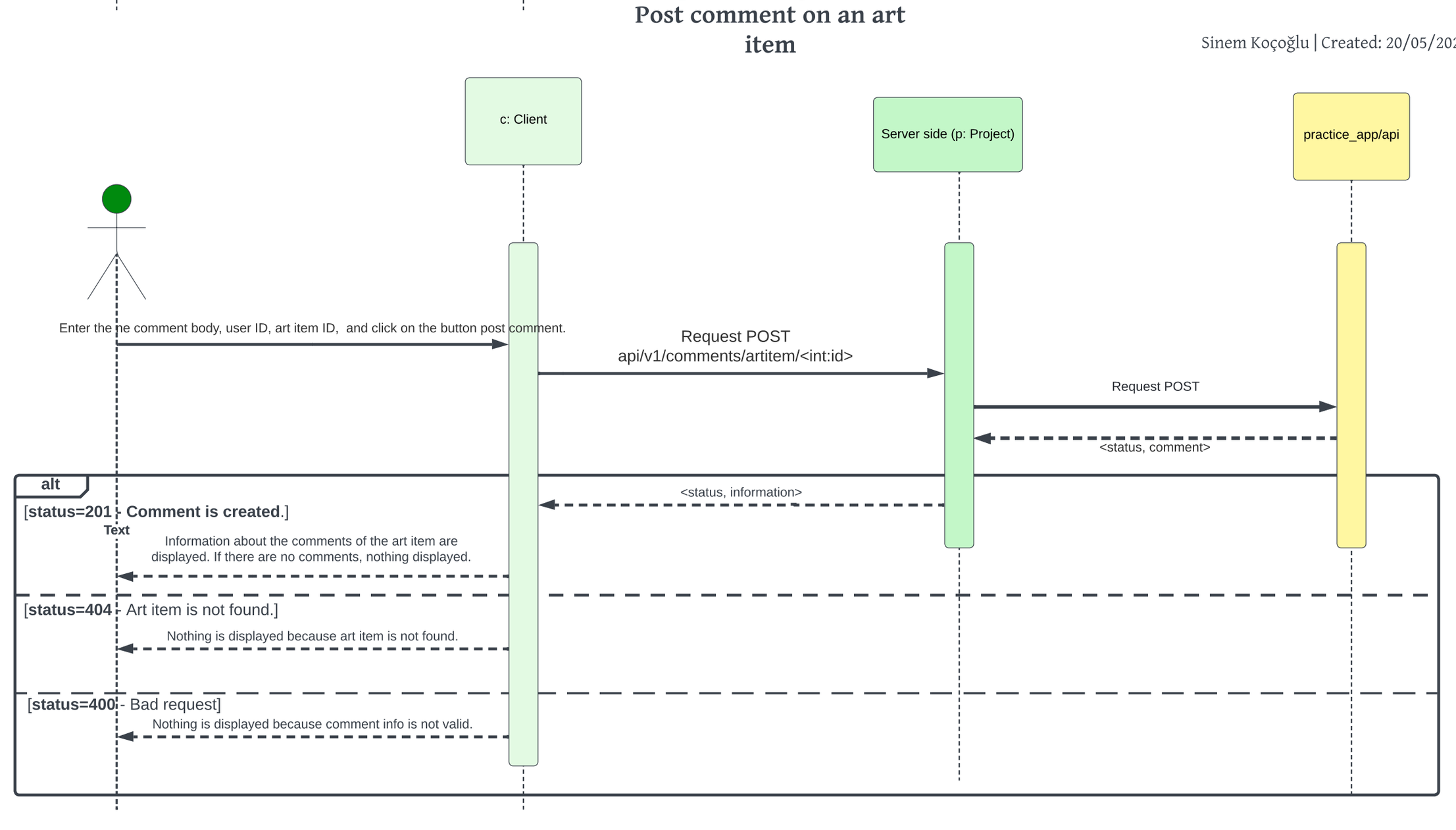
Sequence Diagram #2: Get comments of an art item

Sinem Koçoğlu | Created: 20/05/2022



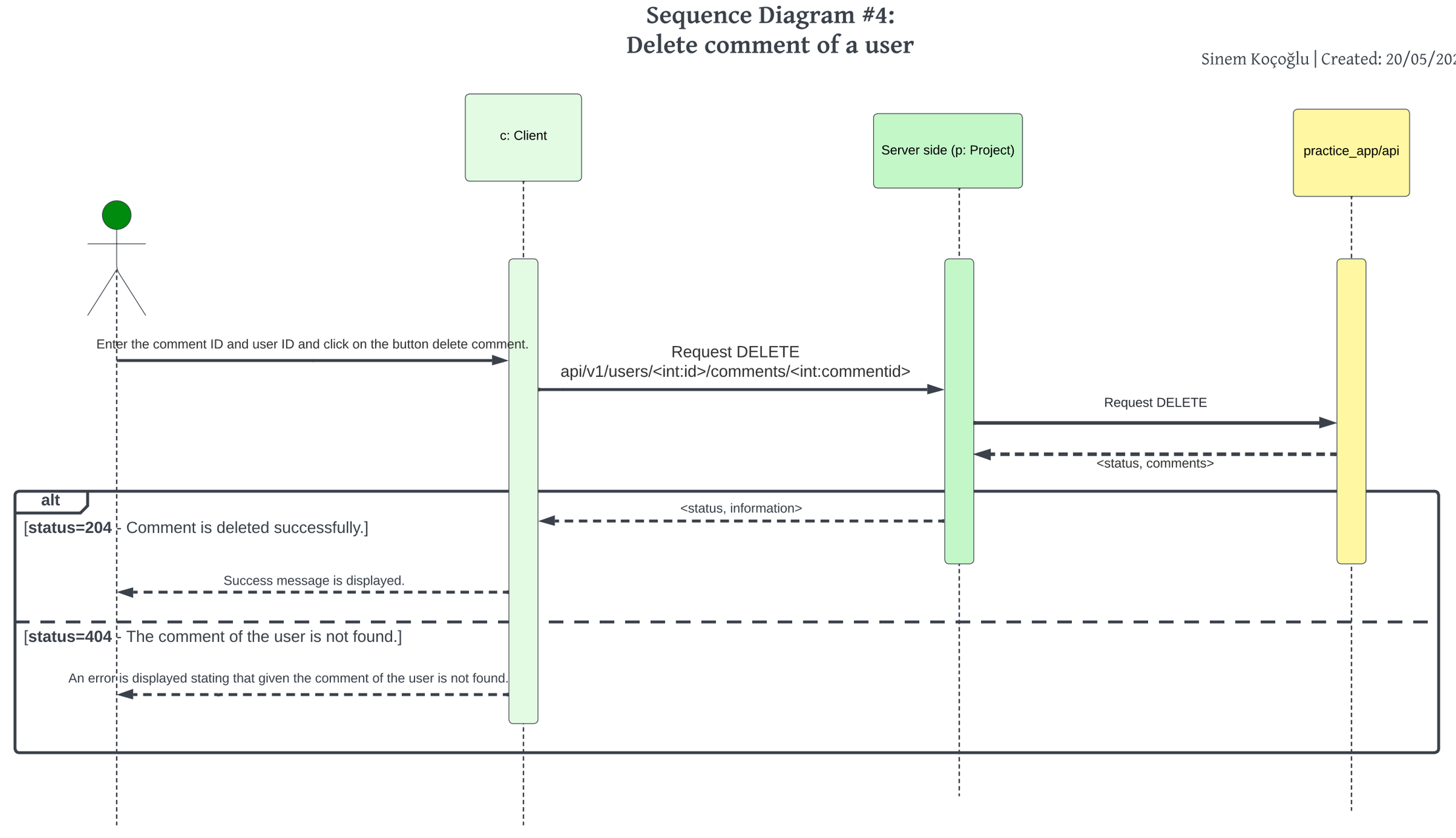
Sequence Diagram #3: Post comment on an art item

Sinem Koçoğlu | Created: 20/05/2022



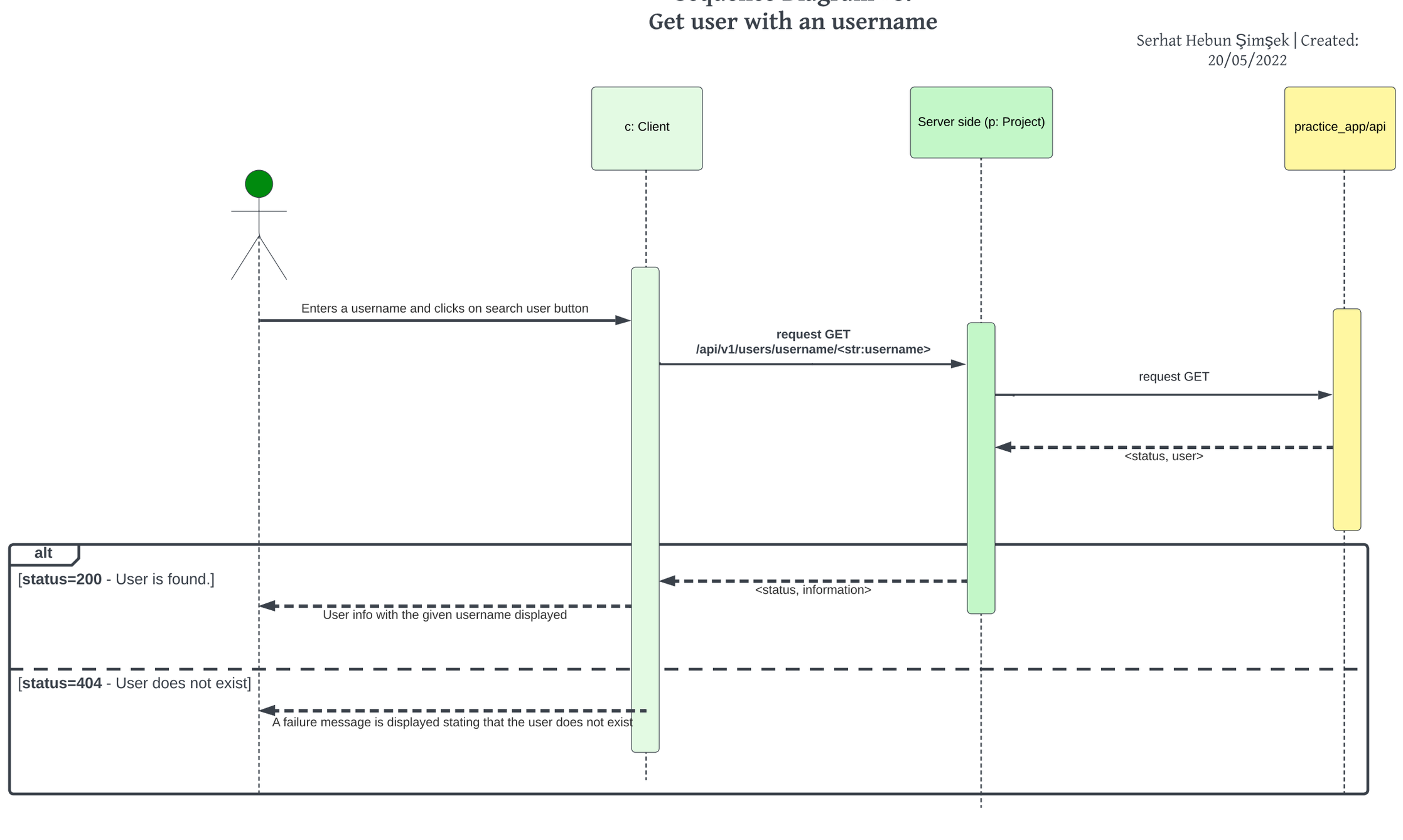
Sequence Diagram #4: Delete comment of a user

Sinem Koçoğlu | Created: 20/05/2022



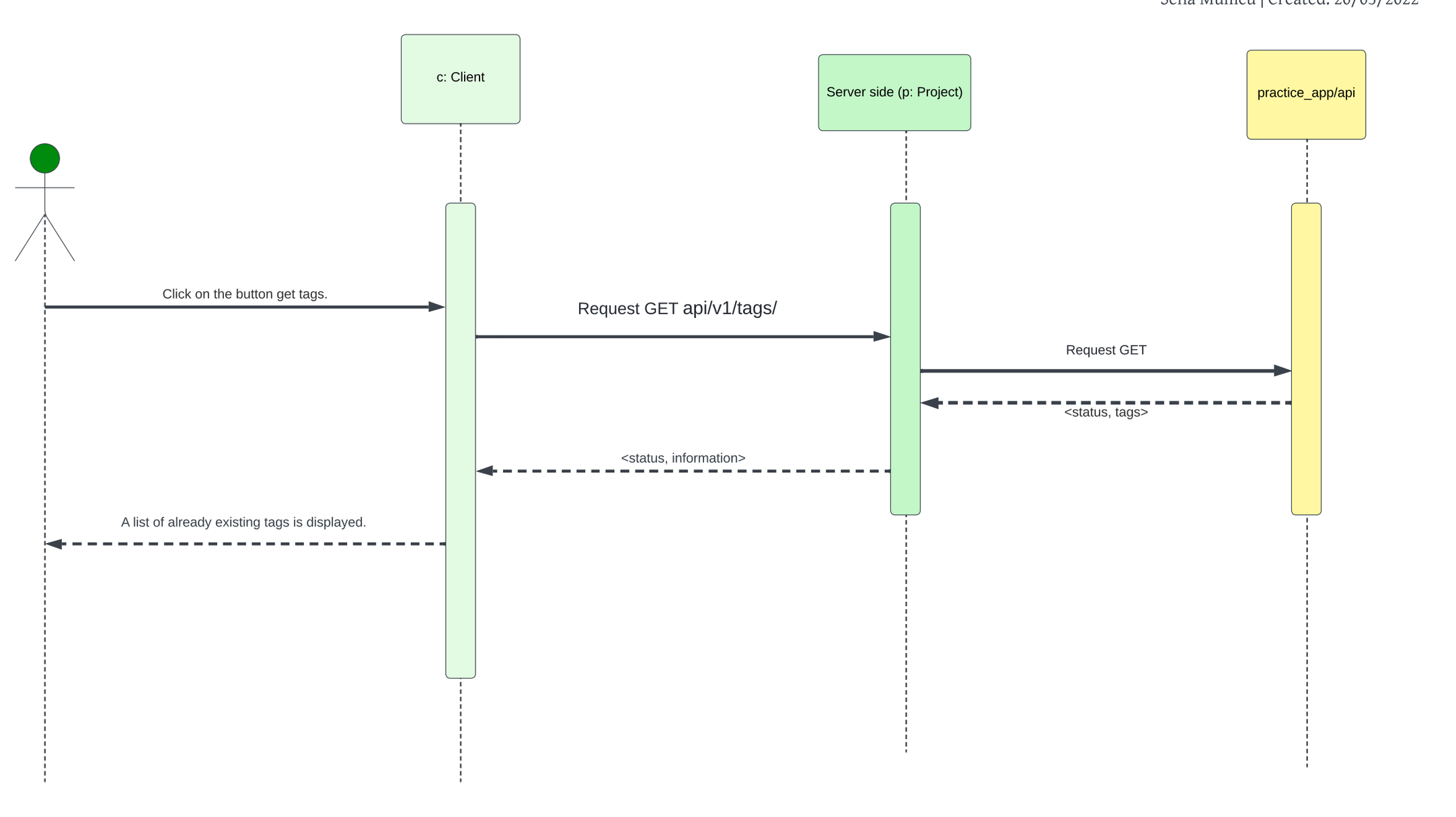
Sequence Diagram #5: Get user with an username

Serhat Hebun Şimşek | Created: 20/05/2022



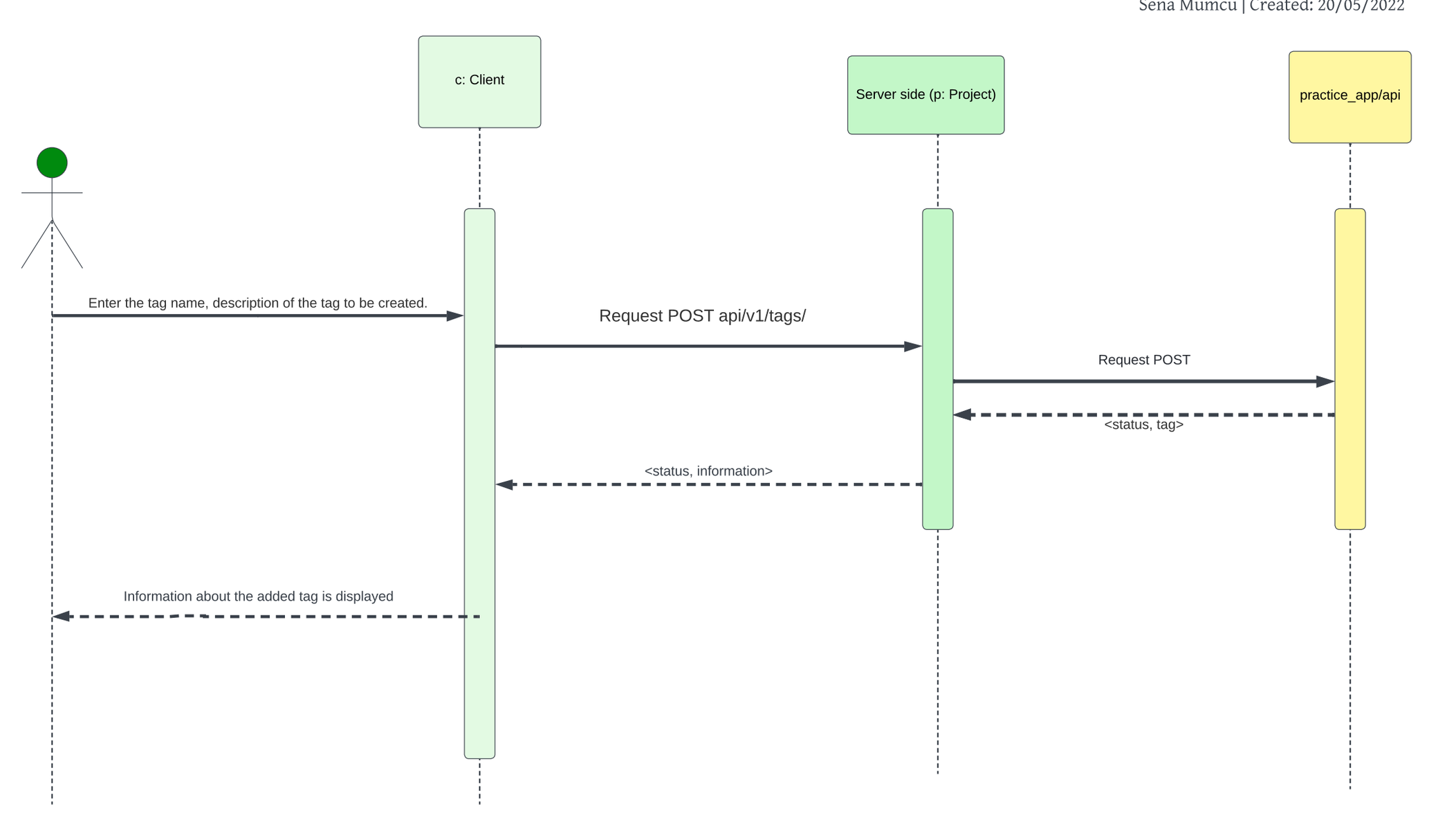
Sequence Diagram #6: List all of the available tags

Sena Mumcu | Created: 20/05/2022



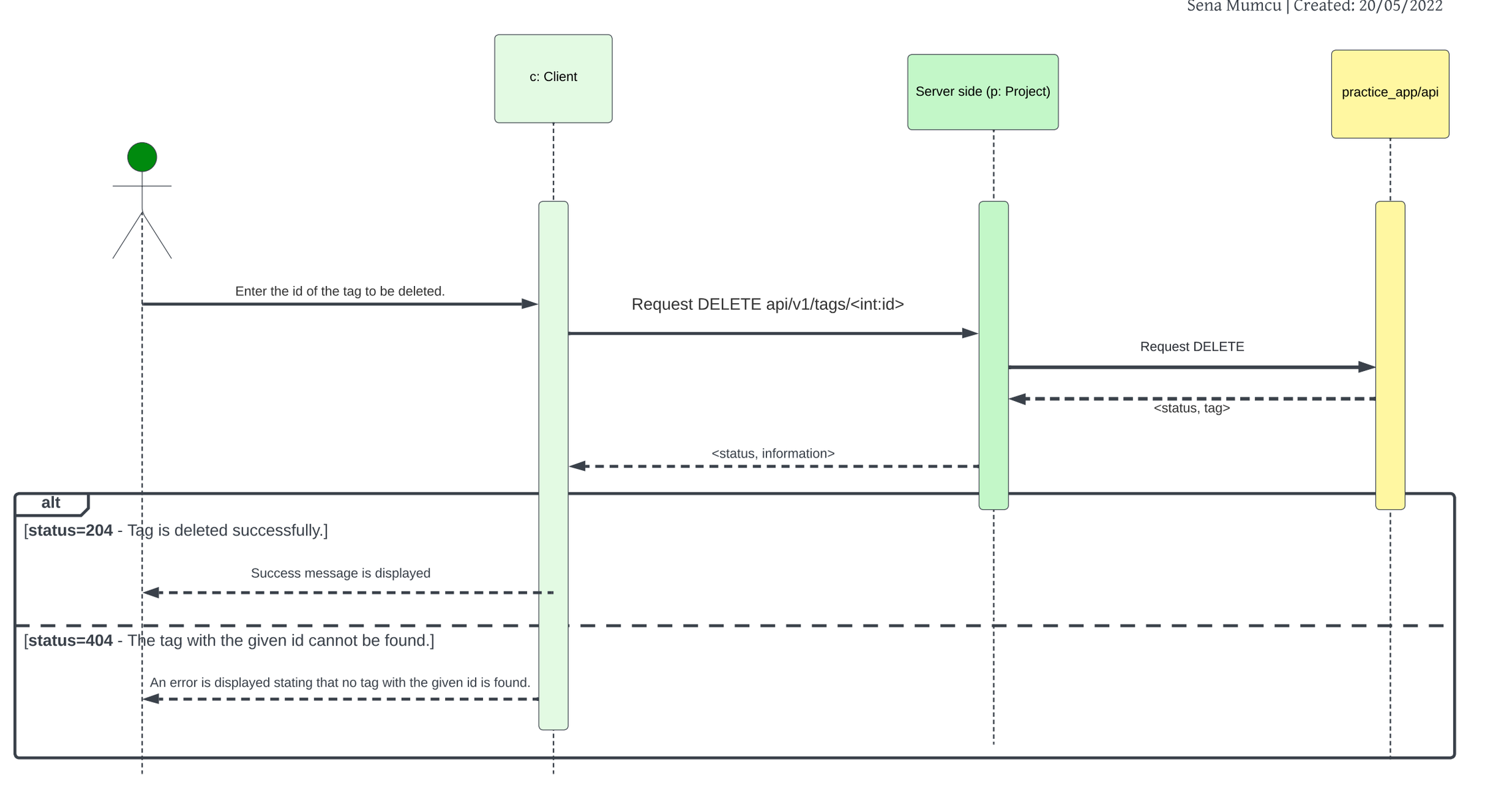
Sequence Diagram #7: Create a new tag

Sena Mumcu | Created: 20/05/2022



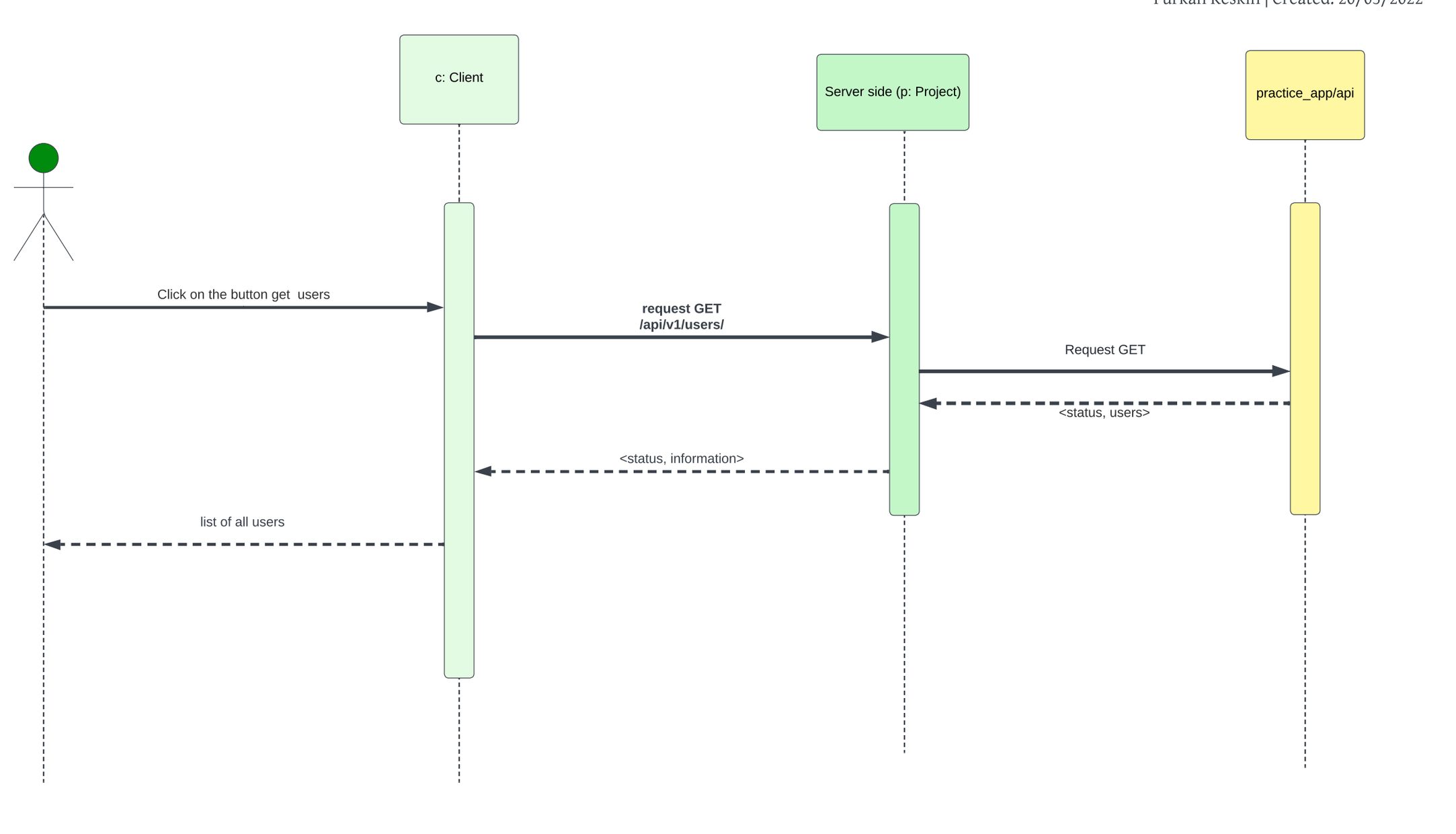
Sequence Diagram #8: Deleting an existing tag

Sena Mumcu | Created: 20/05/2022



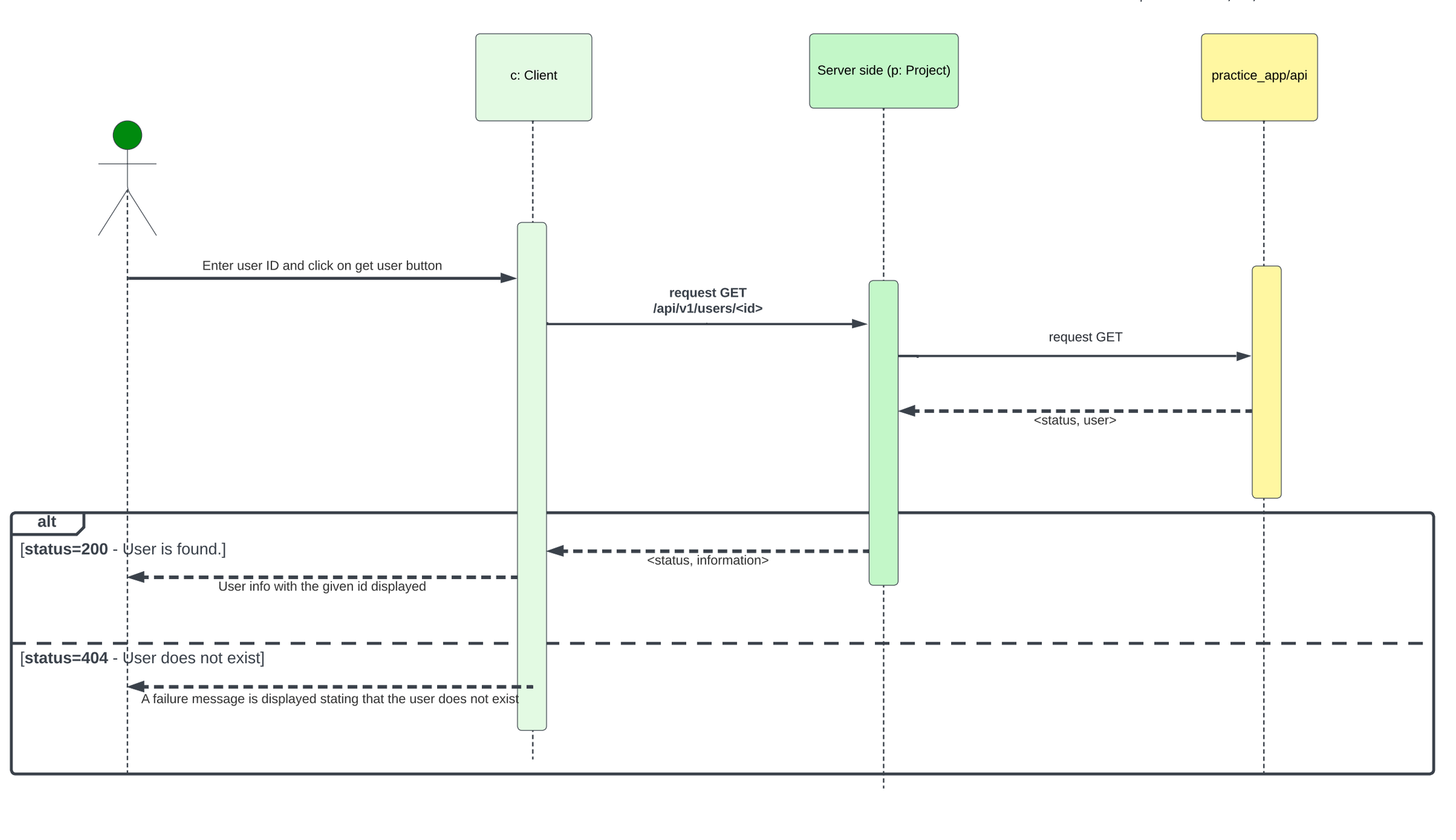
Sequence Diagram #9: List all users

Furkan Keskin | Created: 20/05/2022



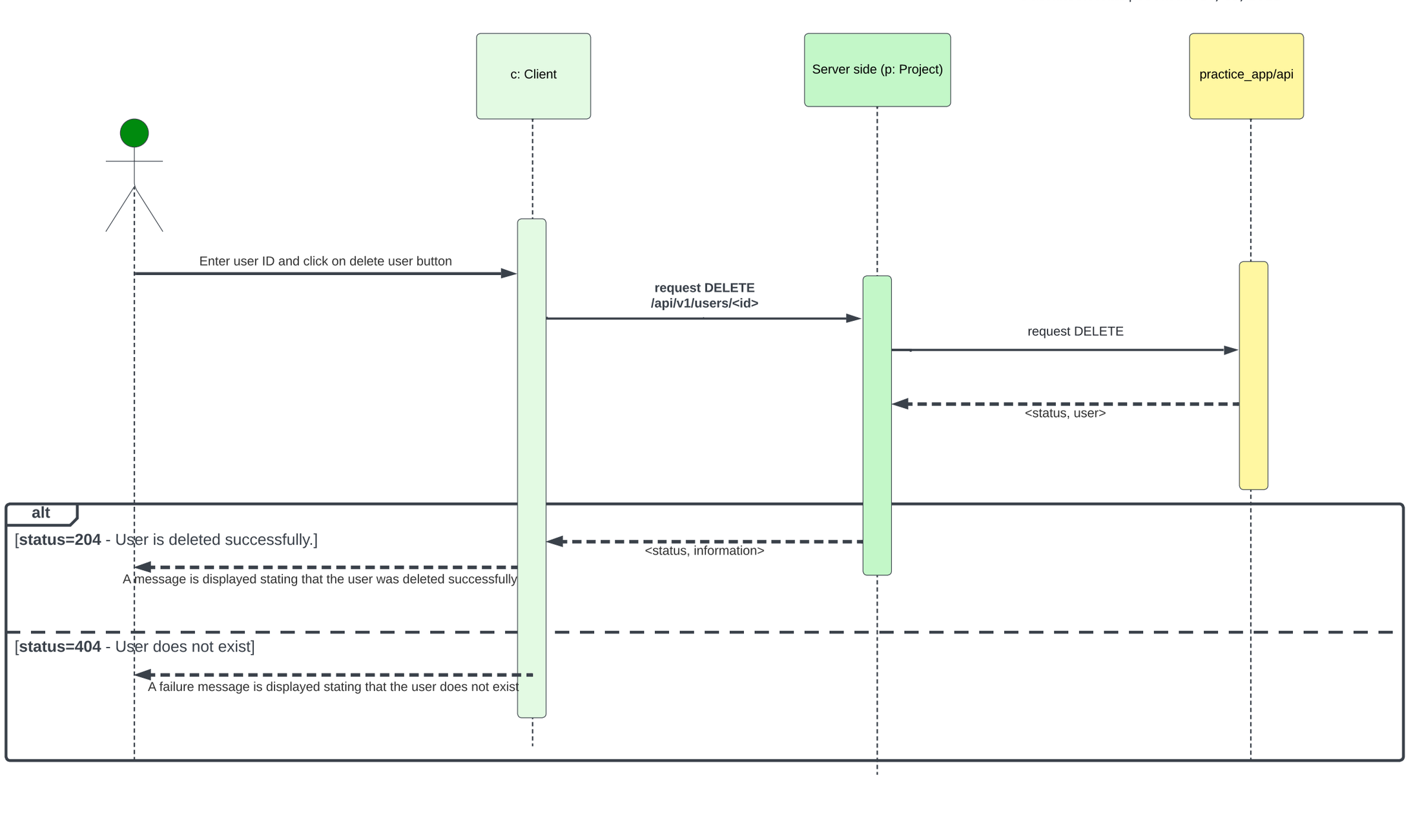
Sequence Diagram #10: Get user with user ID

Furkan Keskin | Created: 20/05/2022



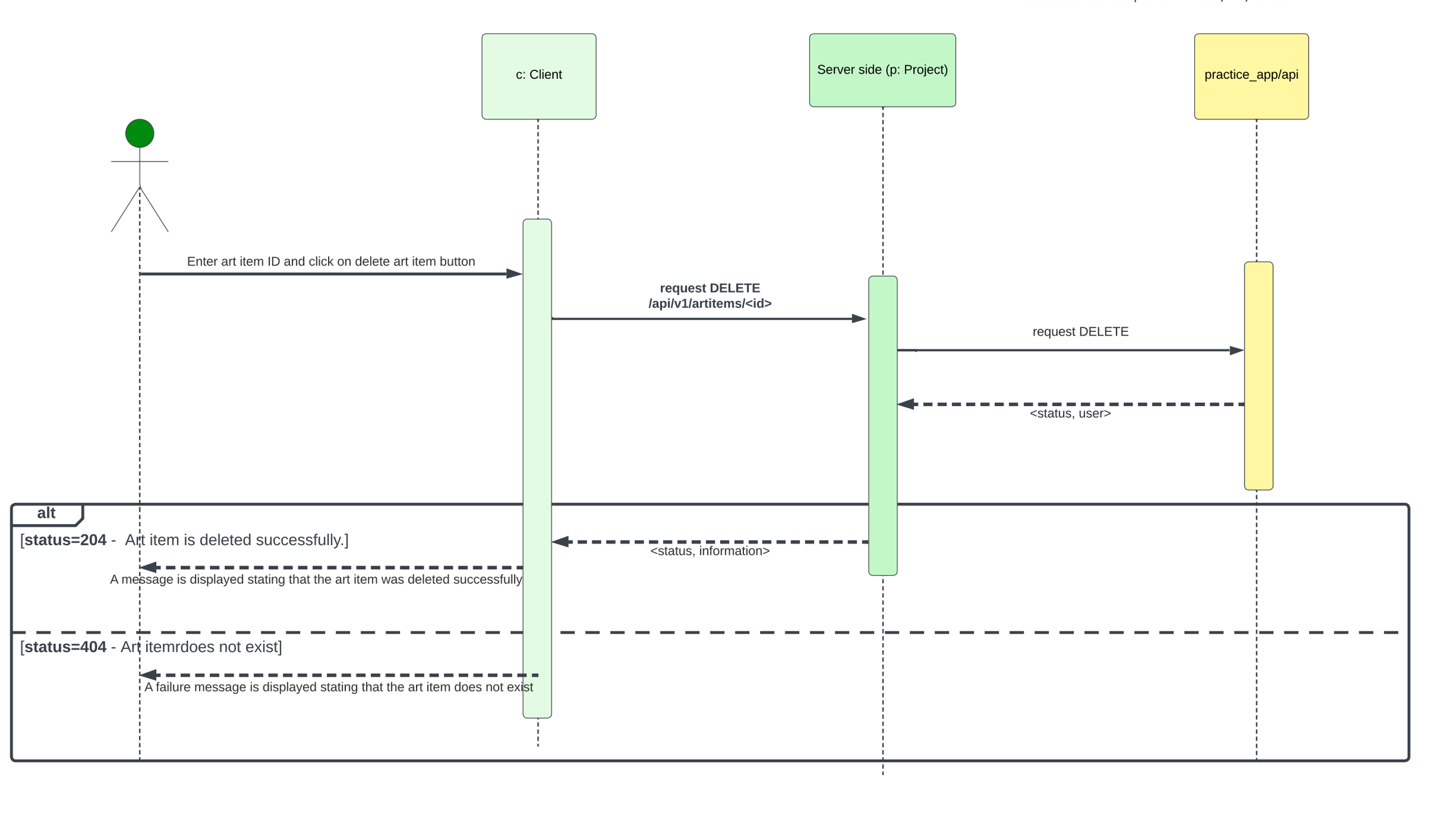
Sequence Diagram #11: Delete user by user ID

Furkan Keskin | Created: 20/05/2022



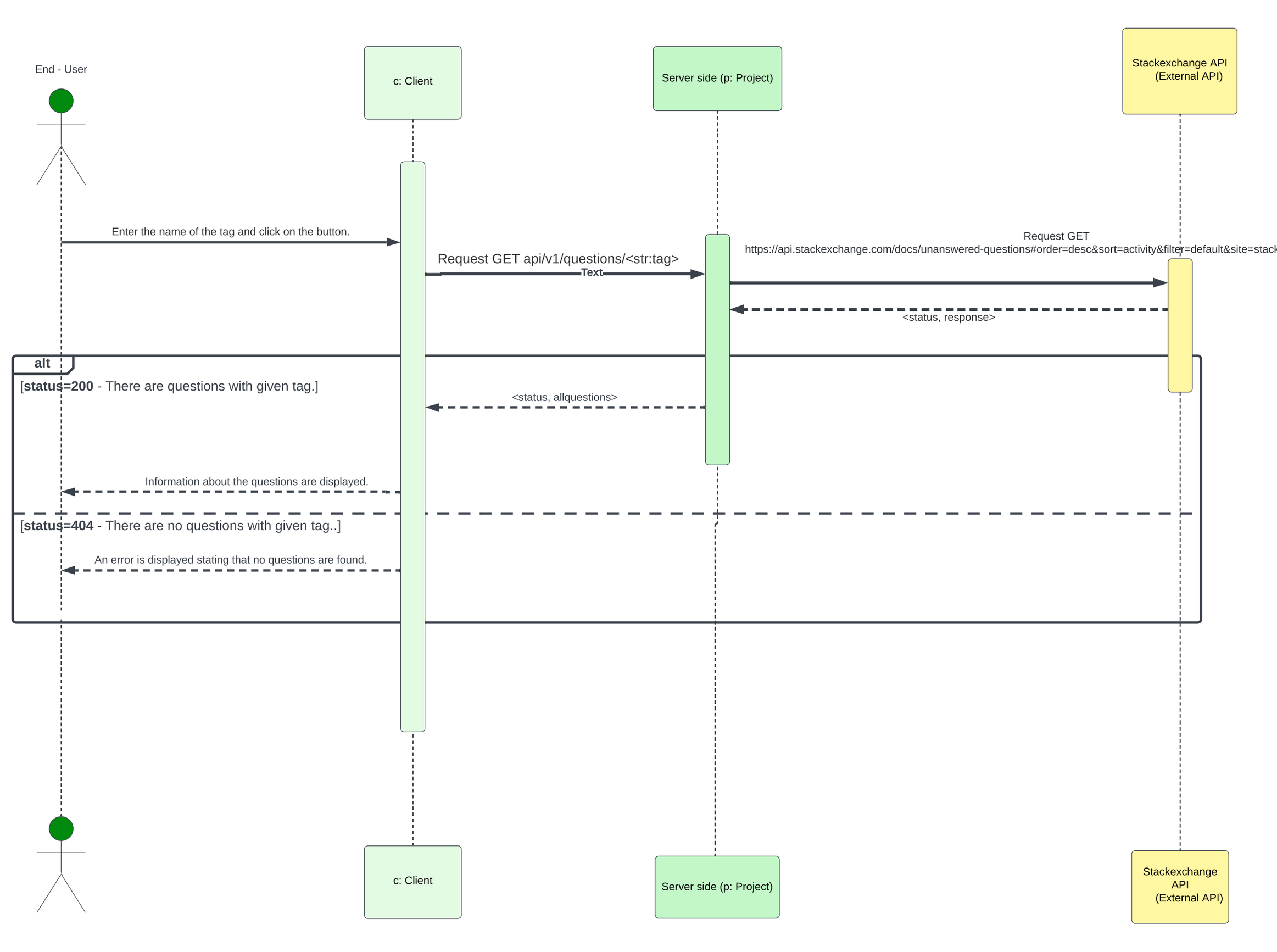
Sequence Diagram #12: Delete art item by art item ID

Furkan Keskin | Created: 20/05/2022



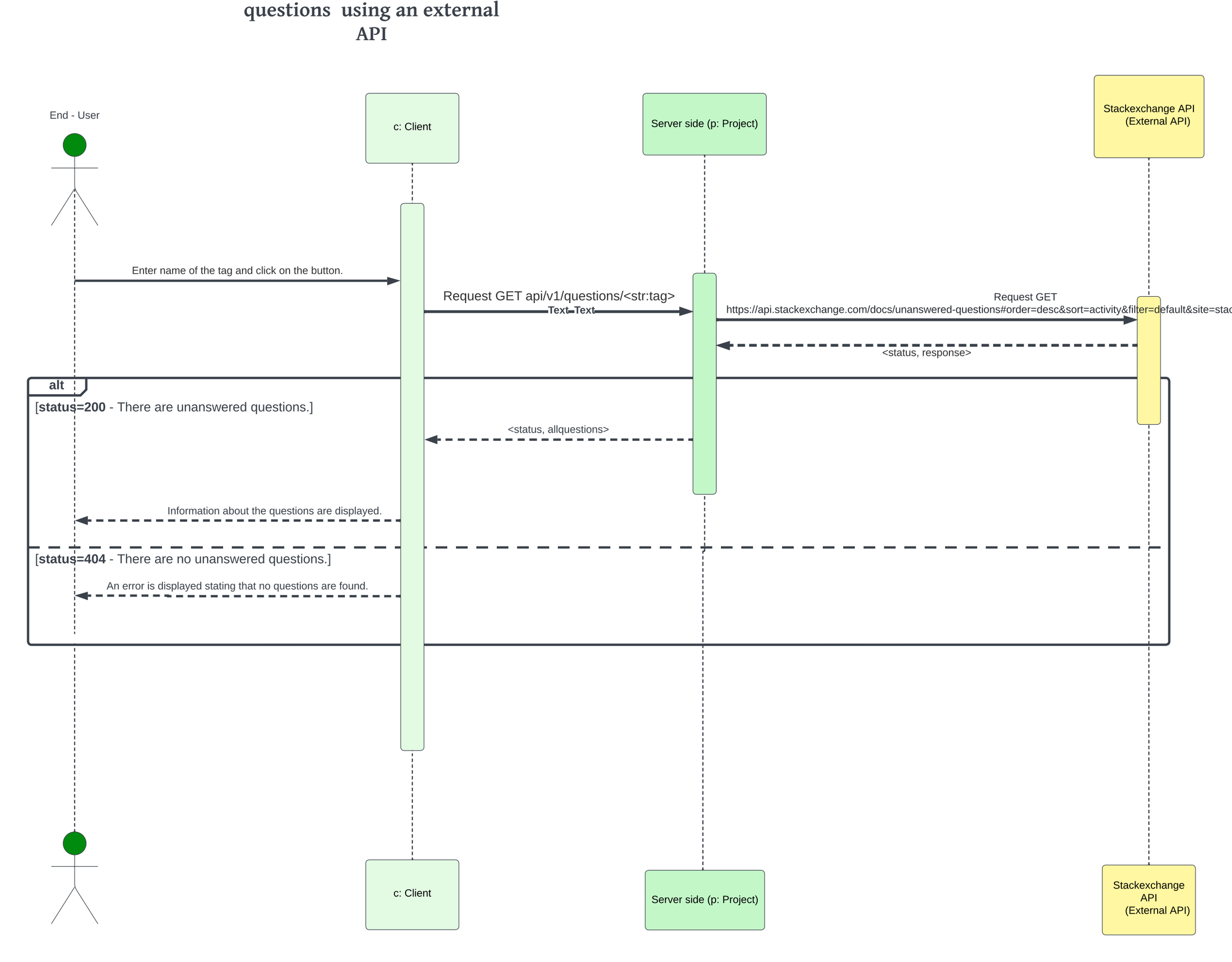
Sequence Diagram #13: Getting a list of unanswered stackexchange questions with given tag using an external API

Elif Bayraktar | Created: 20/05/2022



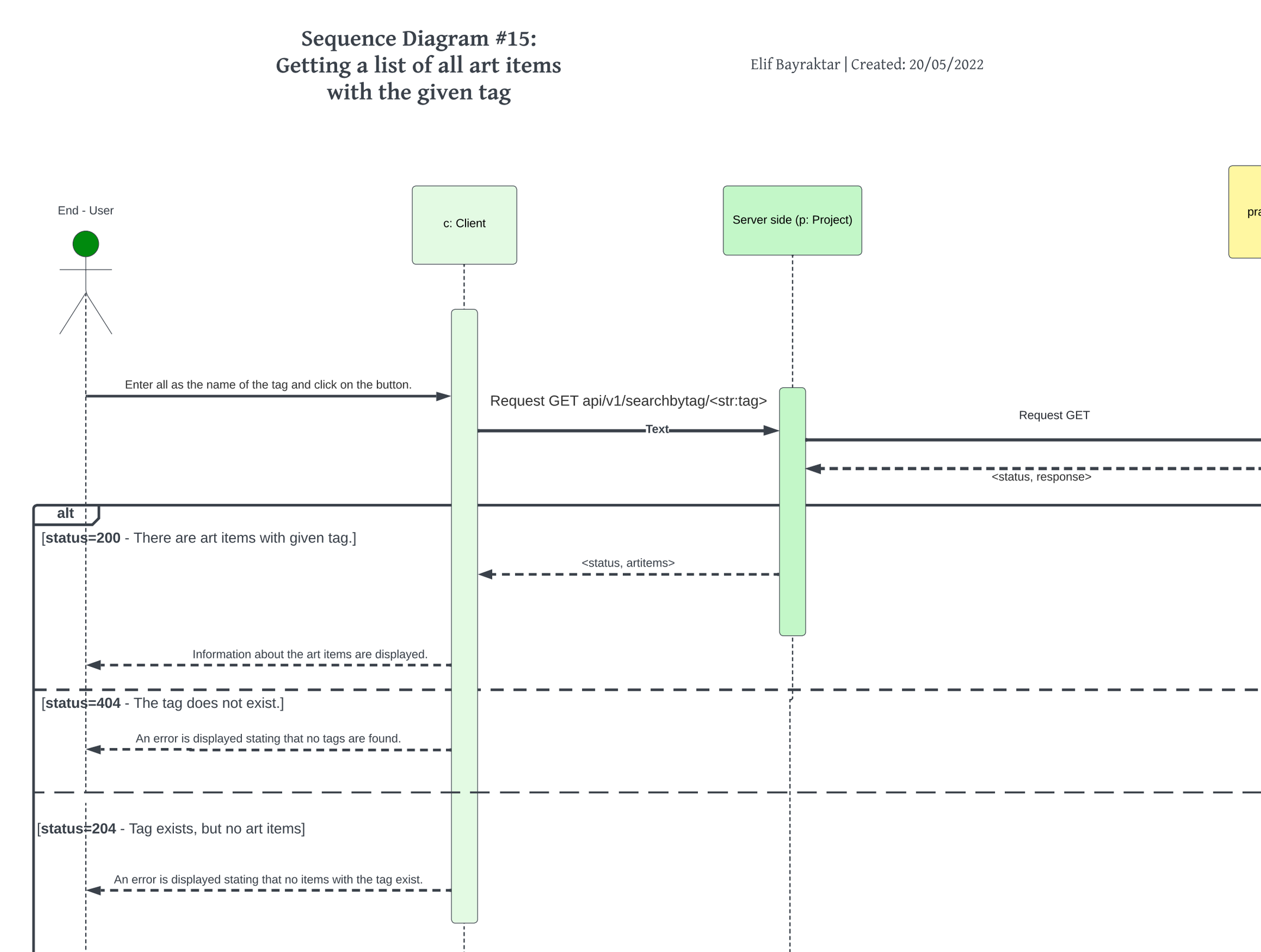
Sequence Diagram #14: Getting a list of all unanswered stackexchange questions using an external API

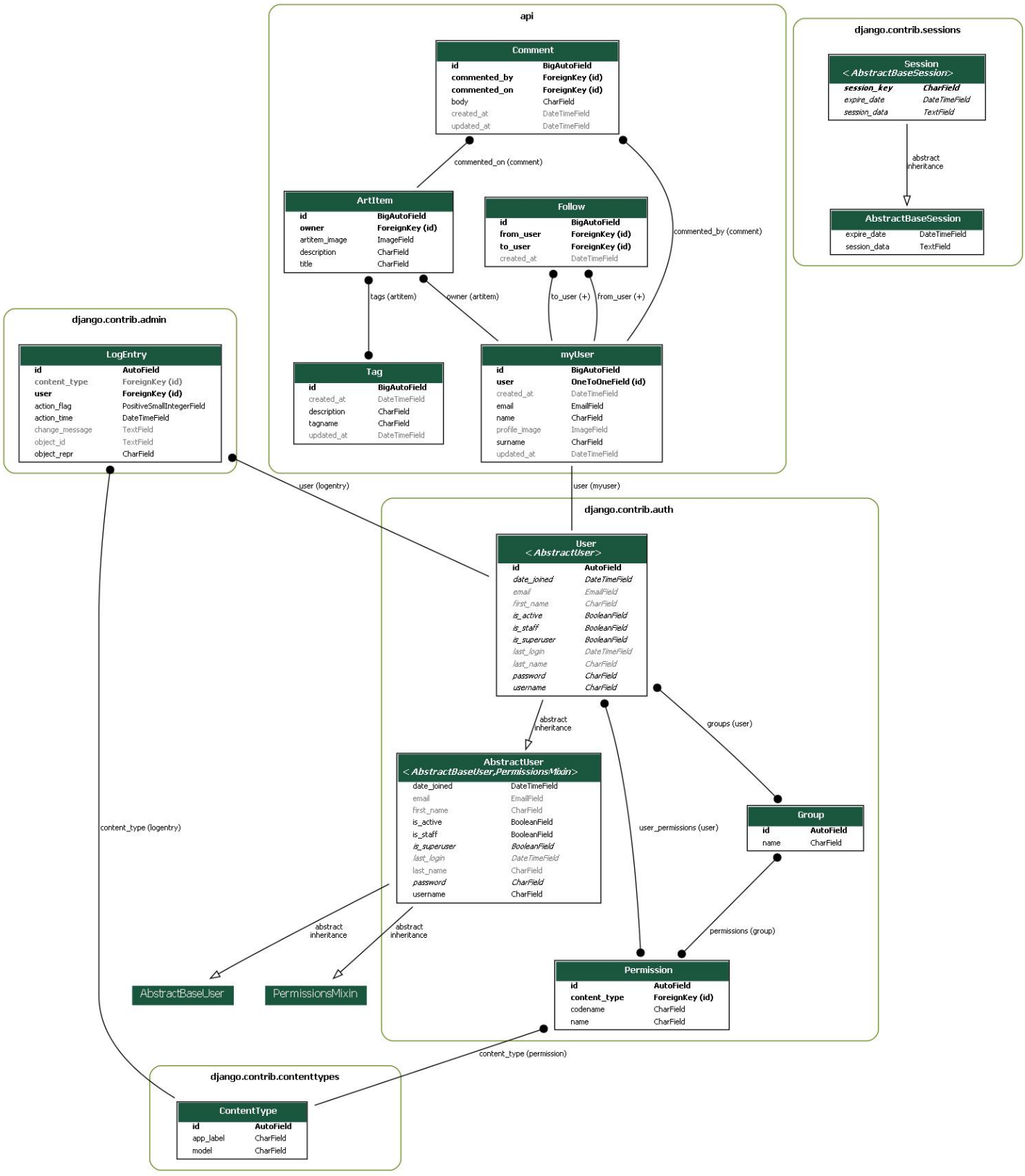
Elif Bayraktar | Created: 20/05/2022



Sequence Diagram #15: Getting a list of all art items with the given tag

Elif Bayraktar | Created: 20/05/2022




api
Comment
id BigAutoField
commented_by ForeignKey (id)
commented_on ForeignKey (id)
body CharField
created_at DateTimeField
updated_at DateTimeField
commented_on (comment)
commented_by (comment)
ArtItem
id BigAutoField
owner ForeignKey (id)
artitem_image ImageField
description CharField
title CharField
Follow
id BigAutoField
from_user ForeignKey (id)
to_user ForeignKey (id)
created_at DateTimeField
tags (artitem)
owner (artitem)
to_user (+)
from_user (+)
Tag
id BigAutoField
created_at DateTimeField
description CharField
tagname CharField
updated_at DateTimeField
myUser
id BigAutoField
user OneToOneField (id)
created_at DateTimeField
email EmailField
name CharField
profile_image ImageField
surname CharField
updated_at DateTimeField
django.contrib.sessions
Session
<AbstractBaseSession>
session_key CharField
expire_date DateTimeField
session_data TextField
abstract inheritance
AbstractBaseSession
expire_date DateTimeField
session_data TextField
django.contrib.admin
LogEntry
id AutoField
content_type ForeignKey (id)
user ForeignKey (id)
action_flag PositiveSmallIntegerField
action_time DateTimeField
change_message TextField
object_id TextField
object_repr CharField
user (logentry)
user (myuser)
django.contrib.auth
User
<AbstractUser>
id AutoField
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField
abstract inheritance
groups (user)
AbstractUser
<AbstractBaseUser,PermissionsMixin>
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField
Group
id AutoField
name CharField
content_type (logentry)
user_permissions (user)
abstract inheritance
abstract inheritance
permissions (group)
AbstractBaseUser
PermissionsMixin
Permission
id AutoField
content_type ForeignKey (id)
codename CharField
name CharField
content_type (permission)
django.contrib.contenttypes
ContentType
id AutoField
app_label CharField
model CharField

DBMS ER diagram (UML notation)

Sena Mumcu | May 20, 2022
Serhat Hebun Şimşek | Elif Bayraktar | Karahan Sarıtaş | Sinem Koçoğlu