Sri Lanka Institute of Information Technology

GadgetBadget Research/ Innovation Support System Web Services

Project Report

IT3030 - Programming Applications and Frameworks 2021
Group Assignment

Group Number: Y3S1.15(DS)-06

Group Members:

1. IT19069432 – Dissanayake D.M.I.M.
2. IT19167510 – Bandara J.M.S.A.
3. IT19075754 – Jayasinghe D.T.
4. IT19211824 – Indrahenaka H.R.T.V.
5. IT19237282 – De Silva H.C.K.

Date of submission: 24/04/2021

Table of Contents

1. Workload Distribution

Registration Number	Name with Initials	Web Service Allocated
IT19069432	Dissanayake D.M.I.M.	Users Service • User Management. • User Role Management. • User Authentication and Authorization. • Communicate with all other web services to get a summary for each non-employee user using Inter-service communication.
IT19167510	Bandara J.M.S.A.	ResearchHub Service • Research Projects and Collaborator details Management per each research. • Research Project Categories Management. • Get a list of funds received by a given research using Inter-service Communication with Fund Service.
IT19075754	Jayasinghe D.T.	Payment Service • Payment Management. • Calculating profits earned for all the payments made. • Communicating with User Service to validate payment method (credit card details) using Inter-service communication when making payments.
IT19211824	Indrahenaka H.R.T.V.	Marketplace Service • Product Management. • Product Category Management. • Communicating with Payment service to obtain a list of payments received for a product using Inter-service communication.
IT19237282	H.C.K. de silva	Funding Service • Funds Management. • Calculating profits earned from all the funds that have been placed by funders. • Communicate with User Service to validate payment method using Inter-service communication when placing funds.

2. Version Controlling

GitHub was used for version control as mentioned in the project requirement specifications provided. Version controlling was used to enable collaborations between group members, keep track of the changes made over time since the project started, and minimize debugging time when integrating the system by resolving conflicts early in the development phase. All five web services were initially committed to the same GitHub repository before commencing the development of the services. The GitHub commit log can be found in the section 2.2. A comprehensive commit log obtained via the master branch of the GitHub repository link provided below.

2.1. Repository Links

- Public GitHub Repository Link: https://github.com/thisisishara/GadgetBadget
- Public GitHub Repository Clone Link: https://github.com/thisisishara/GadgetBadget.git

2.2. Commit Log

The commit log extracted from the GitHub repository insights is attached below and it shows how the commits have been done from March 21 to Apr 21, 2021.

Registration Number	**GitHub Username**
IT19069432 Dissanayake D.M.I.M.	thisisishara
IT19075754 Jayasinghe D.T.	dinushiTJ
IT19167510 Bandara J.M.S.A.	sandunialoki
IT19211824 Indrahenaka H.R.T.V.	TharkanaVish
IT19237282 H.C.K. de silva	it19237282

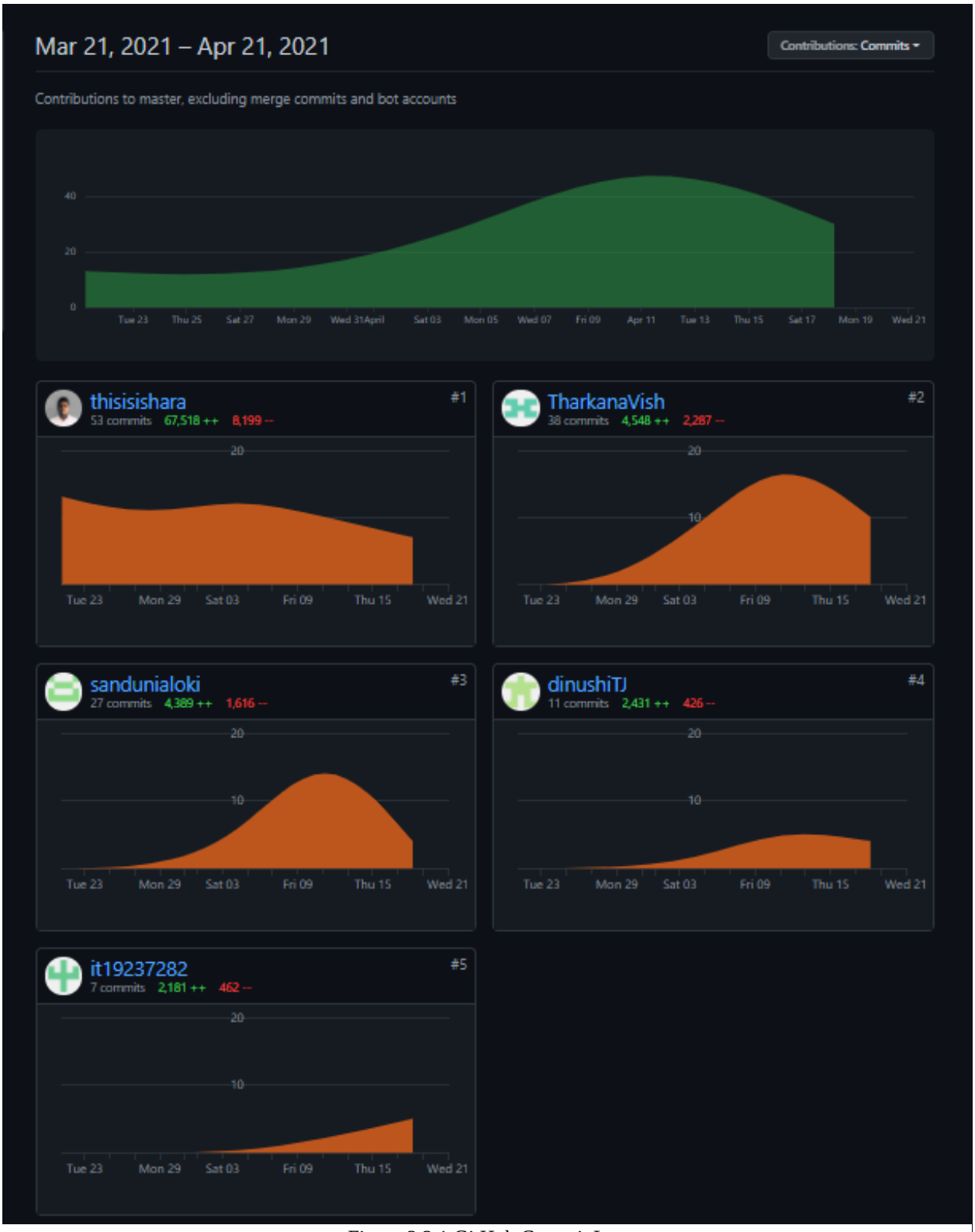


Figure 2.2 1 GitHub Commit Log

3. SE Methodology

The waterfall model was selected over agile methodologies since the requirement set of the project was identified upfront and is non-frequently changing. Not having to release an early product before the system's web services are fully developed was another valid reason for adopting the waterfall model. It helped to list down a specific set of features to be implemented in each web service by each member as the final aim of the project, which helped to have a clear idea about the web services and their functionalities going to be developed by the team, rather than setting goals sprint-vice.

The other reason for choosing the waterfall model was to construct comprehensive API documentation, whereas, in agile methodologies, it is not one of the main concerns. Moreover, since the project timeline was a fixed period and the deadline was clearly stated beforehand, the waterfall model was considered more convenient.

Apart from the development methodologies, as the software architecture of the GadgetBadget system, the microservice architecture was selected. To be compatible with the identified requirements, the GadgetBadget system was supposed to be highly scalable. Therefore, a distributed system with separately developed web services seemed like the most decent option. Development of the microservices done using the resource-oriented architecture (REST) to establish a resource-oriented communication in the web services with its clients.

Moreover, to develop and maintain the code, mainly the object-oriented programming paradigm was used while dividing the classes according to MVC architecture.

4. Time Schedule

4.1. Gantt Chart

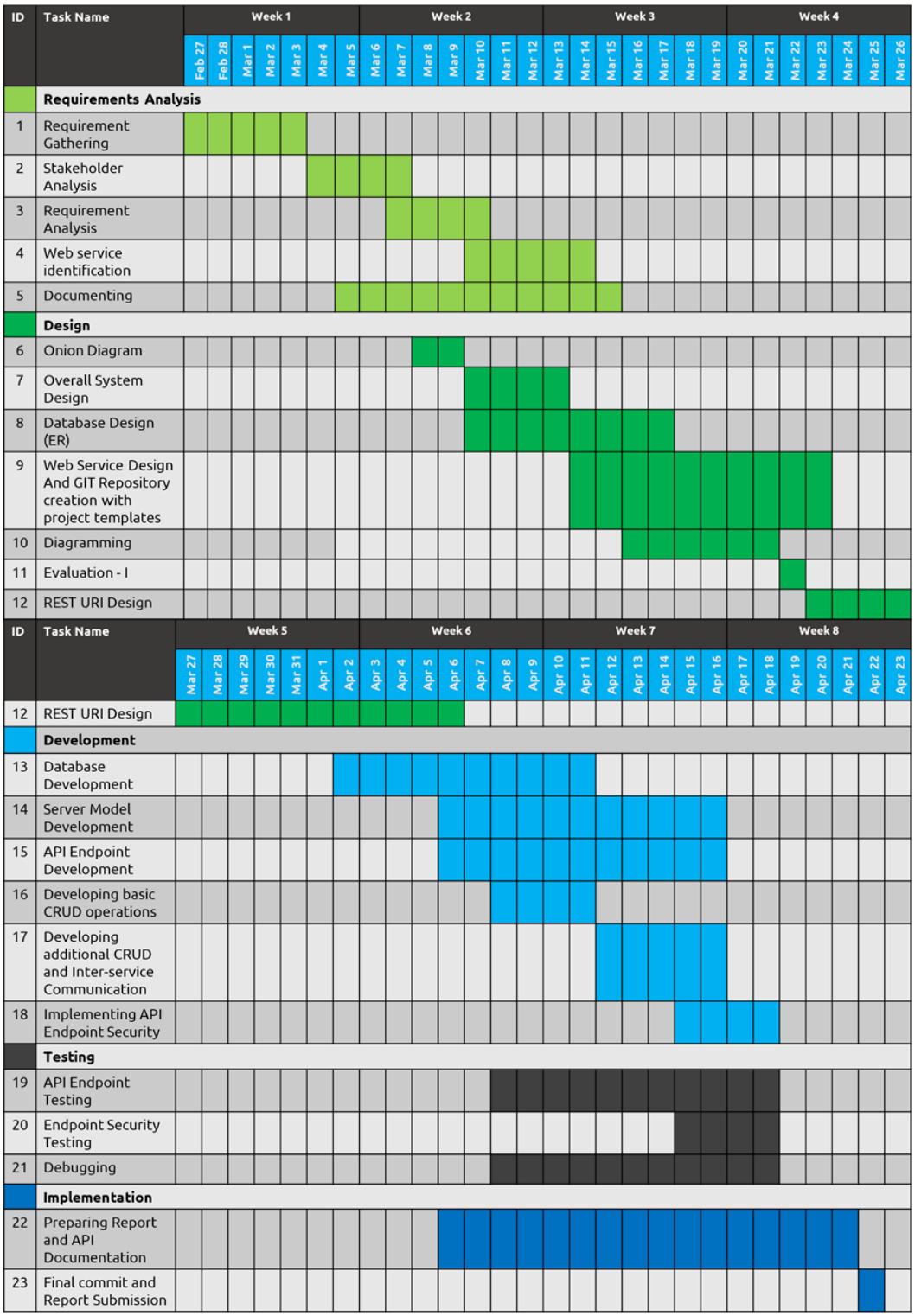


Figure 4.1. 1 Gantt chart

5. Requirements

5.1. Stakeholder Analysis

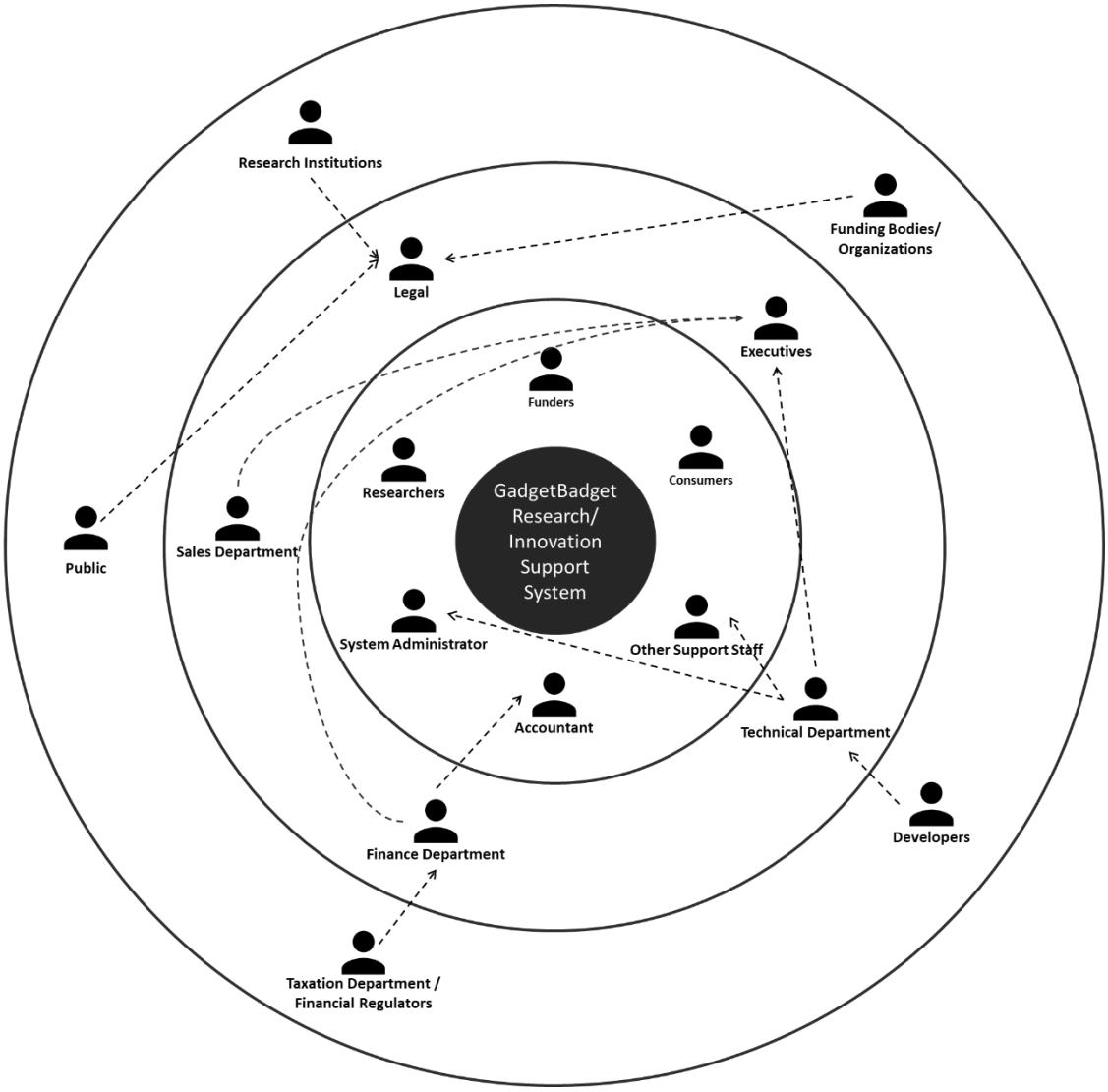


Figure 5.1. 1 Onion Diagram

In the onion diagram of the stakeholder analysis, the innermost circle represents the GadgetBadget Web services. The next layer contains the end-users who are directly interacting with the web services. In the next layer are departments and the employees of the business organization to which GadgetBadget belongs who are profiting and are using the outputs taken from the system to strengthen the business and generate insights. The outermost layer contains the stakeholders who may indirectly affect the system's behavior. They can be indirectly profiting or negatively impacted by the system without direct interactions. [4] [5]. Directly interacting stakeholders with the system

and the list of tasks they perform can be found in the functional requirements subsection of the current section of this report.

5.2. Requirements Analysis

The requirements were identified, gathered, and recorded according to standard requirement gathering methodologies, namely, document analysis, research, and brainstorming. In the requirements gathering phase, all the details provided in the assignment specifications document have been carefully analyzed and the potential requirements found were noted down. Moreover, all group members have performed their research on existing research and innovation supportive web systems and web services, and other research papers on web services to obtain a solid list of well-defined requirements and identify plausible web services. Then stakeholder identification was carried out to determine their needs and expected functions from the web services previously identified.

Finally, the team used brainstorming to identify any of the requirements that might have left out. The finalized set of functional and requirements was concluded by the group as follows.

5.2.1. Functional Requirements

Functional requirements of each web service are shown in a tabular format along with the stakeholder type who requires it as follows.

Web Service	**Functional Requirement**	**Type of the user**
User Service	Add/Remove/View/Update Consumers	Administrators Consumers
	Add/Remove/View/Update Funders	Administrators Funders
	Add/Remove/View/Update Researchers	Administrators Researchers
	Add/Remove/View/Update Employees	Administrators
	Add/Remove/View/Update Payment method details	Administrators Funders Consumers Researchers Payment service (limited) Funding service (limited)
	Change Password	All Users
	Activate/ Deactivate Accounts	Administrators
	Get a list of all users	Administrators
	Get a list of all users with statistics (using inter-service communication)	Administrators
	Add/Remove/View/Update User roles	Administrators

Research Hub Service	Add/Remove/View/Update Research Projects	Administrators Researchers Funders (limited) User Service (limited)
	Add/Remove/View/Update Research Project categories	Administrators Researchers Funders (limited)
	Add/Remove/View/Update collaborators	Administrators Researchers
	Get a list of funds for a specific project (using inter-service communication)	Administrators Researchers
Payment Service	Add/Remove/View/Update Payments	Administrators Consumers Financial Managers User service (limited) Marketplace Service (limited)
	Update service charges	Administrators Financial Managers
	Verify credit cards and payment methods (using inter-service communication)	Administrators Consumers
	Calculate the profit earned from payments by GadgetBadget	Administrators Financial Managers
Funding Service	Add/Remove/View/Update Funds	Administrators Funders Research Hub Service (limited) User service (limited)
	Update service charges	Administrators Financial Managers
	Verify credit cards and payment methods (using inter-service communication)	Administrators Funders
	Calculate the profit earned from funds by GadgetBadget	Administrators Financial Managers
Marketplace Service	Add/Remove/View/Update Products	Administrators Researchers User service (limited)
	Add/Remove/View/Update Product Categories	Administrators Researchers (limited)
	Get a list of payments made for a specific product (using inter-service communication)	Administrators Researchers

5.2.2. Non-functional Requirements

Availability

The web services should be available and accessible at any time, except during a system maintenance. A given service should be working fine independently even though another web service of the system is down or under maintenance except when the inter-service communication is required to fulfill a functional requirement.

Maintainability

Documentation of the APIs of the web services should be well-written and comprehensible by the users or the developers who are implementing client programs to communicate with the services and should mention the correct methods of troubleshooting, types of media consumed by the end points and the URI format.

Portability

Any type of client program should be able to call the API end points and request information from the web services regardless of the language or the technology stack used to develop the client programs.

Reliability

The web services should have backup features in case of an emergency, and it should support alternating the databases and should provide settings to do so accordingly. Each web service is preferred to use its own databases to minimize data transfer delays. In case of an attack, the user service should be able to handle user-accounts accordingly in a way that will minimize the system down-time.

Security

All user credentials and stored data must be secured and organized in a way that only system administrators and authorized users can view or alter them. User and Web service authorization should be properly implemented, and it is preferable to establish stateless authentication using token-based authentication using technologies like JWT. Public and Private Key pairs used to verify tokens must be securely stored to ensure preventing unauthorized access through token hi-jacking. Access permissions should be defined at each endpoint of APIs of the web services according to the user roles to restrict access to specific data. URLs should be designed in a way that they will not disclose any of the sensitive data.

Usability

An easy-to-use URI format should be designed and implemented for the endpoints of the web service APIs, which can be easily comprehended by users or client applications.

5.2.3. Technical Requirements

Technical requirements for the development of the web services are as follows.

IDE:	Eclipse versions between 2020.3 and 2020.9 inclusive
Server:	Apache Tomcat Server Version 9
Java Development Kit:	JDK8
Java Runtime Environment:	JRE 1.8
Operating System:	Any OS that supports eclipse versions mentioned above
Frameworks:	JAX-RS/Jersey 1.19
Libraries:	Bitbucket jose4j for JWT MySQL connector (JDBC) Google GSON for JSON serialization/de-serialization
Dependency Management:	Maven 3
Version Control:	Git and GitHub
Database Management:	MySQL Workbench/ WAMP Server/ XAMPP Server

Recommended Technical Requirements for implementing the web services of GadgetBadget are as follows.

Server:	Apache Tomcat Server Version 9 or above (if hosted locally. Any cloud provided who supports war package hosting will be compatible as well.)
Java Runtime Environment:	JRE 1.8
Operating System:	Any OS that supports JDK 8/JRE 1.8 and Apache Tomcat Server 9
Database Server:	MySQL server/ PhpMyAdmin (If hosted locally) or Any cloud provider that supports MySQL database hosting

5.3. Requirements Modelling

Since the services were developed individually, Modelling UML diagrams had been done separately per each web service. Use case diagrams per each service identified are given below.

Use Case Diagram of the User Service

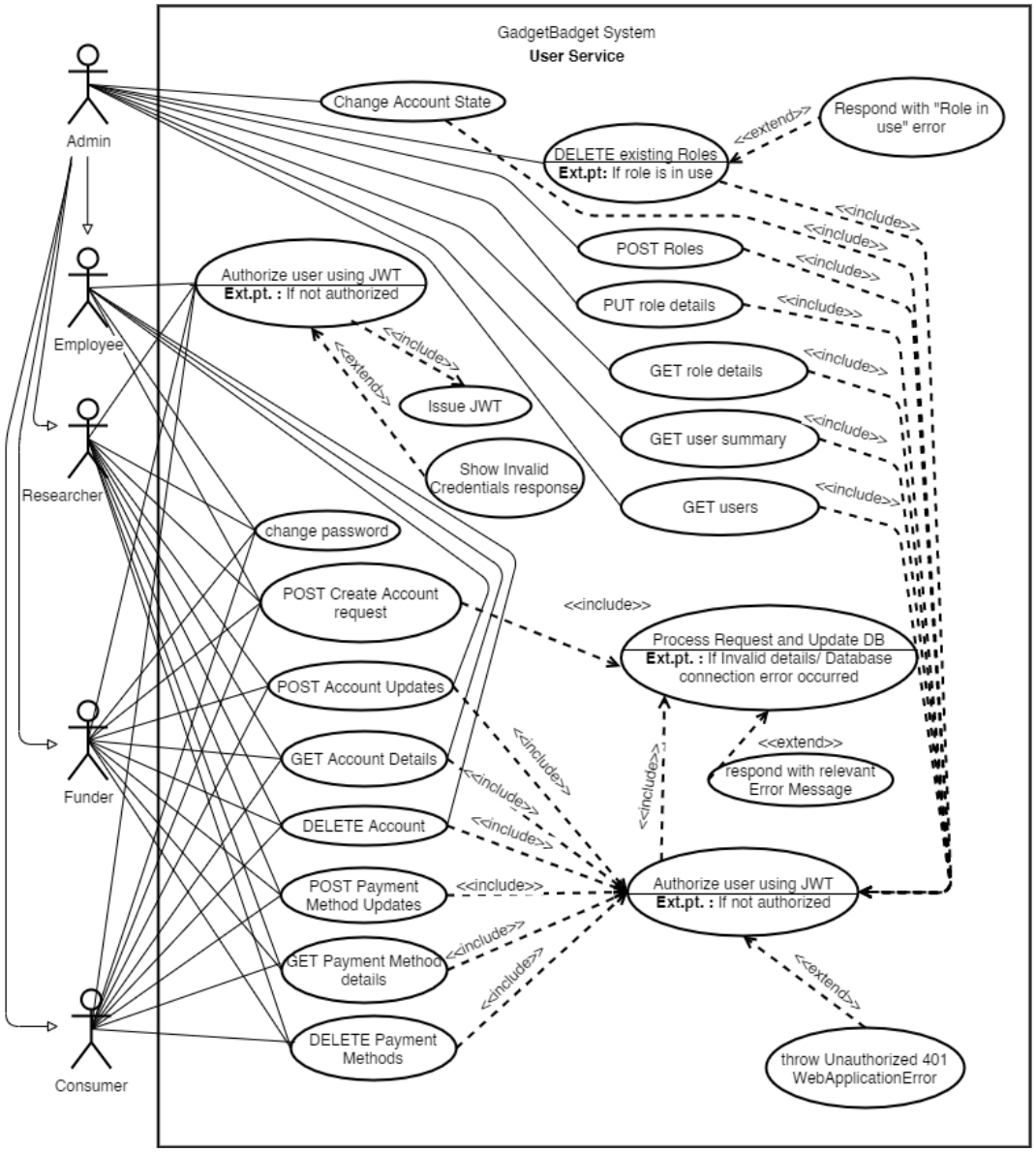


Figure 5.3. 1 User Service Use Case Diagram

In user service, administrators can perform any of the tasks performed by other users shown in the use case diagram. All tasks are well authorized before performed by the system.

Use Case Diagram of the Research Service

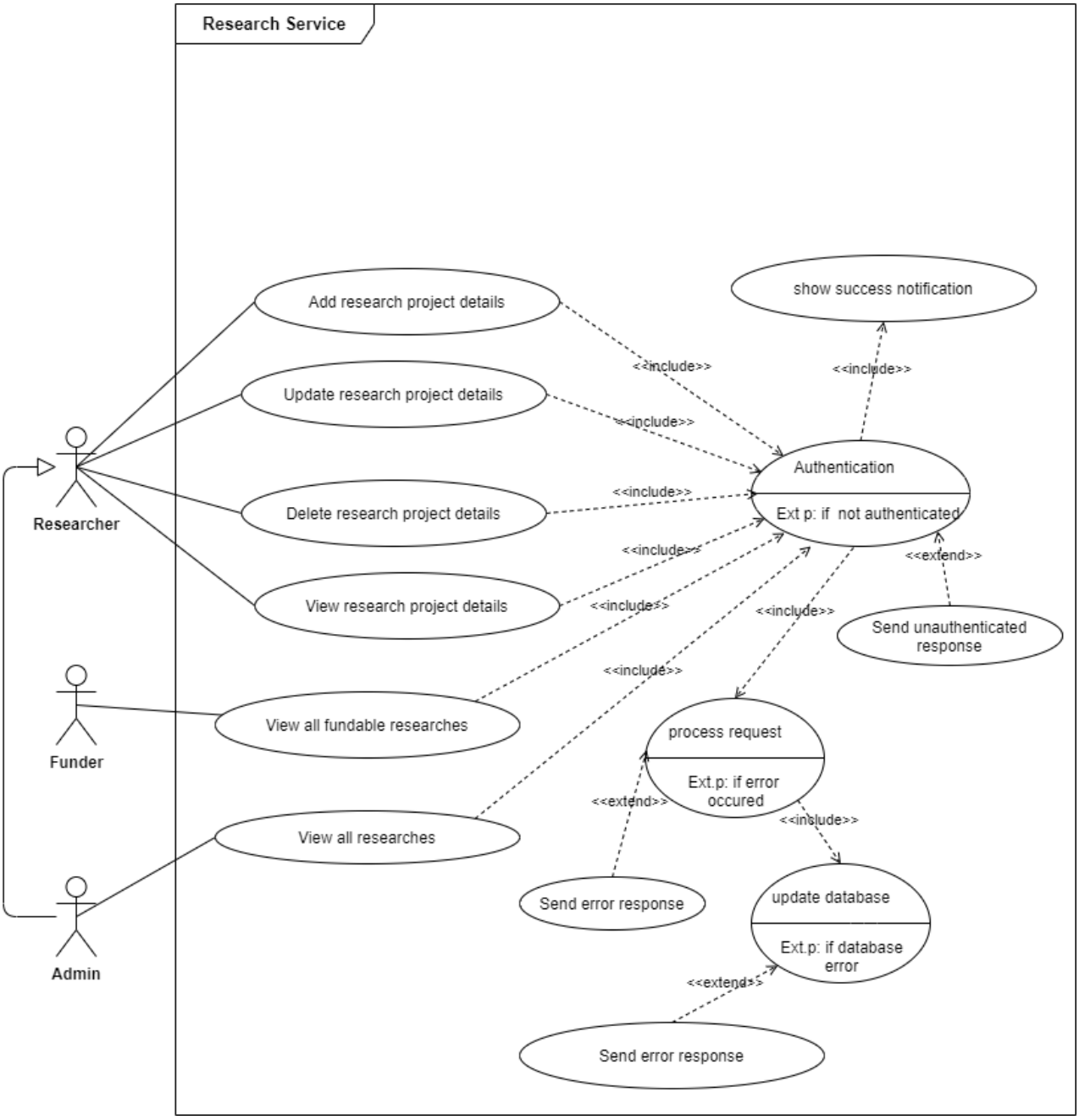


Figure 5.3. 2 ResearchHub service Use Case Diagram

In Research service, the administrators can perform any task that the other users are able to perform. All tasks are well authorized before performed by the system.

Use Case Diagram of the Payment Service

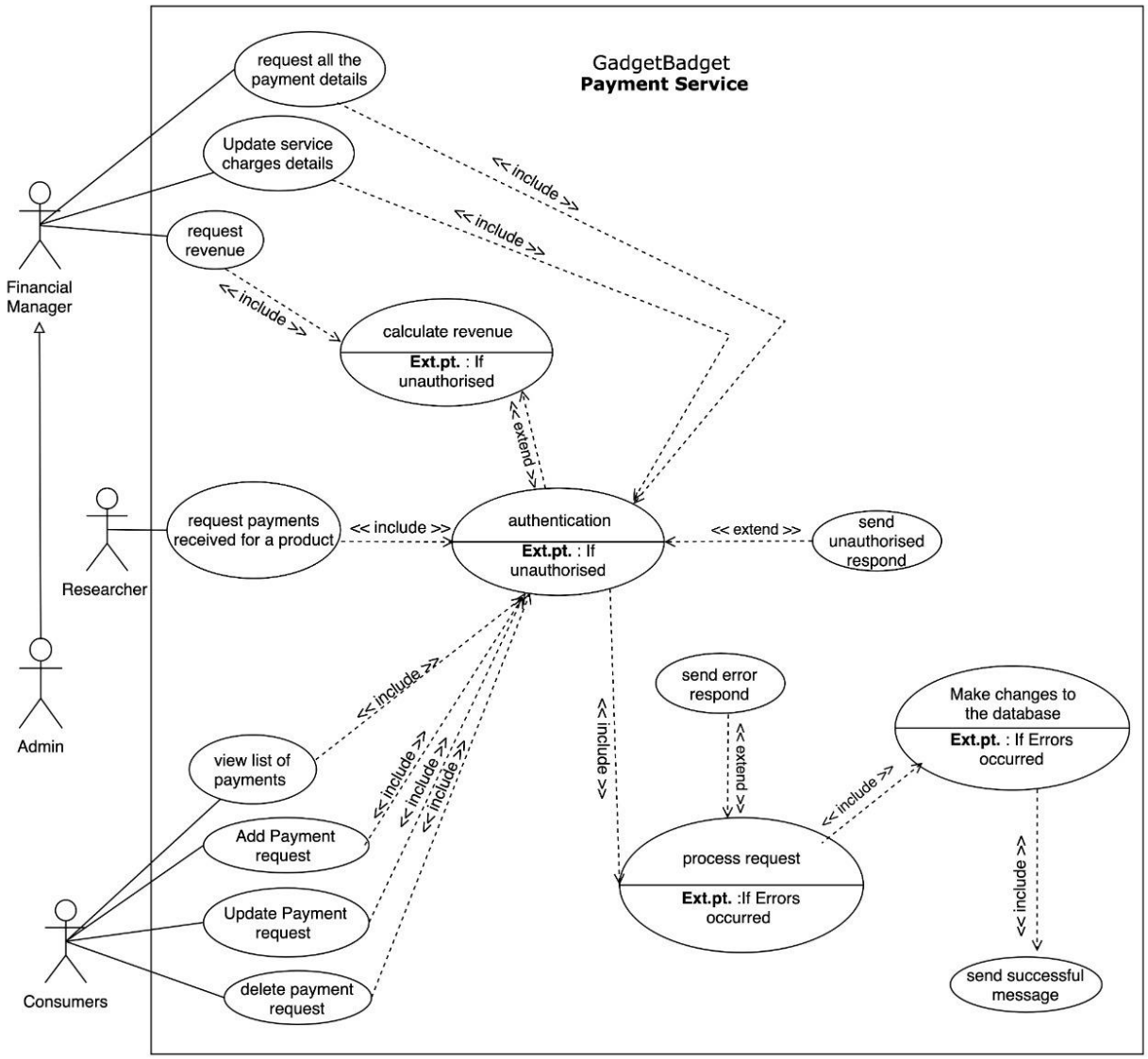


Figure 5.3. 3 Payment Service Use Case Diagram

In Payment service, the administrators can perform any task that the other users are able to perform. All tasks are well authorized before performed by the system. Financial Managers of the GadgetBadget company can calculate the total profit earned through received payments.

Use Case Diagram of the Marketplace Service

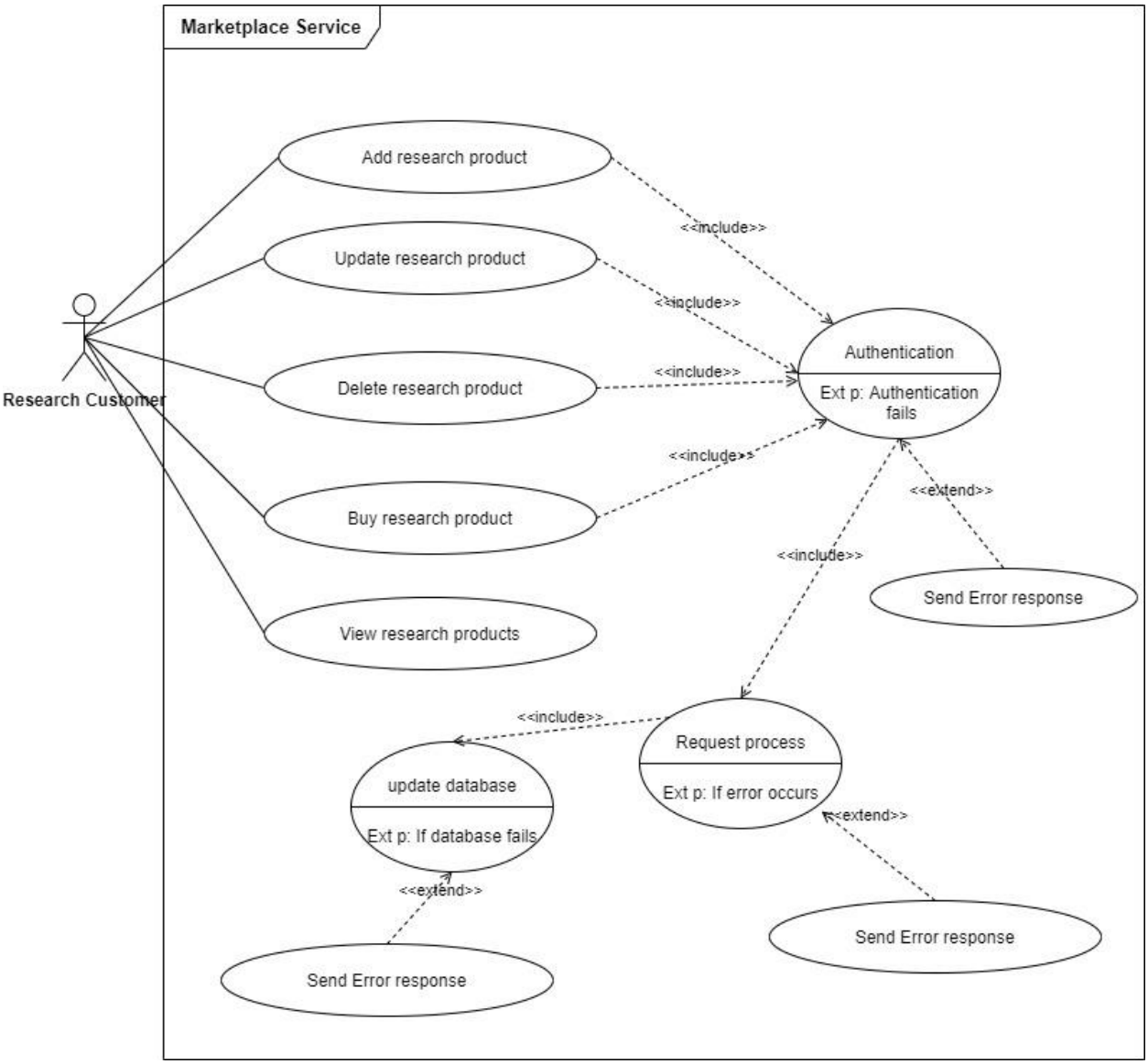


Figure 5.3. 4 Marketplace Service Use Case Diagram

Marketplace service allows the administrators to perform any task the other users can perform. All tasks are well authorized before performed by the system. Error responses and other responses are sent in the form of JSON objects.

Use Case Diagram of the Funding Service

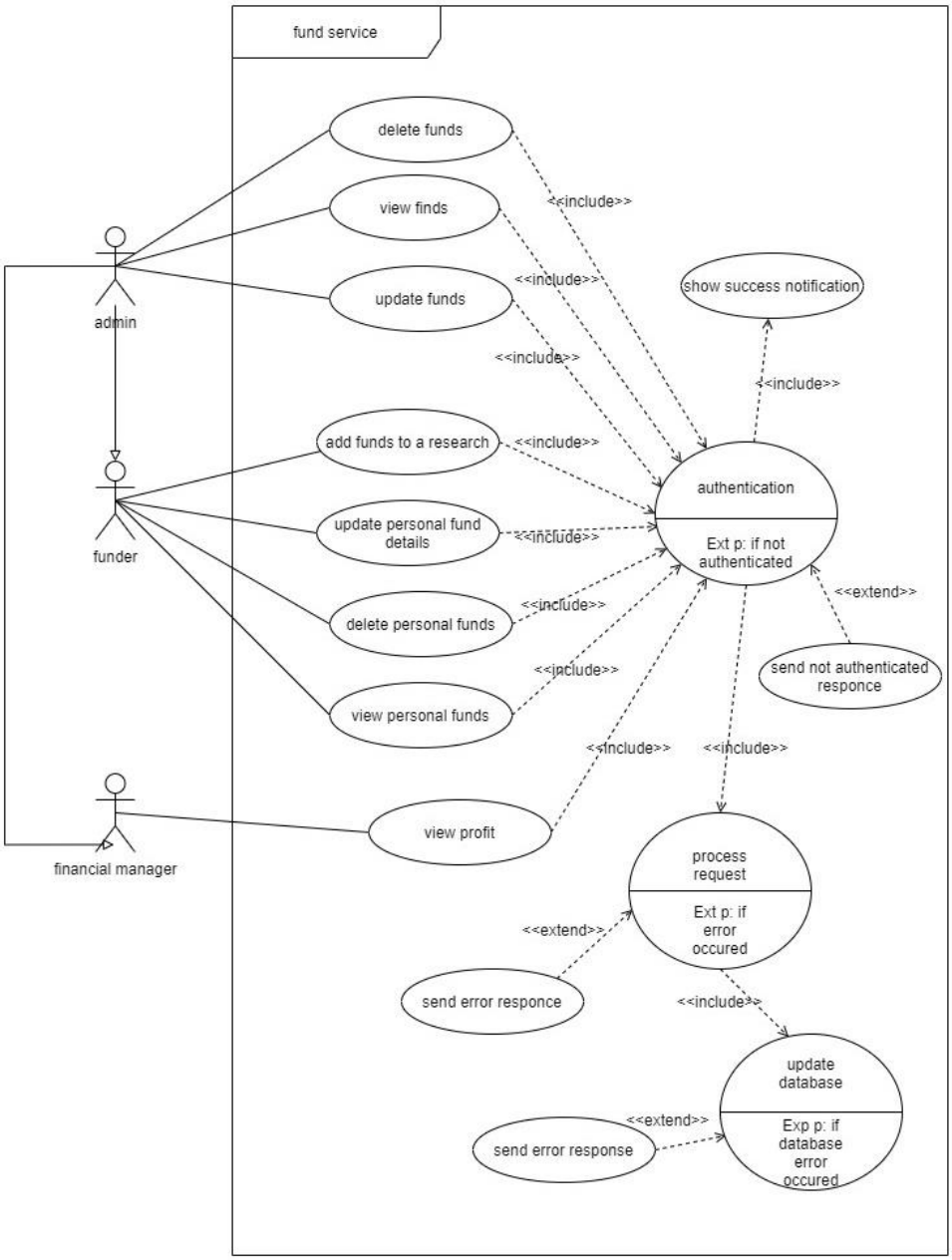


Figure 5.3. 5 Funding Service Use Case Diagram

In funding service, the administrators can perform any task that the other users are able to perform. All tasks are well authorized before performed by the system. Financial Managers of the GadgetBadget company can calculate the total profit earned through received funds.

6. Overall System Design

6.1. Overall System Architecture

The overall design of the GadgetBadget system is shown in the following figure. Stakeholders who are directly interacting with the system may use several client applications based on the nature of their role and note that the client applications have not been developed under this project. A test client was used instead of the client applications mentioned in the figure.

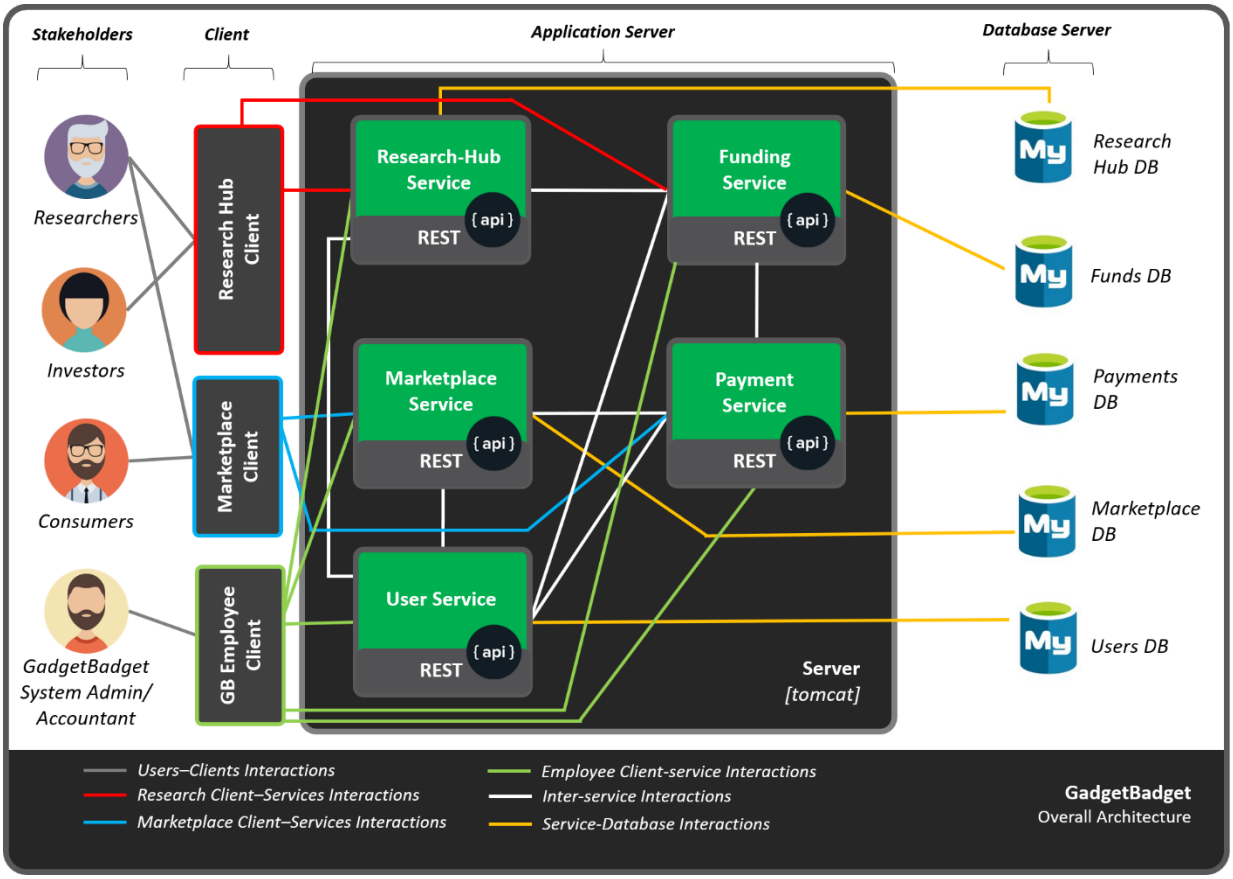


Figure 6.1 1 GadgetBadget Overall System Architecture

6.2. Overall Database Architecture

Each group member developed individual databases per each web service identified. To design and develop the databases, MySQL was the language of choice. Five databases that are independent and can be separately hosted. Inter-service communication in case a different microservice wants to access another service's data.

The primary keys of each table were designed according to a specific format to utilize it in the best way possible. Each primary key consists of 10 characters where the first two are capital letters, which is known as the prefix and is unique for each table. The immediate two characters after the prefix are the last two digits of the ongoing year, and the last six digits are designed to get auto incremented. Therefore, it eliminated the chance of running out of unique primary keys for an extensive period. The format of all the primary keys is given below in a tabular view. (Y represents year and D represents Digit.)

Web Service	**Database Name**	**Relation Name**	**Primary Key Format**
User Service	gadgetbadget_users	employee (EM)	EMYYDDDDDD
			ADYYDDDDDD
			FMYYDDDDDD
		consumer (CN)	CNYYDDDDDD
		funder (FN)	FNYYDDDDDD
		researcher (RS)	RSYYDDDDDD
		role	ADMIN
			CNSMR
			EMPLY
			FNMGR
			FUNDR
			RSCHR
Research Hub Service	gadgetbadget_researchhub	researchproject (RP)	RPYYDDDDDD
		research_category (RC)	RCYYDDDDDD
Marketplace Service	gadgetbadget_marketplace	product (PR)	PRYYDDDDDD
		product_category (PC)	PCYYDDDDDD
Payment Service	gadgetbadget_payments	payment (PY)	PYYYDDDDDD
		service_charges	PYT
Funding Service	gadgetbadget_funding	fund (FD)	FDYYDDDDDD
		service_charges	FND

Additional tables with the prefix "_seq" were used to generate the above unique key. When inserting a new row to one of the above tables, a trigger will run and inserts to the _seq table first. Then its auto-incremented key is taken and added to the prefix with the last two digits of the year and zero filling the rest of the digits using MySQL commands.

Other than the triggers used for primary key generation, on-insert and on-update triggers have been designed for inserting the current timestamp, and the dates are getting auto inserted/updated with these triggers whenever data is inserted or updated in most of the relations. Moreover, stored procedures have been used in places where multiple inserts, deletes, reads, or updates had to be

done. Using stored procedures did reduce the number of database calls which decreases the cost of database designs.

6.3. Additional Diagrams

Please refer appendix section (A-1) to find additional diagrams related to overall system.

7. Individual Sections

7.1. Users Service

User Service is responsible for managing all users who are directly interacting with all web services of GadgetBadget system. It does maintain user details, maintain payment methods added by the users, authenticate users, and Implement a user Authorization mechanism to define permission at end points of different services within the system using JSON Web Tokens with stateless authentication. [Developed by: IT19069432 Dissanayake D.M.I.M.]

7.1.1. Service Design

The user service has been designed according to the MVC architecture. There are two resources, namely, "/security" and "/users". Security resource has sub-resources directly related to the API endpoint security. User resource contains all user account and payment method-related API endpoints. *A complete list of endpoints provided by the user service has been listed in the appendix section of this report. When receiving a request from client applications, the resource classes mentioned above work as connectors that receive HTTP requests and enable communication with the server model. Model classes contain the database querying logic to retrieve data from the backend database upon requests. There are several packages in the server model, namely, com.gadgetbadget.user.model, com.gadgetbadget.user.util, and com.gadgetbadget.user.security.

The security package contains all the security-related code including, JWT token generation, JWT token validation, Server Token generation, Server token validation, Request Filtering for user/ service authentication, and user/service authorization. All services and users are authenticated using JWTs issued by user service and are services are authenticated by unique JWT service tokens. (*many of the diagrams are attached in the appendix section due to limited space caused by the page limit)

7.1.1.1. REST URI Design

URIs have been developed in the following formats shown. A complete list of URIs

Collections:
http://localhost:8081/UserService/userservice/security/roles
http://localhost:8081/UserService/userservice/users/consumers
http://localhost:8081/UserService/userservice/users/consumers/CN21000053/payment-methods

Single Items:
http://localhost:8081/UserService/userservice/users/consumers/CN21000016
http://localhost:8081/UserService/userservice/users/consumers/CN21000053/payment-methods?retrieve=true (credit card number is sent in the payload for security concerns.)

Inter-service communication:
http://localhost:8081/UserService/userservice/users?stats=true (Get statistics from all services)

Security:
http://localhost:8081/UserService/userservice/security/authenticate (Authenticate to get JWT)
http://localhost:8081/UserService/userservice/users/AD21000001/password (Change Password)
http://localhost:8081/UserService/userservice/users/AD21000009?deactivate=true (Deactivate)

7.1.1.2. Database Design – Please refer the appendix section. (A-2)

7.1.1.3. Activity Diagrams – Please refer the appendix section. (A-2)

7.1.1.4. Class Diagram

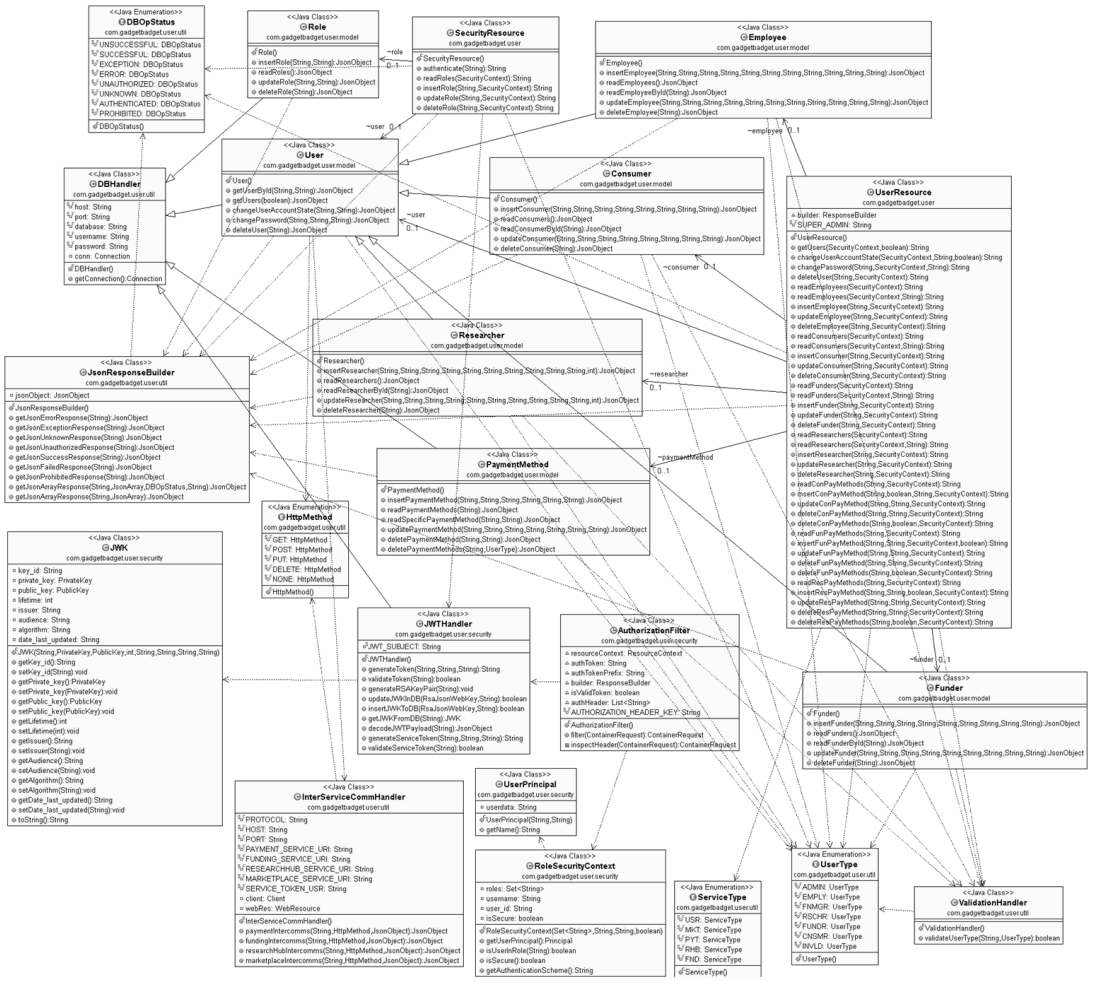


Figure 7.1.1.4. 1 User service class diagram

7.1.2. Service Development and Testing

7.1.2.1. Tools Used

- **Eclipse + JRE 1.8/JDK 8 + Apache Tomcat 9** with **Jersey Library** were used to develop the code base of the user service. Reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 1.19.
- **Maven 3.7** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
- **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development related issues.
- **JWT** was used to implement stateless authentication of both users and services.
- **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in responses.
- **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.

7.1.3. Assumptions

- Users must be authenticated first and then authorized based on the roles and IDs.
- Only Administrators can Activate/ Deactivate user accounts.
- Details of the GadgetBadget employees who are directly involved are also saved in the system
- Stateless authentication makes the web service less costly due to less database calls.
- JWT is only valid for 20 minutes after generation.

7.2. ResearchHub Service

Research hub service is responsible for managing research projects of researchers. Administrators manage the research categories. The service is designed as a REST API where the design of the web service is the MVC architecture where the server model and the connecter are implemented using java language and jersey framework. Each request that reaches the research hub service is filtered and the users and services are properly authorized before granting access to any of the endpoints. Researchers can obtain a list of funds received for the research projects they have uploaded through inter-service communication facilities provided by the research hub service. [Developed by: IT19167510 Bandara J.M.S.A.]

7.2.1. Service Design

Main resource was developed as “/research-projects” and within that there are sub resources, namely, “/collaborators” and “/categories”.

7.2.1.1. REST URI Design

Collections:
http://localhost:8081/ResearchHubService/researchhubservice/research-projects
http://localhost:8081/ResearchHubService/researchhubservice/research-projects/RP21000001/collaborators

Single Items:
http://localhost:8081/ResearchHubService/researchhubservice/research-projects/RP21000001

Inter-service Communication:
http://localhost:8081/ResearchHubService/researchhubservice/research-projects/RP21000002/funds

7.2.1.2. Database Design – Please refer the appendix section. (A-4)

7.2.1.3. Activity Diagrams – Please refer the appendix section. (A-4)

7.2.1.4. Class Diagram

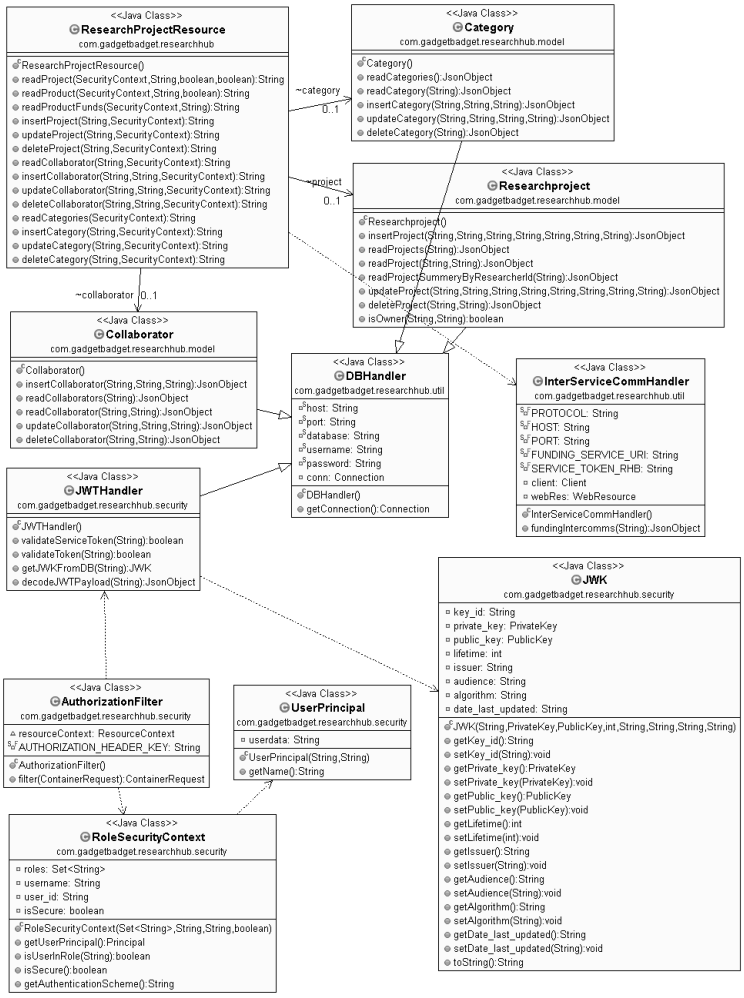


Figure 7.2.1.4. 1 Research Hub Service class diagram

7.2.2. Service Development and Testing

- **Eclipse, JDK 8, JRE 1.8, Tomcat 9 Server, Jersey Library** were used to develop the research hub service along with JAX-RS and Jersey 1.19.
- **Maven 3.7** was used for dependency management to integrate plugins, dependencies easily.
- A **GitHub** repository was used for team collaborations to be able to trace back to previous versions in case of programmatic errors or any other issues.
- **JSON** was selected as the main MediaType both consumed and produced at each endpoint.
- **Postman** was used to test the research hub service's all end points to make sure they are error free.

7.2.3. Assumptions

- All users who reach research hub service must provide a JWT to properly get authorized.
- A research project only belongs to one research category.
- Researchers can add as many collaborators as they want.

7.3. Payment Service

Payment service was designed according to the MVC architecture by having a well-coded server model with the business logic of the GadgetBadget system implemented properly. Consumers can buy products available in the marketplace by making payments via the payment service where the payment service will verify the payment method by establishing a client-server communication with the user service to obtain the payment information. Financial managers can visit "/payments/profit" absolute path of the service to obtain the profit received through the payments consumers have made. [Developed by: IT19075754 Jayasinghe D.T.]

7.3.1. Service Design

The payment service has the main resource named as "/payments" where all HTTP methods related to payments are directed to. Another resource named "/service-charges" is also available which is only authorized to be accessed by administrators and financial managers to update service charges.

7.3.1.1.REST URI Design

Collections:
http://localhost:8081/PaymentService/paymentservice/payments

Single Items:
http://localhost:8081/PaymentService/paymentservice/payments/PY21000003

Inter-service Communication: Uses when inserting and updating payments, to verify credit cards.

7.3.1.2. Database Design – Please refer the appendix section. (A-3)
7.3.1.3. Activity Diagrams – Please refer the appendix section. (A-3)

7.3.1.4. Class Diagram

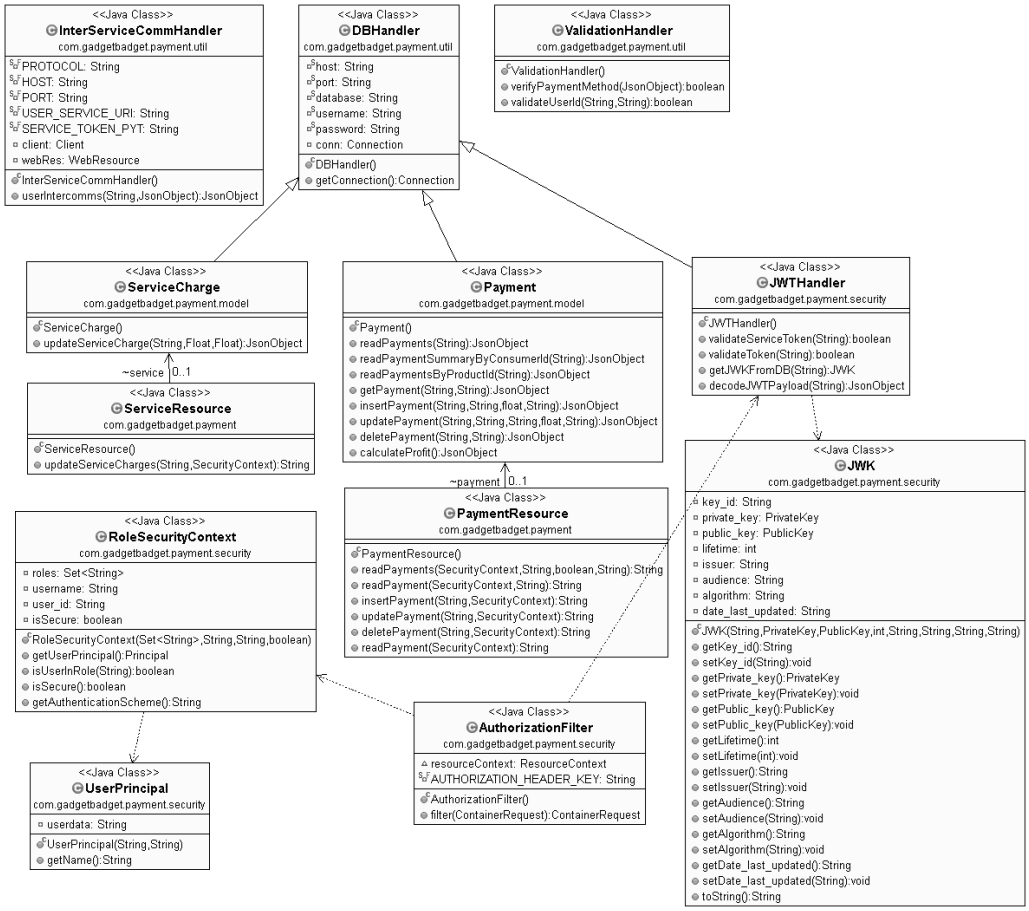


Figure 7.3.1.4. 1 Payment Service class diagram

7.3.2. Service Development and Testing

- **Eclipse, JDK 8, JRE 1.8, Tomcat 9 Server, Jersey Library** were used to develop the payment service along with JAX-RS and Jersey 1.19. according to REST architecture.
- **Maven 3.7** was used for dependency management to integrate dependencies needed.
- **GitHub** repository was used for team collaborations to be able to have a backup with each team member in case the changes made would cause errors.
- **JSON** was selected as the same Media Type at all endpoints for proper data identification.
- **Postman** was used to test the APIs and make sure they are secure and not buggy.

7.3.3. Assumptions

- Only administrators and financial managers can request to get the profit calculation.
- All Payment dates and current service charges are automatically inserted/applied. (triggers)
- An authenticated yet non administrator can only view payments they have made.

7.4. Marketplace Service

Having a server model and a controller to direct request to the relevant methods, marketplace service follows the MVC architecture when it comes to the classes that have been designed. Researchers can manage their products in the marketplace and consumers can view all the products and categories available. Marketplace service mainly focuses on the products and does not get involved with payment related operations. [Developed by: IT19211824 Indrahenaka H.R.T.V.]

7.4.1. Service Design

Marketplace service has "/products" as the main resource path where all product related HTTP methods are implemented with JSON Media Types at all end points for easily grasping and understand the format of the incoming or outgoing payloads. Within the "/product" resource, there are several sub-resources which are used to manage product categories and the researchers can view a list of payment received for each product through inter-service communication established between marketplace service and the payment service.

7.4.1.1. Database Design – Please refer the appendix section. (A-5)
7.4.1.2. Activity Diagrams – Please refer the appendix section. (A-5)
7.4.1.3. Class Diagram

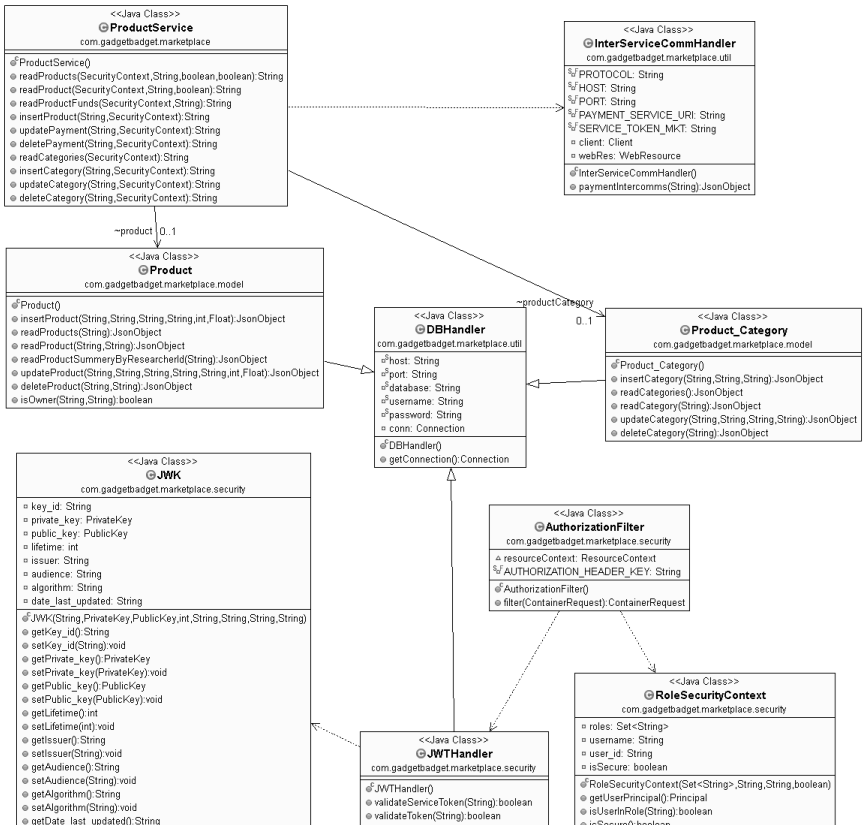


Figure 7.4.1.3. 1 Marketplace Service class diagram

7.4.1.4.REST URI Design

Collections:
http://localhost:8081/MarketplaceService/marketplaceservice/products

Single Items:
http://localhost:8081/MarketplaceService/marketplaceservice/products/PR21000001

Inter-service Communication:
http://localhost:8081/MarketplaceService/marketplaceservice/products/PR21000001/payments

7.4.2. Service Development and Testing

- **Eclipse, JDK 8, JRE 1.8, Tomcat 9, Jersey** was used to develop the payment service using JAX-RS and Jersey 1.19. RESTful web service.
- **Maven 3.7** was used for dependency management to integrate dependencies necessary.
- **GitHub** repository was maintained while having a local backup of the project as well.
- **JSON** objects were used to transfer data between methods and APIs.
- **Postman** was used to test the marketplace service to make sure its bug-free before the system integration.

7.4.3. Assumptions

- Researchers want to see a complete list of payments received for a product they have uploaded.
- Product categories can be only altered by Administrators.
- Only researchers can upload products while only consumers can purchase them.

7.5. Funding Service

Funding service was designed according to REST and MVC while following the best practices of REST URI formats for accessing resources. Funders can place funds for any research project they wish to back and the credit card details are required to continue the fund placement. It is validated upfront with inter-communications established with the user service and if the card is invalidated, placing the fund will get terminated. [Developed by: IT19237282 H.C.K. de silva]

7.5.1. Service Design

Funding service has the main resource as "/funds" and it contains all fund related HTTP methods as sub resources. There is a special sub resource for acquiring the profit earned through funds that have been placed so far and it is only accessible by Administrators and Financial Managers only.

7.5.1.1. REST URI Design

Collections:
http://localhost:8081/FundingService/fundingservice/funds

Single Items:
http://localhost:8081/FundingService/fundingservice/funds/FD21000004

Inter-service Communication: Used when placing a fund or updating a fund. Obtains credit card details saved in the user service and verifies it before placing the fund. Happens internally.

7.5.1.2. Database Design – Please refer the appendix section. (A-6)

7.5.1.3. Diagrams – Please refer the appendix section. (A-6)

7.5.1.4. Class Diagram

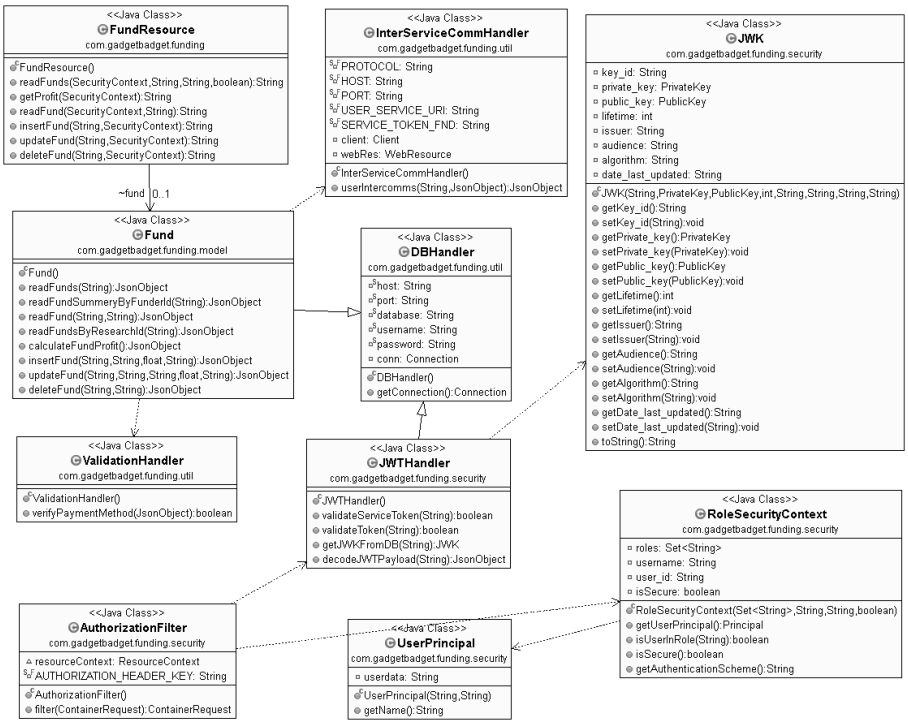


Figure 7.5.1.4. 1 Funding Service class diagram

7.5.2. Service Development and Testing

- **Eclipse, JDK 8, JRE 1.8, Tomcat 9, Jersey, JAX-RS –** for web service development with java with compatible versions of other mentioned software with jersey 1.19
- **Maven 3.7** as the dependency management to manage the external dependencies better.
- **GitHub** repository to manage the version controlling and team collaboration.
- **JSON** as the data type of the requests and responses of resources.
- **Postman** to test the developed API endpoints to make sure they are working as expected.

7.5.3. Assumptions

- Funding managers can see the profit earned through received payments.
- Funds are placed using online payment methods, preferably credit cards.
- Credit cards have a fixed length number and should be verified before placing the fund in db.

8. System Integration

System Integration of all the services was not quite difficult since all the group members were constantly committing the changes with each major feature added to their web services to the same branch in the GitHub repository. Creating additional branches was not a requirement since we were developing separate projects although they were in the same branch, hence, merging branches was not necessary at the end of the web services development.

After pulling all the changes made for all the projects, they were updated using maven – update project feature in the eclipse IDE and cleaned beforehand. Then all the projects were added to the same Tomcat server before starting it and ran all the services together to enable inter-service communication between all five web services and tested the whole system at once using postman to make sure that all the web services are running accordingly and fully supports inter-service communication without any issues.

Note that even if the services are hosted in different servers, inter-communication can be simply enabled by providing the new host server address and ports in the InterServiceCommHandler.java class of each web service.

The databases of all five web services were hosted in each group members MySQL server to enable access for the web services and notice that they can be hosted in various servers as well. If that is the case, the new server address and the post should be updated in the DBHandler.java class which is an easy fix.

Integrated System Testing Results can be accessed in the appendix section of this report.

9. References

[1]. ieee-dataport.org, 'How to Cite References: IEEE Documentation Style', Available: https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf [Accessed: 21-Apr-2021]

[2]. Karl E. Wiegers, 'IEEE Software Requirement Specification Template', 1999, Available: https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc [Accessed on: 15-Apr-2021]

[3]. "IEEE Standard for Software Test Documentation," in IEEE Std 829-1983, vol., no., pp.1-48, 18 Feb. 1983, DOI: 10.1109/IEEESTD.1983.81615.

[4]. M. Bhasi, Smiju Sudevan and K.V. Pramod, "Existing Software Stakeholder Practices an Overview", *International Journal of Computer Applications*, Volume 102, No.3, p. 8-13, September 2014 [Abstract], DOI: 10.5120/17793-8593, Available: ResearchGate, https://www.researchgate.net/publication/284411602_Existing_Software_Stakeholder_Practices_an_Overview [Accessed: 03-Apr-2021]

[5]. bawiki.com, 'Stakeholder Onion Diagram', Available: http://bawiki.com/wiki/Stakeholder-Onion-Diagram.html [Accessed on: 13-Mar-2021]

[6]. Karl E. Wiegers, 'IEEE Software Requirement Specification Template', 1999, Available: https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc [Accessed on: 14-Mar-2021]

[7]. docs.github.com, 'Creating a personal access token', https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token [Accessed on: 13-Apr-2021]

[8]. eclipse-ee4j.github.io, 'Jersey 1.19.1 User Guide', https://eclipse-ee4j.github.io/jersey.github.io/documentation/1.19.1/index.html [Accessed on: 7-Apr-2021]

[9]. www.codejava.net, '' JDBC Examples for Calling Stored Procedures (MySQL), https://www.codejava.net/java-se/jdbc/jdbc-examples-for-calling-stored-procedures-mysql [Accessed on: 6-Apr-2021]

[10]. mkyong.com, 'JDBC Callable Statement – Stored Procedure OUT parameter example', https://mkyong.com/jdbc/jdbc-callablestatement-stored-procedure-out-parameter-example/ [Accessed on: 7-Apr-2021]

[11]. medium.com, '10 Best Practices for Better RESTful API', https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-cbe81b06f291 [Accessed on: 16-Apr-2021]

[12].stackoverflow.com, ‘Jersey POST Method is receiving null values as parameters’, https://stackoverflow.com/questions/10357041/jersey-post-method-is-receiving-null-values-as-parameters [Accessed on: 12-Apr-2021]

[13].moesif.com, ‘REST API Design Best Practices for Parameter and Query String Usage’, https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Parameters-and-Query-String-Usage/ [Accessed on: 13-Apr-2021]

[14].stackoverflow.com, ‘How can I map semicolon-separated PathParams in Jersey?’, https://stackoverflow.com/questions/5503272/how-can-i-map-semicolon-separated-pathparams-in-jersey/5512784 [Accessed on: 14-Apr-2021]

[15].stackoverflow.com, ‘REST API Best practice: How to accept list of parameter values as input [closed]’, https://stackoverflow.com/questions/2602043/rest-api-best-practice-how-to-accept-list-of-parameter-values-as-input [Accessed on: 14-Apr-2021]

[16].jwt.io, ‘Libraries for Token Signing/Verification’, https://jwt.io/ [Accessed on: 13-Apr-2021]

[17].bitbucket.org, ‘Wiki - jose4j / Home’, https://bitbucket.org/b_c/jose4j/wiki/Home [Accessed on: 13-Apr-2021]

[18].bitbucket.org, ‘Wiki - jose4j / JWT Examples’, https://bitbucket.org/b_c/jose4j/wiki/JWT%20Examples [Accessed on 13-Apr-2021]

[19].antoniogoncalves.org, ‘Securing JAX-RS Endpoints with JWT’, https://antoniogoncalves.org/2016/10/03/securing-jax-rs-endpoints-with-jwt/ [Accessed on: 12-Apr-2021]

[20].stackoverflow.com, ‘How to add Headers on RESTful call using Jersey Client API’, https://stackoverflow.com/questions/18342456/how-to-add-headers-on-restful-call-using-jersey-client-api [Accessed on: 14-Apr-2021]

[21].merixstudio.com, ‘9 Best Practices to implement in REST API development’, https://www.merixstudio.com/blog/best-practices-rest-api-development/ [Accessed on: 16-Apr-2021]

[22].merixstudio.com, ‘API documentation - a few tips that will help you write it well’, https://www.merixstudio.com/blog/api-documentation-few-tips-will-help-you-write-it-well/ [Accessed on: 16-Apr-2021]

[23].medium.com, ‘REST: Good Practices for API Design’, https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9 [Accessed on: 16-Apr-2021]

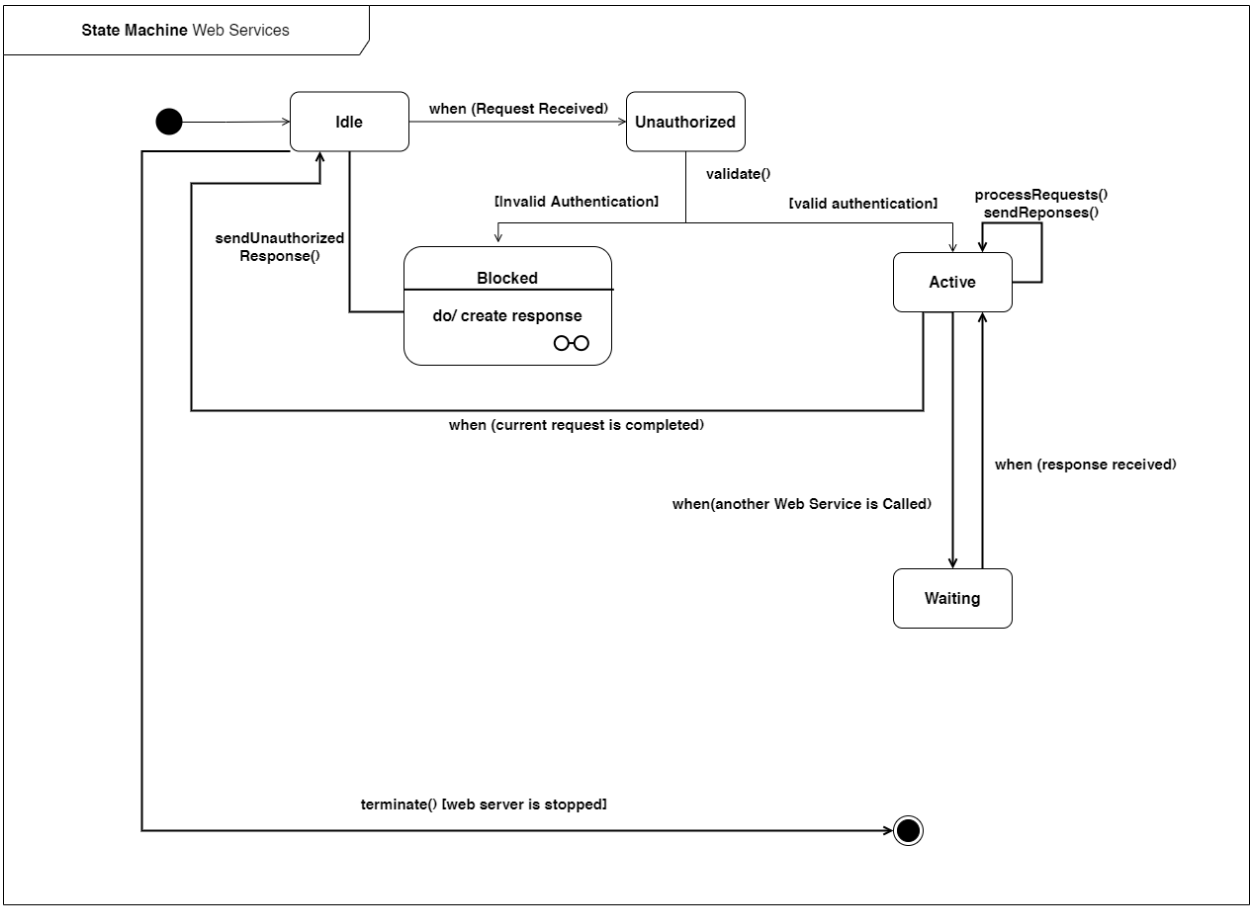
[24].linkedin.com, ‘RESTful API Design Tips’, https://www.linkedin.com/pulse/restful-api-design-tips-ishani-shah?articleId=6699064907759017984 [Accessed on: 17-Apr-2021]

Appendix

Appendix A: Design Diagrams

A-1. Overall System Diagrams

State Machine Diagram for all web services



Assumptions:

1. Web services get terminated when the server is offline

Figure A-1- 1 State Machine of All Services

Deployment Diagram of all web services (in localhost)

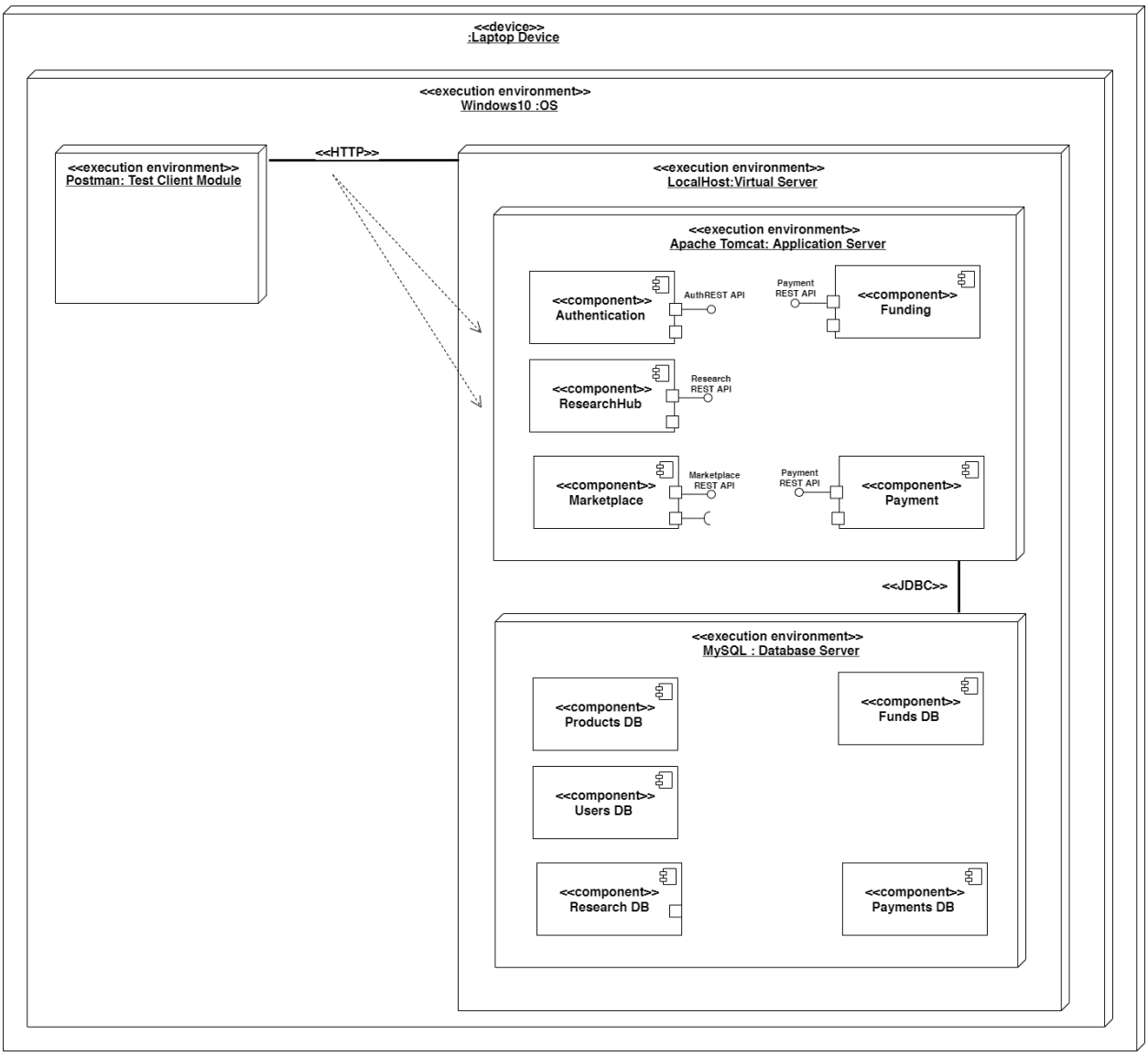


Figure A-1- 2 Physical Diagram of the GadgetBadget web services

A-2. User Service Diagrams

Database Design – EER Diagram of the User Service

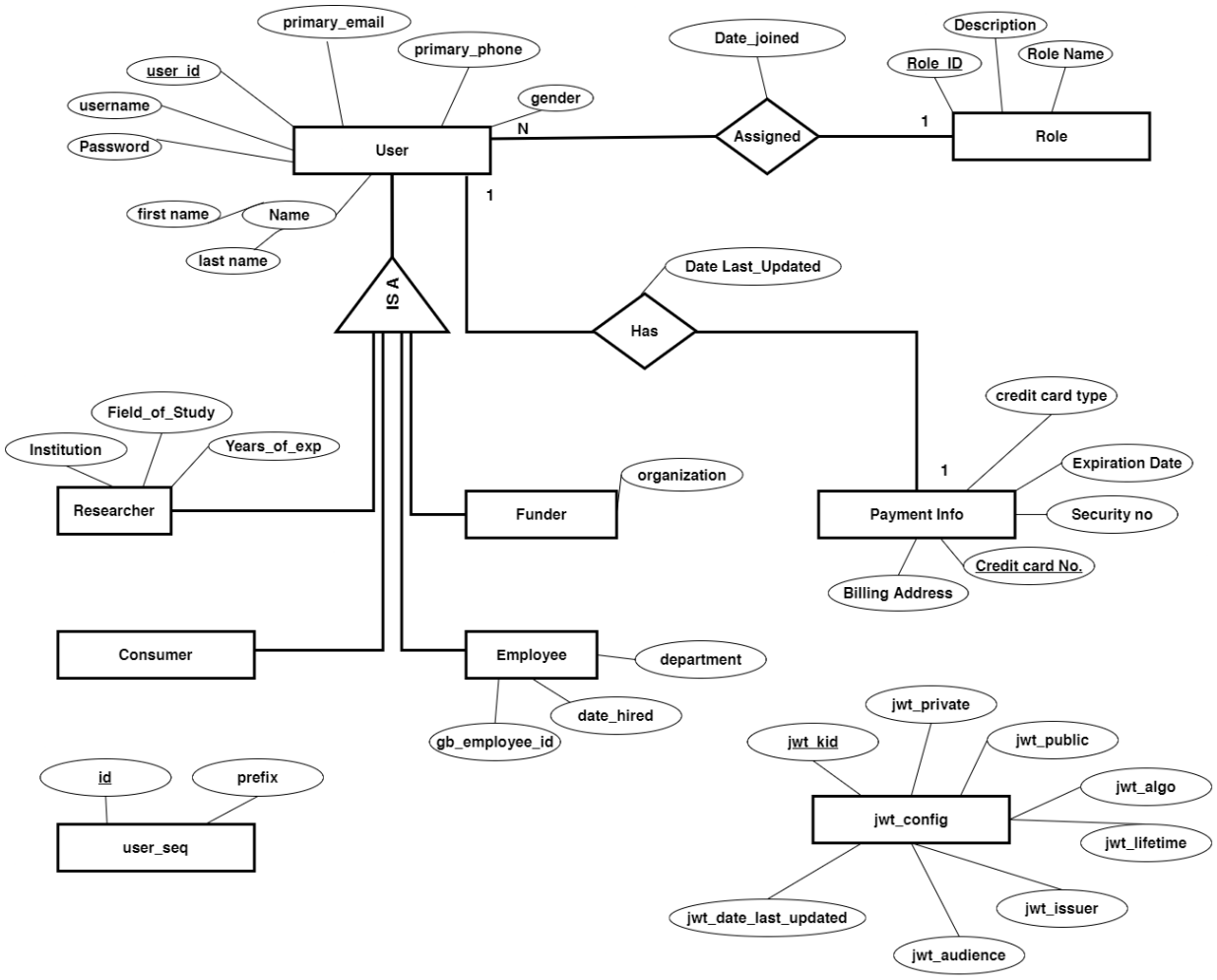


Figure A-2- 1 User service database design

Activity Diagrams of the User Service

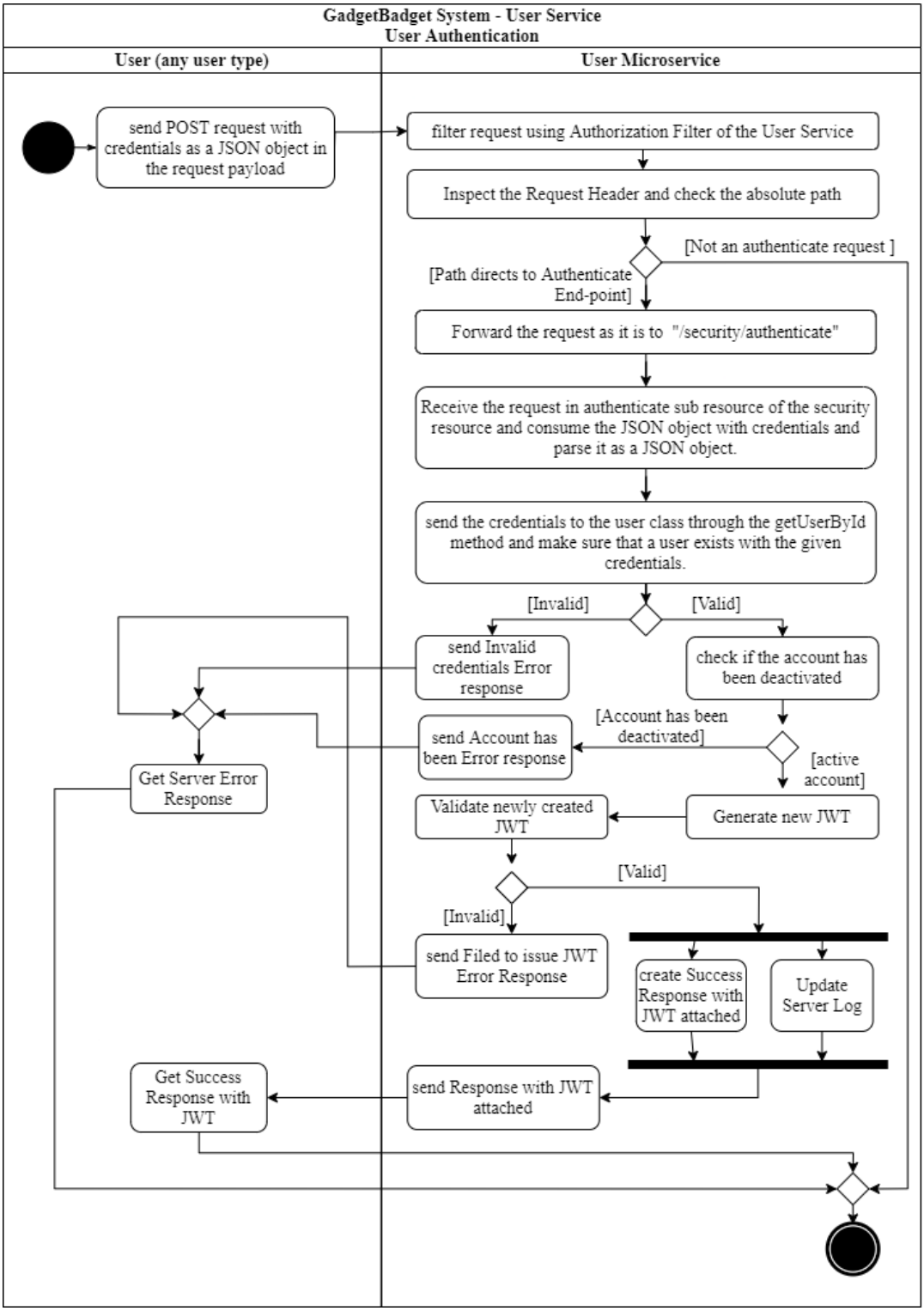


Figure A-2- 2 User Authentication Activity Diagram

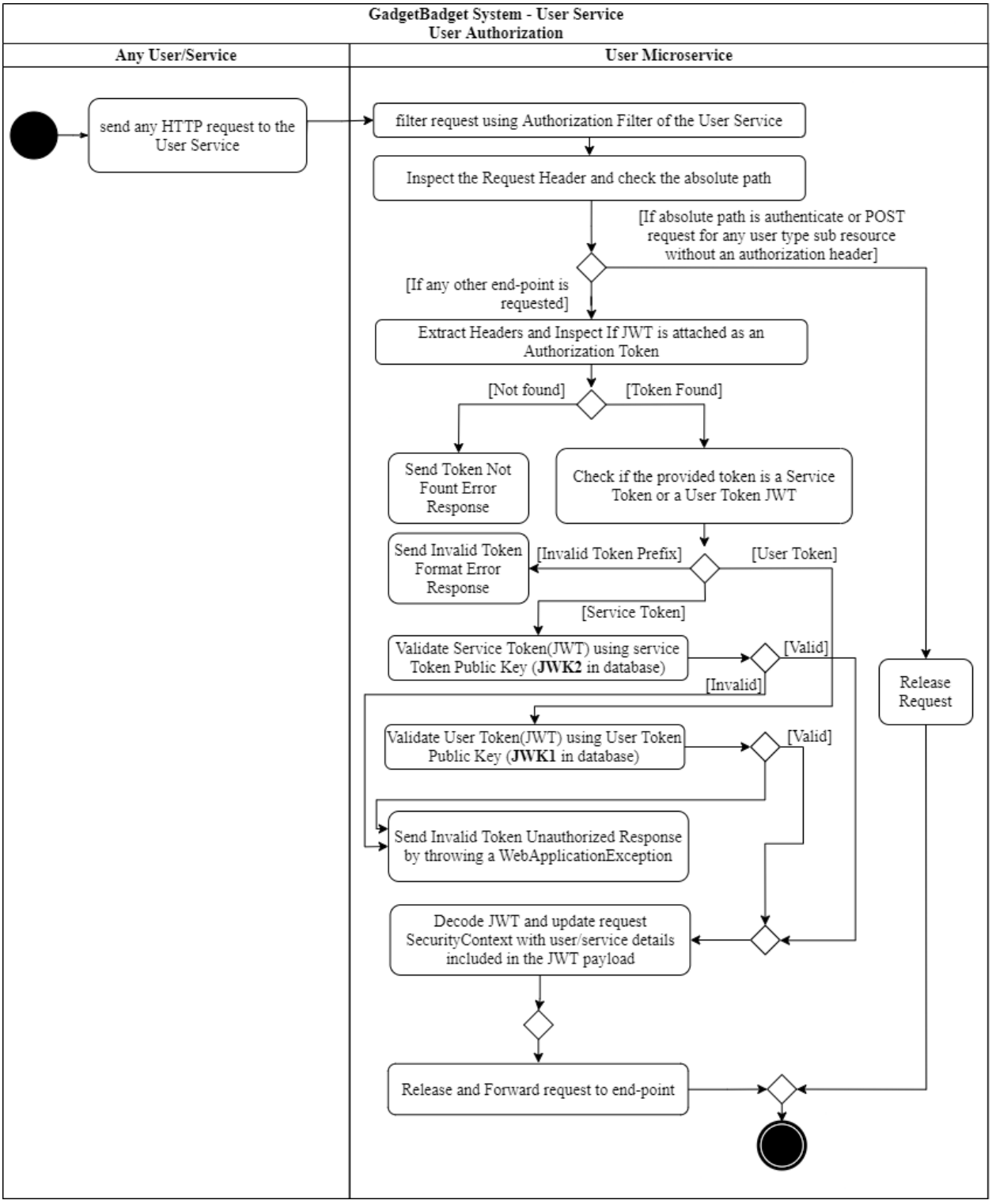
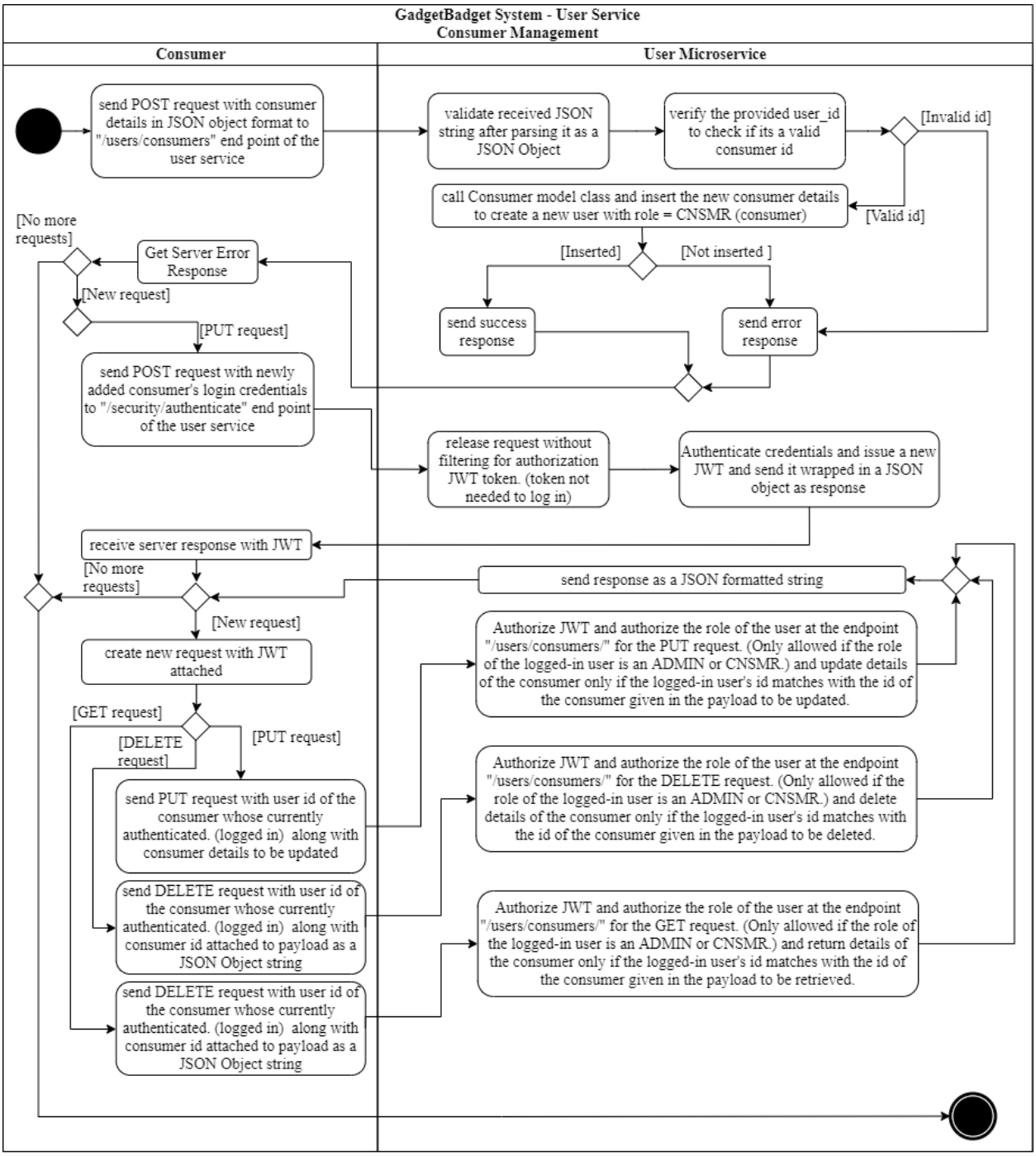
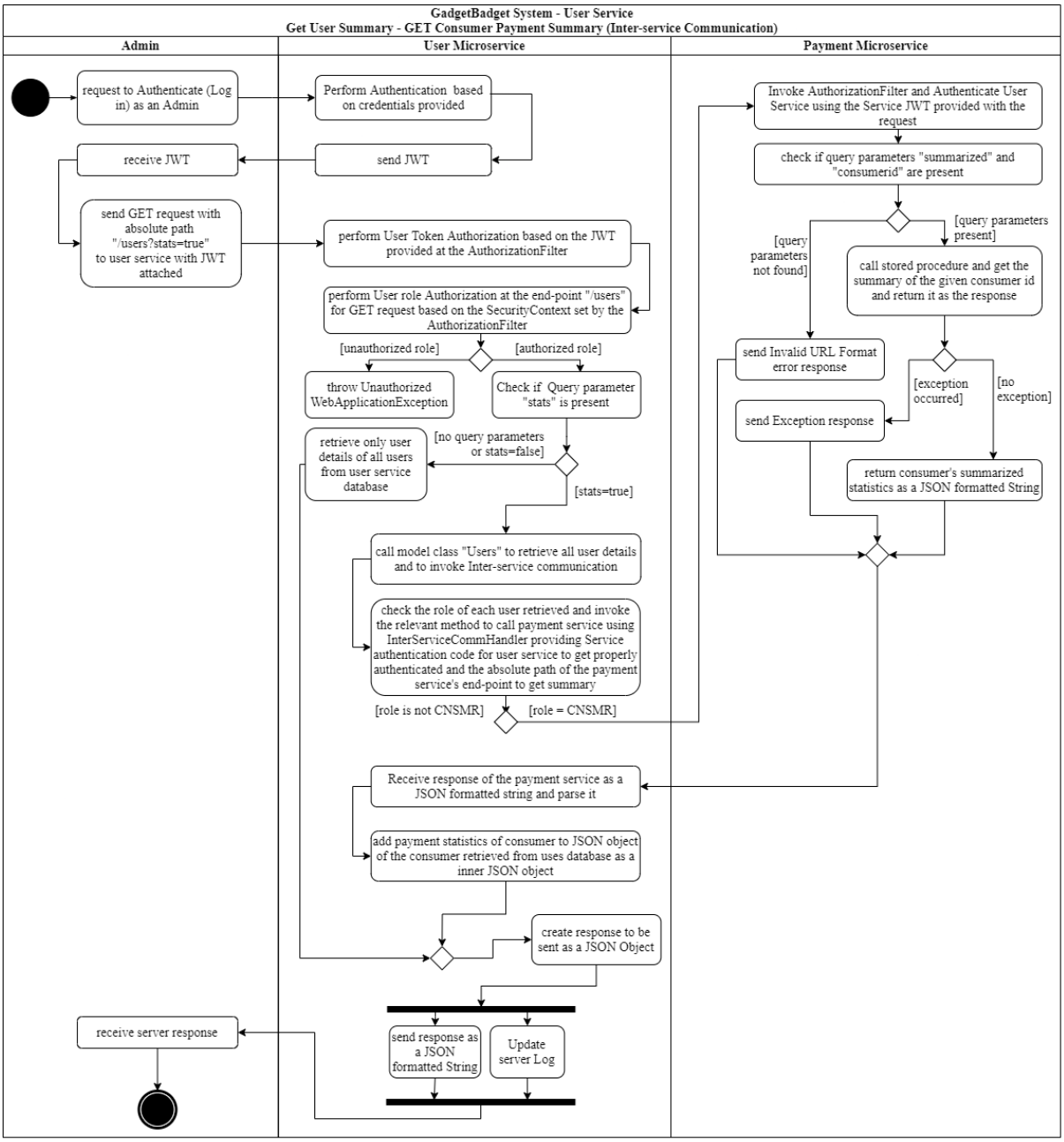


Figure A-2- 3 User Authorization Activity Diagram for user service



*** In the same manner, employees, funders, researchers also can Manage their accounts by following the same steps.**

Figure A-2- 4 Consumer Management - Activity diagram for all HTTP requests. Note that all employee, funder and researcher management are done in the same manner



*** In the same manner, user service performs inter-service communication with other services than payment service, namely, funding service (to access funders' statistics), marketplace service, and research hub service (to obtain researchers' statistics) as well. Employee statistics are not included in the user statistics list.**

Figure A-2- 5 Inter service communication request sent by the user service - Activity Diagram. Note that this is the same exact way used to request summarized details from other services for researcher products and research projects, and funds placed by funders.

A-3. Payment Service Diagrams

Database Design – EER of the Payment Service

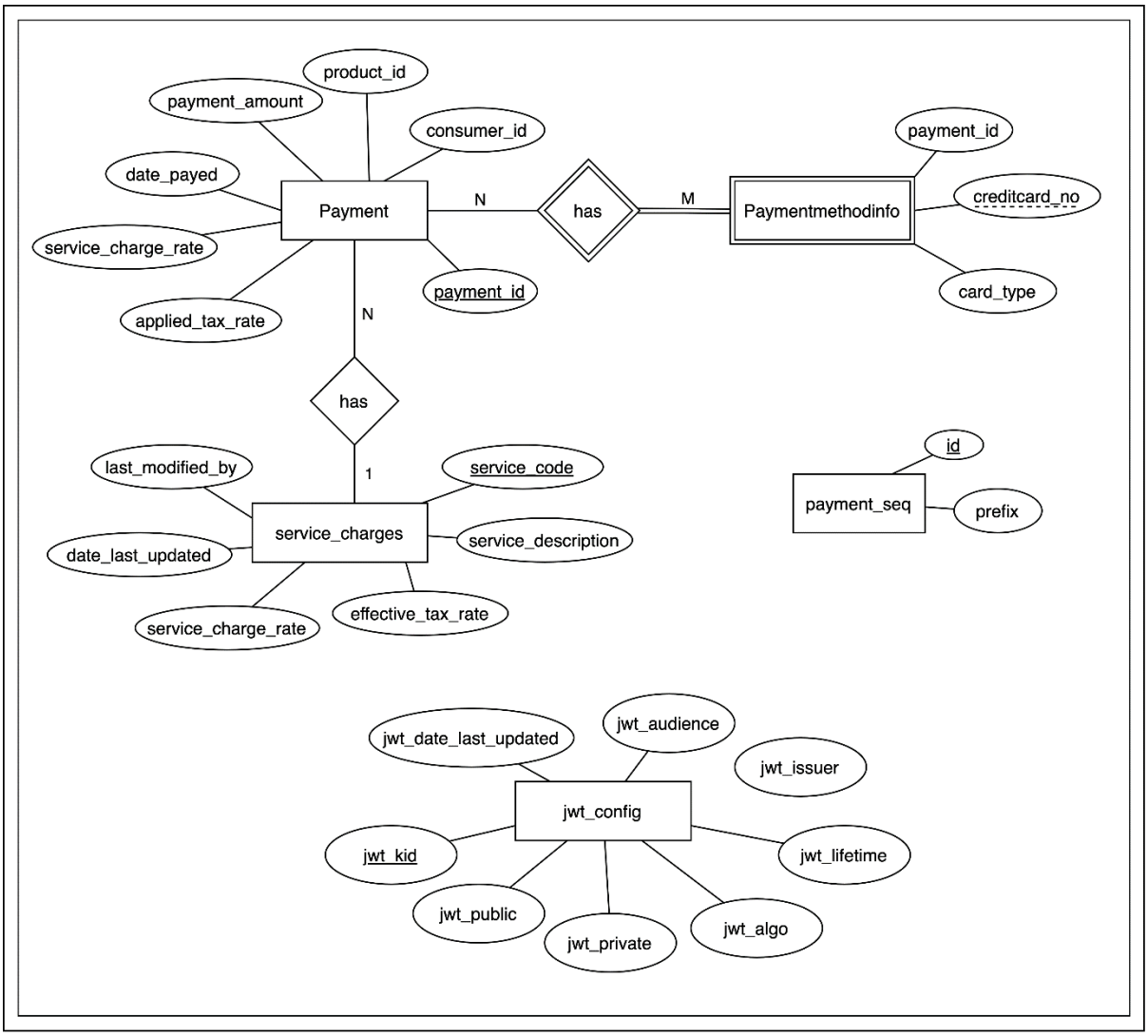


Figure A-3- 1 Database design of the Payment Service

Activity Diagrams of the Payment Service

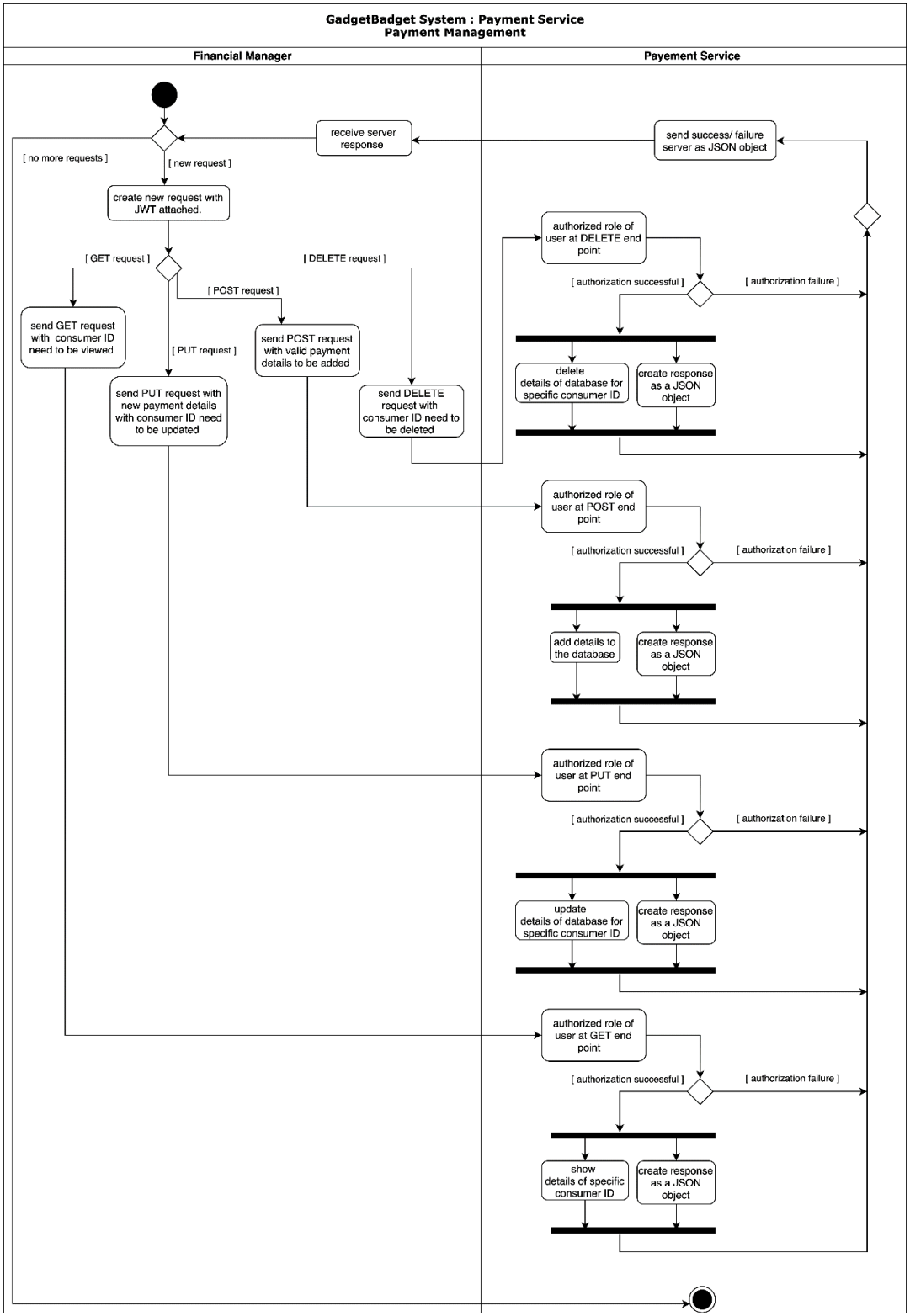


Figure A-3- 2 Activity Diagram for Payment Management HTTP Requests for CRUD operations

A-4. Research Hub Service Diagrams

Database Design – EER of the Research Hub Service

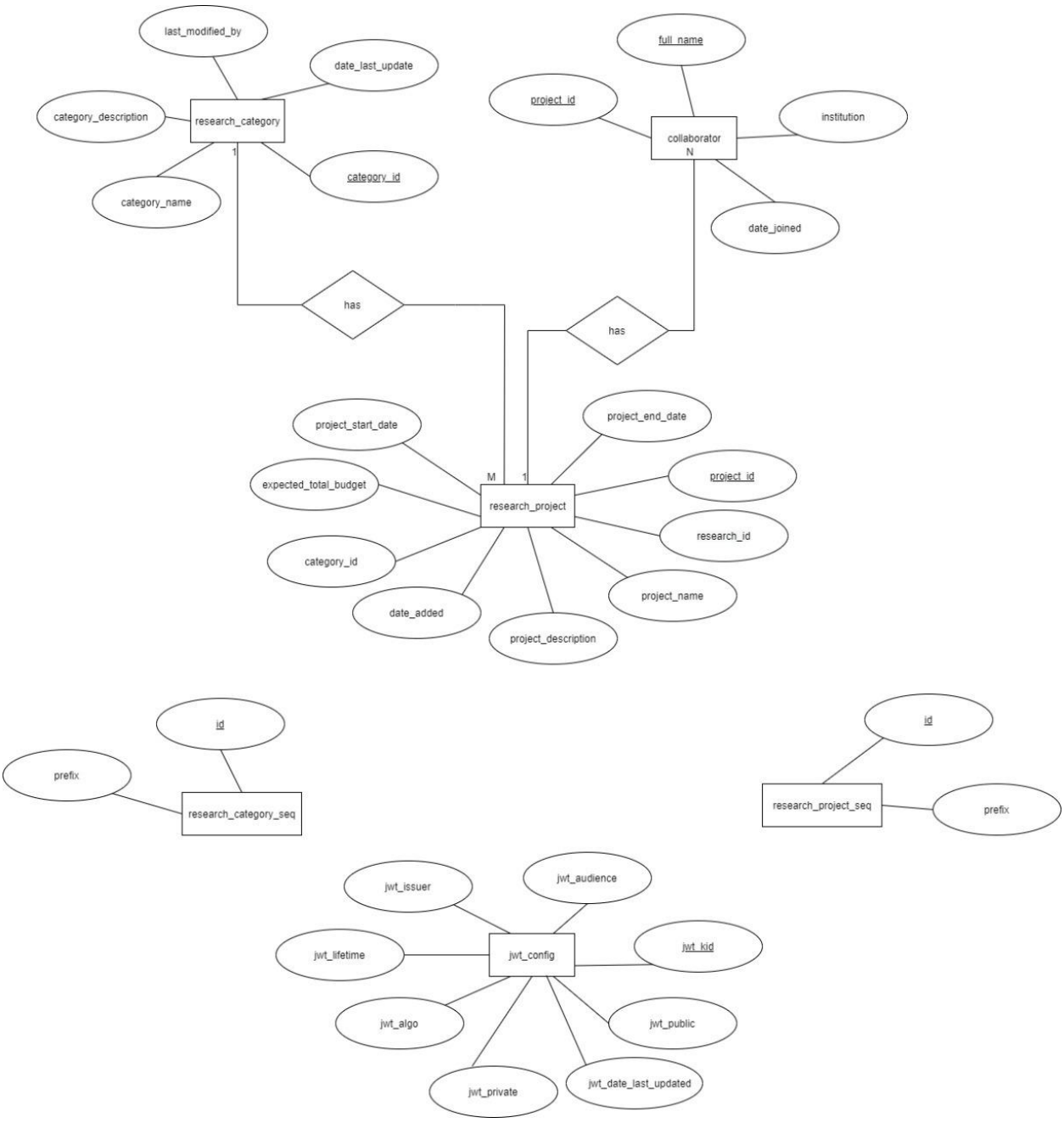


Figure A-4- 1 Database design of the ResearchHub Service

Activity Diagrams of the Research Hub Service

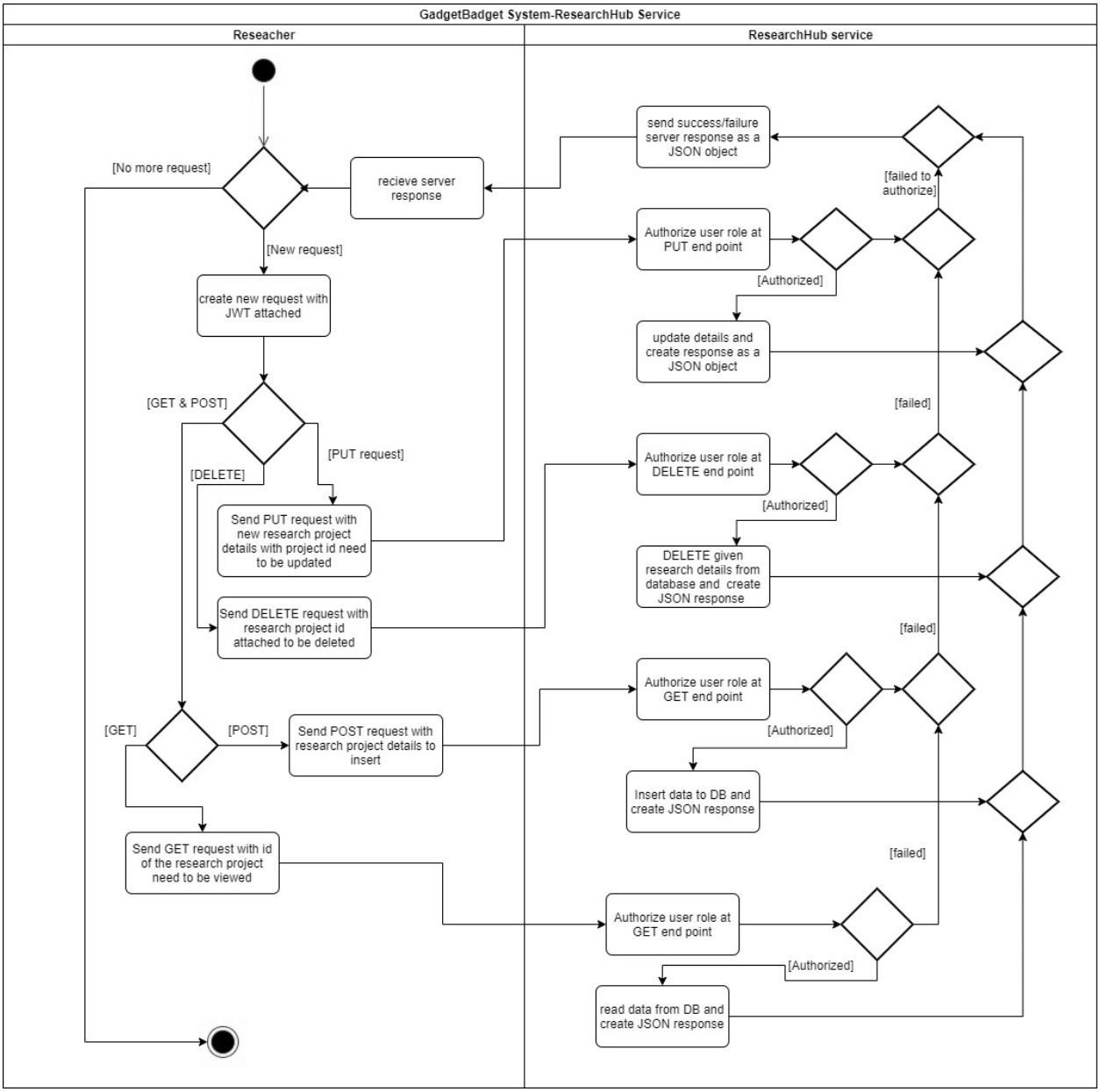


Figure A-4- 2 Activity Diagram for the Research Hub Service

A-5. Marketplace Service Diagrams

Database Design – EER of the Marketplace Service

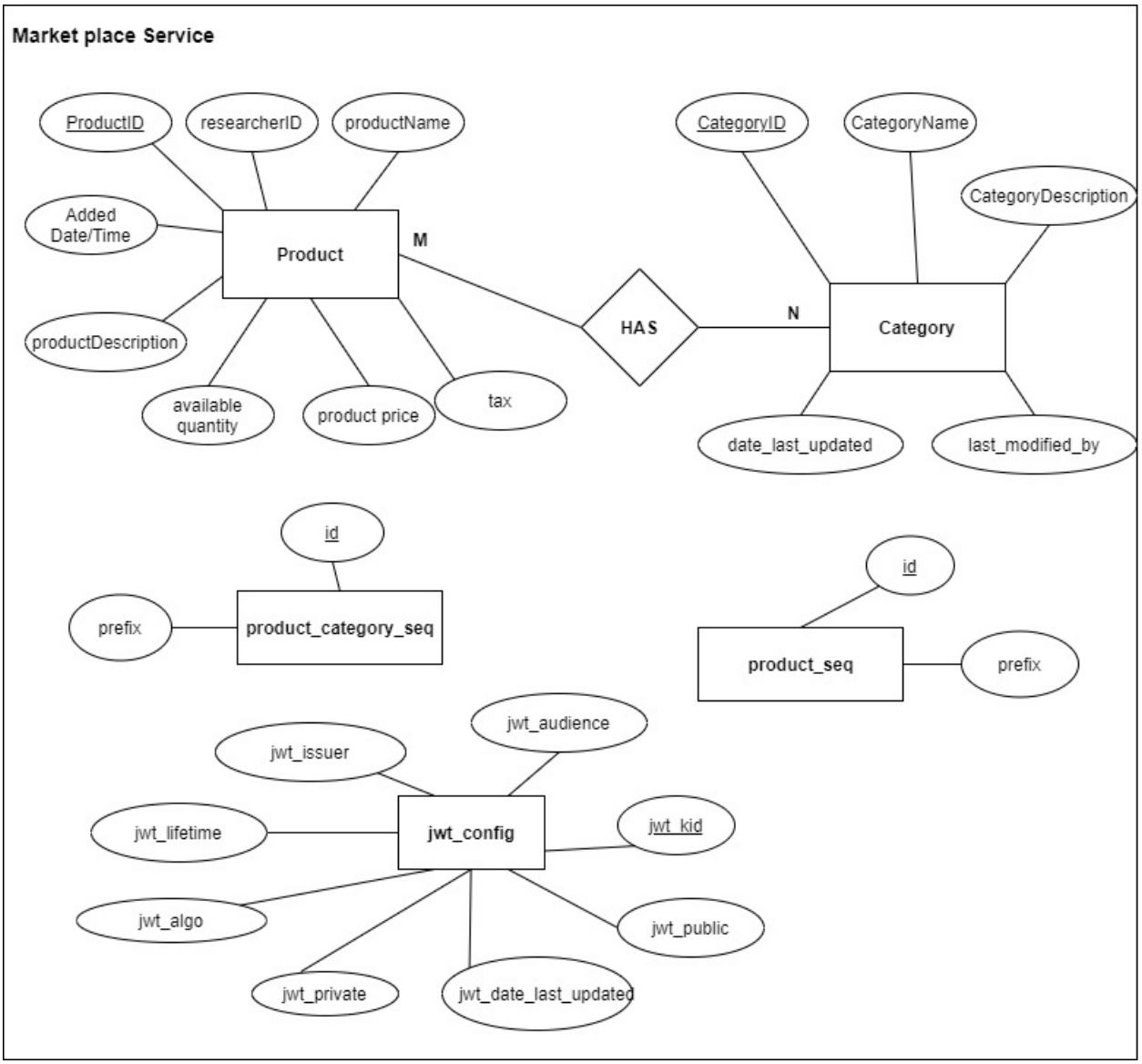


Figure A-5- 1 Database design of the Marketplace Service

Activity Diagrams of the Marketplace Service

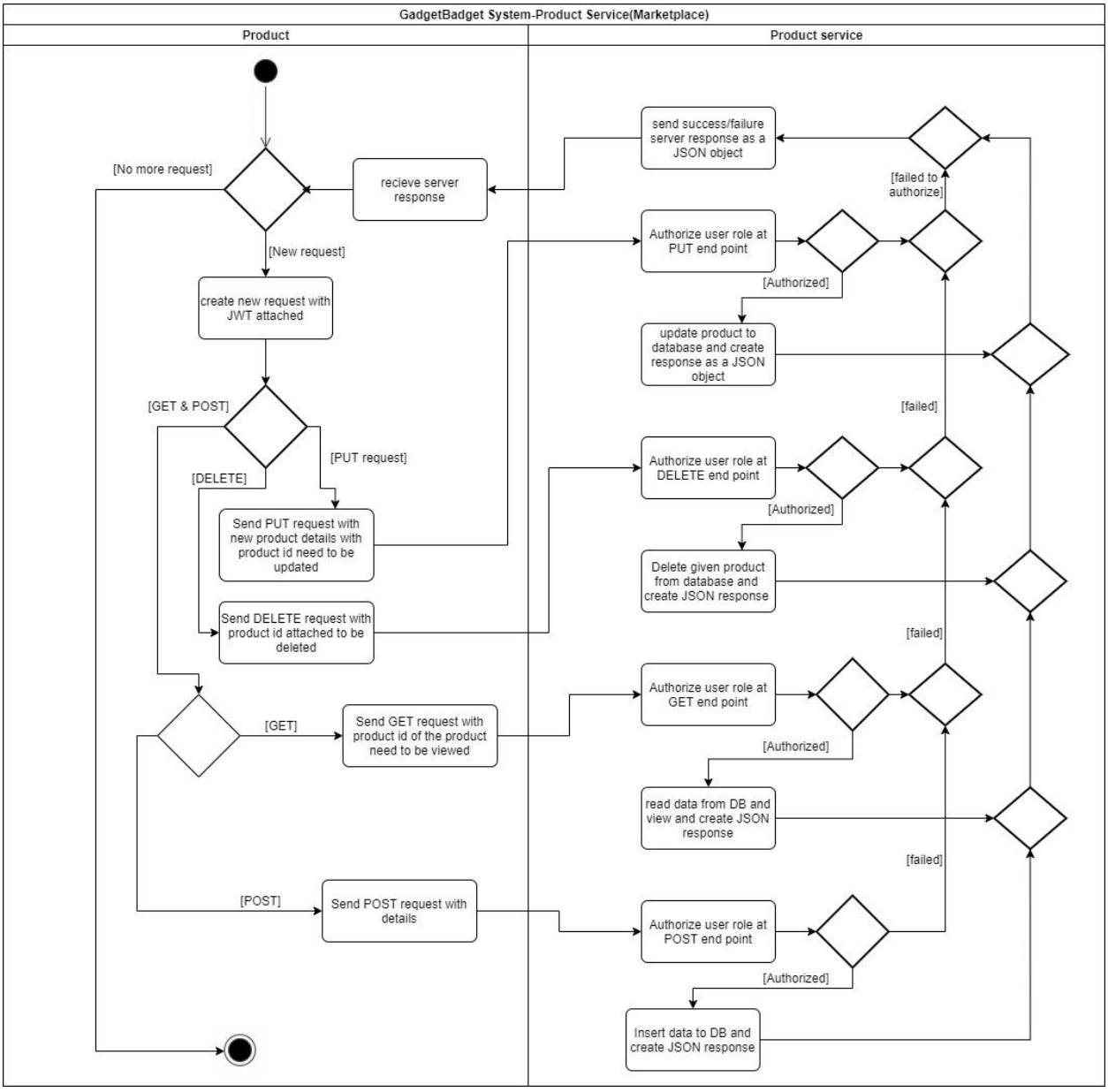


Figure A-5- 2 Activity Diagram for Marketplace Service HTTP requests

A-6. Funding Service Diagrams

Database Design of the Funding Service

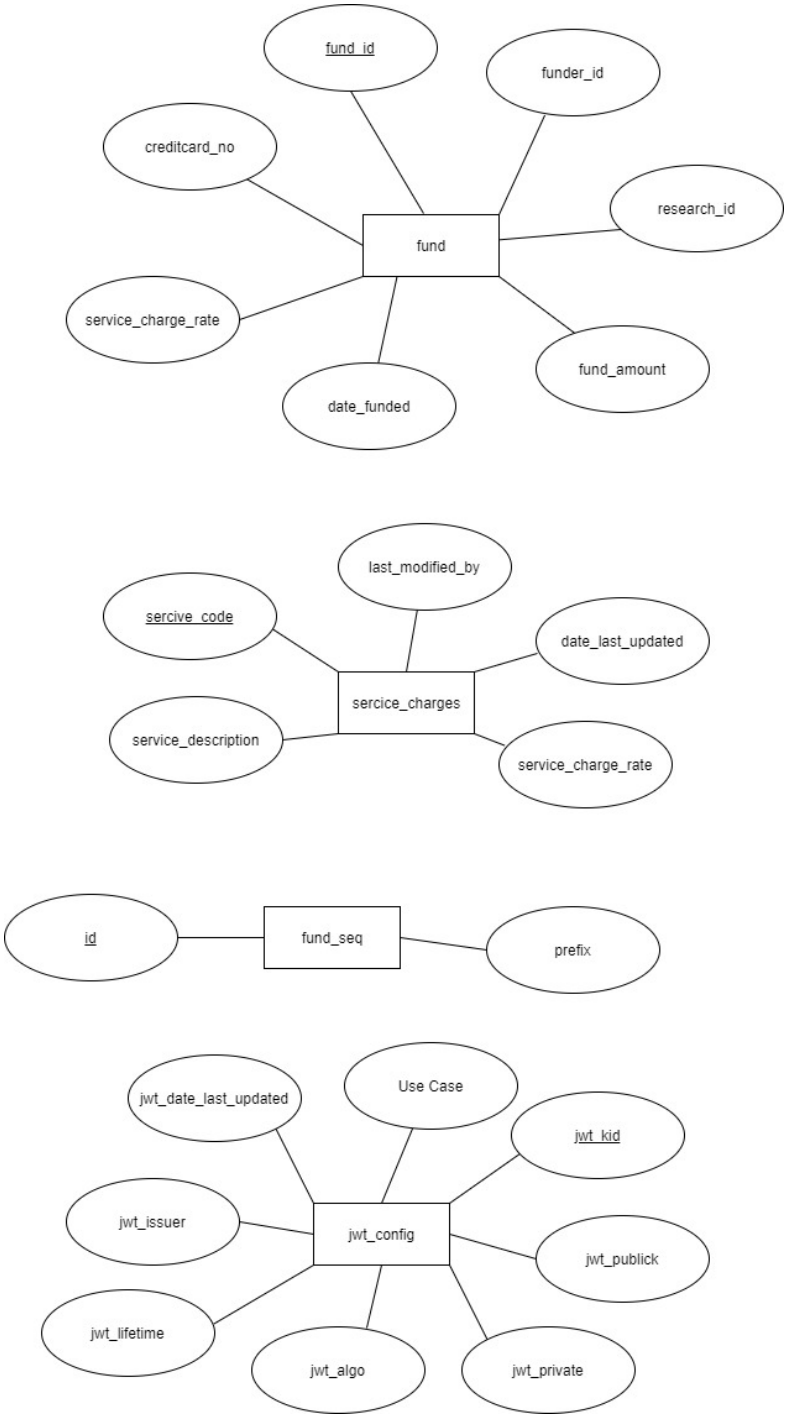


Figure A-6- 1 Database design of the Funding Service

Activity Diagrams

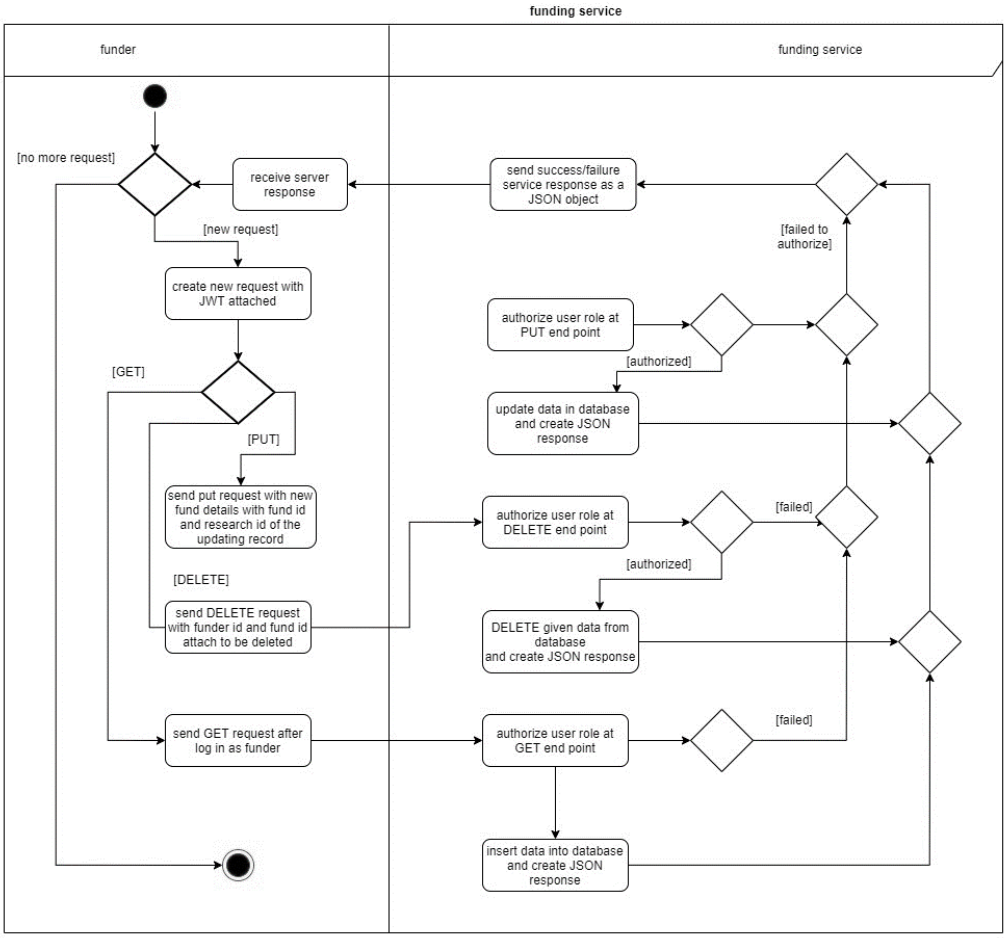


Figure A-6- 2 Activity Diagram for HTTP request received by the Funding Service

Appendix B: Selected Code Listings

B-1. Inter-Service Communication

```
private static final String PROTOCOL = "http://";
private static final String HOST = "127.0.0.1";
private static final String PORT = "8081";
private static final String SERVICE_URI = PROTOCOL + HOST + ":" + PORT + "/{ServiceProjectName
}/{servicename as in the web.xml mapping}/";
    //JWT Service Token
    private static final String SERVICE_TOKEN_USR= "SVC {Service JWT Token here}";
    private Client client = null;
    private WebResource webRes = null;
    /**
     * This method is used to establish service-to-service communication with a
     * specific service with the URL given in SERVICE_URL. Currently configured
     * only to process HTTP GET requests but the method body can be extended to support POST,
     * PUT, and DELETE HTTP methods
     * as well since the method header accepts the type of the HTTP method along with the
     * PAYLOAD in the type of a JSON Object.
     *
     * @param absolutePath  absolute path of the service endpoint is given here.
     * @param httpMethod    specific HTTP Method type of the endpoint specified.
     * @param payload       PAYLOAD to be attached to the request as a JSON Object.
     * @return              returns the original response of the funding service as a JSON obj
     */
    public JsonObject serviceIntercomms(String absolutePath, HttpMethod httpMethod, JsonObject
 payload)
    {
        client = Client.create();
        webRes = client.resource(PAYMENT_SERVICE_URI+absolutePath);

        if (httpMethod == HttpMethod.GET) {
            String output = webRes.header("Authorization", SERVICE_TOKEN_USR)
                    .get(String.class);

            return new JsonParser().parse(output).getAsJsonObject();
        }
        return null;
    }
```

The above method takes an absolute path, payload, and the HTTP method, combines the absolute path with the SERVICE_URI static string and makes a client call with the payload if any. Only the GET method is demonstrated here, thus the payload is not attached. Refer the source code for other related service calls in InterCommHandler.java classes of any of the services.

B-2. JWT Token Generation and Verification

```
private static String JWT_SUBJECT = "gadgetbadget.auth."; //subject prefix of the JWT

/**
 * This method generates expiring enabled JWTs for user authentication purposes.
 *
 * @param username          user-name of the logged in(validated) user
 * @param user_id           corresponding unique user_id
 * @param role              role id of the validated user
 * @return                  a valid JWT with expiration enabled for 30 minutes
 * @throws JoseException    Exception is thrown by issues that might occur while generating the JWT
 * @throws SQLException     Exception is thrown by issues that might occur while accessing the db
 */
public String generateToken(String username, String user_id, String role) throws JoseException, SQLExcepti
on {
    String jwt = null;

    JWT_SUBJECT += username;            // set a unique subject based on user-name per user
    JWK jwk = getJWKFromDB("JWK1");     // get the JSON Web Key attribute values stored in the db

    // Generate the JWT Header Claims
    JwtClaims claims = new JwtClaims();
    claims.setIssuer(jwk.getIssuer());
    claims.setAudience(jwk.getAudience());
    claims.setExpirationTimeMinutesInTheFuture(jwk.getLifetime());
    claims.setGeneratedJwtId();
    claims.setIssuedAtToNow();
    claims.setNotBeforeMinutesInThePast(2);
    claims.setSubject(JWT_SUBJECT);

    // Generate the JWT PAYLOAD
    claims.setClaim("username",username);
    claims.setClaim("user_id",user_id);
    claims.setClaim("role",role);

    // Create the JsonWebSignature based on Claims and PAYLOAD
    JsonWebSignature jws = new JsonWebSignature();

    // The PAYLOAD of the JWS is JSON content of the JWT Claims
    jws.setPayload(claims.toJson());
    jws.setKey(jwk.getPrivate_key());
    jws.setKeyIdHeaderValue(jwk.getKey_id());
    jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);

    // Sign the JWS and produce the compact serialization or the complete JWT/JWS
```

```
    // representation, which is a string consisting of three dot ('.') separated
    // base64url-encoded parts in the form Header.Payload.Signature
    jwt = jws.getCompactSerialization();

    return jwt;
}

/**
 * This method validates JWT Token for User Authentication
 *
 * @param jwt                       JWT in the form of a String
 * @return                          Validity of the token as a boolean value
 * @throws MalformedClaimException  Exception is thrown when there are issues with validation claims
 * @throws JoseException            Exception is thrown when there are JWT encoding issues
 */
public boolean validateToken(String jwt) throws MalformedClaimException, JoseException {
    try
    {
        // Retrieve RSA and JW Key Attribute Values from local storage
        JWK jwk = getJWKFromDB("JWK1");

        // Set JWT Claims for Validation
        JwtConsumer jwtConsumer = new JwtConsumerBuilder()
             .setRequireExpirationTime()
             .setAllowedClockSkewInSeconds(jwk.getLifetime())
             .setRequireSubject()
             .setExpectedIssuer(jwk.getIssuer())
             .setExpectedAudience(jwk.getAudience())
             .setVerificationKey(jwk.getPublic_key())
             .setJwsAlgorithmConstraints(ConstraintType.PERMIT, AlgorithmIdentifiers.RSA_USING_SHA256)
             .build();

        // Validate the JWT and process it to the Claims
        JwtClaims jwtClaims = jwtConsumer.processToClaims(jwt);
        System.out.println("JWT validated. JWT Claims: " + jwtClaims);
        return true;
    }
    catch (InvalidJwtException ex)
    {
        System.out.println("Invalid JWT! " + ex);
        if (ex.hasExpired())
        {
            System.out.println("JWT expired at " + ex.getJwtContext().getJwtClaims().getExpirationTime());
        }
```

```
        if (ex.hasErrorCode(ErrorCodes.AUDIENCE_INVALID))
        {
         System.out.println("JWT had wrong audience: " + ex.getJwtContext().getJwtClaims().getAudience());
        }
        return false;
    }
    catch (Exception ex) {
        System.out.println("Failed to validate the given JWT. Exception Details: " + ex.getMessage());
        return false;
    }
}
```

Above two methods are used to create and verify JWT tokens. Generate method is only used in the user service while all other services are utilizing the verification method stated above to verify user and service JWTs.

B-3. RSA Key Pair Generation

```
/**
* Generates a RSA Public Key Private Key pair and Stores it in the local database under the given key id
* @param key_id    a unique identifier given for each RSA key pair generated
*/
public void generateRSAKeyPair(String key_id) {
   RsaJsonWebKey rsaJsonWebKey;
   try {
       rsaJsonWebKey = RsaJwkGenerator.generateJwk(2048);
       rsaJsonWebKey.setKeyId(key_id);

       System.out.println("RSA Keys were Generated and Saved: " + (insertJWKToDB(rsaJsonWebKey, key_id)?"S
uccessfully.":"Failed."));
   } catch (Exception e) {
       e.printStackTrace();
   }
}
```

Above code segment is used to generate a unique public and private key pair to sign JWTs. Two key pairs were generated initially to uniquely identify users and services separately. New key pairs can be generated to replace the old keys used by the services in case of a major security flaw.

Appendix C: Testing and Results

C-1. POSTMAN Test results of User Service

Users end points

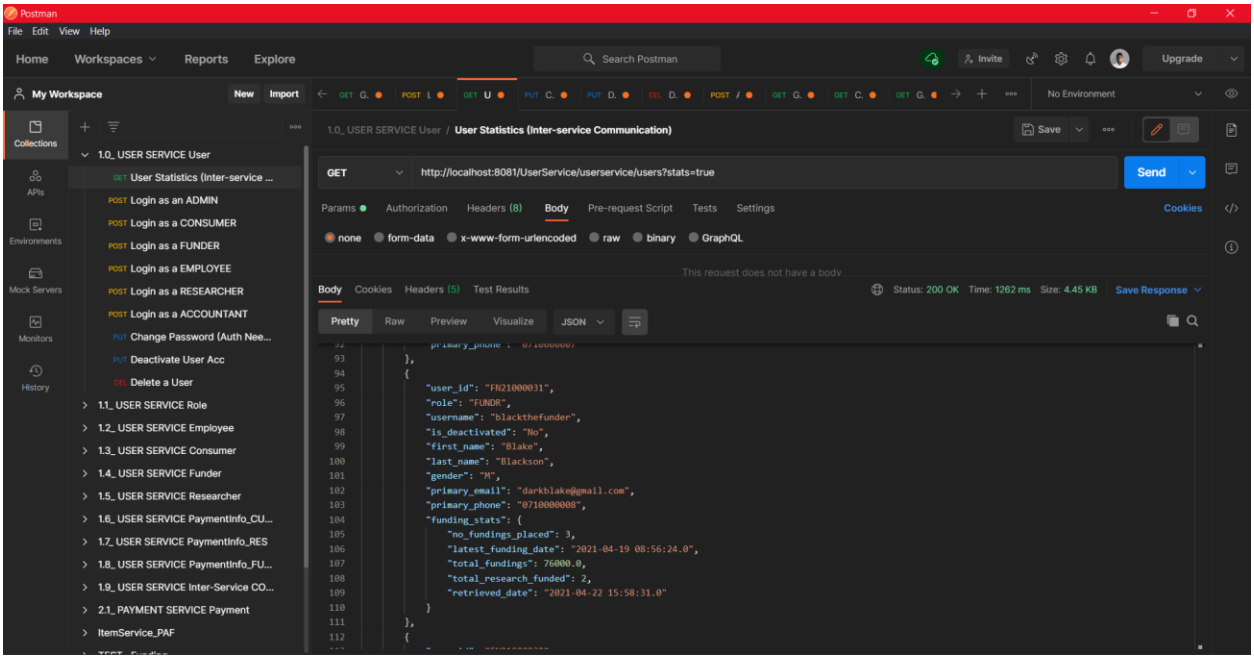


Figure C-1- 1 User service: getting summarized details of all users using inter-service communication

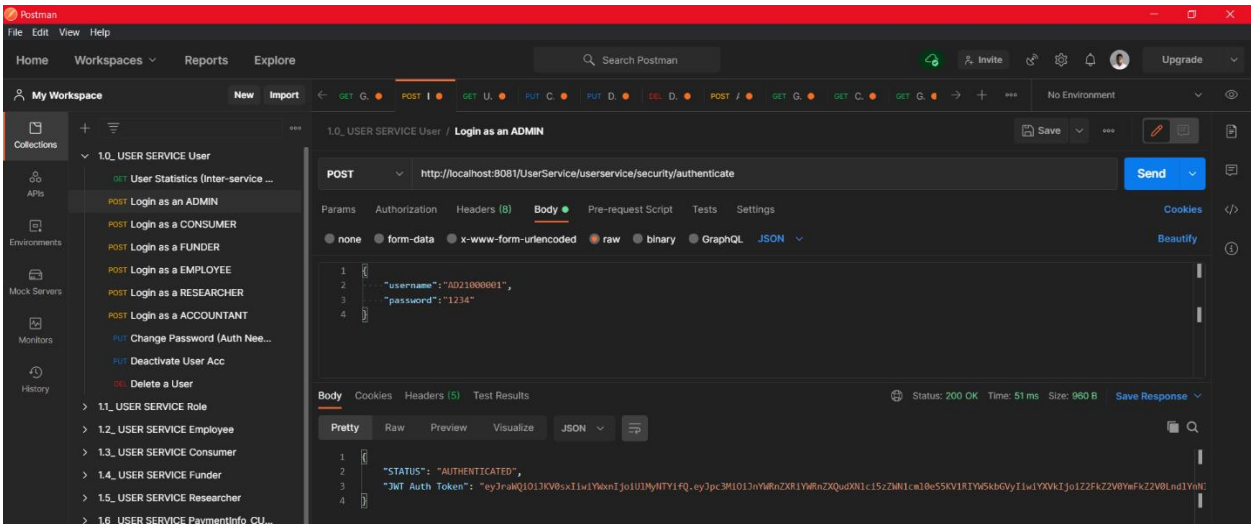


Figure C-1- 2 user service: authentication (login as an ADMIN). The response will contain the JWT token which needs to be included in every other request where authentication is needed.

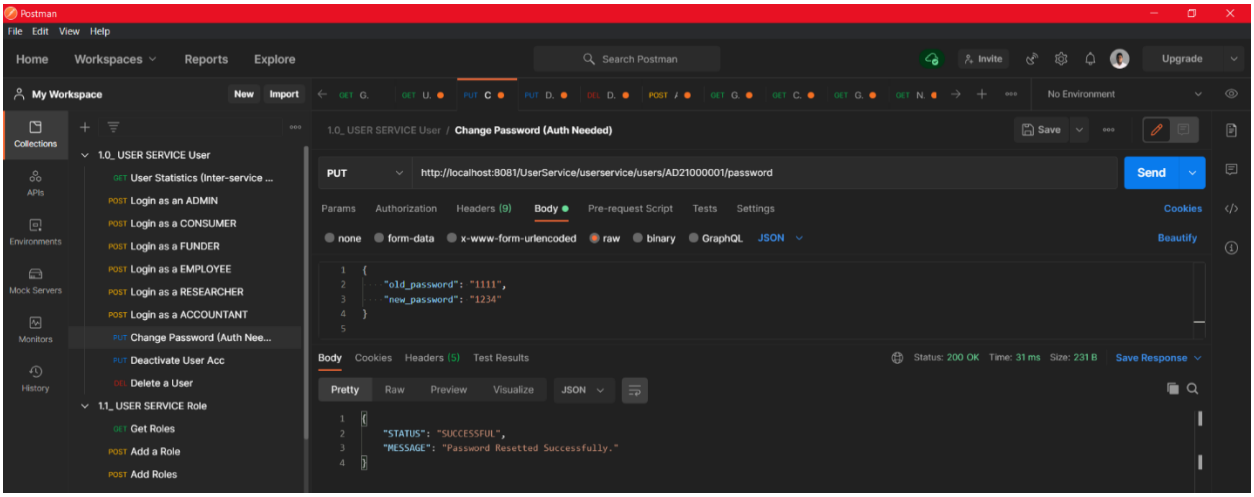


Figure C-1- 3 user service: resetting user password (any user can perform this task after login)

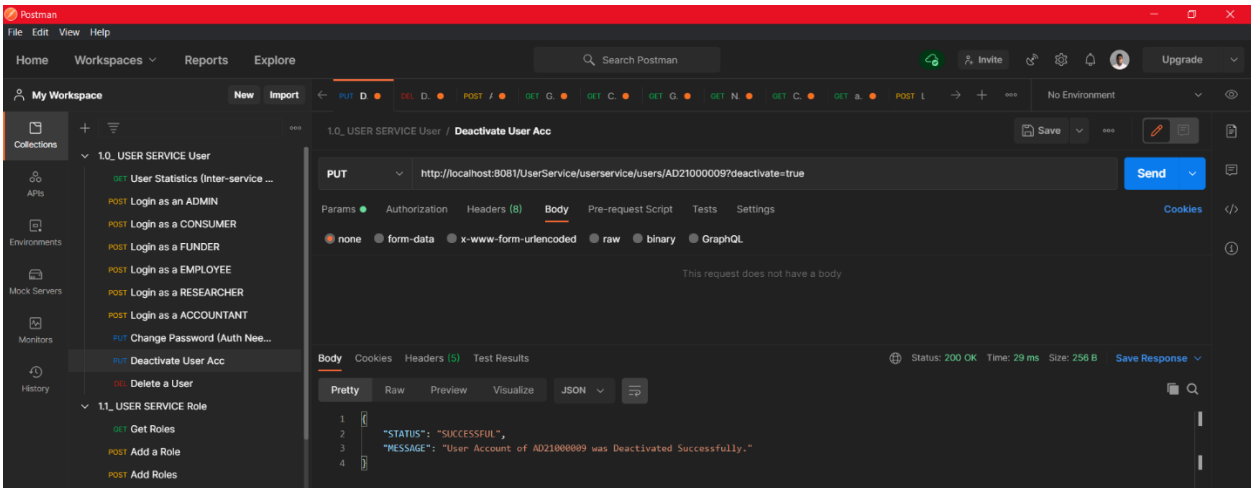


Figure C-1- 4 user service: deactivating a user account. (only administrators can do this)

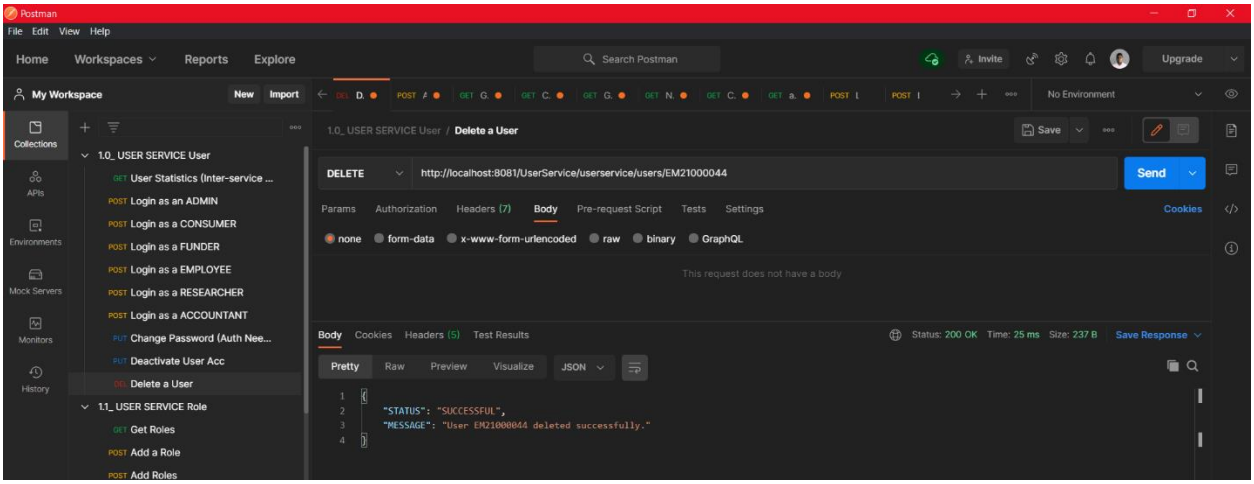


Figure C-1- 5 Deleting a specific user account (Only administrators are authorized)

Specific user type sub resources

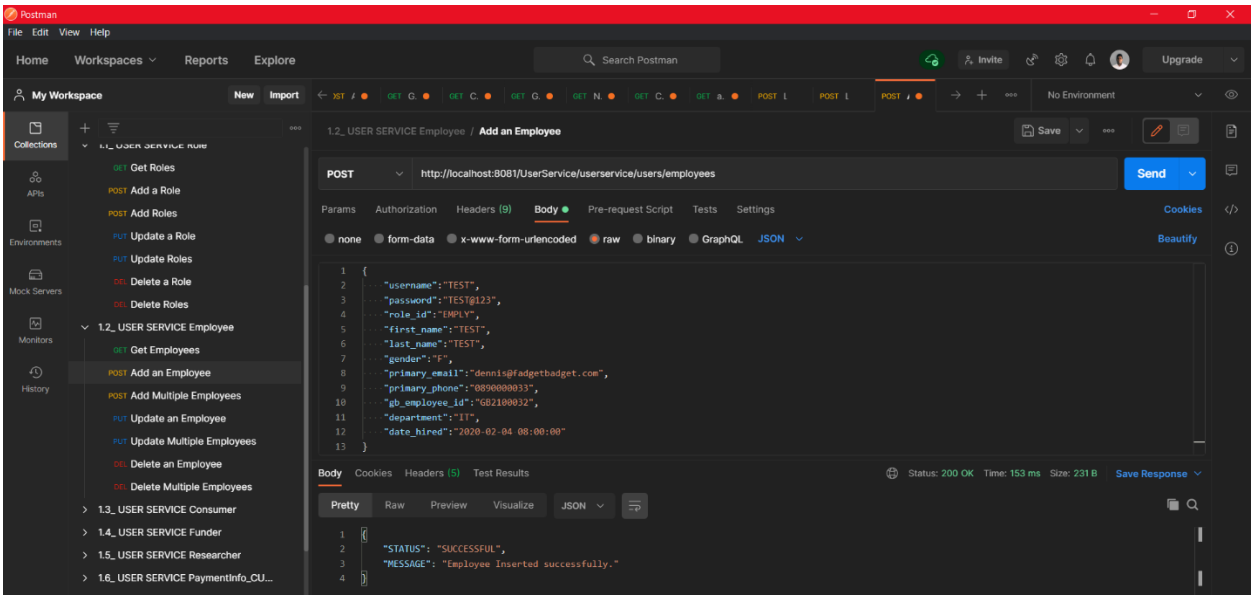


Figure C-1- 6 Insert a new Employee. Note that inserting other user types is done the same way by altering the URI appropriately. Creating accounts does not need authorization tokens unless it is an Admin who tries to insert multiple accounts at ones.

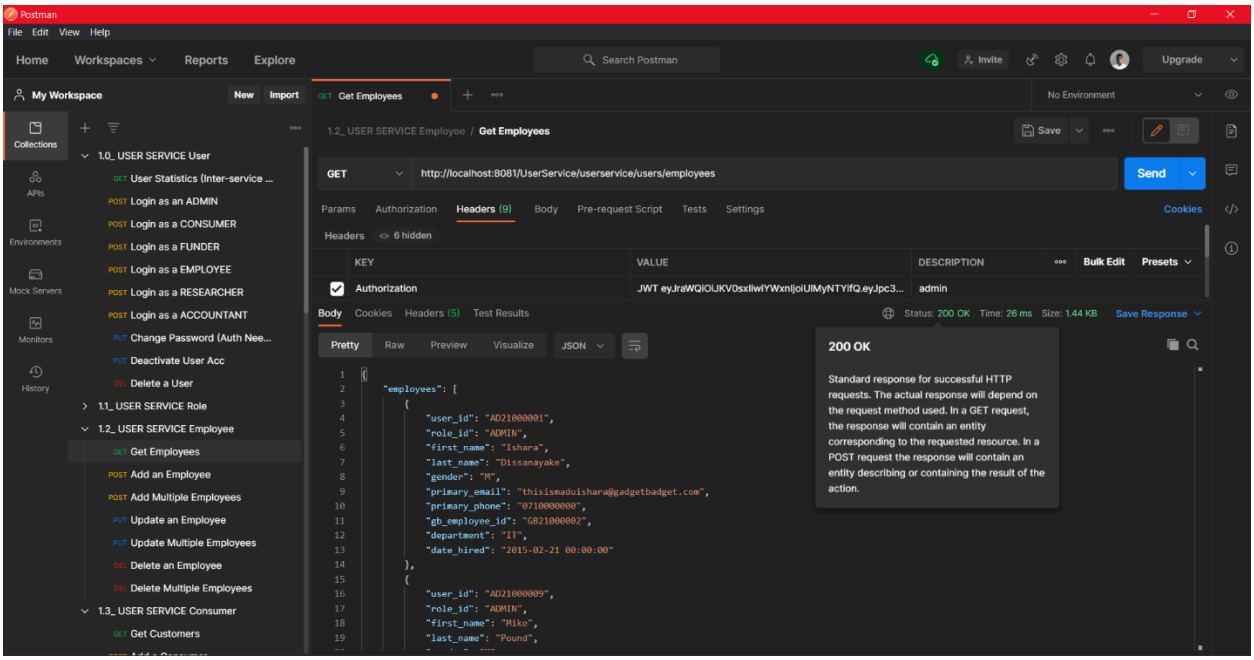


Figure C-1- 7 Retrieving list of all users of a specific type (only admins are allowed)

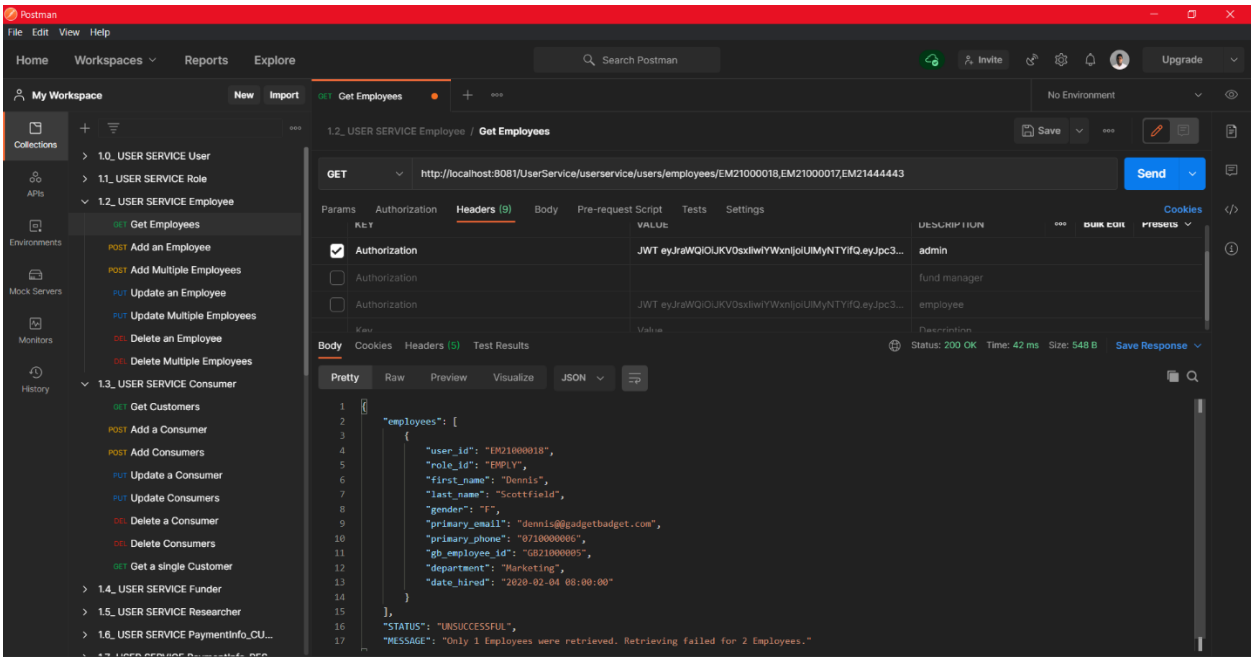


Figure C-1- 8 Administrators can retrieve single/multiple users of a specific user type by providing the comma separated user ids after the sub collection path. (Non-administrators are prohibited to do so)

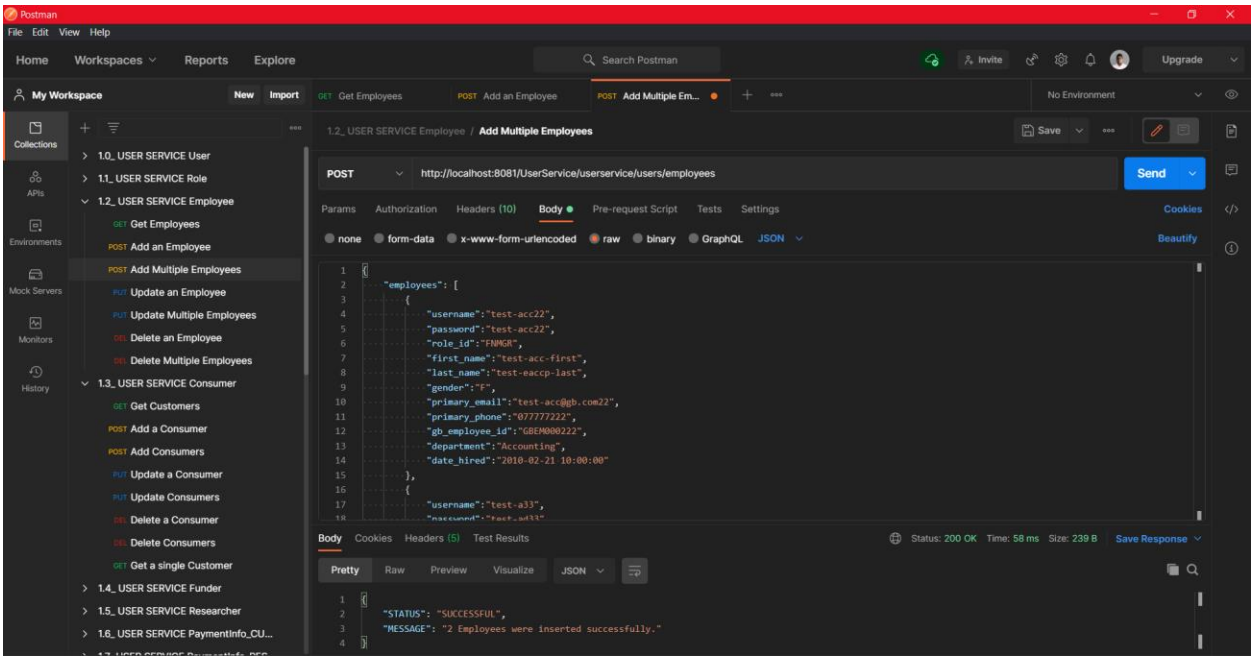


Figure C-1- 9 Administrators can add multiple users at the same time by providing user details in a JSON array within the wrapper JSON object. (Non-administrators cannot do this)

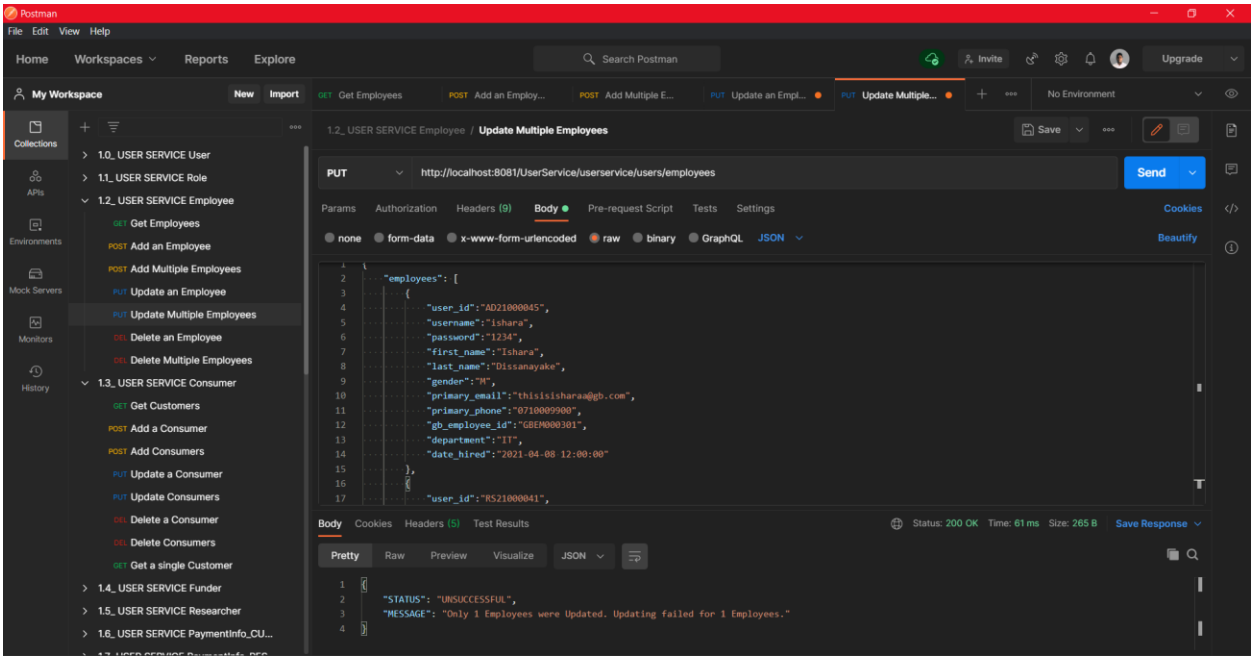


Figure C-1- 10 Updating single/multiple users of a specific user type (employee in this case) can be done by administrators to any account. Non-administrators cannot edit multiple accounts at once or update an account that does not belong to them. Entering a non-existing user id will cause user not found errors.

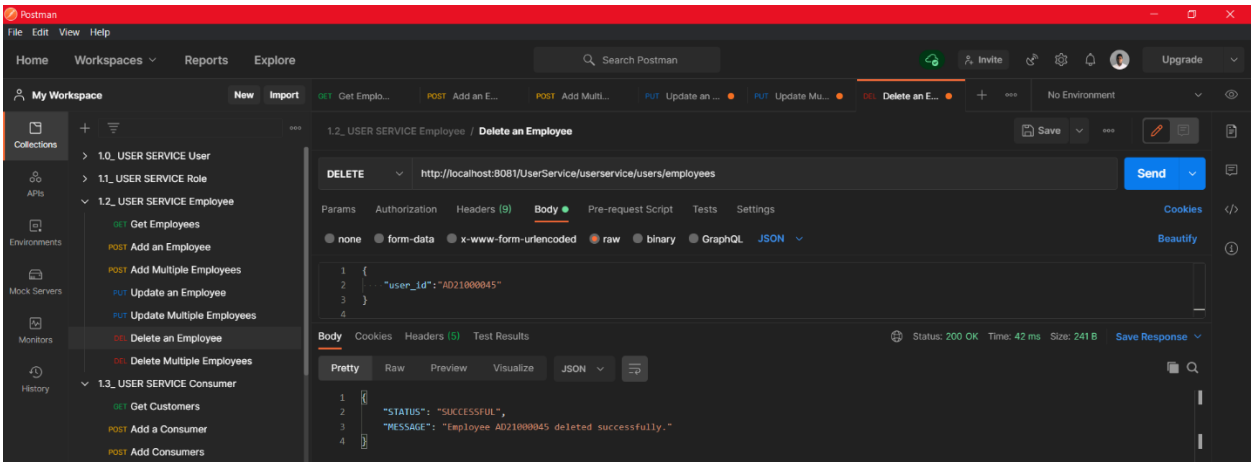


Figure C-1- 11 Deleting a user belongs to a specific user type can be done by the administrator. Non-administrators can only delete their own account.

Only administrators can delete multiple accounts at once. Providing a non-existing user id will cause User not found exceptions.

Payment Method Details of Users (consumers/funders/researchers) – Special Cases

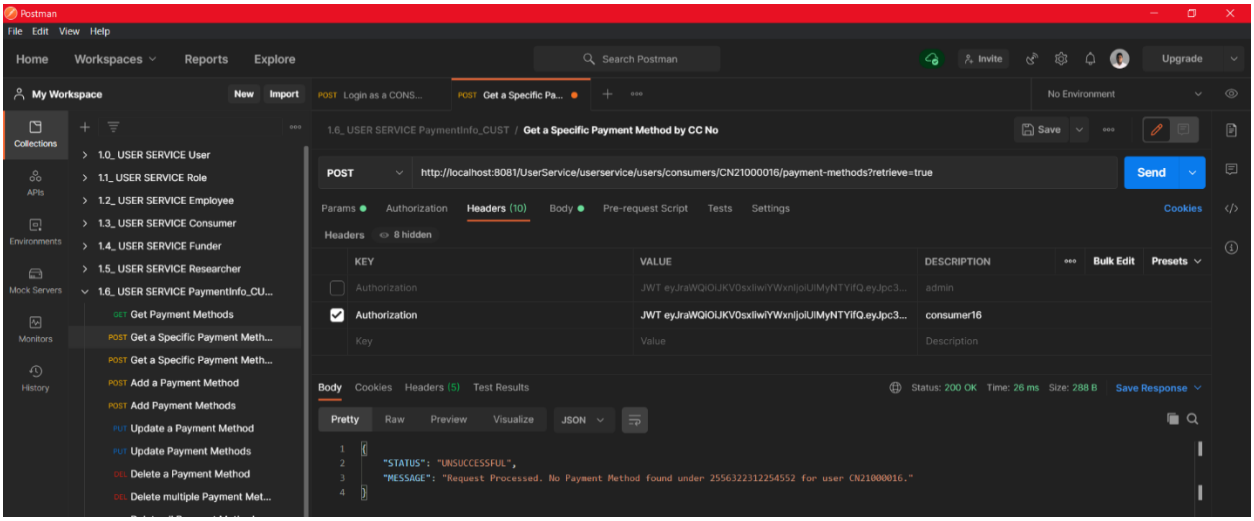


Figure C-1- 12 Users can get details of only one payment method by setting the query parameter "retrieve=true" and providing the credit card number in the payload. This is done as a POST request for security concerns. Trying to retrieve a non-existing credit card will get an UNSUCCESSFUL response.

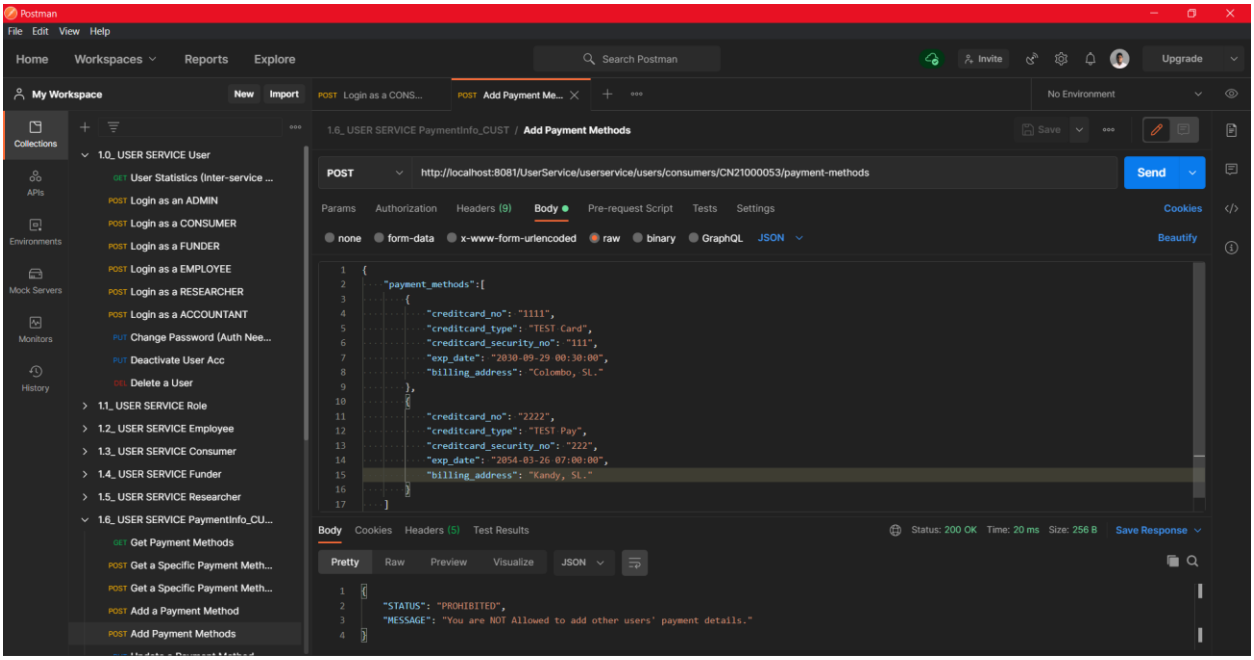


Figure C-1- 13 Multiple payment Insertion is also allowed, and non-administrators can only add their payment methods. any failures will be indicated in the result as a SUCEESSFUL or UNSUCCESSFUL status with the counts

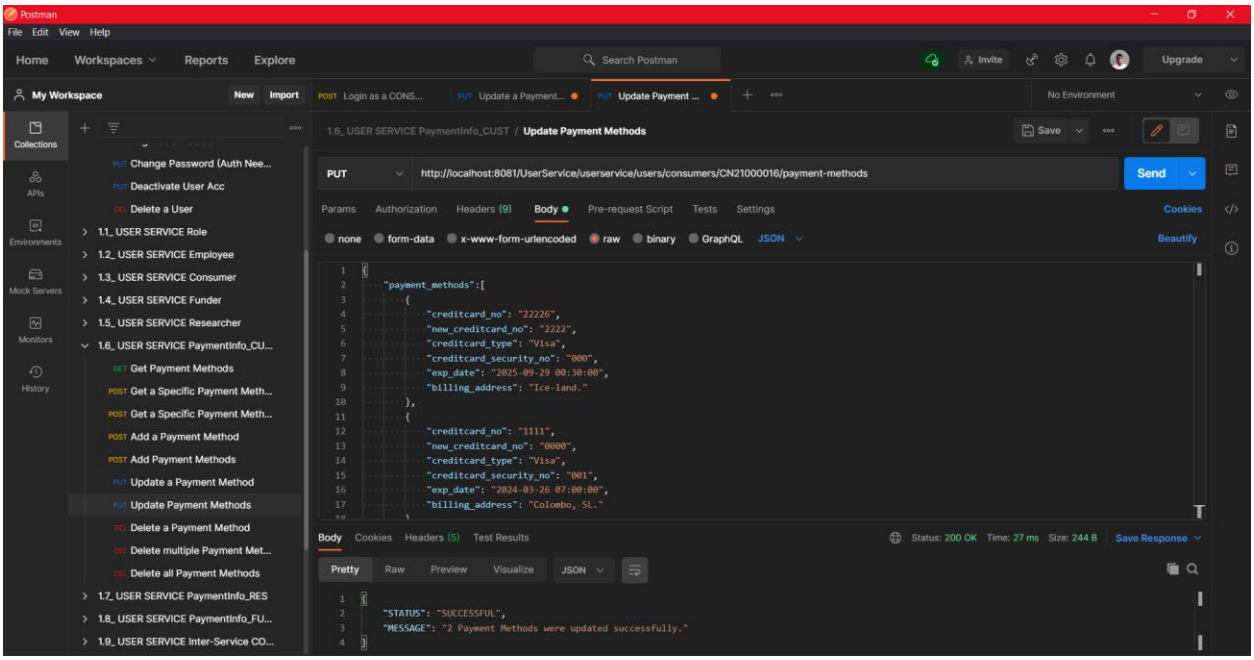


Figure C-1- 14 Multiple updates are allowed for any user while non-administrators should only update their own payment methods or else the request will get invalidated with a “PROHIBITED” response.

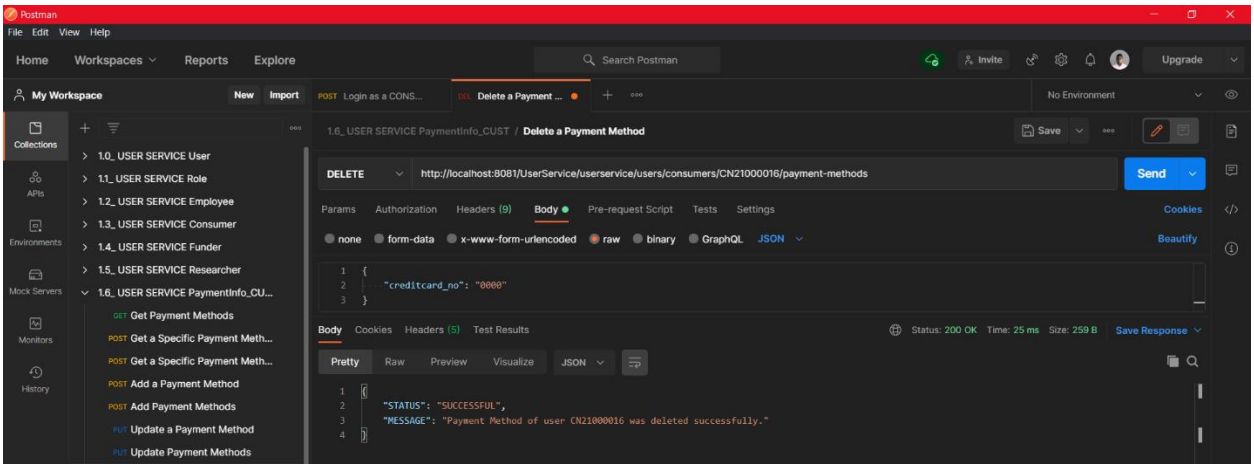


Figure C-1- 15 Deleting a single payment method

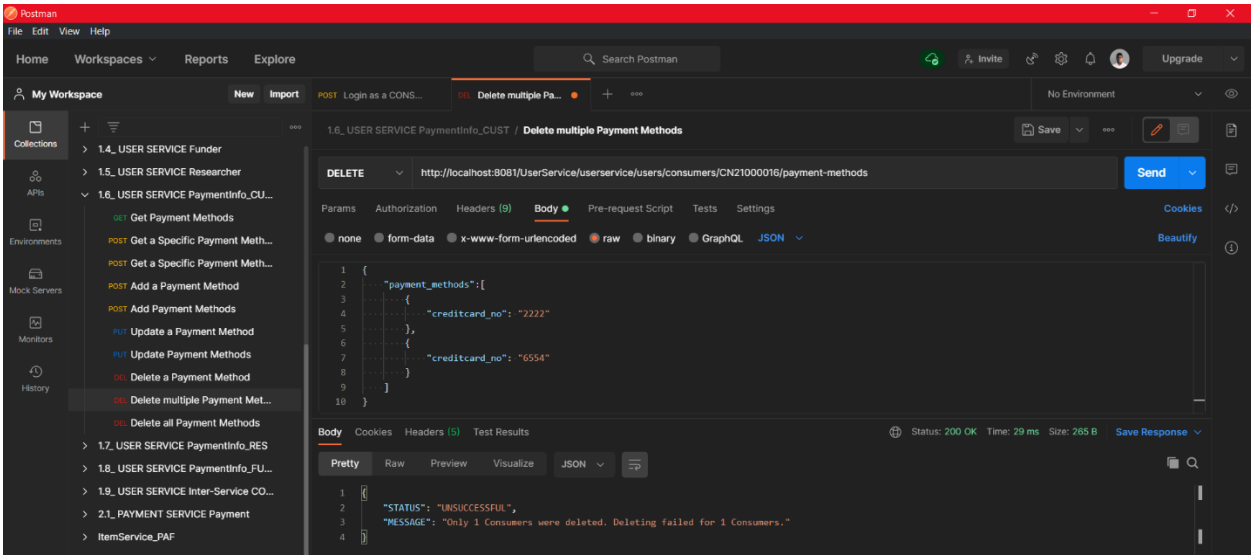


Figure C-1- 16 Deleting multiple payment methods

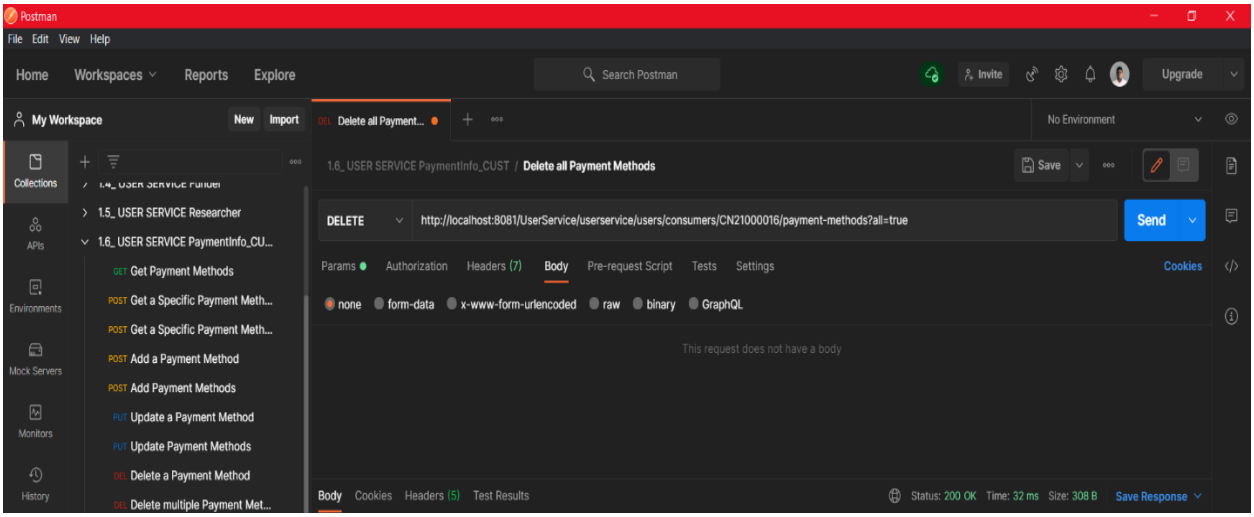


Figure C-1- 17 Deleting all Payment methods of a given user

Note that although Employees and Consumers are taken for postman demonstrations, these restrictions, features, and authorization mechanisms have been applied for all the users. Payment end points work the same way. Only special test cases are given. Performing CRUD operations on ROLES can be done in the same manner by sending HTTP request to http://localhost:8081/UserService/userservice/security/roles. Only Administrators can alter Role details.

C-2. POSTMAN Test results of Payment Service

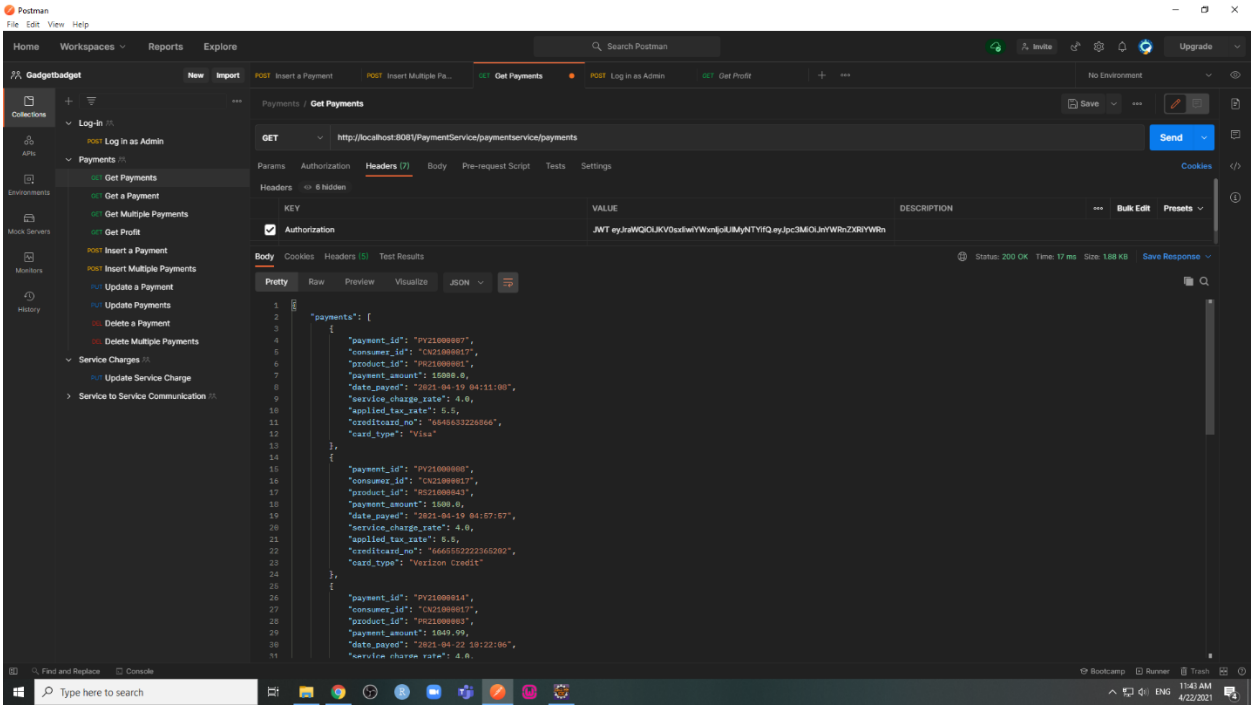


Figure C-2- 1 Get the list of all payments. (only administrators and financial managers can perform this task)

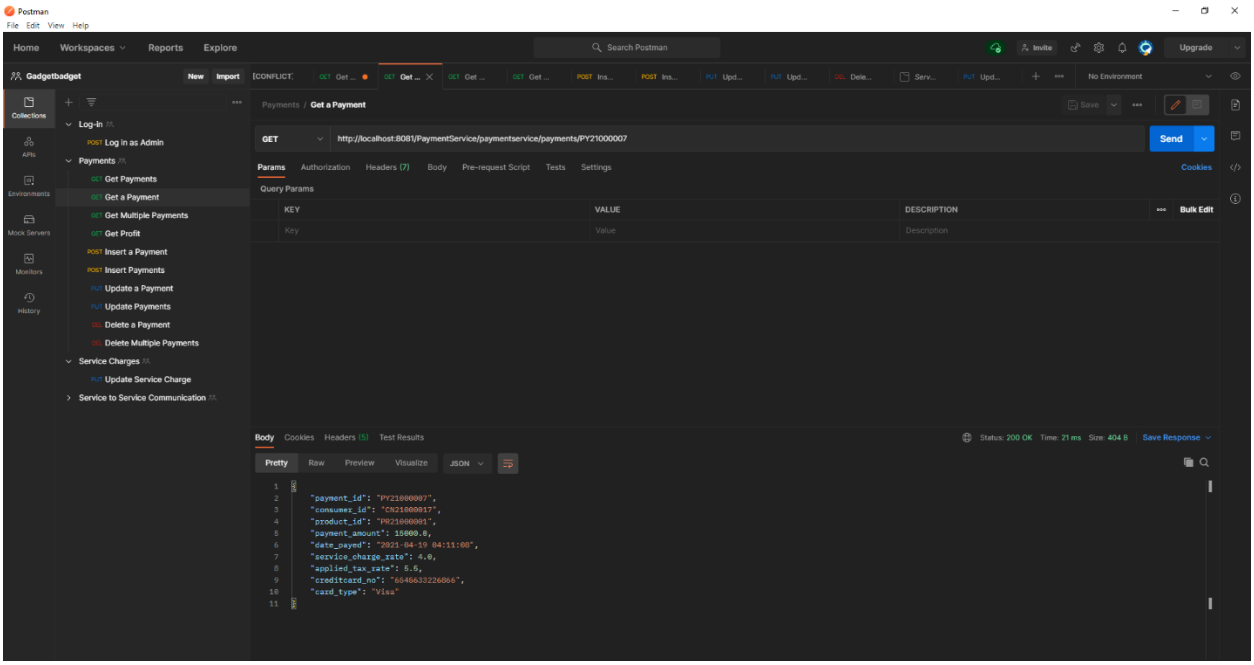


Figure C-2- 2 A specific payment can be retrieved by giving the payment id in the URL. (Administrators and financial managers can retrieve any payment done by any user while consumers are prohibited from viewing payments that do not belong to them)

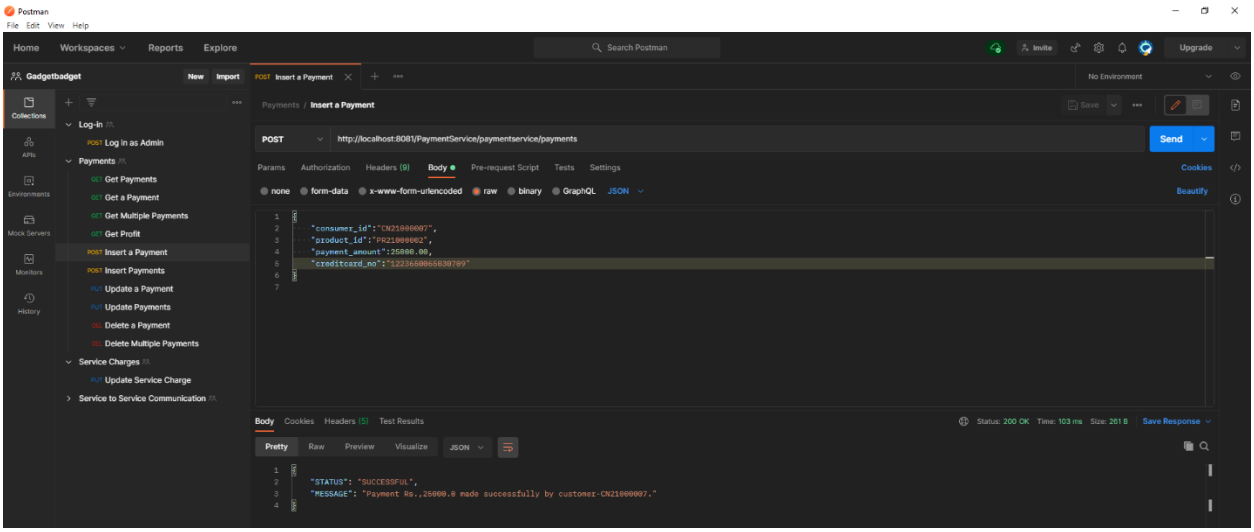


Figure C-2- 3 Inserting only one payment. only allowed users are admin, financial manager and consumer.

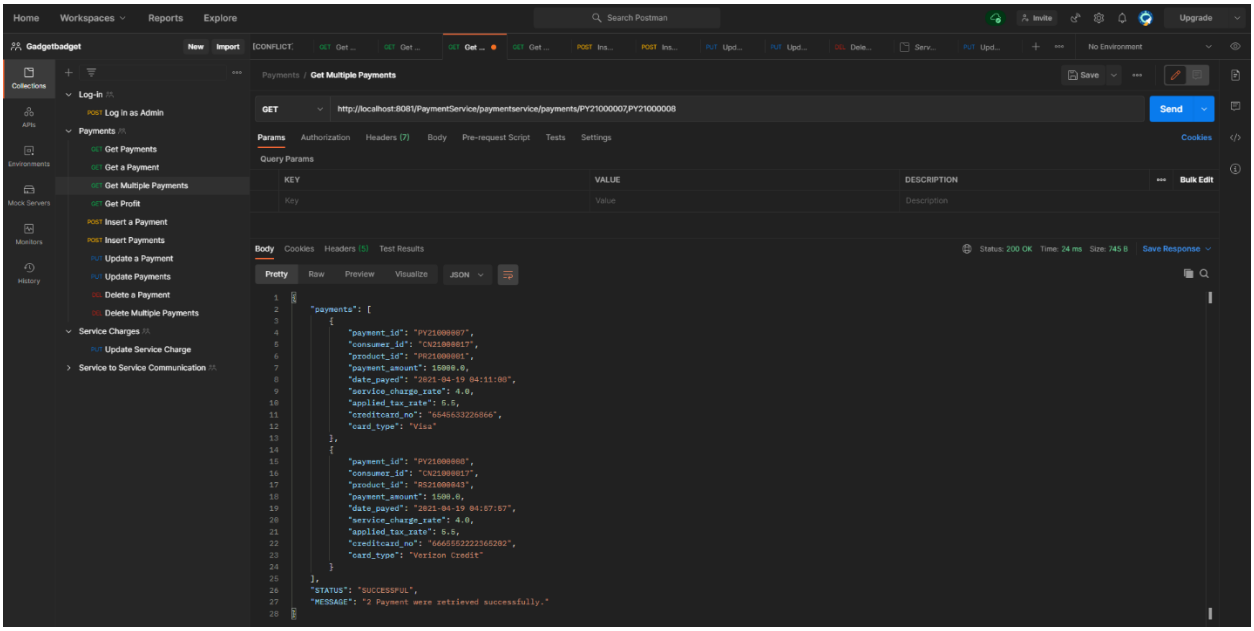


Figure C-2- 4 Get multiple Payments by giving or payment IDs. (Non-admin or financial managers are allowed only to retrieve their own payments)

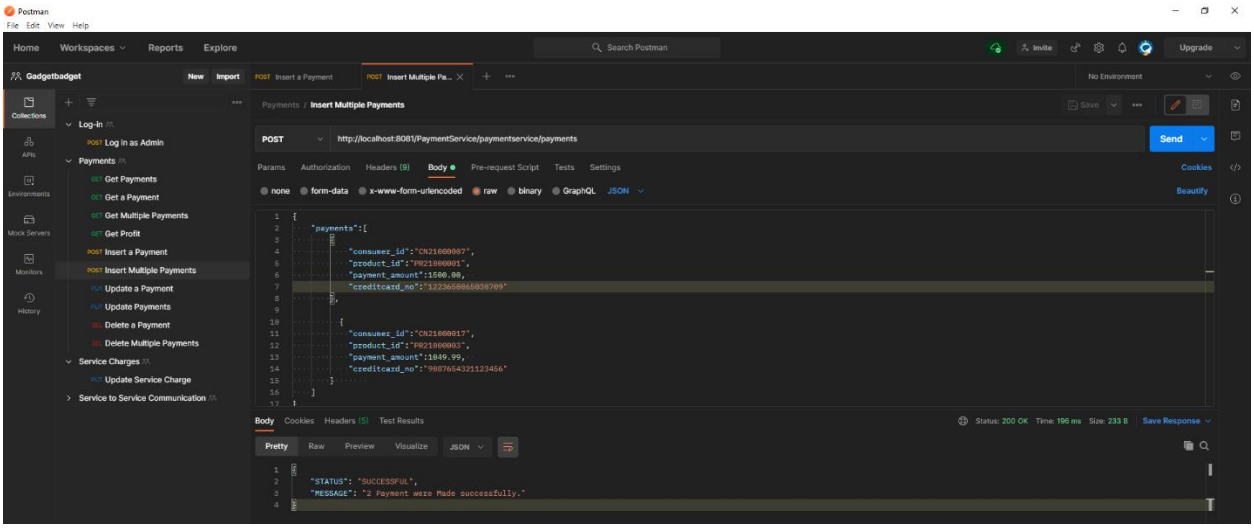


Figure C-2- 5 Insert Multiple payments at once (only allowed for financial managers and administrators to add multiple payments for any user. Consumers can only add multiple payments under their own consumer id)

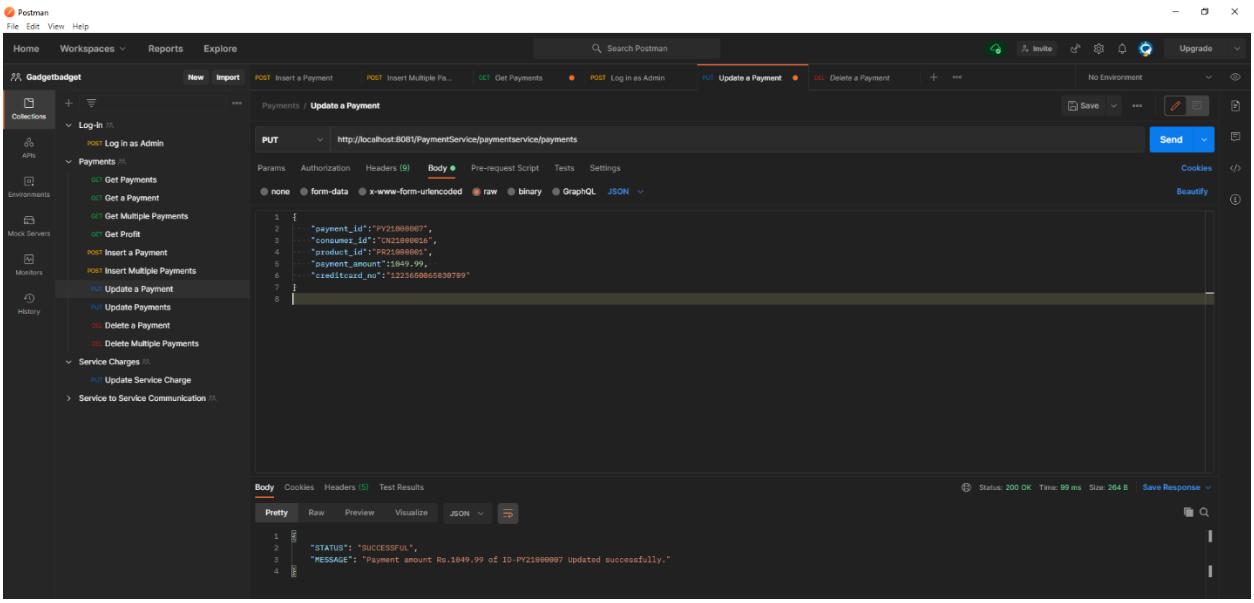


Figure C-2- 6 Update a single payment at once. Consumers are only allowed to update their own payments

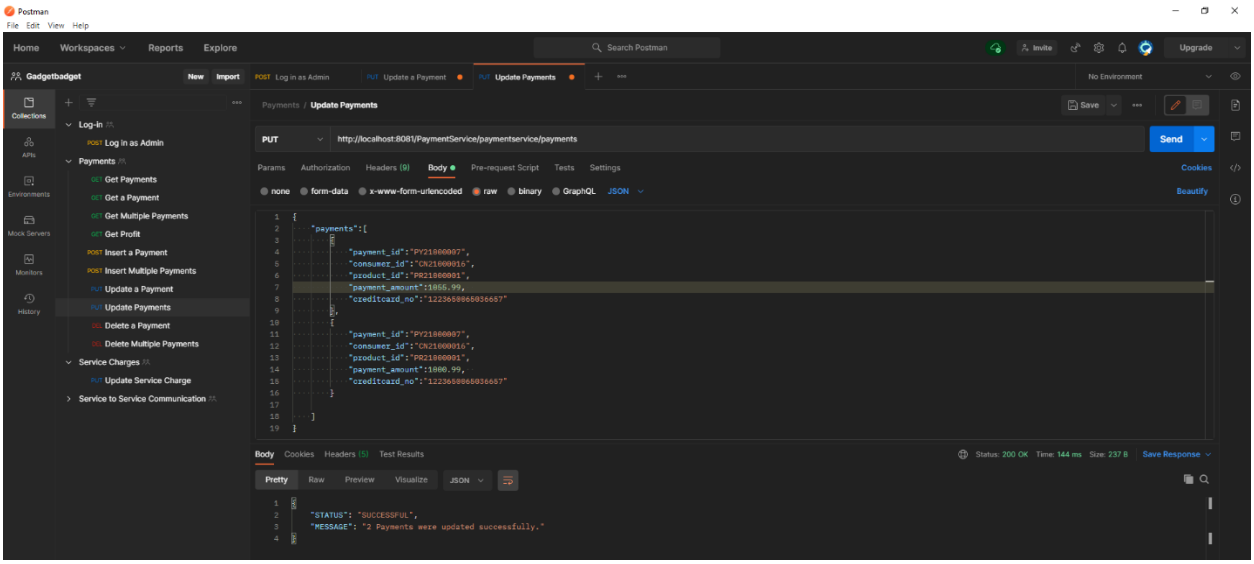


Figure C-2- 7 Updating multiple payments at once. (Note that consumers can only update payments under their own id)

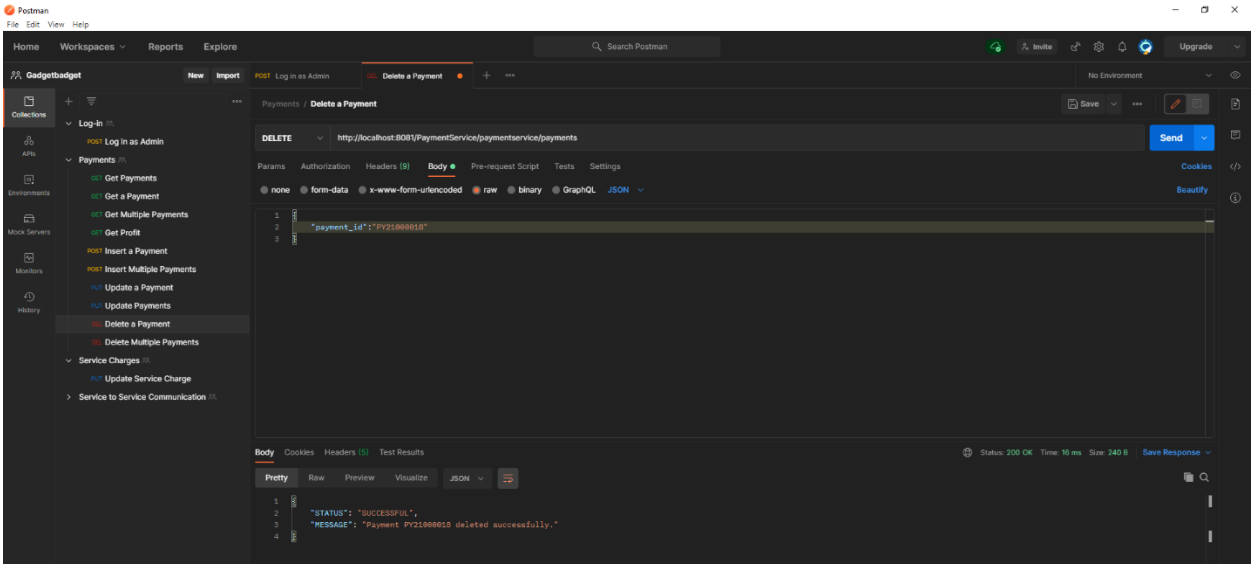


Figure C-2- 8 Deleting a payment

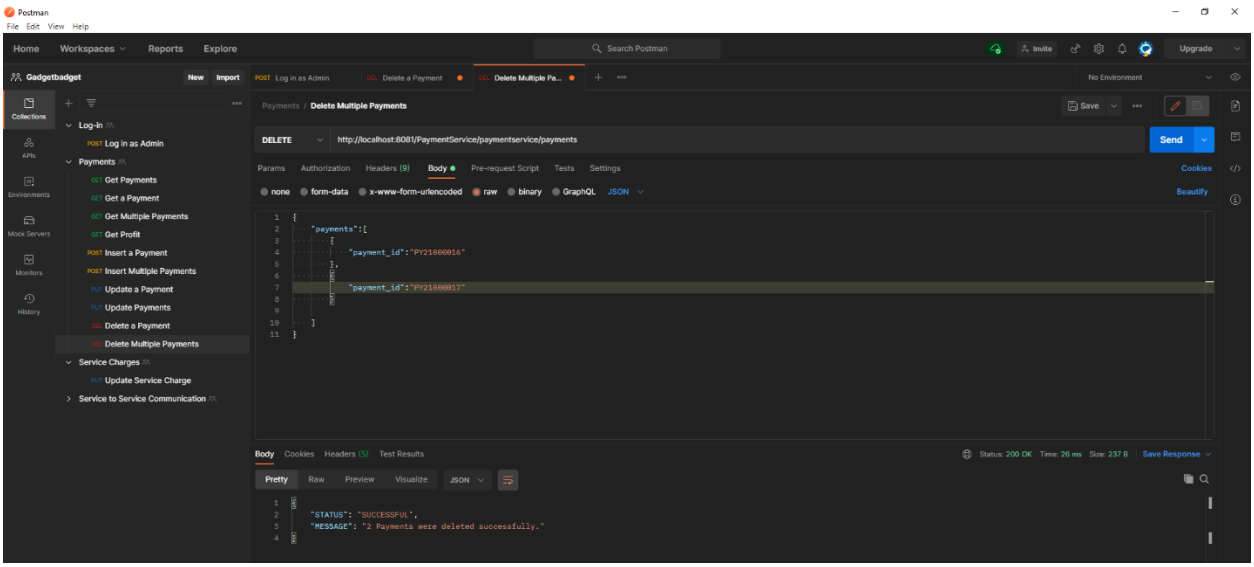


Figure C-2- 9 Deleting multiple payments at once. Note that consumers must delete payments made under their own IDs. Trying to delete other payments will cause "PROHIBITED" status.

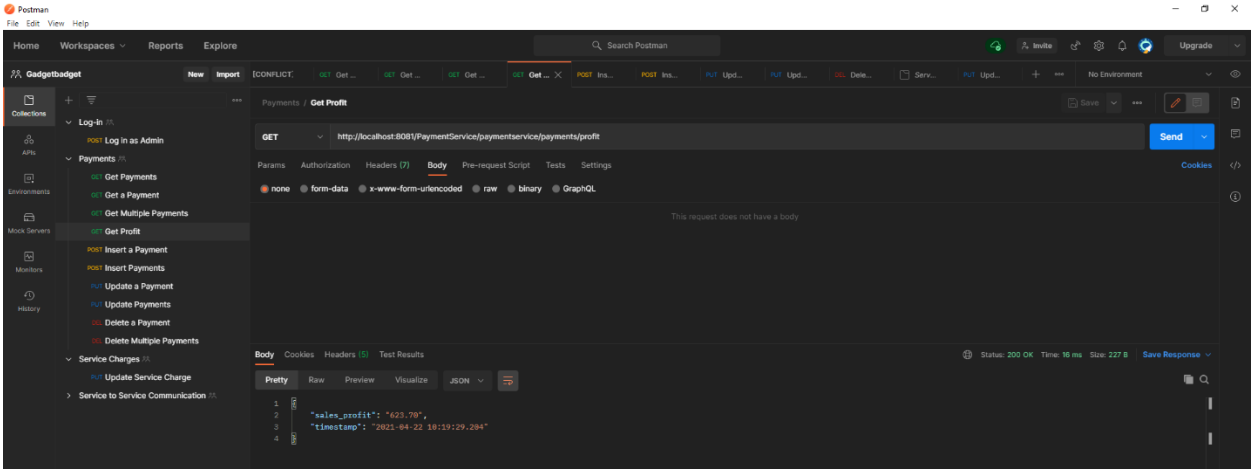


Figure C-2- 10 Financial managers and administrators are allowed to view total profit earned through received payments considering the service charges and tax rates.

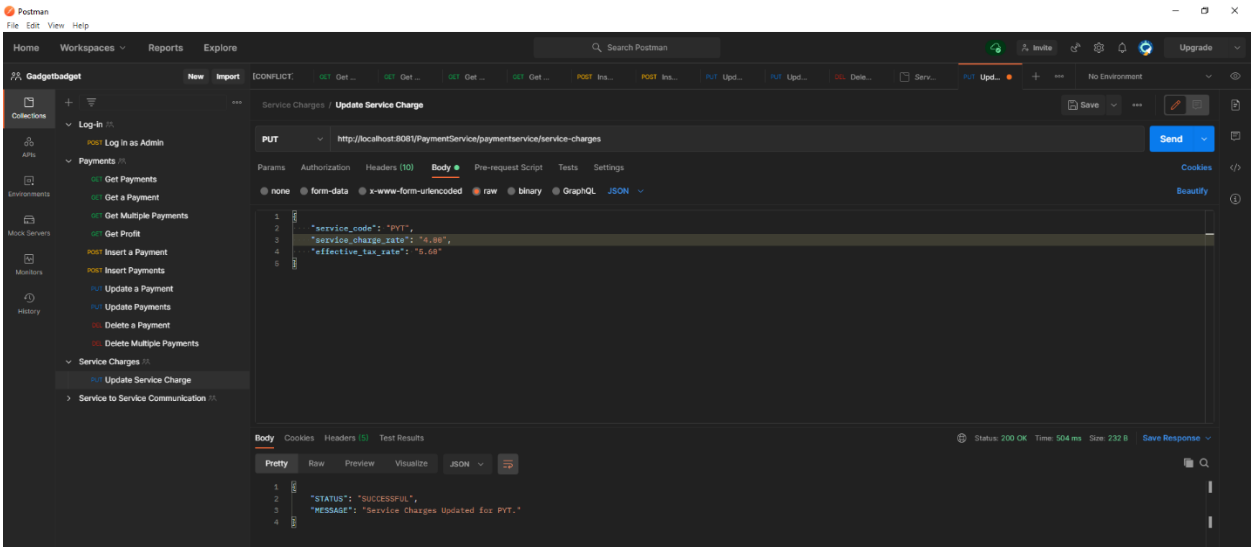


Figure C-2- 11 Administrators and Financial Managers can update the service change as well.

C-3. POSTMAN Test results of Research Hub Service

Research Projects

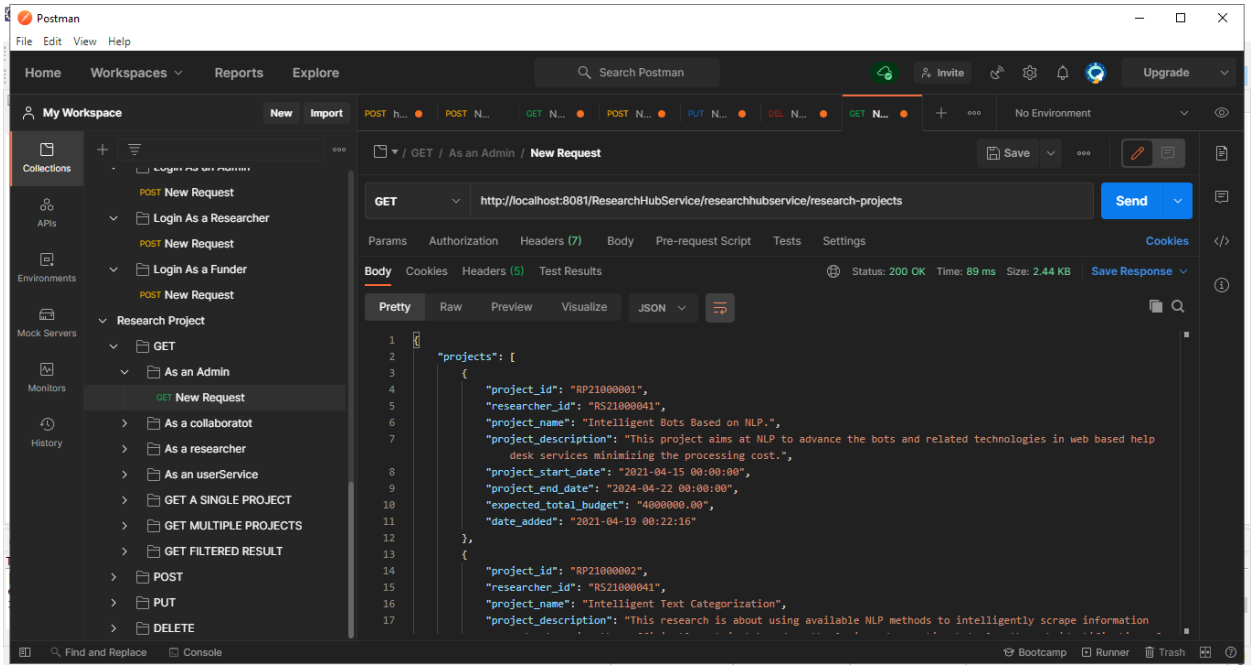


Figure C-3- 1 Read all research projects. (only authorized users are researchers, funders, and administrators)

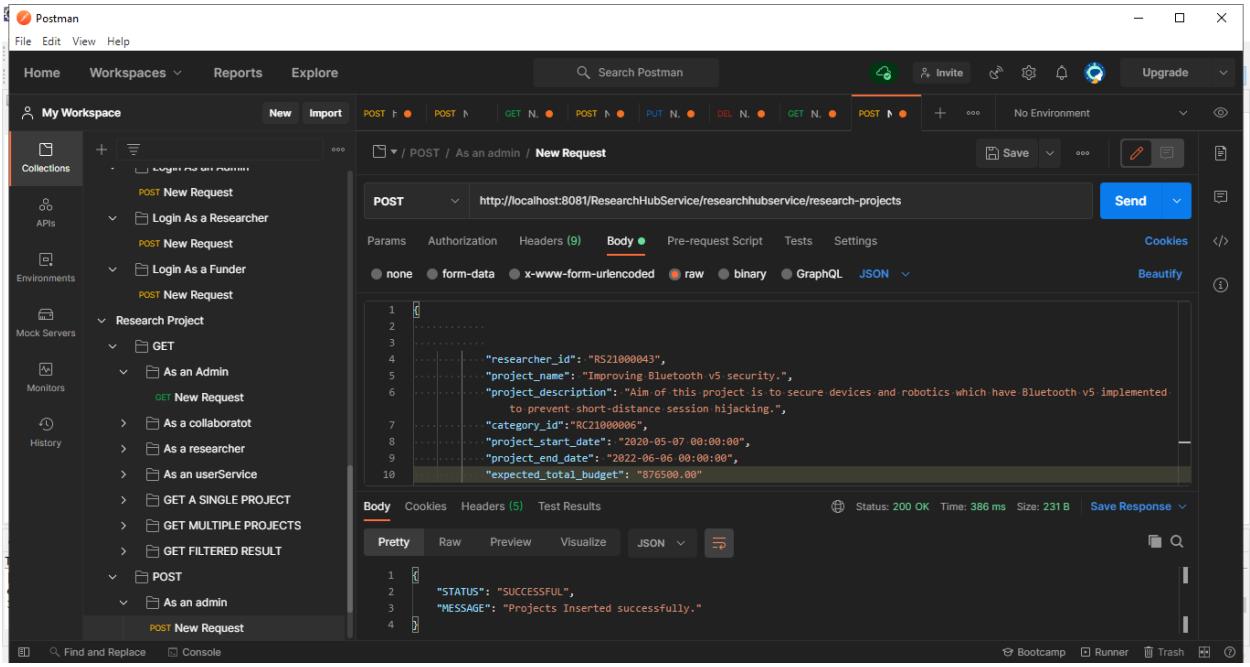


Figure C-3-2 Inserting a research project

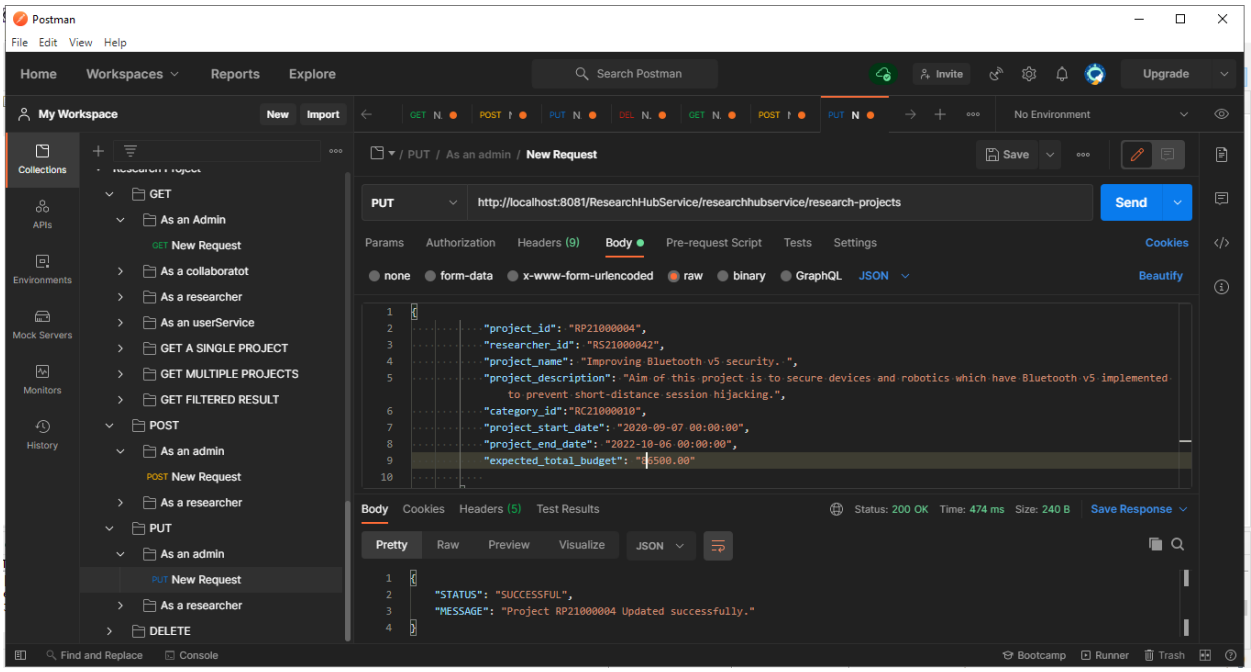


Figure C-3-3 Updating a research project

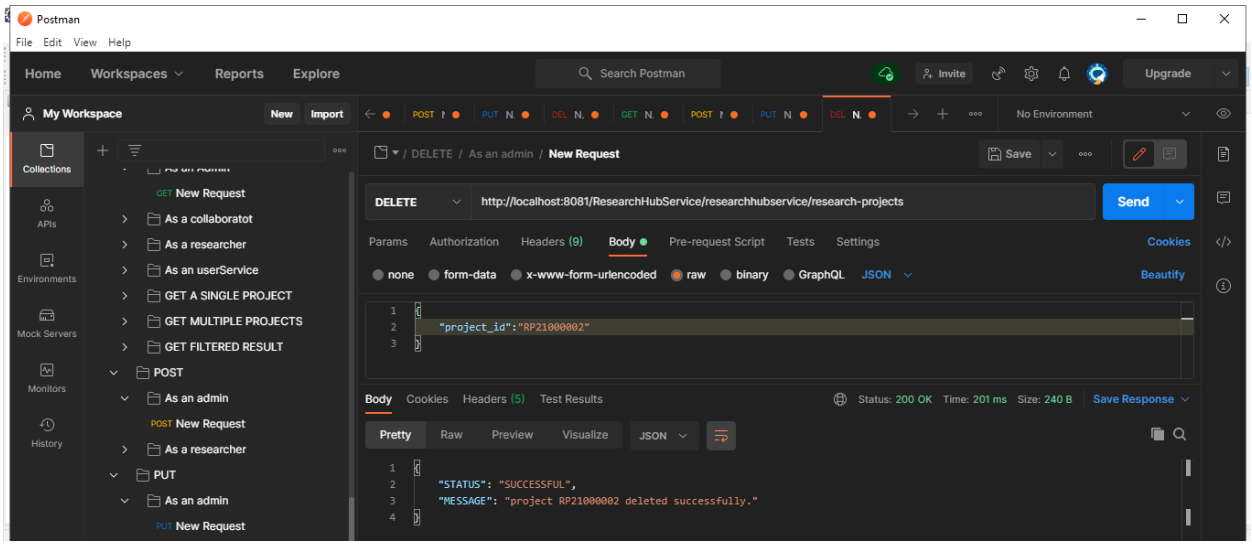


Figure C-3- 4 Deleting a single research project

Note that collaborator, categories CRUD operations can be performed by directing HTTP requests to http://localhost:8081/ResearchHubService/researchhubservice/{project_id}/collaborators and http://localhost:8081/ResearchHubService/researchhubservice/{project_id}/ctegories

C-4. POSTMAN Test results of Marketplace Service

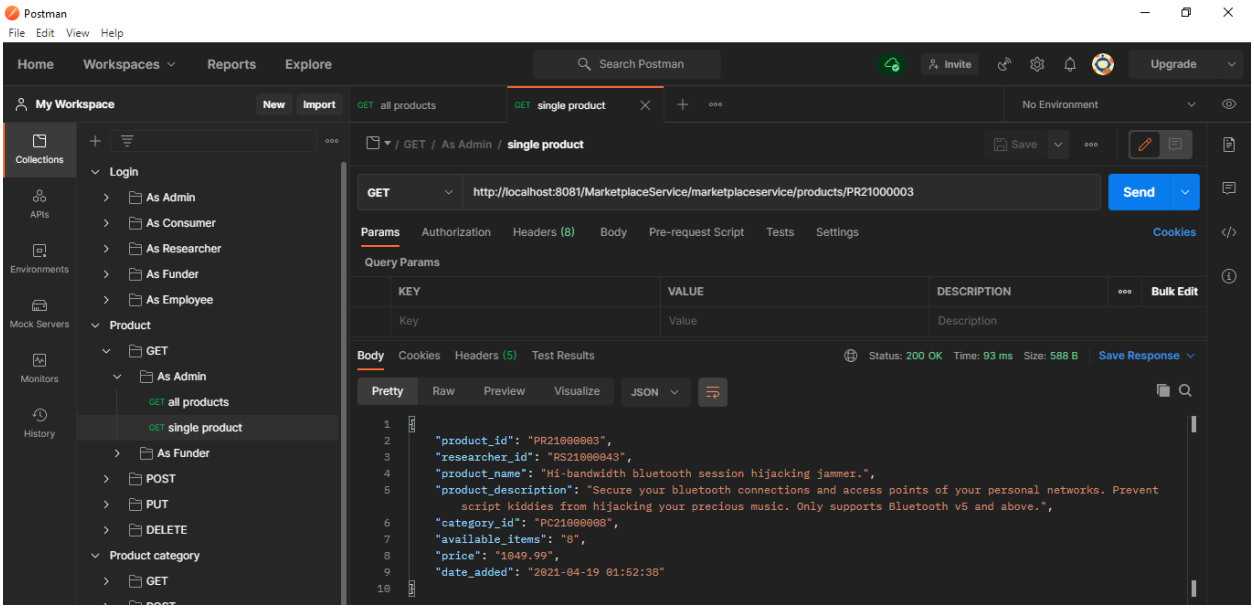


Figure C-4- 1 Retrieving a specific product from the marketplace service. Note that the researchers can only view products they have uploaded while consumers can view all products

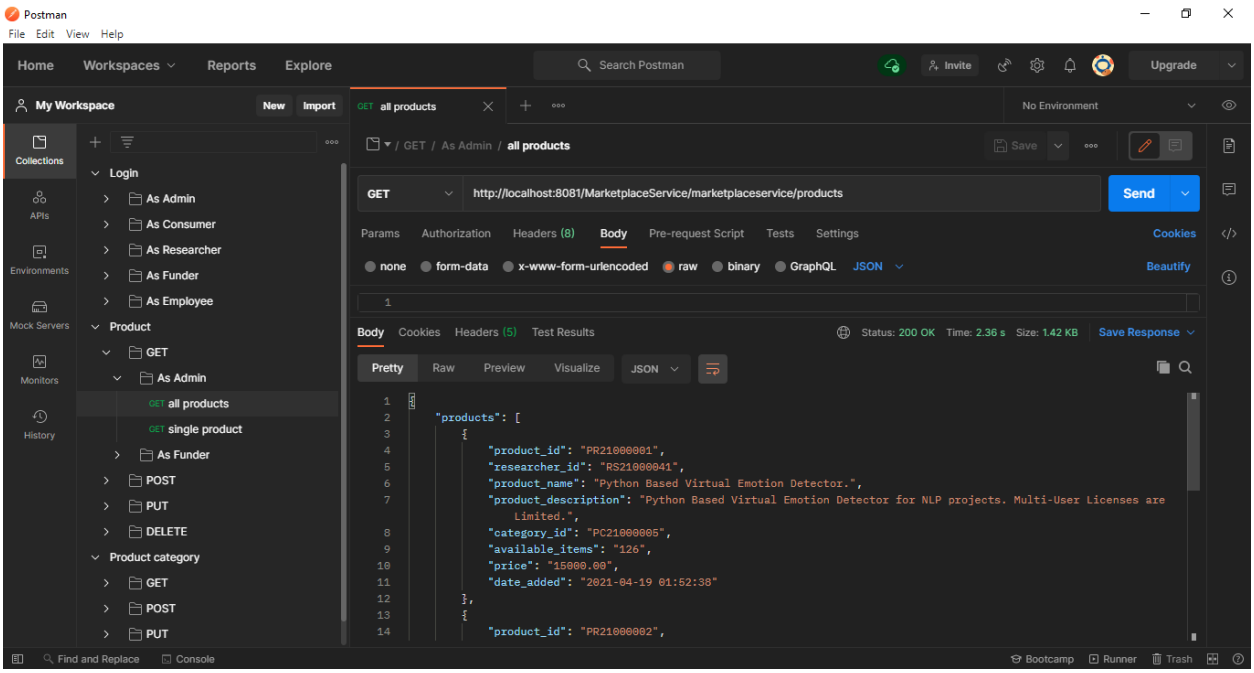


Figure C-4- 2 Getting the list of all products

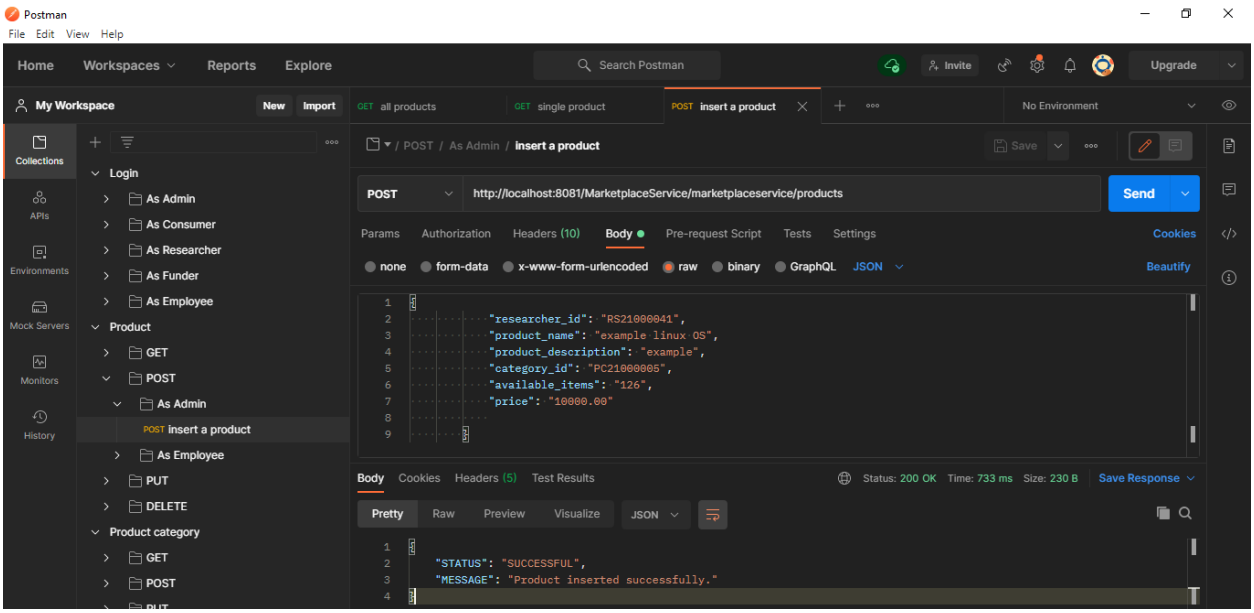


Figure C-4- 3 Inserting a new product. Researchers are only allowed to add products under their own ID

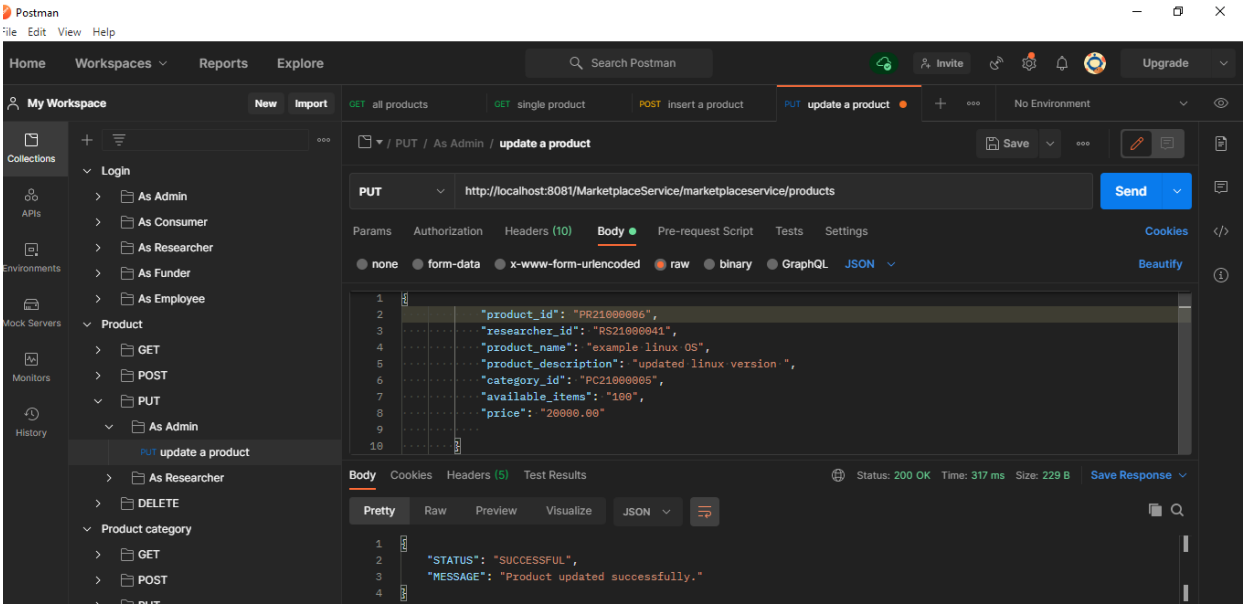


Figure C-4- 4 Updating a specific product

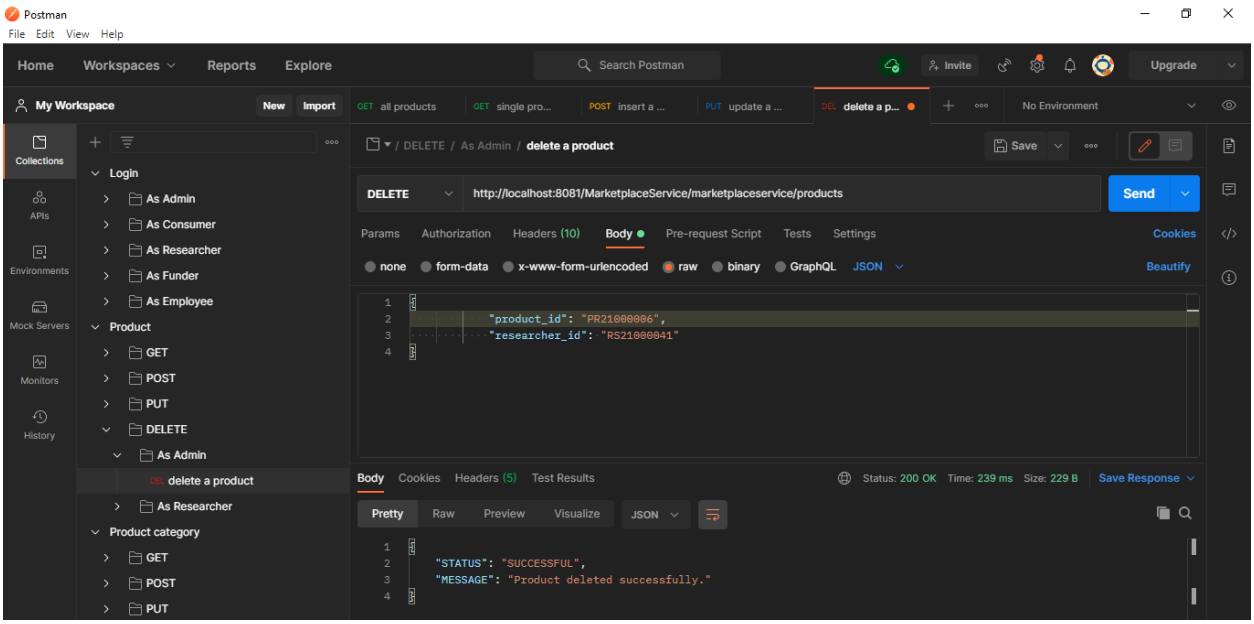


Figure C-4- 5 Deleting a specific Product

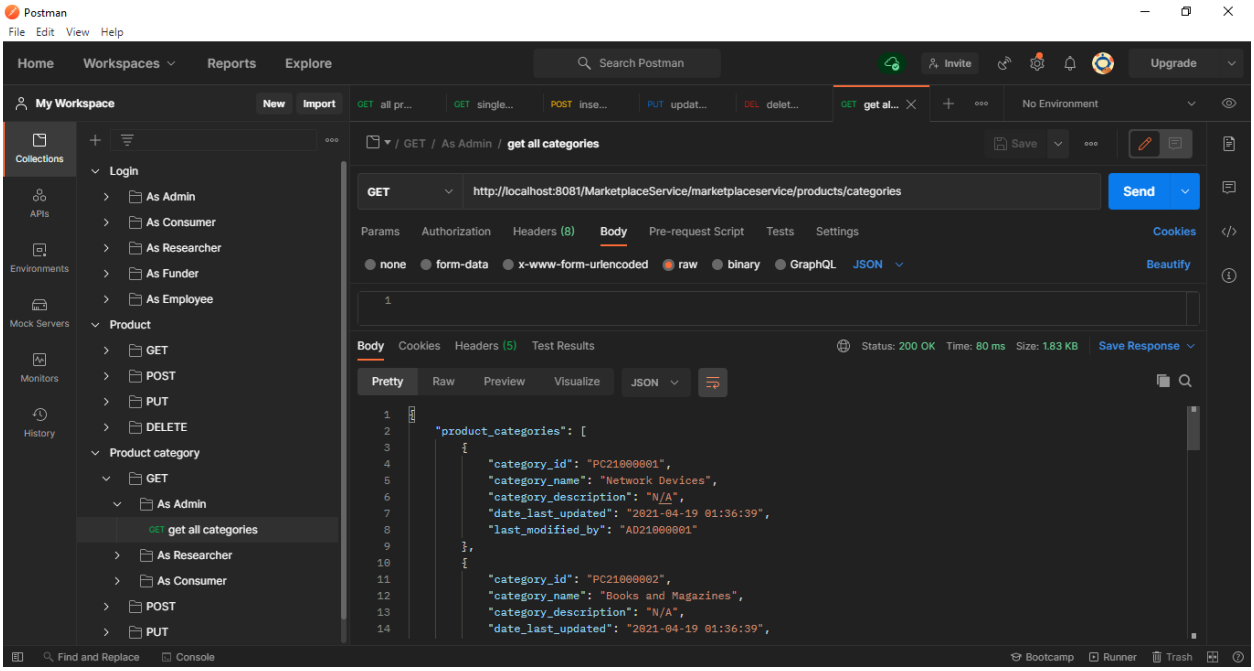


Figure C-4- 6 Get All product categories

Note that PUT, POST, and Delete operations can be done in the same manner for categories.

C-5. POSTMAN Test results of Funding Service

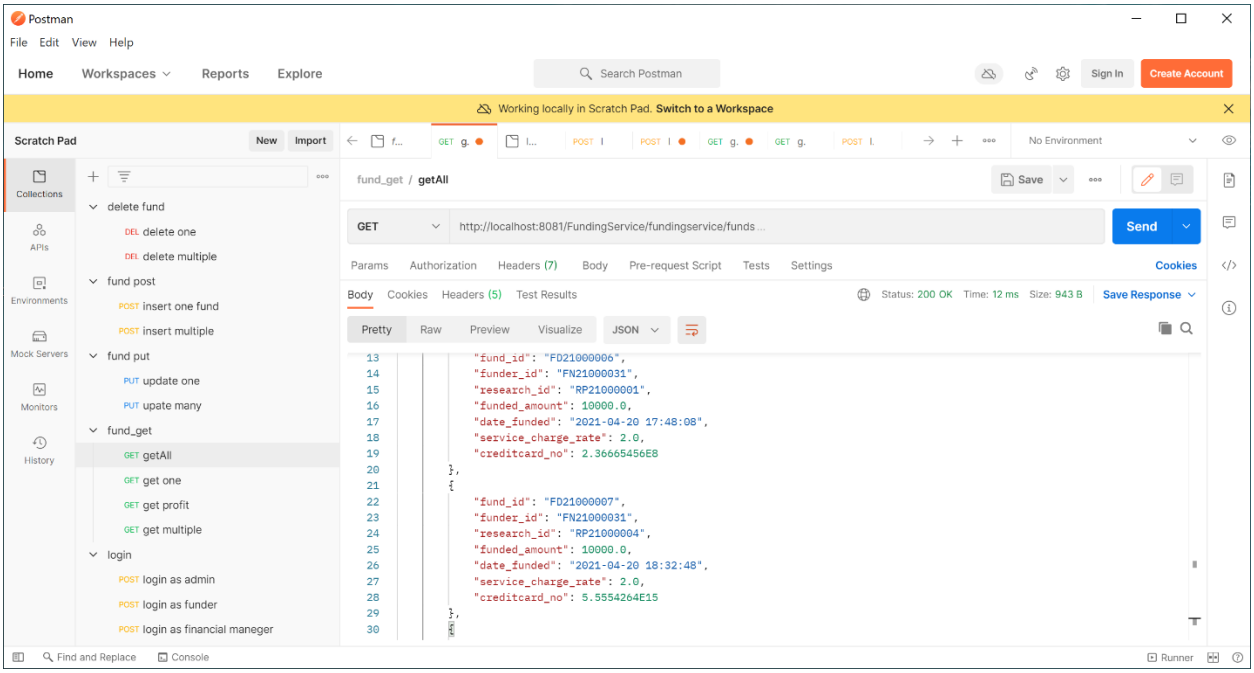


Figure C-5- 1 Get a list of all funds. Only admins can retrieve all the funds. funders can view only the funds placed under their user id.

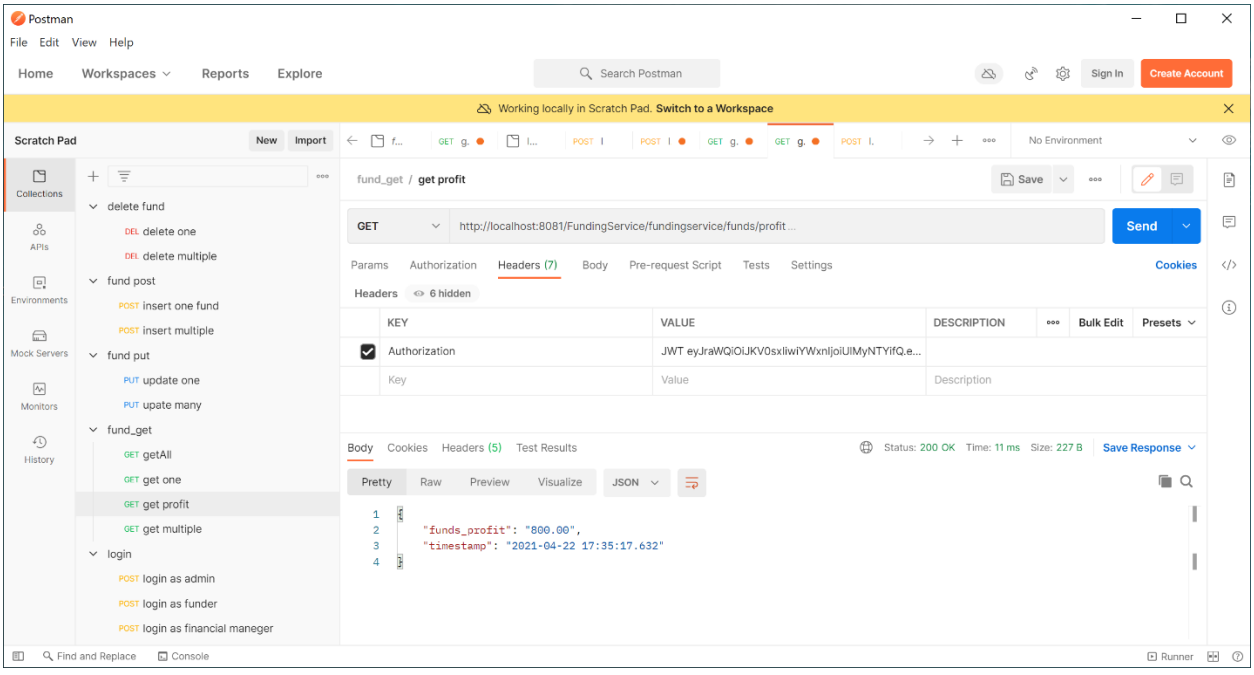


Figure C-5- 2 Financial Managers and Administrators can view the profit earned through the funds received

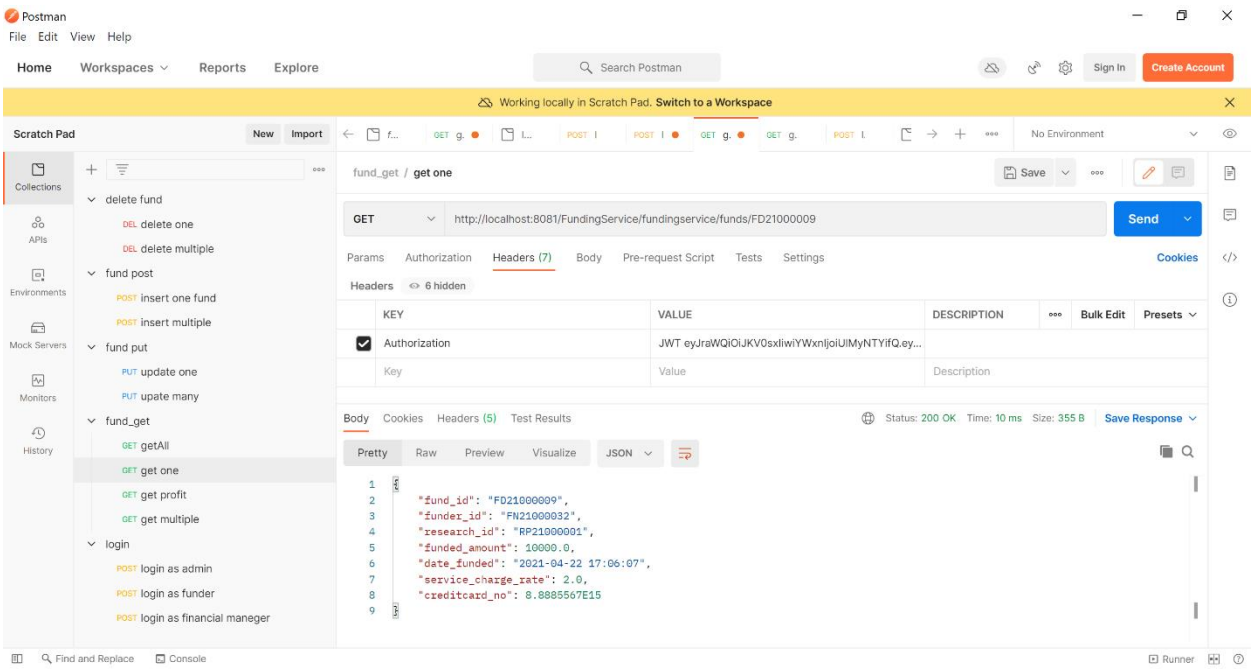


Figure C-5- 3 Retrieving only one fund

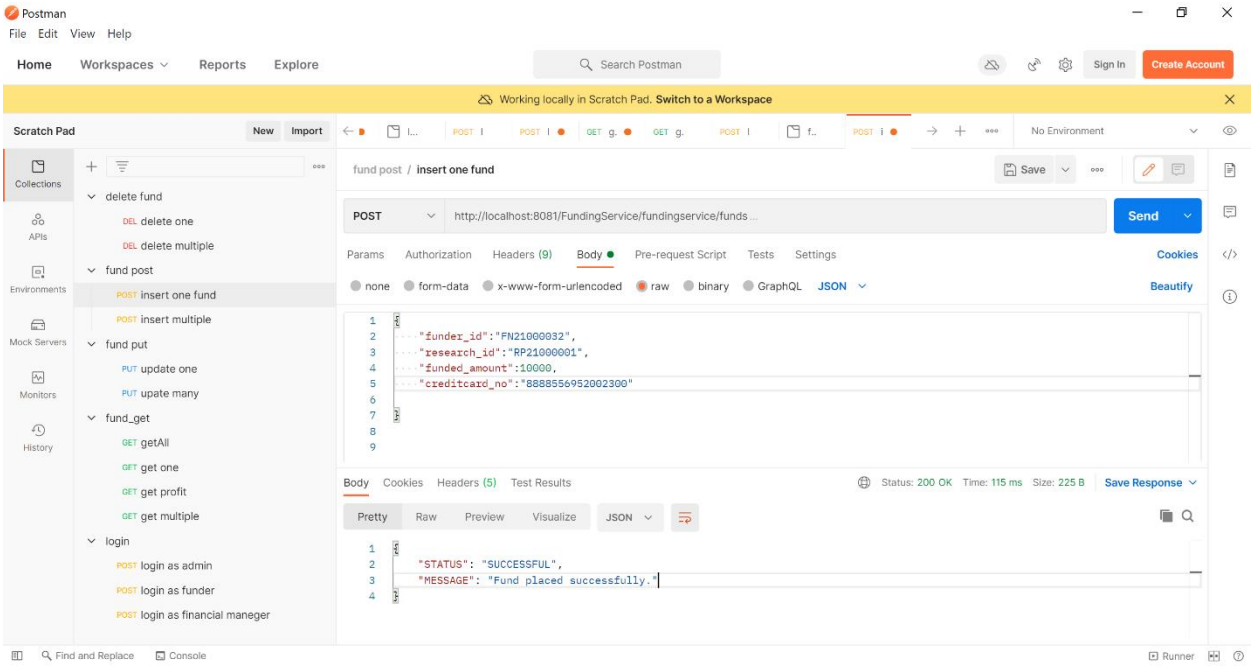


Figure C-5- 4 Insert Single Fund

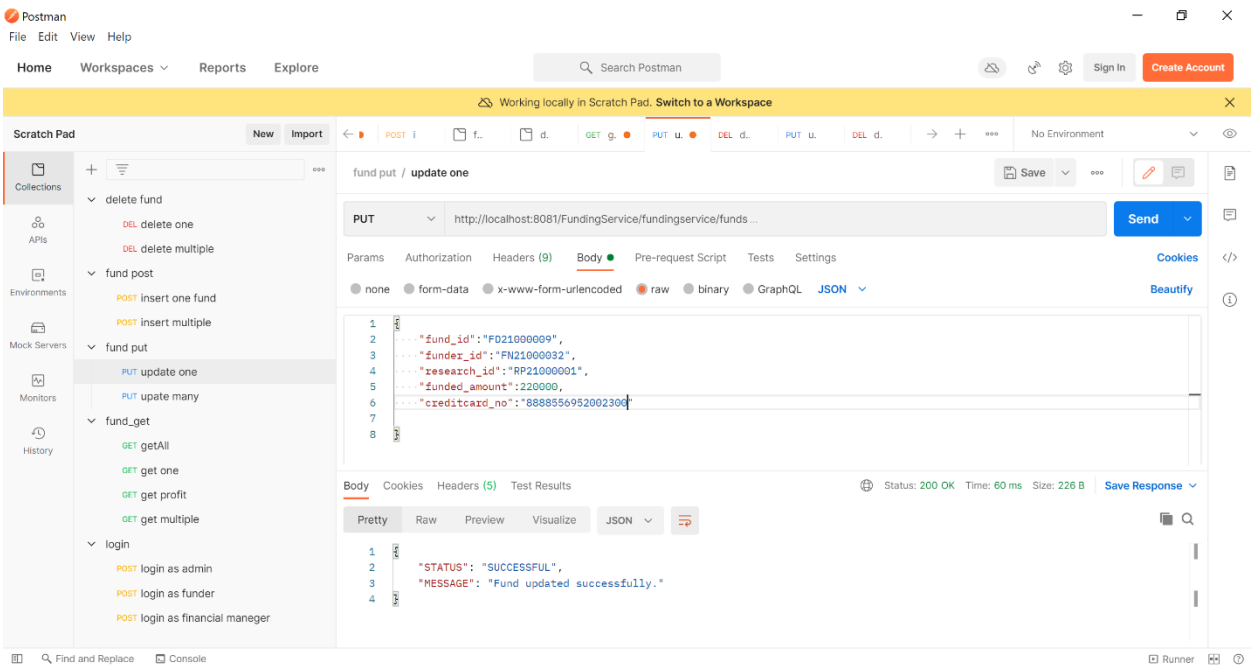


Figure C-5- 5 Updating an existing fund

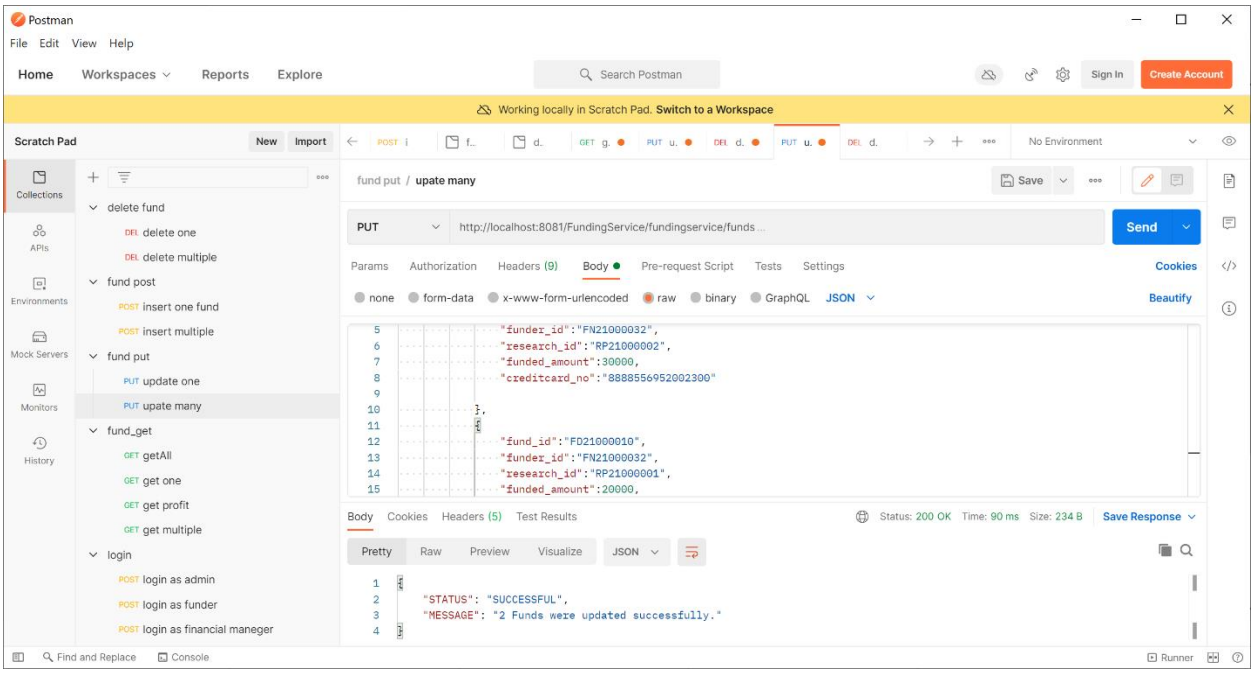


Figure C-5- 6 Updating Multiple Funds at a time.

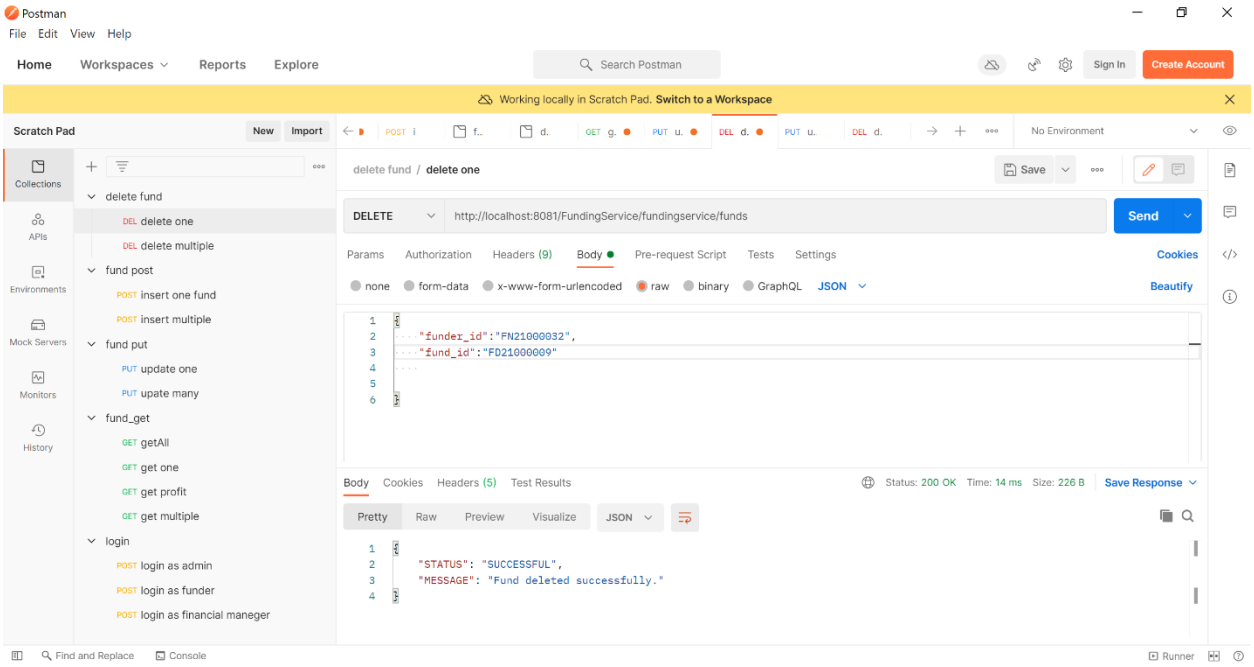


Figure C-5- 7 Deleting a single Fund

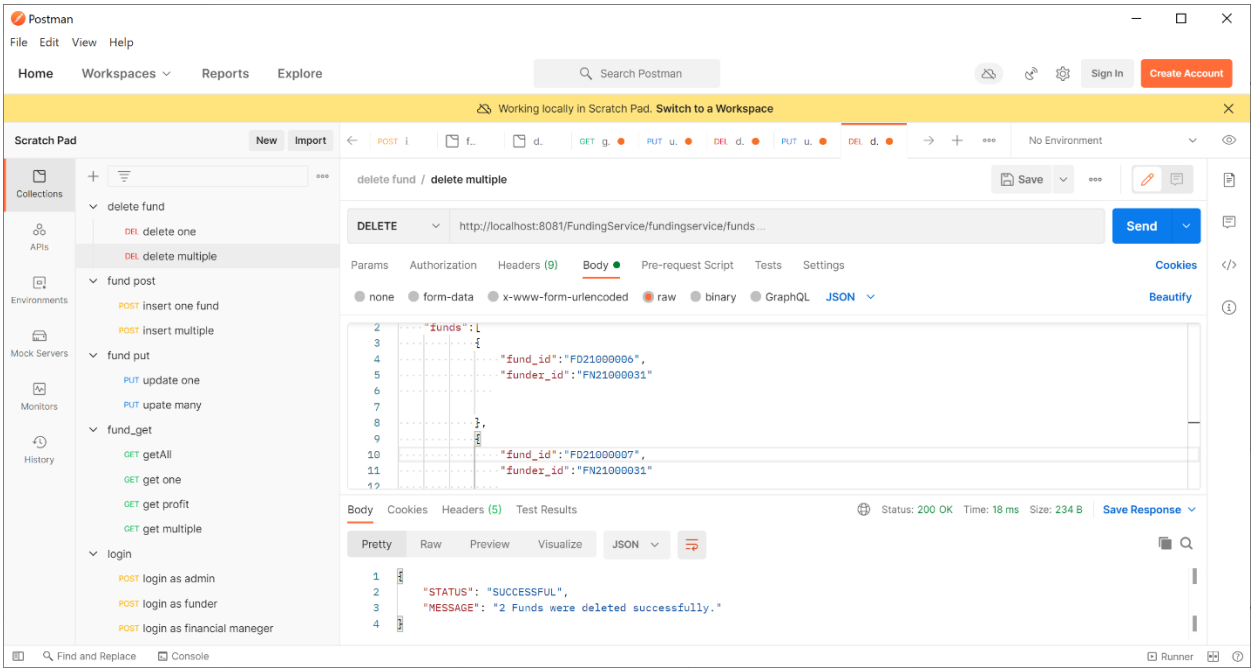


Figure C-5- 8 Deleting multiple funds at once