Final Report - Team 22c

1 FINAL LIST OF REQUIREMENTS IMPLEMENTED

The final implementation of User Microservice delivers a set of features, meeting all specified requirements. Users can sign up, sign in, and edit their profiles, including details like name, bio, and favorite books. The system also supports various user roles, such as admins, regular users, and authors, each with different permissions. Users can change passwords, admins have control over book collections and user management, and authors can add their own books. Search options allow users to find other users and books based on various criteria. Social interactions are facilitated through following and unfollowing by users. Users can delete, deactivate, and reactivate their accounts. The system collects analytics, including review and comment numbers, last login, and follower counts. Users can also access and view their own analytics.

- Allows users to sign up for accounts
- Allows users to sign in to their accounts
- Allows users to edit info such as
  - Name
  - Bio
  - Profile picture
  - Location
  - Favorite book genres
  - Favorite book
- Allows users to assume certain roles
  - Admins
  - Regular users
  - Authors
- Allows users to change their password once logged in
- Allows admins to add, edit and remove books from collections
- Allows admins to remove / ban users
- Allow authors to add their own books
- Allows users to search other users based on criteria:
  - Name
  - İnterests
  - Favorite books
  - Connections
- Allows users to follow / unfollow other users
- Allows users to delete accounts
- Allows users to deactivate accounts
- Allows users to reactivate accounts
- Collects analytics on:
  - Review Number
  - Comment Number
  - Last Login
  - Followers Number
  - Following Number
- Allows user to see their own analytics

2 DESIGN PATTERN IMPLEMENTATION

2.1 Chain Of Responsibility

After carefully looking at all of the design patterns provided in the lecture slides, as well as many more, we have decided to choose two of them: Chain Of Responsibility and Strategy. This choice was crucial to the development of our system, as we wanted a design that would allow us to follow best practices and keep our application flexible and extensible.

First of all, Chain Of Responsibility allows us to create a processing pipeline that can be modified and extended along the production of our microservice, without generating new errors or the need to refactor code. Each node of the pipeline addresses one specific responsibility such as: verification of user credentials, checking if the fields passed on in the requests are valid, existence of objects in database, and many different regulations that can be introduced at any point of the development process. Moreover, by using this design pattern, we ensure that updates to one part of the code don't affect the uncoupled components as the nodes of the chain are oblivious to neighbors, as this is abstracted via an abstract parent class that handles the next node.

We have decided to implement this design pattern for different types of requests, therefore for each type of object (Book, User, Analytics), we have different request handlers with their own specific abstract and concrete implementations.

Firstly, the handlers (or validators) concerning user data are the concrete implementations of the following schemes.

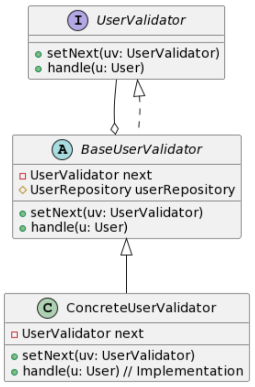
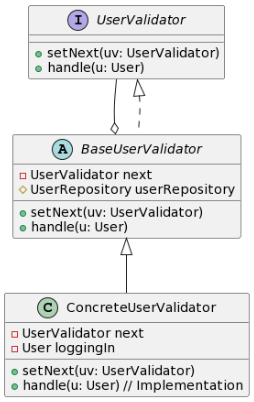

I UserValidator
setNext(uv: UserValidator)
handle(u: User)
A BaseUserValidator
UserValidator next
UserRepository userRepository
setNext(uv: UserValidator)
handle(u: User)
C ConcreteUserValidator
UserValidator next
setNext(uv: UserValidator)
handle(u: User) // Implementation

Figure 1: All handlers besides PasswordValidator scheme



I UserValidator
setNext(uv: UserValidator)
handle(u: User)
A BaseUserValidator
UserValidator next
UserRepository userRepository
setNext(uv: UserValidator)
handle(u: User)
C ConcreteUserValidator
UserValidator next
User loggingIn
setNext(uv: UserValidator)
handle(u: User) // Implementation

Figure 2: PasswordValidator scheme

The first one is the implementation of all validator classes besides PasswordValidator, which follows the second one. The requests of the handlers are User objects that are created using the following.

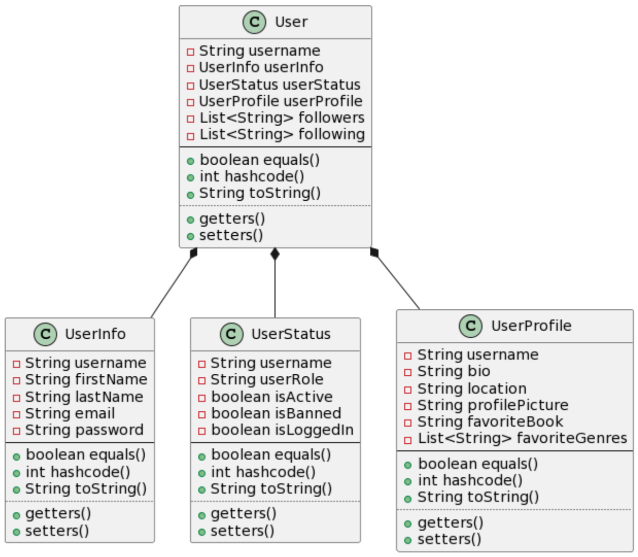


Figure 3: User Class

By glancing over the "createUser" method that is called upon using the user creation endpoint (localhost:8080/user/), we can see the logic behind this design pattern. It uses an ExceptionHandler to catch any of the exceptions thrown by the validators, and only if all nodes allow the request to pass, then the request is passed on to the UserService class that takes care of the actual creation of the new user.

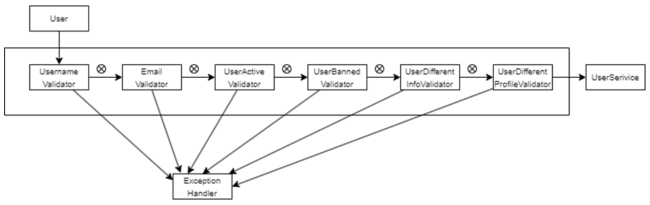


Figure 4: Create User Endpoint Error

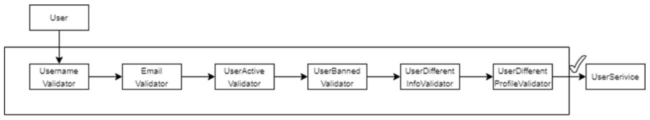


Figure 5: Create User Endpoint

Another endpoint that uses a similar logic is the deactivateSelf (localhost:8080/user/deactivate/username).

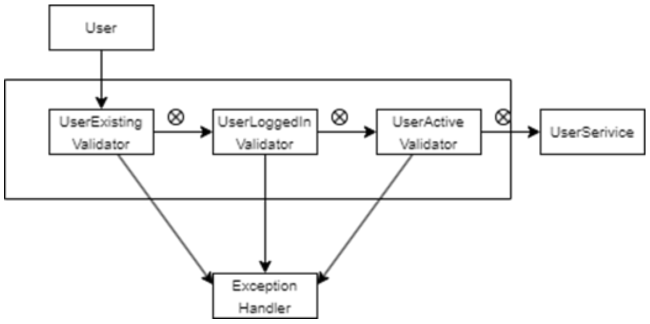


Figure 6: Deactivate Self Endpoint Error

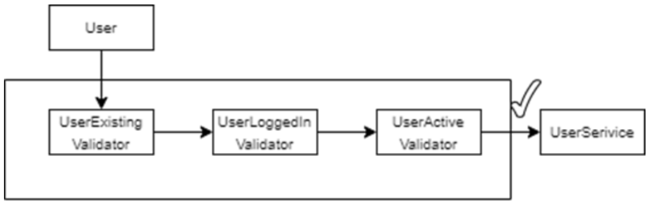


Figure 7: Deactivate Self Endpoint

The commits and merge requests related to the implentation of user validators can be found here:
Implementation in the endpoints: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/89aaa142b8ad98bb9ef62d2af2cdc731d5f61a23,
Validator classes: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/002fbd846c9b417d6c799d8bc4ff19fb791142de,
Test classes for user validators: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/cbba1e7c2cb706ccbf77

9d7bf79d14a7cb52209e.

The concrete implementations of the analytics handlers follow a similar approach, but have different requests (the analytics class).

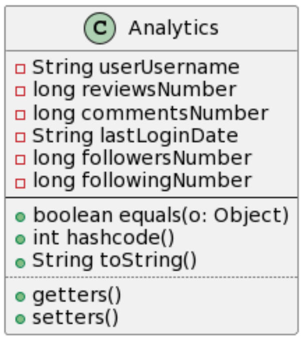


Figure 8: Analytics Class

The first schema concerns the comment, followersNumber, followingNumber and reviewNumber handlers and it has the purpose of checking whether the field values are legal or not (cannot be lesser than 0). The second contains the AnalyticsIDExists and AnalyticsCreationUsername handlers, and the third one category is implemented only by the usernameValidator. The main reason for having multiple schemes is because different nodes check different conditions and some require other objects to be able to handle that request (checking if a username exists in the user database vs. in the analytics database)

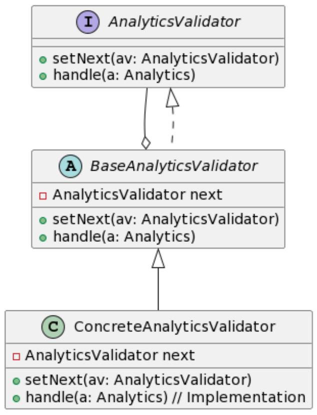

I AnalyticsValidator
setNext(av: AnalyticsValidator)
handle(a: Analytics)
A BaseAnalyticsValidator
AnalyticsValidator next
setNext(av: AnalyticsValidator)
handle(a: Analytics)
C ConcreteAnalyticsValidator
AnalyticsValidator next
setNext(av: AnalyticsValidator)
handle(a: Analytics) // Implementation

Figure 9: First Analytics Validator Schema

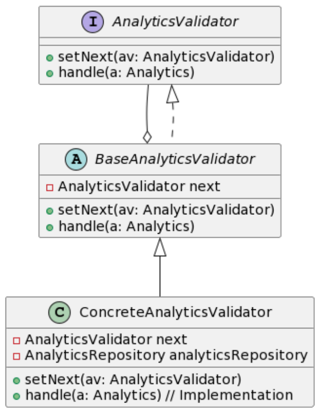

I AnalyticsValidator
setNext(av: AnalyticsValidator)
handle(a: Analytics)
A BaseAnalyticsValidator
AnalyticsValidator next
setNext(av: AnalyticsValidator)
handle(a: Analytics)
C ConcreteAnalyticsValidator
AnalyticsValidator next
AnalyticsRepository analyticsRepository
setNext(av: AnalyticsValidator)
handle(a: Analytics) // Implementation

Figure 10: Second Analytics Validator Schema

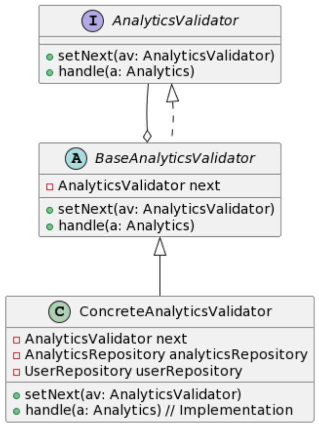

I AnalyticsValidator
setNext(av: AnalyticsValidator)
handle(a: Analytics)
A BaseAnalyticsValidator
AnalyticsValidator next
setNext(av: AnalyticsValidator)
handle(a: Analytics)
C ConcreteAnalyticsValidator
AnalyticsValidator next
AnalyticsRepository analyticsRepository
UserRepository userRepository
setNext(av: AnalyticsValidator)
handle(a: Analytics) // Implementation

Figure 11: Third Analytics Validator Schema

This is the logical scheme for the editAnalytics method that is called whenever we reach the localhost:8080/analytics/username for the PUT request.

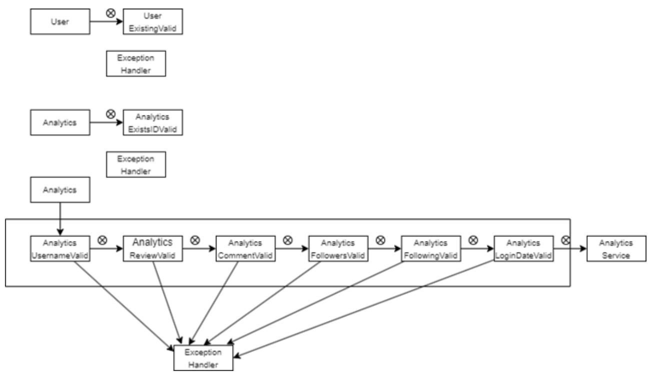


Figure 12: Edit Analytics Endpoint Error

Again, the AnalyticsService that takes care of actually editing the object is only called after all of the checks have been successful, but in this case we are dealing with multiple chains in the same method. This happens because not only do we need to check to see if the user exists, but also if the analytics object passes all of our conditions. The request is successful if and only if all chains terminate.

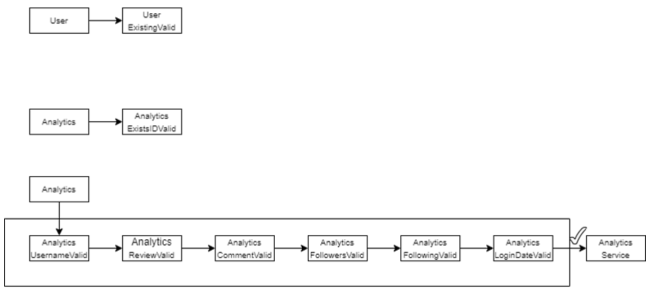


Figure 13: Edit Analytics Endpoint

The commits and merge requests related to the implentation of analytics validators can be found here:
Implementation in the endpoints: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/89aaa142b8ad98bb9ef62d2af2cdc731d5f61a23,
Validator classes: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/566d2710ff04c799e2cffae250dafb44f5971b8c,
Test classes for analytics validators: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/0741198529ffc268e6e80b94dc3d8da6956380dd.

Lastly, the book handlers all follow the same format, and again a different type of request.

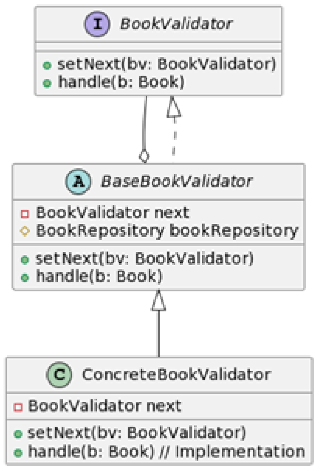

I BookValidator
setNext(bv: BookValidator)
handle(b: Book)
A BaseBookValidator
BookValidator next
BookRepository bookRepository
setNext(bv: BookValidator)
handle(b: Book)
C ConcreteBookValidator
BookValidator next
setNext(bv: BookValidator)
handle(b: Book) // Implementation

Figure 14: Book Validators Scheme

The commits and merge requests related to the implentation of book validators can be found here:
Implementation in the endpoints: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/caa85422e1be344306ec1121a6faa665c3005fc3,
Validator classes: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/8dba24e0d84ff2dbc11aae3e3675b17842561d10,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/6b18225550a6a003c93dd2c3b8cf1be73f6672ee,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/8ae7824b88fcffc822842eef34836919498a3c2b,
Test classes for book validators: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/64ec9b403a38283728bc37694d6a88995bc0270d, https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/620b1327f5085205f5e71c19b4ed55bf778802b3.

One flaw that might arise by using Chain of Responsibility is by not being properly accustomed to this design patter, a request might be unhandled if the chain of validators in not configured properly, thus bringing the entire program to a halt.

2.2 Strategy

Our second choice was using the Strategy pattern in the context of the verification of the user role. This pattern allows us to find a systematic way of defining and encapsulating a family of procedures, as well as making them interchangeable. At this point of development, our application only has 2 different roles: the administrator role and the author role. In the future, as we will expand our functionalities, by having an organized and flexible design we will be able to easily create new classes that implement verification for multiple roles, all encapsulated in our original code. This means that we get to reuse code, have a better maintainability and flexibility of our program and also support dynamic selection of the needed implementation of the role-checking algorithm. For now, the design follows the scheme presented here.

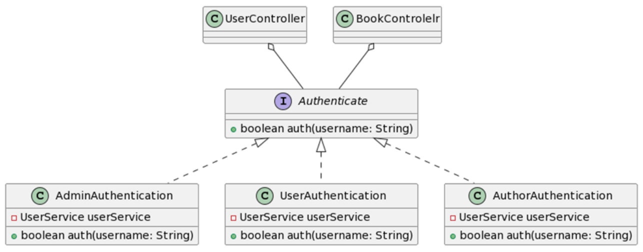


Figure 15: Strategy Pattern

Whenever a book is created, or edited, we call the AuthorAuthentication strategy that checks whether an user is capable of this action. This checking takes place in the User part of things, where we check if an user is an admin or not. This then allows us to make decisions that let the user delete objects and edit them (ban or deactivate users).

The commits and merge requests related to the implentation of the Authentication Strategy can be found here:
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/d6a3c52b51d52f8f2e74ac8116ae8fce8ef165bd,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/8e40d5fda0caa2160873a51e8588dbcc9beb0dac,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/79e9bb107527b6f1c782551f0d89b10afdc814e5,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/12ff593753df56644ceedbddbd71f8061274c000,
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/1fbd0bbcd39f81ef9aacb80982a68eab65c889f3.

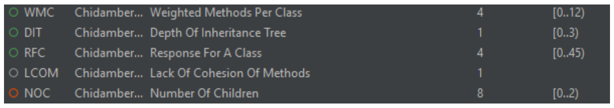
However, there are a couple of contraries of using this design pattern: the implementation must be aware of the context to decide upon a concrete class to use, as well as there being new objects that are created, making the code more complex.

3 SOFTWARE QUALITY REPORT

For the analysis of the quality of our software, we decided to use MetricsTree. When first running it a few problems were noticeable in the following classes:

3.1 Base Validators

All three of the Base Validators present a high Number of Children (NOC). This is to be expected as the other validators need to extend this in order to avoid reusing code (boilerplate code). As such, we did not deem this a problem.

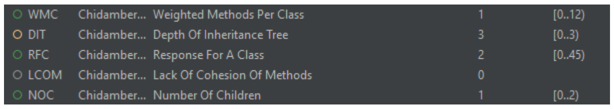


WMC	Chidamber...	Weighted Methods Per Class	4	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	4	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	8	[0..2)

Figure 16: Base User Validator

3.2 Custom Exceptions

All of the custom exceptions present a Depth of Inheritance of 3. This cannot be lowered as they need to extend the Exception class.

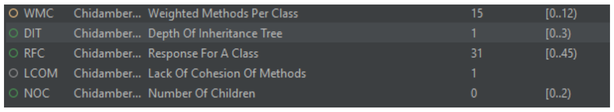


WMC	Chidamber...	Weighted Methods Per Class	1	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	3	[0..3)
RFC	Chidamber...	Response For A Class	2	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	0	
NOC	Chidamber...	Number Of Children	1	[0..2)

Figure 17: Invalid User Exception

3.3 Analytics Controller

The Analytics Controller class has a medium number of WMC. As there are no other metrics that indicated something wrong with the class and it would be very hard to split since it has quite a targeted functionality we decided not to refactor it, although that is something that could be done in the future.

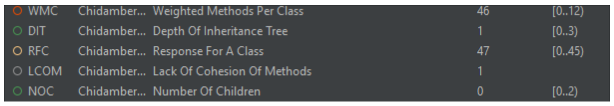


WMC	Chidamber...	Weighted Methods Per Class	15	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	31	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 18: Analytics Controller

3.4 Book Controller

The Book Controller class has a very high WMC and medium RFC. This class should be refactored in the future but we decided to focus on more pressing matters.

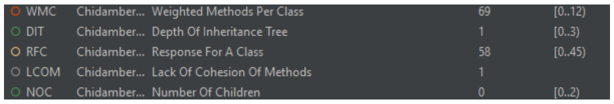


WMC	Chidamber...	Weighted Methods Per Class	46	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	47	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 19: Book Controller

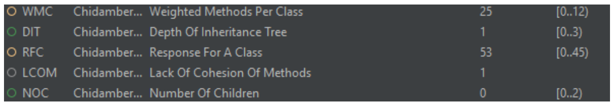
3.5 User Controller

The User Controller class had a very high WMC and medium RFC. This revealed that the class was way too big and needed to be split up. We decided to move some of its methods into 2 new controllers: Admin Controller and Follow Controller. The result was a significant improvement on the code metrics, while the new classes also presented good behaviour. This can be observed in this commit: https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/ecf563d529b82cdd2a1ffda984aafc6e871b7b35



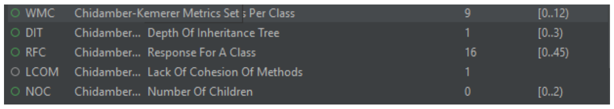
WMC	Chidamber...	Weighted Methods Per Class	69	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	58	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 20: User Controller Before



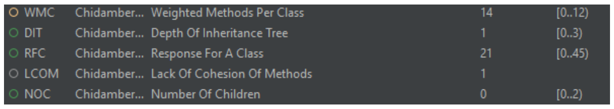
WMC	Chidamber...	Weighted Methods Per Class	25	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	53	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 21: User Controller After



WMC	Chidamber-Kemerer Metrics Set ; Per Class		9	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	16	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 22: Admin Controller



WMC	Chidamber...	Weighted Methods Per Class	14	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	21	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

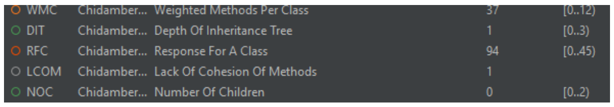
Figure 23: Follow Controller

3.6 Analytics and Book

When it comes to the Analytics and Book classes, they have a medium WMC, but, almost all of their methods are either getters or setters and they do not have enough attributes to apply Extract Class Refactoring. Therefore, we do not think that it is possible to improve this.

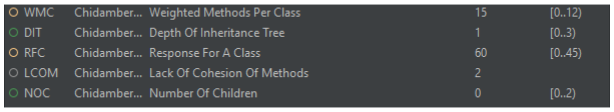
3.7 User Service

The User Service class also had some problems with a high WMC and very high RMC. We decided to move some of its methods to 2 other new services based on their functionality: Follow Service and Search Service. This has massively improved those metrics making the User Service Class easier to understand and maintain. This can be observed in the following commit:
https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/ecf563d529b82cdd2a1ffda984aafc6e871b7b35



WMC	Chidamber...	Weighted Methods Per Class	37	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	94	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 24: User Service Before



WMC	Chidamber...	Weighted Methods Per Class	15	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	60	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	2	
NOC	Chidamber...	Number Of Children	0	[0..2)

Figure 25: User Service After

3.8 User

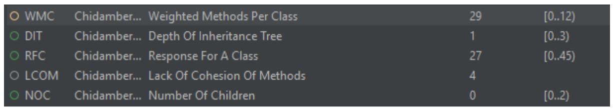
Lastly, in the User class we observed a very high WMC and medium RFC. This indicated to us that the class was too big and needed to be split up. We decided to use Extract Class Refactoring in order to group some attributes and responsibilities of it into other classes. In the end, we created 3 new user utilities classes: UserInfo (which concerns the name, email, password and their related methods), UserProfile (for bio, location, profile picture, favorite book and favorite genres) and the UserStatus(active, logged in, banned, user role). This can be seen in the following commits:

https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/73dd20893688b0d5707e55f95901c0e2178f316a

https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/c8714c9541dc386b531d158ba731fef33d33caee

https://gitlab.ewi.tudelft.nl/cse2115/2023-2024/group-22/team-22c/-/commit/fcfa6498630f7d1ff19af0b838eb0abeb6b0c121

Ultimately, we managed to heavily reduce the WMC and RFC of the User class to acceptable levels:



WMC	Chidamber...	Weighted Methods Per Class	29	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	27	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	4	
NOC	Chidamber...	Number Of Children	0	[0..2)

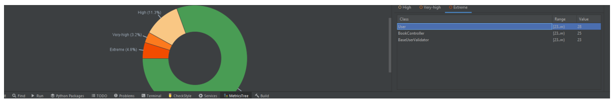
Figure 26: User After

3.9 Coupling Between Objects (CBO)

Lastly, the CBO is mainly good, but there is room for improvement. The User class is one of the ones that can be improved, although part of its coupling could be explained by the refactoring that we have done earlier. The Book Controller's extreme coupling could be explained by the use of validators / handlers and also its need to access the User Repository for the authors of the books. We believe that there is no way to improve the BaseUserValidator's coupling.

As for other classes with relatively smaller problems, the User Controller's is again due to the use of validators, while the User Service requires to use the other Services for Info, Profile and Status after our refactoring.

There are also a few exceptions and validators with higher than normal coupling but we also believe that these cannot be improved.



Class	Range	Value
User	[23..∞)	28
BookController	[23..∞)	25
BaseUserValidator	[23..∞)	23

Figure 27: CBO

4 MUTATION TESTING REPORT

Mutation testing represents the technique used by the production team for enriching the test suite throughout the development process, ensuring that the built test suite achieves a high level of fault detection capability. In this context, the PIT tool was used for the automation of the process of creating mutants and checking whether the existing tests were able to kill the respective mutants or not.

4.1 Final metrics and results

The final version of the test suite registers these values for the specific metrics:

(1) Line coverage: 92%
(2) Mutation score: 78%

38 out of the total of 54 implemented classes are 100% Line Covered by their respective tests, while the other classes are more than 80% Line Covered.

Regarding the Mutation Score, Figure 28 shows the distribution of the Mutation Score achieved by each implemented production class.

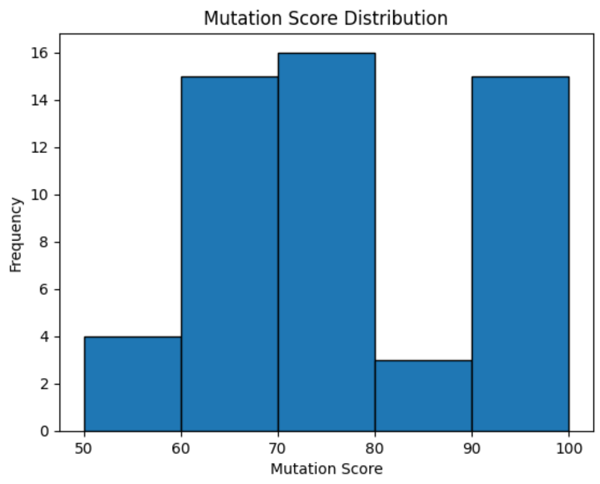


Figure 28: Mutations Score Distribution

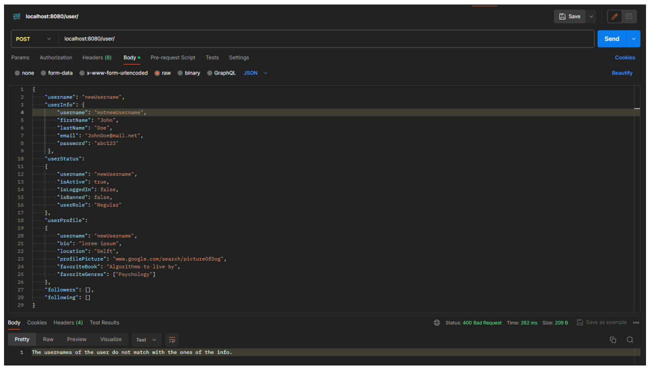
4.2 History of improving Mutation Coverage

The table below documents the commits made to the production code to improve the mutation score. Each row of the table contains the SHA of a particular commit together with the mutation score before and after the commit had been made.

Mutation Score Improvements		
SHA of the commit	MS Before (%)	MS After (%)
da1c32e6	0	12
e8f0b1ba	19	27
af5d4859	27	38
419b9ba4	55	57
07e38106	55	65
07000c53	65	69
bab2d46f	72	82

5 SYSTEM TESTING

On top of the usual Unit and Mutation Testing, we have also used System Testing to assert the capabilities of our micro-service, as well as it's behaviour in different scenarios. Therefore, we have come up with suite of tests that verifies how the program behaves in different situations. Firstly, we will go over the basic features of an user:
The first test handles the creation of a regular user. By passing into the body of the request the information needed, we can submit a request to our service, thus verifying whether the data is eligible and granting a appropriate answer. We can see that if we the usernames do not match inside the User entity, we receive an error message.



POST localhost:8080/user/

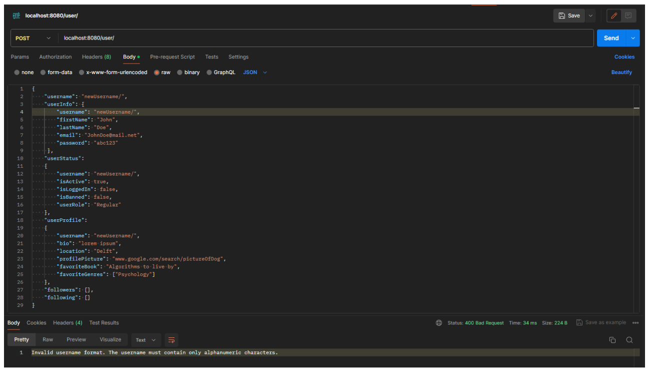
```
{
    "username": "newUsername",
    "userInfo": {
        "username": "notnewUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus":
    {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": ["Psychology"]
    },
    "followers": [],
    "following": []
}
```

Status: 400 Bad Request Time: 262 ms Size: 209 B

```
The usernames of the user do not match with the ones of the info.
```

Figure 29: Not Matching Name

Error messages are also granted if the username contains illegal characters (anything that is not alfa-numeric) or if the isActive field is set to false.



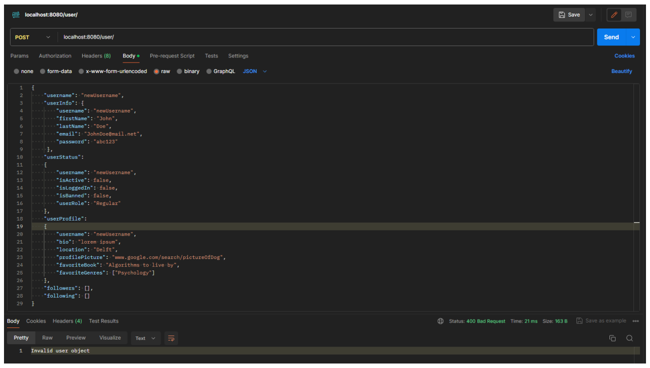
POST localhost:8080/user/

```
{
    "username": "newUsername/",
    "userInfo": {
        "username": "newUsername/",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus":
    {
        "username": "newUsername/",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "newUsername/",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": ["Psychology"]
    },
    "followers": [],
    "following": []
}
```

Status: 400 Bad Request Time: 34 ms Size: 224 B

```
Invalid username format. The username must contain only alphanumeric characters.
```

Figure 30: Invalid Name



POST localhost:8080/user/

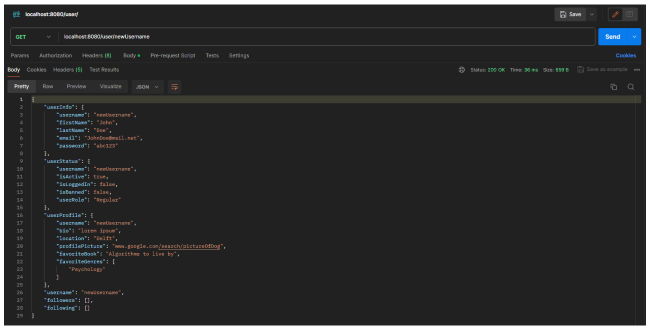
```
{
    "username": "newUsername",
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus":
    {
        "username": "newUsername",
        "isActive": false,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": ["Psychology"]
    },
    "followers": [],
    "following": []
}
```

Status: 400 Bad Request Time: 21 ms Size: 163 B

```
Invalid user object
```

Figure 31: User not active

Finally when the user entity passes all the checks, it is created in our database.


localhost:8080/user/

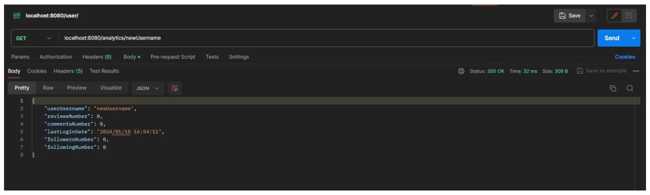
GET localhost:8080/user/newUsername

Status: 200 OK Time: 36 ms Size: 659 B

```
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": [
            "Psychology"
        ]
    },
    "username": "newUsername",
    "followers": [],
    "following": []
}
```

Figure 32: Created User

As the analytics entity is created whenever we can also check if it's properly registered.


localhost:8080/user/

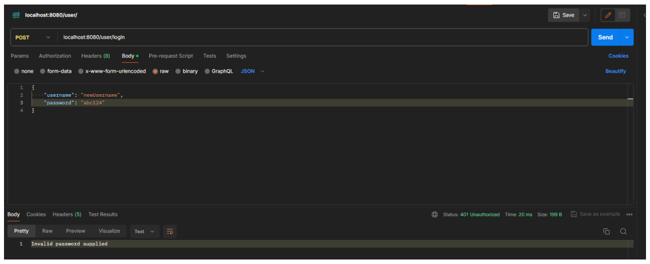
GET localhost:8080/analytics/newUsername

Status: 200 OK Time: 32 ms Size: 309 B

```
{
    "userUsername": "newUsername",
    "reviewsNumber": 0,
    "commentsNumber": 0,
    "lastLoginDate": "2024/01/18 16:54:11",
    "followersNumber": 0,
    "followingNumber": 0
}
```

Figure 33: Analytics Entity

Now, we can login by passing the user entity to the request's body.


localhost:8080/user/

POST localhost:8080/user/login

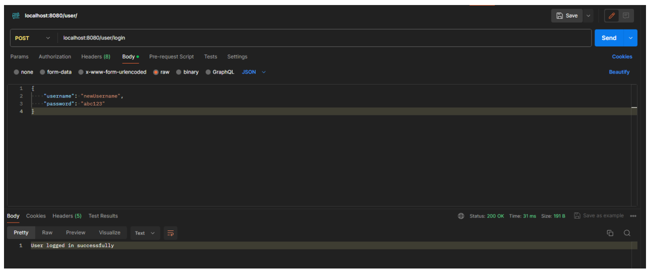
Body: raw JSON

```
{
    "username": "newUsername",
    "password": "abc124"
}
```

Status: 401 Unauthorized Time: 20 ms Size: 199 B

```
Invalid password supplied
```

Figure 34: Invalid Login


localhost:8080/user/

POST localhost:8080/user/login

Body: raw JSON

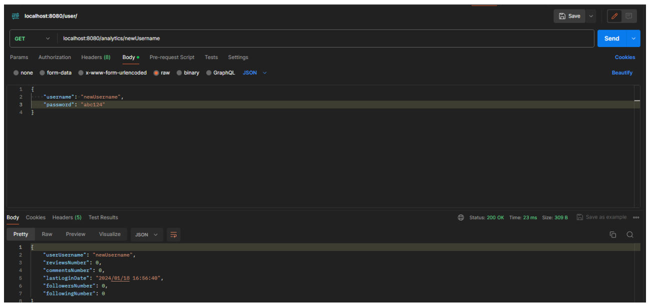
```
{
    "username": "newUsername",
    "password": "abc123"
}
```

Status: 200 OK Time: 31 ms Size: 191 B

```
User logged in successfully
```

Figure 35: Valid Login

Now that we are logged in, we can check the analytics once again to see if our last login has been updated, as well as our isLoggedIn field being changed to true.


localhost:8080/user/

GET localhost:8080/analytics/newUsername

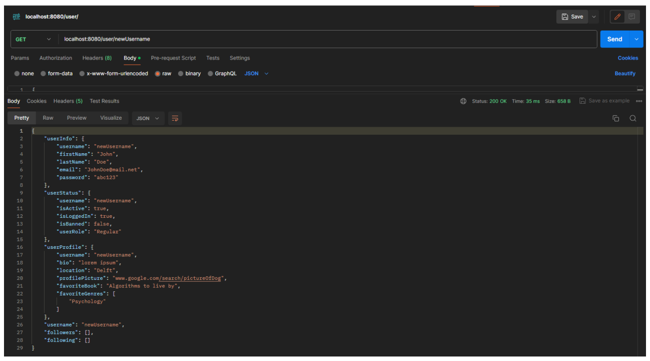
Body: raw JSON

```
{
    "username": "newUsername",
    "password": "abc124"
}
```

Status: 200 OK Time: 23 ms Size: 309 B

```
{
    "userUsername": "newUsername",
    "reviewsNumber": 0,
    "commentsNumber": 0,
    "lastLoginDate": "2024/01/18 16:56:40",
    "followersNumber": 0,
    "followingNumber": 0
}
```

Figure 36: Analytics Changed


localhost:8080/user/

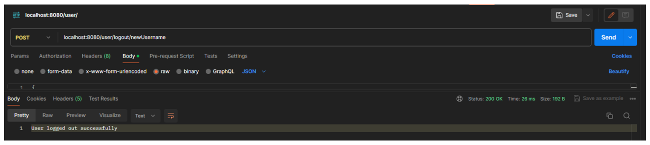
GET localhost:8080/user/newUsername

Status: 200 OK Time: 35 ms Size: 658 B

```
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": true,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": [
            "Psychology"
        ]
    },
    "username": "newUsername",
    "followers": [],
    "following": []
}
```

Figure 37: LoggedIn Field Changed

Now we logout and see that our isLoggedIn field switches back to false.

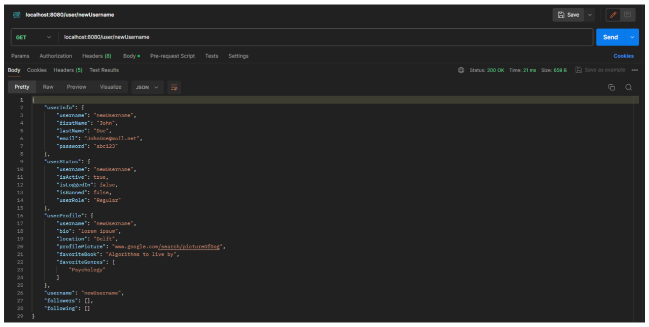

localhost:8080/user/

POST localhost:8080/user/logout/newUsername

Status: 200 OK Time: 26 ms Size: 192 B

```
User logged out successfully
```

Figure 38: Logging Out


localhost:8080/user/newUsername

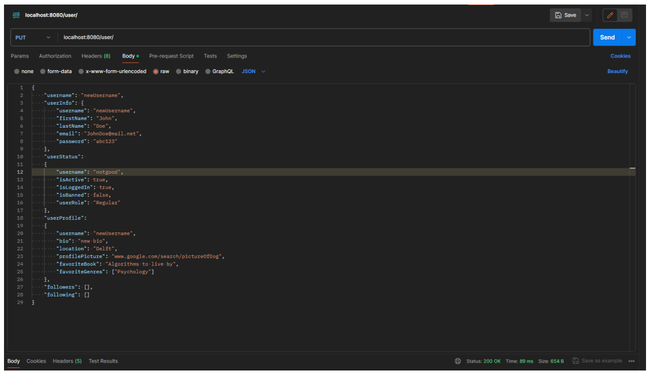
GET localhost:8080/user/newUsername

Status: 200 OK Time: 21 ms Size: 659 B

```
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": [
            "Psychology"
        ]
    },
    "username": "newUsername",
    "followers": [],
    "following": []
}
```

Figure 39: LoggedIn Field Changed

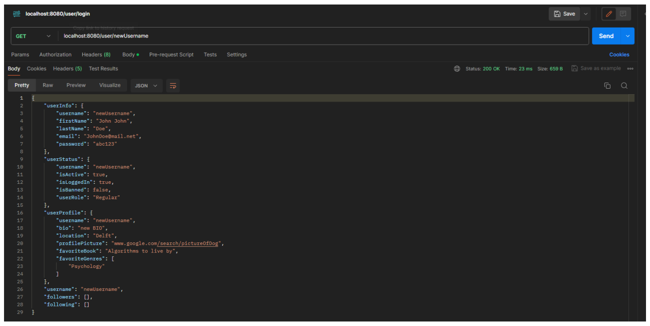
The second test handles editing and deleting your own user. We first change our bio and one of the usernames.



```
localhost:8080/user/
PUT localhost:8080/user/
none  form-data  x-www-form-urlencoded  raw  binary  GraphQL  JSON
{
    "username": "newUsername",
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus":
    {
        "username": "notgood",
        "isActive": true,
        "isLoggedIn": true,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "newUsername",
        "bio": "new bio",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": ["Psychology"]
    },
    "followers": [],
    "following": []
}
Status: 200 OK  Time: 89 ms  Size: 654 B
```

Figure 40: Edit User

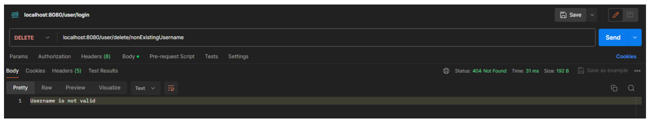
We see that the username doesn't change but our bio does.



```
localhost:8080/user/login
GET localhost:8080/user/newUsername
Status: 200 OK  Time: 23 ms  Size: 659 B
JSON
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": true,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername",
        "bio": "new BIO",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": [
            "Psychology"
        ]
    },
    "username": "newUsername",
    "followers": [],
    "following": []
}
```

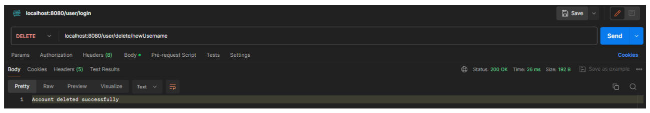
Figure 41: Edited User Get

After this, we delete our own user, if we pass an invalid username an error pops up, but then we successfully delete our entity.



```
localhost:8080/user/login
DELETE localhost:8080/user/delete/nonExistingUsername
Status: 404 Not Found  Time: 31 ms  Size: 192 B
Text
Username is not valid
```

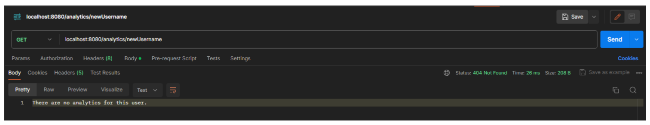
Figure 42: Delete Not Exists



```
localhost:8080/user/login
DELETE localhost:8080/user/delete/newUsername
Status: 200 OK  Time: 26 ms  Size: 192 B
Text
Account deleted successfully
```

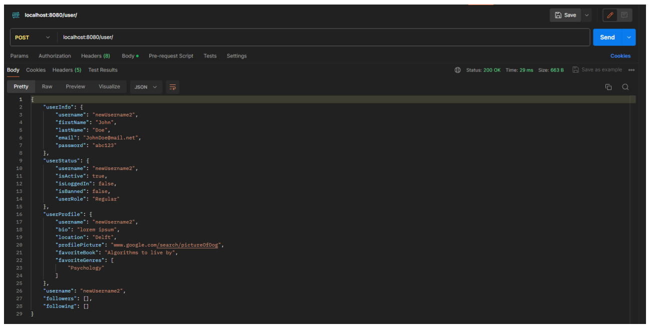
Figure 43: Delete Own User

Now, we will check if the analytics has also been deleted.



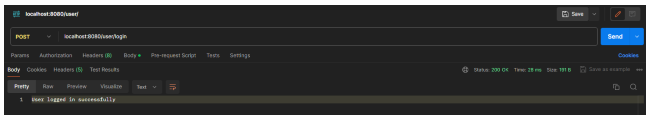
```
localhost:8080/analytics/newUsername
GET localhost:8080/analytics/newUsername
Status: 404 Not Found  Time: 26 ms  Size: 208 B
Text
There are no analytics for this user.
```

Figure 44: Deleted Analytics



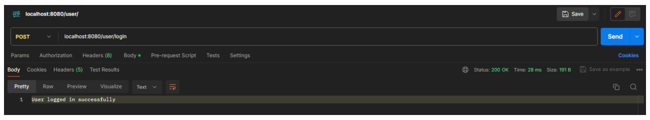
```
localhost:8080/user/
POST localhost:8080/user/
Status: 200 OK  Time: 29 ms  Size: 663 B
JSON
{
    "userInfo": {
        "username": "newUsername2",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername2",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername2",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
        "favoriteGenres": [
            "Psychology"
        ]
    },
    "username": "newUsername2",
    "followers": [],
    "following": []
}
```

Figure 45: Create New User



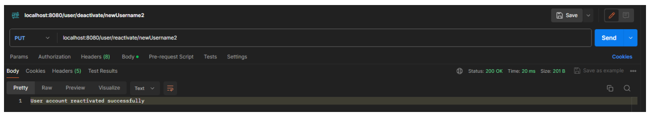
```
localhost:8080/user/
POST localhost:8080/user/login
Status: 200 OK  Time: 28 ms  Size: 191 B
Text
User logged in successfully
```

Figure 46: Log In



```
localhost:8080/user/
POST localhost:8080/user/login
Status: 200 OK  Time: 28 ms  Size: 191 B
Text
User logged in successfully
```

Figure 47: Deactivate Self



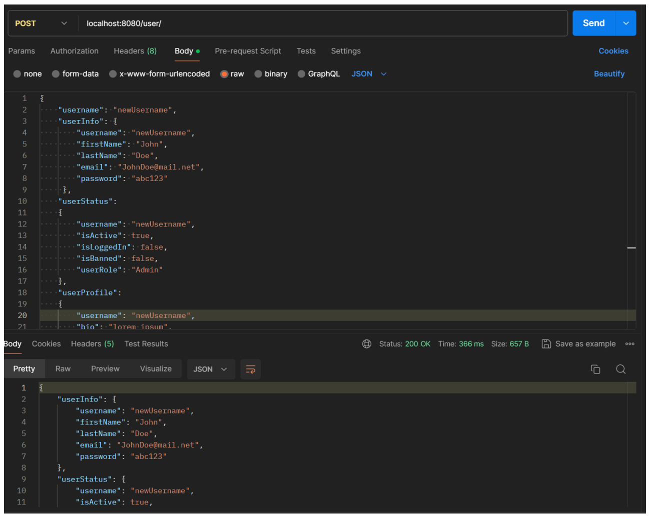
```
localhost:8080/user/deactivate/newUsername2
PUT localhost:8080/user/reactivate/newUsername2
Status: 200 OK  Time: 20 ms  Size: 201 B
Text
User account reactivated successfully
```

Figure 48: Reactivate Self

For the third test, we will see test the activation status of our user. We start by creating a new user and logging in. Then, we will try and deactivate our account and reactivate it again.

Successfully Tested Admin Scenarios:

First test for an admin scenario is creating an admin user, logging in to the system, updating admin's user info, and logging out from the system.



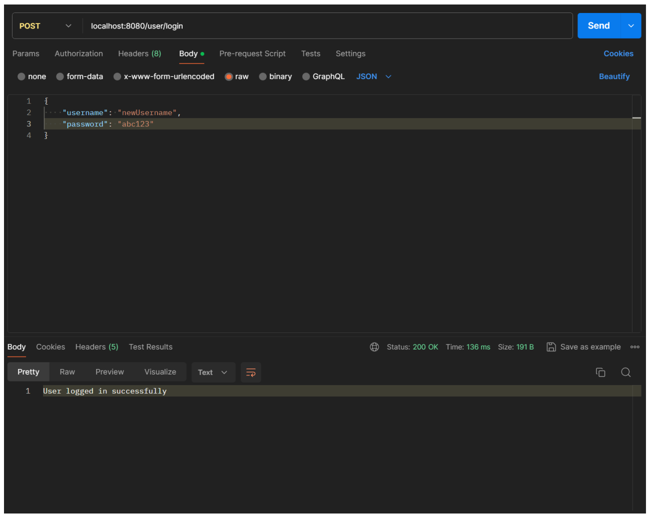
localhost:8080/user/

```
{
    "username": "newUsername",
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus":
    {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Admin"
    },
    "userProfile":
    {
        "username": "newUsername",
        "bio": "lorem ipsum",
```

Status: 200 OK Time: 366 ms Size: 657 B

```
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
```

Figure 49: Create Admin User



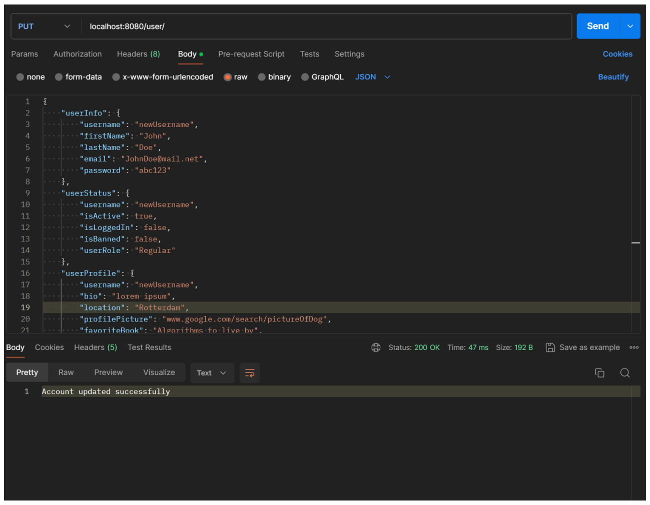
localhost:8080/user/login

```
{
    "username": "newUsername",
    "password": "abc123"
}
```

Status: 200 OK Time: 136 ms Size: 191 B

```
User logged in successfully
```

Figure 50: Log in



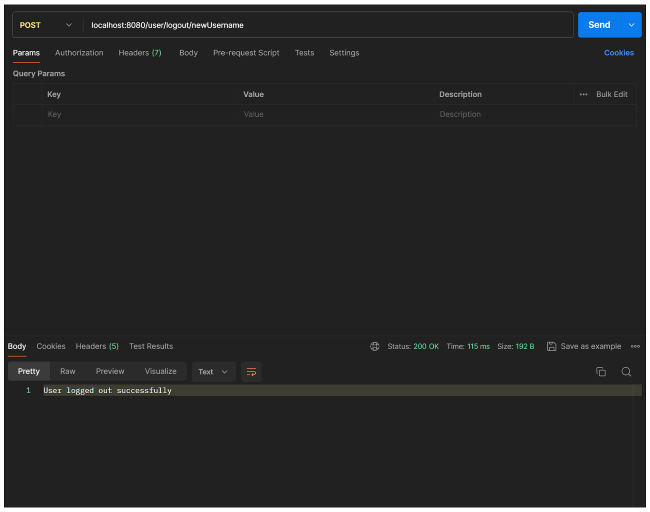
localhost:8080/user/

```
{
    "userInfo": {
        "username": "newUsername",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "newUsername",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "newUsername",
        "bio": "lorem ipsum",
        "location": "Rotterdam",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
```

Status: 200 OK Time: 47 ms Size: 192 B

```
Account updated successfully
```

Figure 51: Update admin's user info



localhost:8080/user/logout/newUsername

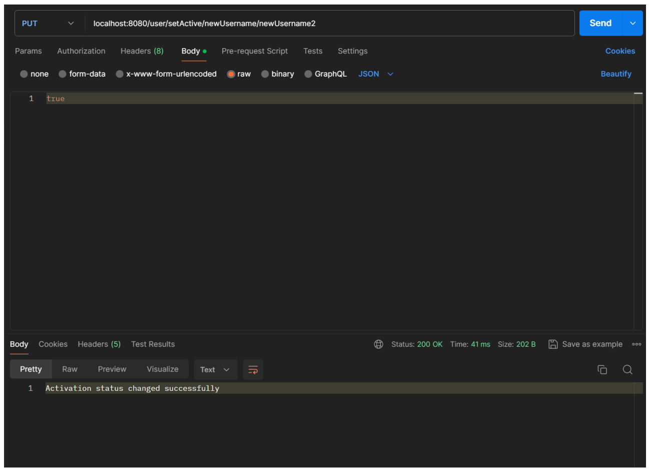
Key	Value	Description
Key	Value	Description

Status: 200 OK Time: 115 ms Size: 192 B

```
User logged out successfully
```

Figure 52: Log out

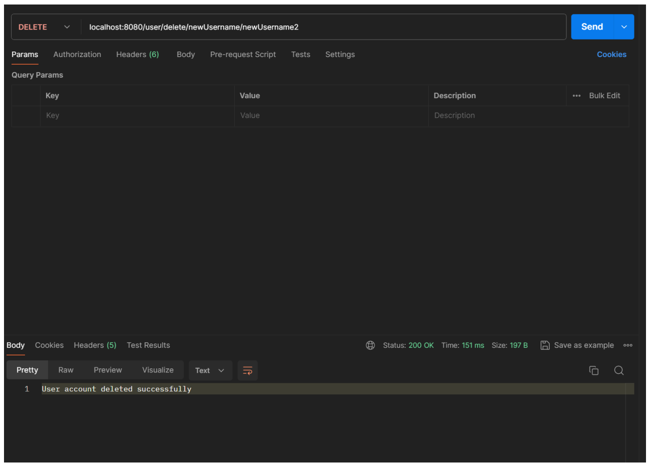
Second test includes changing activation of a regular user with logging in to, and logging out from the system.



```
PUT  localhost:8080/user/setActive/newUsername/newUsername2
Body: raw  Text
true

Status: 200 OK  Time: 41 ms  Size: 202 B
Activation status changed successfully
```

Figure 53: Set a regular user active

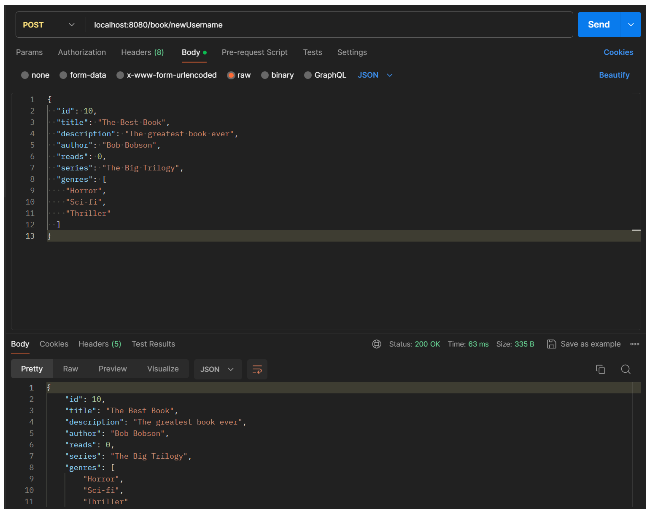
Third test for an admin is for deleting a regular user with logging in to, and logging out from the system.



```
DELETE  localhost:8080/user/delete/newUsername/newUsername2

Status: 200 OK  Time: 151 ms  Size: 197 B
User account deleted successfully
```

Figure 54: Delete a regular user

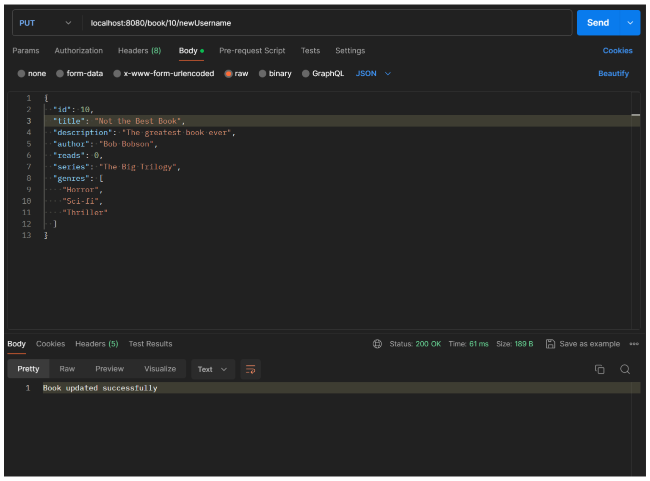
Fourth test is for creating a book as an admin, with loggin in to, and logging out from the system



```
POST  localhost:8080/book/newUsername
Body: raw  JSON
{
  "id": 10,
  "title": "The Best Book",
  "description": "The greatest book ever",
  "author": "Bob Bobson",
  "reads": 0,
  "series": "The Big Trilogy",
  "genres": [
    "Horror",
    "Sci-fi",
    "Thriller"
  ]
}

Status: 200 OK  Time: 63 ms  Size: 335 B
{
    "id": 10,
    "title": "The Best Book",
    "description": "The greatest book ever",
    "author": "Bob Bobson",
    "reads": 0,
    "series": "The Big Trilogy",
    "genres": [
        "Horror",
        "Sci-fi",
        "Thriller"
```

Figure 55: Create book

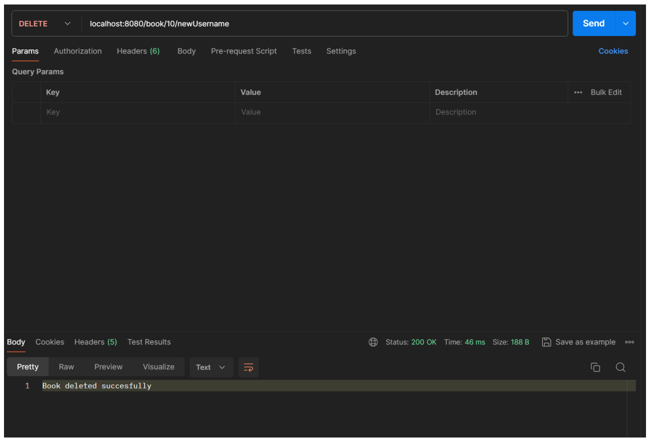
Fifth test for an admin is updating a book that is created before with logging in to and logging out from the system.



```
PUT  localhost:8080/book/10/newUsername
Body: raw  JSON
{
  "id": 10,
  "title": "Not the Best Book",
  "description": "The greatest book ever",
  "author": "Bob Bobson",
  "reads": 0,
  "series": "The Big Trilogy",
  "genres": [
    "Horror",
    "Sci-fi",
    "Thriller"
  ]
}

Status: 200 OK  Time: 61 ms  Size: 189 B
Book updated successfully
```

Figure 56: Update book

And finally the last test consists of deleting a book that is created before with logging in to and logging out from the system



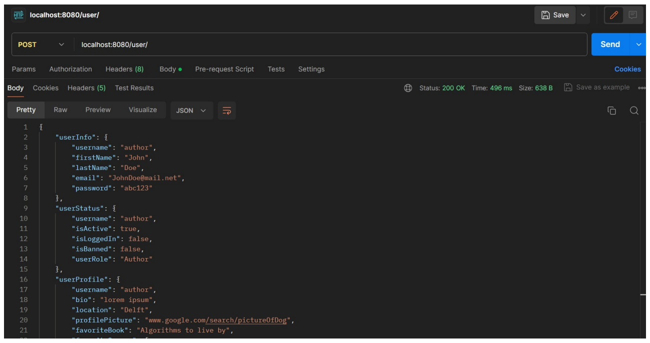
```
DELETE  localhost:8080/book/10/newUsername
```

Status: 200 OK Time: 46 ms Size: 188 B

```
Book deleted succesfully
```

Figure 57: Delete book

Successfully Tested Author Scenarios:
To build upon the testing of the basic features of a regular user, we will go over the available features of an user with the role of an Author. For the beginning we will create an instance of an Author ("userRole" is set to Author).



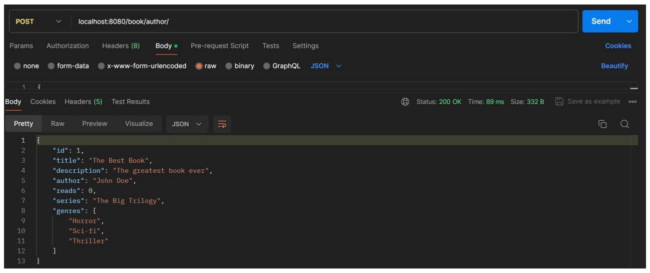
```
POST  localhost:8080/user/
```

Status: 200 OK Time: 496 ms Size: 638 B

```
{
    "userInfo": {
        "username": "author",
        "firstName": "John",
        "lastName": "Doe",
        "email": "JohnDoe@mail.net",
        "password": "abc123"
    },
    "userStatus": {
        "username": "author",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Author"
    },
    "userProfile": {
        "username": "author",
        "bio": "lorem ipsum",
        "location": "Delft",
        "profilePicture": "www.google.com/search/pictureOfDog",
        "favoriteBook": "Algorithms to live by",
```

Figure 58: Created Author

Now, we can add a new book entity to the database. In this case, the book is written by the author sending the request (i.e. personal book of the author).



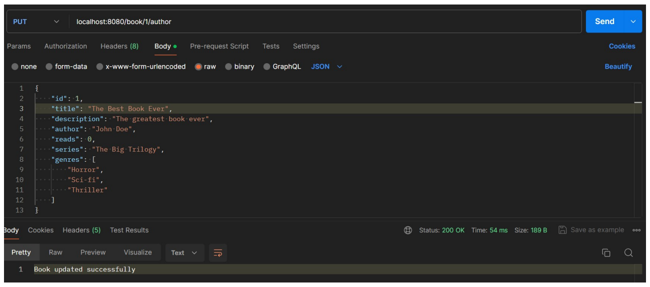
```
POST  localhost:8080/book/author/
```

Status: 200 OK Time: 89 ms Size: 332 B

```
{
    "id": 1,
    "title": "The Best Book",
    "description": "The greatest book ever",
    "author": "John Doe",
    "reads": 0,
    "series": "The Big Trilogy",
    "genres": [
        "Horror",
        "Sci-fi",
        "Thriller"
    ]
}
```

Figure 59: Created Own Book

As the previously added book is authored by the sender of the request, the same sender can update the information related to the book. (The title of the book is changed).



```
PUT  localhost:8080/book/1/author
```

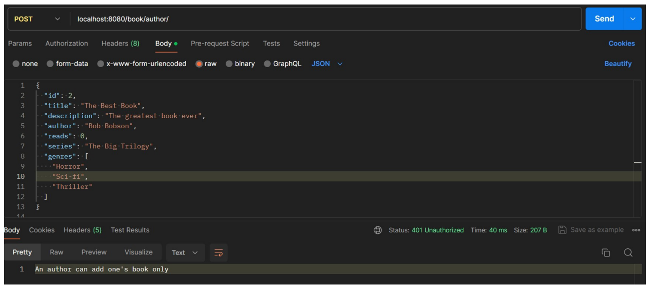
```
{
    "id": 1,
    "title": "The Best Book Ever",
    "description": "The greatest book ever",
    "author": "John Doe",
    "reads": 0,
    "series": "The Big Trilogy",
    "genres": [
        "Horror",
        "Sci-fi",
        "Thriller"
    ]
}
```

Status: 200 OK Time: 54 ms Size: 189 B

```
Book updated successfully
```

Figure 60: Updated Own Book

The next test handles the scenario in which an author tries to add a book that is not authored by oneself. This action generates a suggestive error message.



```
POST  localhost:8080/book/author/
```

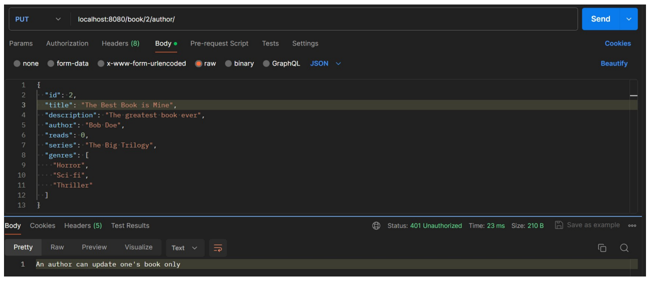
```
{
  "id": 2,
  "title": "The Best Book",
  "description": "The greatest book ever",
  "author": "Bob Bobson",
  "reads": 0,
  "series": "The Big Trilogy",
  "genres": [
    "Horror",
    "Sci-fi",
    "Thriller"
  ]
}
```

Status: 401 Unauthorized Time: 40 ms Size: 207 B

```
An author can add one's book only
```

Figure 61: Invalid Book Creation

Using the same argument as in the previous example, the system also handles scenarios in which an author tries to update a book that is not authored by oneself.



```
PUT  localhost:8080/book/2/author/
```

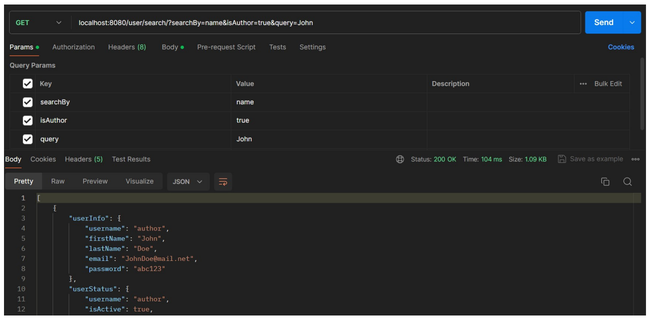
```
{
  "id": 2,
  "title": "The Best Book is Mine",
  "description": "The greatest book ever",
  "author": "Bob Doe",
  "reads": 0,
  "series": "The Big Trilogy",
  "genres": [
    "Horror",
    "Sci-fi",
    "Thriller"
  ]
}
```

Status: 401 Unauthorized Time: 23 ms Size: 210 B

```
An author can update one's book only
```

Figure 62: Invalid Update Book

Now, we will check the functionality of searching a user given either one's first name or last name using the previously created Author instance.



GET localhost:8080/user/search/?searchBy=name&isAuthor=true&query=John

Query Params

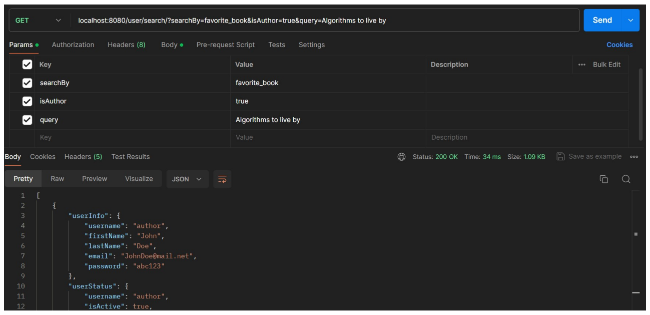
Key	Value	Description
searchBy	name	
isAuthor	true	
query	John	

Status: 200 OK Time: 104 ms Size: 1.09 KB

```
[
    {
        "userInfo": {
            "username": "author",
            "firstName": "John",
            "lastName": "Doe",
            "email": "JohnDoe@mail.net",
            "password": "abc123"
        },
        "userStatus": {
            "username": "author",
            "isActive": true,
```

Figure 63: Search User by Name

We repeat the same test for checking the functionality of searching an user by one's favorite book.



GET localhost:8080/user/search/?searchBy=favorite_book&isAuthor=true&query=Algorithms to live by

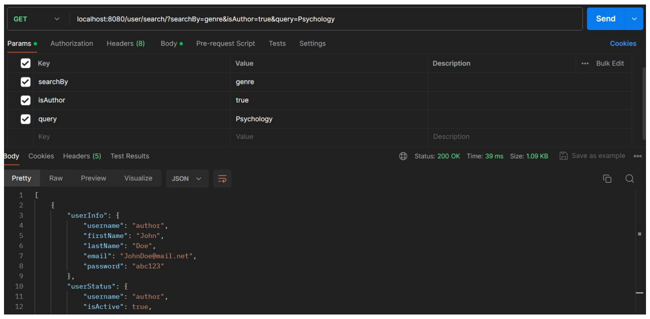
Key	Value	Description
searchBy	favorite_book	
isAuthor	true	
query	Algorithms to live by	

Status: 200 OK Time: 34 ms Size: 1.09 KB

```
[
    {
        "userInfo": {
            "username": "author",
            "firstName": "John",
            "lastName": "Doe",
            "email": "JohnDoe@mail.net",
            "password": "abc123"
        },
        "userStatus": {
            "username": "author",
            "isActive": true,
```

Figure 64: Search User by Fav Book

Once again, we repeat the same test for checking the functionality of searching an user by one's favorite genres.



GET localhost:8080/user/search/?searchBy=genre&isAuthor=true&query=Psychology

Key	Value	Description
searchBy	genre	
isAuthor	true	
query	Psychology	

Status: 200 OK Time: 39 ms Size: 1.09 KB

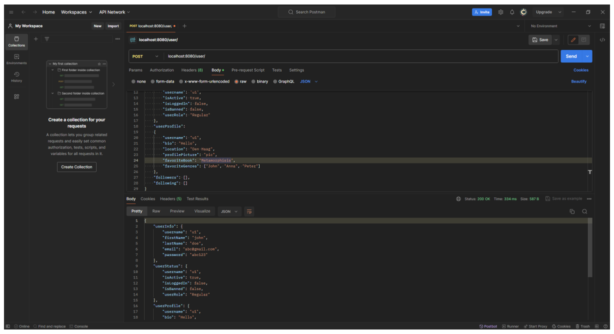
```
[
    {
        "userInfo": {
            "username": "author",
            "firstName": "John",
            "lastName": "Doe",
            "email": "JohnDoe@mail.net",
            "password": "abc123"
        },
        "userStatus": {
            "username": "author",
            "isActive": true,
```

Figure 65: Search User by Genres

Successfully Tested Follower Functionality Scenario:

For testing the follower functionality we have opted for this scenario: 2 users will create an account, one of them logs in, tries to follow a non-existent user and then follows the second user. In the end, the second user deletes his account.

Two users are created correctly and added to the database



POST localhost:8080/user/

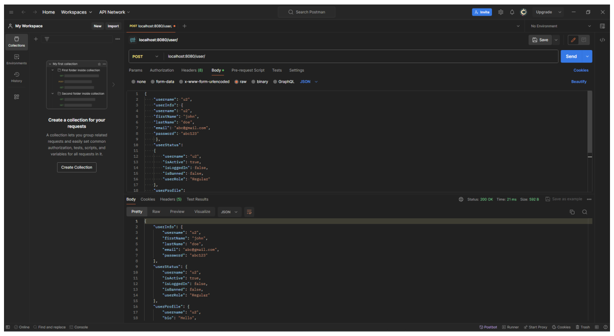
Body: raw JSON

```
        "username": "u1",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "u1",
        "bio": "Hello",
        "location": "Den Haag",
        "profilePicture": "pic",
        "favoriteBook": "Metamorphisis",
        "favoriteGenres": ["John", "Anna", "Peter"]
    },
    "followers": [],
    "following": []
}
```

Status: 200 OK Time: 334 ms Size: 587 B

```
{
    "userInfo": {
        "username": "u1",
        "firstName": "john",
        "lastName": "doe",
        "email": "abc@gmail.com",
        "password": "abc123"
    },
    "userStatus": {
        "username": "u1",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "u1",
        "bio": "Hello",
```

Figure 66: Creating User 1



POST localhost:8080/user/

Body: raw JSON

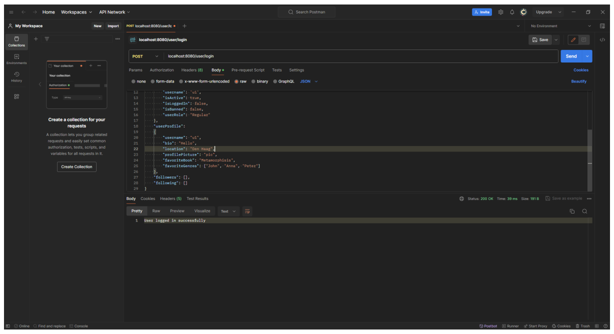
```
{
    "username": "u2",
    "userInfo": {
    "username": "u2",
    "firstName": "john",
    "lastName": "doe",
    "email": "abc@gmail.com",
    "password": "abc123"
     },
    "userStatus":
    {
        "username": "u2",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
```

Status: 200 OK Time: 21 ms Size: 592 B

```
{
    "userInfo": {
        "username": "u2",
        "firstName": "john",
        "lastName": "doe",
        "email": "abc@gmail.com",
        "password": "abc123"
    },
    "userStatus": {
        "username": "u2",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile": {
        "username": "u2",
        "bio": "Hello",
```

Figure 67: Creating User 2

The first user successfully logs in



POST localhost:8080/user/login

Body: raw JSON

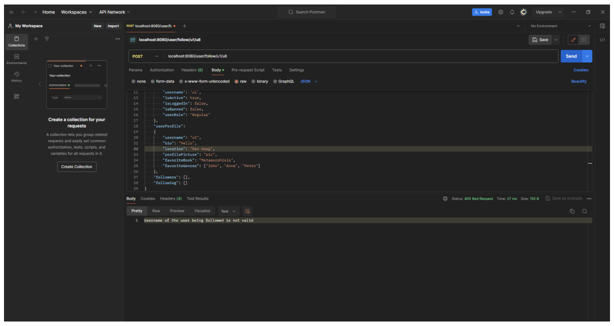
```
        "username": "u1",
        "isActive": true,
        "isLoggedIn": false,
        "isBanned": false,
        "userRole": "Regular"
    },
    "userProfile":
    {
        "username": "u1",
        "bio": "Hello",
        "location": "Den Haag",
        "profilePicture": "pic",
        "favoriteBook": "Metamorphisis",
        "favoriteGenres": ["John", "Anna", "Peter"]
    },
    "followers": [],
    "following": []
}
```

Status: 200 OK Time: 39 ms Size: 191 B

```
User logged in successfully
```

Figure 68: Login

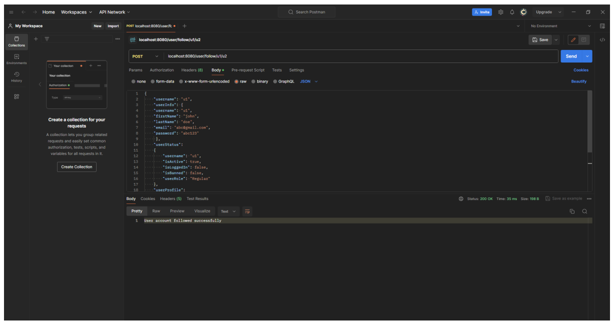
The first user tries to follow an account that doesn't exist and we are alerted of that fact



```
POST localhost:8080/user/follow/u1/u8

    "username": "u1",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":
  {
    "username": "u1",
    "bio": "Hello",
    "location": "Den Haag",
    "profilePicture": "pic",
    "favoriteBook": "Metamorphisis",
    "favoriteGenres": ["John", "Anna", "Peter"]
  },
  "followers": [],
  "following": []
}

Status: 400 Bad Request  Time: 27 ms  Size: 192 B

Username of the user being followed is not valid
```

Figure 69: Attempted Follow

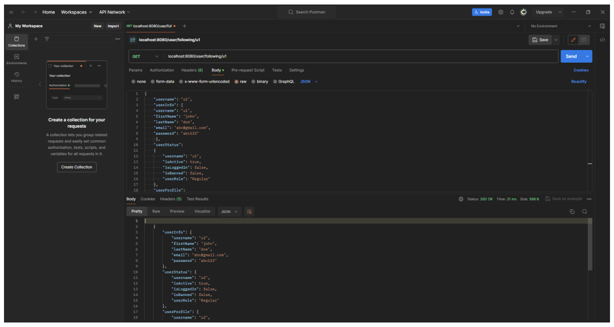
The first user successfully follows the second user



```
POST localhost:8080/user/follow/u1/u2

{
  "username": "u1",
  "userInfo": {
  "username": "u1",
  "firstName": "john",
  "lastName": "doe",
  "email": "abc@gmail.com",
  "password": "abc123"
  },
  "userStatus":
  {
    "username": "u1",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":

Status: 200 OK  Time: 35 ms  Size: 198 B

User account followed successfully
```

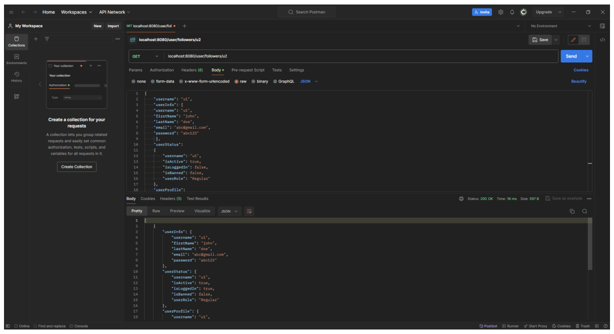
Figure 70: Follow

We use get requests to get the following list of the first user and the followers of the second one. As we can see, everything worked as intended.



```
GET localhost:8080/user/following/u1

{
  "username": "u1",
  "userInfo": {
  "username": "u1",
  "firstName": "john",
  "lastName": "doe",
  "email": "abc@gmail.com",
  "password": "abc123"
  },
  "userStatus":
  {
    "username": "u1",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":

Status: 200 OK  Time: 21 ms  Size: 598 B

[
    {
        "userInfo": {
            "username": "u2",
            "firstName": "john",
            "lastName": "doe",
            "email": "abc@gmail.com",
            "password": "abc123"
        },
        "userStatus": {
            "username": "u2",
            "isActive": true,
            "isLoggedIn": false,
            "isBanned": false,
            "userRole": "Regular"
        },
        "userProfile": {
            "username": "u2",
```

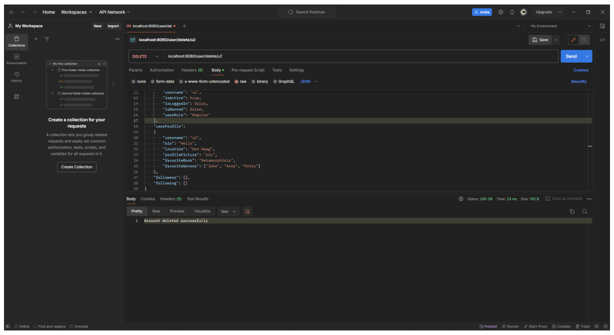
Figure 71: Following List of User 1



```
GET localhost:8080/user/followers/u2

{
  "username": "u1",
  "userInfo": {
  "username": "u1",
  "firstName": "john",
  "lastName": "doe",
  "email": "abc@gmail.com",
  "password": "abc123"
  },
  "userStatus":
  {
    "username": "u1",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":

Status: 200 OK  Time: 16 ms  Size: 597 B

[
    {
        "userInfo": {
            "username": "u1",
            "firstName": "john",
            "lastName": "doe",
            "email": "abc@gmail.com",
            "password": "abc123"
        },
        "userStatus": {
            "username": "u1",
            "isActive": true,
            "isLoggedIn": true,
            "isBanned": false,
            "userRole": "Regular"
        },
        "userProfile": {
            "username": "u1",
```

Figure 72: Followers List of User 2

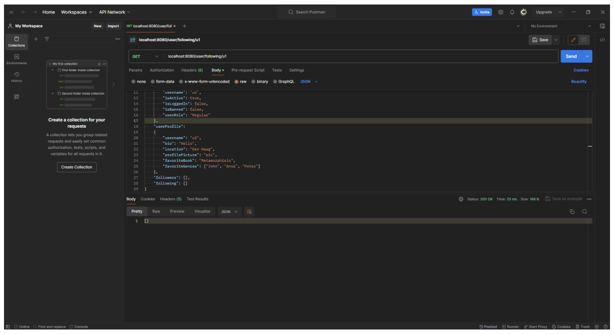
The second user now deletes his account (after logging in). This is done in order to check if he also disappears from the other's following list.



```
DELETE localhost:8080/user/delete/u2

    "username": "u2",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":
  {
    "username": "u2",
    "bio": "Hello",
    "location": "Den Haag",
    "profilePicture": "pic",
    "favoriteBook": "Metamorphisis",
    "favoriteGenres": ["John", "Anna", "Peter"]
  },
  "followers": [],
  "following": []
}

Status: 200 OK  Time: 23 ms  Size: 192 B

Account deleted successfully
```

Figure 73: Delete User 2

Once again we get the following list of the first user. The second user was deleted from it. This means that all of the functionality works as intended



```
GET localhost:8080/user/following/u1

    "username": "u2",
    "isActive": true,
    "isLoggedIn": false,
    "isBanned": false,
    "userRole": "Regular"
  },
  "userProfile":
  {
    "username": "u2",
    "bio": "Hello",
    "location": "Den Haag",
    "profilePicture": "pic",
    "favoriteBook": "Metamorphisis",
    "favoriteGenres": ["John", "Anna", "Peter"]
  },
  "followers": [],
  "following": []
}

Status: 200 OK  Time: 25 ms  Size: 166 B

[]
```

Figure 74: Following List of User 1

6 INTEGRATION REPORT

Our system in general did not have a difficult time with integration because it was mostly self-contained and externally focused. For example, our User class is self-contained because our microservice keeps track of users and is not so much invested in what actions users can perform, as such it is mostly up to the Review Microservice to make changes on their end to access our api.

Similarly, the Book class and its controllers/services are intended to keep track of books and as such the Book Microservice which handles book actions will be polling us rather than the other way around. This means that it is up to them to make integration changes for the Book class.

However, the Analytics class does contain some aspects which require integration with the other services on our side. Since we need to access the repository of actions which users have taken in order to construct accurate analytics profiles. This means our API needs to poll from the Review Microservice.

An initial look at Team 22b's Review Microservice revealed a flaw in the integration. Our service did not have the functionality to collect data from their repositories, specifically when it came to review and comment numbers. To fix this, we will implement a request which we send to their microservice every time we want to fetch the most recent statistics on a user.

We verified that our fix was correct by simulating a review posting on postman. We first created a user, switched to the review microservice, posted a review, and then verified that the review count had increased on our microservice.

However, in the end team 22b proposed their own solution to the issue from their end, and it was decided that this is the path which would be pursued. As such, the integration issue was fixed with no changes from us.

Integration with Team 22b's Review Microservice was successful in terms of endpoints and apart from the aforementioned issues did not run into any problems. There was no integration on our side with the 22c Book Microservice and so we could not test this.

The code for the integration fix is in a branch called "integration". The fix is located in the analytics service.