Design

1. Resue Plan
2. Service
3. Unit Test
4. Design Diagram
    a. Development Environment
    b. Database Design Diagram
    c. Design Sequence Diagram
    d. CI/CD Pipeline
    e. Use Case Diagram
    f. Layered Architecture Diagram

Reuse Plan

Component	Version	Notes	Documentation
React	16.13.1	Front end framework	https://reactjs.org/
Node	12.18.3 LTS	Download Node via nvm (node version manager) in the azure server, specify the node version as 12.18.3	https://nodejs.org/en/
React Spring	`8.0.5`	Cover our UI related animations needs.	https://www.react-spring.io/docs
MaterialUI	4.11.0	React component for faster and easier web development	https://material-ui.com/
Spring Boot	2.3.4	Powerful and production-ready framework	https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
Travis CI	N/A	Build CI/CD pipeline	https://docs.travis-ci.com/
Docker	19.03.12	Build a docker image for each commit	https://docs.docker.com/
watchtower	latest	Pull and update docker images automatically	https://docs.watchtower.com/
docker-compose	1.17.1	Manage backend service and watchtower	https://docs.docker.com/
Nginx		Used for reverse proxy, and HTTP cache	
Azure	N/A		https://docs.microsoft.com/en-us/azure/?product=featured
Color hunt	N/A	Colour selector	https://colorhunt.co/
Marvel app	N/A	Draw UI diagram	https://marvelapp.com/
Axure	N/A	Tool for drawing UI	https://www.axure.com/
GoodNotes	5.5.4	Hand drawing for UI diagram draft	https://support.goodnotes.com/hc/en-us/categories/360000080575-GoodNotes-5
Springfox	2.9.2	Generate API documentation via Component Scan	https://github.com/springfox/springfox/tree/master/docs
Yandex	N/A	Provide email service for application e.g. no-replty@eportfolio.tech	https://cloud.yandex.com/docs
Azure	N/A	Deploy Docker container to container instance	
GitGuardian	N/A	Automated secrets detection service	
SonarQube	`7.9.4-community`	code inspection to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities	https://sonarqube.eportfolio.tech/ https://hub.docker.com/_/sonarqube

Services

Self-hosted Service

The self-hosted services are hosted by two virtual machines.

Machine	Public IP

A	Accessible via a reverse proxy
B	103.108.228.162

Service Provide by Machine A

Service	Port	Note	Availab le
ssh	22	use public-key authentication	:ok:
NGINX	80 and 443	use `http://dev.eportfolio.tech/` to access the website	:ok:
TiDB Cluster	4000	Use `jdbc:mysql://dev.eportfolio.tech:34000 /eportfolio` as a connection string to access the cluster	:ok:

Service Provide by Machine B

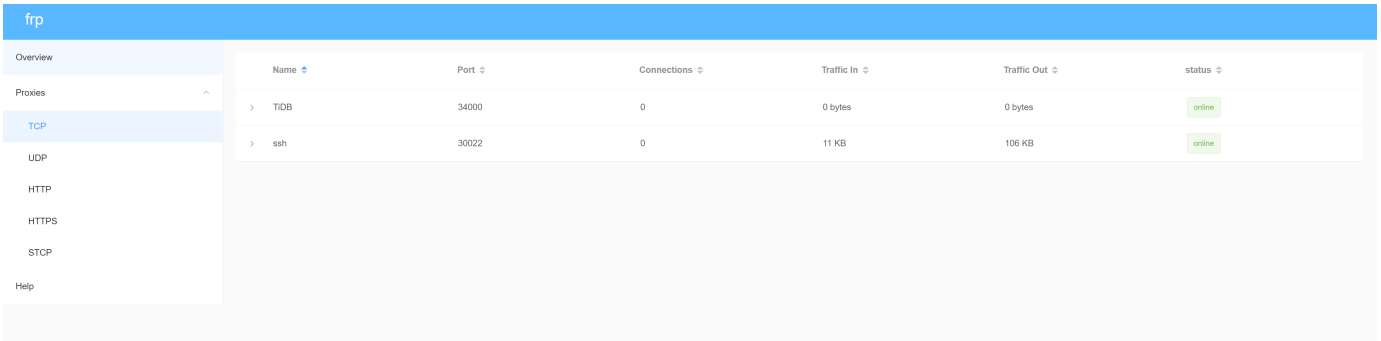
Service	Port	Note	
ssh	22	use `ssh dev.eportfolio.tech -p 30022` to access it	:ok:
backend server on WatchTower	8090	Request to https://api.eportfolio.tech/ will be reverse proxied to this service	:ok:
frp	7000	See dashboard below	:ok:
MongoDB	37017	See	:ok:
Azure Blob Storage Emulator	10000	use `DefaultEndpointsProtocol=http; AccountName=devstoreaccount1; AccountKey=Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT 50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr /KBHBeksoGMGw==;BlobEndpoint=http://haswf.com: 10000/devstoreaccount1;` as connection string	:ok:

:ok:Port Mapping

The following table shows the mapping between the original port and Destination port. This information is recorded to help configure NGINX and frp in the future.

Service	Destination Port	Source Port
SSH	22	30022
HTTP	80	7000
HTTPS	443	7000
TiDB	4000	34000

Dashboard



You can check the health of each reverse proxy on http://frp.eportfolio.tech/. Use the below credential to log in. When a service is available, its status will be online.

Username: comp30022

password: renlord

Other Services

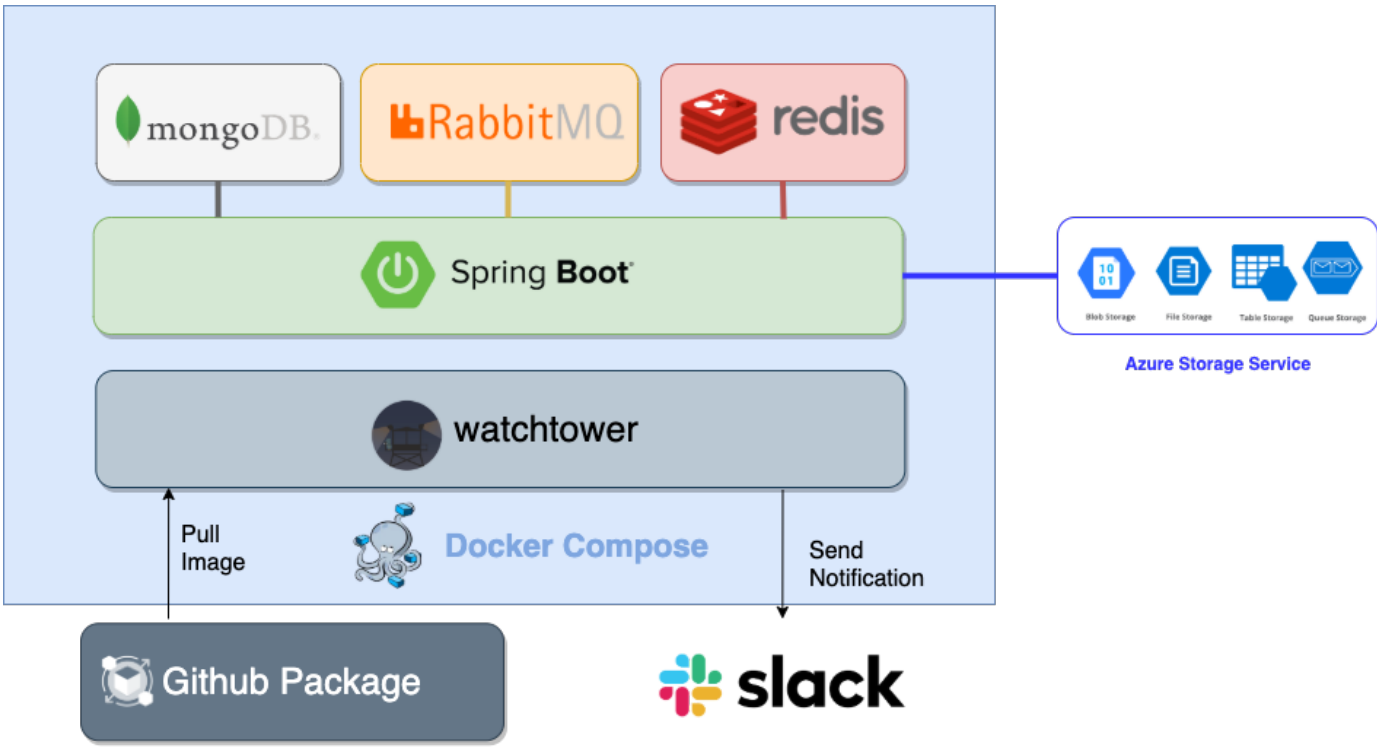
Azure Storage Service

Design Diagram

1. Development Environment
2. Database Design Diagram
3. Design Sequence Diagram
4. CI/CD Pipeline
5. Use Case Diagram
6. Layered Architecture Diagram

Docker Compose Diagram

Overview

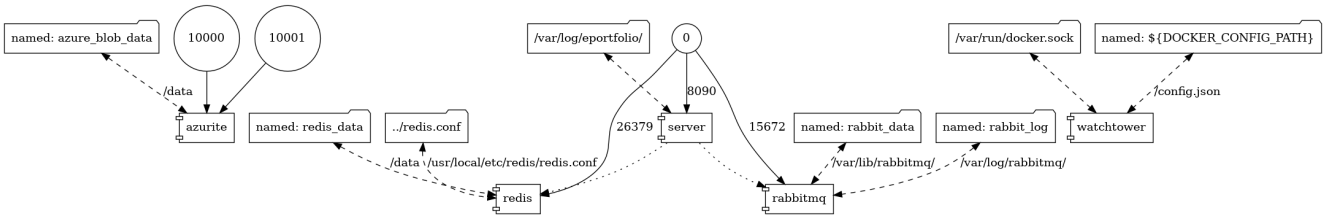


Docker-compose Diagram

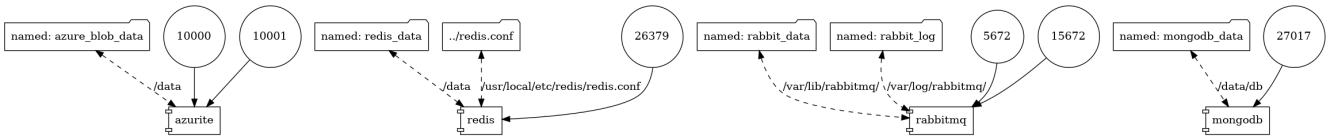
The following diagram illustrates the services defined in docker-compose.yml for the development environment.

- The package symbol represents service.
- The folder symbol stands for volumes, which is the preferred mechanism for persisting data generated by and used by Docker containers.
- The dashed line represents volume mapping between service and volume. For example, volume `redis_data` is mapped into `/data` directory in `redis` container.
- The Filled line illustrates the dependency between services. In the following service, `server` depends on `rabbitmq` and `rediq`. Therefore, `redis` and `rabbitmq` will be initialised before spinning up the `server`.

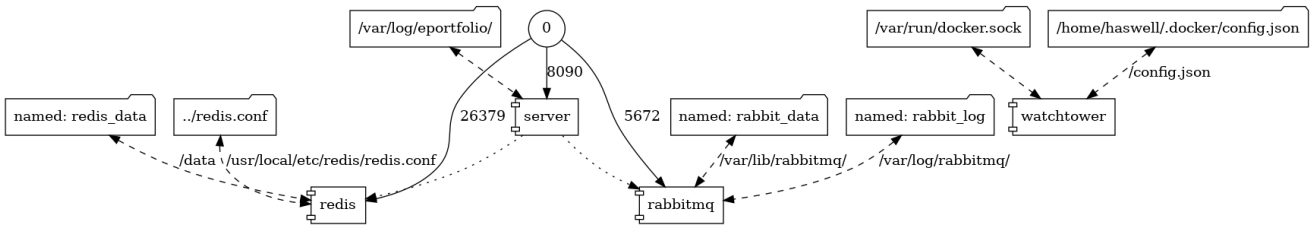
Development



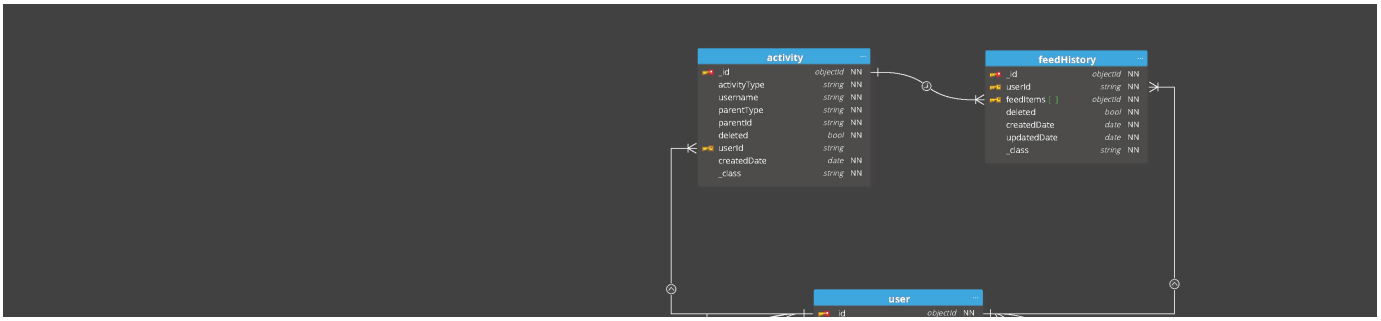
Local

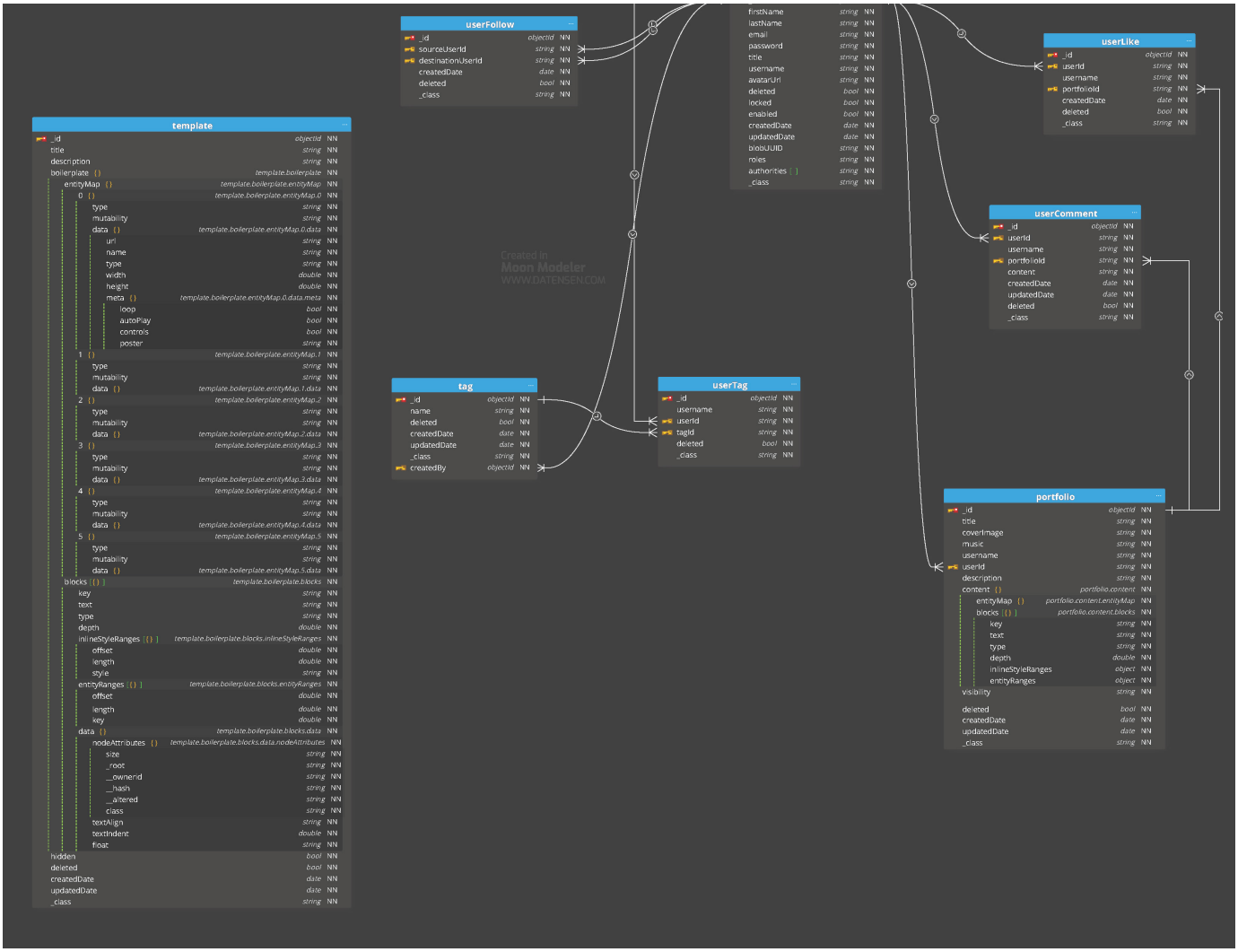


Production

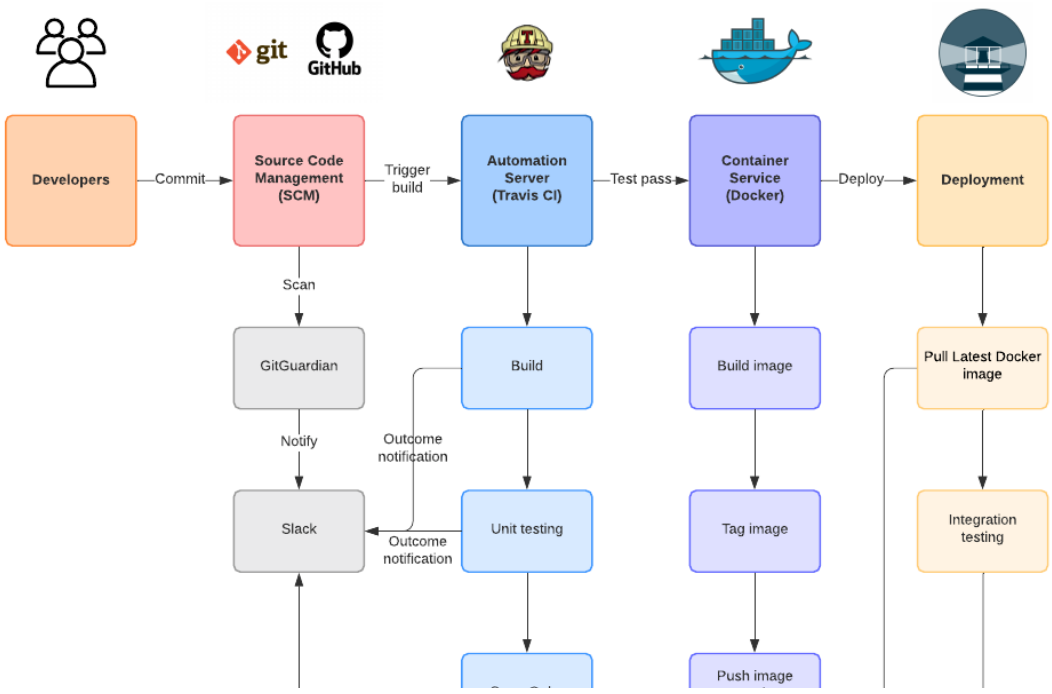


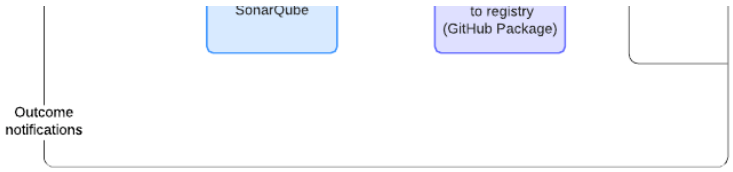
Database Diagram



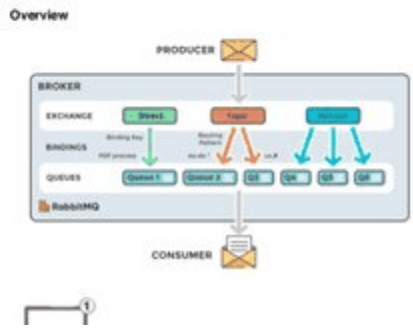


CI/CD Pipeline

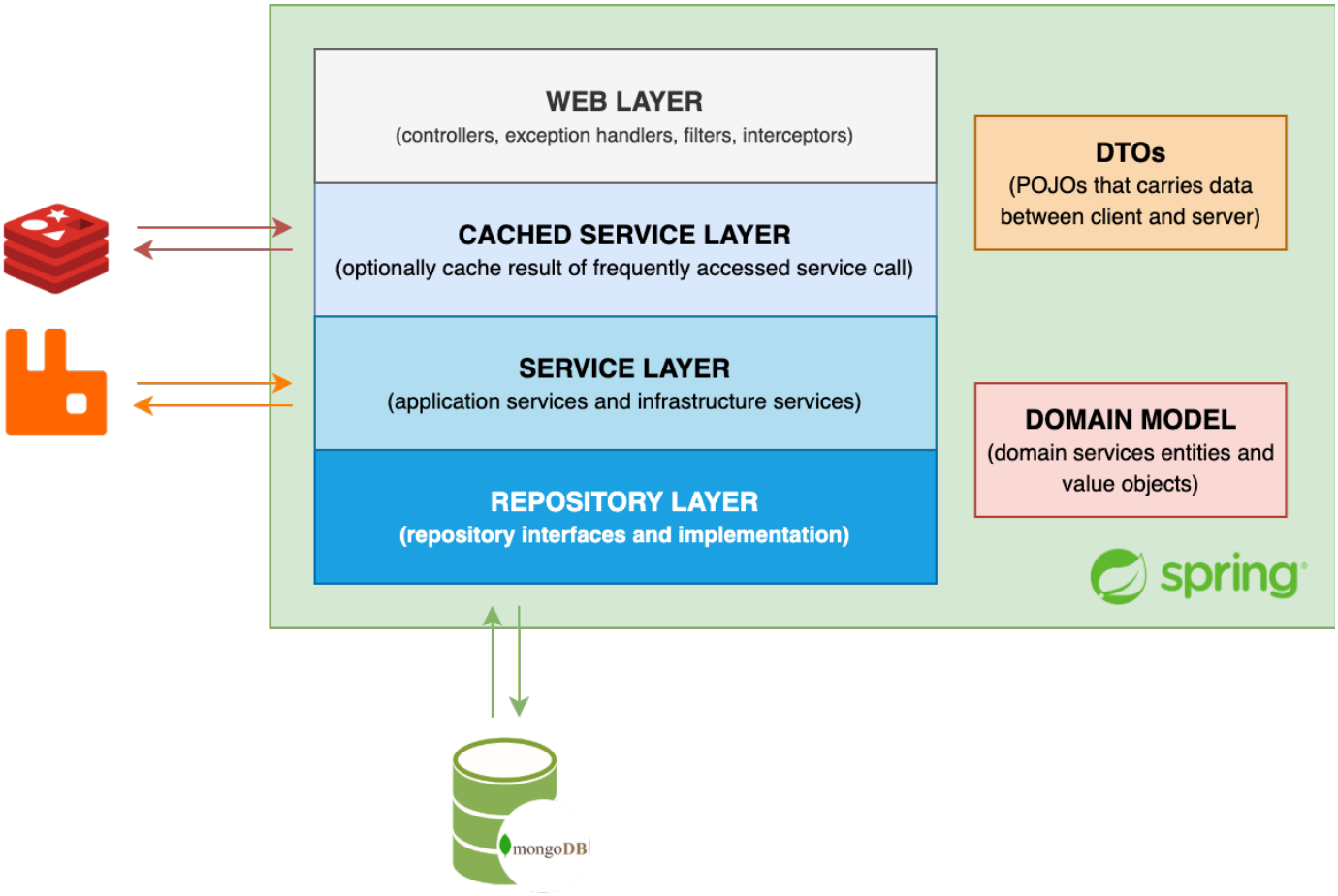




Message Queue Diagram

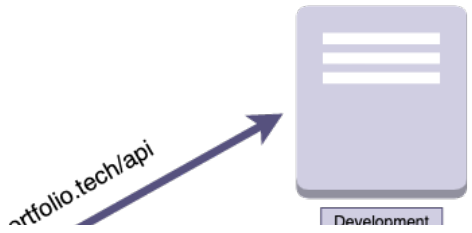


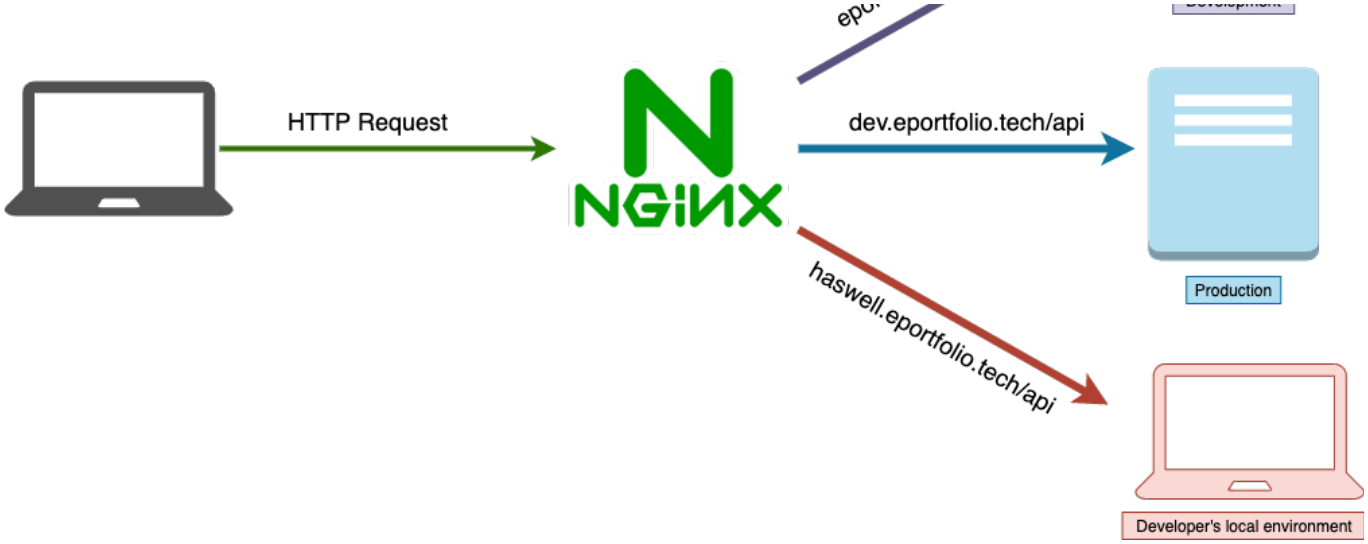
Layered Architecture Diagram



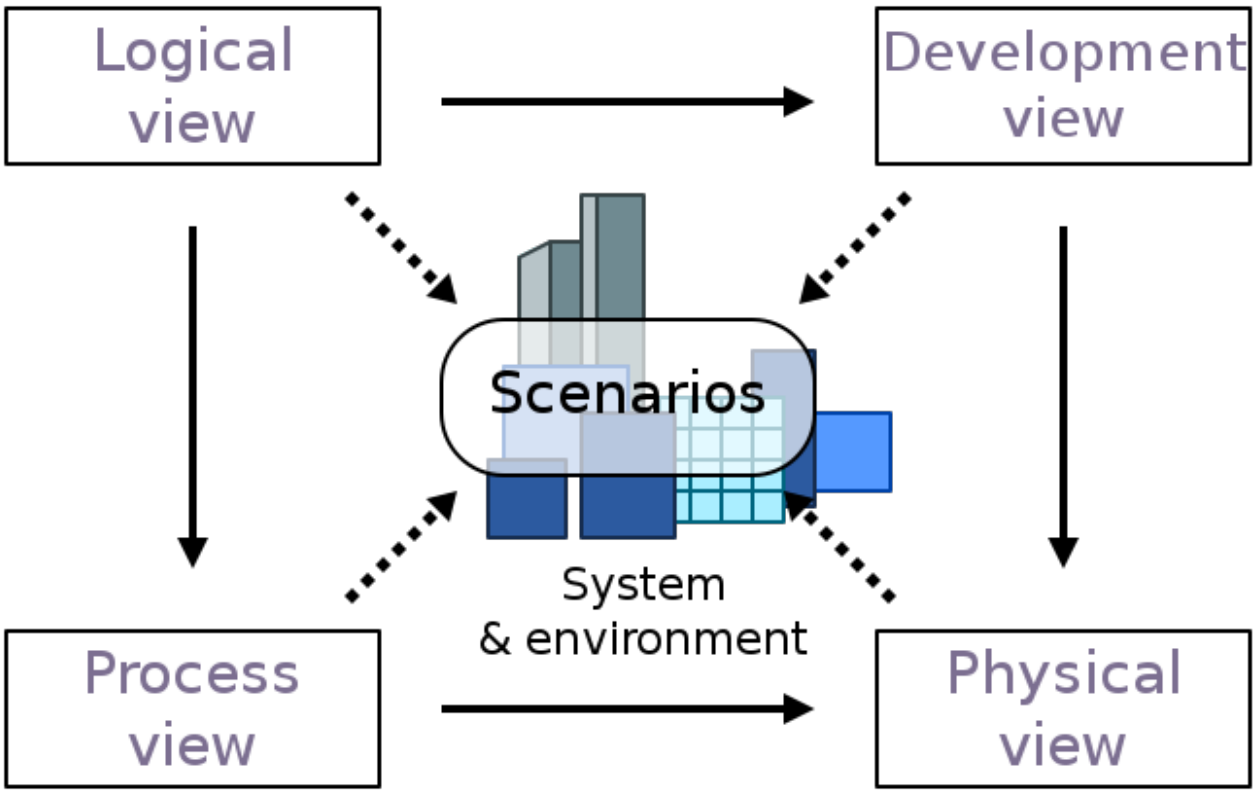
Reverse Proxy

Using reverse proxy for collaboration





4+1 Architectural View Model



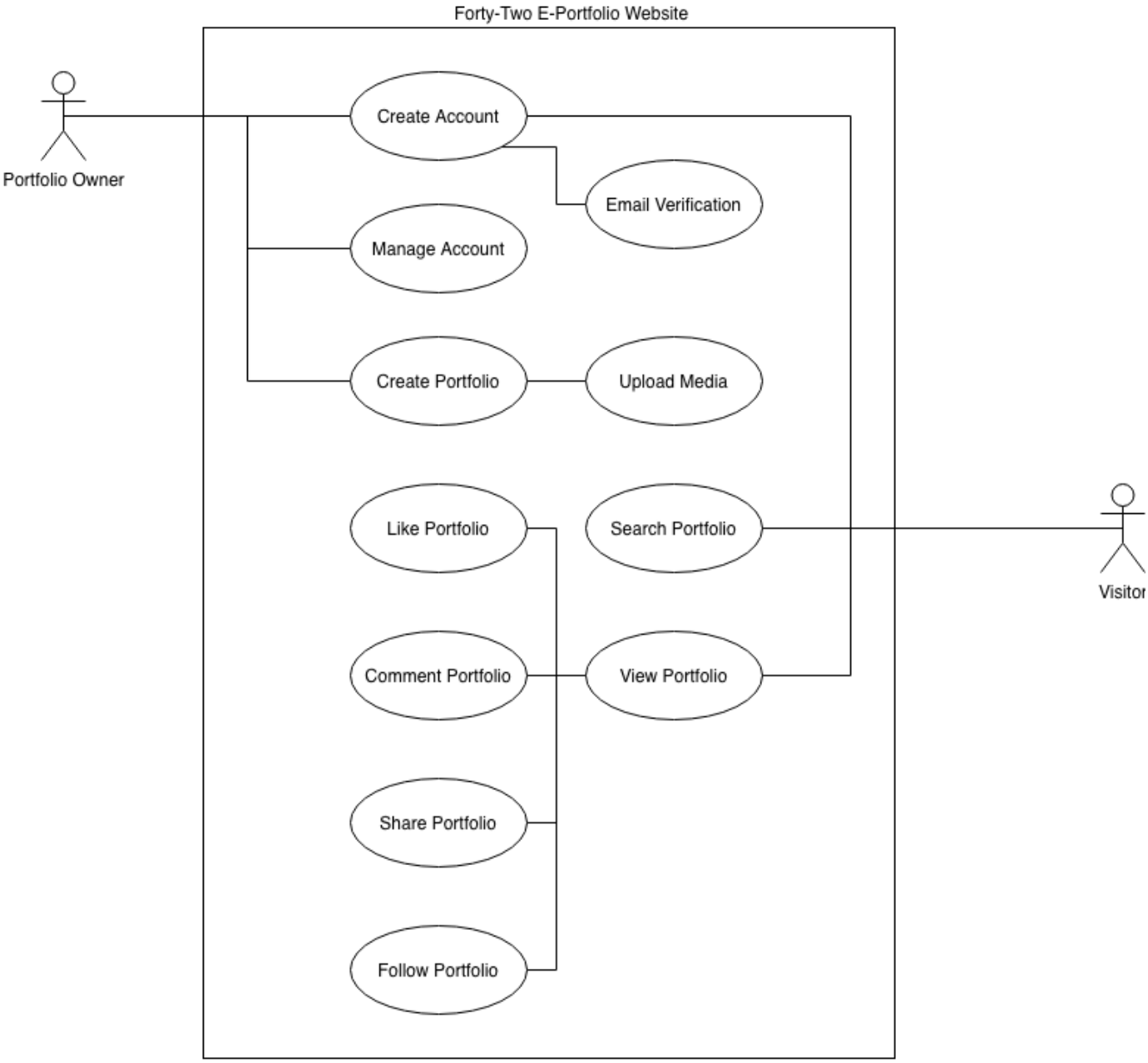
4+1 is a view model used for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views".[1] The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineers, and project managers. The four views of the model are logical, development, process and physical view. In addition, selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence, the model contains 4+1 views:[1]

- ***Logical view***: The logical view is concerned with the functionality that the system provides to end-users. UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams.[2]
- ***Process view***: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and scalability, etc. UML diagrams to represent process view include the sequence diagram, communication diagram, activi ty diagram.[3][2]

- ***Development view***: The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.[2]
- ***Physical view***: The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view. UML diagrams used to represent the physical view include the deployment diagram.[2]
- ***Scenarios***: The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the **use case view**.

Scenarios (Use Case Diagram)

Use Case Diagram

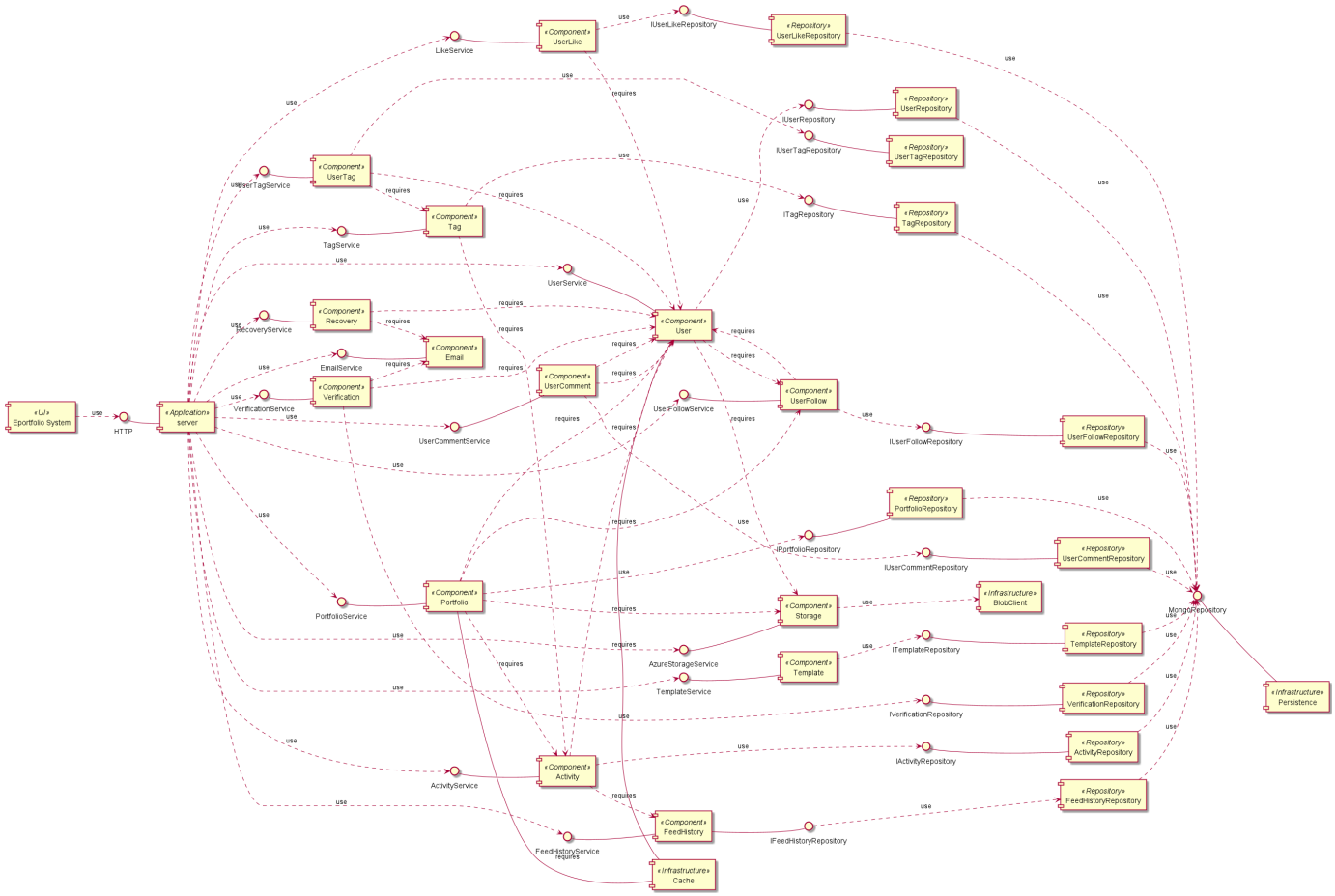


n Forty-Two, a user enters personal information and sets a password to register an account. After the user finishes the email verification, he/she would be unlocked full functions. The user could get recommendations on the explore page. Besides, the user could search specific portfolios by name or tag and view the portfolios (if public). In addition, users can create their own portfolio by a few templates. The user could set his/her

portfolio as private (only registered users can see) or public (everyone can see). The user could upload text, pictures and videos. Moreover, the user could manage the account by updating personal information, tags, resetting passwords and managing following users and checking followers. More importantly, the user could like, comment, share and follow other users' portfolios. After following other users, he/she would get no

Development View

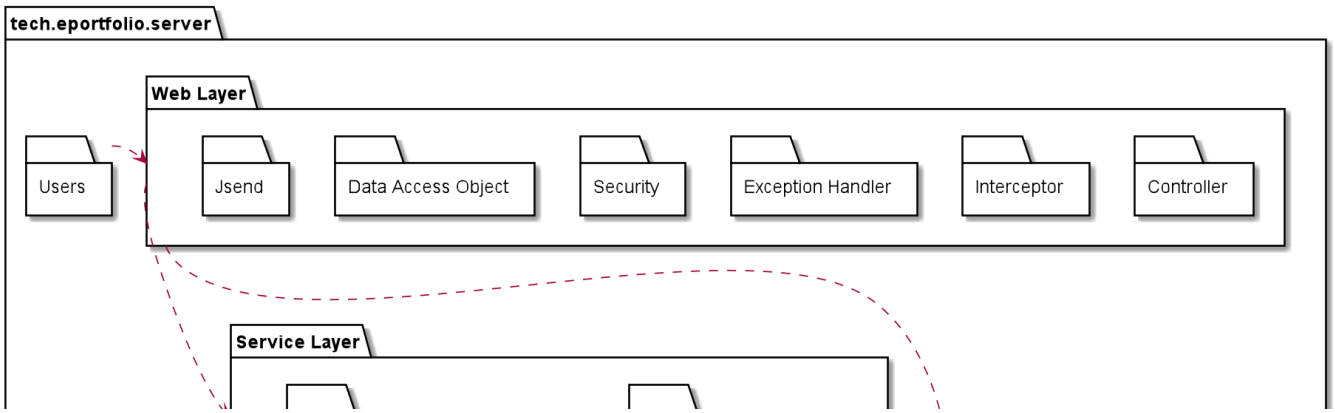
Component Diagram

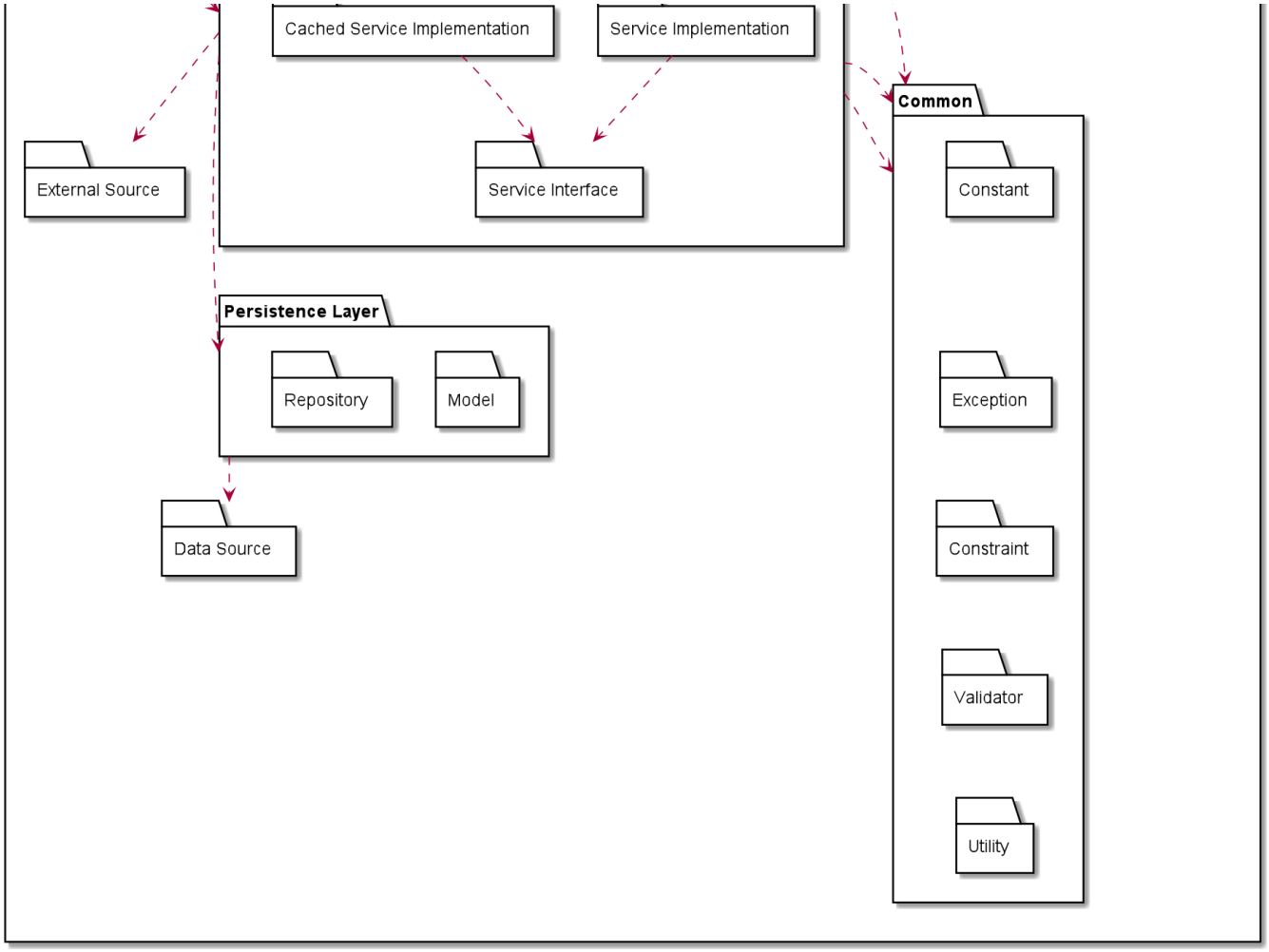


Package Diagram

- A persistence layer has the ability to form and read data from the database and then to pass to the service layer.
- A cached service helps to maximize the program performance while the service implementation has the logic of functionalities.
- A web layer is responsible for controlling, security, and web&bean configuration.
- A common package holds tools for the program to run successfully.

Package Diagram





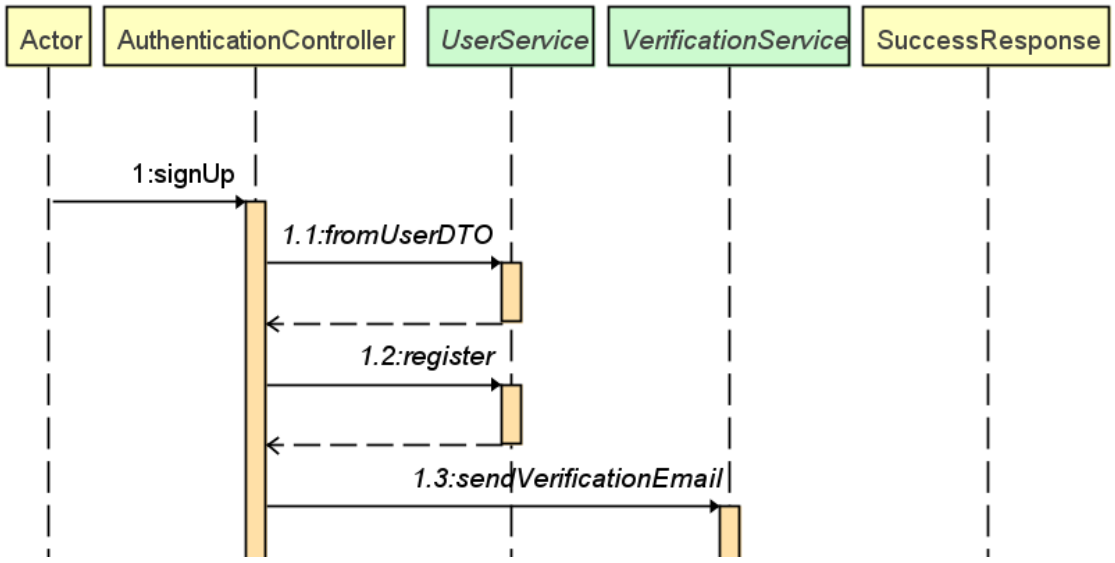
Process View

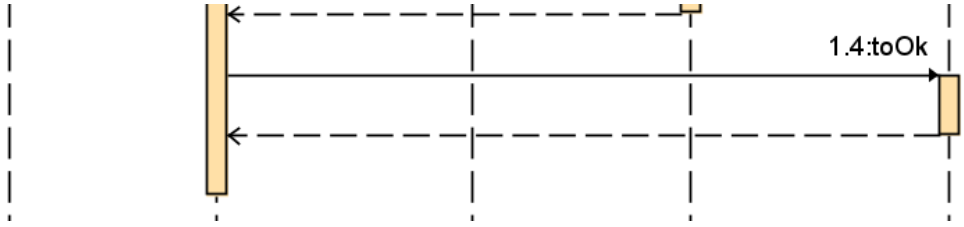
Sequence Diagram

Authentication Controller

Sign Up

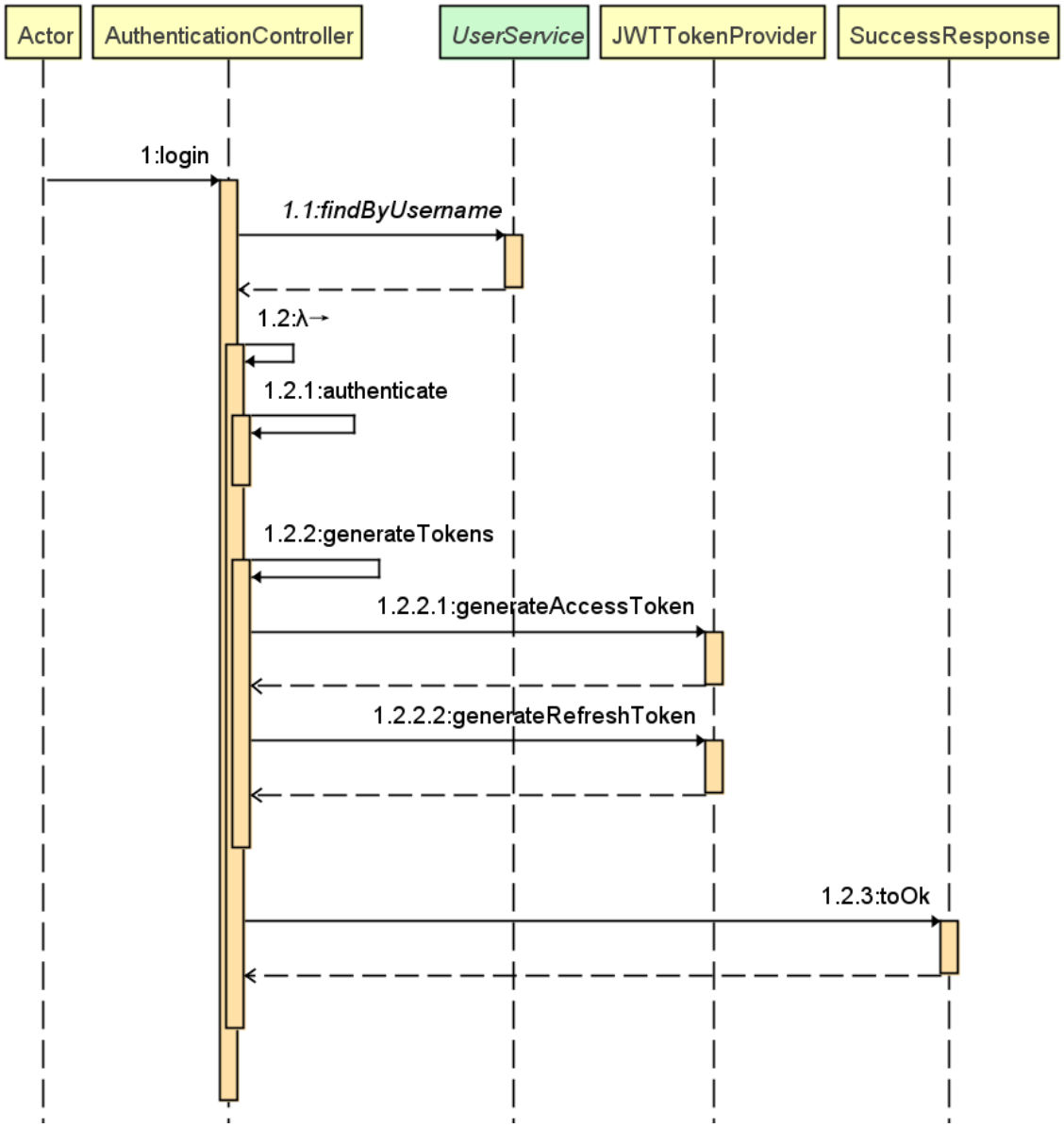
This endpoint helps new user sign up as a registered user with a unique email and some personal details.





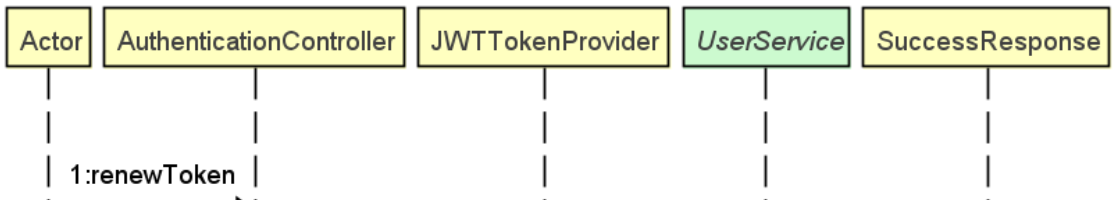
Login

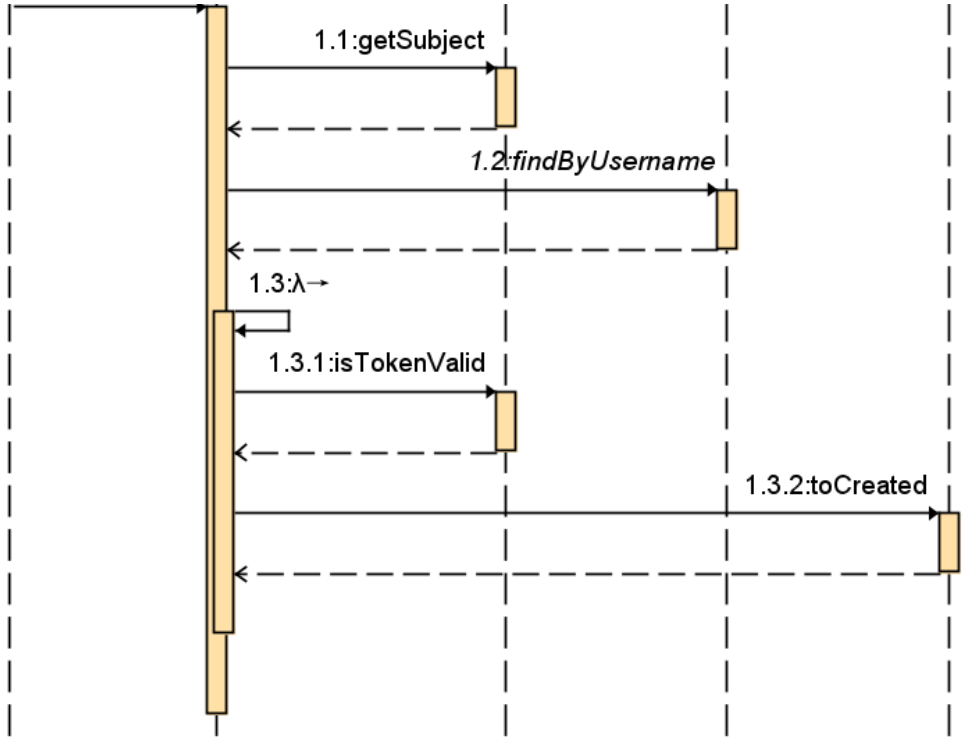
This endpoint helps registered user login with email and password.



Renew Token

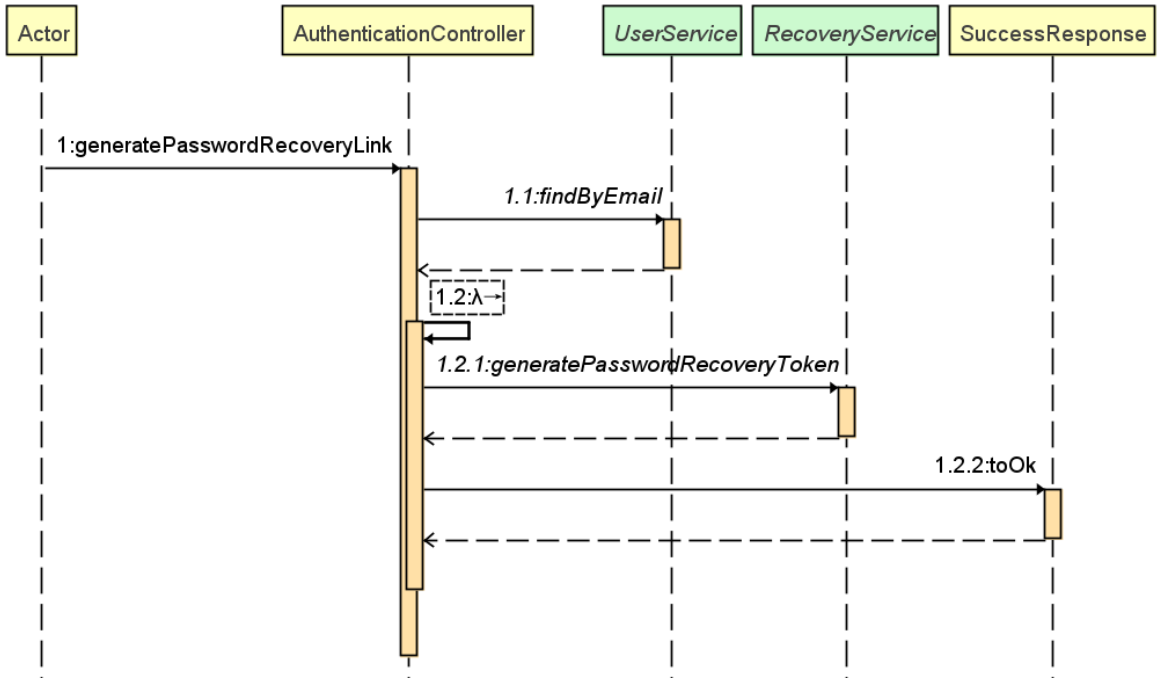
This endpoint helps user update the temporary token that only lasts for 30 minutes.





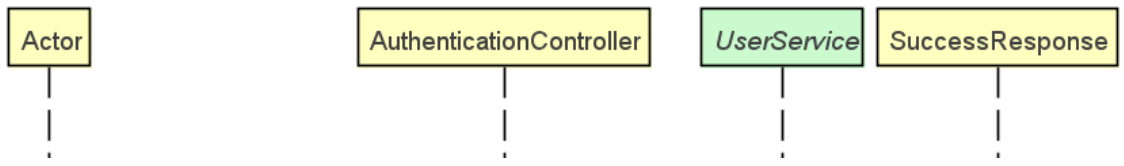
Generate Password Recovery Link

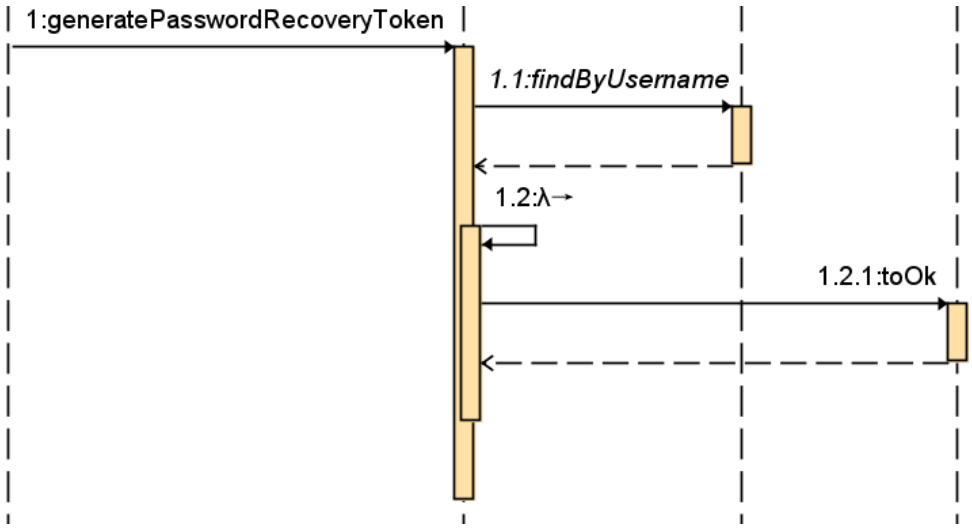
If a user forgot his/her password, this endpoint helps the user to generates a recovery link (with a recovery token) which is used for the frontend to guide the person to reset the password (only for testing).



Generate Password RecoveryToken

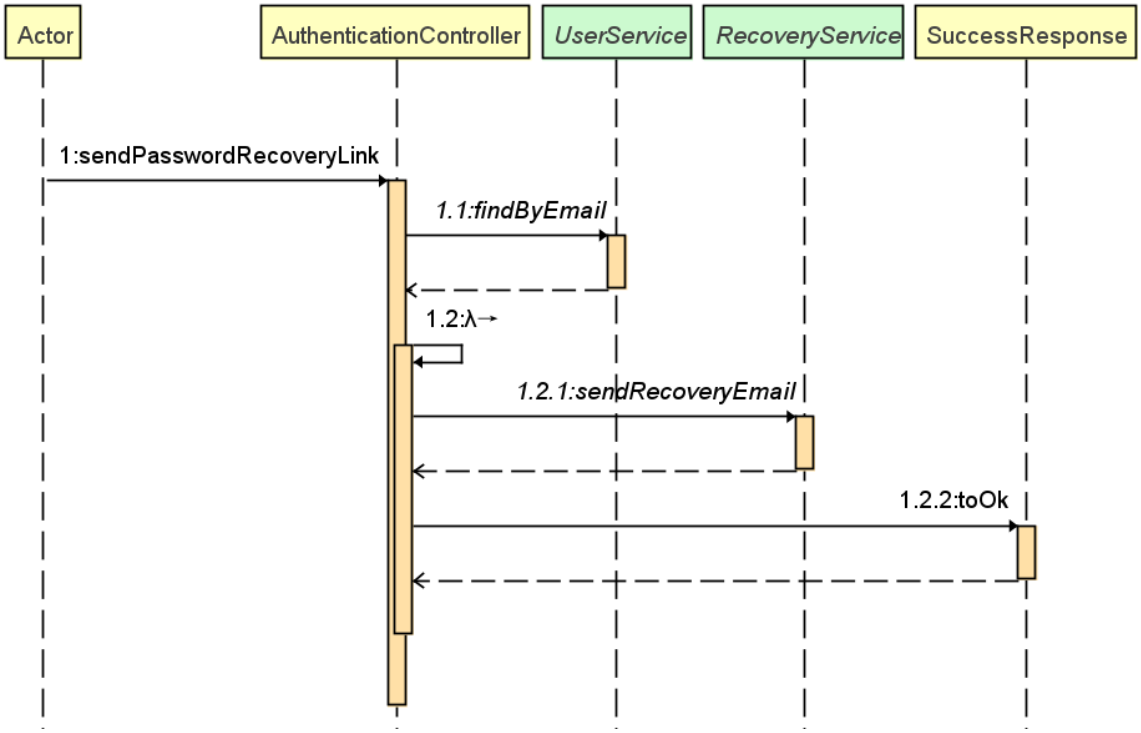
This endpoint helps the user to directly generate recovery token which is used to reset the password (only for testing).





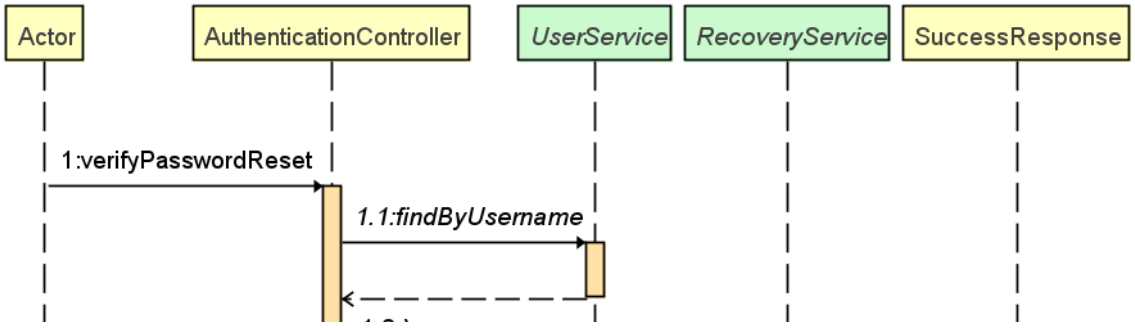
Send Password Recovery Link

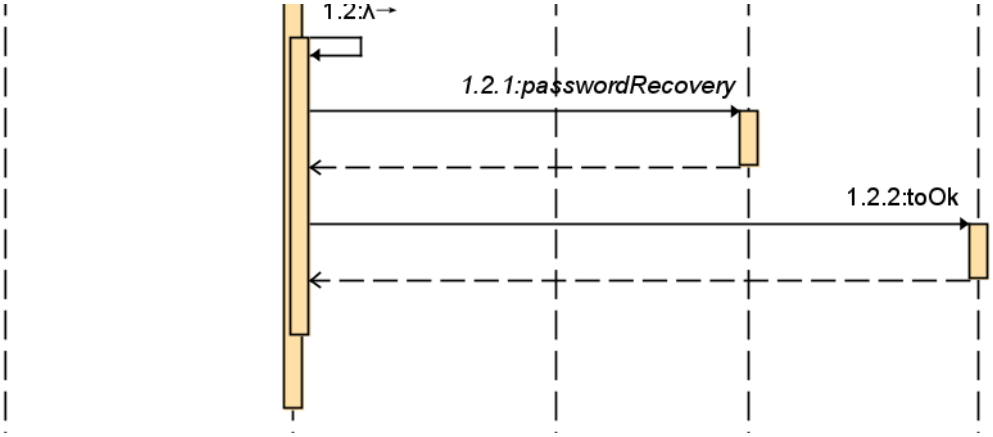
This endpoint helps the user to generates a recovery link (with a recovery token) and sends it to his/her email which is used for the frontend to guide the person to reset the password.



Verify Password Reset

This endpoint helps the user to verify the recovery token and reset the password if the token is valid.

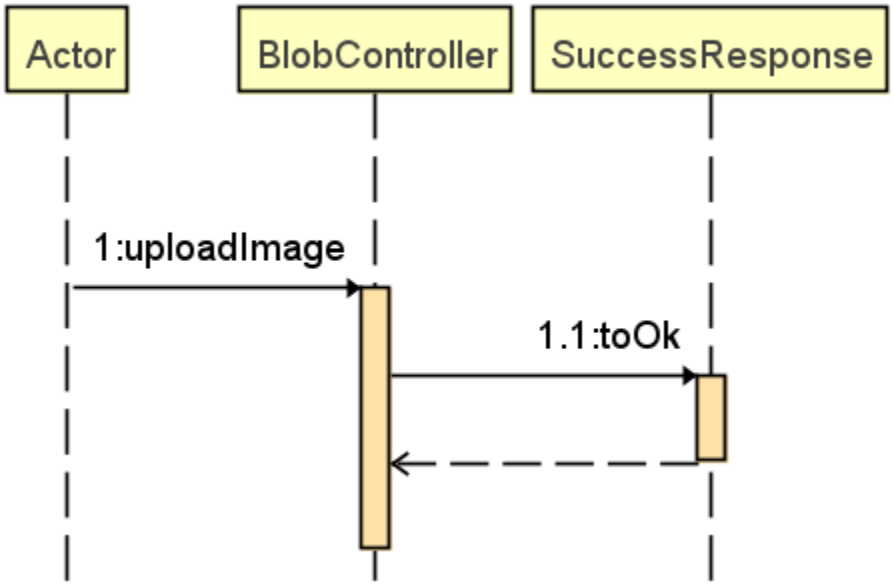




Blob Controller

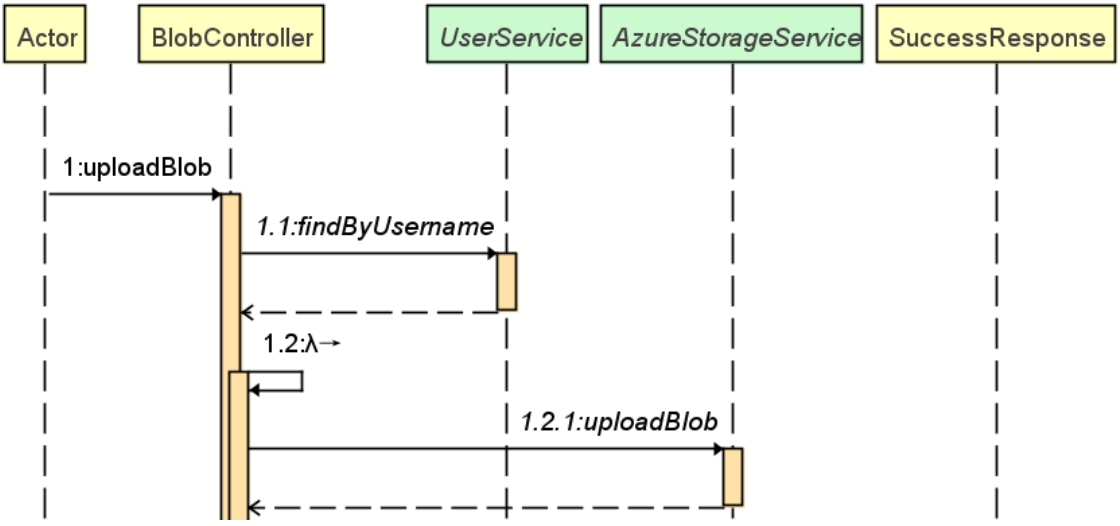
Upload Image

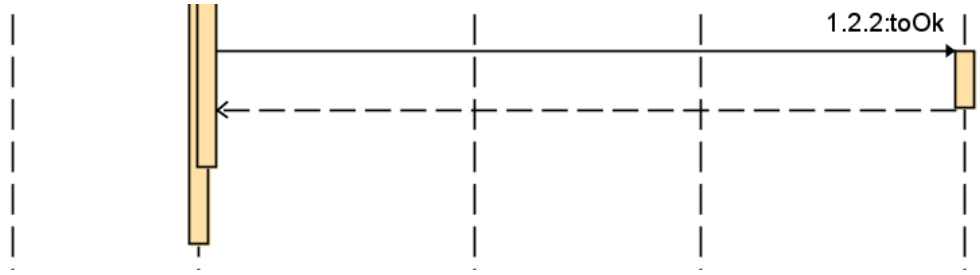
This endpoint helps the user to upload a new image.



Upload Blob

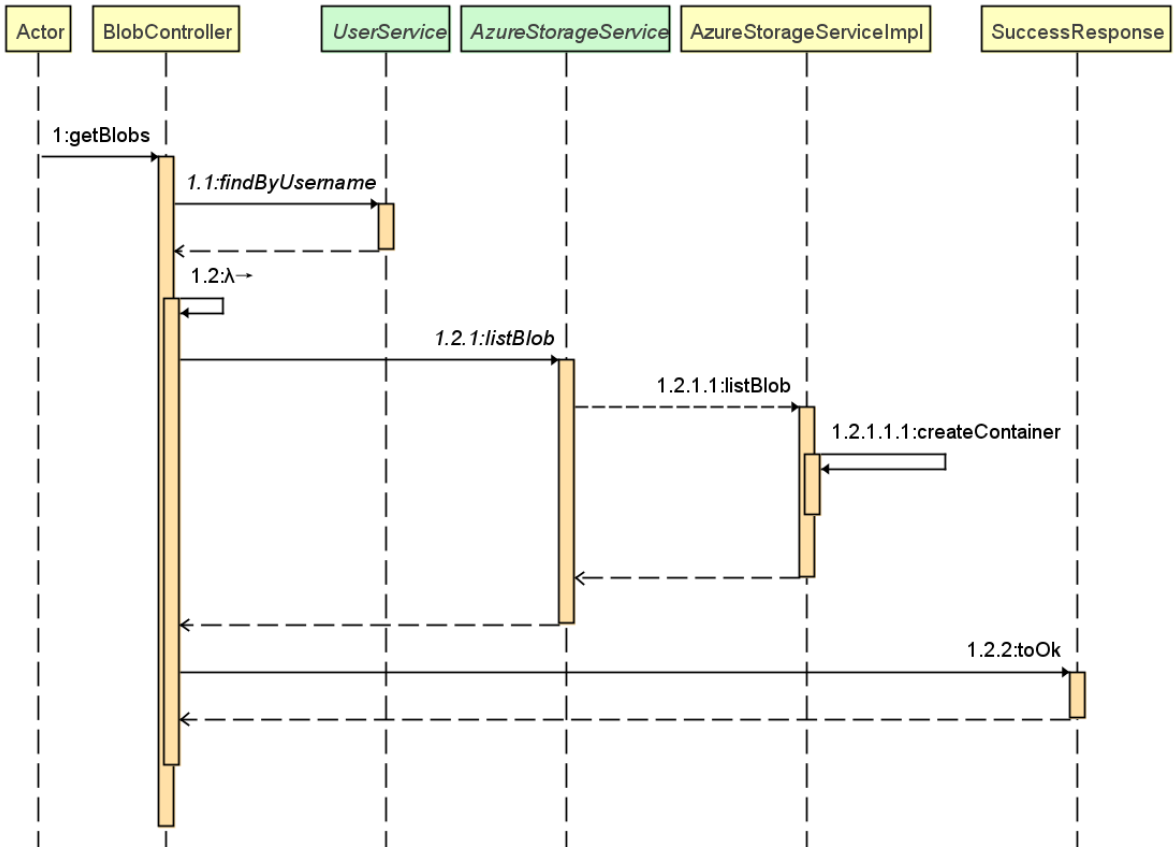
This endpoint helps the user to upload any type of files.





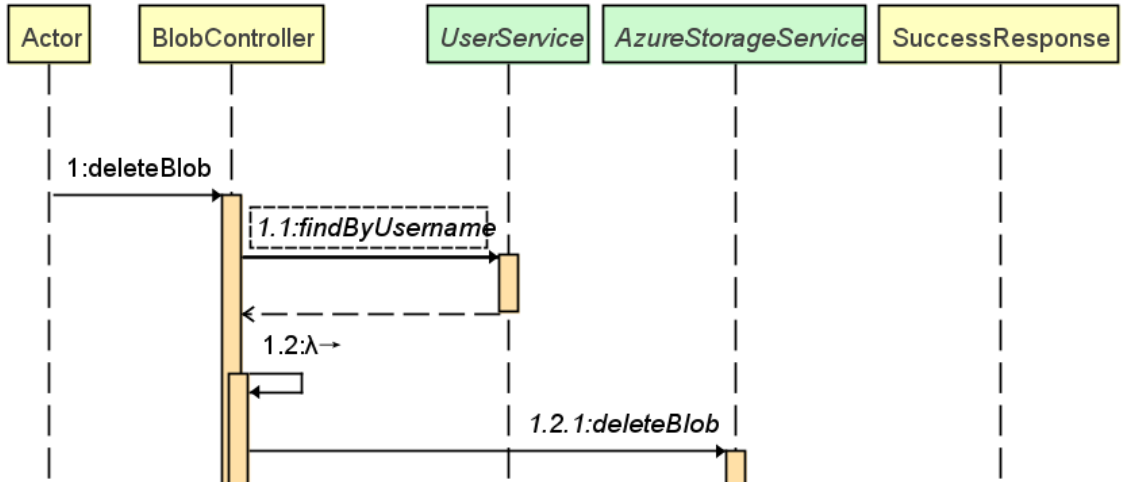
Get Blobs

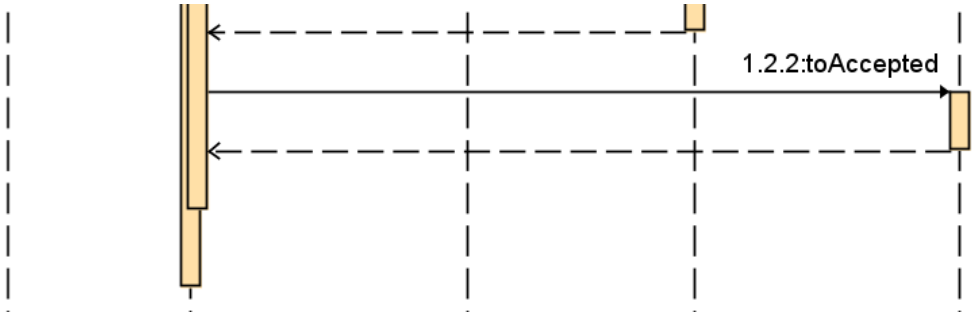
This endpoint returns all of the links of the files that the user uploaded before.



Delete Blob

This endpoint deletes a specific file from a user.

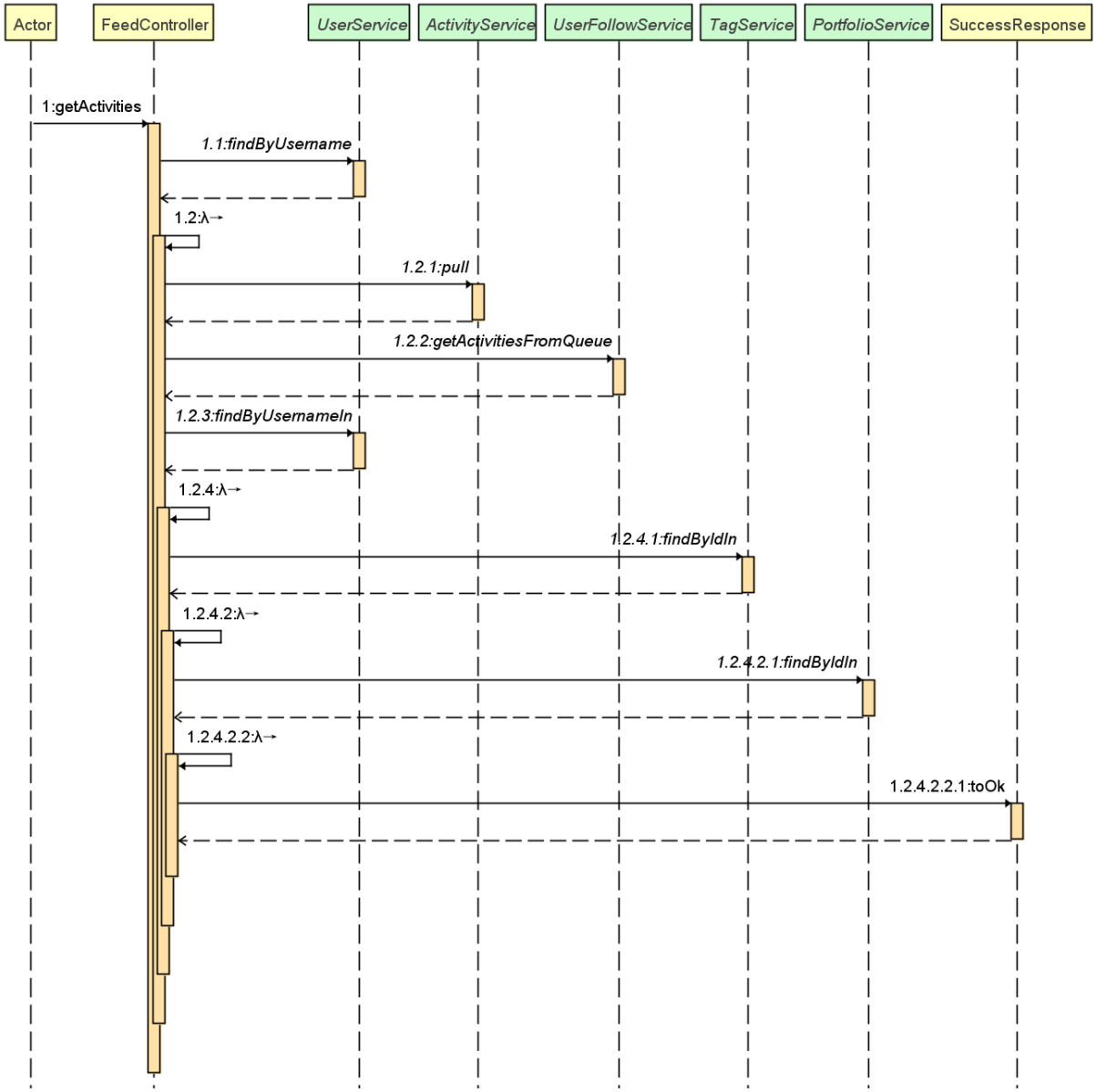




FeedController

Get Activities

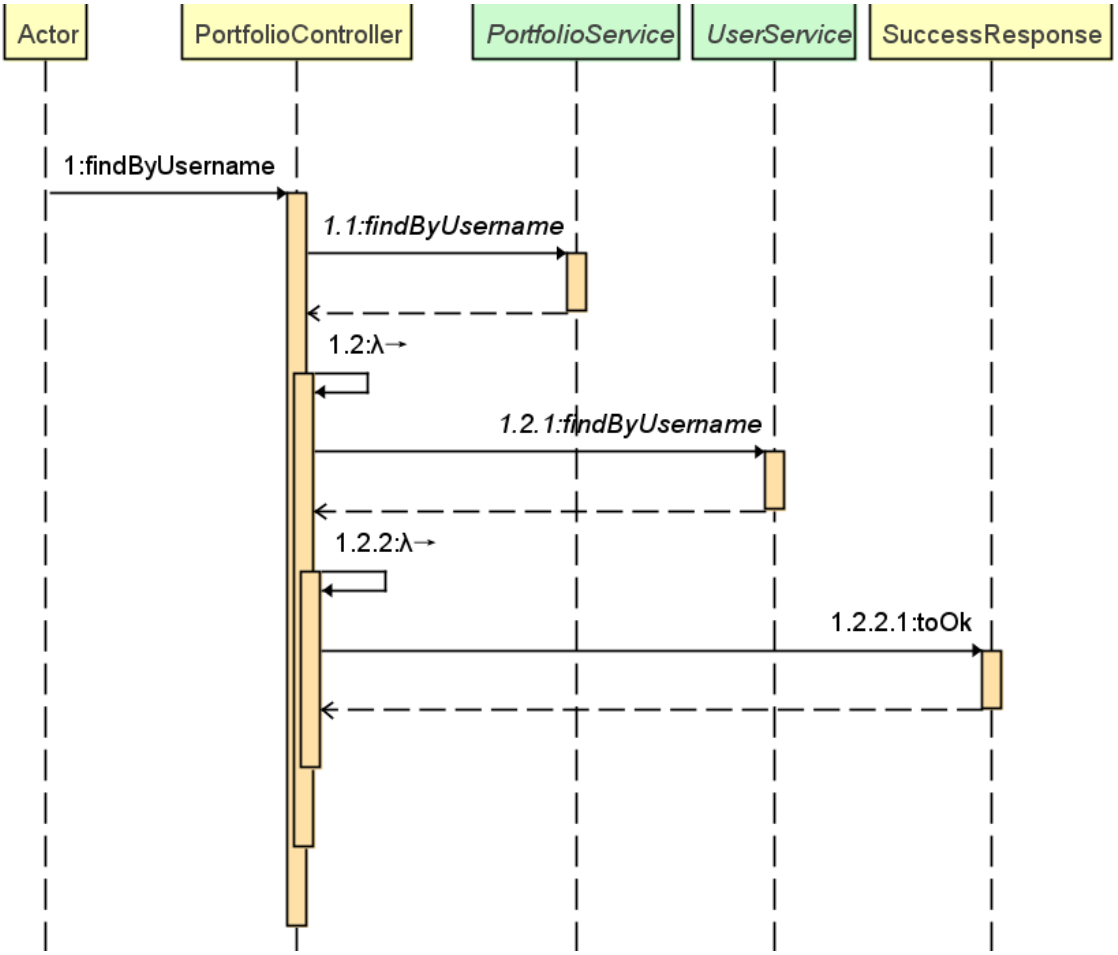
This endpoint returns some recommended e-portfolios, tags, and follow-up of the following e-portfolios for the user.



PortfolioController

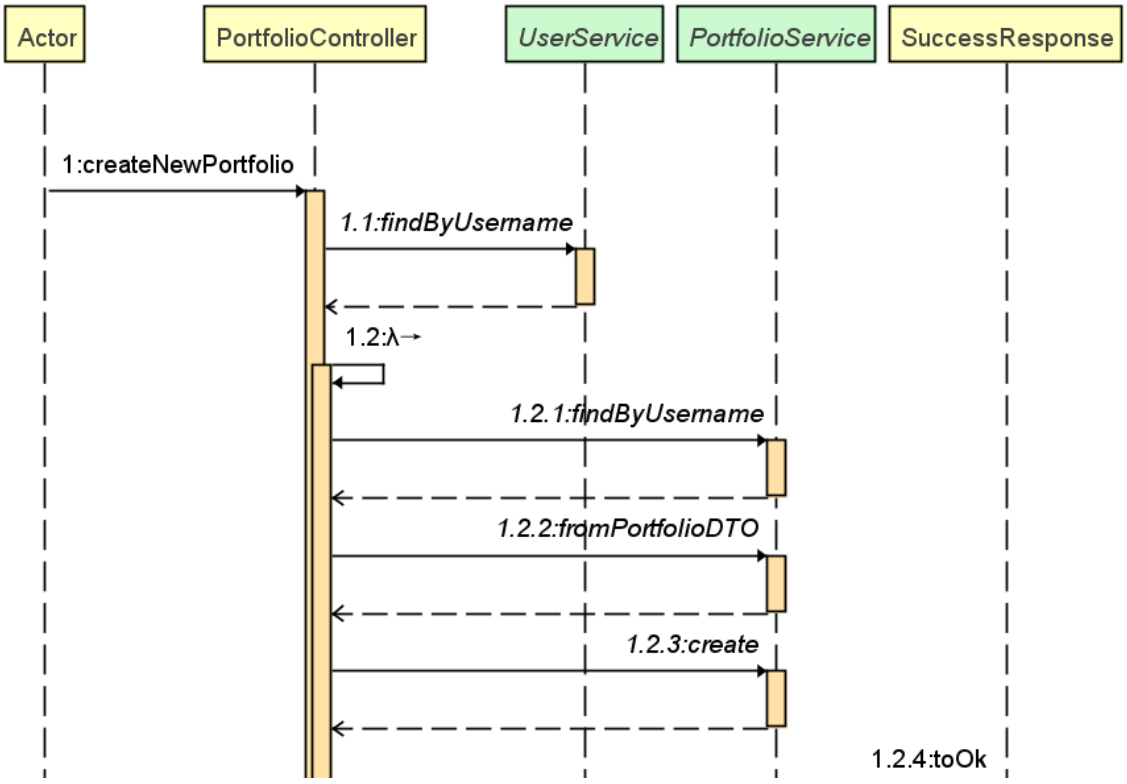
Find Portfolio By Username

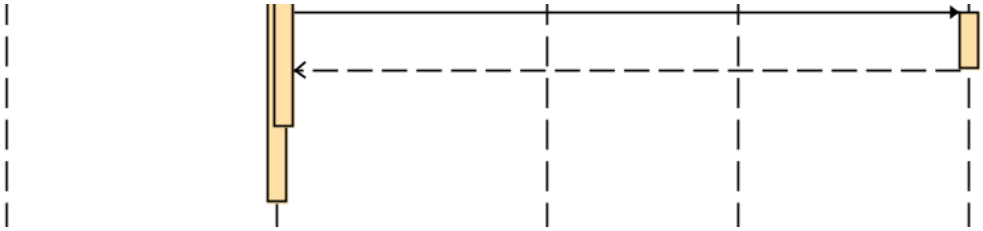
This endpoint returns the details of an e-portfolio.



Create Portfolio

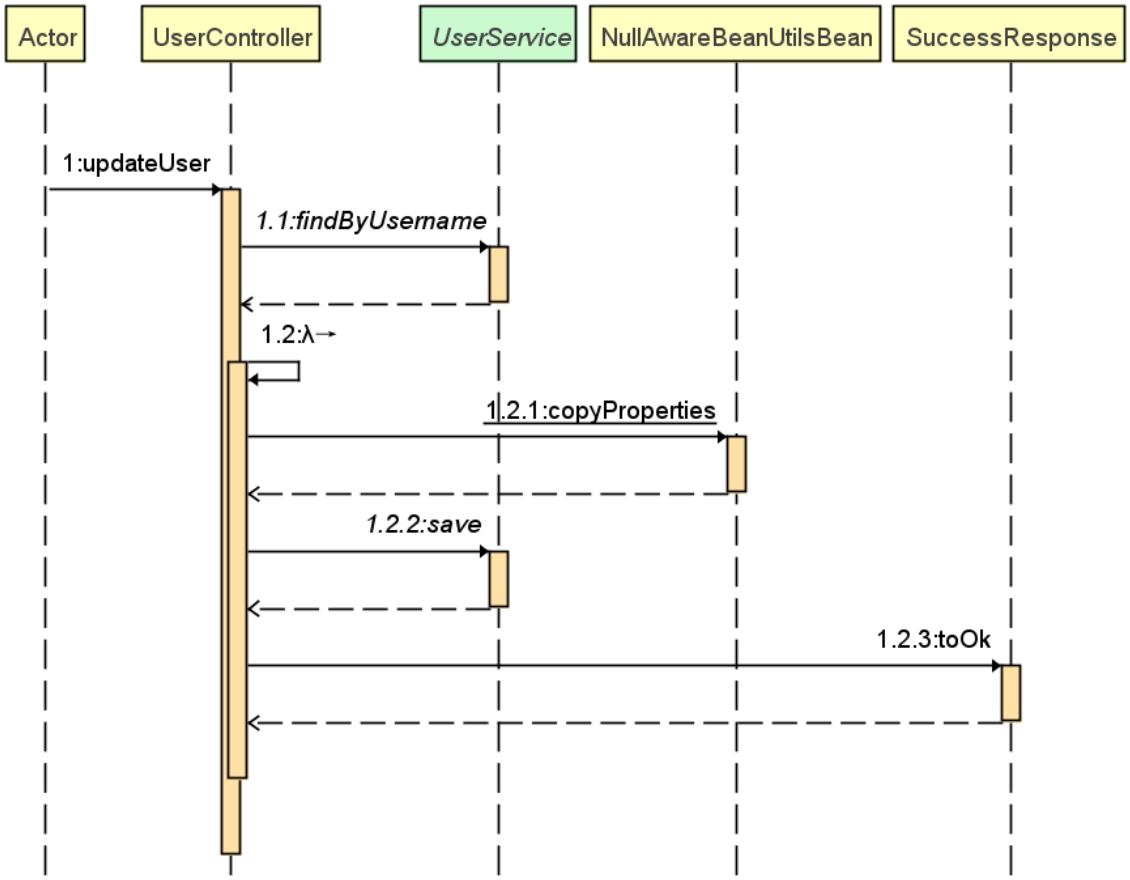
This endpoint helps the user to create a new e-portfolio.





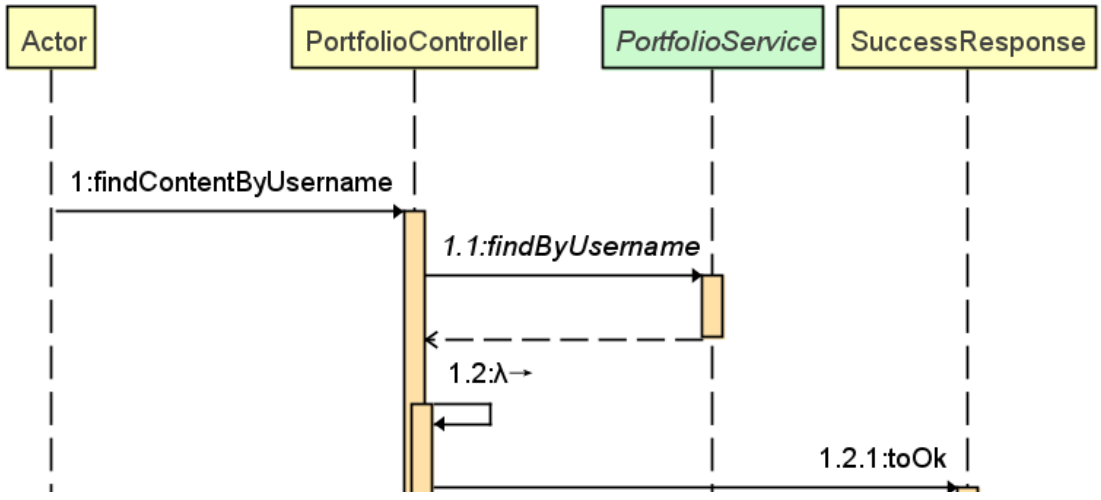
Update Portfolio

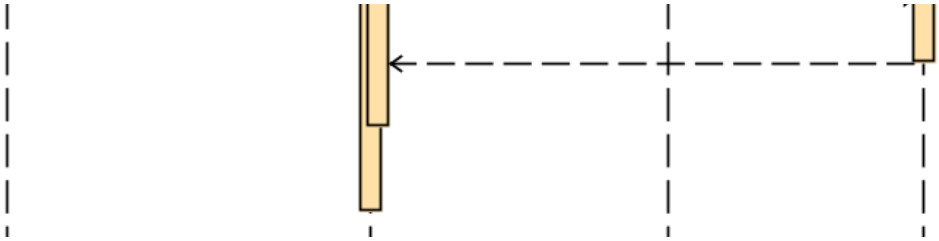
This endpoint helps the user to update the information of his/her e-portfolio.



Find Content By Username

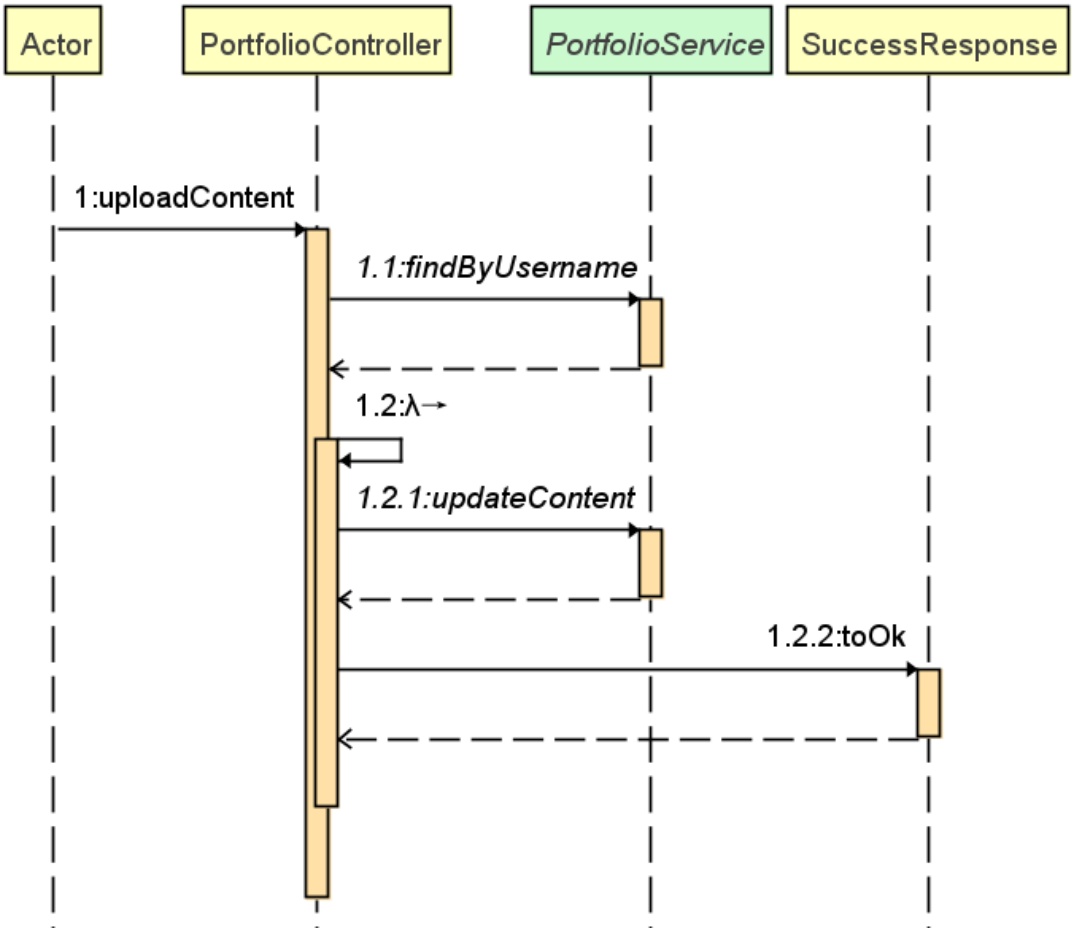
This endpoint returns the content of an e-portfolio.





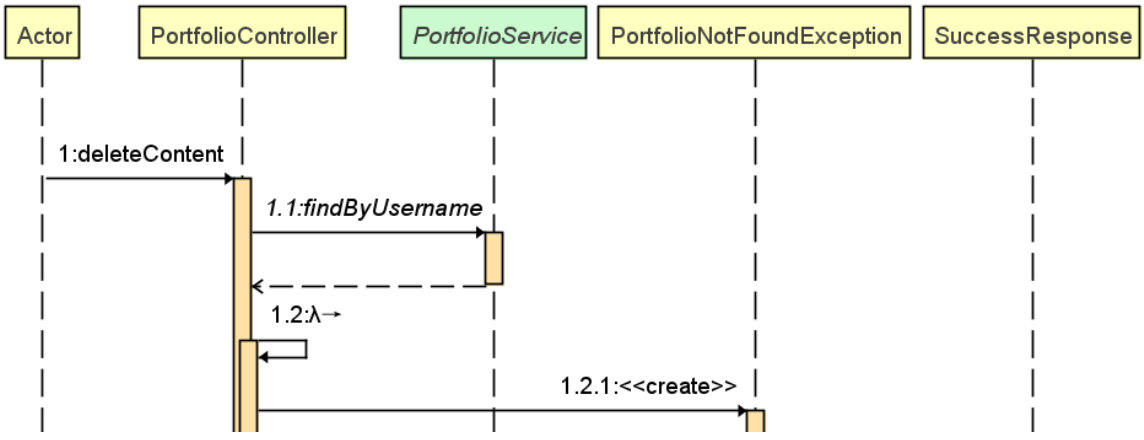
Upload Content

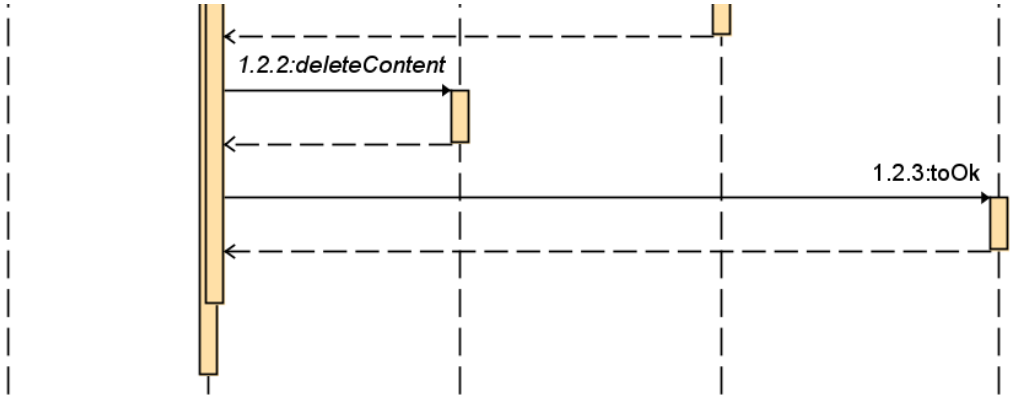
This endpoint helps the user to update the content of an e-portfolio.



Delete Content

This endpoint helps the user to delete the content of an e-portfolio.

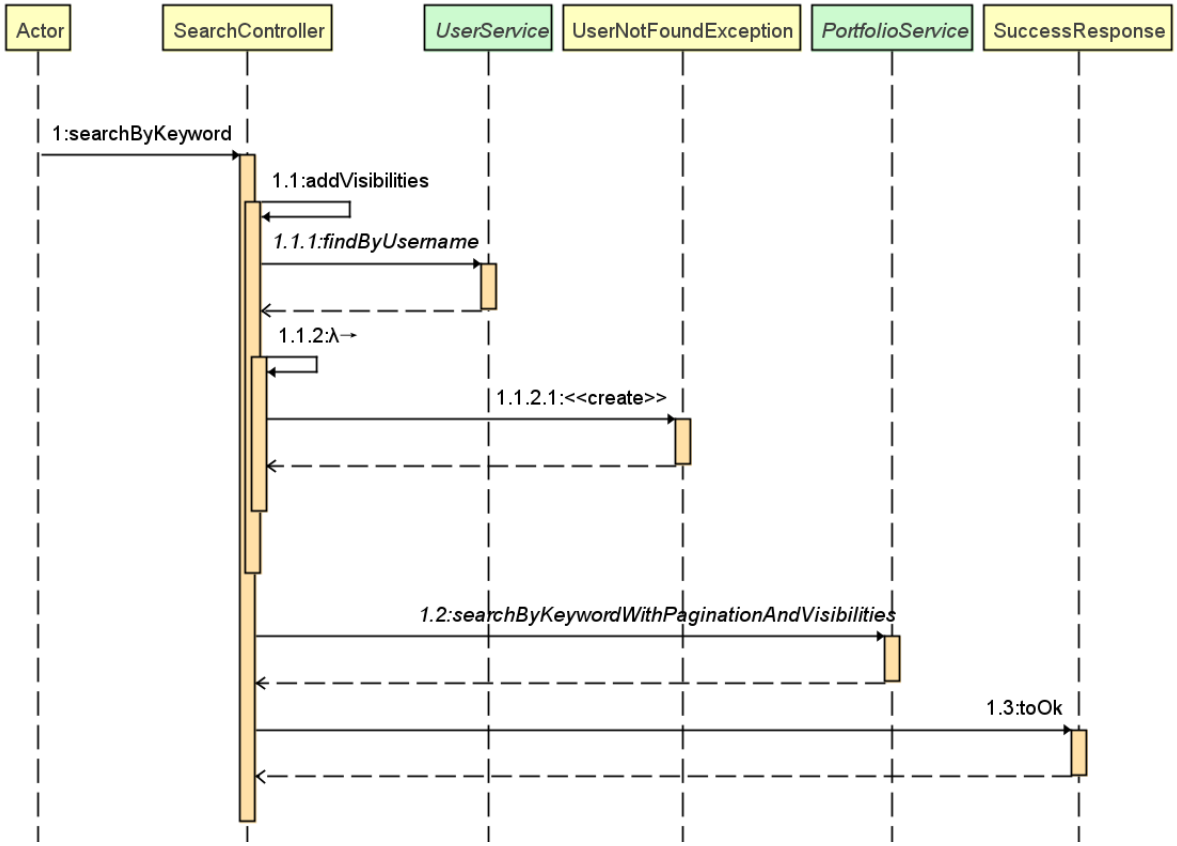




SearchController

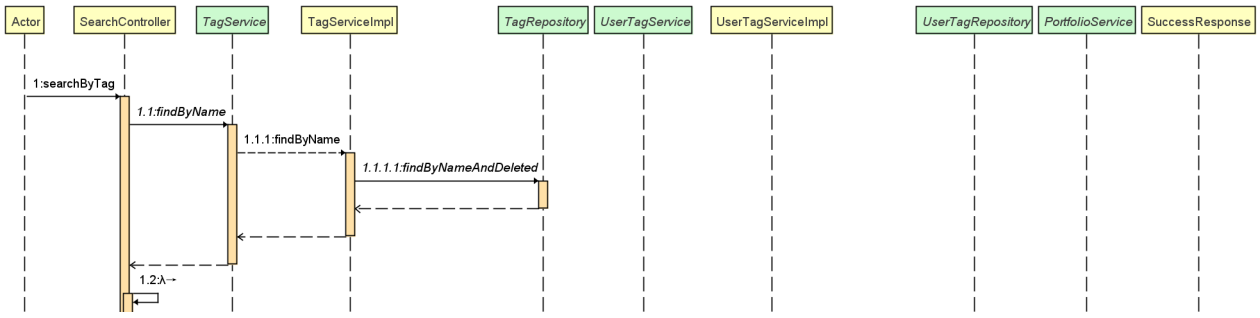
Search By Keyword

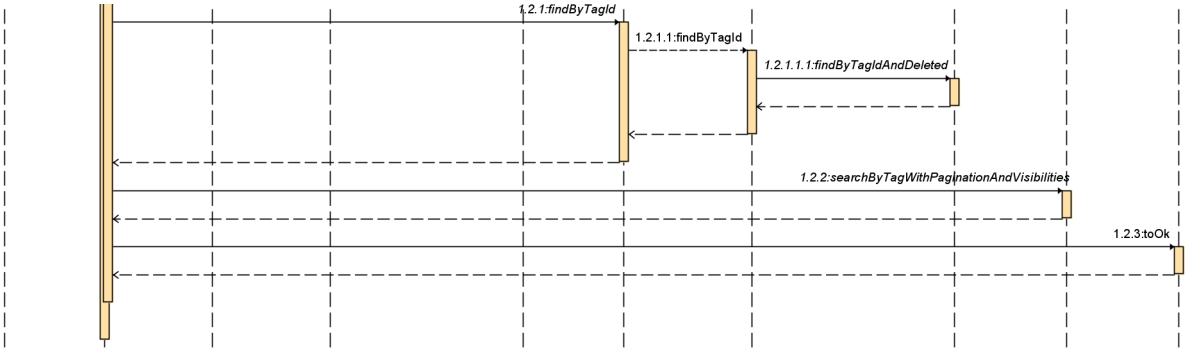
To search a keyword (e.g. username) for one category. The API will respond portfolios.



Search By Tag

To search portfolios by entering a tag name. The related portfolios will be shown only if the tag name is exactly matched.

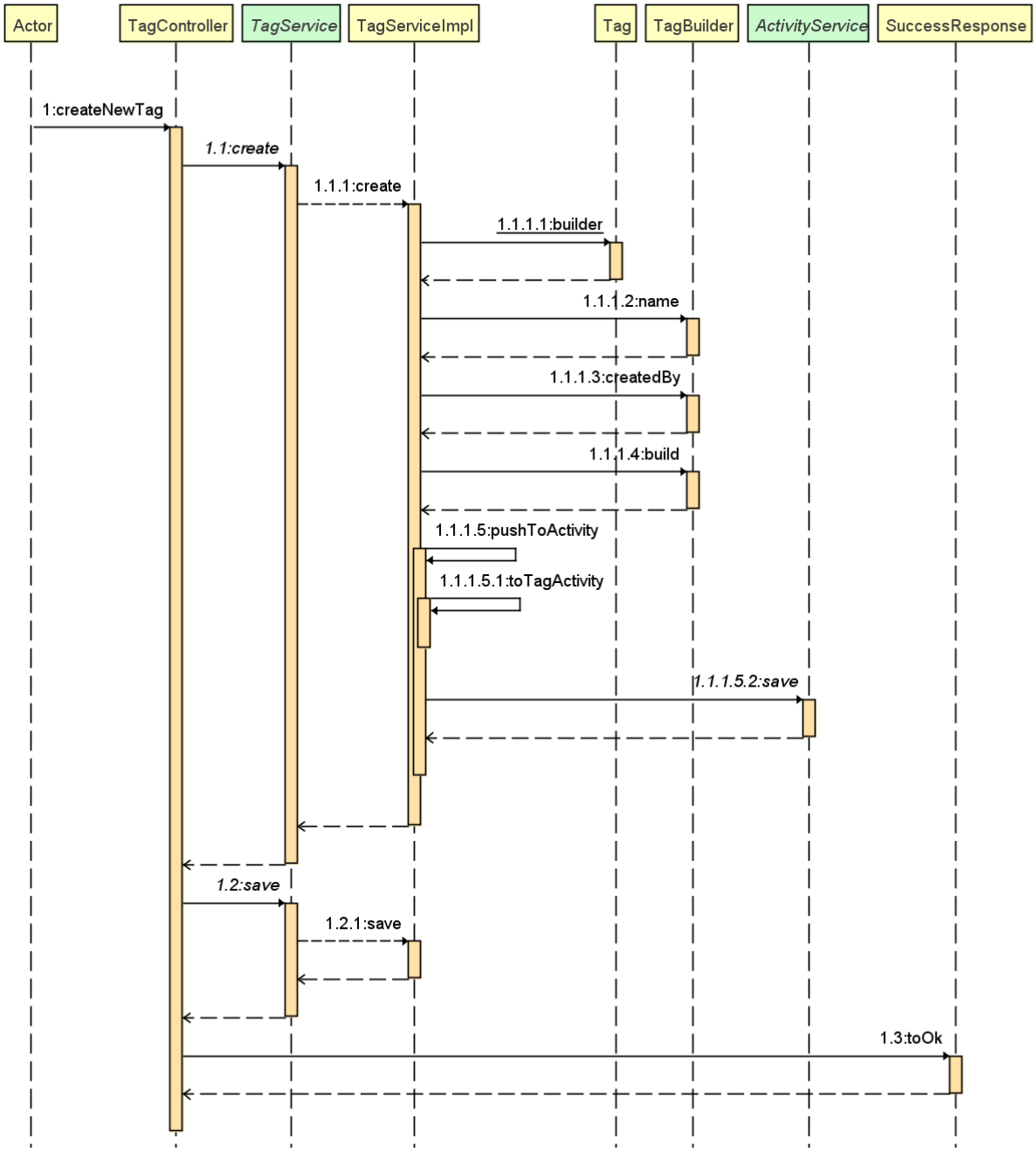




Tag Controller

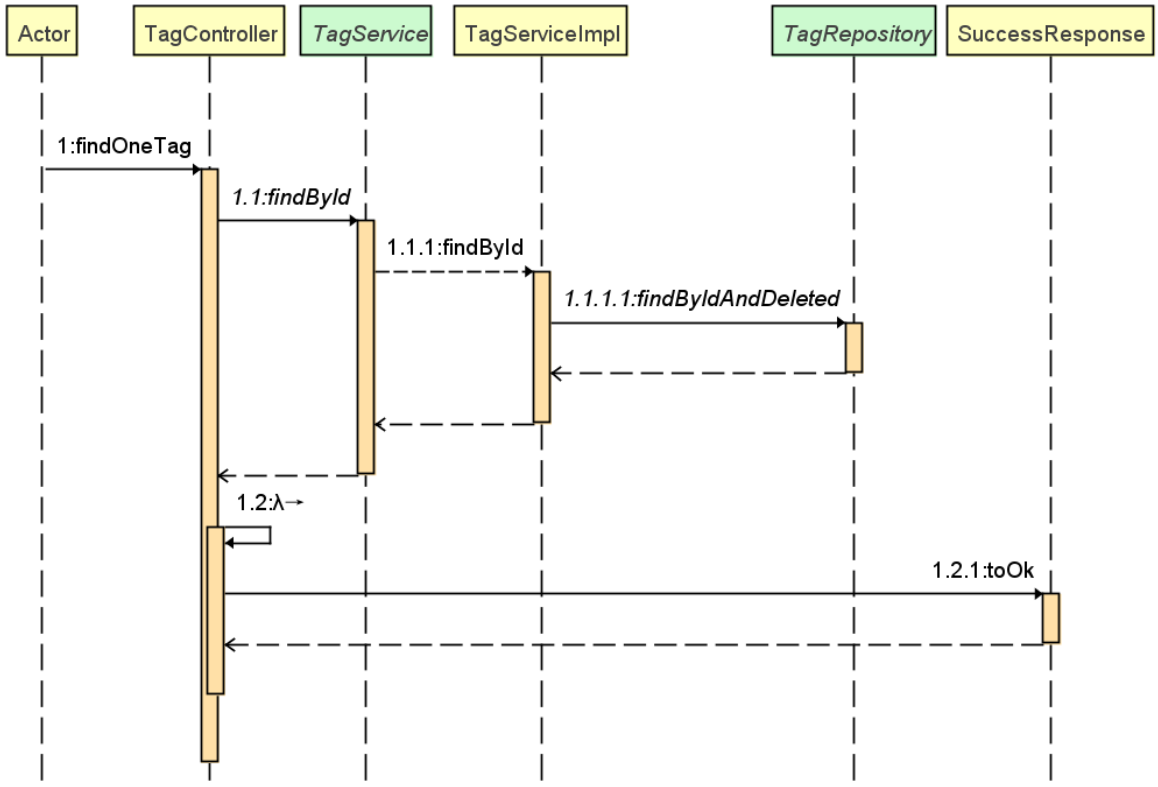
Create Tag

The process to create a tag and save to the database.



Find Tag By Id

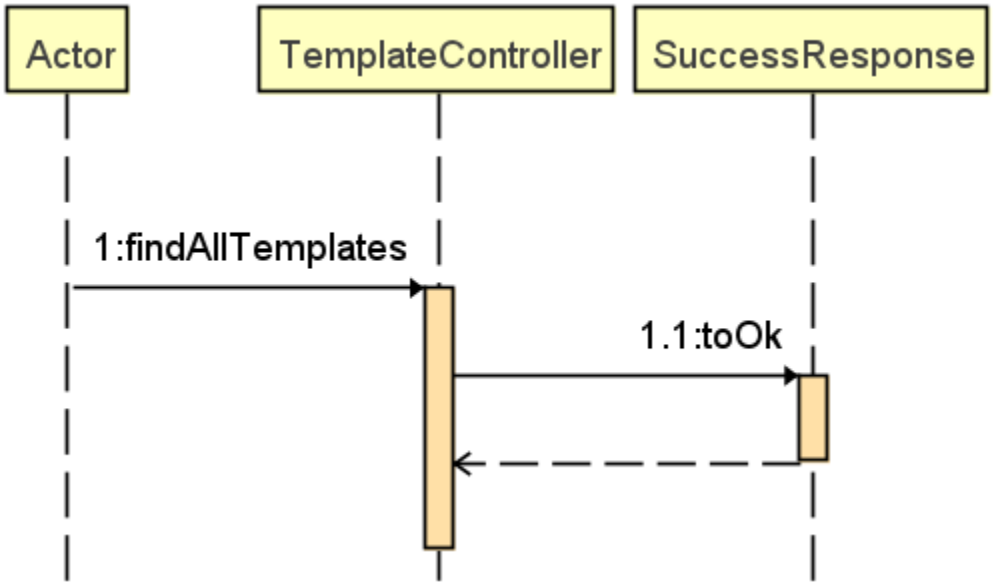
This API takes a tag id as input parameter and return the tag if the ID is existing in the database.



Template Controller

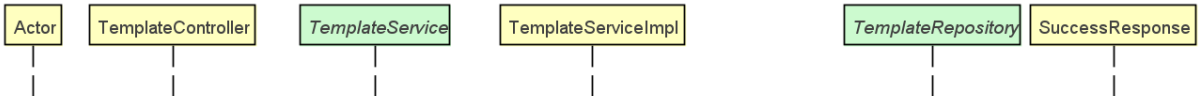
Find All Templates

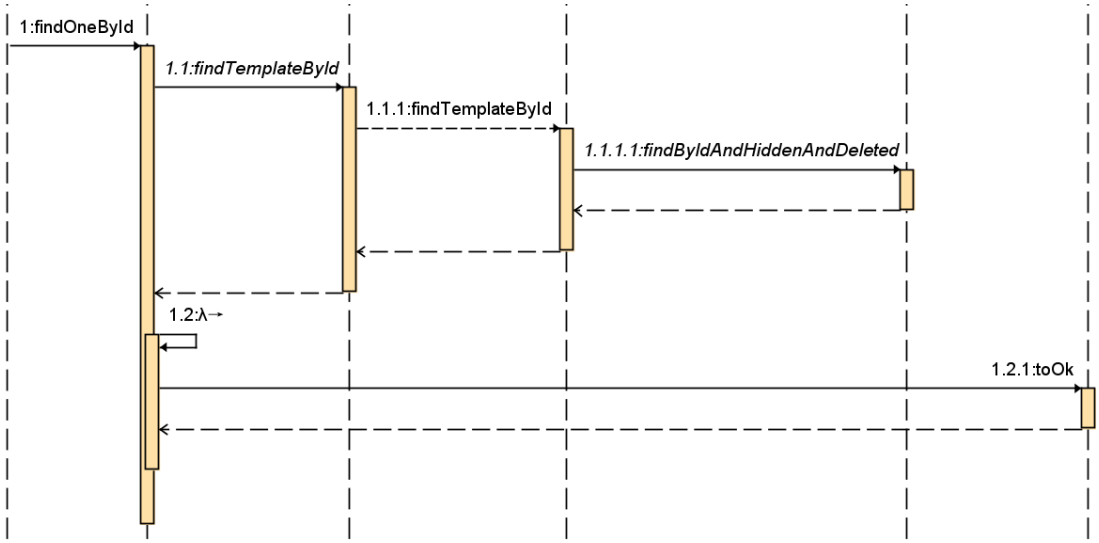
The endpoint to find existing templates.



Find Template By Id

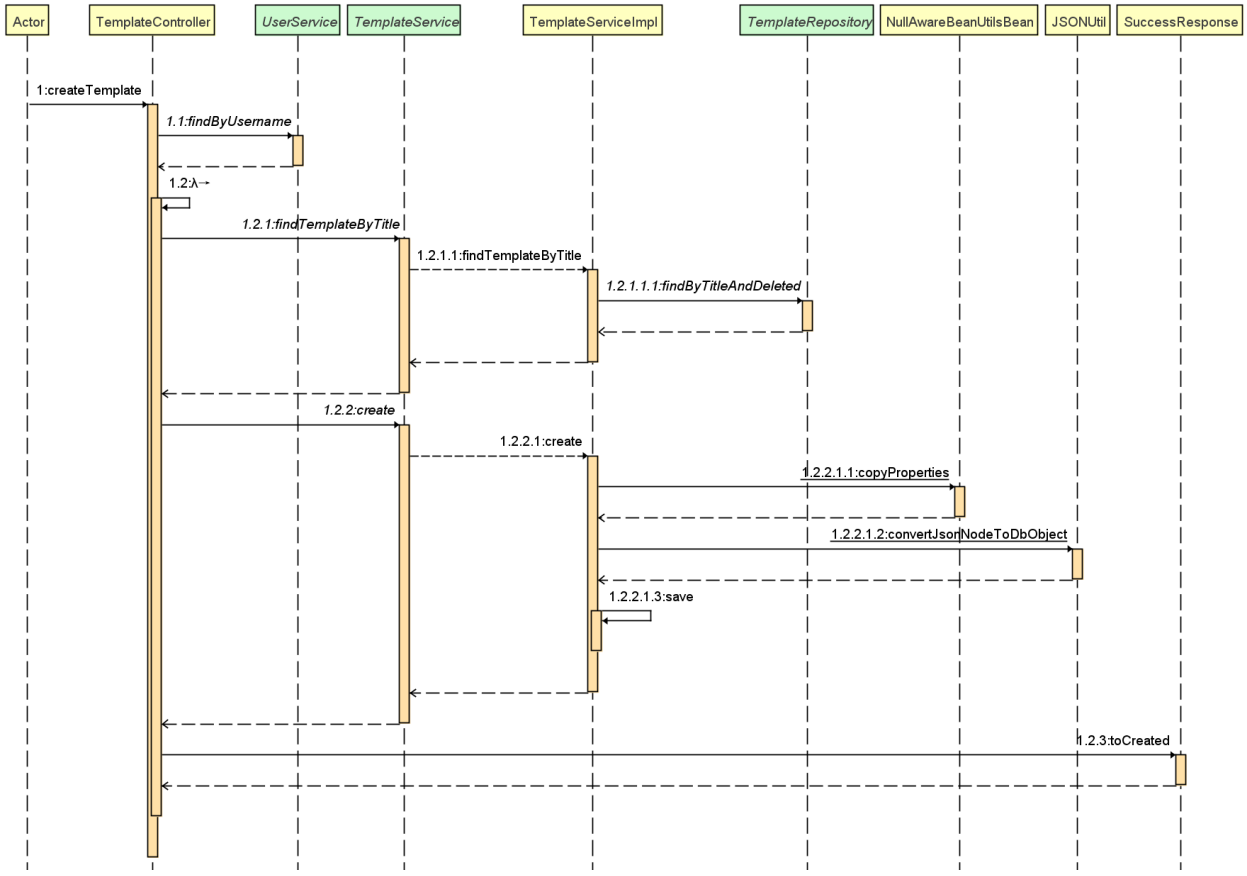
The endpoint takes an Id of a template to find whether it is in the database. If true, the related information of the template will be returned.





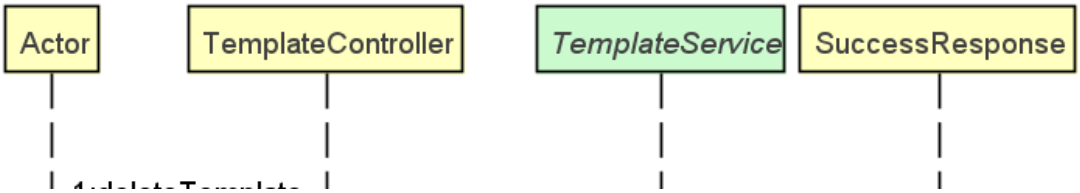
Create Template

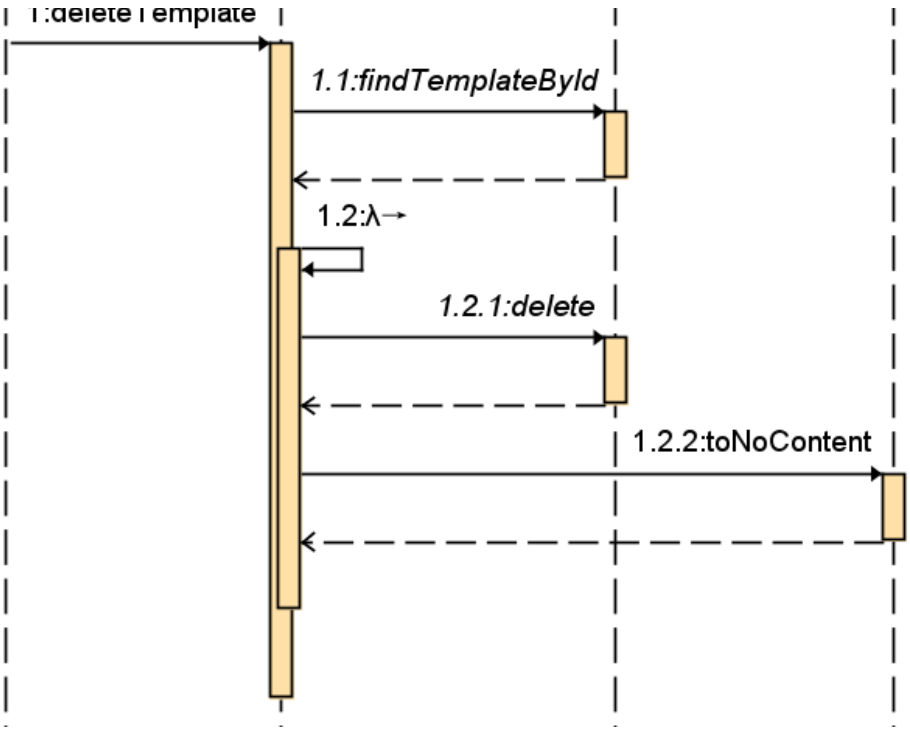
The process to create a template for users to edit.



Delete Template

The endpoint to delete a template.

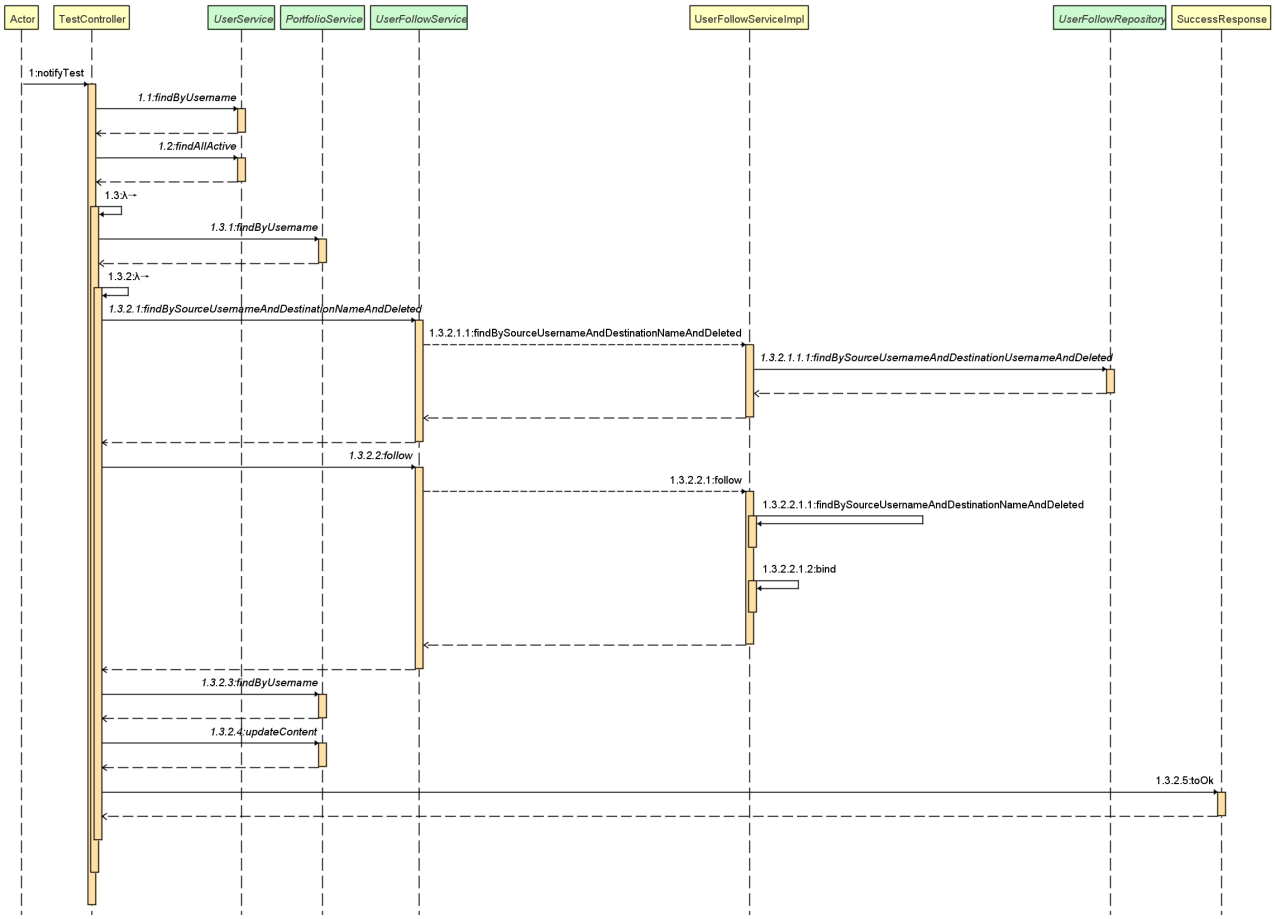




Test Controller

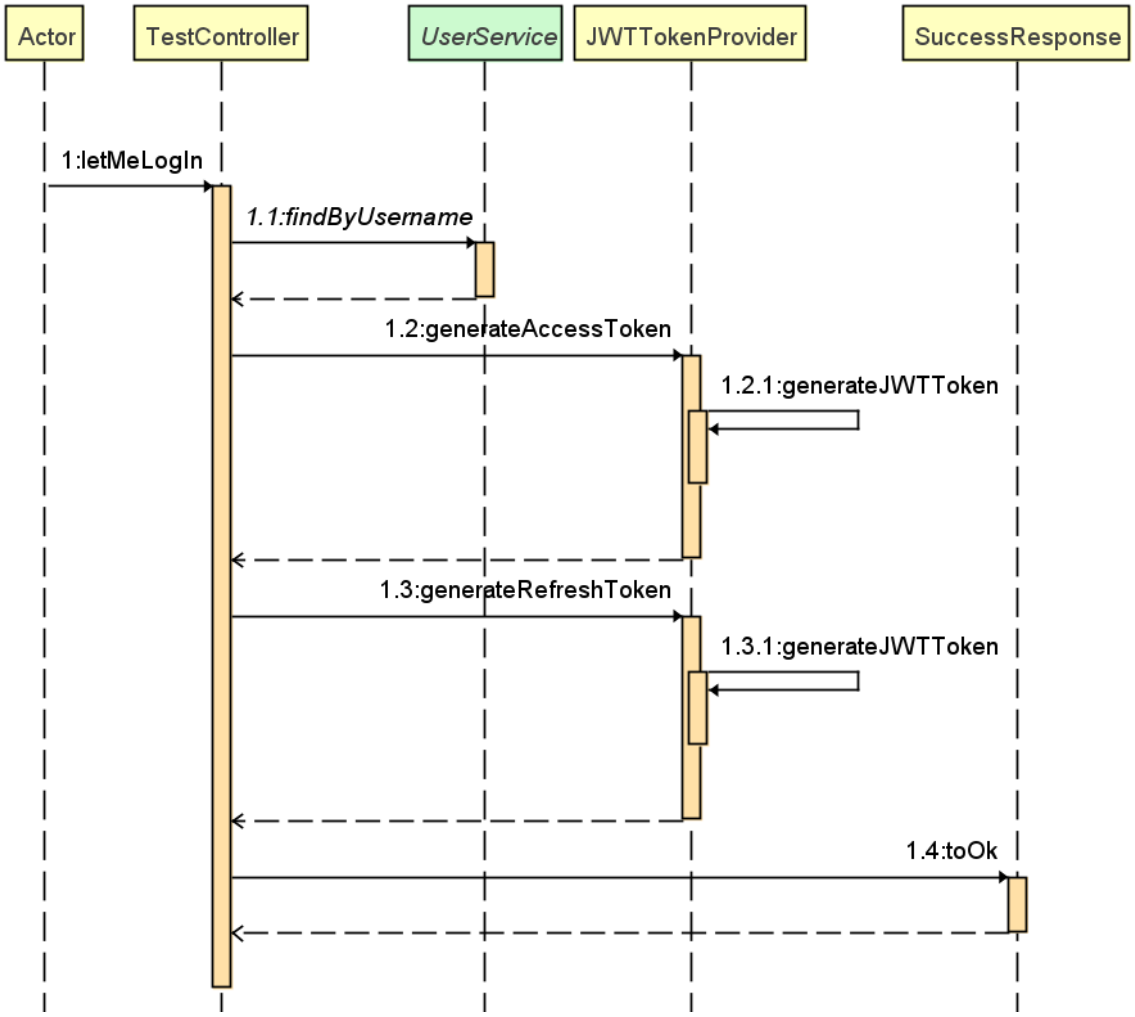
Notify Test

The endpoint to test whether a user can get notification from the followed users.

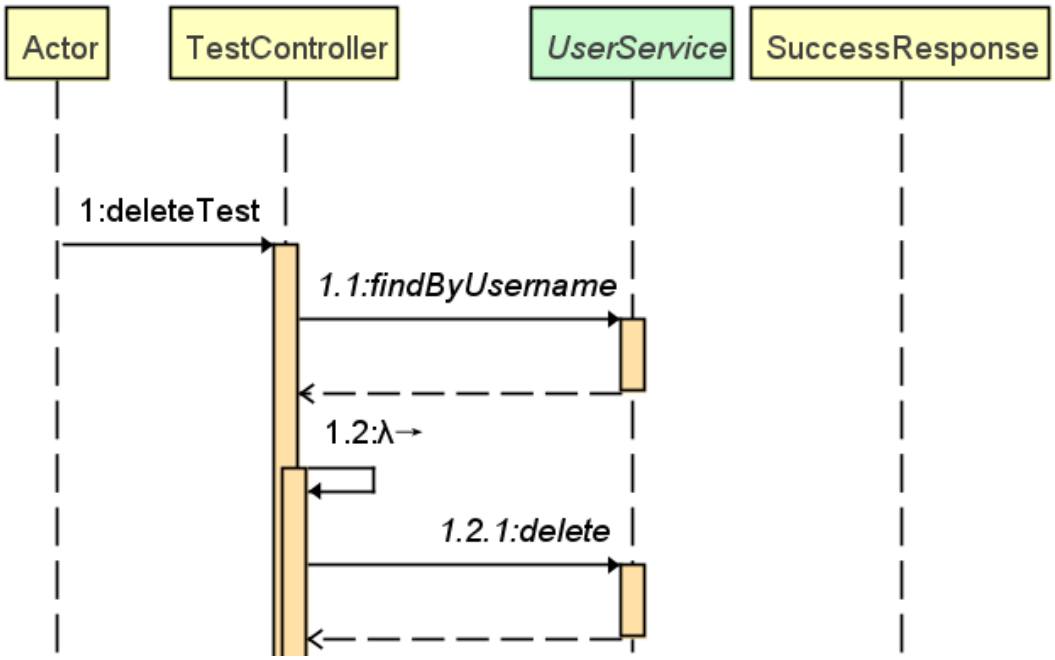


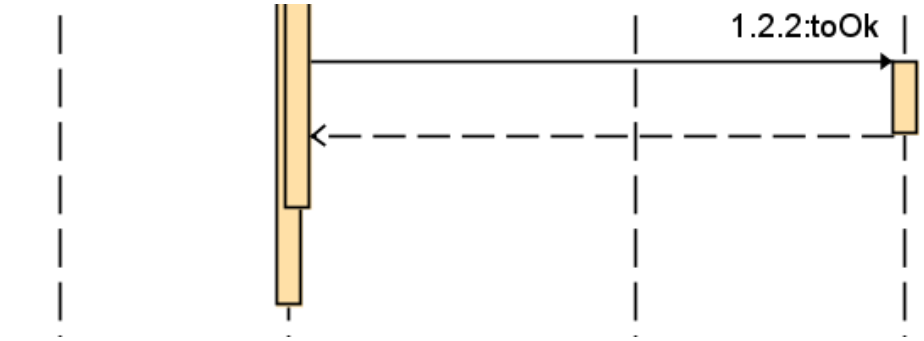
Login As Test

The endpoint to conveniently log in as the test user.



Delete Test

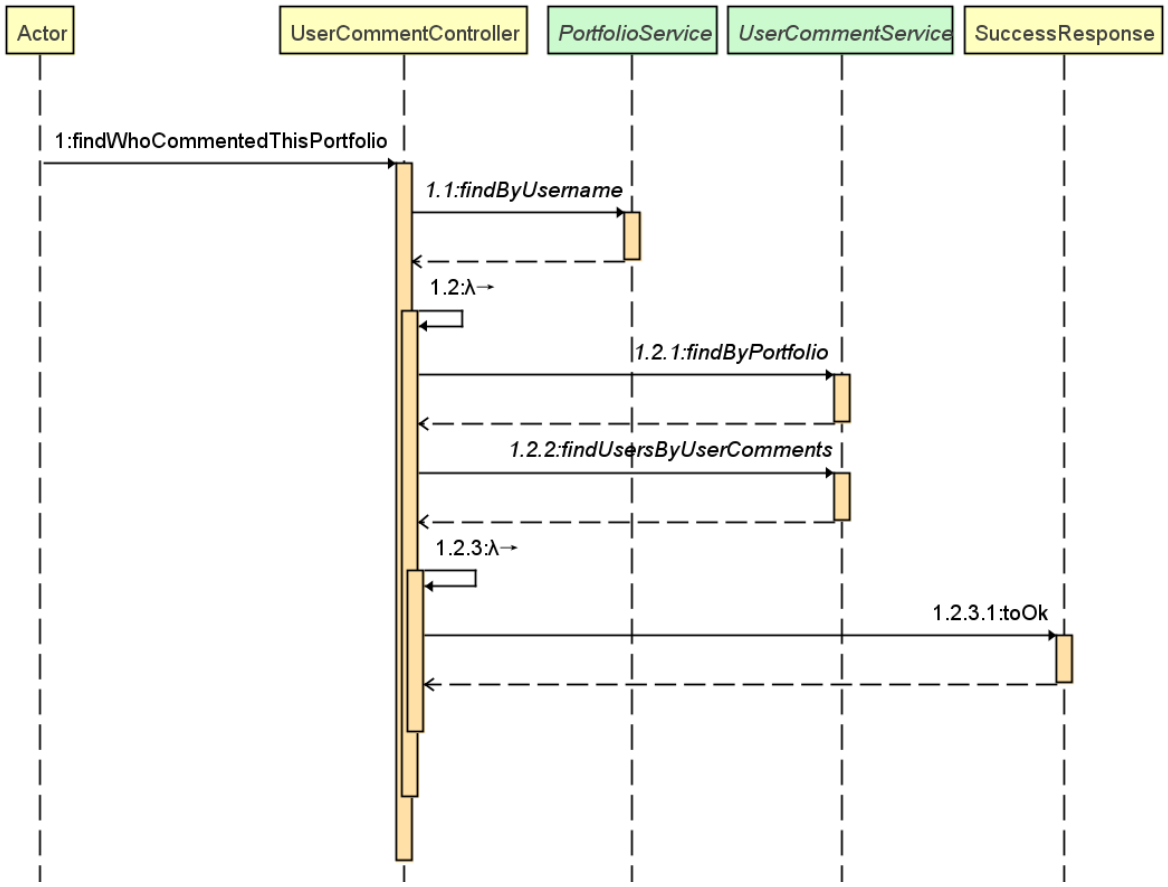




User Comment Controller

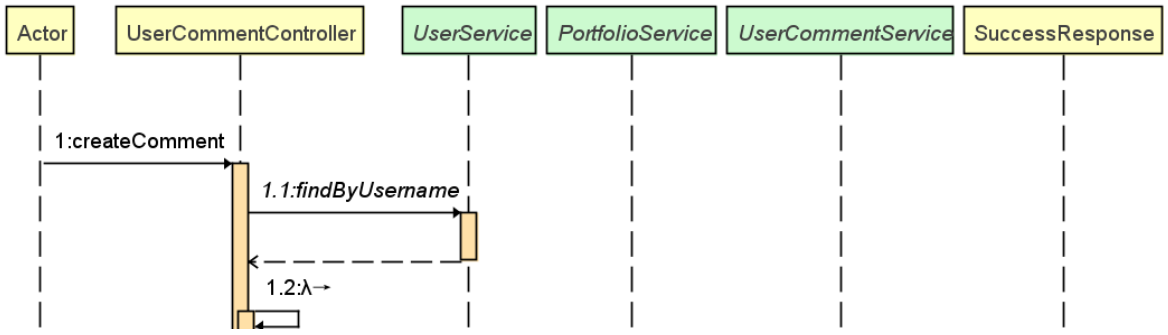
Find Who Commented This Portfolio

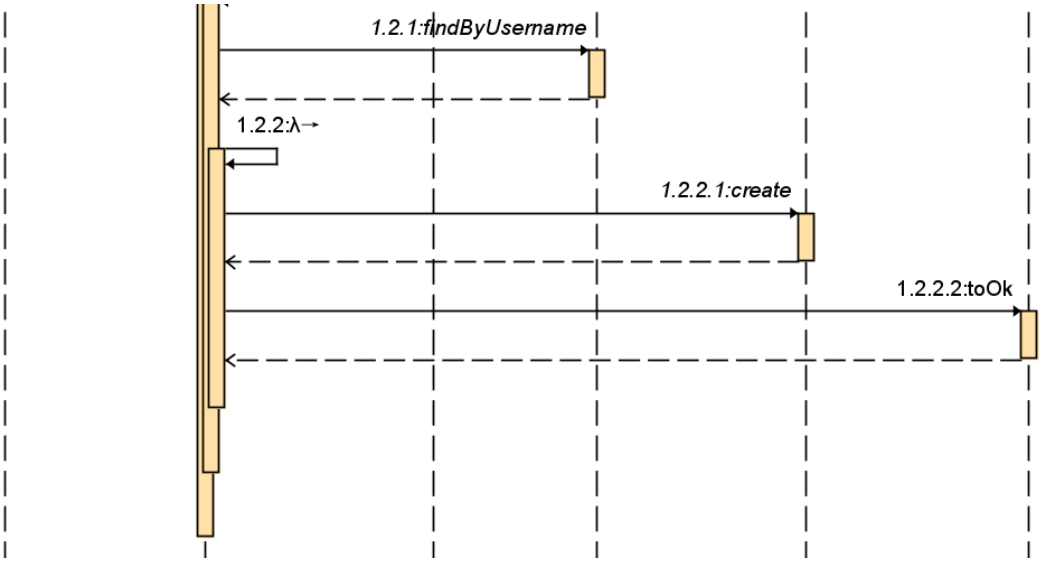
The endpoint for users to find who commented on his/her portfolio.



Create Comment

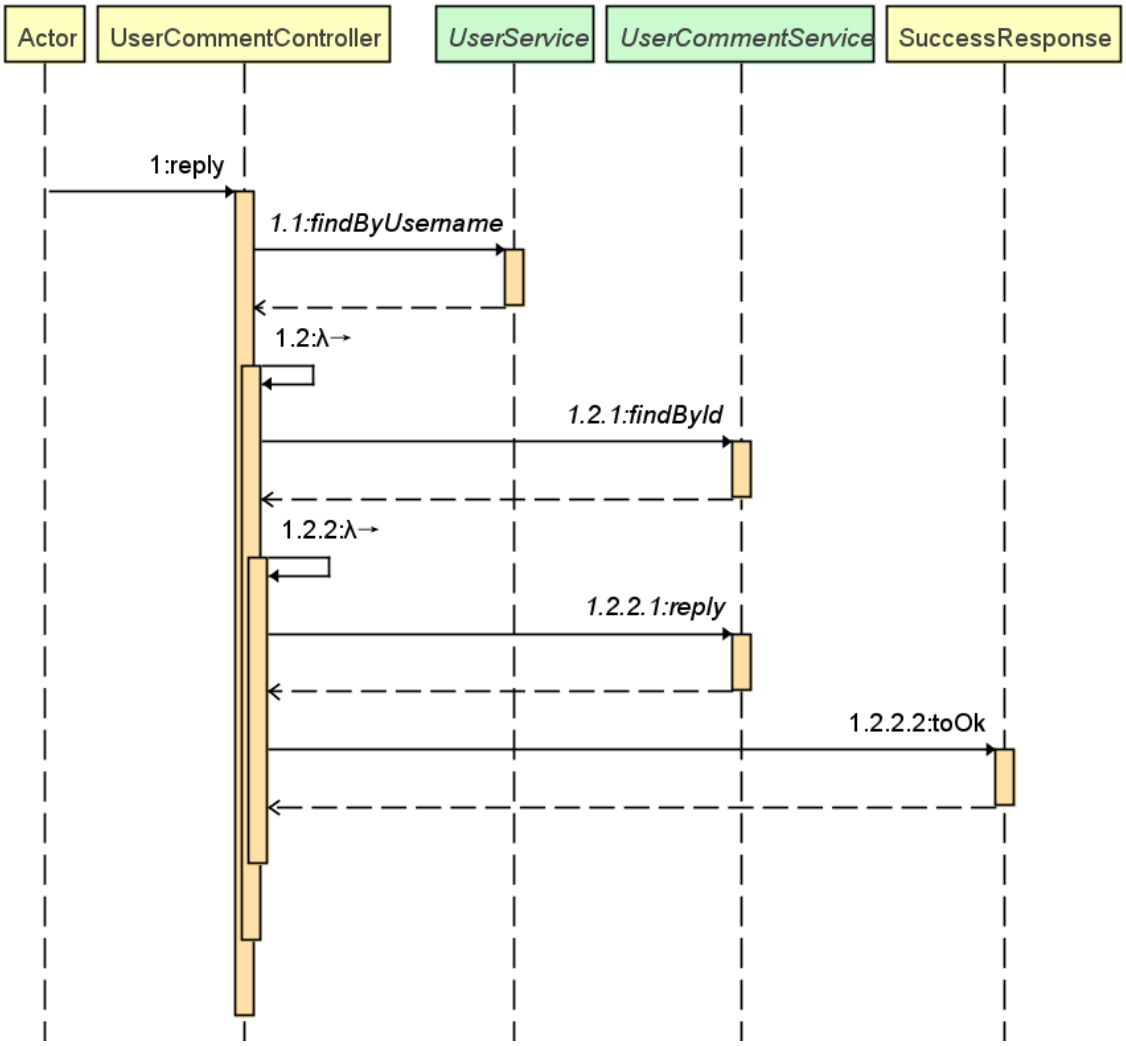
The endpoint for users to write comments on portfolios





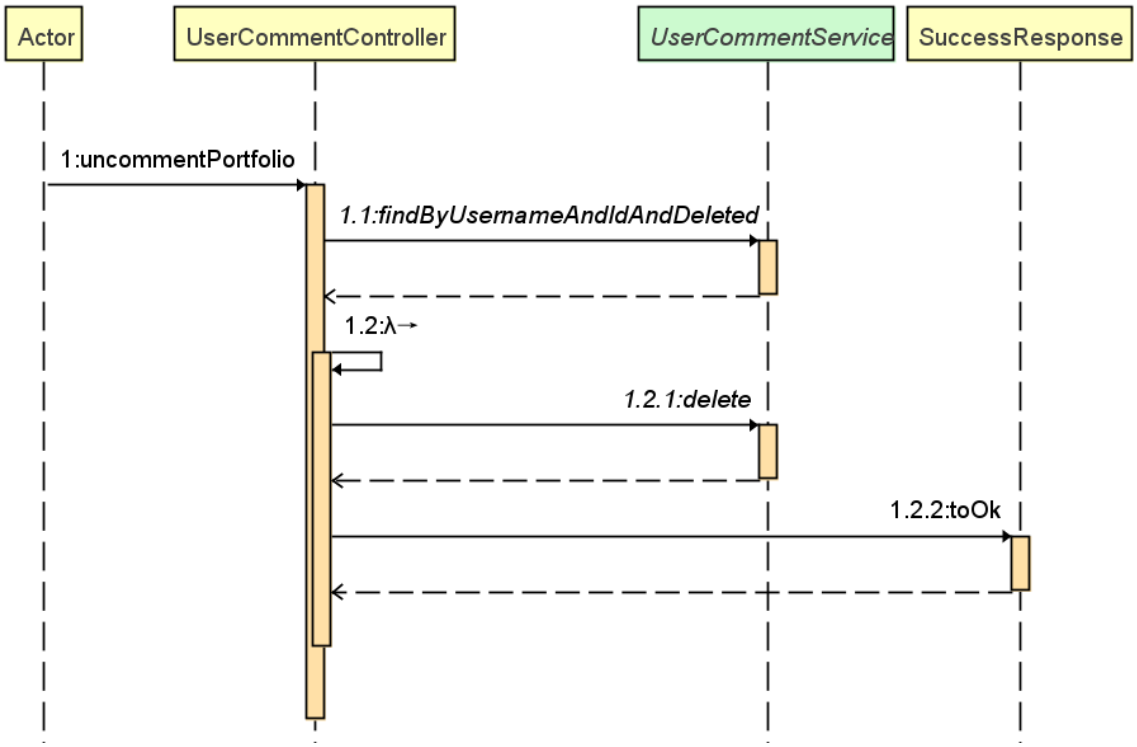
Reply

The endpoint for users to reply to others' comments



Delete Comment

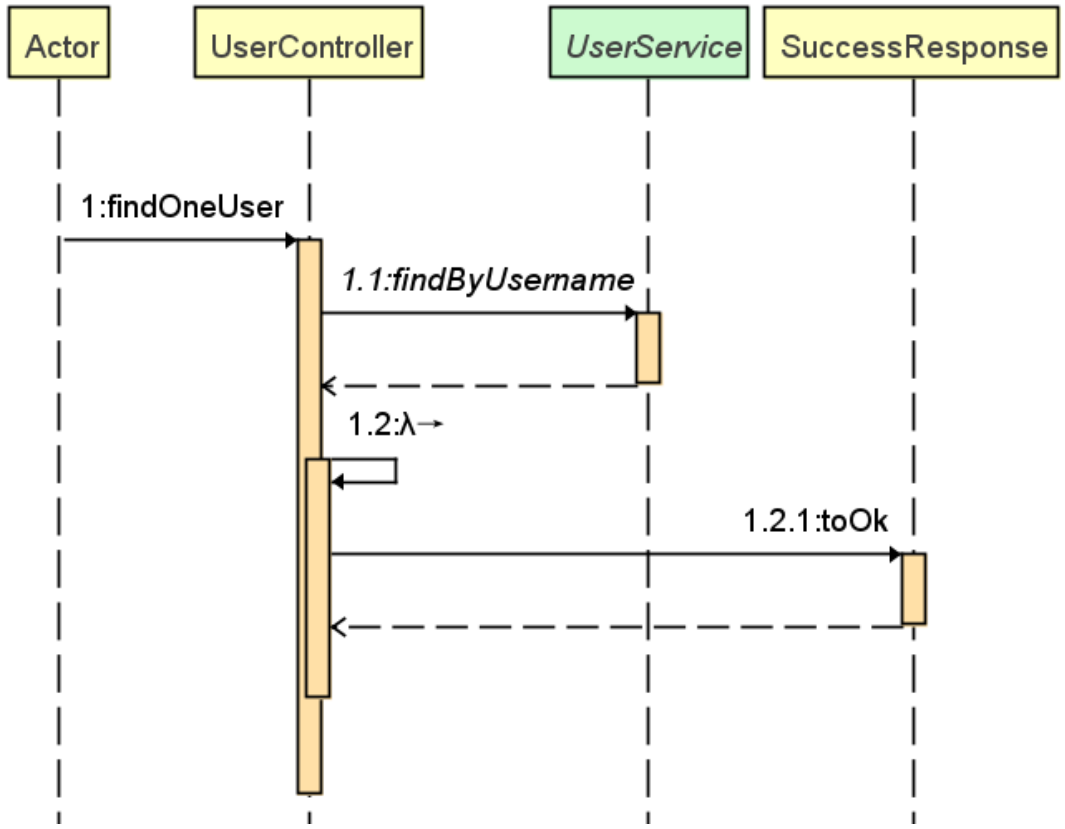
The endpoint for users to delete his/her comments.



User Controller

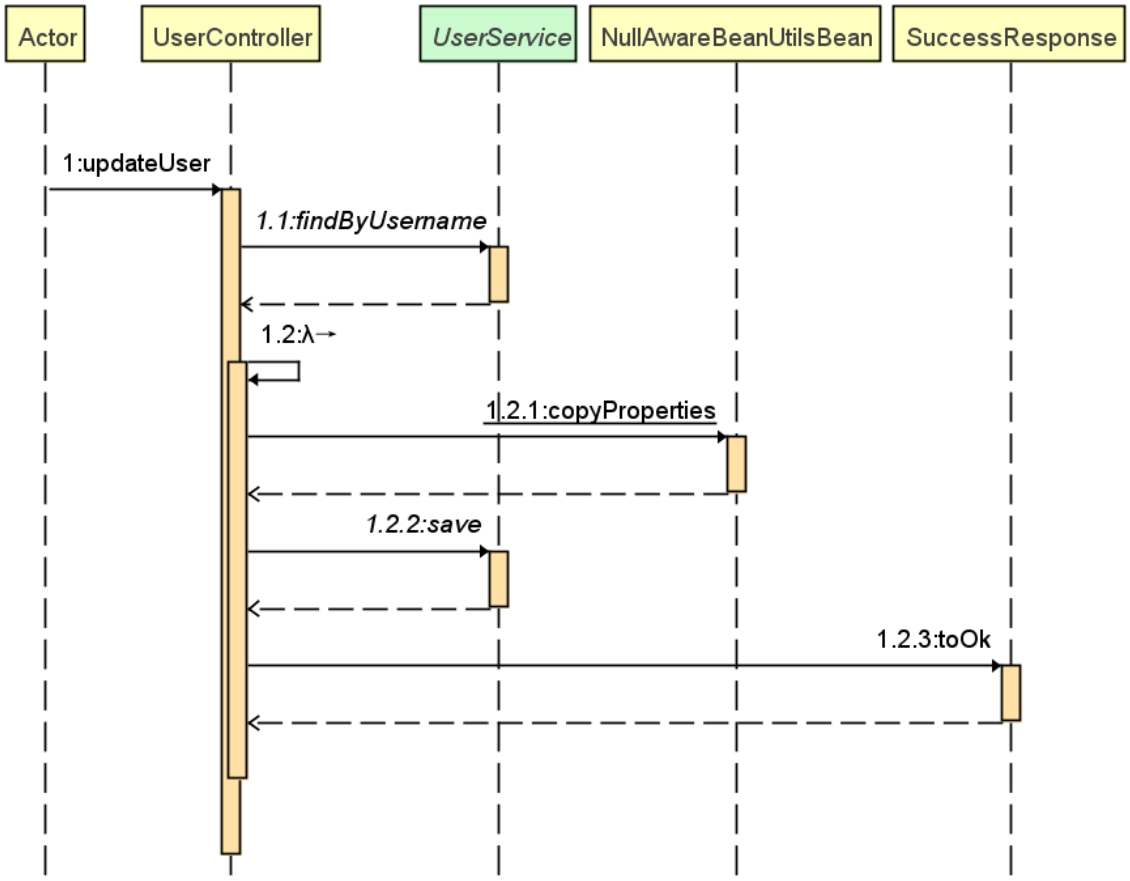
Find One User

This endpoint returns the detail of a user.



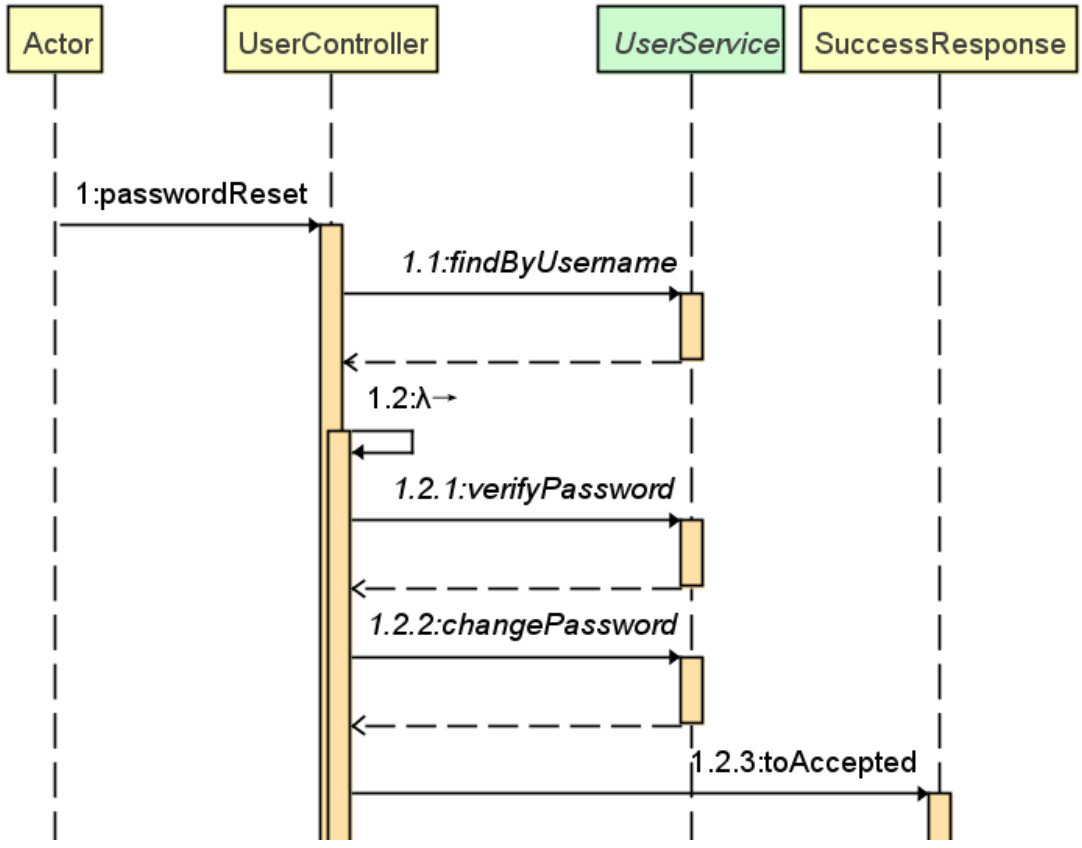
Update User

This endpoint updates the information of the user.



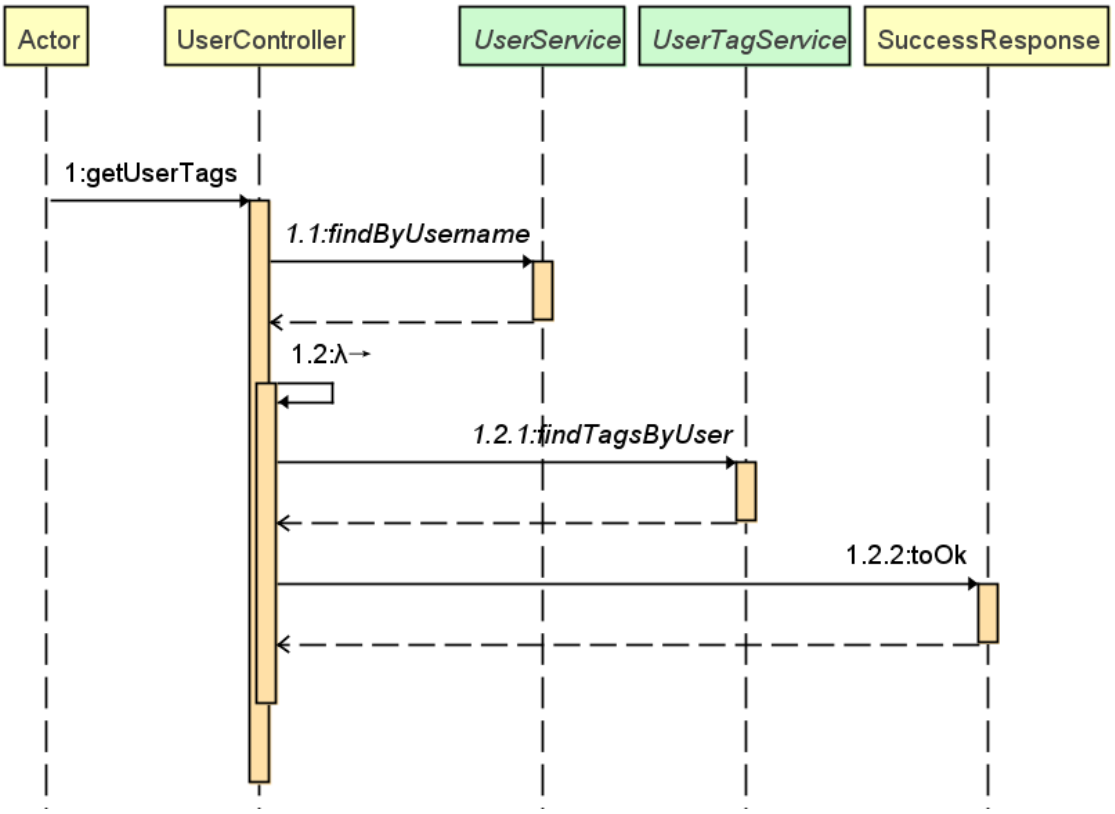
Reset Password

This endpoint helps the user to reset his/her password.



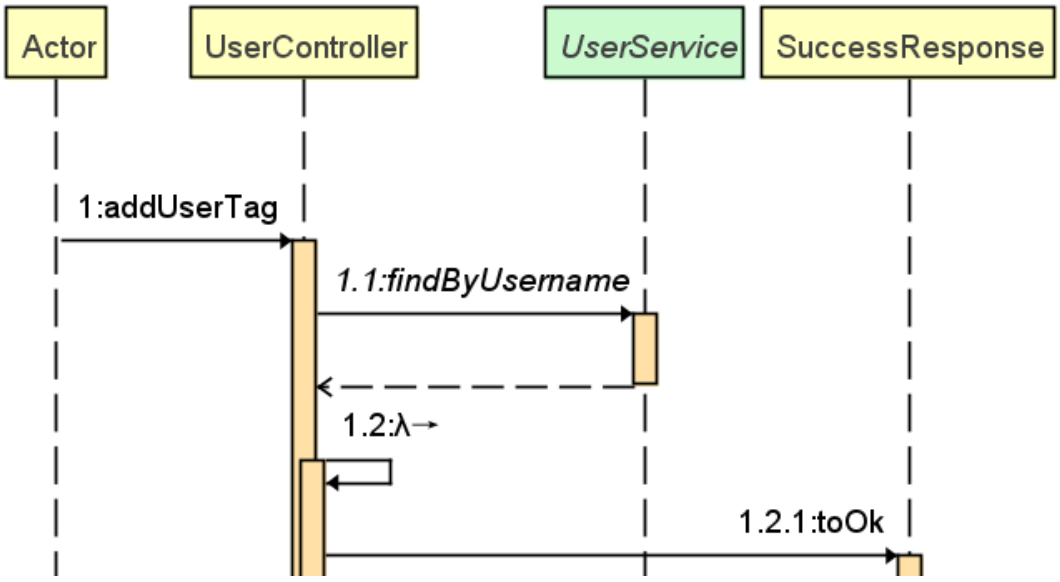
Get User Tags

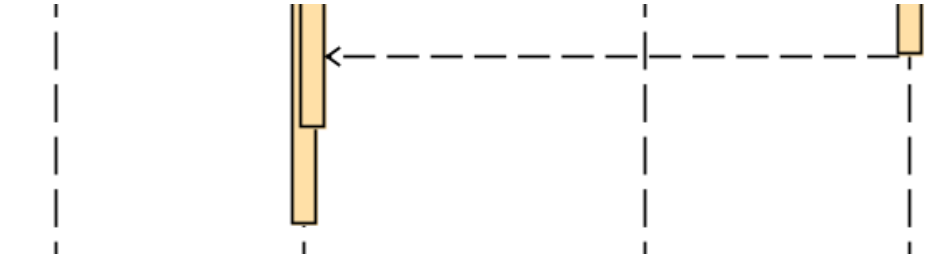
This endpoint returns all of the tags of a user.



Add User Tag

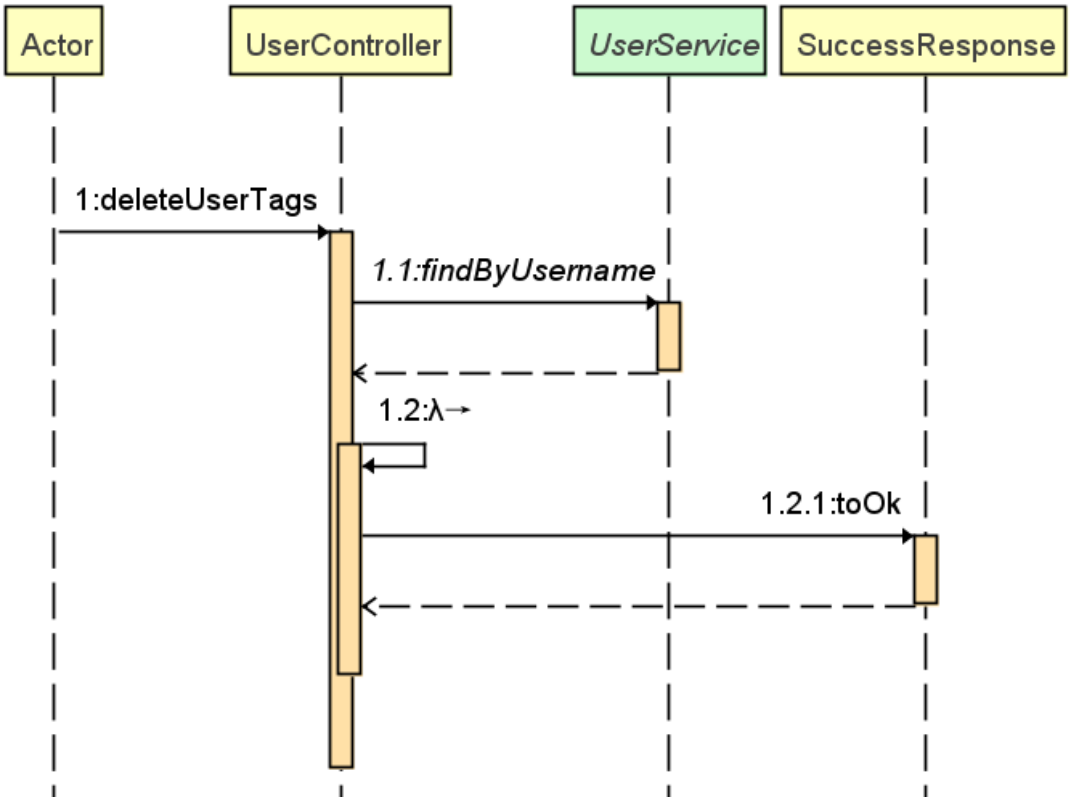
This endpoint helps the user to add a user tag.





Delete User Tags

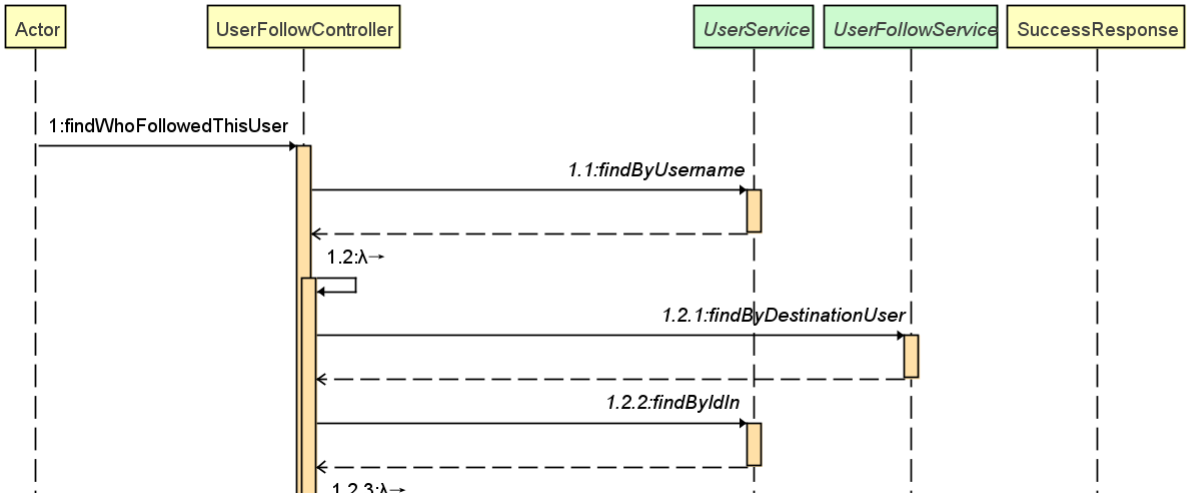
This endpoint helps the user to delete a user tag.

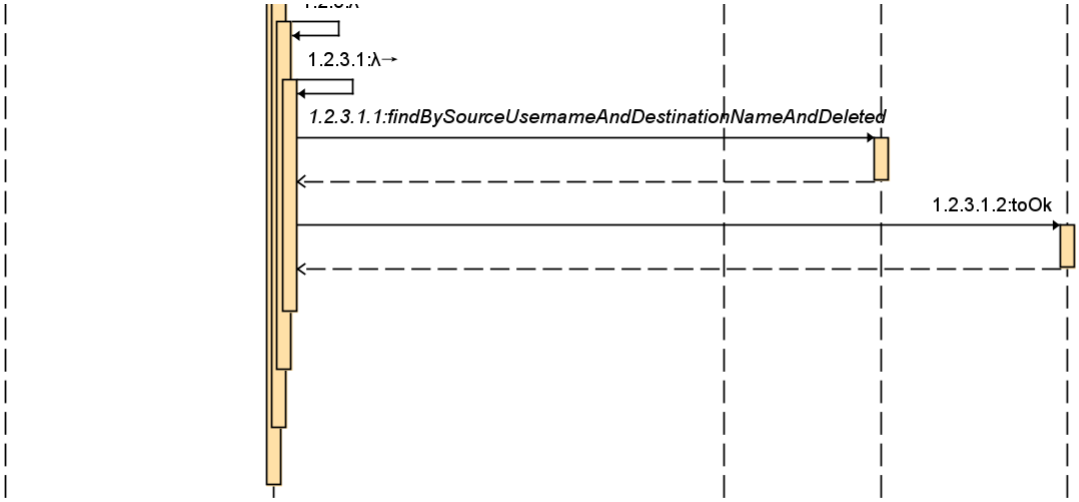


UserFollowController

Find Who Followed This User

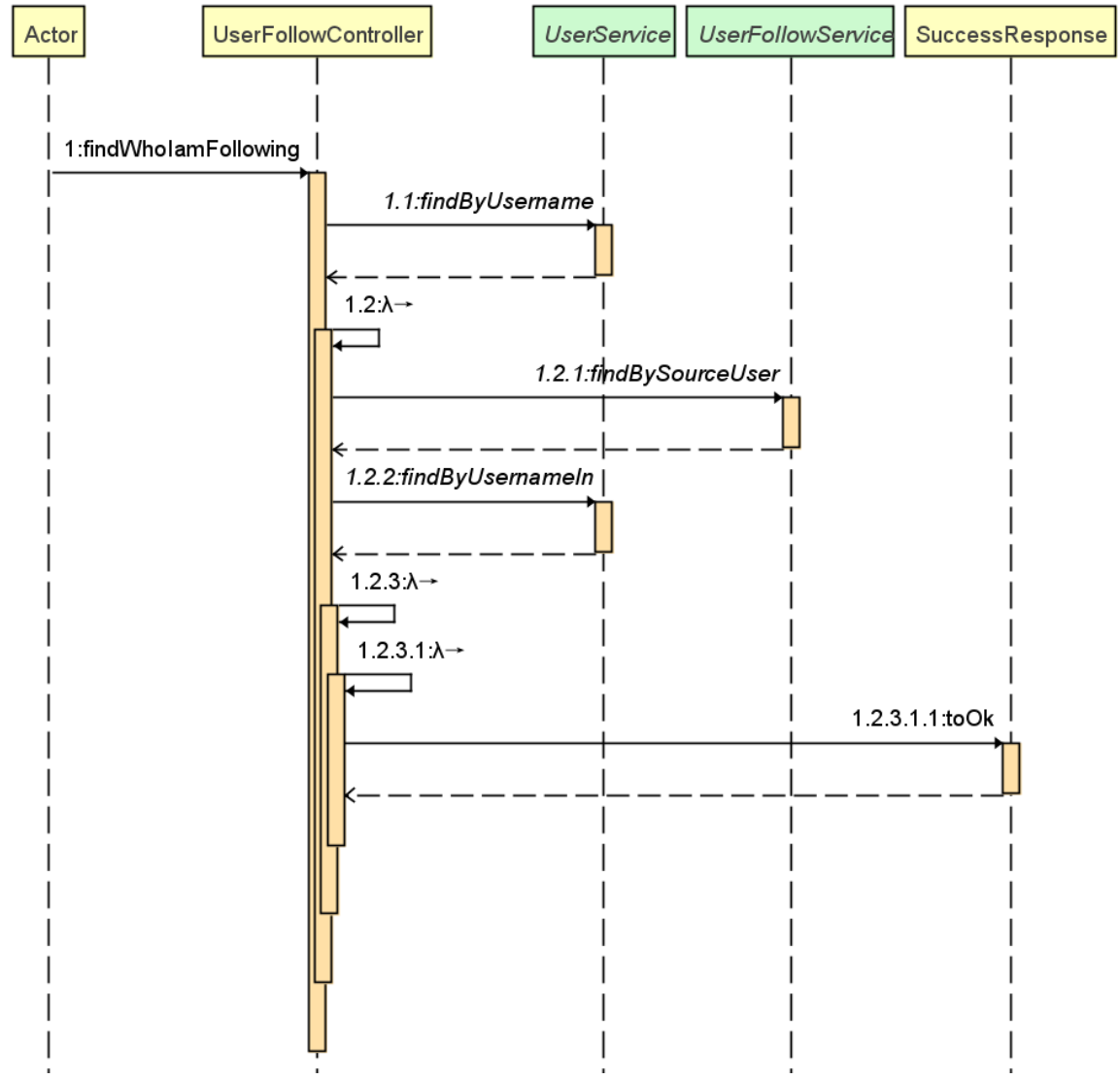
This endpoint helps the user to find out the users who are following him/her.





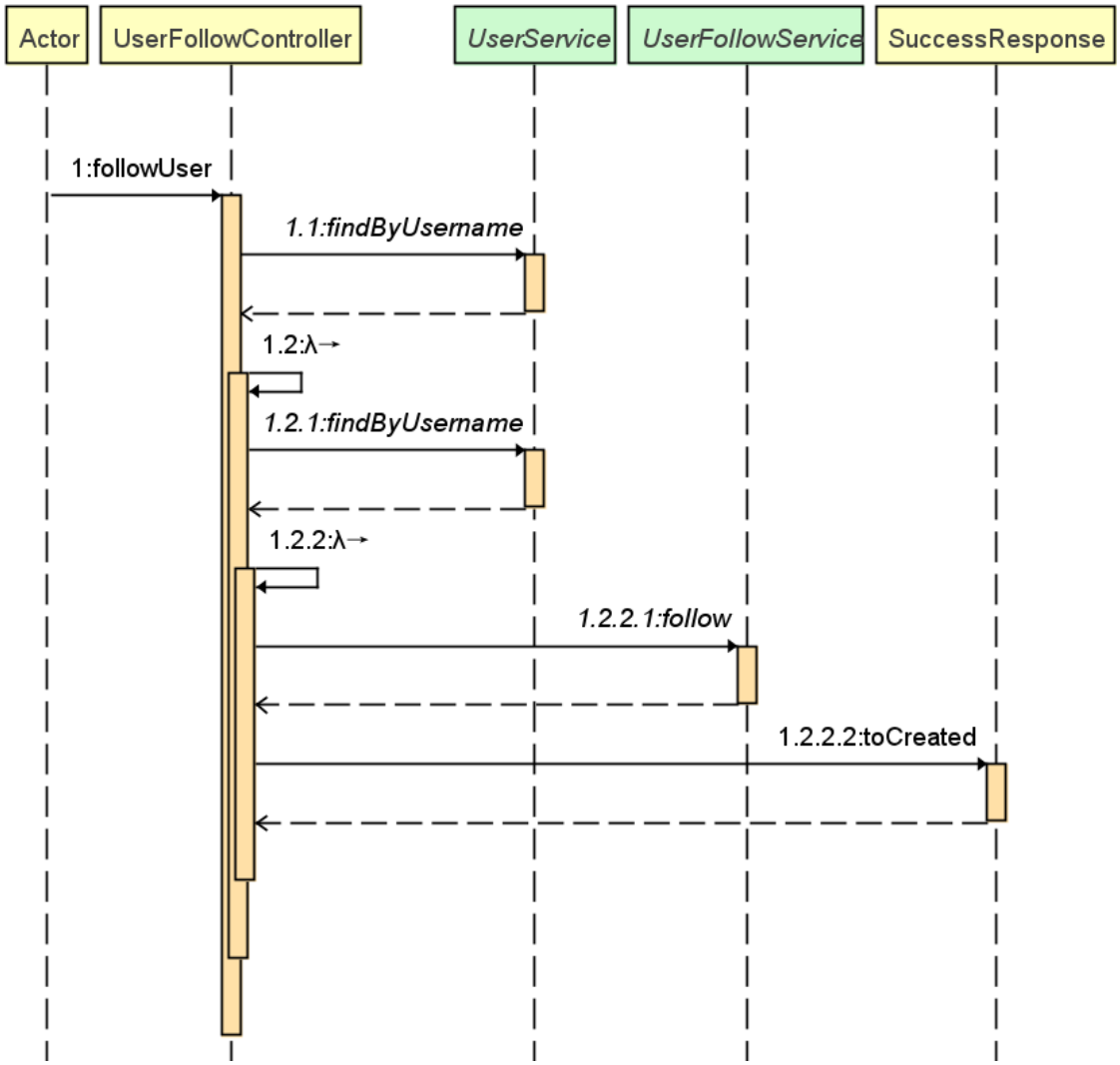
Find Who I am Following

This endpoint helps the users to check out the users he/she is following.



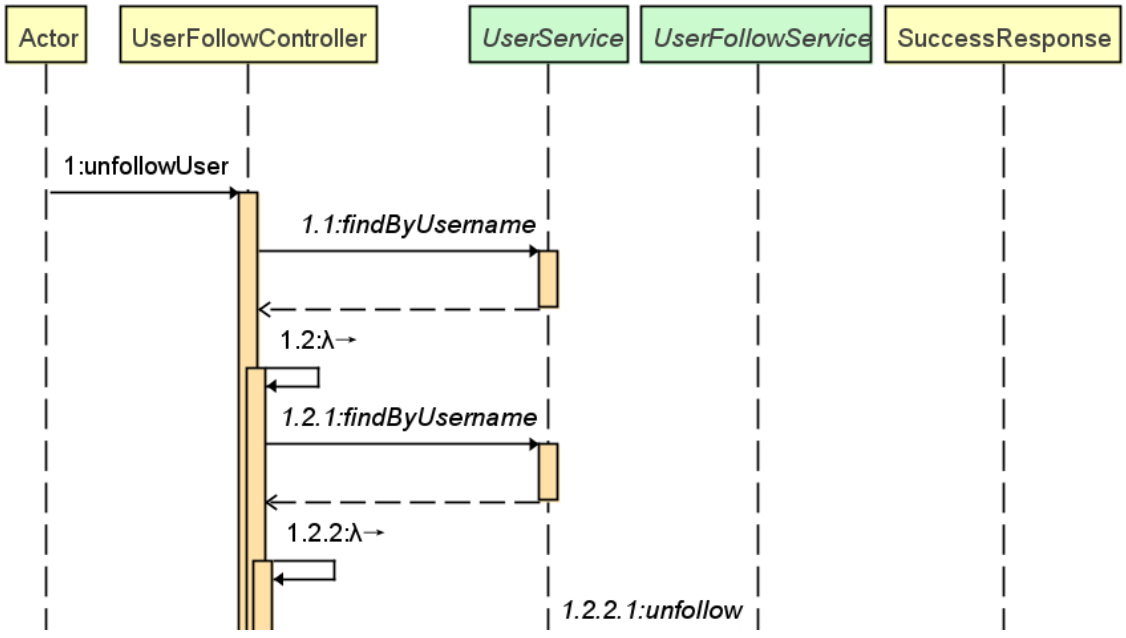
Follow User

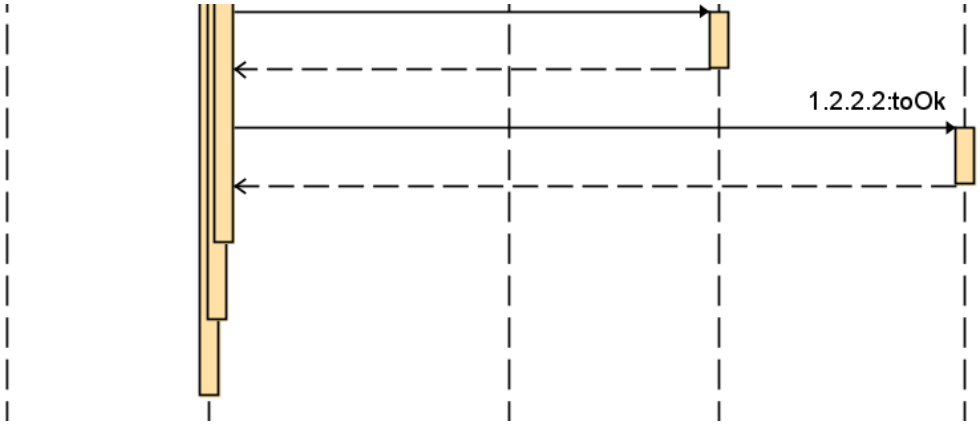
This endpoint helps the user to follow a user.



Unfollow User

This endpoint helps the user to unfollow a user.

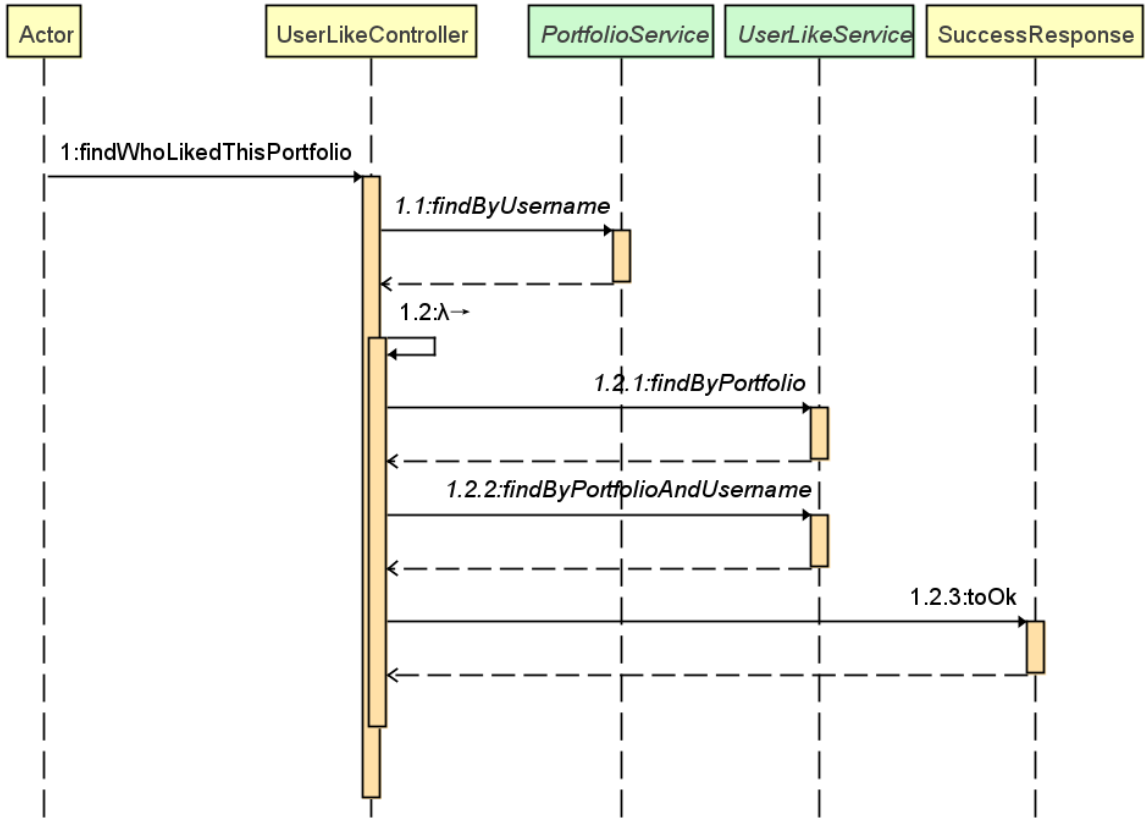




User Like Controller

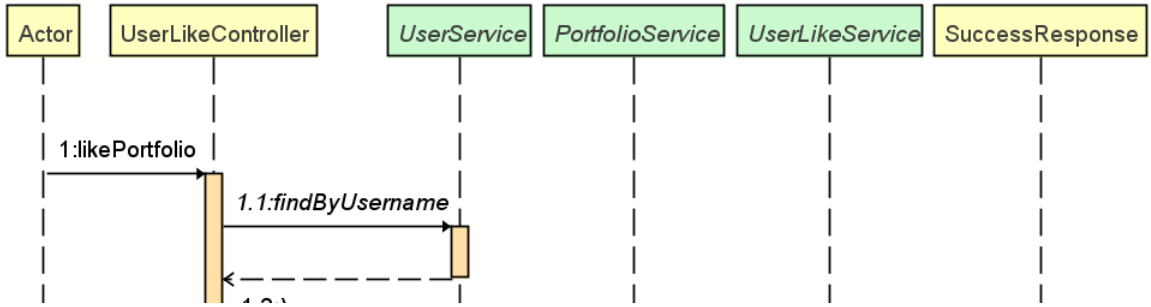
Find Who Liked This Portfolio

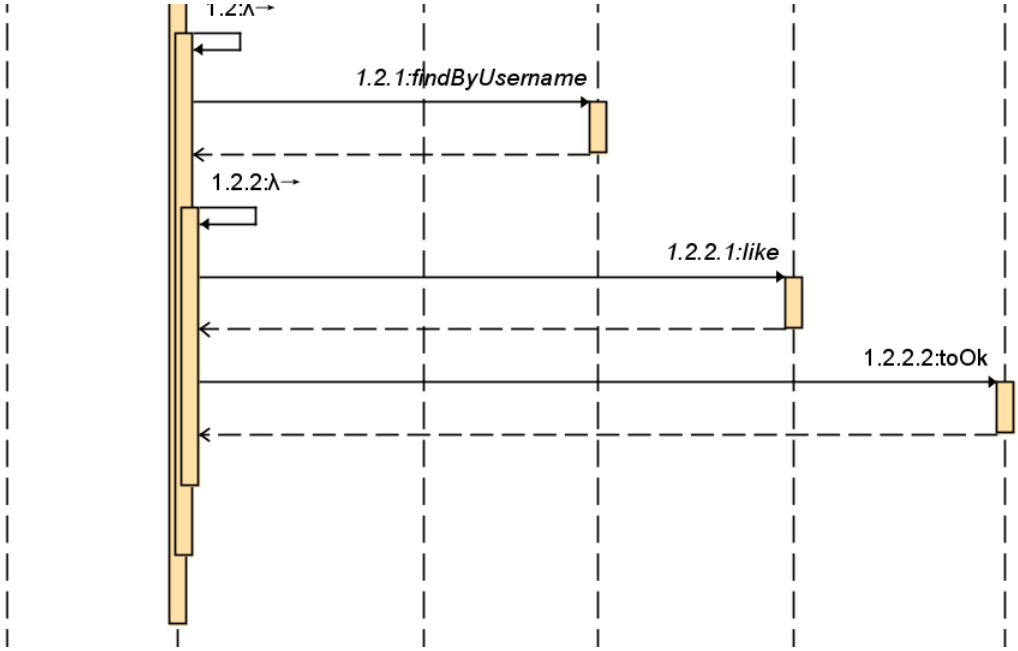
This endpoint returns the users who liked the e-portfolio.



Like Portfolio

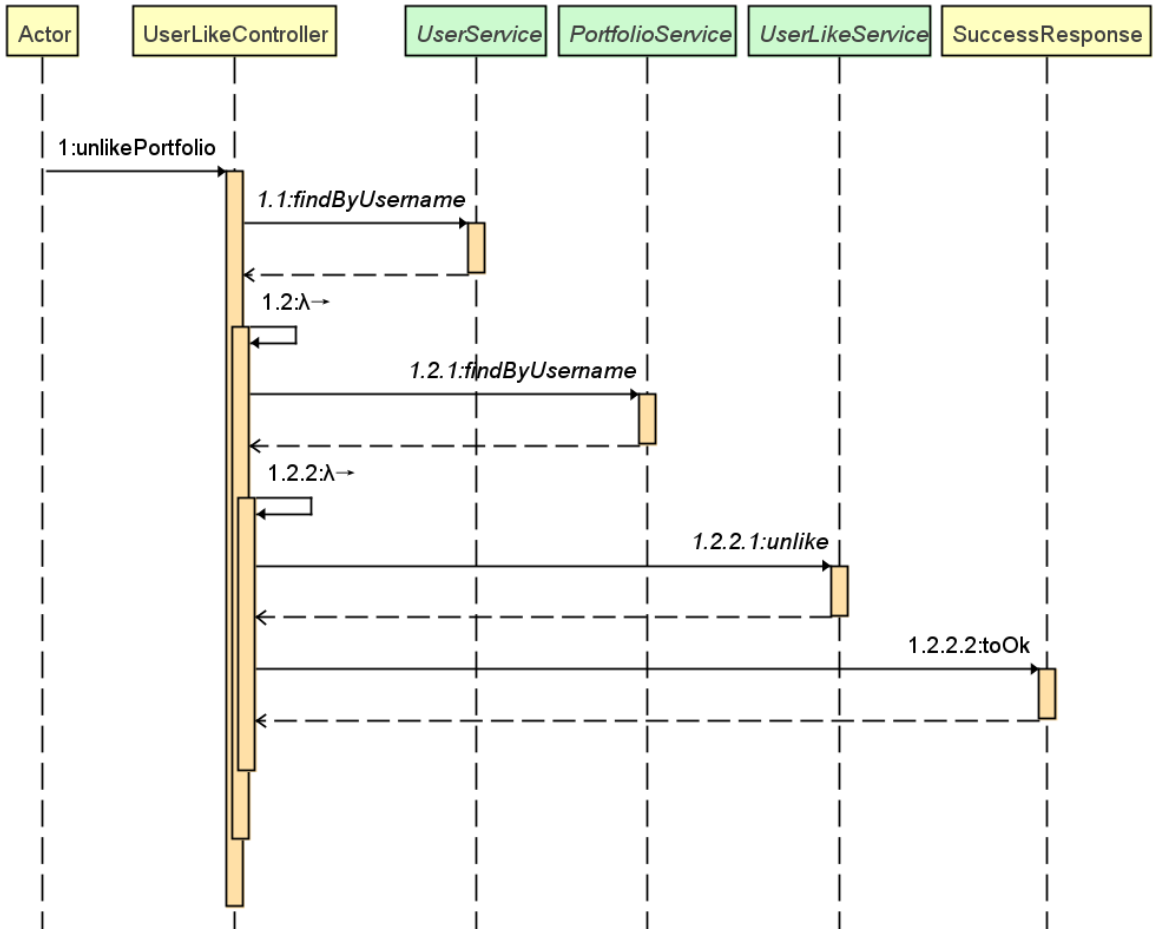
This endpoint helps to user to like an e-portfolio.





Unlike Portfolio

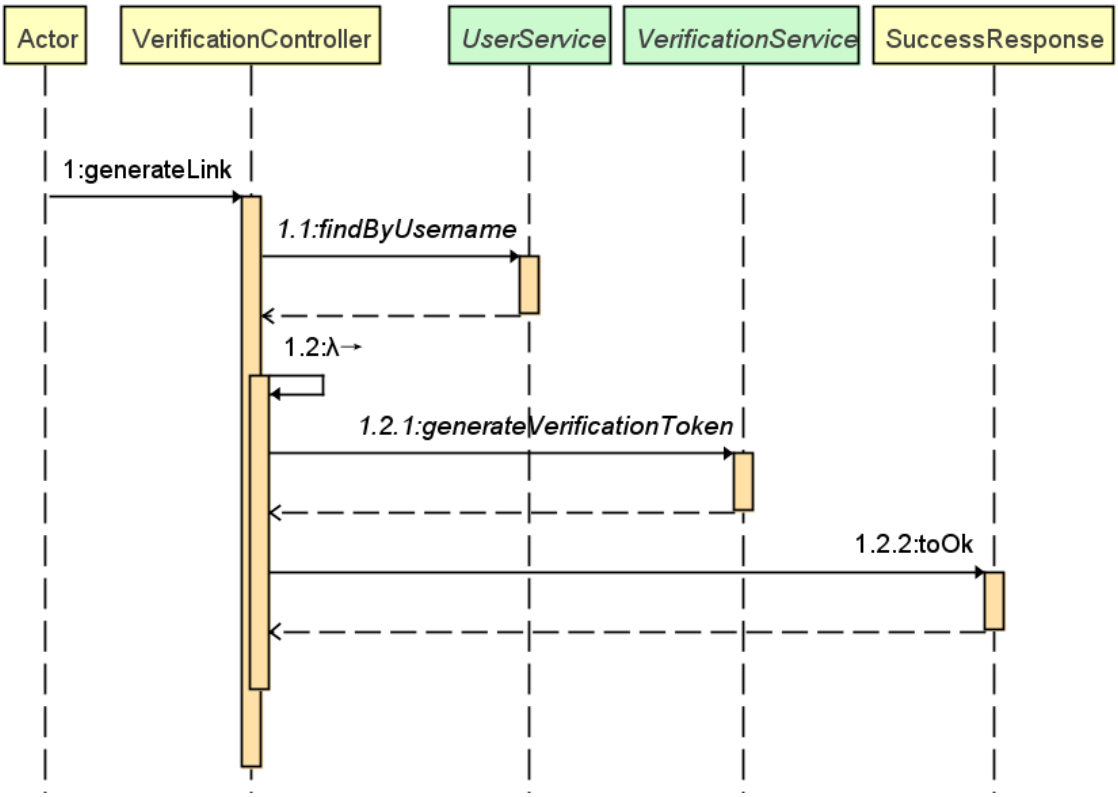
This endpoint helps the user to unlike an e-portfolio.



Verification Controller

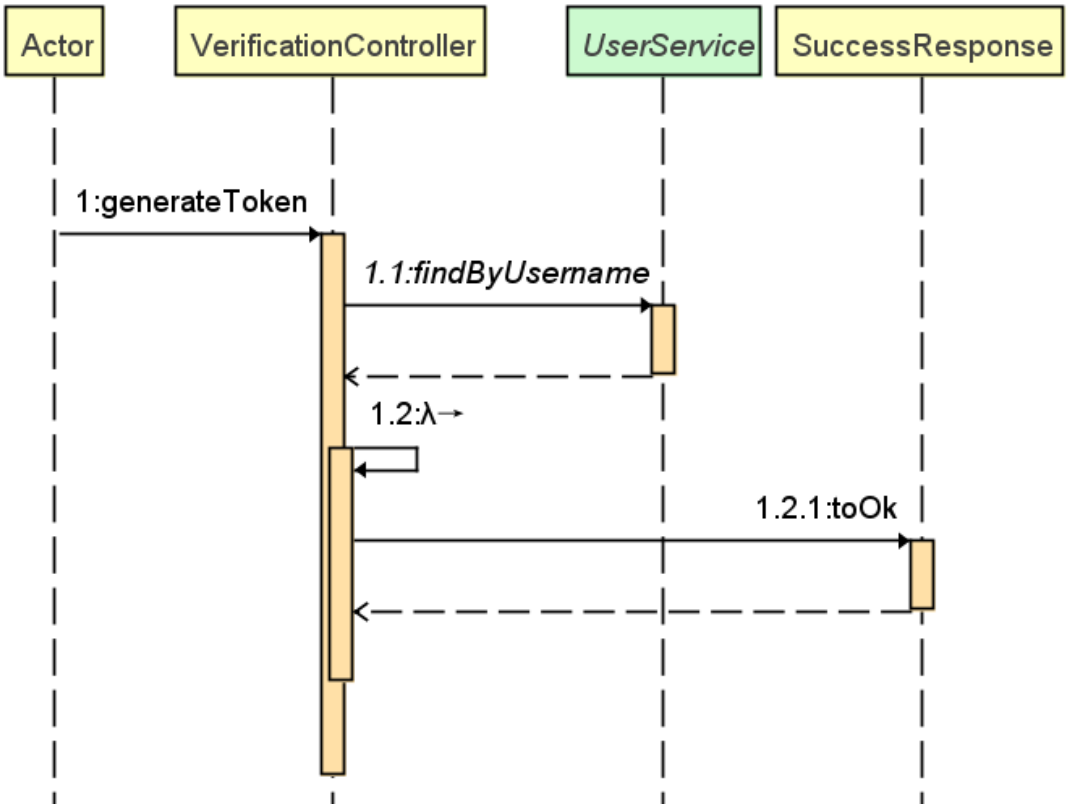
Generate Link

This endpoint helps the user to generate a verification link that used to verify his/her email. (only for testing)



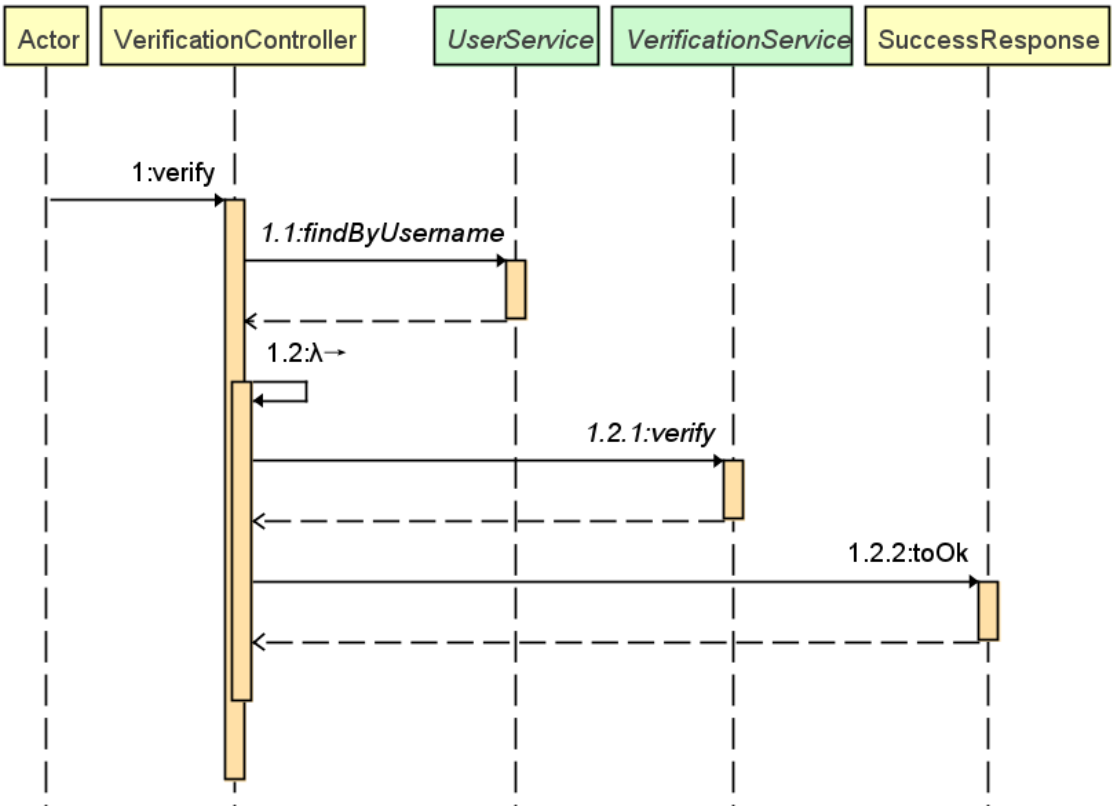
Generate Token

This endpoint helps the user to generate a verifying token to verify his/her email. (only for testing)



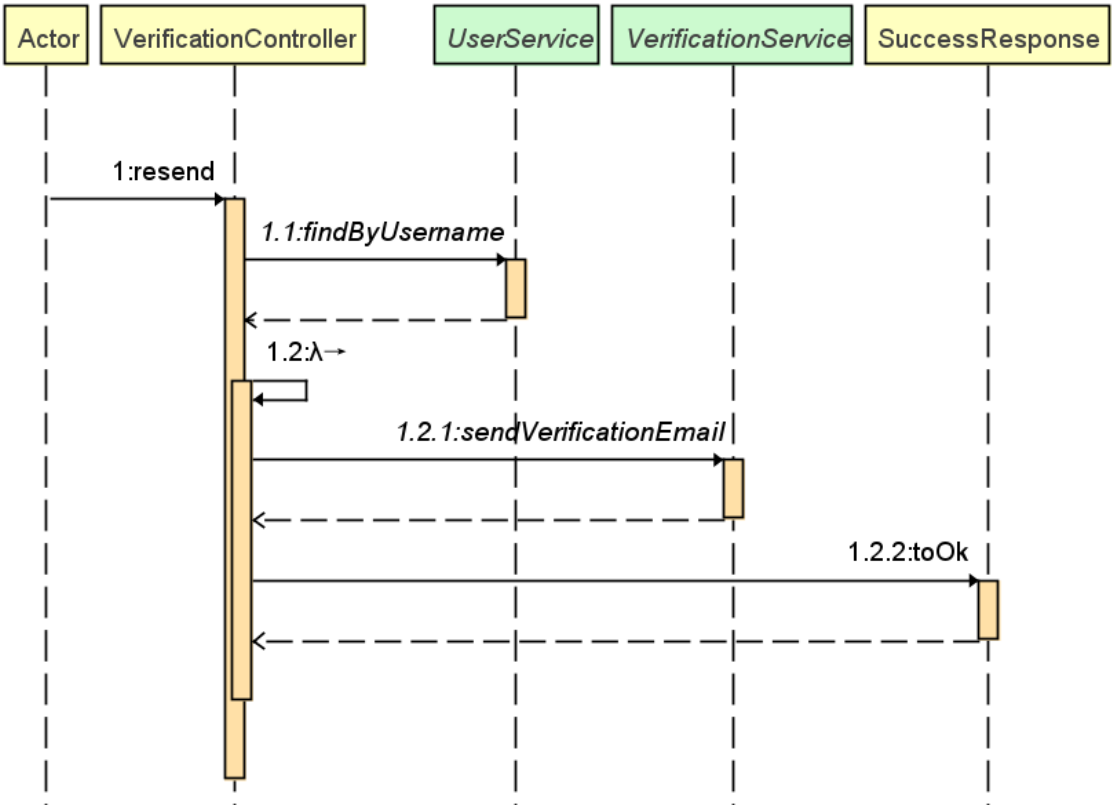
Verify

This endpoint helps the user to verify his/her his/her email with a verifying token.



Resend

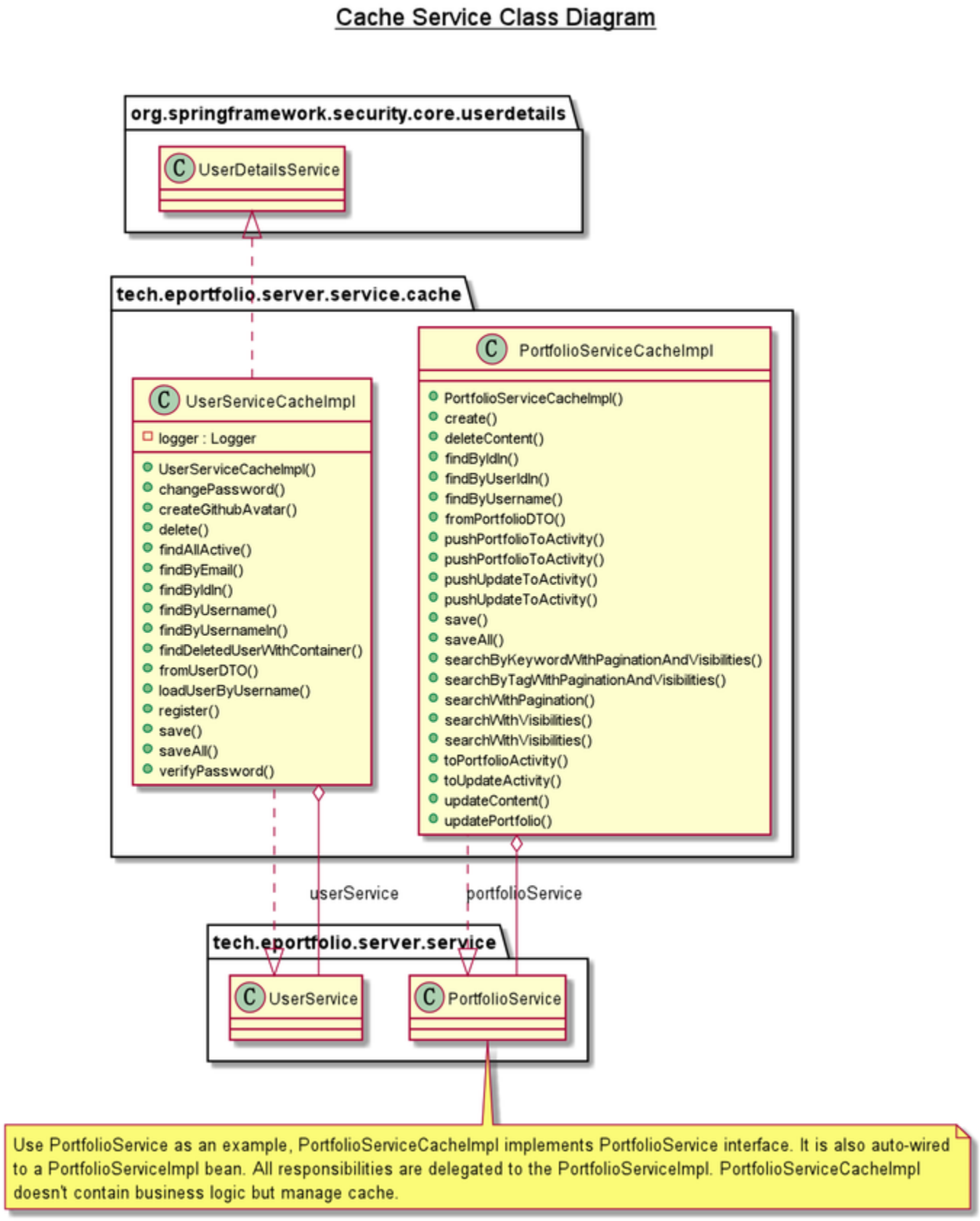
This endpoint sends an email to the user that includes a link to verify the email.



Logical View

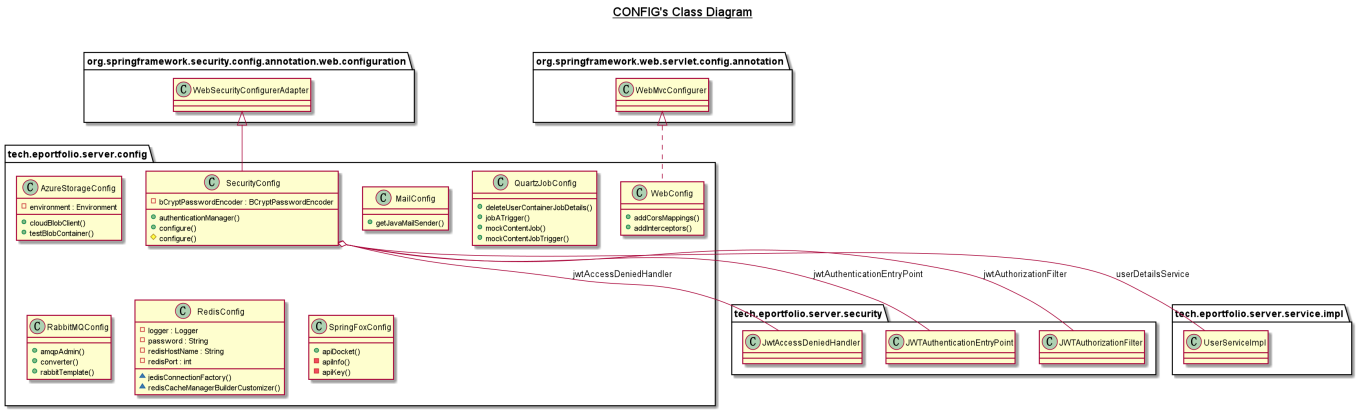
Cache Service Class Diagram

The cache service diagram shows that the original user and portfolio service implementations are replaced by the high-performance cache implementation which can save them into Redis Cache. The cache option can be simply changed as they all implement the service interface.



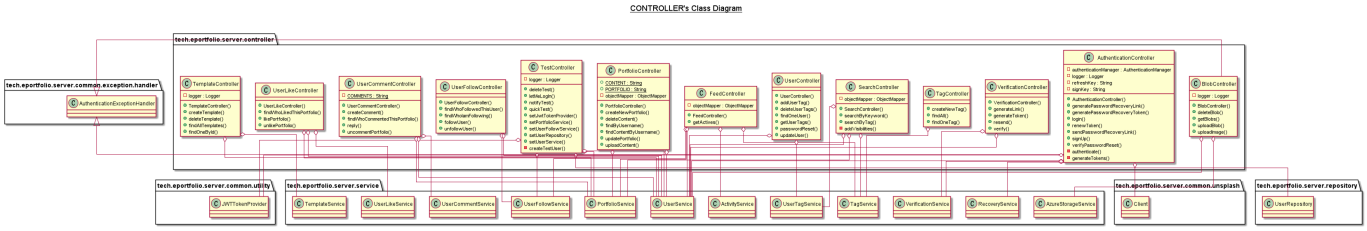
Config Class Diagram

The config class diagram shows the storage of various configurations, such as web config (connect to Swagger, bind Ip addresses) or Redis config (build Redis Cache manager).



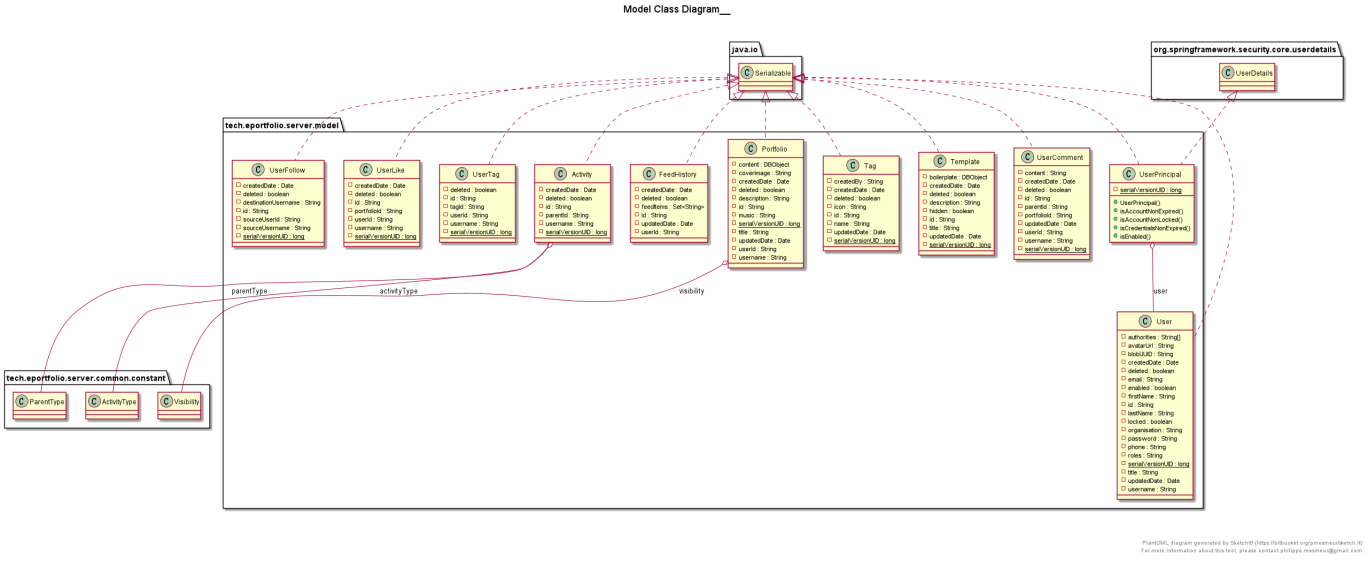
Controller Class Diagram

It can be clearly seen that controllers are above the service layer. After the controller layer is called, it will then call the method implemented in the service layer.



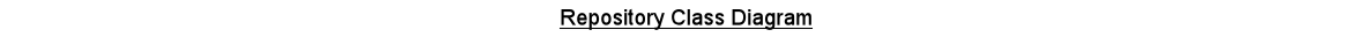
Model Class Diagram

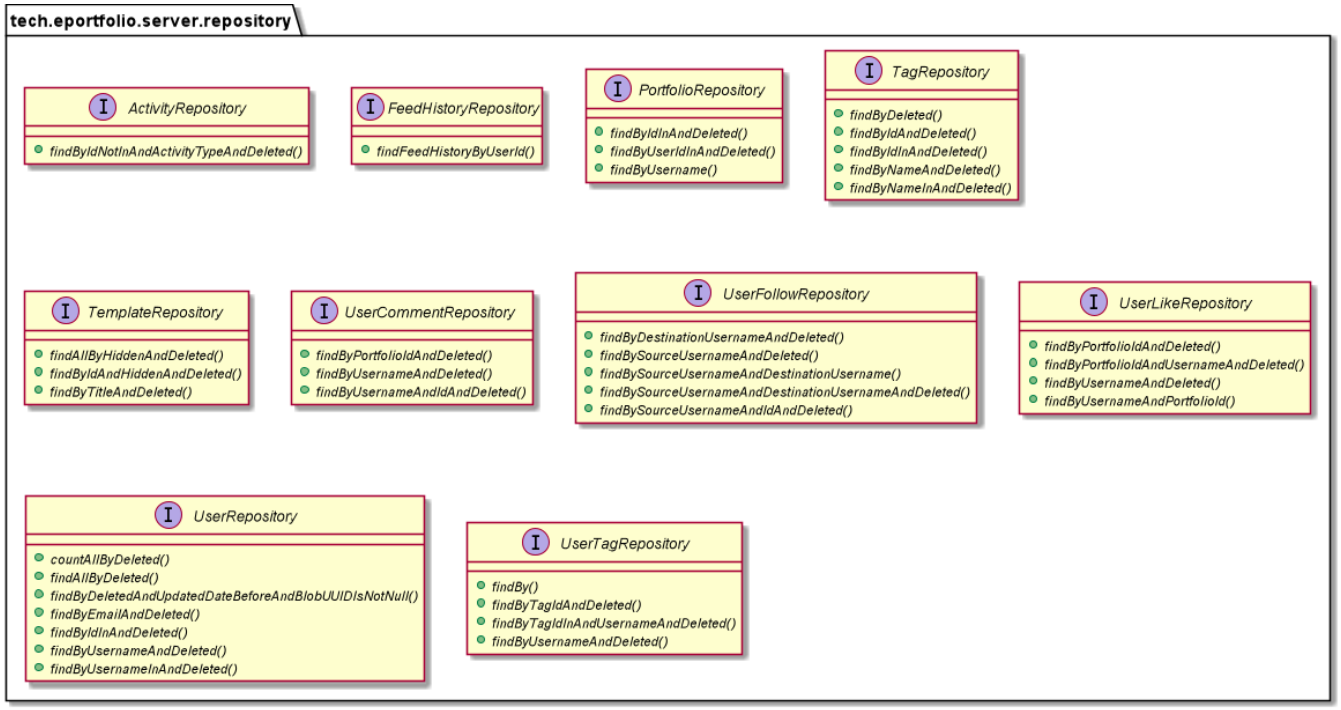
The model class diagram shows how we create the database and which data is stored in it. One should be realized that models are serialized to be able to store data into JSON files.



Repository Class Diagram

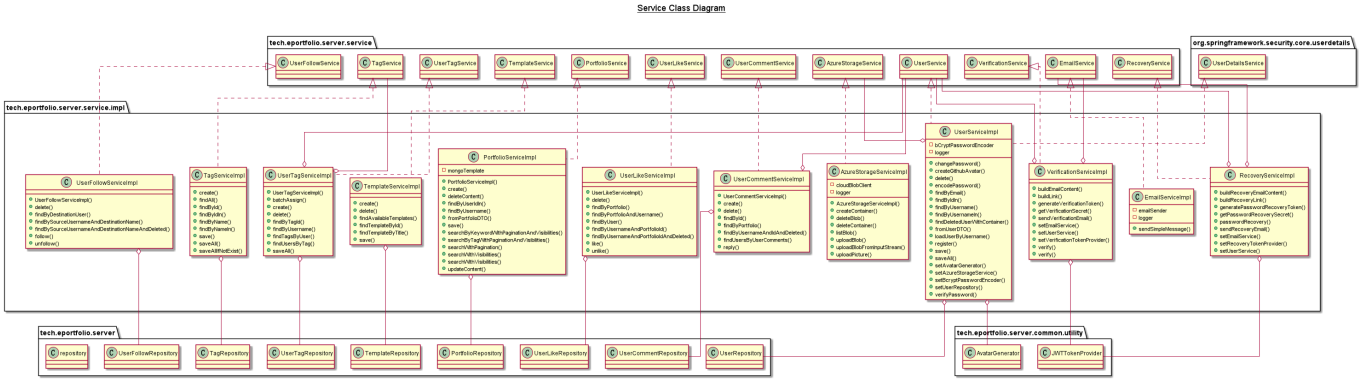
A repository class diagram aims to avoid typing SQL query codes. It replaces the complex codes to clear Java method (e.g. select user_id from user => findbyId()).





Service Class Diagram

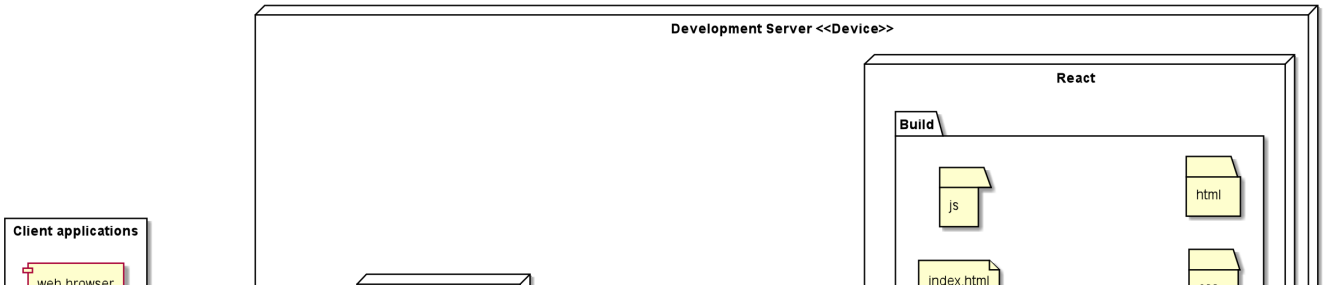
As mentioned in the controller class diagram, the task for the service layer is to call the algorithm of functionalities such as log in or sign up for the controllers. The typical process showed in the figure is that when an API is called, the controller will choose the correct algorithm (a.k.a service) to run which means the controller passes the job into the service layer. Then the service implementation may need to check information in the database.

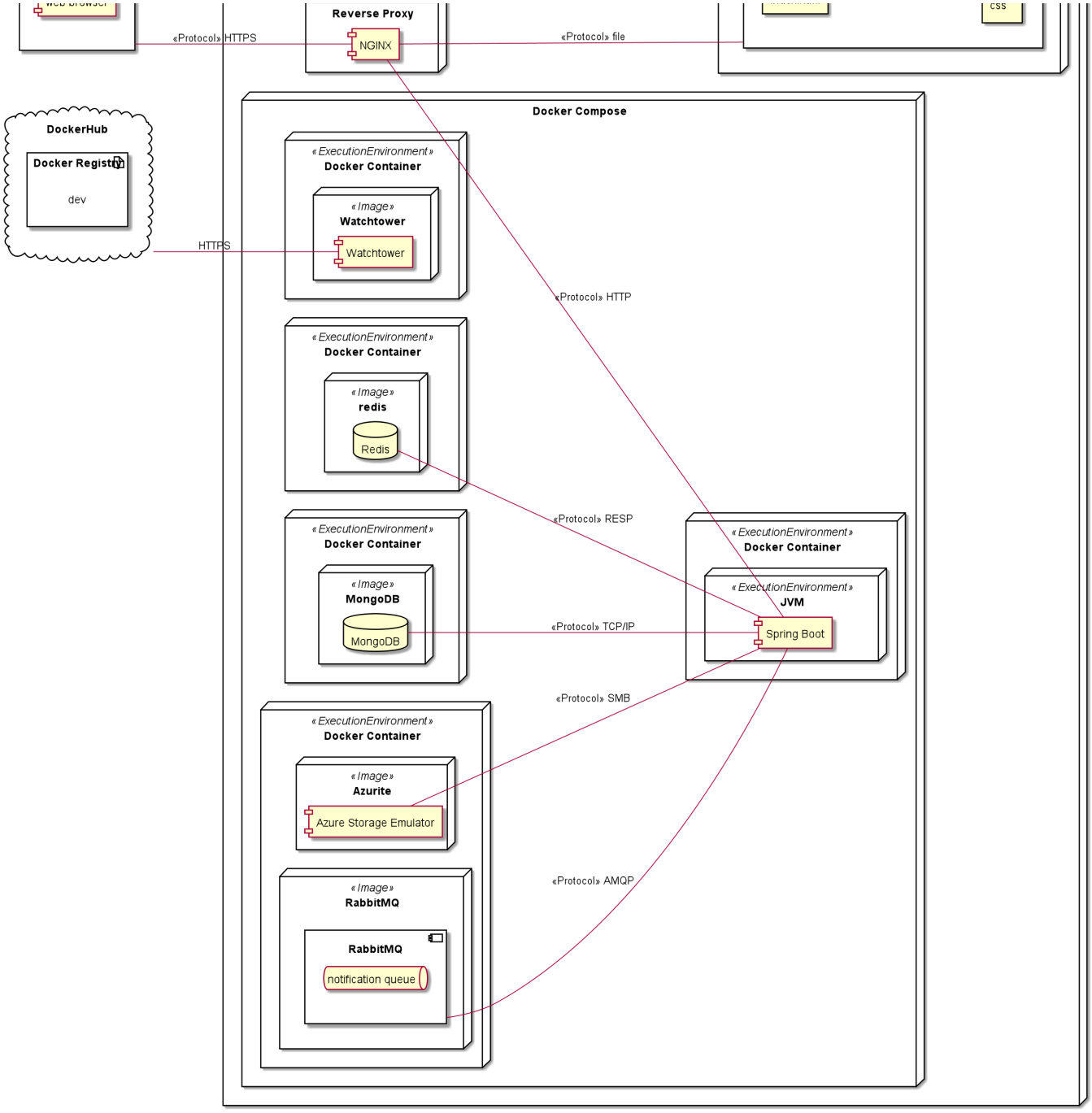


Physical View

Development Deployment Diagram

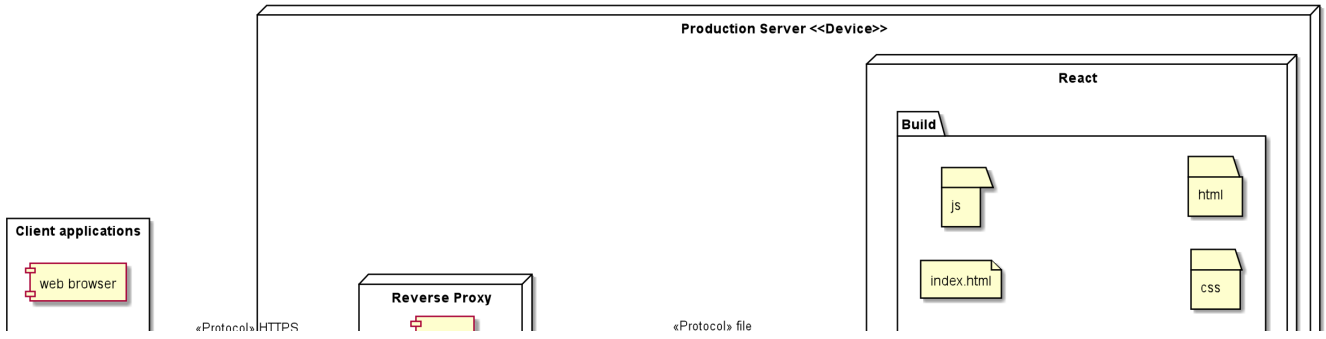
This diagram illustrates the process and the method we deploy our website into the development environment. Nginx is used for reverse proxy and balance loading from the frontend and backend. All services are packaged into Docker Compose so that they can start simultaneously by a simple click.

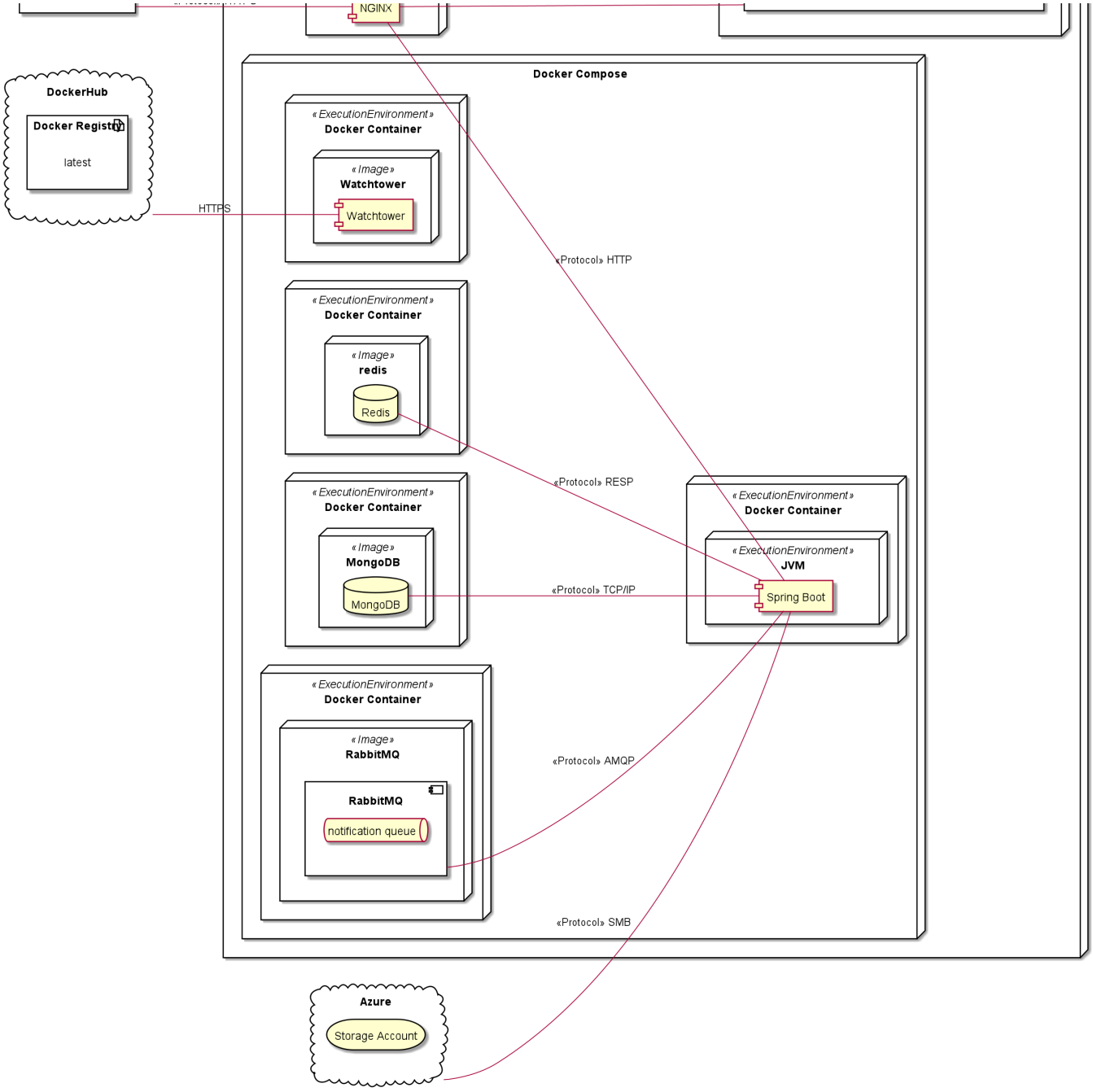




Production Deployment Diagram

This diagram illustrates the process and the method we deploy our website into the production environment. The main difference is that in the production environment Azure is an external service.




NGINX
Docker Compose
DockerHub
Docker Registry
latest
HTTPS
«ExecutionEnvironment»
Docker Container
«Image»
Watchtower
Watchtower
«Protocol» HTTP
«ExecutionEnvironment»
Docker Container
«Image»
redis
Redis
«Protocol» RESP
«ExecutionEnvironment»
Docker Container
«Image»
MongoDB
MongoDB
«Protocol» TCP/IP
«ExecutionEnvironment»
Docker Container
«ExecutionEnvironment»
JVM
Spring Boot
«ExecutionEnvironment»
Docker Container
«Image»
RabbitMQ
RabbitMQ
notification queue
«Protocol» AMQP
«Protocol» SMB
Azure
Storage Account