Sri Lanka Institute of Information Technology

GadgetBadget System

Final Report
Programming Applications and Framework (IT3030)
2021

IT3030PAF2021_GroupProject_GroupY3S1.15(DS)-03

Submitted by:

1. IT19021430 - J.R. Hillary
2. IT19162010 - S.K. Yadushika
3. IT19175058 - M.S.S Mariyam
4. IT19080840 - K. Kovishwakarunya
5. IT19058474 - T. Vithyashagar

Table of Contents

1.0 Member details and workload distribution

Student ID	Student Name	Workload Distribution
IT19021430	Hillary J.R.(Group Leader)	Product Service Management • Add new products • View products • Update products • Delete products
IT19058474	Vithyashagar S.T.	Manufacturer Service Management • Add Manufacturer service • Update Manufacturer service • Remove Manufacturer service • Read Manufacturer services Patent Management • Add Patent Application • Remove Patent Application • Read Patent Applications
IT 19080840	Kovishwakarunya K.	Payment Service Management • Add backer and buyer payments. • View payments • Update Payment details • Delete payments
IT 19175058	Mariyam M.S.S.	Campaign Management • Insert new concepts • View concepts • Update concept details • Delete concept
IT 19162010	Yadushika S.K.	User Management • Add user (Register) • View user profile • Update user details • Delete user

2.0 Clickable link to the public VCS repo

https://github.com/HillaryJR/IT3030_PAF_2021_GroupProject_Group03.git

DB Query included

3.0 SE methodology

Agile methodology has been used in this project to achieve an organized and efficient software development process. Agile is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. This approach can be used for software that provide a continual planning, learning, collaboration and development. Also, it helps a development team to easily adjust to software changes.

Agile methodology centers around time boxed cycles known as sprint. Sprint is a short period of time during which the team will work to develop a set of features. These features are collected as 'user stories.' This makes the development process more manageable. During the sprint, the team develops the planned features and involves in daily stand up meetings to discuss the progress and problems. At the end of the sprint, the next sprint is planned. Similarly, our team analyzed the requirements, and followed a sprint of two weeks. Short meetings were held daily, to discuss the progress.

The reason for choosing agile methodology as the software development methodology was because it provides a better understanding about the software. Further, this approach provides a better control over the project, higher flexibility towards changes and a better predictability. More importantly the stake holder involvement is high in this approach, which leads to better customer satisfaction.

4.0 Time schedule (Gantt chart)

The gantt chart is a project management tool which gives a visual display of the whole project, timelines and deadline of all tasks, relationships and dependencies between various activities and project phases. Therefore, Gannt chart is chosen as a project management tool by the Development team of Gadget Badget system development since the tool assists in the planning and scheduling of project. This tool gave the team members visible workloads, as well as current and future availability which allowed the tam for more accurate scheduling.

	February		March					April				
TASK	Week3	Week4	Week1	Week2	Week3	Week4	Week5	Week1	Week2	Week3	Week4	Week5
Read the assignment & created the group	■	■										
Registered the group		■										
1st group meeting and discussion (MS-Teams)			■									
Refer similar websites and understanding the Aim of project			■									
Requirement gathering and analysis			■	■								
Set the scope and functions				■								
Divided functions (Services) within members				■								
Draw ER diagrams					■							
Draw system overall architecture					■							
Draw overall activity diagrams					■							
Draw use case diagrams					■							
Feedback session						■						
Creation of Data Bases						■						
Draw class diagrams						■						
Finalizing the tools and technologies to be used						■						
Created github repository for project						■						
Started to code with regular commits to git.							■	■	■			
Completing each service									■			
Interservice communication									■			
Fine tuning each service									■			
Testing										■		
Final report writing										■	■	
Report submission											■	

Figure 1: Gannt Char

5.0 Requirements

5.1 Stakeholder Analysis (Onion diagram)

The stakeholder analysis is a process of identifying these people before the project begins, grouping them according to their levels of participation, interest, and influence in the project and determining how best to involve and communicate each of these stakeholder groups throughout. According to the stakeholder analysis diagram any organizational project, all of the internal people and teams who the project will involve or affect are called its stakeholders.

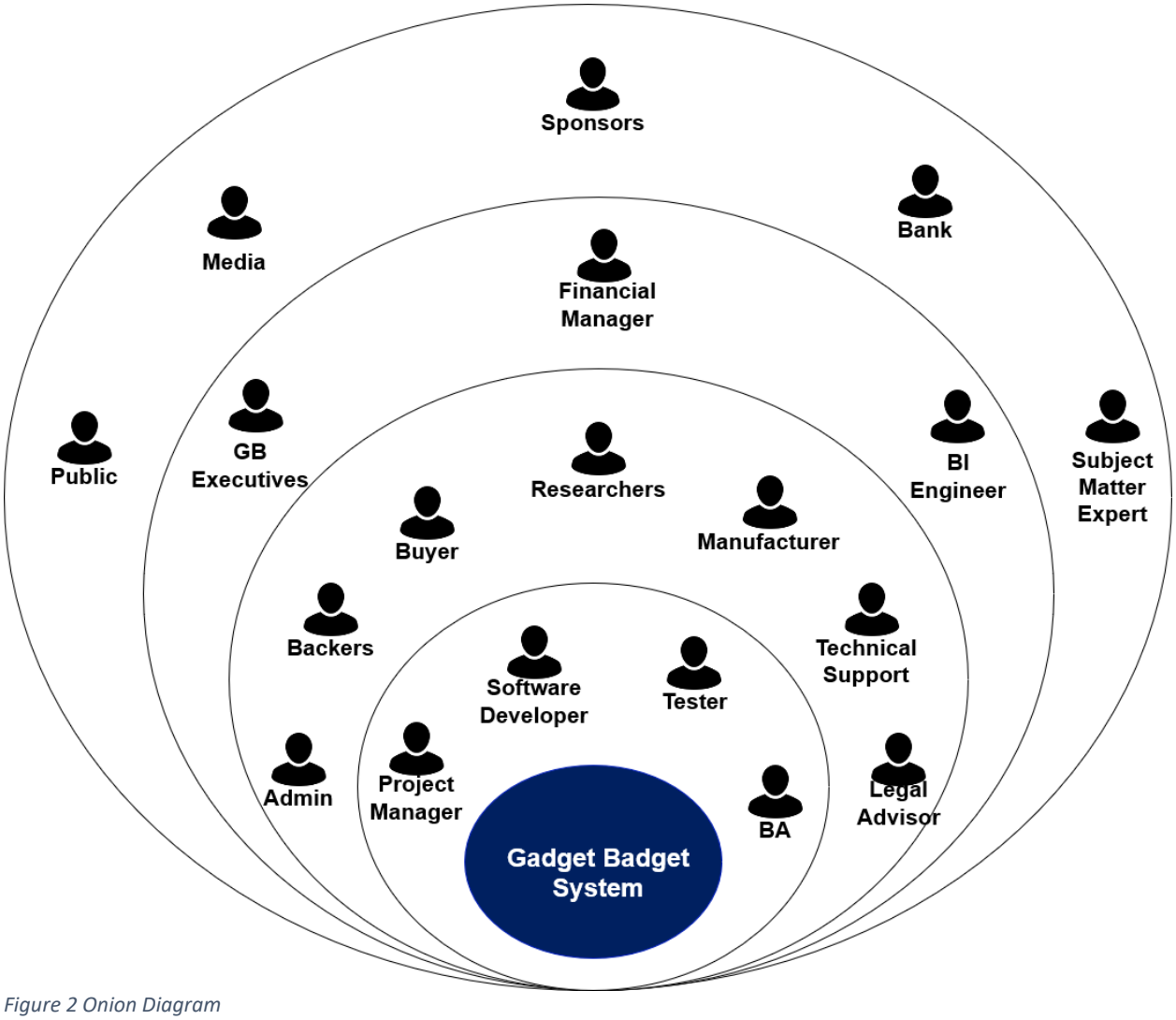


Figure 2 Onion Diagram

5.2 Requirement Analysis

5.2.1 Functional Requirements

In simple terms functional requirements are the primary ways in which the customer communicates their requirements to the project team.
In accordance with the project requirement specification and scope we as a team came up with five individual web services to fulfill the functional requirements of the system in an efficient manner.

1. User service
   Various user aspects are handled by this service. Users should be able to login to the system as a researcher who uploads innovations, as a consumer who can pledge money for concepts initiated by the researcher as well as buy the ultimate product after completion of project. In addition, a user should also be able to login as a manufacturer assisting concept completion. Individual user profiles are also maintained by this service.

2. Campaign Service
   User and innovations are handled by this service. Researchers must be able to manage concepts they add while any other consumer should be able to back concept added by a researcher.

3. Product Service
   Managing the selling of new innovative projects are handled by this service.
   A launched concept can be published so that it could be viewed and bought by any consumer.

4. Manufacturer Service
   Assisting completion of Researcher concepts are handled by this service.
   If requested by the researcher, a manufacturer can be assigned to assist and complete a concept.

5. Patent Service
   Guiding researchers to obtain concept patents are handled by this service.
   If researcher requires their innovation to be approved with patent rights, the system functions as an agent to proceed the steps for the process.

6. Payment Service
   User payments are handled by this service. Backer payments and buyer payments are handled with adherence to company policies ultimately providing digital receipts for payment confirmations.

5.2.2 Non-functional Requirements

Nonfunctional requirements specify how a system should behave and what limits are there to fulfill a system's functional requirement. Simply put, nonfunctional requirements are system properties and user expectations.

1. Scalability

As the stakeholders start increasing rapidly, the system should be able to handle the concurrent users of the system to provide high customer satisfaction.

2. Extensibility

New features for an individual functionality should be added without hindering the existing functionality.

3. Availability

The system should be highly scalable thus enabling concurrent users to interact with the systems without any hinderance and providing the accessibility to any user at any required time.

4. Security

Since there are many users and user levels accessing the system, the system must restrict users accessing unauthorized information. User data should be hashed and stored in a confidential manner.

5. Flexibility

Many aspects of the service should be able to be handled by various technologies without affecting each other.

6. Performance

When multiple users access the system the response time taken for a request should be less enabling the user to continue using the system with ease.

5.2.3 Technical Requirements

A technical requirement specification for a web-based service or other software product defines its purpose, functionalities, and behavior. According to this, the system can be accessed through operating systems such as Windows, MacOS or Linux. The web browsers like Google Chrome, Mozilla Firefox and Opera will support a lot better than the old browsers (e.g., Internet Explorer). The web application can also be accessed through the browsers of mobile phones. Authentication implemented and the security features are developed.

6.0 System's Overall Design

6.1 Overall architecture

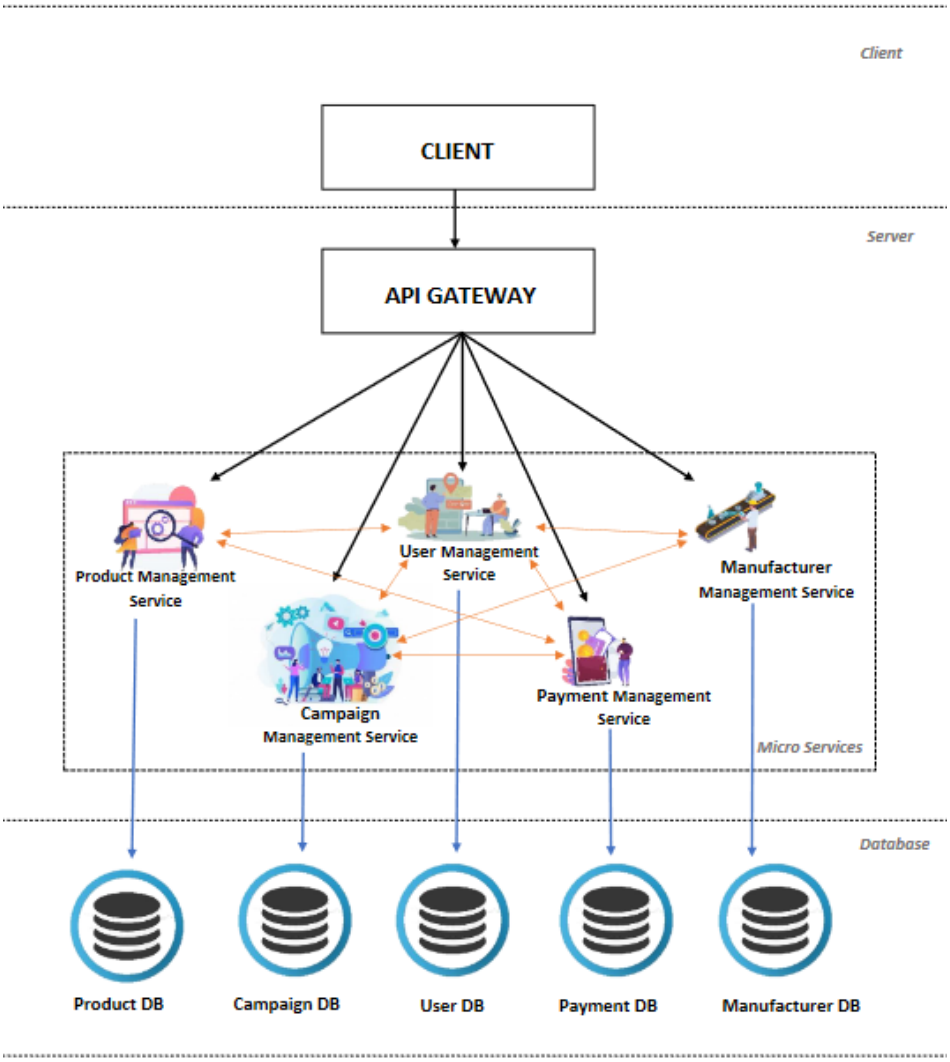


Figure 3: Overall architecture diagram

The Gadget Badget system was developed based on the Micro service architectural style as per the requirements, to increase the scalability of the system. Microservices are an architectural style that is used, instead of developing a single application for the system. Each service runs in its own process thus forming a larger microservices-based application. The services can communicate with clients, and often each other, using lightweight protocols, often over messaging or HTTP. The terms RESTful API and Microservices go hand-in-hand when building a microservices-based application. RESTful APIs are the rules, commands and protocols that integrates the individual microservice, so they function as a single application. The APIs merely expose database CRUD operations performed by these services. The Gadget Badget system basically consists of a client side who request for the resources and a server side who has the resources and responds to those requests. Like almost any other software's, the API will reflect the needs of the humans who interact with it. The chosen RESTful APIs and microservices architecture offers the system with scalability, resiliency, easy upgrades, security, smaller teams for development and many more advantages.

6.2 Overall DB design-ER

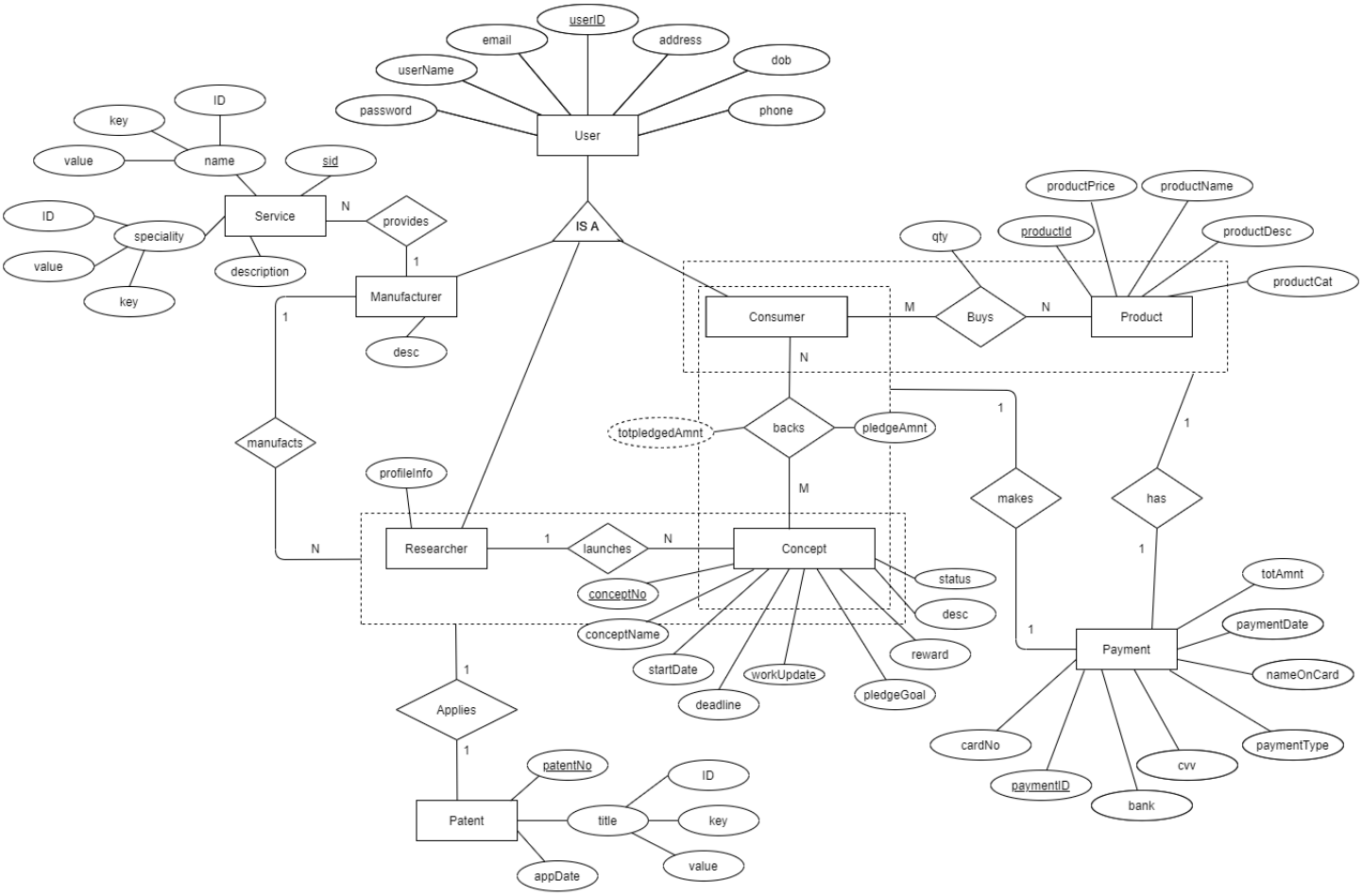


Figure 4: Overall ER diagram

- The **users** of the overall system can be a **consumer**, **researcher**, or a **manufacturer**. Therefore, they share an IS-A relationship. The three entities are assumed to cover the user and are disjoint.
- The researcher is responsible to launch a **concept**. A researcher can launch many concepts and a concept is launched by one researcher only.
- The consumers can support to produce these concepts by backing them. A pledge amount must be provided to back the concept. A consumer can back many concepts, while a concept can be backed by many consumers.
- **Payment** must be made for each concept that is backed by a consumer.
- If the concept reaches the total pledge goal amount, the system provides the ability to sell them through the system itself. Therefore, the database stores details of the **products** too.
- A consumer can buy many products, while a product can be bought by many consumers.
- For each product bought by the consumer, a payment must be made.
- When launching a concept, the researcher can apply for **patent** for the concept, if he/she wishes.

- Further, the researchers can also get the help of a **manufacturer** to manufacture the concept that is launched by them. Manufacturer can manufact many concepts launched by a researcher.
- The manufacturers provide a **service** to manufact the product. A manufacturer can provide many services.

6.3 Activity Diagrams

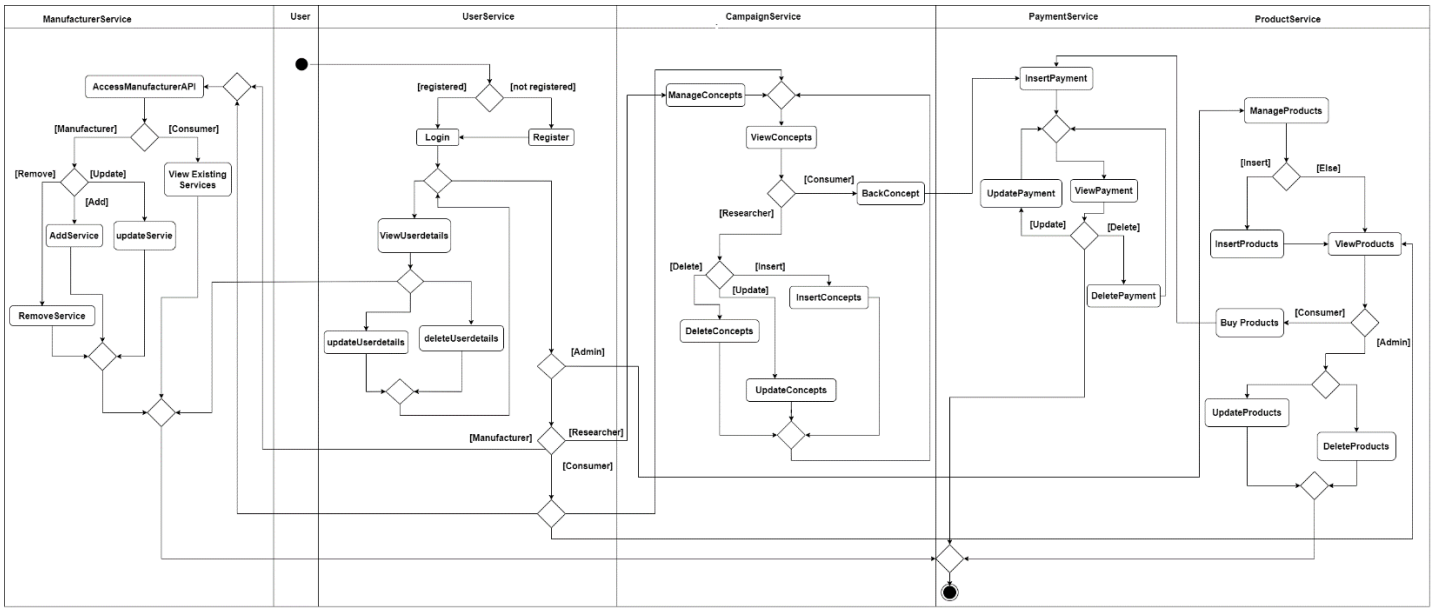


Figure 5: Overall Activity Diagram

- The above diagram illustrates about the overall dynamic aspects of the system.
- The Users of the system can be a Researcher, Manufacturer, Backer, or a consumer.
- To perform any tasks in the system the user must be logged in.
- If user do not have an account to login, he/she can register to the system by providing user specific details.
- All users can view their profile and update their details and if necessary, they can also delete the account.
- If the logged user is a Researcher, he/she can post their concepts in the system to collect the demanding fund by the crowd funding concept.
- Also, the researcher can request a patent through the system where the application will be forwarded to the relevant department representing GadgetBadget as an agent.
- If the logged user is a backer, if he/she wishes to support a concept posted by a researcher then he/she may go through the process and fund the project.
- If the logged user is a manufacturer, he/she can provide(post) services that could support the researcher to finish the concept by producing the product.
- If the researcher wishes to produce the product, he/she can browse the system for potential Manufacturers and handover the production.
- If the researcher wishes to add finished products to the system, he/she can make request to System to add the product by providing necessary information.
- Consumers can view the products that are posted in the system and can make purchases.

Consumer/Backer can make payment by providing the requiring details to the system.

- Please find more information on Appendix 1: Other relevant diagrams – Class diagram, Use Case diagram, Flow chart diagram

7.0 Individual section

7.1 User Management Service - Yadushika S.K. IT19162010

7.1.1 API design

The application programming interface of the user management service is implemented according to the gathered requirements. The main resource used for the implementation of the API is 'user.' All create, read, update and delete operations are performed on this resource. The API design is as follows:

POST

- Resource: User
- Request: POST *http://localhost:8009/UserService / UserService /Users*
- Media Type: APPLICATION_FORM_URLENCODED
- Method: insertUser()
- Data: userName, password, email, address, dob, phone, desc, profileInfo
- Response: String status message "User Inserted successfully" or "Error while inserting the user"

GET

- Resource: User
- Request: GET *http://localhost:8009/UserService/ UserService/Users/<UserName>/<password>/<type>*
  - Consumer : *http://localhost:8009/UserService / UserService/Users /<UserName> /<password>/consumer*
  - Manufacturer : *http://localhost:8009/UserService / UserService/Users /<UserName> /<password>/manufacturer*
  - Researcher : *http://localhost:8009/UserService / UserService/Users /<UserName> /<password>/researcher*
- Media Type: TEXT_HTML
- Method: userLogin()
- Data: {}
- Response: An HTML Table, String status message "Login Successful !! You're logged as <userName>" or "Error while logging".

GET

- Resource: User
- Request: GET *http://localhost:8009/UserService / UserService/Users /<type>*
  - Consumer : *http://localhost:8009/UserService / UserService/Users/consumer*
  - Manufacturer: *http://localhost:8009/UserService/ UserService/Users/manufacturer*
  - Researcher *: http://localhost:8009/UserService / UserService/Users/researcher*

- Media Type: TEXT_HTML
- Method: readUsers ()
- Data: {}
- Response: An HTML Table or "Error while reading the users".

- **PUT**
- Resource: User
- Request: PUT
  http://localhost:8009/UserService / UserService /Users
- Media Type: APPLICATION_JSON
- Method: updateUser ()
- Data:
  - Consumer

```
{
  "type": "consumer",
  "userID": "<userID>",
  "userName": "<userName>",
  "password": "<password>",
  "email": "<email> ",
  "address": "<address>",
  "dob":"<dob>",
  "phone":"<phone>"
}
```

  - Manufacturer

```
{
    "type": "manufacturer",
    "userID": "< manufacturerID>",
    userName": "<userName>",
    "password": "<password>",
    "email": "<email> ",
    "address": "<address>",
    "dob":"<dob>",
    "phone":"<phone>",
    "desc":"<desc>"
}
```

  - Researcher

```
{
    "type" : "researcher",
    "userID" : "< researcherID>",
    "userName" : "<username>",
    "password" : "<password>",
    "email" : "<email> ",
    "address": "<address>",
    "dob":"<dob>",
    "phone":"<phone>",
    "profileInfo":"<profileInfo>"
}
```

- Response: String status message “Updated Successfully” or “Error while updating the user”

DELETE

- Resource: User
- Request: DELETE
  http://localhost:8009/UserService / UserService /Users
- Media Type: APPLICATION_XML
- Method: deleteUser()
- Data: <userData>
  <type>type</type>
  <userID>ID</userID>
  </userData>
- Response: String status message “user deleted successfully” or “Error while deleting the user”

7.1.2 Tools and Technology used

1. **JAX-RS** - JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture. It uses Java programming language annotations to simplify the development of RESTful web services. Jersey which is an open-source framework for developing REST web services supporting JAX-RS API is used in this project implementation. Further, a resource can also be represented in various representation methods with the use of JAX-RS

2. **Apache-Maven** – Maven is used as the dependency management tool. It is a project management tool that is based on Project Object Model which makes the build process easy thus providing a uniform build system. All dependencies and plugins are added with the help of pom.xml file, which is an XML representation of a maven project. Presence of pom.xml helps us keep track of the project configurations and other quality project information. maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

3. **Eclipse IDE** – The project is developed using eclipse as the IDE. it is an integrated development environment for developing projects from various languages. The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. One major reason to choose eclipse as our IDE is the tight integration of Apache maven into the IDE. Eclipse auto formats source code allowing customization and increased readability. Eclipse reports errors automatically to users rather than logging it to the console. Another added advantage in using eclipse is that it can also support debugging and testing program code while also assisting us use code quality tools easily to maintain our projects in an efficient manner.

4. **MySQL workbench** – The backend of the service is implemented using MYSQL workbench since it provides scalability, high performance, and easier management of data. MySQL workbench is an integrated development environment for MYSQL server. It delivers visual tools for creating and executing queries easily. The usage of objects such as stored procedures, functions assist us to implement business logic of our service easily hence improving the performance of the business operations. MySQL also provides connectors and drivers (ODBC, JDBC, etc.) that allow all forms of applications to make use of MySQL as a preferred data management server.

5. **Postman** – Postman is used as the test client of this service. It is an interactive tool used as the test client for the service. It is used because it is more efficient in creating HTTP requests and reading responses, while the work is less tedious.it has the ability to make various types of HTTP requests (GET, POST, PUT, PATCH), saving environments for later use, converting the API to code for various languages (like JavaScript, Python) . Postman is mainly chosen as our test client as it provides a friendly GUI for creating requests and receiving responses. Insertion of API request URL, HTTP verb corresponding to the operations and input data fields needed for program execution is highly sufficient to carry out the testing process hence making our work easy.

6. SonarLint is a free IDE extension that lets you fix coding issues before they exist! It squiggles flaws so that they can be fixed before committing code. Like a spell checker, SonarLint highlights Bugs and Security Vulnerabilities as you write code, with clear remediation guidance so you can fix them before the code is even committed. It is open source, totally free and supports multiple IDE flavors. As we are using Eclipse IDE For the implementation, we can get it directly from the Eclipse Marketplace, and it will then detect new bugs and quality issues as you code

Please refer the Appendix A: User Management Service - Yadushika S.K. IT19162010 for more information on User Management Service diagrams
Appendix A includes

- Appendix A.1: Internal Logic diagrams- Appendix A.1.1: Class diagrams, Appendix A.1.2: Architecture diagrams
- Appendix A.2: Database design of service -ER
- Appendix A.3: Other relevant design diagrams – Appendix A.3.1: Activity Diagrams, Appendix A.3.2: Use Case Diagrams
- Appendix A.4: Testing Results
- Appendix A.5: Postman API test result screenshots

7.2 Campaign Management service - Mariyam M.S.S. IT19175058

7.2.1 API design

The application programming interface of the campaign management service is implemented according to the gathered requirements. The main resource used for the implementation of the API is 'concept.' All create, read, update and delete operations are performed on this resource. The API design is as follows:

POST

- Resource: Concept
- Request: POST
  http://localhost:8083/Campaign_Management_Service/ConceptService/Concepts/insert/<researchername>/<manufacturername>
- Media Type: APPLICATION_FORM_URLENCODED
- Method: insertConcepts()
- Data: conceptName, conceptDesc, startDate, deadline, pledgeGoal, reward, workUpdt, researcherID, manufactID
- Response: String status message "Concept Details Inserted Successfully" or "Error while launching the concept"

GET

- Resource: Concept
- Request: GET
  http://localhost:8083/Campaign_Management_Service/ConceptService/Concepts
- Media Type:
- Method: readMyConcepts()
- Data: researcherID => Passed as path parameter
- Response: An HTML Table

GET

- Resource: Concept
- Request:
  GET
  http://localhost:8083/Campaign_Management_Service/ConceptService/Concepts
- Media Type:
- Method: readAllConcepts()
- Data: {}
- Response: An HTML Table

PUT

- Resource: Concept
- Request:
  PUT
  http://localhost:8083/Campaign_Management_Service/ConceptService/Concepts
- Media Type: APPLICATION_JSON
- Method: updateConcept()
- Data: conceptCode => Passed as path parameter

{ “conceptName”: “<conceptName>”,
“conceptDesc”: “<conceptDesc >”,
“pledgeGoal”: “<pledgeGoal >”,
“reward”: “<reward >”,
“workUpdt”: “<workUpdt >”}

- Response: String status message “Concept Details Updated Successfully” or “Error while updating the concept”

DELETE

- Resource: Concept
- Request : DELETE *http://localhost:8083/Campaign_Management_Service/ConceptService/Concepts*
- Media Type:
- Method: deleteConcept()
- Data: : conceptCode => Passed as path parameter
- Response: String status message “Concept deleted successfully” or “Error while deleting the concept”

POST

- Resource: Pledge
- Request: POST *http://localhost:8083/Campaign_Management_Service/PledgeConcept/Pledges/insertPledge/<consumername>/<conceptName>*
- Media Type: APPLICATION_XML
- Method: insertPledges()
- Data: conceptID,backerID,pledgedAmnt
- <pledgeData>
- <pledgedAmnt>Amount</pledgedAmnt>
- </pledgeData>
- Response: String status message “You pledged the concept Successfully!” or “Failed to pledge the concept!”

7.2.2 Tools and Technology used

1. **JAX-RS**: JAX-RS is a java programming API developed for applications that use REST architecture. It is an open source framework and assists in developing RESTFUL web services. Reason behind the use of JAX-RS is that it simplifies the development of RESTFUL web services, by providing annotations to define resources and the actions performed on them. Further, a resource can also be represented in various representation methods with the use of JAX-RS.

2. **Apache Maven**: Maven is used as the dependency management tool. It makes the build process easy and maintains the build with well defined class paths and libraries. Maven can provide information about the project by using dependency lists and logs. Maven repositories that can be accessed to configure the project dependencies stands as an added advantage. All dependencies and plugins are added with the help of pom.xml file, which is an XML representation of a maven project. This helps to keep track of the project information easily.

3. **Eclipse IDE**: Eclipse is an integrated development environment for developing projects from various languages. The main reason for the use of eclipse is that it provides tight integration of apache maven into the IDE, that ease the launch of maven builds. Eclipse also contains a basic workspace which supports multiple perspectives. It facilitates debugging and testing while providing extensible plugin system too. Therefore, it makes the editing and testing process easy. Eclipse also supports the use of code quality tools that provides a quality and efficiency to the code.

4. **MySQL**: MySQL is used as the database server since it provides scalability, high performance, and easier management of data. Further, it makes it easier to store and retrieve data via simple queries. MySQL workbench is used as the database tool to develop the databases. Workbench is an integrated development environment for MySQL server. It delivers visual tools for creating and executing queries easily.

5. **Postman**: Postman is an interactive tool used as the test client for the service. It is used because it is more efficient in creating HTTP requests and reading responses, while the work is less tedious. The testing process was made easier since the only task to be done, was to select the appropriate HTTP verb, provide the correct URL and data, followed by sending the request and receiving the response. It does not require any complex coding; therefore, it is user friendly and fast.

6. **SonarLint**: SonarLint is a free IDE extension, that is used in the project as a code quality management tool. The reason for using SonarLint is that, it is open source, and can be added as an extension in the IDE easily. Further, it helps to fix coding issues before they exist. It provides instant feedbacks in the IDE while writing the code and also provides a rich description of the issue that makes it easier to understand the issue. Thus, it helps to maintain a consistent and quality source code with ease.

Please refer the Appendix B: Campaign Management Service - Mariyam M.S.S. IT19175058 for more information on more information on Campaign Management Service diagrams
Appendix B includes

- Appendix B.1: Internal Logic diagrams - Appendix B.1.1: Class diagram, Appendix B.1.2: Architecture diagram
- Appendix B.2: Database design of service - ER
- Appendix B.3: Other relevant design diagrams - Appendix B.3.1: Activity Diagram, Appendix B.3.2: Use Case Diagram
- Appendix B.4: Testing Results
- Appendix B.5: Postman API test result screenshots

7.3 Manufacturer and Patent services – Vithyashagar S.T. IT19058474

7.3.1 API design – Manufacturer Service

The API of the Manufacturer Service management service is implemented in accord with the requirements gathered. The resource used under this service is 'Services' provided by Manufacturer. All Create, Read, Update, and Delete operations are performed on this resource with the usage of HTTP verbs GET, PUT, POST and DELETE.

GET (To get all the Manufacturer services available)

- Resource: ManufacturerSevices
- Request: GET
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/
- Media Type: TEXT_HTML
- Method: readServices()
- Data: {}
- Response: An HTML Table or *"Error while reading the Services"*

GET (To get a specific Manufacturer's services available by providing the manufacturer ID)

- Resource: ManufacturerSevices
- Request: GET
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/Manufacturer/{ID}
- Media Type: TEXT_HTML
- Method: readSpecificManufacturerServices()
- Data: ManufacturerID
- Response: An HTML Table or *"Error while reading the Services"*

GET (To get a specific Manufacturer's services available by username that logged in)

- Resource: ManufacturerSevices
- Request: GET
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/Manufacturer/{ID}
- Media Type: TEXT_HTML
- Method: getSpecificManufacturerServices()
- Data: ManufacturerID
- Response: An HTML Table or *"Error while reading the Services"*

GET (To get a specific Manufacturer's ID, by providing the username that logged in)

- Resource: ManufacturerSevices
- Request: GET
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/ManufacturerID/{Uname}
- Media Type: TEXT_HTML
- Method: getManufacturerID()
- Data: UserName
- Response: An HTML Table or *"Error while reading the Services"*

POST (To Insert a specific Manufacturer's services)

- Resource: ManufacturerSevices
- Request: POST
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/insert/{uname}
- Media Type: APPLICATION_FORM_URLENCODED
- Method: getSpecificManufacturerServices()
- Data: UserName, Name, Specialty, Description
- Response: String status message *"Service Created Successfully"* or *"Error while reading the Services"*

PUT

- Resource: ManufacturerSevices
- Request: PUT
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/
- Media Type: APPLICATION_JSON
- Method: updateService()
- Data: {"SID", "ServiceID", "Name", "Speciality", "Description"}
- Response: String status message " *Updated successfully "* or "*Error while updating the Service*"

DELETE

- Resource: ManufacturerSevices
- Request: DELETE
  http://localhost:8080/ManufacturerService/ManufacturerService/Service/
- Media Type: APPLICATION_JSON
- Method: deleteService()
- Data: {"SID"}
- Response: String status message "*Service Deleted"* or "*Error while Deleting Service*"

7.3.2 API design – Patent Service

GET (To get all the Patent forms available)

- Resource: PatentSevices
- Request: GET
  http://localhost:8080/PatentService/PatentService/Patent/
- Media Type: TEXT_HTML
- Method: readPatents()
- Data: {}
- Response: An HTML Table or *"Error while reading the Patent Forms"*

GET (To get a concept's Description by providing the conceptName)

- Resource: PatentSevices
- Request: GET
  http://localhost:8080/PatentService/PatentService/Patent/ConceptDesc/{concept}
  Media Type: APPLICATION_XML
- Method: getConceptDescription()
- Data: conceptName
- Response: XML value or *"Error while reading the Description"*

GET (To get a concept's ID by providing the conceptName)

- Resource: PatentSevices
- Request: GET
  http://localhost:8080/PatentService/PatentService/Patent/ConceptID/{conceptName}
- Media Type: APPLICATION_XML
- Method: getSpecificManufacturerServices()
- Data: conceptName
- Response: XML value or *"Error while reading the ID"*

GET (To get a specific Researcher's ID, by providing the username that logged in)

- Resource : PatentSevices
- Request :
  GET
  http://localhost:8080/PatentService/PatentService/Patent/Reasearcher/{Uname}
- Media Type: APPLICATION_XML
- Method: getManufacturerID()
- Data: UserName
- Response: An HTML Table or *"Error while reading the Address"*

GET (To get all the Patent forms available for a specific logged user)

- Resource: PatentSevices
- Request:
  GET
  http://localhost:8080/PatentService/PatentService/Patent/{name}
- Media Type: TEXT_HTML
- Method: readSPatents()
- Data: UserName
- Response: An HTML Table or *"Error while reading the Patent Forms"*

POST (To Insert a Patent Application form)

- Resource: PatentSevices
- Request:
  POST *http://localhost:8080/PatentService/PatentService/Patent/{uName}*
- Media Type: APPLICATION_FORM_URLENCODED
- Method: insertPatent ()
- Data: UserName, Title, applied date
- Response: String status message "*Patent Form Created Successfully"* or "Error while Inserting Patent*"*

DELETE (To delete a Patent Application Form**)**

- Resource: PatentSevices
- Request:
  DELETE
  http://localhost:8080/PatentService/PatentService/Patent/
- Media Type: APPLICATION_JSON
- Method: deletePatent()
- Data: {"PID"}
- Response: String status message "*Patent Application Deleted"* or "*Error while Deleting Patent*"

7.3.3 Tools and Technology used

1. **JAX-RS** (API Development): Though there are many API Development technologies, I chose JAX-RS because it is nothing more than a specification, a set of interfaces and annotations which makes the API creation easier

2. **Apache-Maven** (Project Management tool): The main reason for using maven is, its streamlined, XML-based configuration model enables to rapidly describe or grasp the outlines of Java-based project. Also, maven supports test-driven development, long-term project maintenance, declarative configuration and wide range of plugins make the development process smooth.

3. **Eclipse IDE** (IDE) : I have used Eclipse IDE to develop the project since it is an extensible plug-in system for customizing the environment which makes the development easier.

4. **MySQL workbench** - DB: Since MySQL provides connectors and drivers (ODBC, JDBC, etc.) that allows us to connect all forms of applications, makes it as the perfect choice as my DB Server.

5. **Postman** (Test Client) - Because of its friendly GUI for creating requests and receiving responses I have used postman as my test client. Also, it manages collection of request and responses effortlessly as it does not need any complex coding.

6. **SonarLint** (Code Quality) - Since SonarLint is an IDE extension that helps to detect and fix quality issues as we code, it is way easier to use. Also, it works like a spell checker, so we can fix flaws before committing code.

Please refer the Appendix C: Manufacturer and Patent services – Vithyashagar S.T. IT19058474 for more information on more information on Manufacturer and Patent Management Service diagrams
Appendix A includes

- Appendix C.1: Internal Logic diagrams – Manufacturer Service - Appendix C.1.1: Class Diagram, Appendix C.1.2: Architecture Diagram
- Appendix C.2: Database design of service – ER – Manufacturer Service
- Appendix C.3: Other relevant design diagrams – Manufacturer Service - Appendix C.3.1: Activity Diagram, Appendix C.3.2: Use Case Diagram
- Appendix C.4: Internal Logic diagrams – Patent Service - Appendix C.4.1: Class diagram, Appendix C.4.2: Architecture diagram
- Appendix C.5: Database design of service – ER –Patent Service
- Appendix C.6: Other relevant design diagrams – Patent Service - Appendix C.6.1: Activity Diagram, Appendix C.6.2: Use Case Diagram
- Appendix C.7: Testing Results
- Appendix C.8: Postman API test result screenshots

7.4 Product Management service - Hillary J.R. IT19021430

7.4.1 API design

The API (Application Programming Interface) of the campaign management service is implemented according to the gathered requirements and the scope set by the team. The main resource used for the implementation of the API is 'product.' All create, read, update and delete operations are performed on this resource. The API design is as follows:

POST

- Resource: Product
- Request: POST
  http://localhost:8005/ProductService/ProductService/Product
- Media Type: APPLICATION_FORM_URLENCODED
- Method: insertProduct ()
- Data: productName, productPrice, productDesc, productCat, productQty
- Response: String status message "Product details Inserted successfully" or "Error while inserting the product"

POST

- Resource: Buys
- Request: POST

http://localhost:8001/ProductService/ProductService/Buying/insertConsumerProductBuys/yashwin

- Media Type: APPLICATION_FORM_URLENCODED
- Method: insertProductConsumer ()
- Data: consumerCode,productCode,qty
  Username – passed as path param
- Response: String status message "Purchase Recorded Successfully" or "Error while retrieving values from database"

GET

- Resource: Product
- Request: GET
  http://localhost:8005/ProductService/ProductService/Product
- Media Type:
- Method: readProduct ()
- Data: {}
- Response: An HTML Table or "Error while reading the products"

PUT

- Resource: Product
- Request: PUT
  http://localhost:8005/ProductService/ProductService/Product
- Media Type: APPLICATION_JSON
- Method: updateProduct ()
- Data: { "productId": "<productId >",
  "productCode": "<productCode >",
  "productPrice": "<productPrice >",
  "productDesc": "<productDesc >",

“productCat”: “<productCat >”,
“productQty”: “<productQty >”
}

- Response: String status message “Product details Updated successfully” or “Error while updating the product”

DELETE

- Resource: Concept
- Request: DELETE
  http://localhost:8005/ProductService/ProductService/Product
- Media Type: APPLICATION_XML
- Method: deleteProduct ()
- Data: <productData>
  <productId>ID</productId>
  </productData>
- Response: String status message “Product Deleted successfully” or “Error while deleting the product.”

7.4.2 Tools and Technology used

1. **JAX-RS**: I have chosen JAX-RS since Jakarta RESTful Web Services, (JAX-RS) formerly Java API for RESTful Web Services) is a Jakarta EE API specification that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern. Jersey, the reference implementation of JAX-RS, implements support for the annotations defined in JSR 311, making it easy for developers to build RESTful web services by using the Java programming language.

2. **Apache Maven**: Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects. Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM) Maven can add all the dependencies required for the project automatically by reading pom file. Maven makes easy to start project in different environments and one doesn't need to handle the dependencies injection, builds, processing, etc. Since One can easily build their project to jar, war etc. This tool was chosen.

3. **Eclipse IDE**: The Eclipse IDE was chosen Since Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse also contains a basic workspace which supports multiple perspectives. The Eclipse IDE provides support for the Maven build. Another advantage is that since we used GIT Version Controlling GIT integration is very effective. You can easily manage repositories and connect them to projects, and the project integration into GIT is virtually seamless through Eclipse IDE. The simplified IDE makes it easy to write clean and efficient code and while providing different views and perspectives it is also very easy to debug code therefore the chosen development environment was eclipse IDE

4. **MySQL**: MySQL is a relational database management system based on SQL. The application is used for a wide range of purposes. MySQL provides connectors and

drivers ODBC, JDBC which makes it easy to communicate with the methods implemented and platform-independence of Java. MySQL was chosen as the database server since it provides some advantages such as Data security, On Demand Scalability, High performance, etc. MySQL workbench and PhpMyAdmin was used as the database tool to develop the databases.

5. **Postman**: The chosen test client for the user service was Postman since Postman is an API (application programming interface) development Testing Tool which helps to build, test and modify APIs. It has the ability to make various types of HTTP requests (GET, POST, PUT, PATCH), saving environments for later use, converting the API to code for various languages. It does not require any complex coding; therefore, it is user easy to use and fast.

6. **SonarLint:** Sonar Lint was the chose code quality achieving tool used in the user service. SonarLint is a free IDE extension that lets you fix coding issues before they exist Like a spell checker, SonarLint highlights Bugs and Security Vulnerabilities as you write code, with clear remediation guidance so you can fix them before the code is even committed. Configuration of the tool was also easily since it was just a plugin to be added from eclipse workspace.

Please refer the Appendix D: Product Management service - Hillary J.R. IT19021430 for more information on Product Management Service diagrams
Appendix D includes

- Appendix D.1: Internal Logic diagrams - Appendix D.1.1: Class diagram, Appendix D.1.2: Architecture diagram
- Appendix D.2: Database design of service - ER
- Appendix D.3: Other relevant design diagrams - Appendix D.3.1: Activity Diagram, Appendix D.3.2: Use Case Diagram
- Appendix D.4 Testing Results
- Appendix D.5: Postman API test result screenshots

7.5 Payment Management service - Kovishwakarunya K. IT19080840

7.5.1 API design

The application programming interface of the payment management service is implemented in accordance with the business requirements. The resource manipulated under this service is 'payment'. All create, read, update, and delete operations are performed on this resource with the aid of HTTP verbs GET, PUT , POST and DELETE.

POST

- Resource: Payments
- Request: POST
  http://localhost:8080/PaymentService/PaymentResource/Payments/backerinsert/<ConceptName>
- Media Type: APPLICATION_FORM_URLENCODED
- Method : insertBackerPayment ()
- Data : PaymentType , Bank , PaymentDate,CardNumber,NameOnCard , CVV, cardExpiryMonth , cardExpiryYear, consumerId , conceptId.
- Response: String status message "Backer payment Inserted Successfully Your payment ID is <PaymentID>" or "Error while inserting user payment"

POST

- Resource: Payments
- Request: POST
  http://localhost:8080/PaymentService/PaymentResource/Payments/buyerinsert/test/<ProductName>
- Media Type: APPLICATION_FORM_URLENCODED
- Method insertBuyerPayment ()
- Data: PaymentType , Bank , PaymentDate,CardNumber,NameOnCard , CVV, cardExpiryMonth , cardExpiryYear, consumerId , ProductId .
- Response: String status message "Buyer payment Inserted Successfully Your payment ID is <PaymentID>" or "Error while inserting user payment"

GET

- Resource: Payments
- Request: GET
  http://localhost:8080/PaymentService/PaymentResource/Payments
- Media Type:
- Method: readPayments()
- Data: {}
- Response: An HTML Table or "Error while reading the payments"

GET

- Resource: Payments
- Request: GET
  http://localhost:8080/PaymentService/PaymentResource/Payments/specificUser/<NameOncard>
- Media Type:
- Method: readSpecificUserPayments ()
- Data: Name
- Response: An HTML Table or "Error while reading the payments

PUT

- Resource: Payments
- Request: PUT
  http://localhost:8080/PaymentService/PaymentResource/Payments/paymentDetailUpdate/<PaymentCode>
- Media Type: APPLICATION_JSON
- Method: updatePaymentDetails()
- Data: PaymentCode -> Path Parameter

  {
  "paymentType": "<paymentType>",
  "bank": "<bank>",
  "cardNo": "<cardNo>",
  "NameOnCard": "<NameOnCard>",
  "cvv": "<cvv>",
  "cardExpMonth": "<cardExpMonth>",
  "cardExpYear": "<cardExpYear>"
  }
- Response: String status message "Payment Details Updated Successfully" or "Error while Updating the payment details"

PUT

- Resource: Payments
- Request: PUT
  http://localhost:8080/PaymentService/PaymentResource/Payments/updateStatus/<Concept Name>
- Media Type: APPLICATION_FORM_URLENCODED
- Method: updatePaymentStatus()
- Data: ConceptID
- Response: String status message "Concept payment status Updated Successfully" or "Error while updating the concept payment details"

DELETE

- Resource: Payments
- Request: DELETE
  http://localhost:6944/PaymentService/PaymentResource/Payments
- Media Type: APPLICATION_XML
- Method: deletePayment()
- Data: <paymentData>

  <status>Declined</status>

  </ paymentData >
- Response: String status message "Payment deleted successfully" or "Error while deleting payment"

7.5.2 Tools and Technology used

1. **JAX-RS** - JAX-RS jersey implementation is chosen as the API for service development. It is a java programming language API specification that assists in creating RESTful web services adhering to Representational state transfer architectural pattern. Jersey which is an open-source framework for developing REST web services supporting JAX-RS API is used in this project implementation.
   The reason for using this API is that it provides immense support to implement services and the underlying logic of it with the use of various annotations to define service resources and operations performed on them where each resource can be represented in various representation methods too.

2. **Apache-Maven** – Maven is a project management tool that is based on Project Object Model. The main reason for using maven is its flexible nature in functioning as a dependency management tool which makes the build process easy thus providing a uniform build system. The ability to access maven repositories to configure our project dependencies is an advantage which can reduce the difficulties in handling them manually. Presence of pom.xml helps us keep track of the project configurations and other quality project information.

3. **Eclipse IDE** – The project is developed using eclipse as the IDE. It is the basic integrated development environment used to implement our project. One major reason to choose eclipse as our IDE is the tight integration of Apache maven into the IDE. Launching maven builds within eclipse is easy and maven dependencies are resolved within eclipse workspace which assists installing dependencies without installing to local maven repositories. Eclipse contains a base workspace in which the user could work in many perspectives and also contains an extensible plugin system to dynamically create the environment we require for our project. Another added advantage in using eclipse is that it can also support debugging and testing program code while also assisting us use code quality tools easily to maintain our projects in an efficient manner.

4. **MySQL workbench** – The backend of the service is implemented using MYSQL workbench. MySQL workbench is an integrated development environment for MYSQL server. MySQL is used as the relational database in order to access and manage data of our project ensuring operations of our project to be quick and query efficient. The usage of objects such as stored procedures, functions assist us to implement business logic of our service easily hence improving the performance of the business operations.

5. **Postman** – Postman is used as the test client of this service. It is an interactive tool used as the test client to verify the API of our project. Postman is mainly chosen as our test client as it provides a friendly GUI for creating requests and receiving responses. It works on the backend of our system ensuring that the system works as intended. Using this tool to manage collection of request and responses is very easy as it does not need any complex coding. Insertion of API request URL, HTTP verb corresponding to the operations and input data fields needed for program execution is highly sufficient to carry out the testing process hence making our work easy.

6. **Sonar Lint** – Sonar lint is used as the code quality tool in the service. It is basically an IDE extension used to detect and fix issues in the code allowing a errorless quality code to result. This tool was chosen mainly due to the flexible use of it where this tool can be easily added up into the IDE which is being used for the service reducing any unnecessary hassle.

Please refer the Appendix E: Payment Management service - Kovishwakarunya K. IT19080840 for more information on Payment Management Service
Appendix E includes

- Appendix E.1: Internal Logic diagrams - Appendix E.1.1: Class diagram, Appendix E.1.2: Architecture diagram
- Appendix E.2: Database design of service – ER
- Appendix E.3: Other relevant design diagrams - Appendix E.3.1: Activity Diagram, Appendix E.3.2: Use Case Diagram
- Appendix E.4 Testing Results
- Appendix E.5: Postman API test result screenshots

8.0 System integration details

The basic communication among services were mainly implemented by considering the HTTP request response pull communication mechanism. As the initial step, we analyzed the workflow of the system to understand which services require communication among each other, identifying the server and client services. Jersey client API, which is a high-level java technology API, was used to implement the client which helps to create client services for any HTTP based RESTful web service. Each service that stands as a client and server carried out the inter service communication process and it was tested using the postman tool, evaluating whether the server provides the correct response relevant to the client request. As a development team, the collaboration was maintained using GitHub ensuring proper version control among the services.

9.0 Appendix

Appendix 1: Other relevant diagrams – Class diagram, Use Case diagram, Flow chart diagram

A.1 Class Diagram

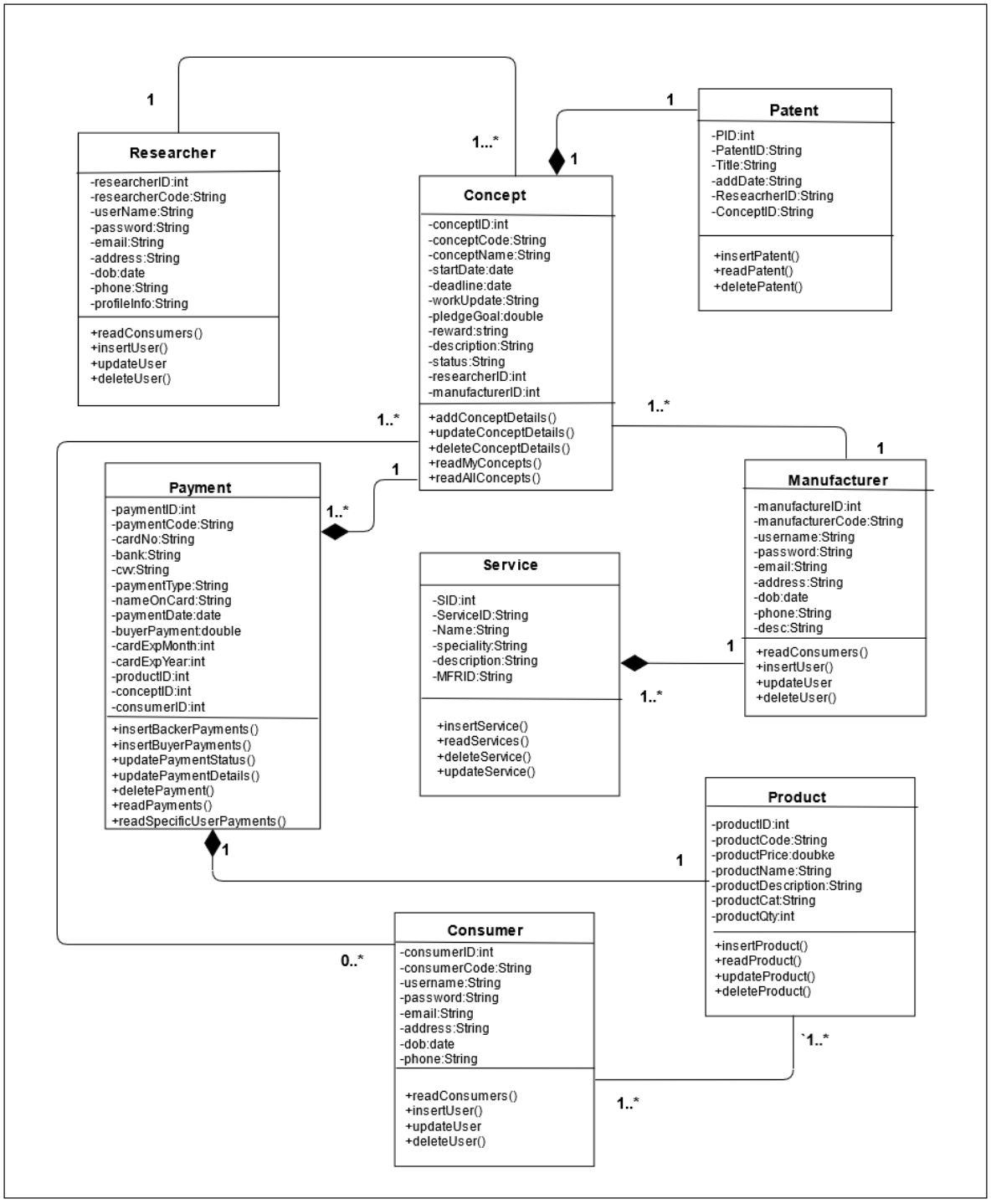


Figure 6: System Overall Class Diagram

The above diagram illustrates the attributes and operations of the classes belonging to the services hence providing an overall idea of the constraints imposed on the system. The services user, concepts and products are associated with one another whereas payments and patents participate as a composite relationship to concept, product and service respectively. The composite relationships indicated imply that payment cannot exist without a concept or product transaction and patent cannot exist without a service. A concept and product can be pledged and bought by many consumers whereas one consumer can pledge or buy many concepts and products.

A.2 Use Case Diagram

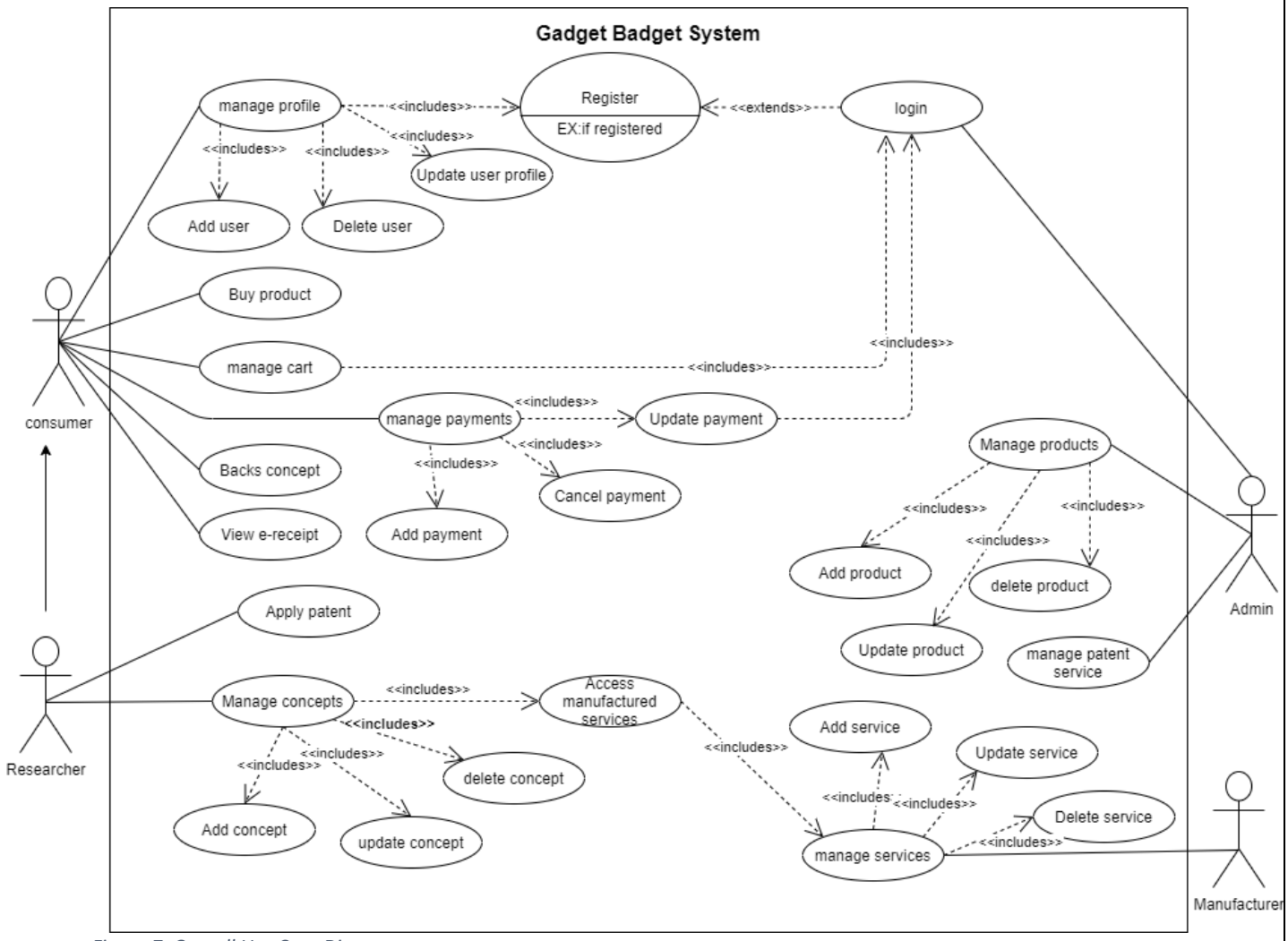


Figure 7: Overall Use Case Diagram

- The actors of the system include, a researcher, consumer, admin, and a manufacturer.
- All actors must register and login to perform any action.
- The researcher can perform actions like managing concepts which include the addition, view, update, and deletion of concepts. Also, the researcher can apply for patents and request the service of a manufacturer.
- Consumers can back a project or buy a product. In addition, the consumers may manage their profile by updating or deleting the profile.
- Admin is the actor in charge of managing any products sold through the system. They can add, view, update or delete products.
- Manufacturer provides a service to the researcher and thus, they are the actor in charge of managing services. Manage services include addition, update, and deletion of products.
- A consumer can manage payments for any backing or products bought by them.

A.3 Overall flow chart Diagram

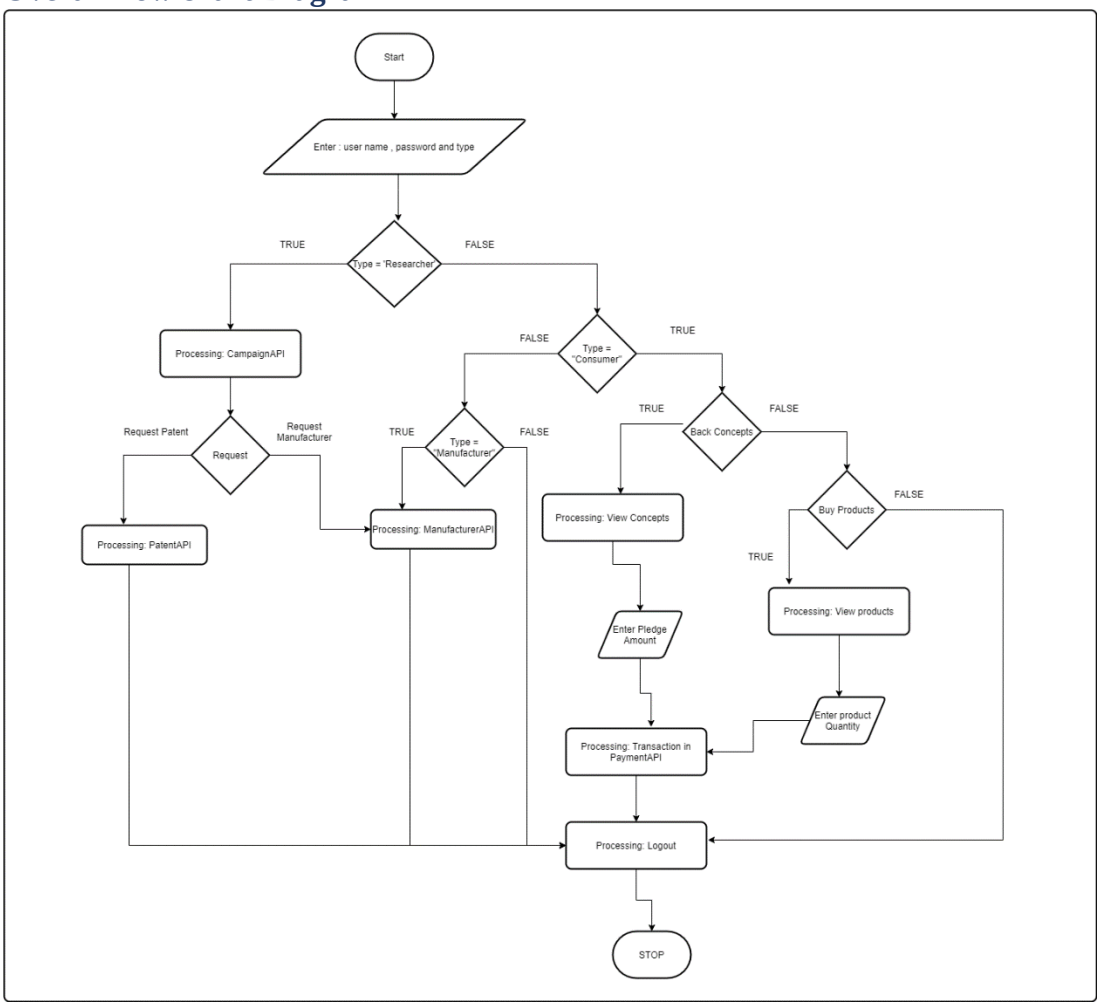


Figure 8: Overall Flow Chart Diagram

Please refer Appendix F: Overall flow chart Diagram

- The process starts with a user logging in by providing the username, password and the user type.
- If the logged user is a researcher, they can process operations in the campaign API. While processing the researcher can request a patent or a manufacturer service based on their requirement. Request of a patent will lead to the access of patent API and the request of a manufacturer service will lead to the manufacturer API.
- If the logged user is a consumer, they can perform two operations. If preferred the consumer can back a concept or buy a product. Each of these operations will lead to the payment API.
- If the logged user is a manufacturer, they can process operations in the manufacturer API.
- After completing the operations required, all users can log out of the system, which will mark the end of the process.

Appendix A: User Management Service - Yadushika S.K. IT19162010

Appendix A.1: Internal Logic diagrams

Appendix A.1.1: Class diagrams

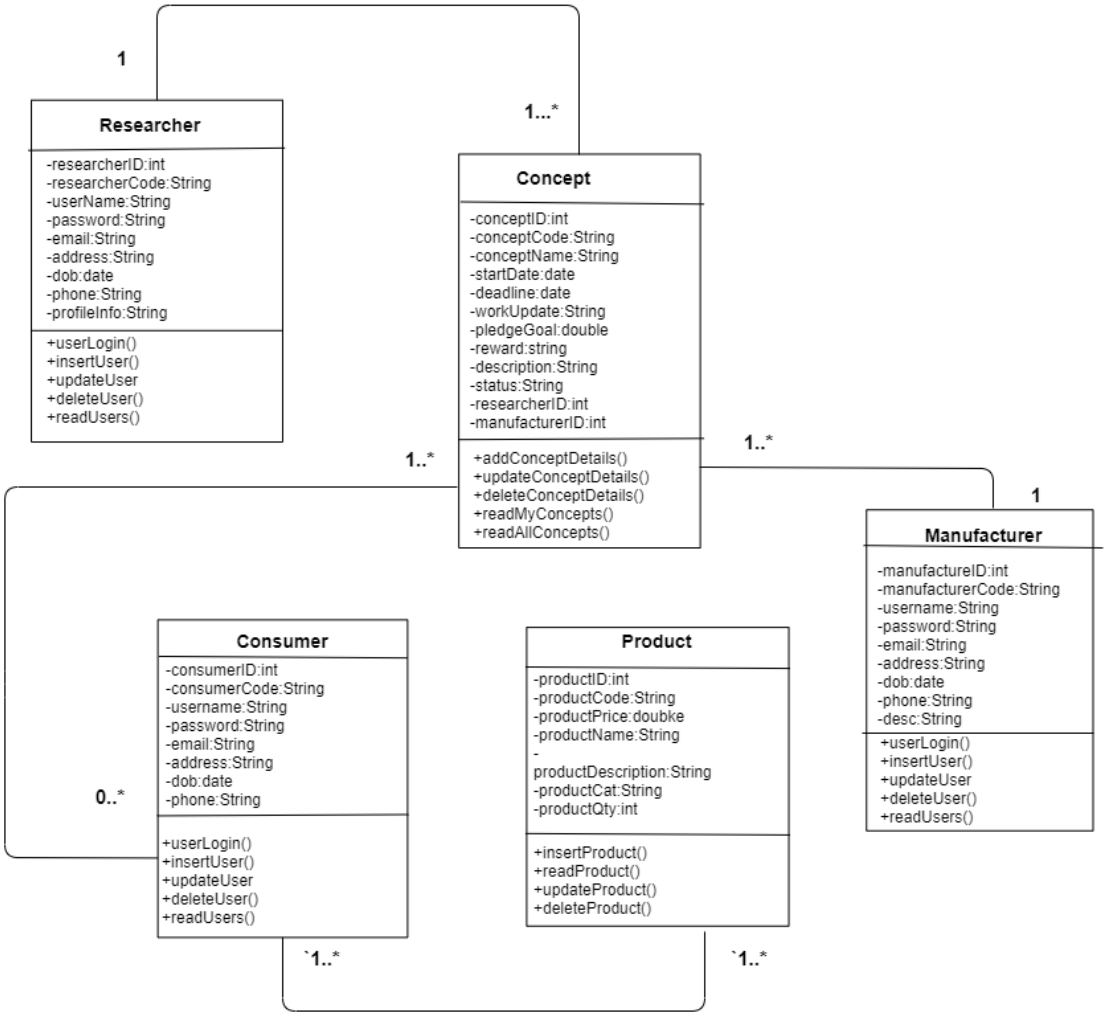


Figure 9 Class Diagram

User service is implemented to manage three types of users (Consumers, Manufacturers, Researchers) of the system. Consumers can buy many products and back many concepts. Manufacturer can manufact many concepts which are launched by researchers. Researcher can launch many concepts but particular concepts can be launched by one researcher. Concept should be manufactured by single manufacturer. Concept can be backed by many consumers and product can be bought by many consumers.

Appendix A.1.2: Architecture diagrams

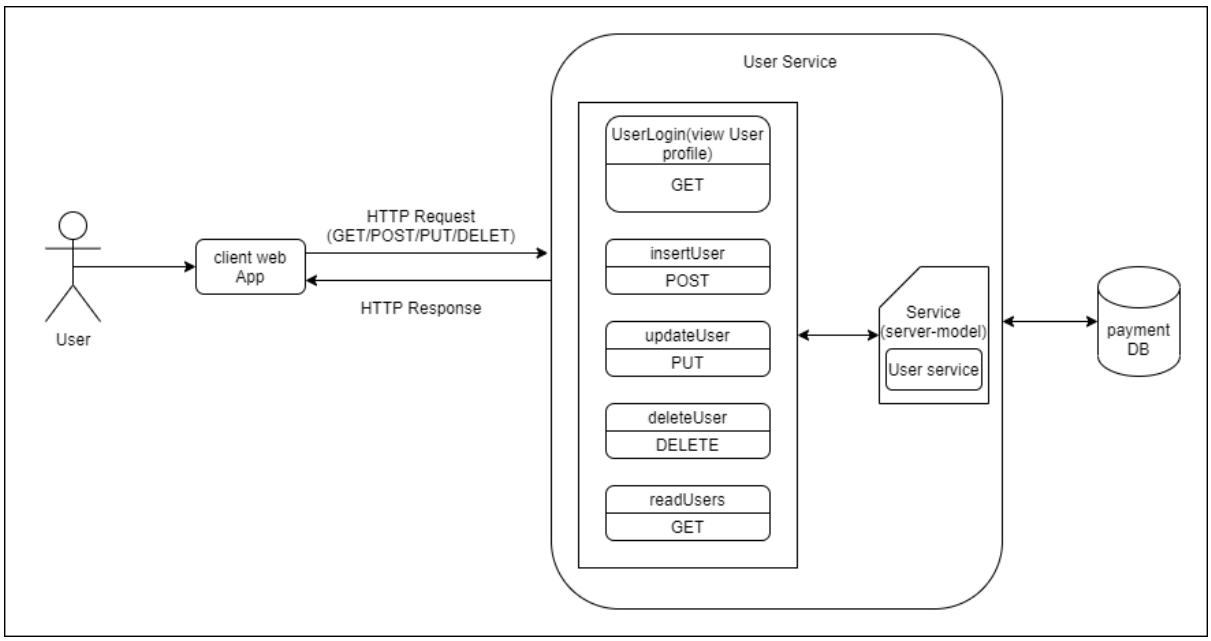


Figure 10 Architecture Diagram

The User management service is implemented as resources and service packages. The User service operations are implemented in the SERVICE section which serves as a server model to communicate with the database which stores user data. The main resource manipulated using the HTTP verbs in this service is “User”. The user sends an HTTP request to perform operations like insert, view, delete and update of users according to the user type via the client APP. These requests are process by the server model and response is sent back to the client APP as HTTP response.

Appendix A.2: Database design of service -ER

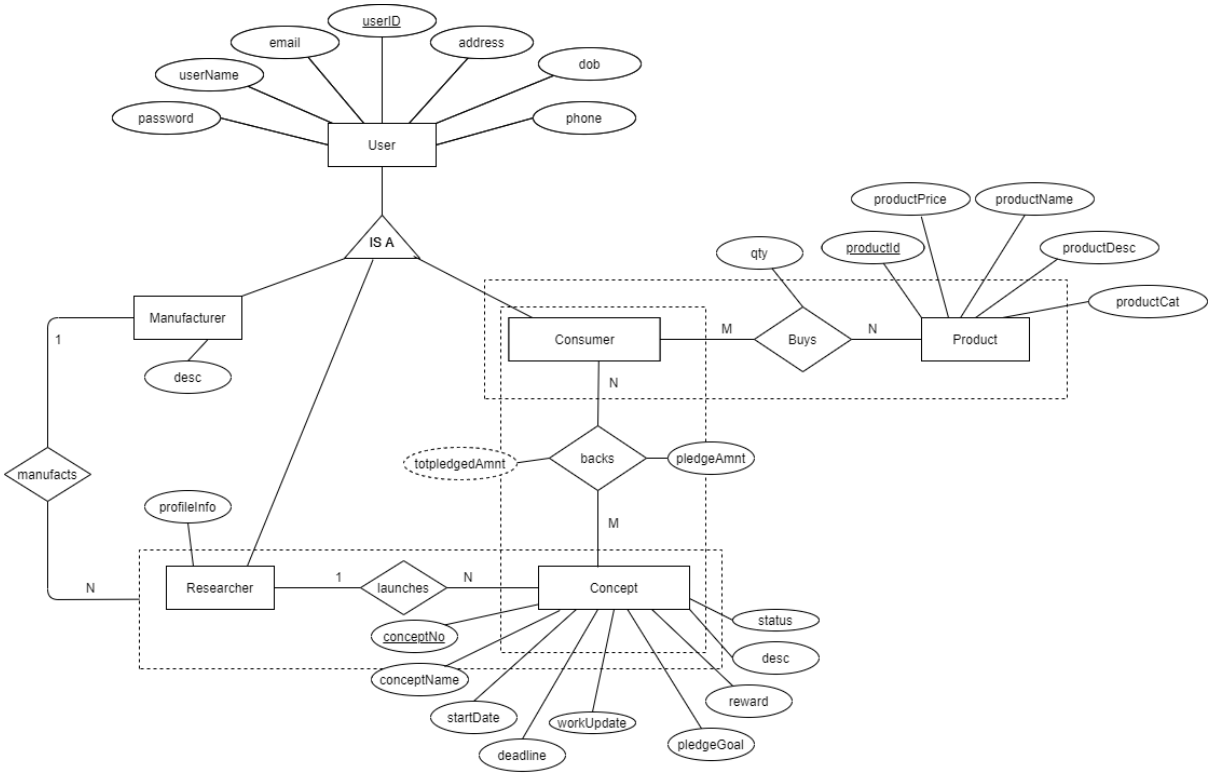


Figure 11 ER Diagram

Users are classified as Consumer, Manufacturer and Researcher. consumer backs concept and buys products. A backer can back a concept added by a researcher by providing a pledge amount. once a concept is completed and is launched a buyer can buy the projects. Manufacturer manufacts the concepts which are launched by researchers. User service keep tracks of all the details of the user and manage them.

Appendix A.3: Other relevant design diagrams

Appendix A.3.1: Activity Diagrams

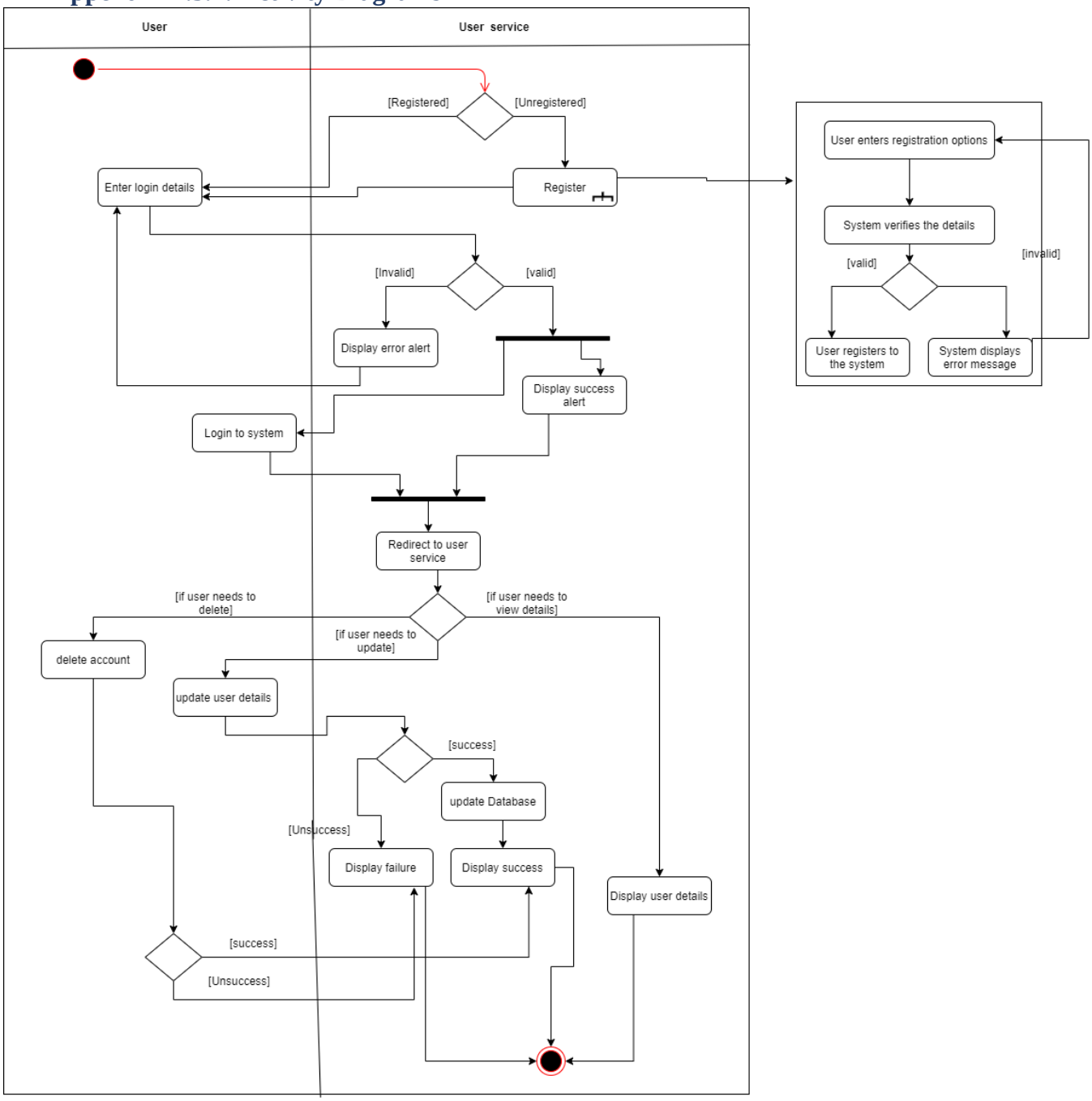


Figure 12Activity Diagram

Initially users need to register themselves. Users should have separate user accounts according to the user type to full fill different tasks. According to the user type (consumer, manufacturer, researcher) they need to insert valid details, and if the registration is success, they can login to the system by entering login credentials. If the user is already registered, they can directly login to the service. Once they redirected to the service, they can view their details. If they wish they can update their details. If the updated details ah valid database will be updated and success alert is sent to the user. If the details are invalid, user will be getting error message. User can delete their account and they will be getting success or failure messages.

Appendix A.3.2: Use Case Diagrams

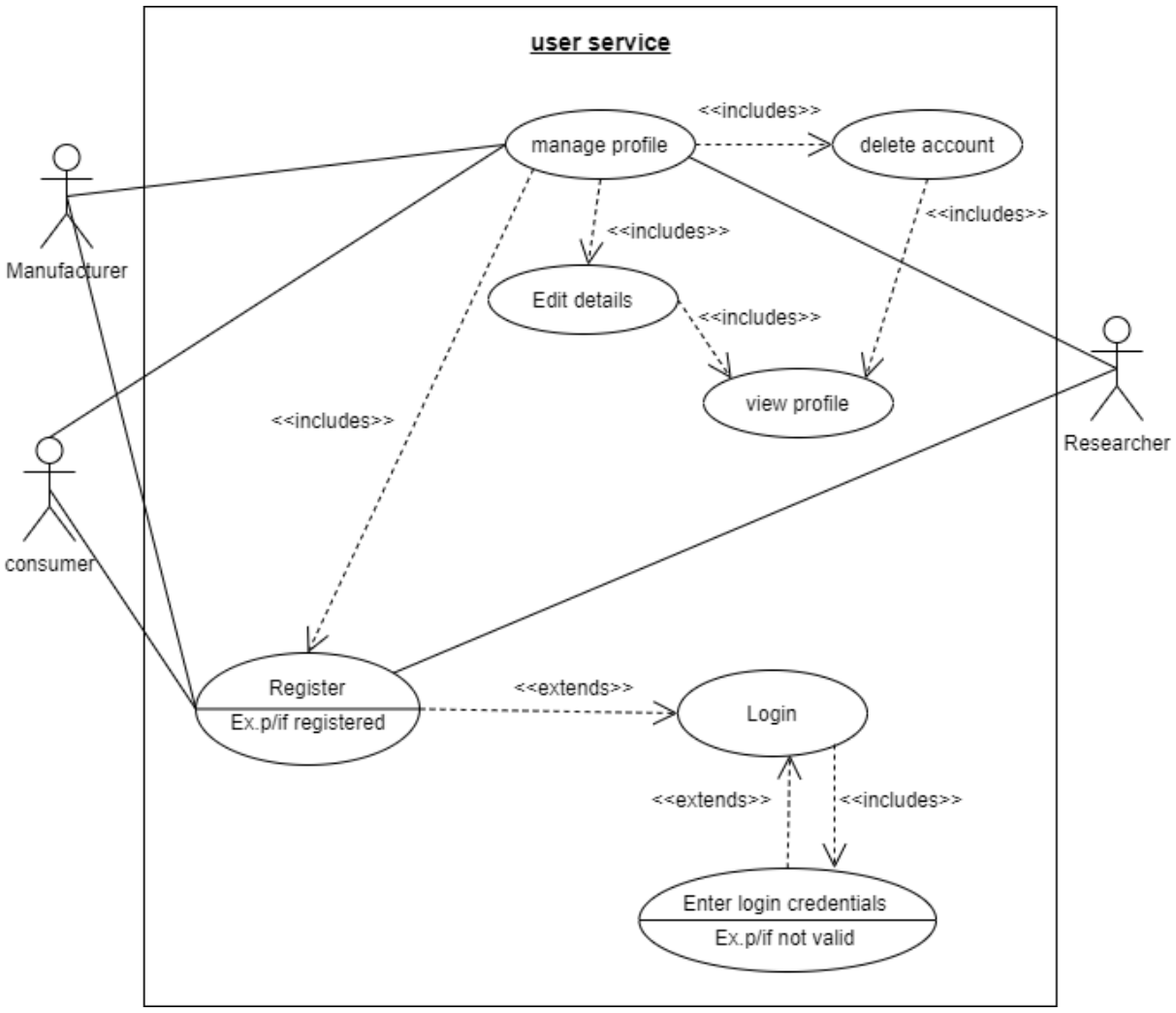


Figure 13 Use case Diagram

The actors involved in the user service are consumer, manufacturer, and researcher.
The above diagram illustrates the various functions performed by the actors involving in the user service. All three actors can do same actions. The users deal with the user service as front-end users. Users can register themselves to be a member of the service, as well as they can view update and delete the details.

Appendix A.4: Testing Results

Test case ID	Description	Test input	Expected Output	Test output	Test status
1	Insert null values for username field	userName = “”	“user name or password cannot be null”	“user name or password cannot be null”	Pass
2	Update manufacturer table, phone number should have 10 values only.	phone= “0094772534678”	“Error while updating the user”	“Error while updating the user”	Pass
3	Insert invalid email ID	Email = “abc”	“Invalid email”	“Invalid email”	Pass
4	Update password column in manufacturer table.	password = “abc123”	“updated successfully”	“updated successfully”	Pass
5	Delete manufacturer user account by giving userID which is not in the Database	manufacturerID = 26	“Error while deleting the user”	“Error while deleting the user”	Pass
6	Delete researcher account by giving valid ID.	manufacturerID = 2	“Researcher deleted successfully”	“Researcher deleted successfully”	Pass
7	Login to the account by passing valid userName, password and type as URL parameters	userName = “krish” password = “krish123” type = “consumer”	“Login successful You’re logged as krish” and display user details of “krish”	“Login successful You’re logged as krish” and display user details of “krish”	Pass
8	Read all users of manufacturer table.	Passing user type = “manufacturer” as URL parameter.	Display manufacturer table	Display manufacturer table	Pass
9	Update Consumer details	Input existing user id details to be updated	“updated successfully”	“updated successfully”	Pass

Appendix A.5: Postman API test result screenshots

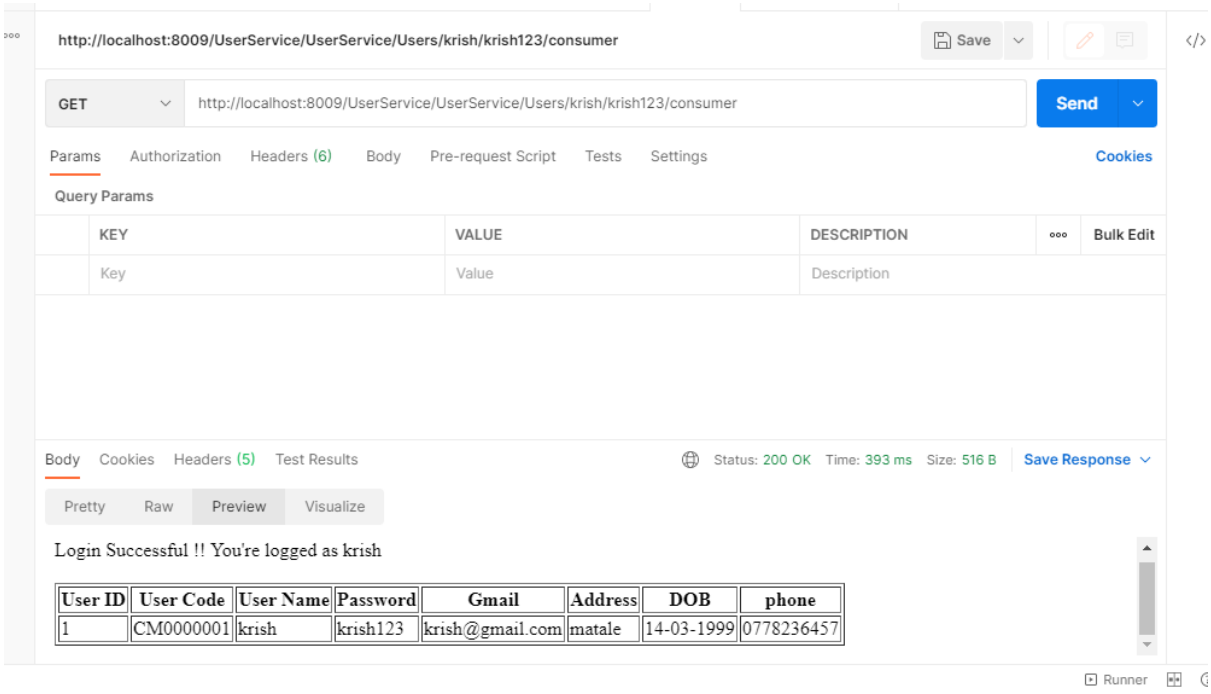


Figure 14: Login to service and view user profile Test Result

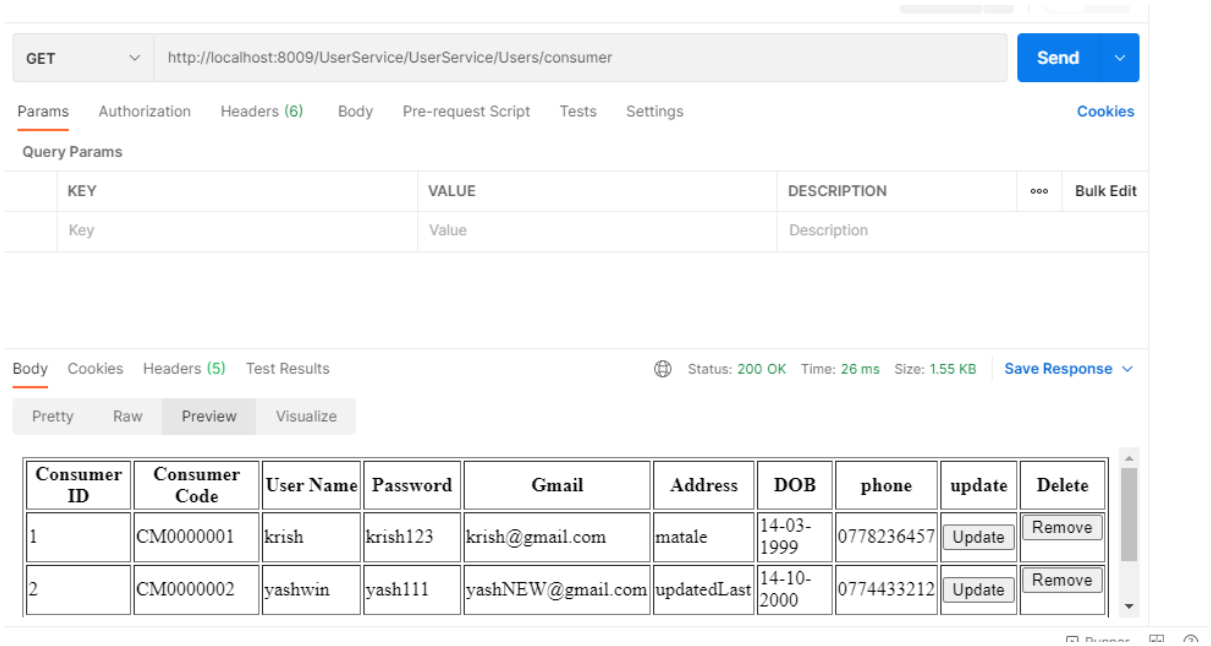


Figure 15: Read all consumer details Test Result

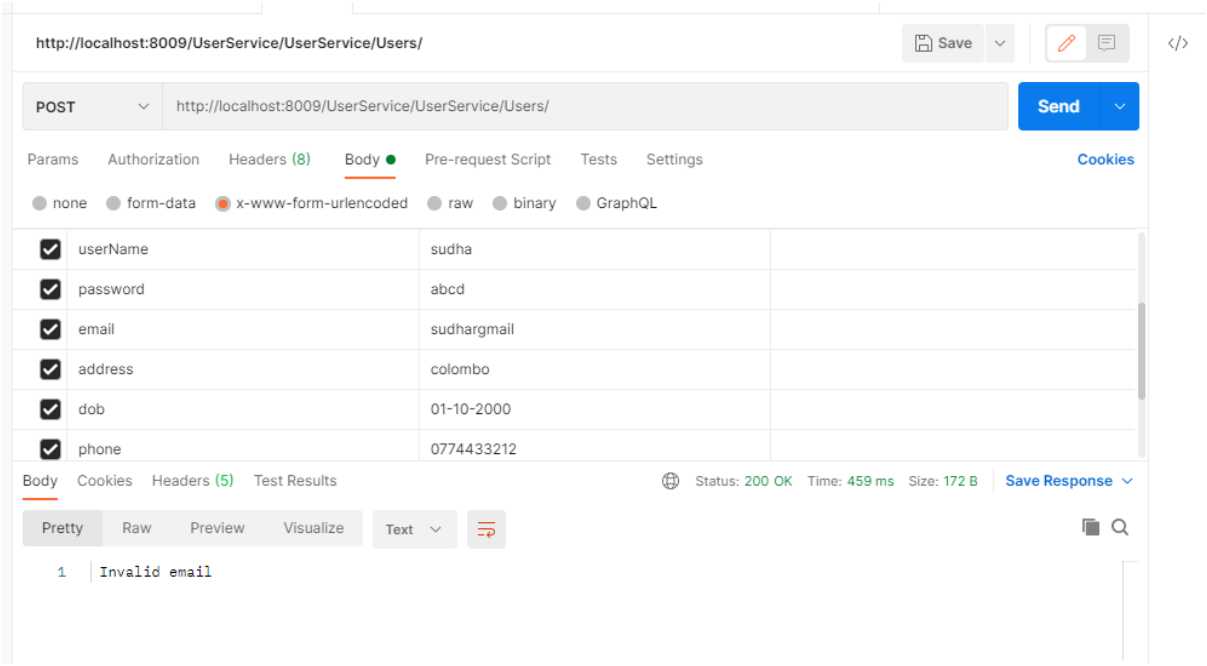


Figure 16: email validation for the insertion Test Result

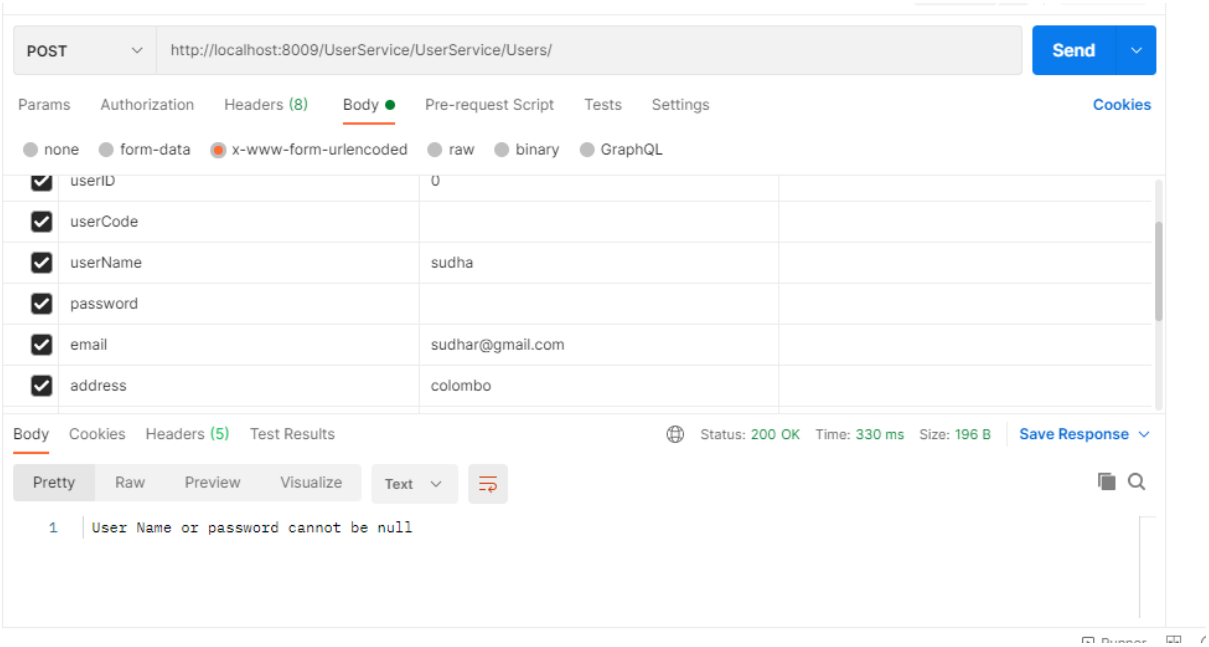


Figure 17: Username or password cannot be null during the insertion Test Result

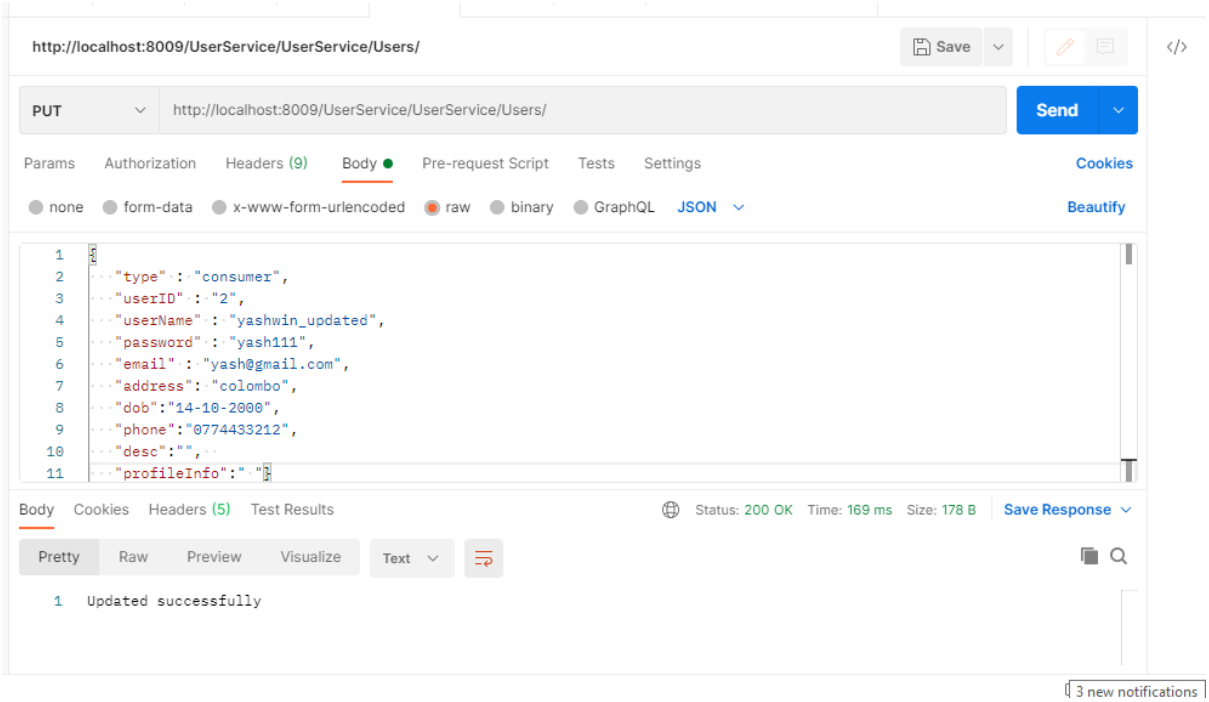


Figure 18: Update user details Test Result

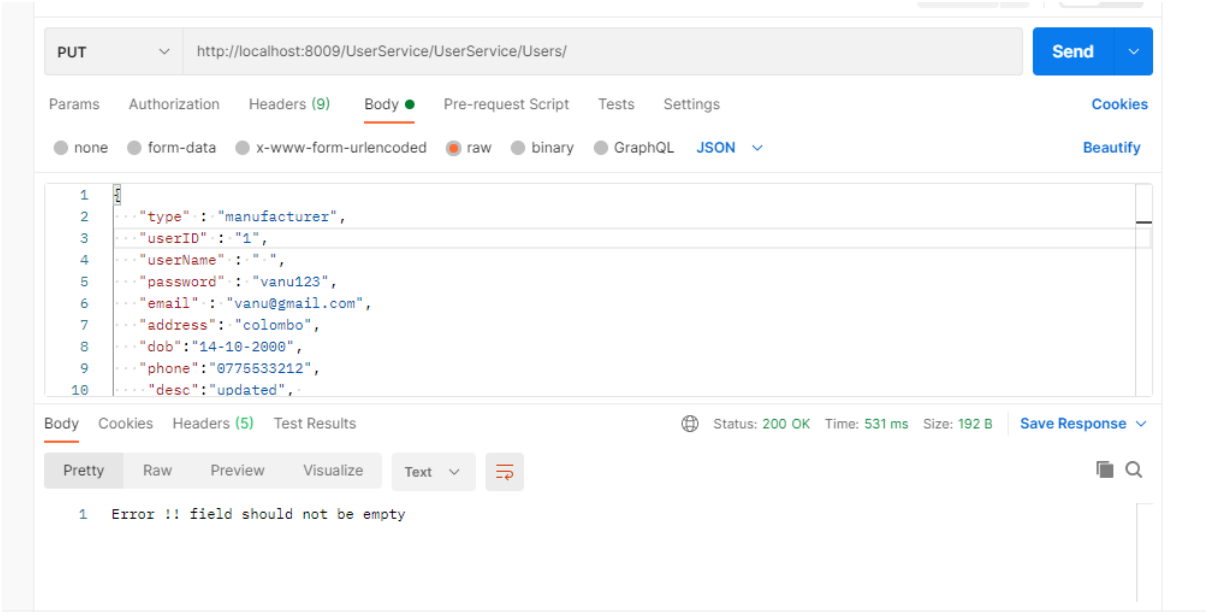


Figure 19: Null values are not accepted during the update Test Result

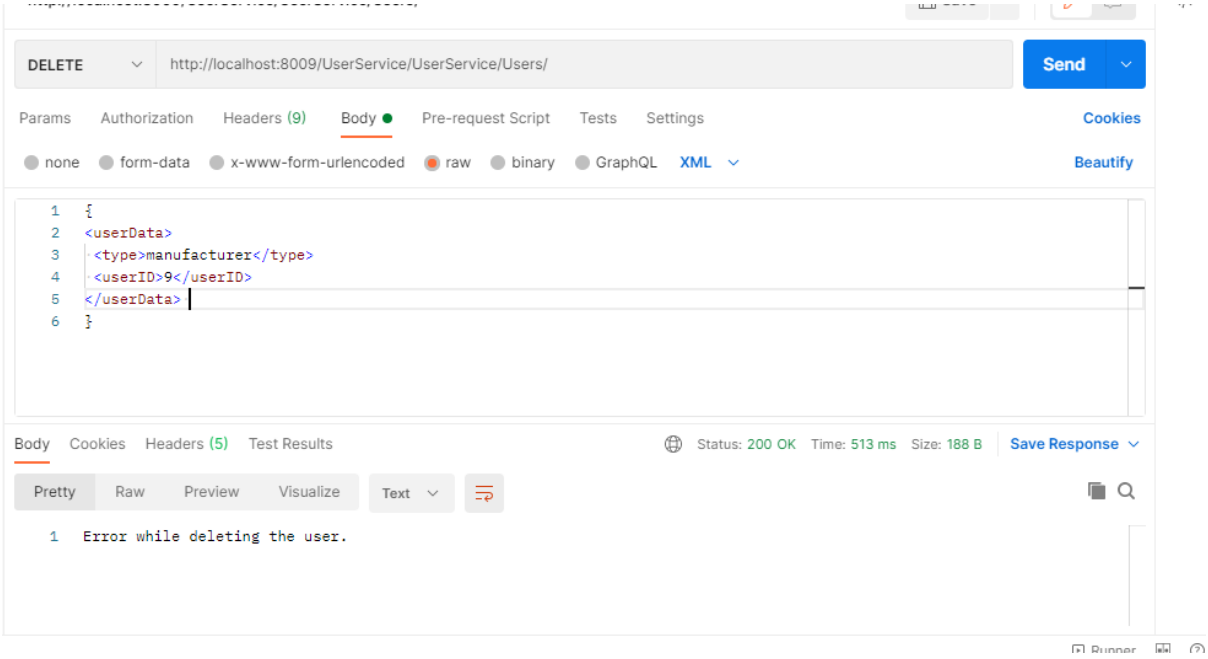


Figure 20: Invalid user ID for the deletion Test Result

Appendix B: Campaign Management Service - Mariyam M.S.S. IT19175058

Appendix B.1: Internal Logic diagrams

Appendix B.1.1: Class diagram

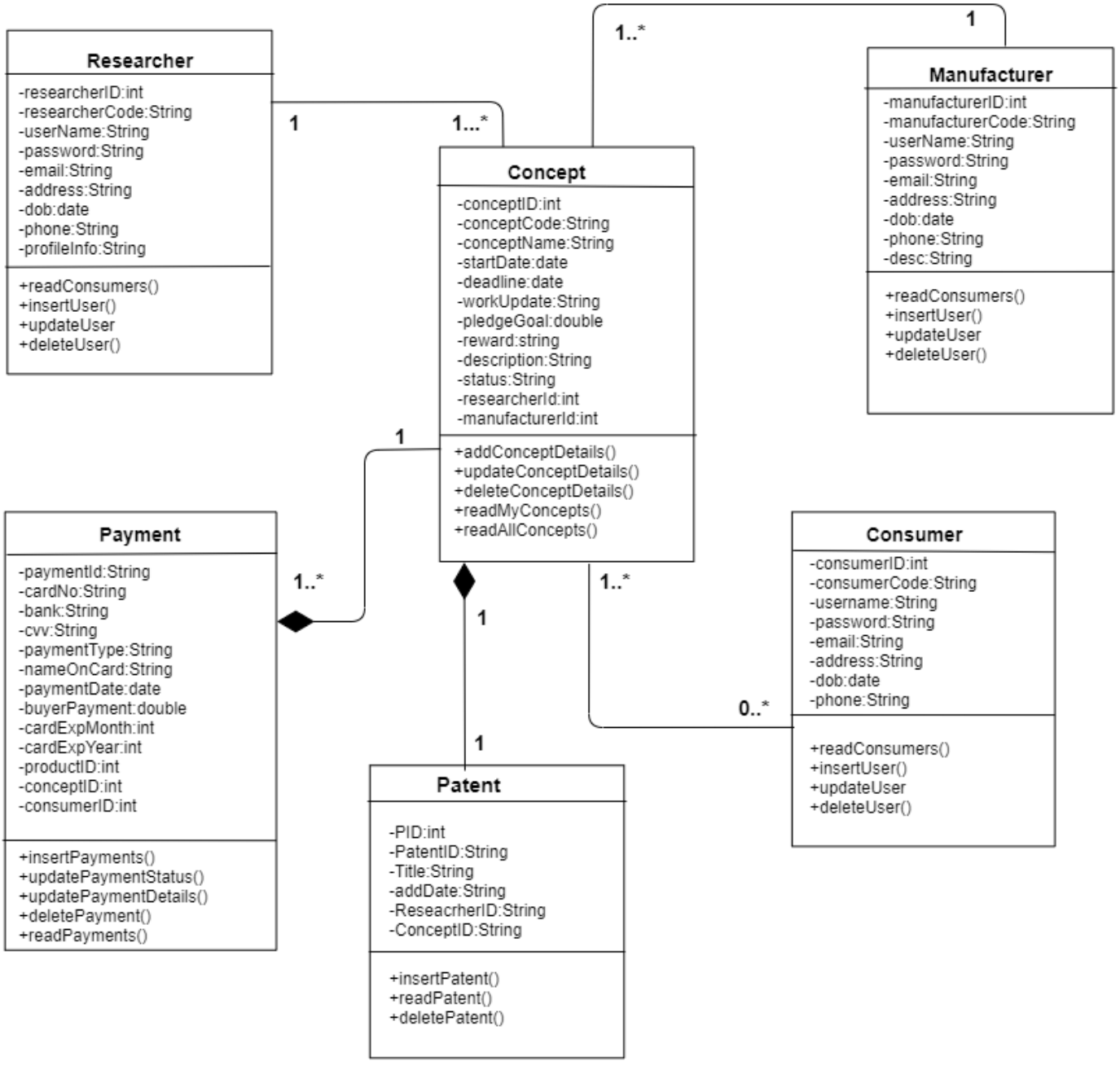


Figure 21 Class Diagram

- The main class associated with campaign management service is concept.
- Each concept has an associated researcher since these concepts are added and maintained by the researcher. Therefore, there is an association relationship between the researcher and the concept.

- Similar each concept can be backed by many consumers. Consumers back a concept in order to fund for the concept. Thus, an association can be observed between a concept and a consumer too.
- For any backed concept, there will be a payment. Payment is considered as a part of concept because it is necessary only if a concept is backed by a consumer. Further payment cannot exist unless a concept is backed. Therefore, payment and concept share a composition relationship.
- A concept can be transformed into a product by the researcher himself or get the help of a manufacturer. Therefore, concept and manufacturer maintain an association.
- Patent can be requested by a researcher for a concept if they prefer. Therefore, patent is a part of concept, which will exist only if the concept exists. It is illustrated as composition in the class diagram.

Appendix B.1.2: Architecture diagram

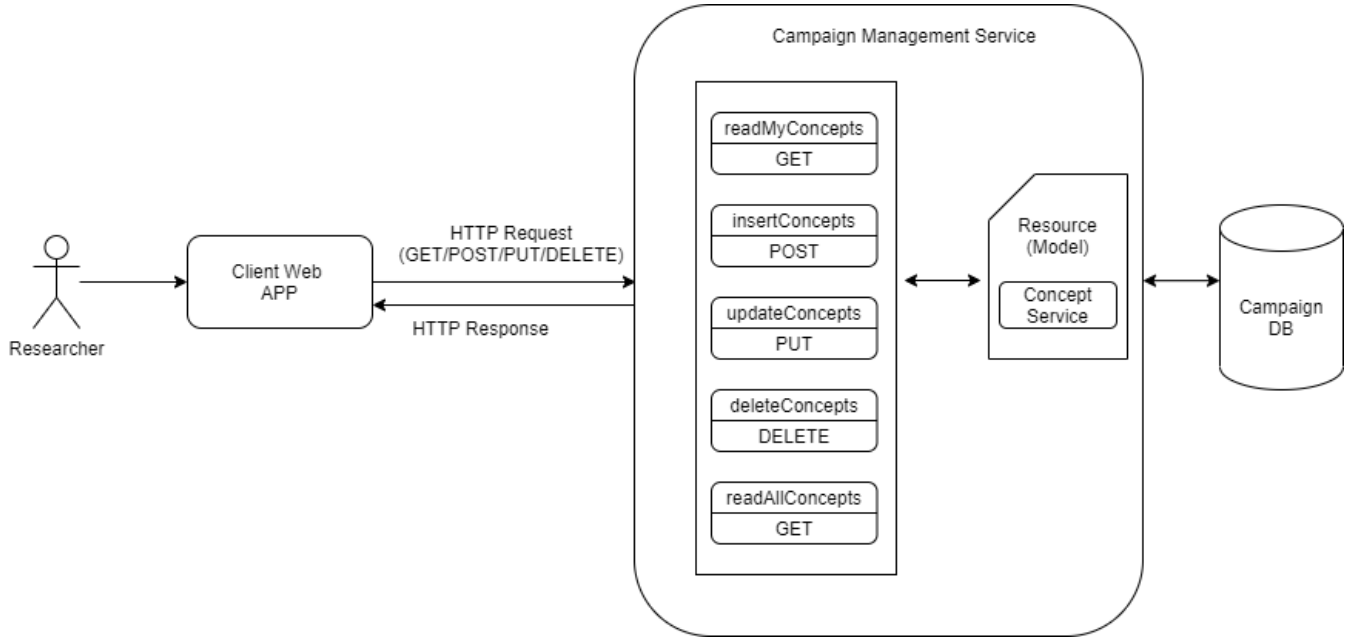


Figure 22 Architecture Diagram

- The campaign management service is implemented as resources and service packages.
- The CRUD operations are implemented under the resources which is the server model.
- It serves as the medium for communication with the database where all details about concepts are stored.
- The main resource that is responsible for the service is concepts.
- A researcher sends an HTTP request to perform operations like insert, view, delete and update of concepts via the client APP.
- These requests are process by the server model and response is sent back to the client APP as HTTP response.

Appendix B.2: Database design of service - ER

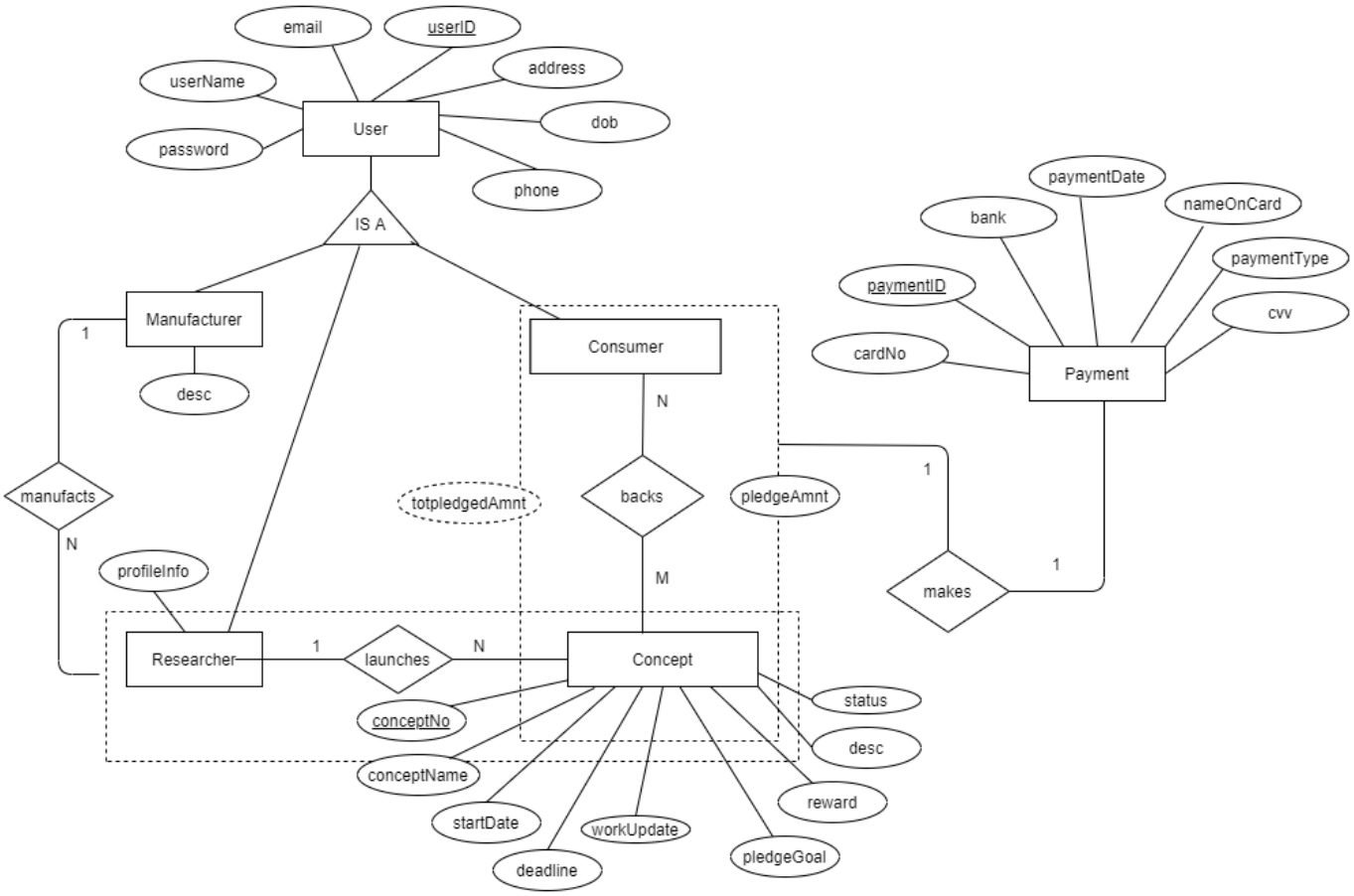


Figure 23 ER Diagram

- The main entity associated with the campaign management service is the concept.
- These concepts are launched by a researcher, by providing a concept name, description, start date, deadline, pledge goal and a reward.
- A consumer can back a concept by providing a pledge amount. The total pledged amount for a project is derived by calculating the sum of pledged amounts. Therefore, the backing transaction is also tracked by the campaign service.
- The consumers make payment for each concept that are backed by them. This is then tracked by the payment service.
- A concept that is launched by a researcher can be manufactured by a manufacturer if
- required.

Appendix B.3: Other relevant design diagrams

Appendix B.3.1: Activity Diagram

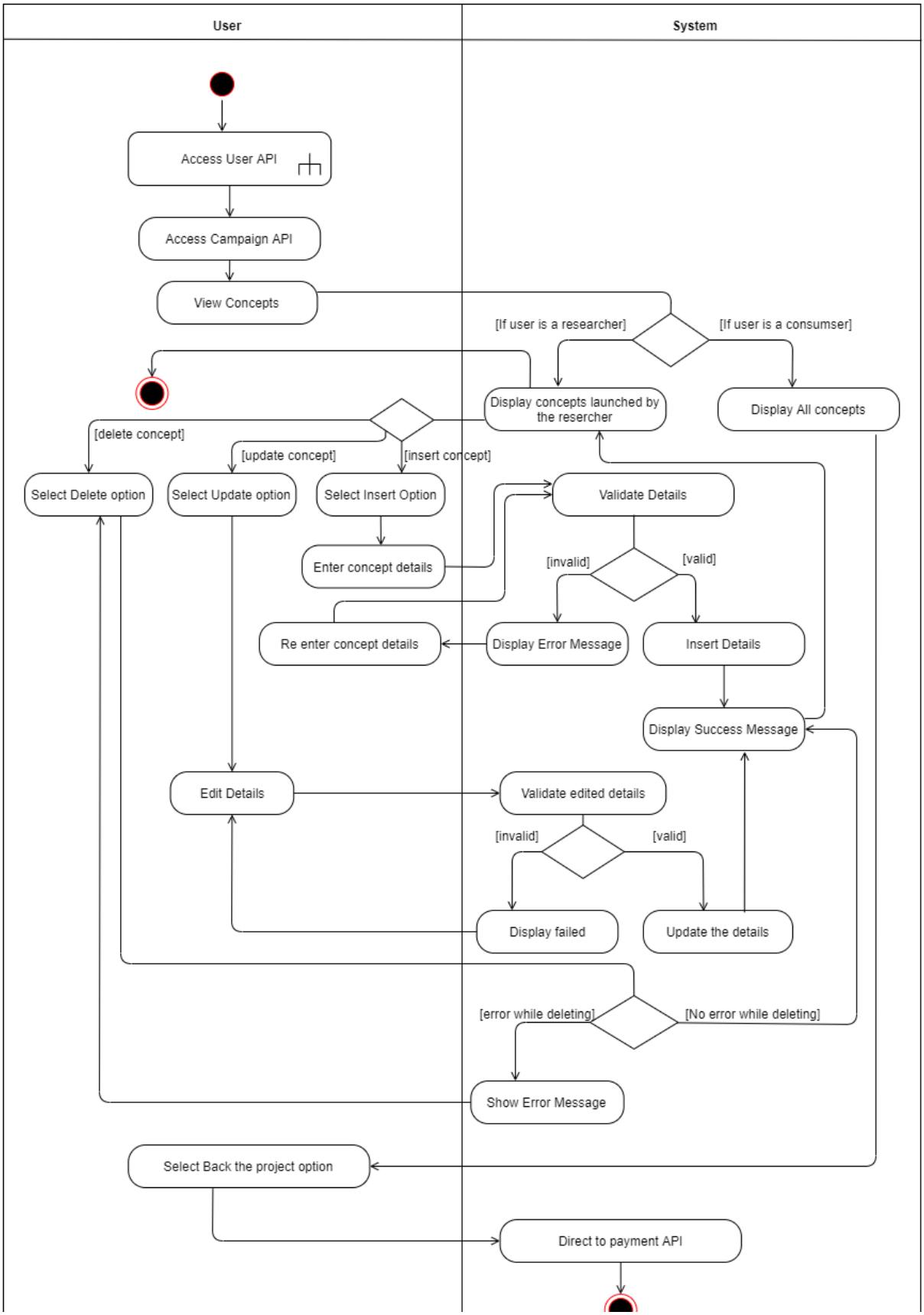


Figure 24 Activity Diagram

- The campaign API can be accessed by two types of users after logging in via the user API.
- The researcher can manage the projects/concepts, while a consumer can fund the project by backing the available concepts.
- On access by a researcher, the system will display the concepts that were already launched by the researcher.
- Then the researcher can select among three options, insert, update and delete.
- On select of the 'insert' option, the researcher must enter the concept details and continue. The system will then validate the details. If the provided information is valid, the system will store the concept information in the database and prompt a success message. If the provided information is invalid the system will display an error message and the researcher can re-enter the information.
- On select of the 'update' option, the researcher can update the necessary details. The system will validate these details too. If the provided new information is valid the system will update the database and provide a success response message. If any invalid information is provided, the system will display an error message and the researcher can re-edit the details.
- If the selected option is 'delete' the system will delete the records of the particular concept. If the system could not delete the record due to a particular reason, the system will prompt an error message. On contrast if the record was deleted successfully, a success message will be displayed.
- The system will display the concept details again after all three operations mentioned above.
- Moreover, if the current user is a consumer, the system will display all the available concept details in the system. The consumer can view them and back a project if they wish to help funding the project. On backing a concept, the consumer will be directed to the payment API, which will mark the end of the campaign management service.

Appendix B.3.2: Use Case Diagram

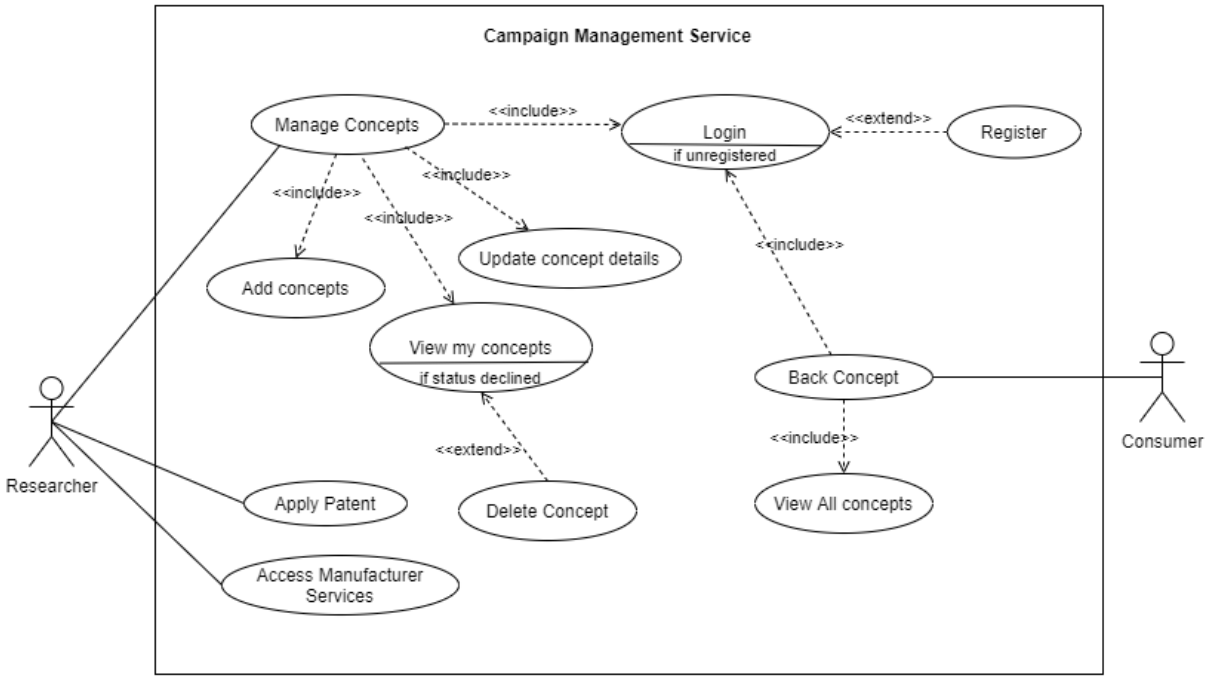


Figure 25Use case Diagram

- The actors involved in the campaign management service are researchers and consumers.
- The researcher can manage his concepts, apply a patent for a concept or request the service of a manufacturer for the completion of his product. Manage concepts include operations such as insert, view, delete and update of a concept as represented in the use case above.
- The consumer can support for a project by backing the project. For this purpose, the consumer can initially view all the concepts and fund a concept which he/she prefers.
- All the actions of a consumer and a researcher requires a valid login. If they are not registered to the system, they must register and then log in to perform any activity.

Appendix B.4: Testing Results

Test Case ID	Description	Test input(s)	Expected Output(s)	Actual Output(s)	Test results
1	Any required field cannot be null for insert	conceptName = " "	Display message "Error while launching the concept"	Displayed the message "Error while launching the concept"	PASS
2	Pledge goal should be a number during insert/update	pledgeGoal = "abc"	Display message "Error while launching the concept"	Displayed the message "Error while launching the concept"	PASS
3	Researcher ID should be a number for insert	researcherID = "abd1"	Display message "Error while launching the concept"	Displayed the message "Error while launching the concept"	PASS
4	Update concept details	"conceptID": "3", "conceptName": "cooler", "conceptDesc": "An attachable wall cooler", "startDate" : "2021-05-05", "deadline" : "2022-01-23", "reward" : "A free cooler", "workUpdt" : "To be started"	Display message "Concept details updated Successfully!"	Displays the message "Concept details updated Successfully!"	PASS
5	Update a record that has a status 'completed'	"conceptID": "8" (status completed)	Display message "Concept cannot be updated"	Displays message "Concept cannot be updated"	PASS
6	Update a record with an ID not present in the database	"conceptID": "15", "conceptName": "cooler", "conceptDesc": "An attachable wall cooler", "startDate" : "2021-05-05", "deadline" : "2022-01-23", "reward" : "A free cooler", "workUpdt" : "To be started"	Display message "Error while updating the concept"	Displays the message "Error while updating the concept"	PASS
7	Delete a record that has a status 'completed'	<conceptData> <conceptID>8</conceptID> </conceptData> (Status that is completed)	Display message "Concept cannot be deleted"	Displays message "Concept cannot be deleted"	PASS

8	Delete a record with an ID not present in the database	<conceptData> <conceptID>18</conceptID> </conceptData>	Display message “Error while deleting the concept”	Displays the message “Error while deleting the concept”	PASS
9	Delete a record with the correct concept id	<conceptData> <conceptID>5</conceptID> </conceptData>	Display message “Concept details deleted sucessfully”	Displays the message “Concept details deleted sucessfully”	PASS
10	Display the correct sum of pledged amount for a concept on view of the concept details	“conceptID”: “2” “backerID”: “5” “pledgedAmnt”: “5000” “conceptID”: “2” “backerID”: “10” “pledgedAmnt”: “4000”	Show pledged amount as 9000 for concept with ID 2 in the view	Shows pledged amount as 9000 for concept with ID 2 in the view	PASS

Appendix B.5: Postman API test result screenshots

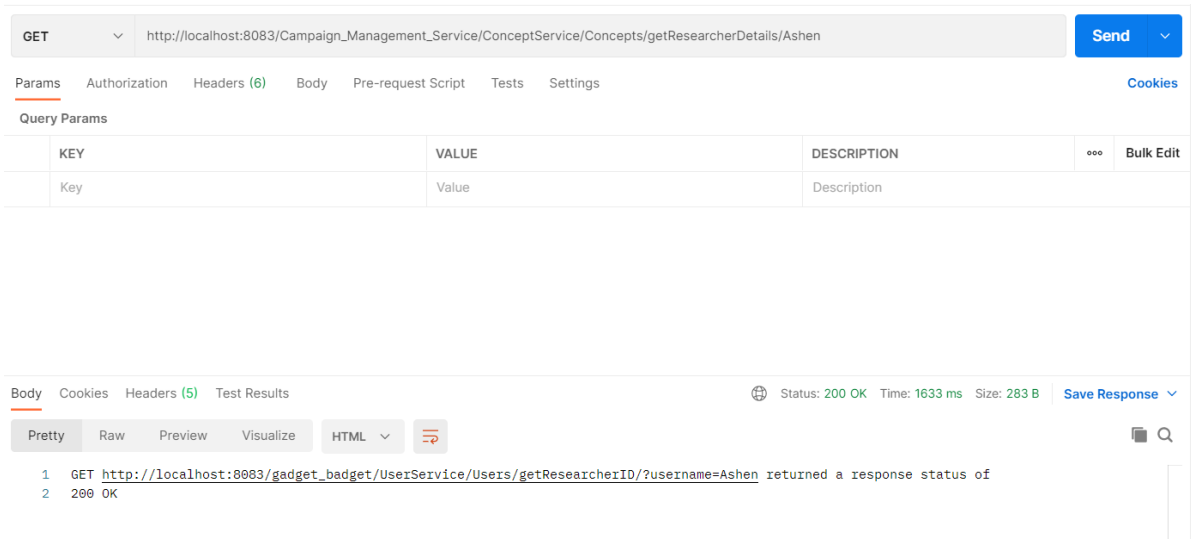


Figure 26: Check Inter Service Communication Test Result

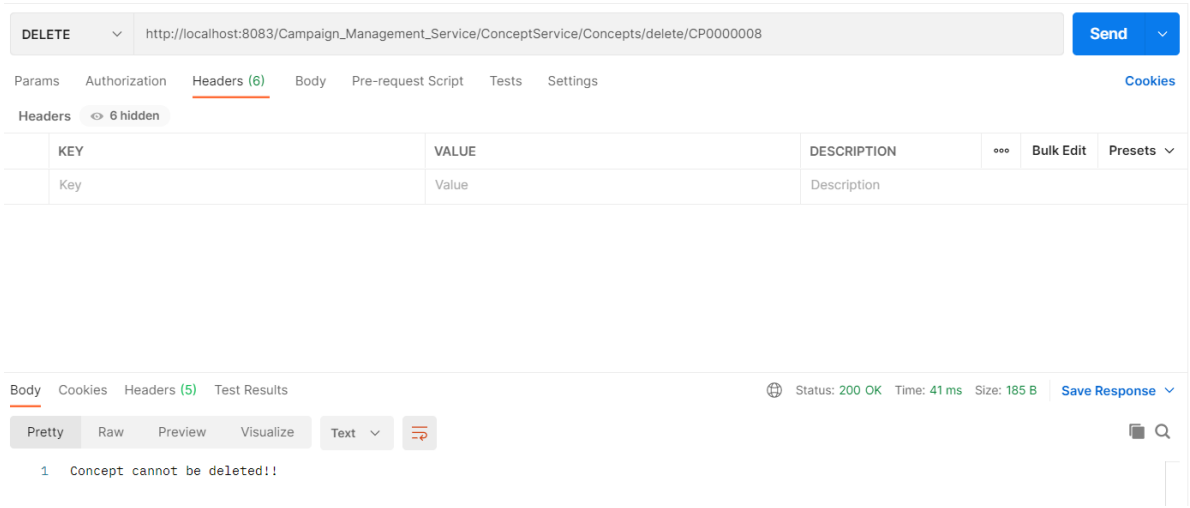


Figure 27: Invalid delete (deleted a concept with status complete) Test Result

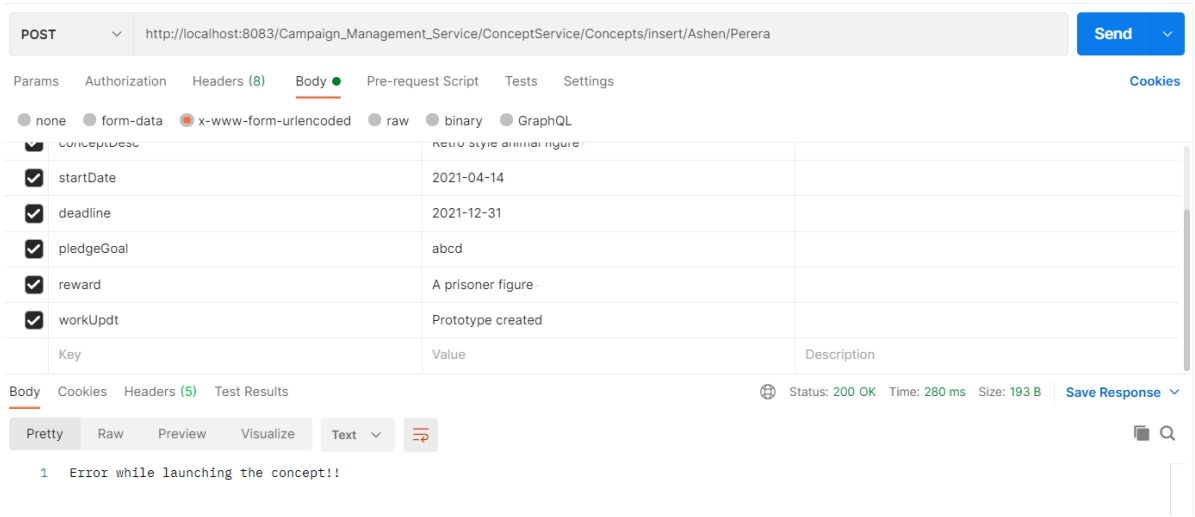


Figure 28: Invalid insert (updating a concept that is completed) Test Result

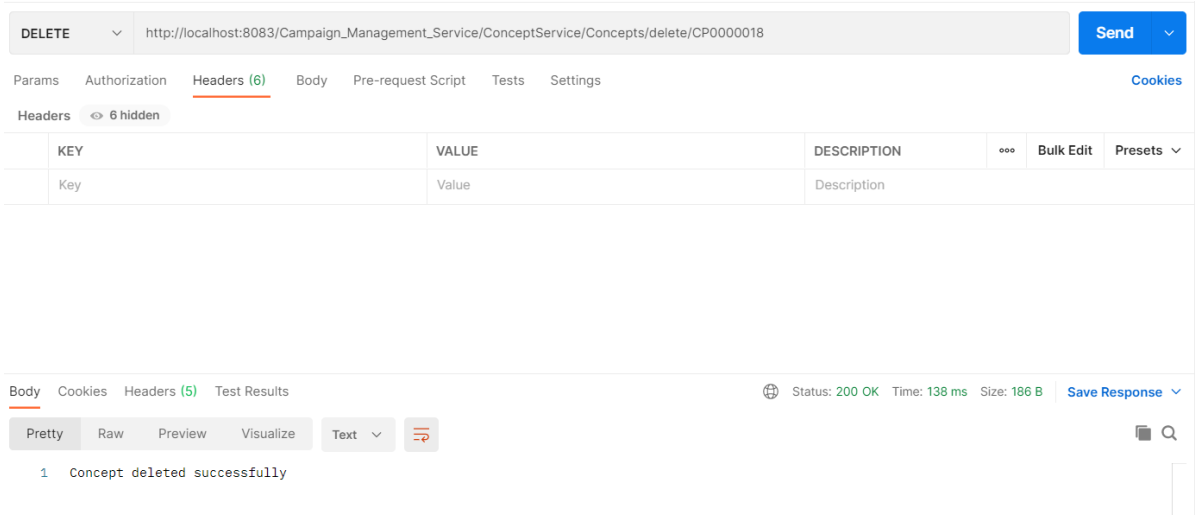


Figure 29: Valid delete Test Result

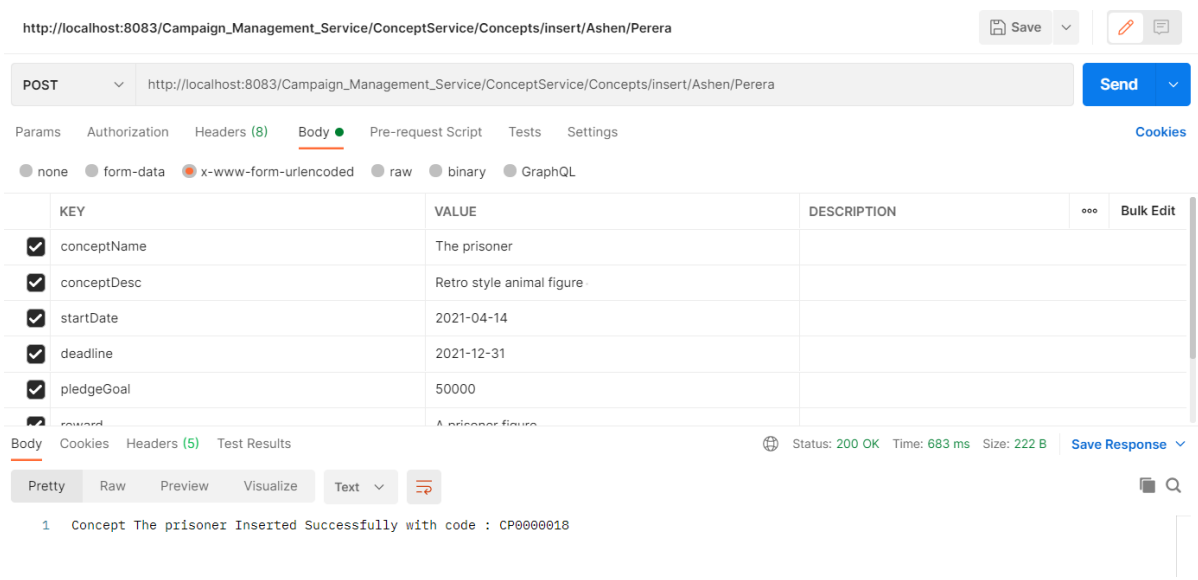


Figure 30: Valid Insert Test Result

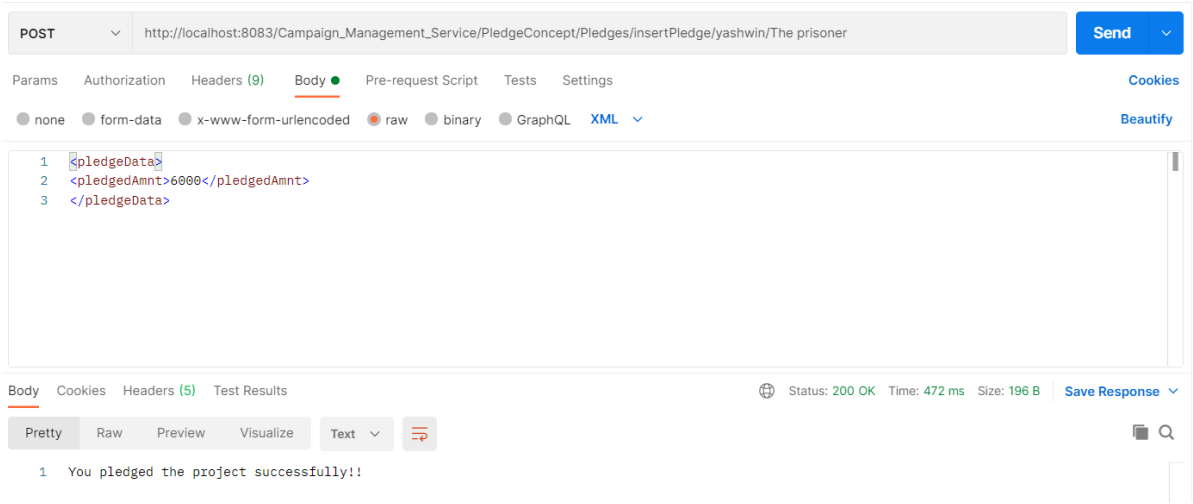


Figure 31: Valid Pledge Transaction Test Result

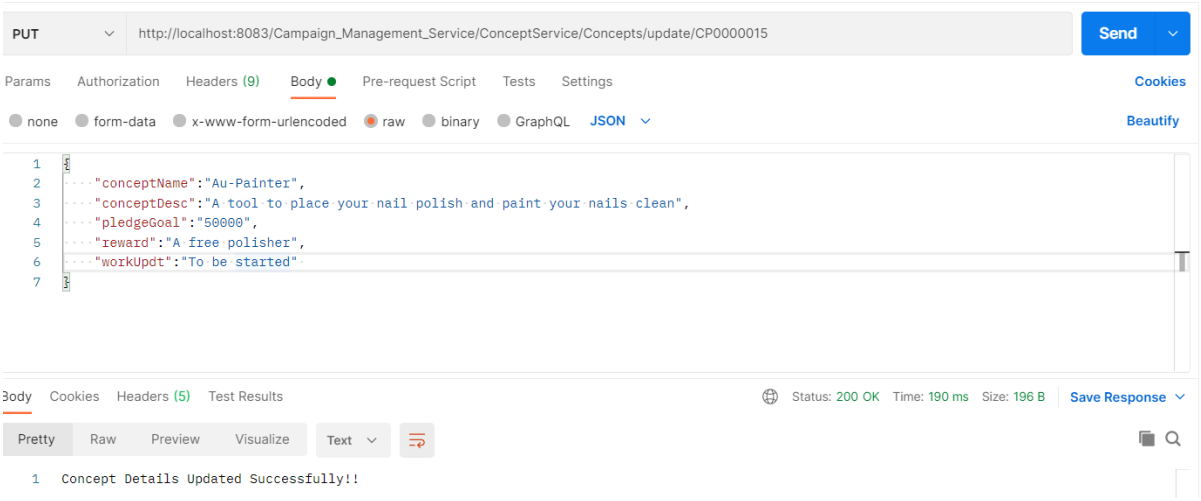
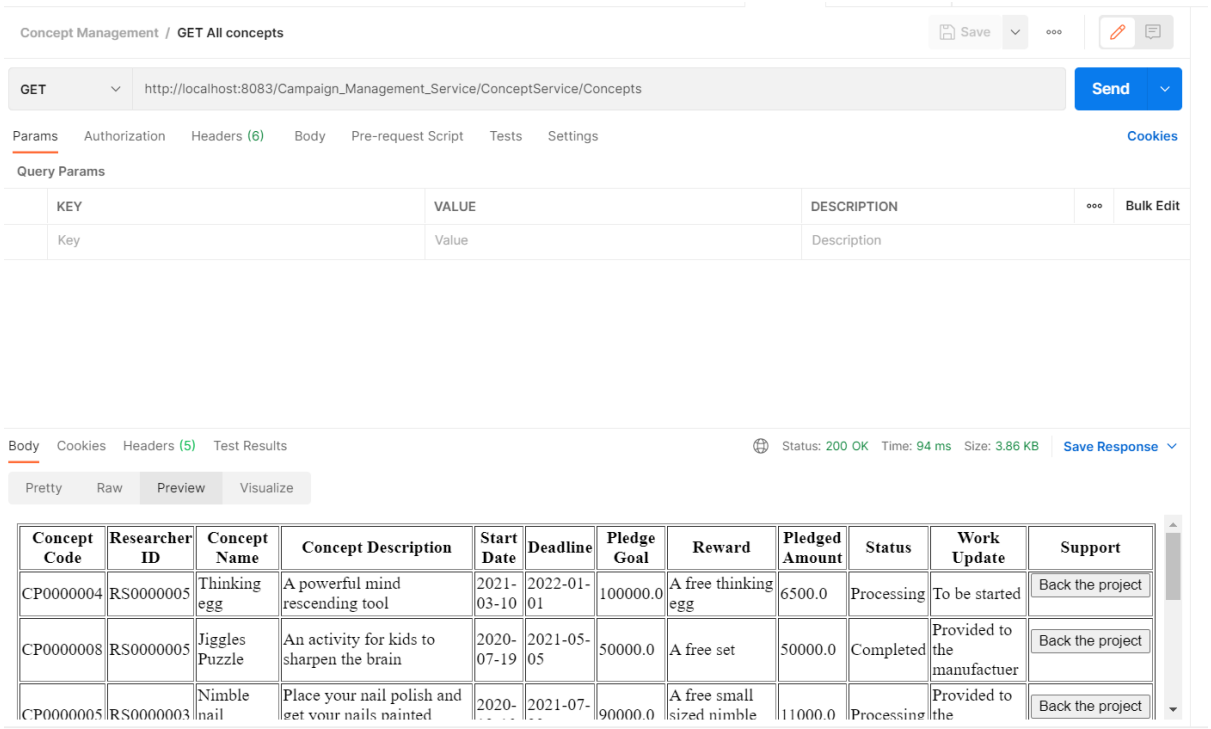
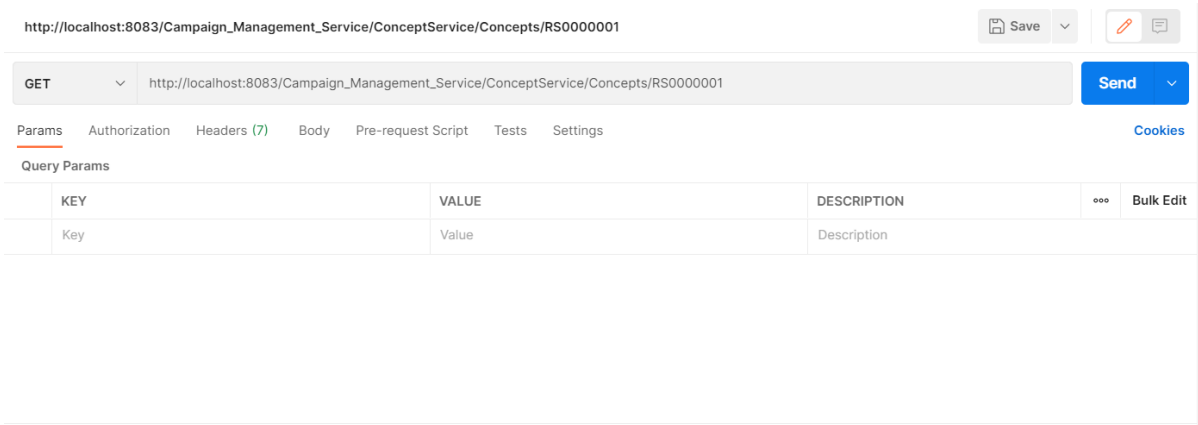


Figure 32: Valid update Test Result



Concept Code	Researcher ID	Concept Name	Concept Description	Start Date	Deadline	Pledge Goal	Reward	Pledged Amount	Status	Work Update	Support
CP0000004	RS0000005	Thinking egg	A powerful mind rescending tool	2021-03-10	2022-01-01	100000.0	A free thinking egg	6500.0	Processing	To be started	Back the project
CP0000008	RS0000005	Jiggles Puzzle	An activity for kids to sharpen the brain	2020-07-19	2021-05-05	50000.0	A free set	50000.0	Completed	Provided to the manufactuer	Back the project
CP0000005	RS0000003	Nimble nail	Place your nail polish and get your nails painted	2020-	2021-07-	90000.0	A free small sized nimble	11000.0	Processing	Provided to the	Back the project

Figure 33: View All concepts Test Result



Concept Code	Concept Name	Concept Description	Start Date	Deadline	Pledge Goal	Reward	Pledged Amount	Status	Work Update
CP0000014	Wall builder Lamp	Sticker portable lamps	2020-05-13	2021-05-05	68000.0	Animal designed wall builders	7000.0	Processing	To be started
CP0000015	Polisher tool	Simple auto nail painter tool	2020-05-13	2021-05-05	50000.0	A free polisher	11000.0	Processing	To be started
CP0000018	The prisoner	Retro style animal figure	2021-04-14	2021-12-31	50000.0	A prisoner figure	0.00	Processing	Prototype created

Figure 34: View for a specific researcher Test Result

Appendix C: Manufacturer and Patent services – Vithyashagar S.T. IT19058474

Appendix C.1: Internal Logic diagrams – Manufacturer Service

Appendix C.1.1: Class Diagram

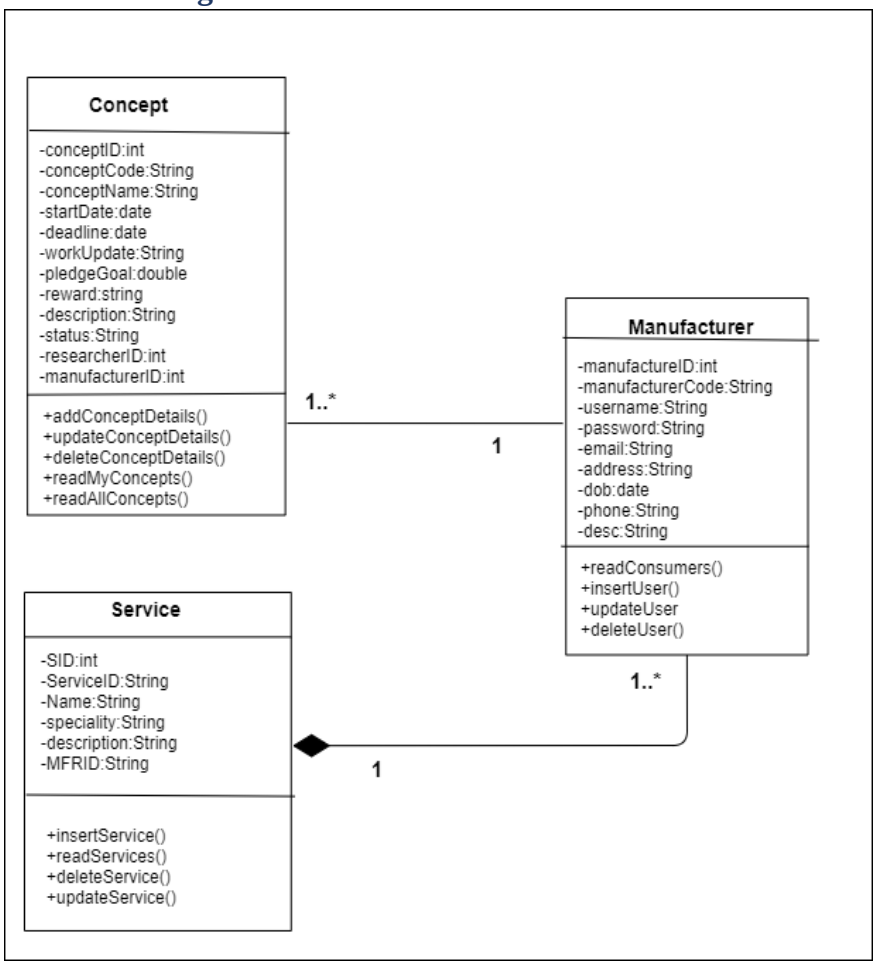


Figure 35: Manufacture Service class diagram

The above diagram demonstrates about the relationships between the services. Manufacturer is the one who introduces services to the system so without Manufacturer service cannot be existed so the connection between them is illustrated using **composition** and a concept can also exist without the services, so it shares an **association** relationship.

Appendix C.1.2: Architecture Diagram

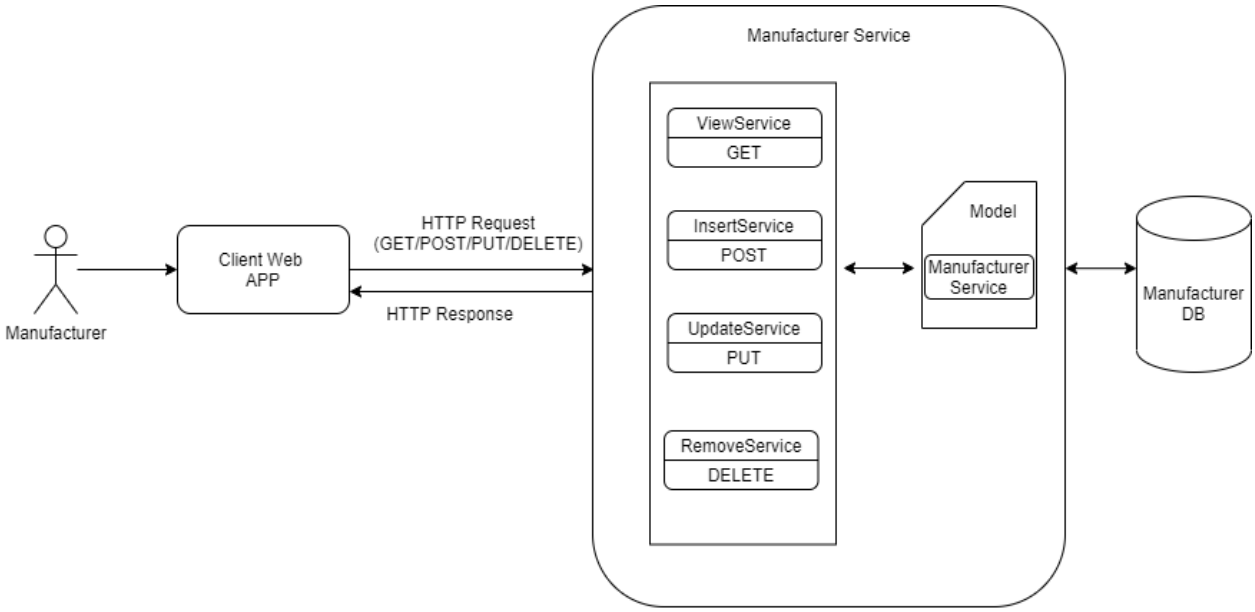


Figure 36: Manufacture Service architecture diagram

The above diagram shows the architecture of the service. The ManufacturerService service database operations are created in the Model class which serves as a server model to communicate with the database which stores ManufacturerService data. The main resource manipulated using the HTTP verbs in this service is “ManufacturerService”.

Appendix C.2: Database design of service – ER – Manufacturer Service

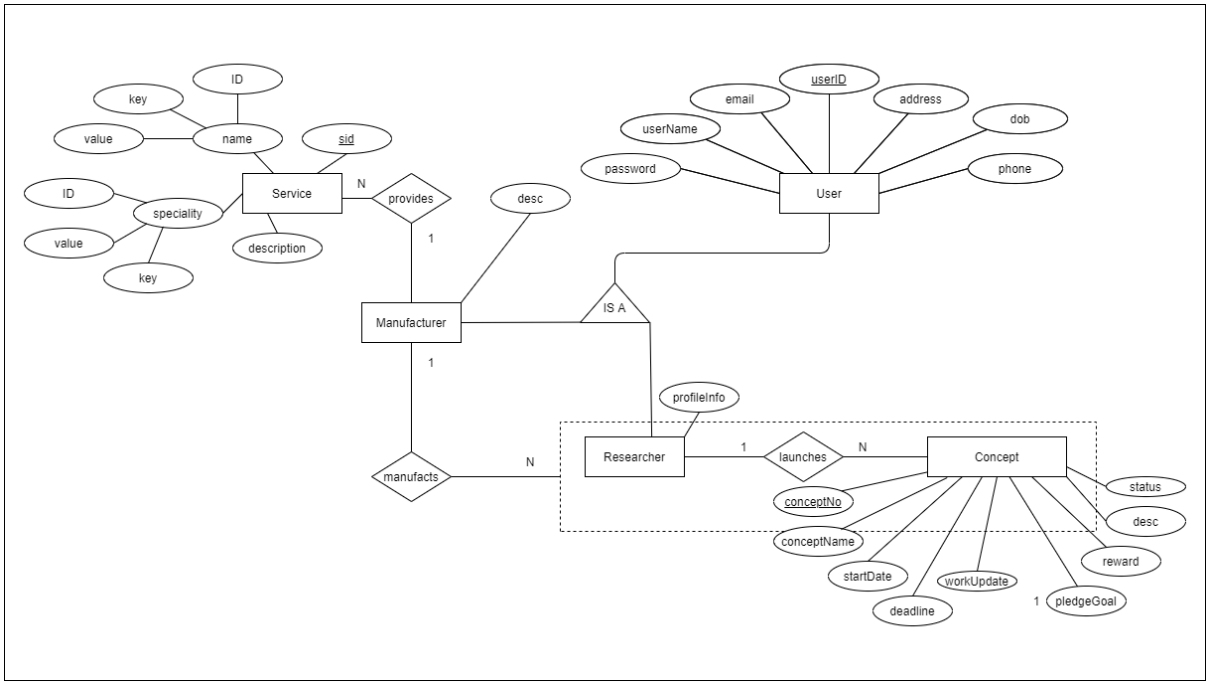


Figure 37: Manufacture Service ER diagram

This figure shows the relationship between the entities.A manufacturer can provide particular services by advertising through the system. A particular service can be later accessed by Reasearcher as per their concept needs.

Appendix C.3: Other relevant design diagrams – Manufacturer Service

Appendix C.3.1: Activity Diagram

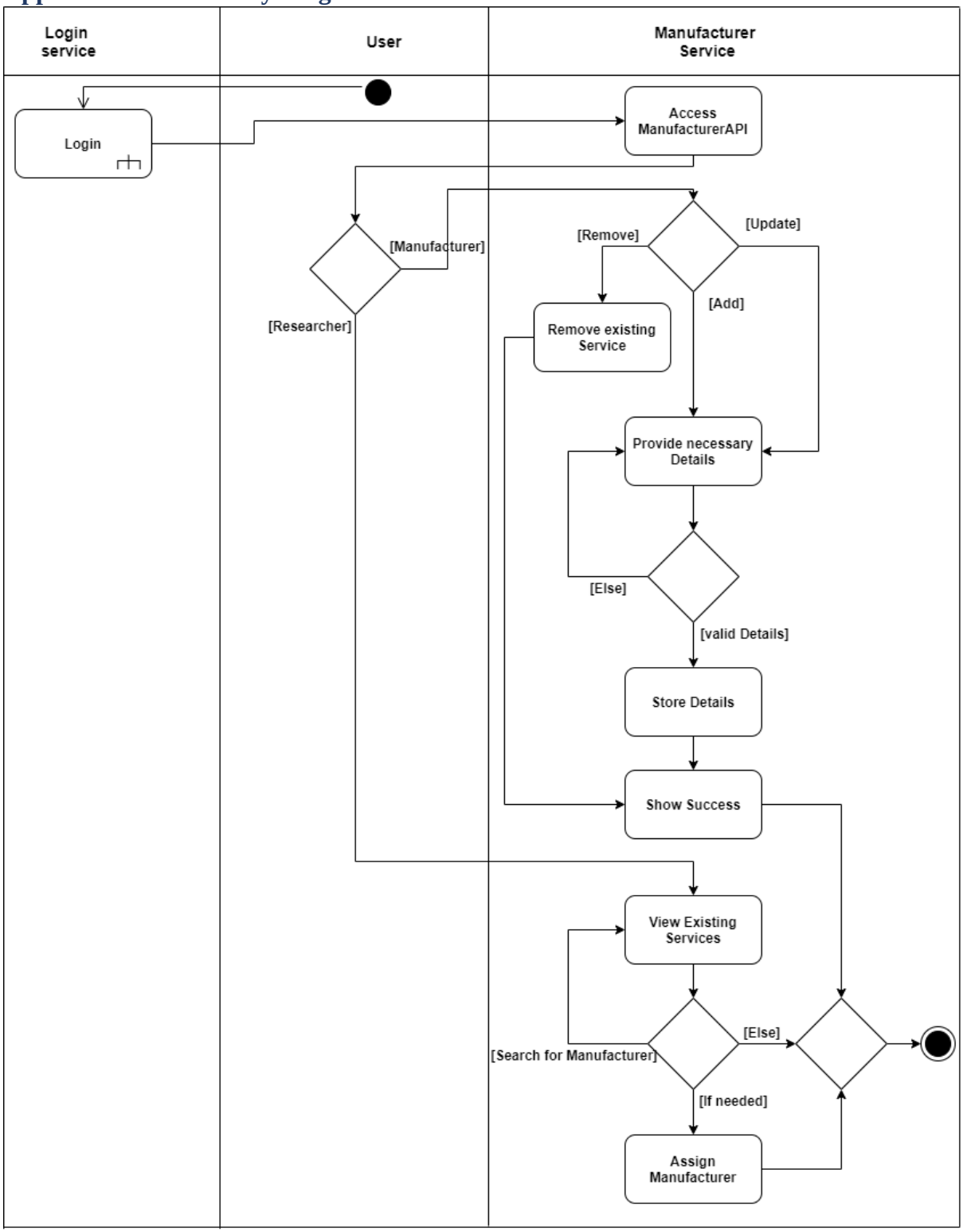


Figure 38: Manufacture Service Activity diagram

Following diagram describes about the internal flow of the ManufacturerService.

Appendix C.3.2: Use Case Diagram

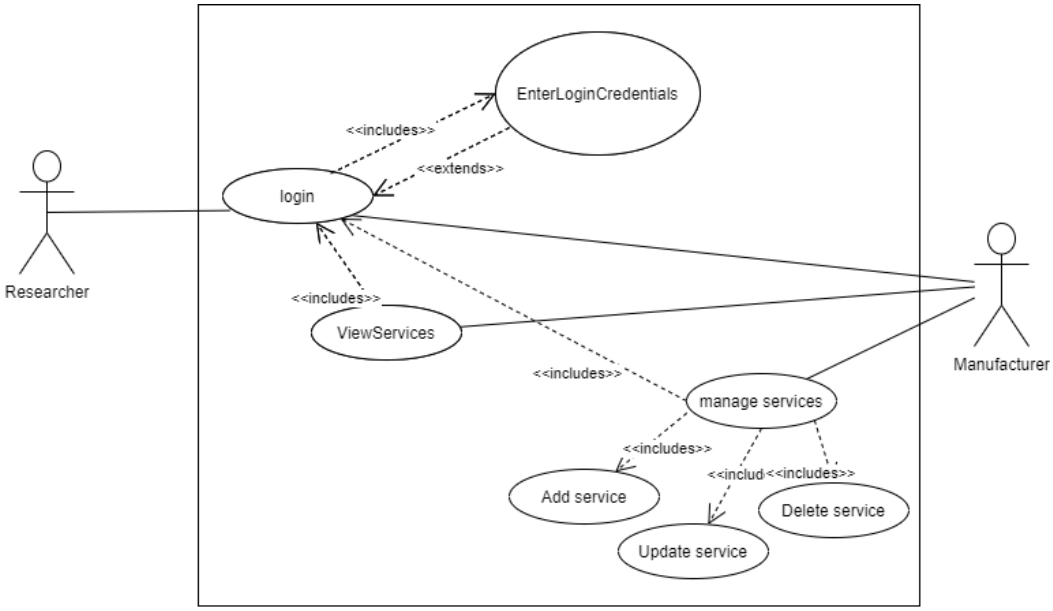


Figure 39: Manufacture Service use case diagram

The over figure explains about the Interactions between the actors and the system. Actors involved in this service are researcher and the Manufacturer.

Appendix C.4: Internal Logic diagrams – Patent Service

Appendix C.4.1: Class diagram

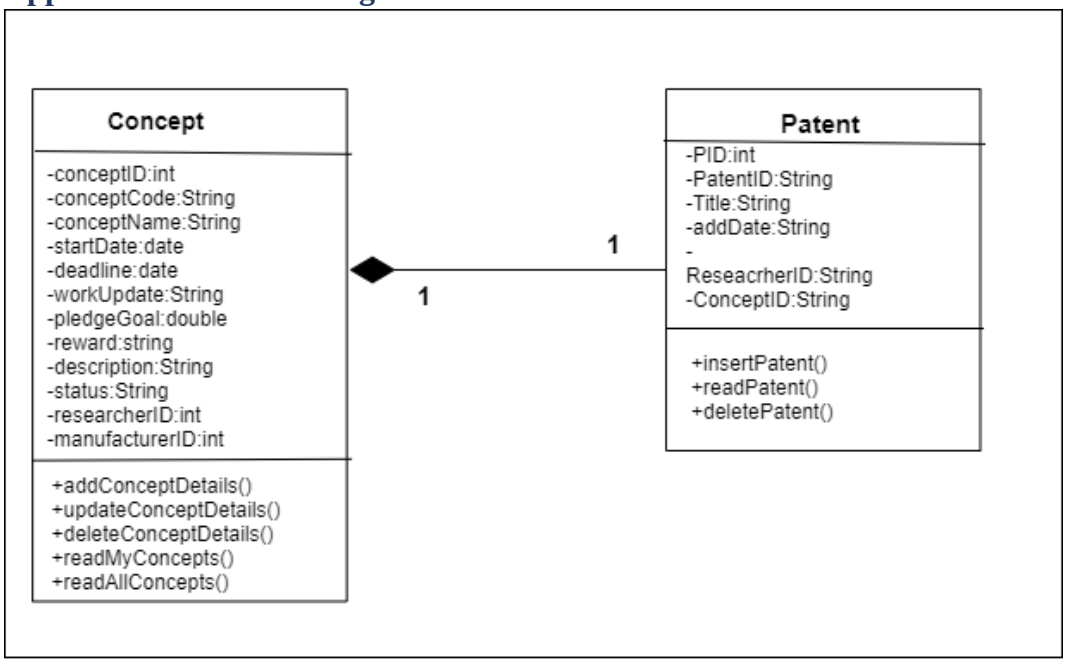


Figure 40: Patent Service class diagram

The above diagram demonstrates about the relationships between the services. Without a concept there cannot be an existing patent application since its mapped as **Composition**.

Appendix C.4.2: Architecture diagram

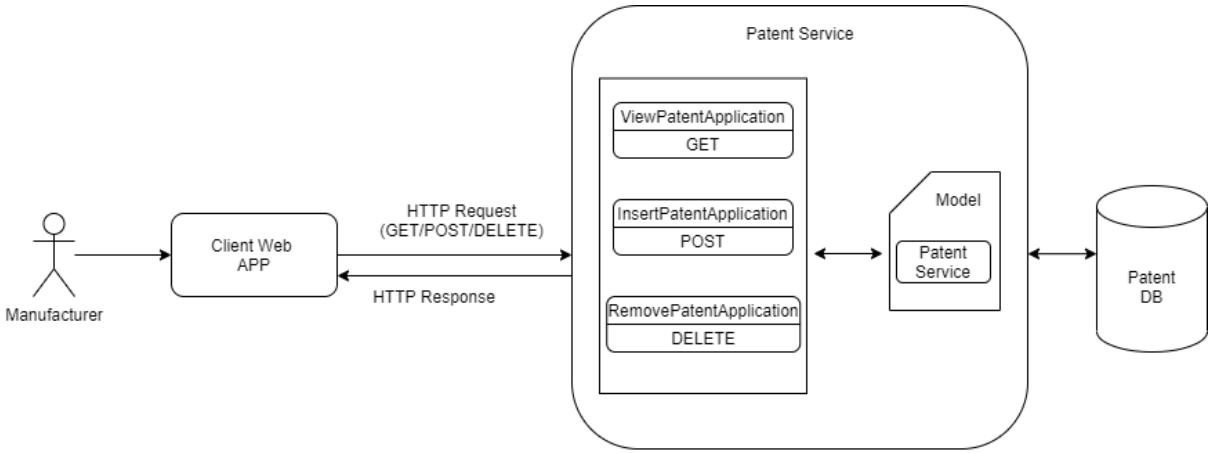


Figure 41: Patent Service architecture diagram

The above diagram shows the architecture of the service. The Patent service database operations are created in the Model class while acting as a server model to communicate with the database which stores Patent data. The main resource manipulated using the HTTP verbs in this service is “PatentService”.

Appendix C.5: Database design of service – ER –Patent Service

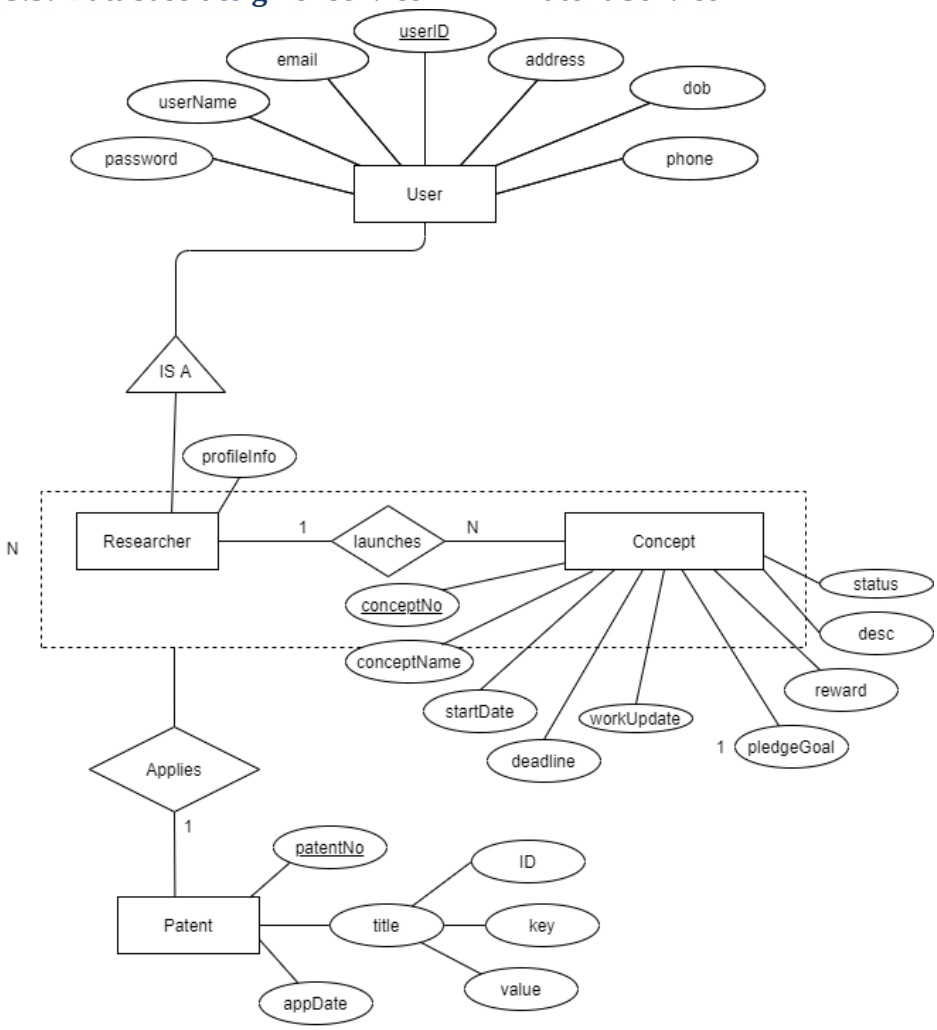


Figure 42: Patent Service ER diagram

This figure shows the relationship between the entities.A Reasearcher can request the system to Process the patent application form as the middle man.

Appendix C.6: Other relevant design diagrams – Patent Service

Appendix C.6.1: Activity Diagram

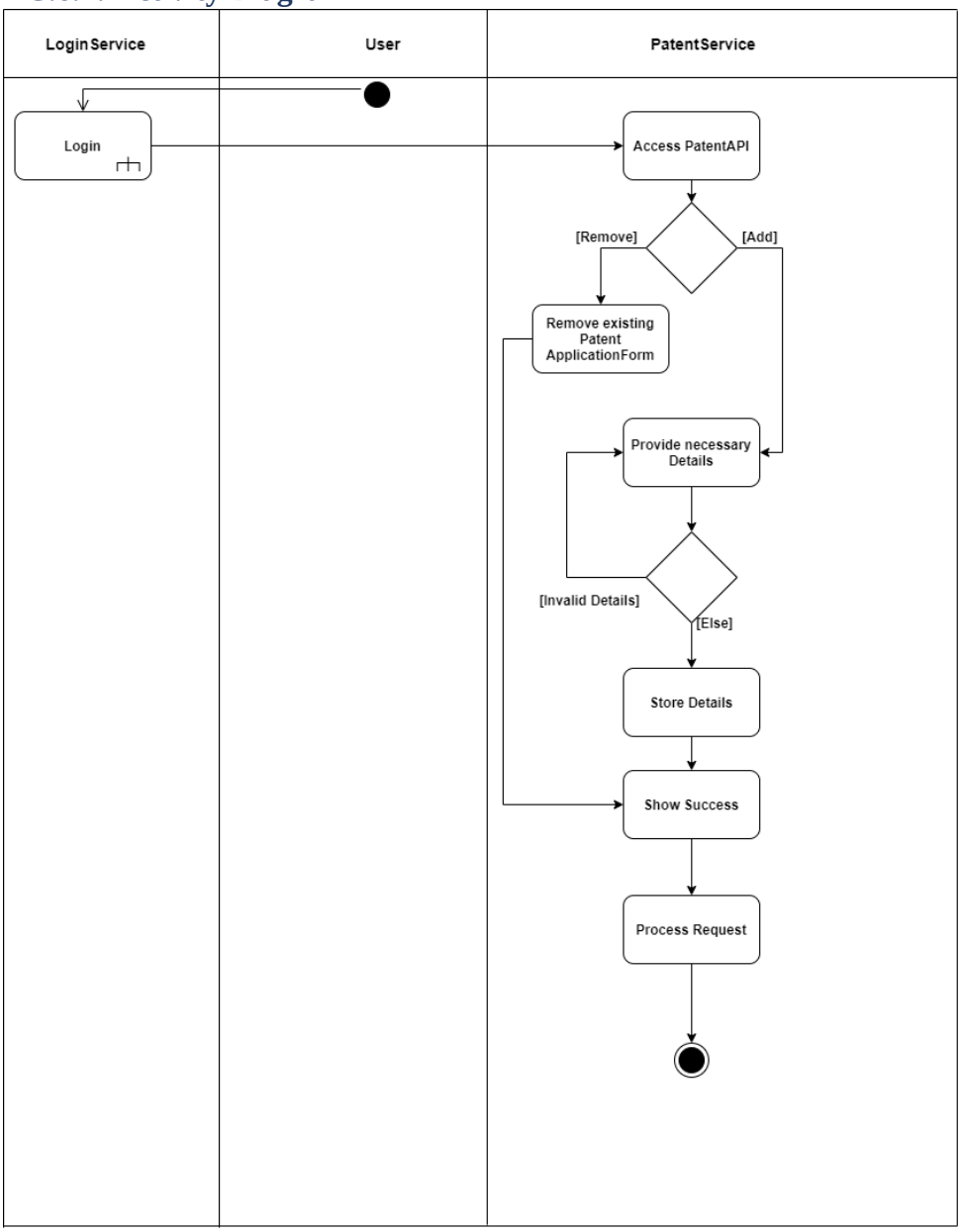


Figure 43: Patent Service activity diagram

Following diagram describes about the internal flow of the PatentService.

Appendix C.6.2: Use Case Diagram

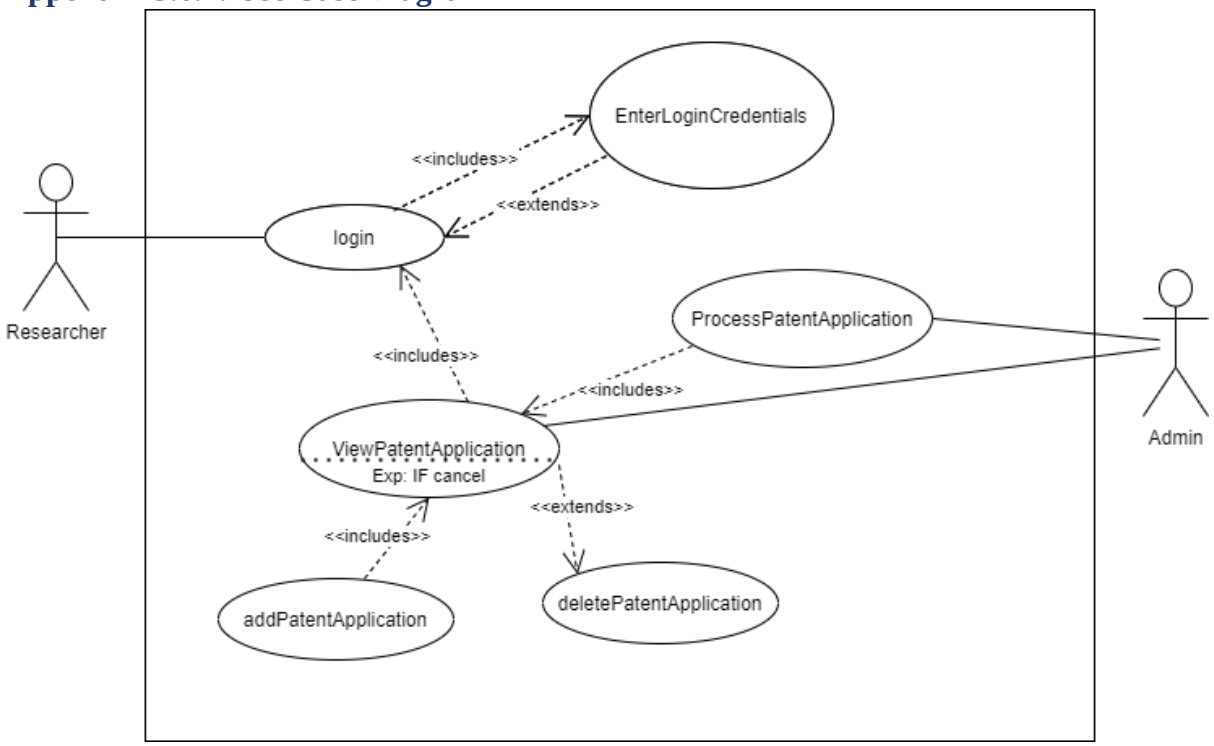


Figure 44: Patent Service use case diagram

The figure explains about the Interactions between the actors and the system. Actors involved in this service are researcher and the Admin.

Appendix C.7: Testing Results (Manufacturer Service)

Test case ID	**Description**	**Test input**	**Expected Output**	**Test output**	**Test status**
1	Insert null values for any column field	*Speciality = “”*	*“Error while Inserting Service”*	*“Error while Inserting Service”*	Pass
2	Get User ID from UserService	*userName = “krish”*	*“CM0000001”*	*“CM0000001”*	Pass
3	Get Manufacturer specific details when providing the username	*userName = “krish”*	Table of Services added by the user	Table of Services added by the user	Pass
4	Update a Service by providing necessary details	Name = “<updated value>”	*“Updated successfully”*	*“Updated successfully”*	Pass
5	Delete a Service by providing the SID	*SID = “MS0000001”*	*“Service Deleted”*	*“Service Deleted”*	Pass

Appendix C.8: Postman API test result screenshots – Manufacturer

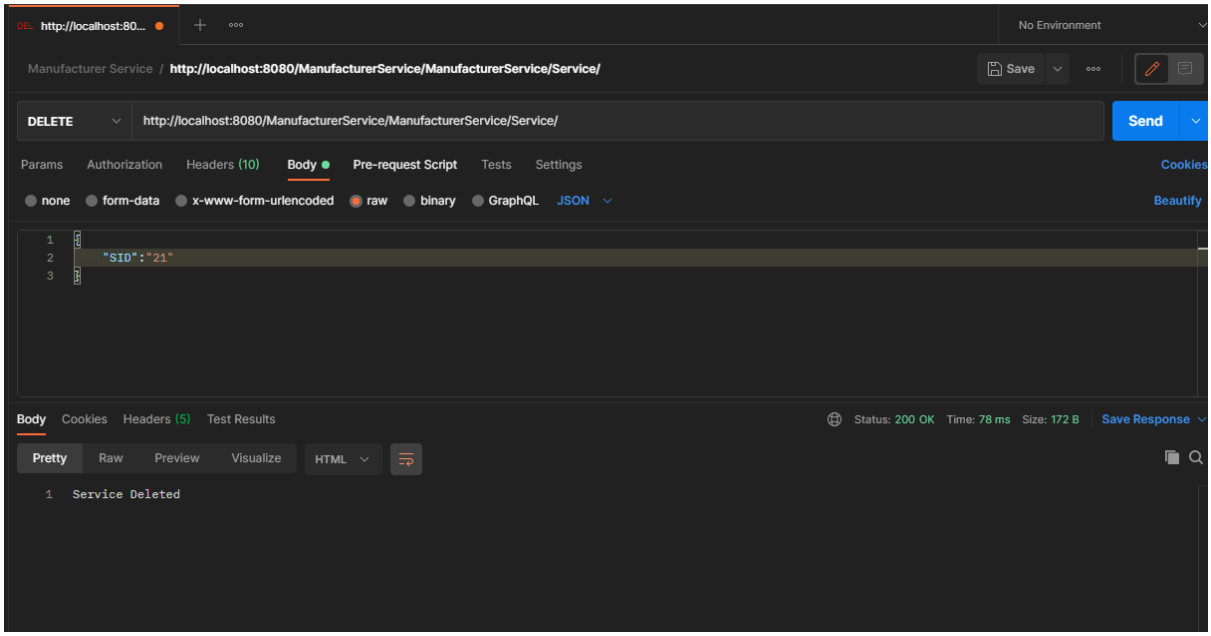


Figure 45:Delete Manufacturer Service Test Result

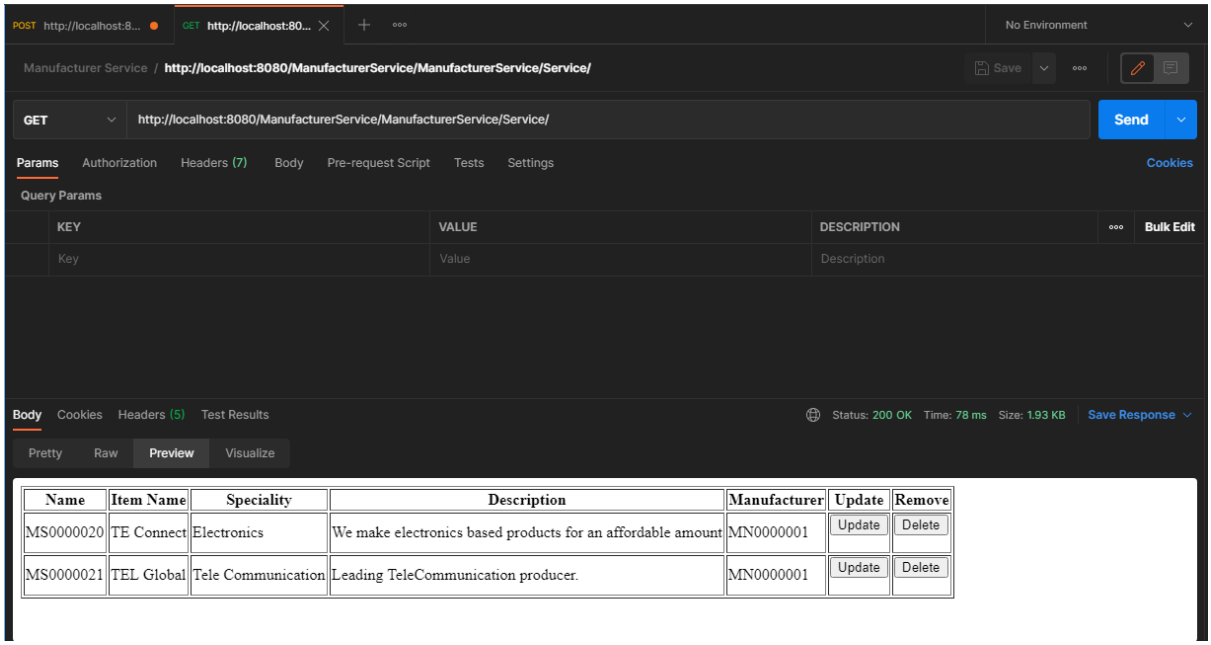


Figure 46: Get Manufacturer Service Test Result

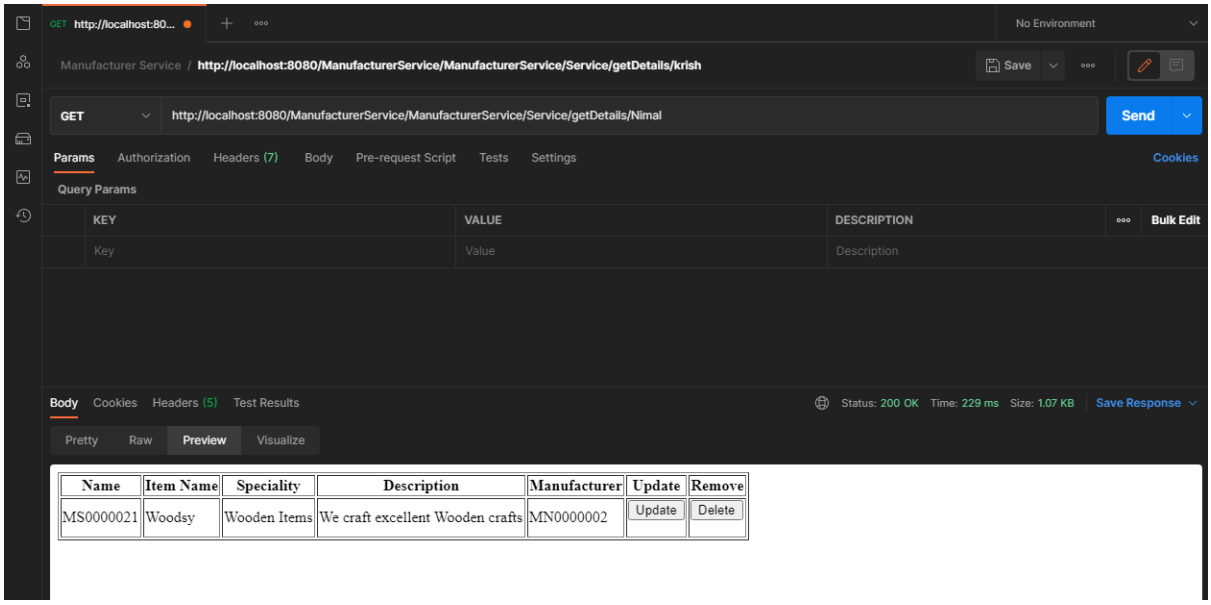


Figure 47: Get User Specific Details Test Result

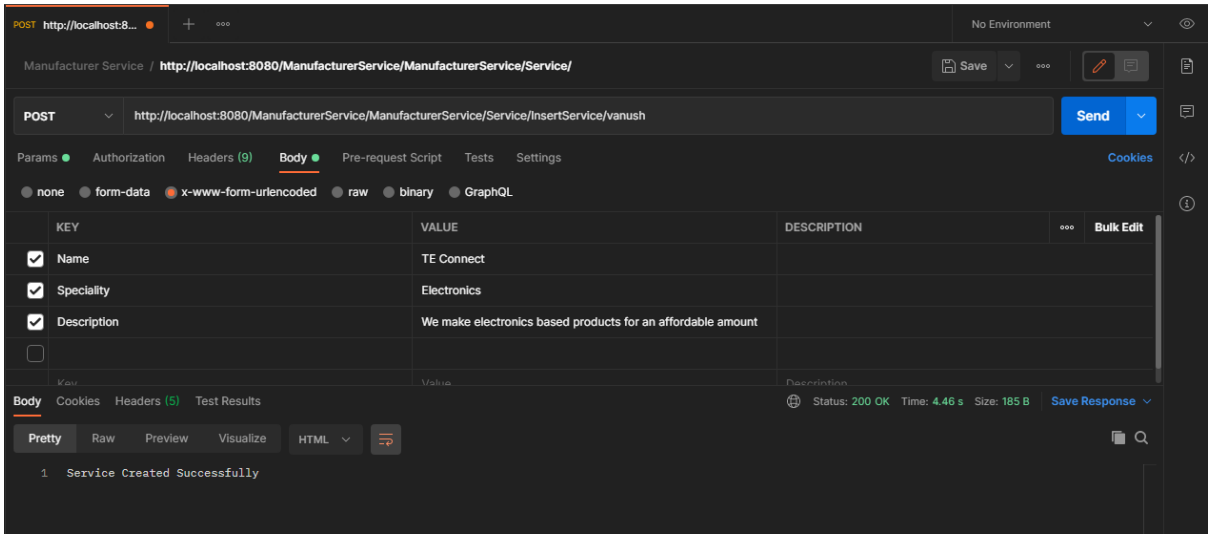


Figure 48: Insert Manufacturer Service Test Result

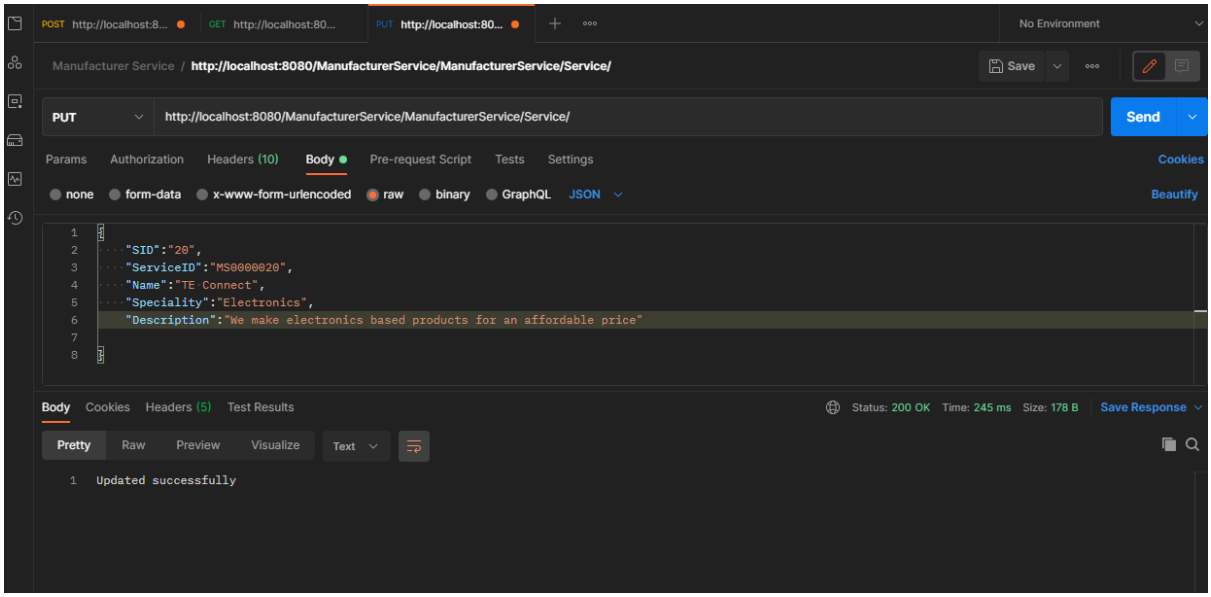


Figure 49: update manufacturer Service Test Result

Appendix C.9: Postman API test result screenshots – Patent

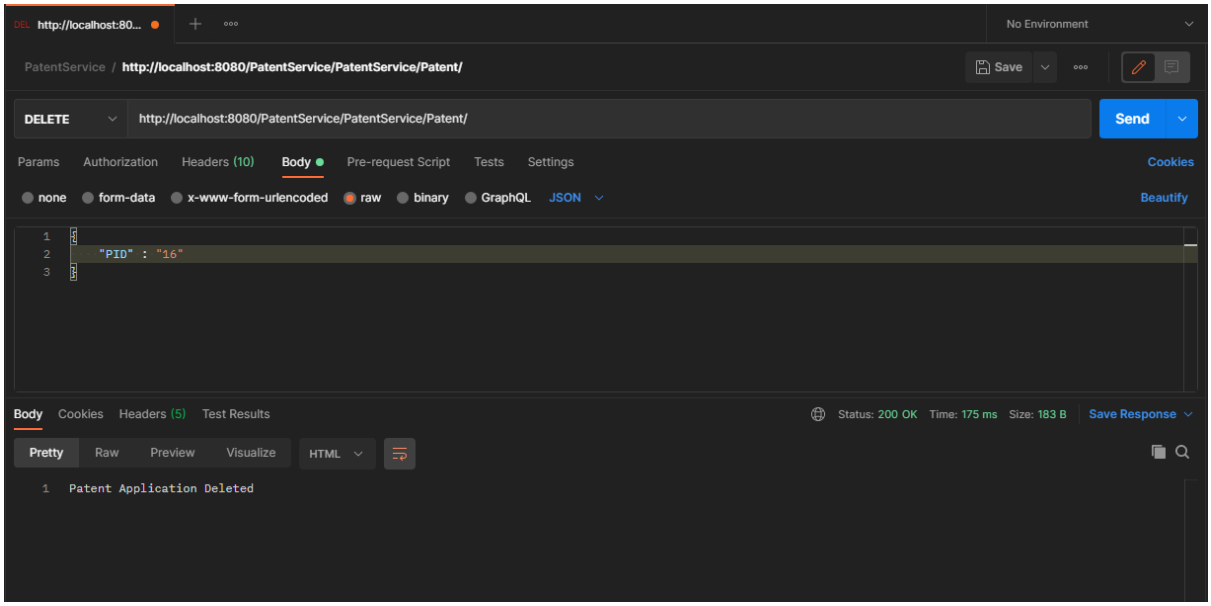


Figure 50: Delete Patent Application Test Result

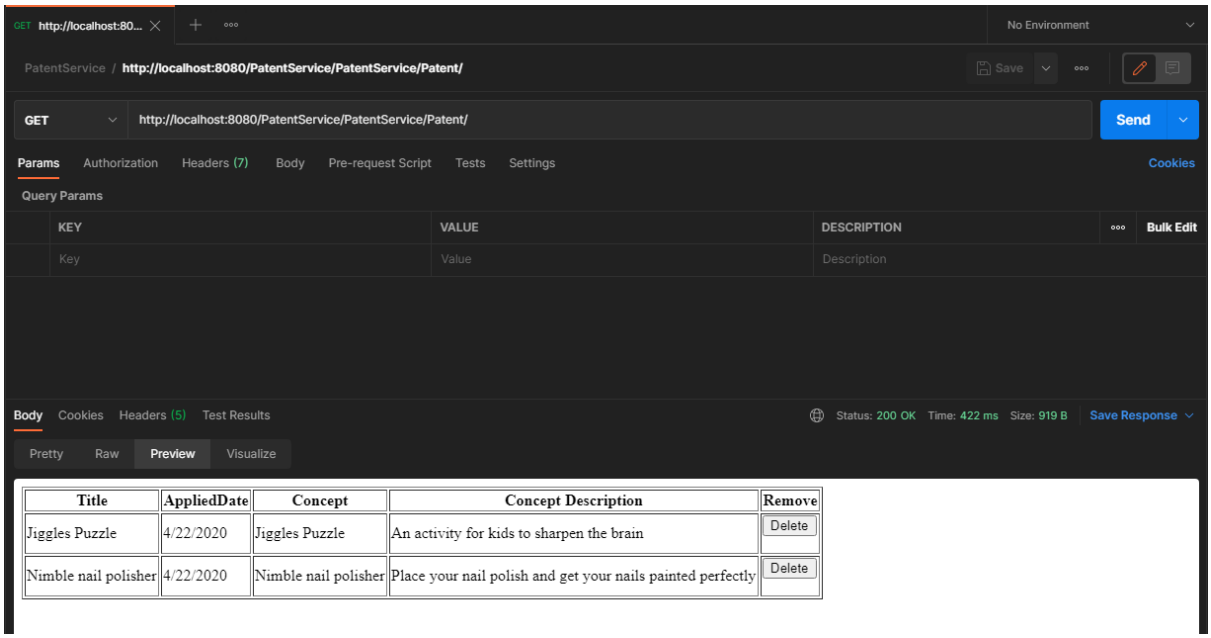


Figure 51: Get All Patent Applications Test Result

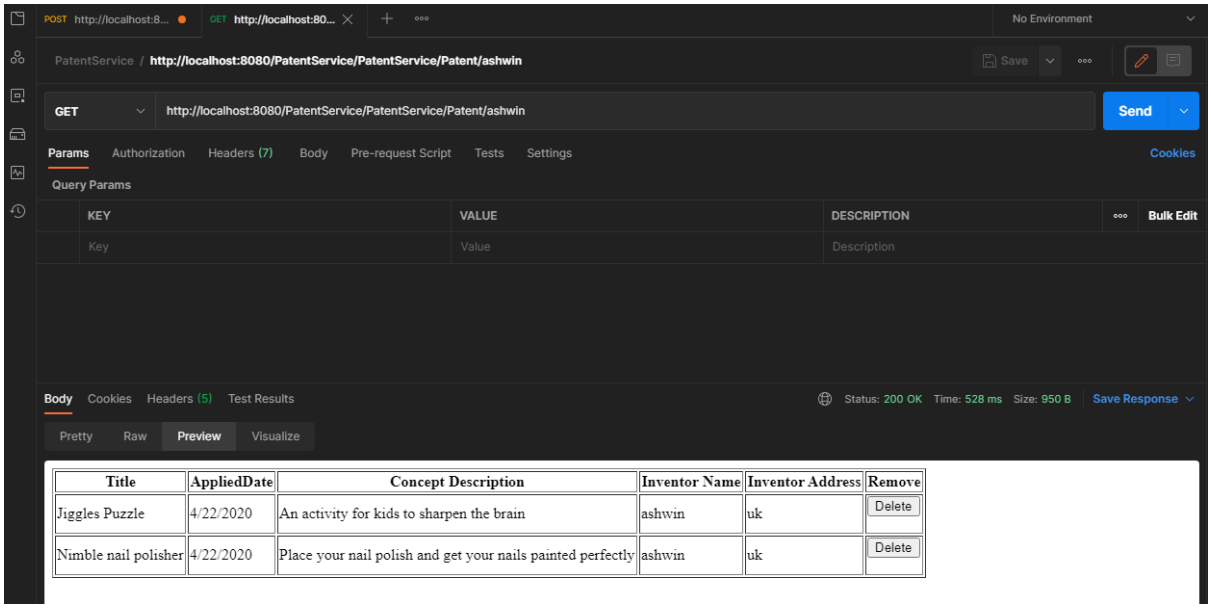


Figure 52: Get User Specific Patent Applications Test Result

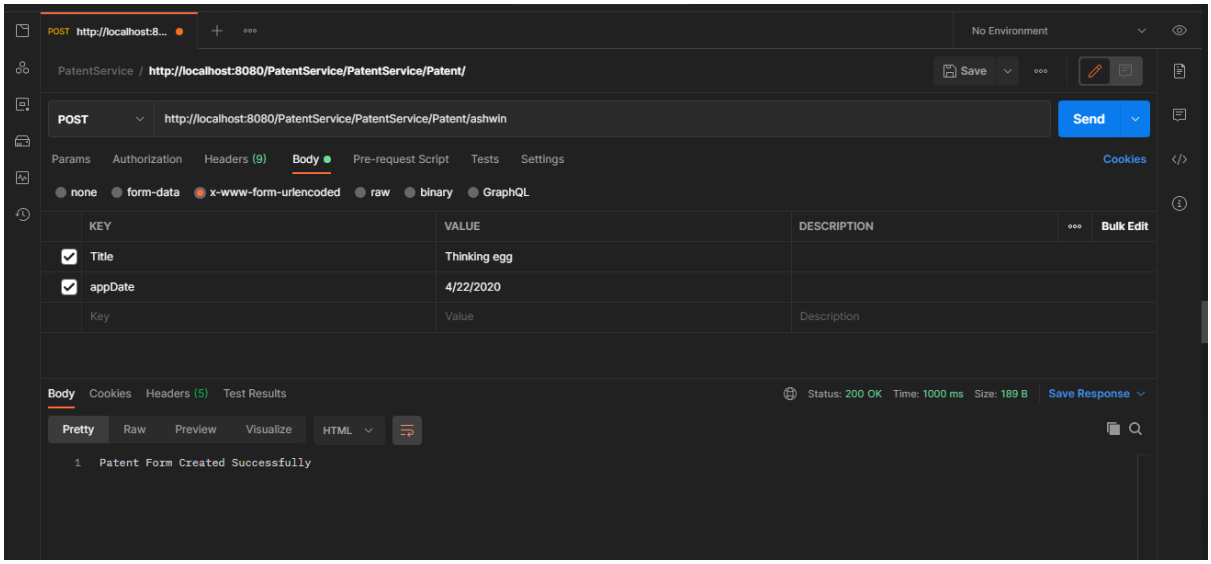


Figure 53: Patent Insert Application Test Result

```
public class Hashing {
    // We need a bytesToHex method first. So, from -
    // http://stackoverflow.com/a/9855338/2970947
    final protected static char[] hexArray = "0123456789ABCDEF"
        .toCharArray();

    public static String bytesToHex(byte[] bytes) {
      char[] hexChars = new char[bytes.length * 2];
      int v;
      for (int j = 0; j < bytes.length; j++) {
        v = bytes[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
      }
      return new String(hexChars);
    }

    // Change this to something else.
    private static String SALT = "M@nufa7urer8$7*";
    // A password hashing method.
    public String hashPassword(String in) {
      try {
        MessageDigest md = MessageDigest
            .getInstance("SHA-256");
        md.update(SALT.getBytes());        // <-- Prepend SALT.
        md.update(in.getBytes());
        // md.update(SALT.getBytes());     // <-- Or, append SALT.

        byte[] out = md.digest();
        return bytesToHex(out);            // <-- Return the Hex Hash.
      } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
      }
      return "";
```

The Above code was used to implement the security feature of the project.

Appendix D: Product Management service - Hillary J.R. IT19021430

Appendix D.1: Internal Logic diagrams

Appendix D.1.1: Class diagram

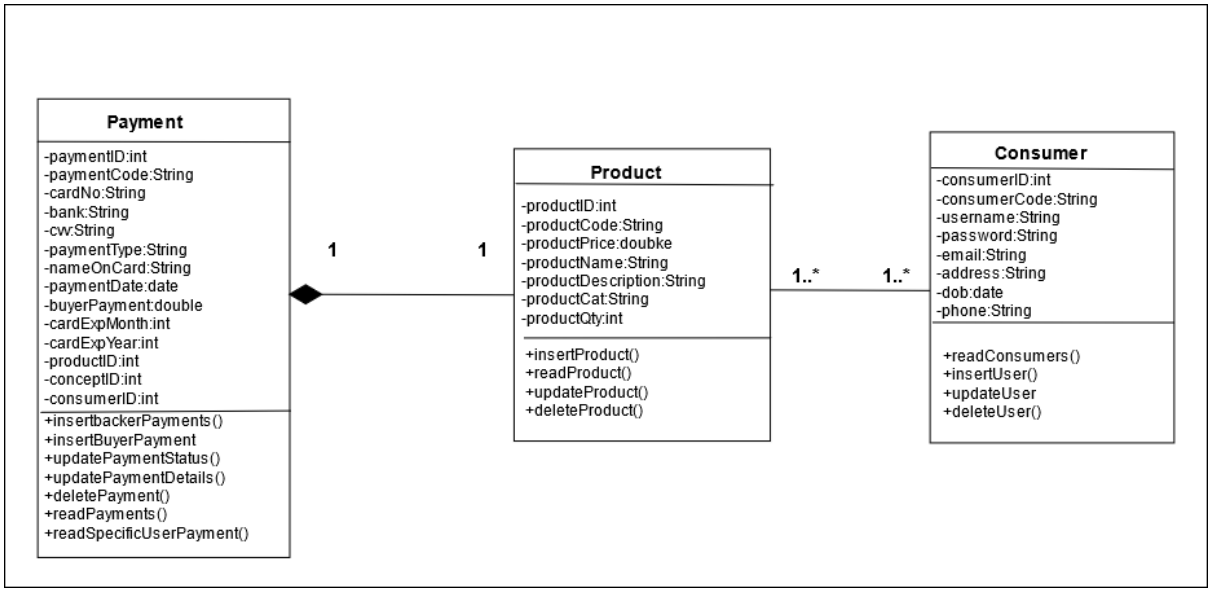


Figure 54: Product Service Class diagram

When a concept reaches its fully pledged target amount the owner of that concept can manufacture this concept and sell the final product through the platform its self. The GB System not only creates a place for young researchers to raise funds for their concepts but also it creates the market through the GB online platform to sell their end products.

The above class diagram shows the associated main classes with the Product Management Service. The product class mainly communicates with the Payment and Consumer classes. A product that is been put to the market can be bought by a consumer type user therefore each product will have a payment. The composite relationships indicated imply that payment cannot exist without a product.

Appendix D.1.2: Architecture diagram

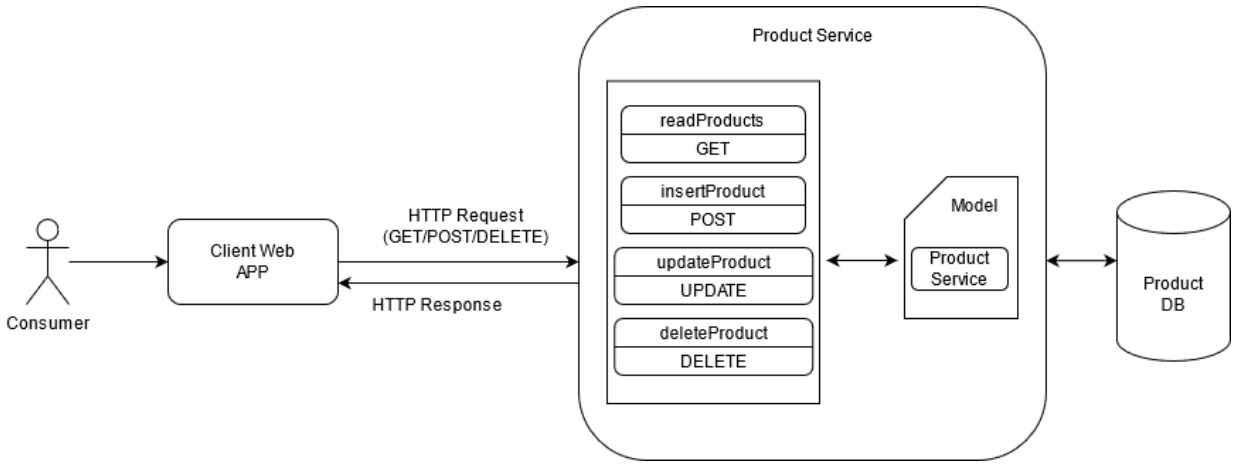


Figure 55: Product Service architecture diagram

- The above architecture diagram for the product management service USE MVC WITH REST
- In the above MVC is pretty straightforward. There is a Model, a Controller and a View. When we create a website, it all come together as '*client sends REST keyword request to server Then server matches the requested URL to the controller action which then calls the model for data gathering/processing, gets the result and returns the result back to the client as a request*.
- The product management services are implemented as com and model packages
- The CRUD operations are implemented under the com package which is the server model. This servers as the medium for communication with the database where all details about products are stored. The main resource that is responsible in the product management service is the product
- HTTP requests are sent by the consumer to perform operations like insert, view, delete and update of users according to the user type via the client APP. These requests are process by the server model and response is sent back to the client APP as HTTP response.

Appendix D.2: Database design of service - ER

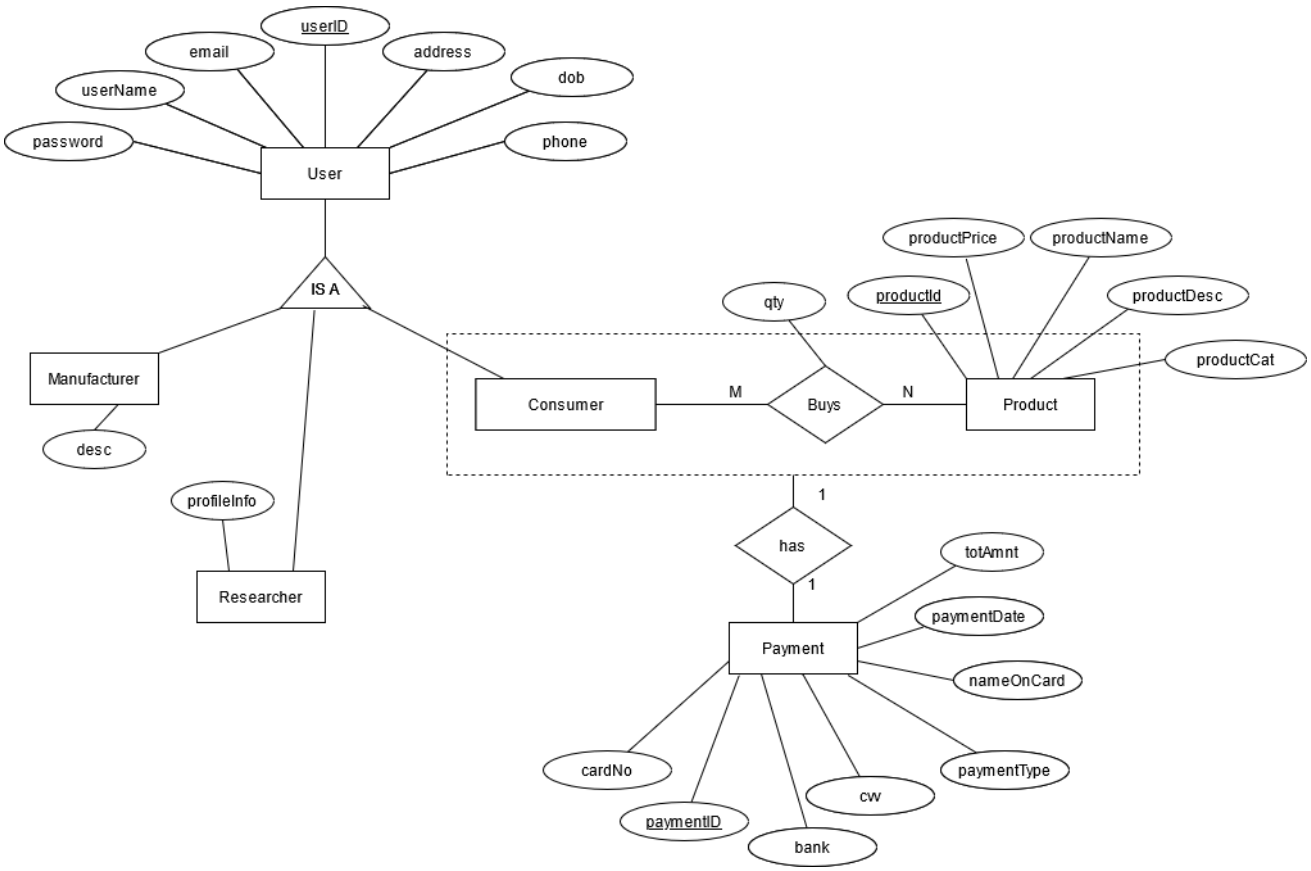


Figure 56: Product Service ER Diagram

In the above ER diagram drawn for the Product Management Service the main entities which are associated with the product entity are the User ISA Consumer entity and Payment entity. When a product is a put to market by the researcher after successfully launching the concept through GB Platform A registered consumer in the GB system can buy these products. One product can be abought by many consumers and one consumer can buy many products. A consumer needs to make payment for each product that bought. This will be then monitored and handled by the payment service.

Appendix D.3: Other relevant design diagrams

Appendix D.3.1: Activity Diagram

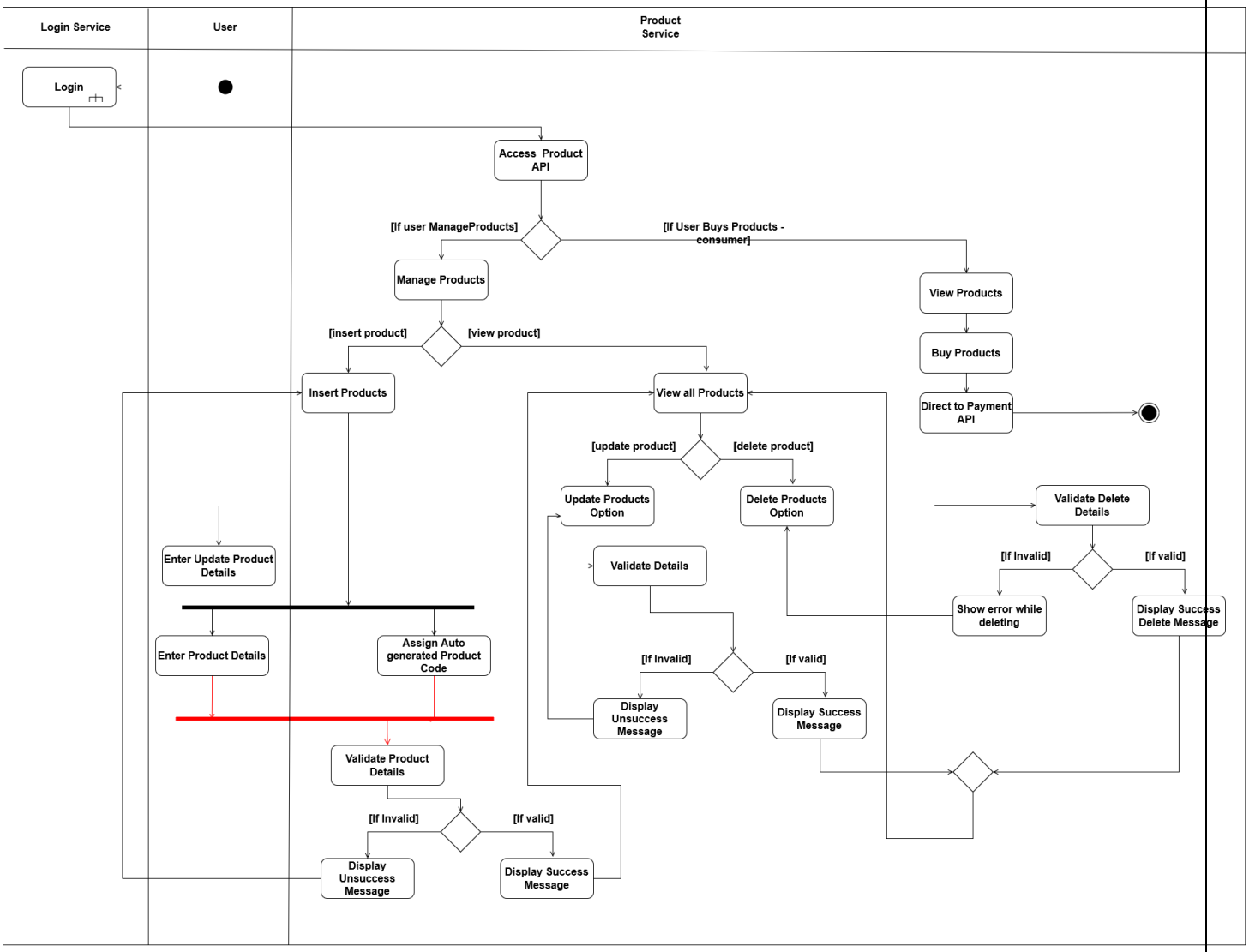


Figure 57: Product Service Activity diagram

The Product service and API can be accessed by two types of users after logging they are the consumer and admin. An admin can manage products, he will be granted with options such as insert new product, view existing products and update or delete them. While inserting a new product an auto generated productCode will be automatically added by the product service for the inserting details. The inserted details will be validated if successful it will display the successful message and redirect the user to the view all products. If unsuccessful it will display the error message and user have to re-enter the details. If the user deletes a product after validation if the process was successful it will prompt the successful message else it will prompt the error message. A user who is consumer the system will display all available products the user can view all products and purchase them which will redirect to the payment API.

Appendix D.3.2: Use Case Diagram

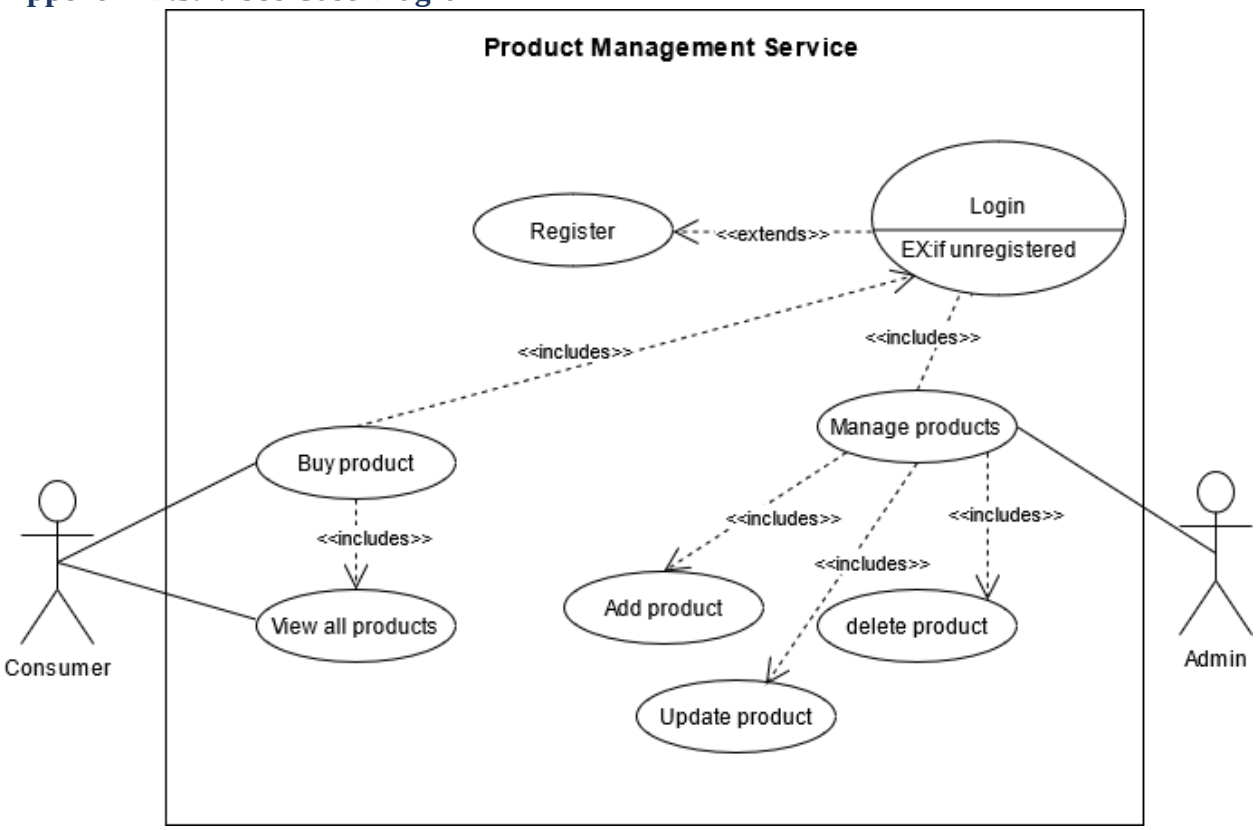


Figure 58: Product Service Use case diagram

As shown in the above Use Case diagram drawn for the Product Management Service, the main actors involved in the product management service are the Admin and Consumer. An admin manages the products. Manage products includes operations such as Add product, update product and Delete product. The consumer can buy a product in order to buy a product he should first view all products. Both the actor's consumer and admin should have a valid login in order to do any operations if they do not have valid login. If they are not registered to the system, they should first register and create an account.

Appendix D.4 Testing Results

Test Case ID	Description	Test input(s)	Expected Output(s)	Actual Output(s)	Test results
1	Any required field cannot be null for insert	productName= “ ”	Display message “Error while inserting the product”	Displayed the message “Error while inserting the product”	PASS
2	productPrice should be a number/ Float value	productPrice = “abc”	Display message “Error while inserting the product”	Displayed the message “Error while inserting the product”	PASS
3	productQty should be a number for insert	productQty = “r”	Display message “Error while inserting the product”	Displayed the message “Error while inserting the product”	PASS

4	Update product details	{
		"productId":"6",
		"productCode":"PD0000005",
		"productName":"Normal gas",
		"productPrice":"1000",
		"productDesc":"Not included",
		"productCat":"Electrical",
		"productQty":"4"
		}
			Display message “Product details Updated successfully”	Displays the message “Product details Updated successfull!”	PASS

5	Update a record with an ID not present in the database	{
		"productId":"100",
		"productCode":"PD0000005",
		"productName":"Normal gas",
		"productPrice":"1000",
		"productDesc":"Not included",
		"productCat":"Electrical",
		"productQty":"4"
		}
			Display message “Error while updating the product”	Displays the message “Error while updating the product”	PASS

6	Delete a product record with correct productId	<productData> <productId>5</productId> </productData>	Display message “Product Deleted successfully”	Displays message “Product Deleted successfully”	PASS

7	Delete a product record with an ID not present in the database	<productData> <productId>100</productId> </productData>	Display message “Error while deleting the product”	Displays the message “Error while deleting the product”	PASS
8	Get consumerCode from user service by passing userName	Passing valid user name from query param http://localhost:8001/ProductService/ProductService/Buying/getConsumerCode/yashwin	Display “CM00000002”	Displayed output “CM00000002”	PASS
9	Invallid username while retrieving consumerCode from the user service	Passing valid user name from query param http://localhost:8001/ProductService/ProductService/Buying/getConsumerCode/invallidName	Display “Error while retrieving values from database”	Display “Error while retrieving values from database”	PASS

Appendix D.5: Postman API test result screenshots

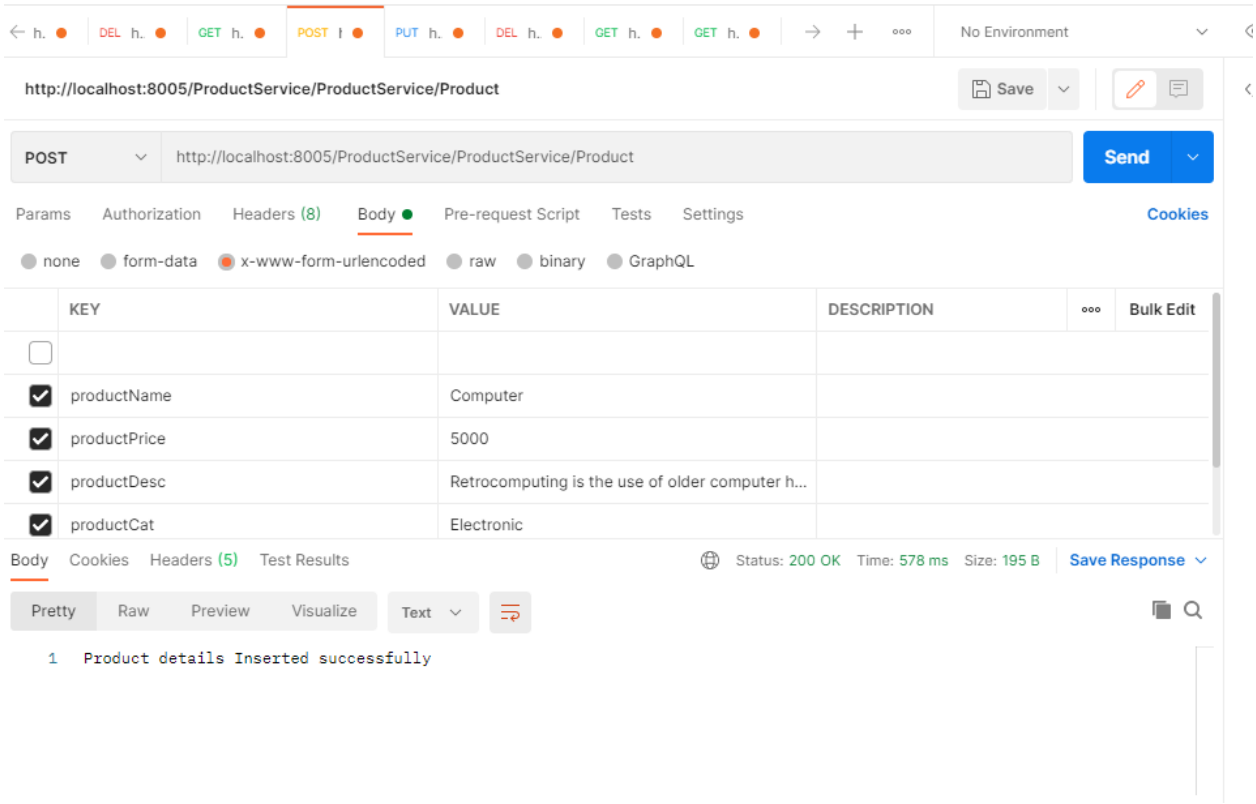


Figure 59: Product Insertion Successful Test Result

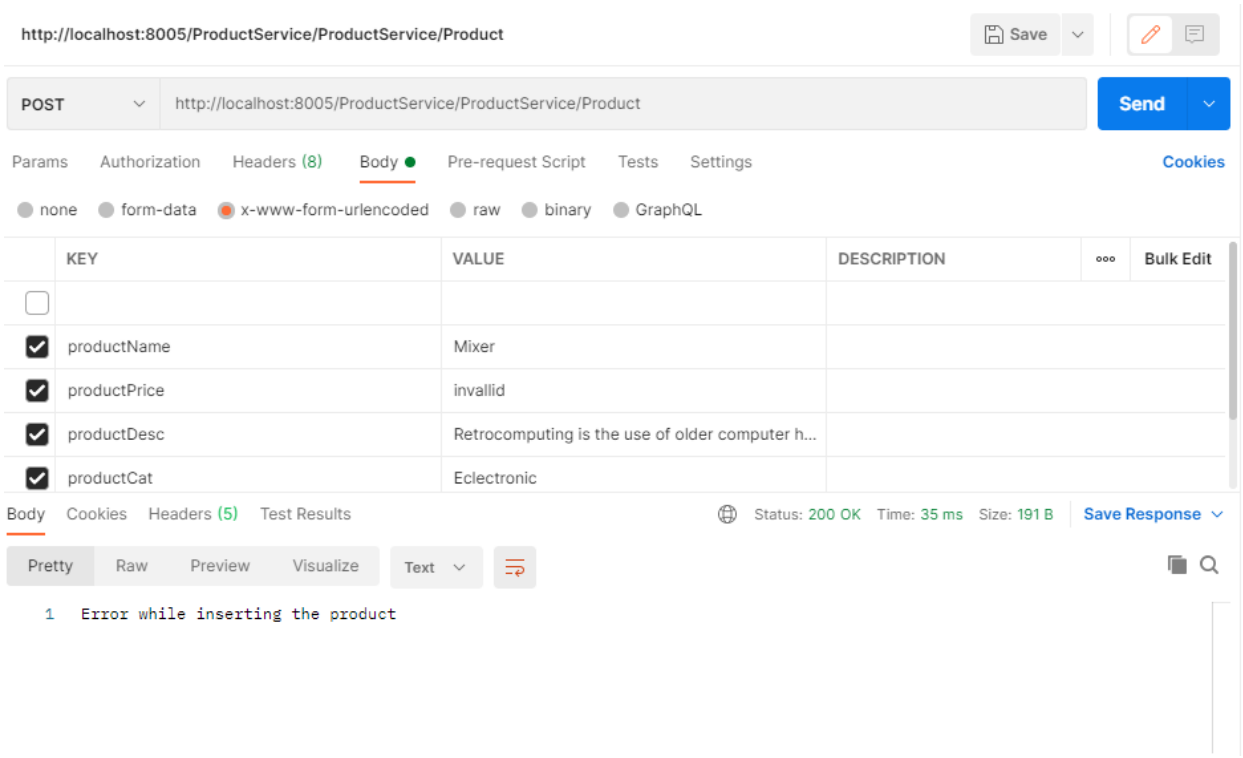


Figure 60: Invalid Insertion prompt error message - price should be Float value Test Result

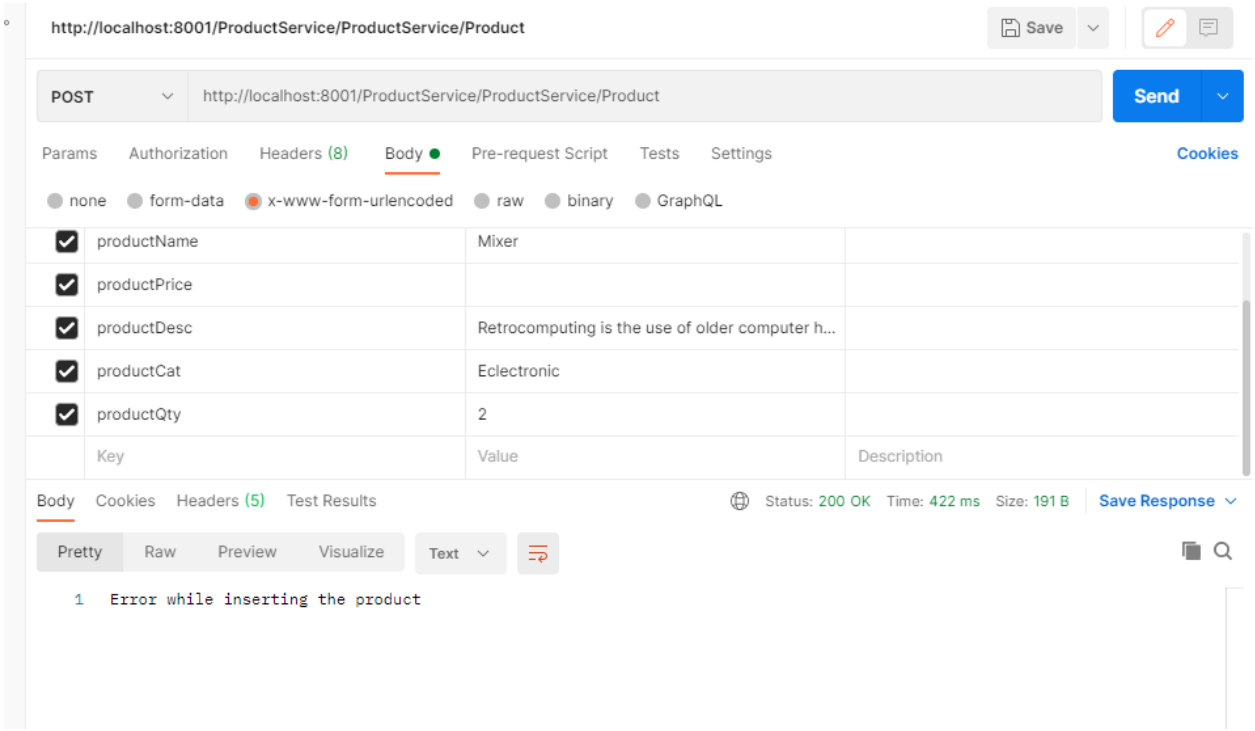


Figure 61: Prompt Error if inserted values are null Test Result

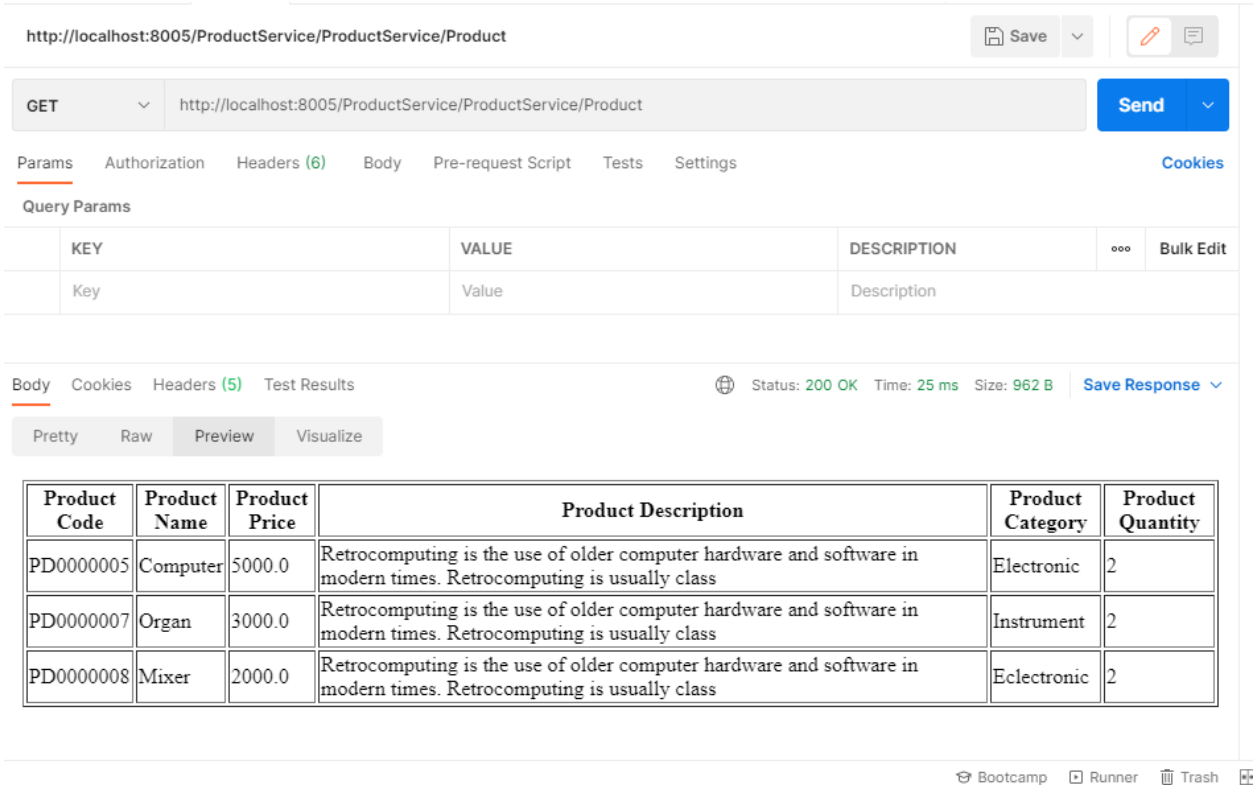


Figure 62: Retrieve All products successful Test Result

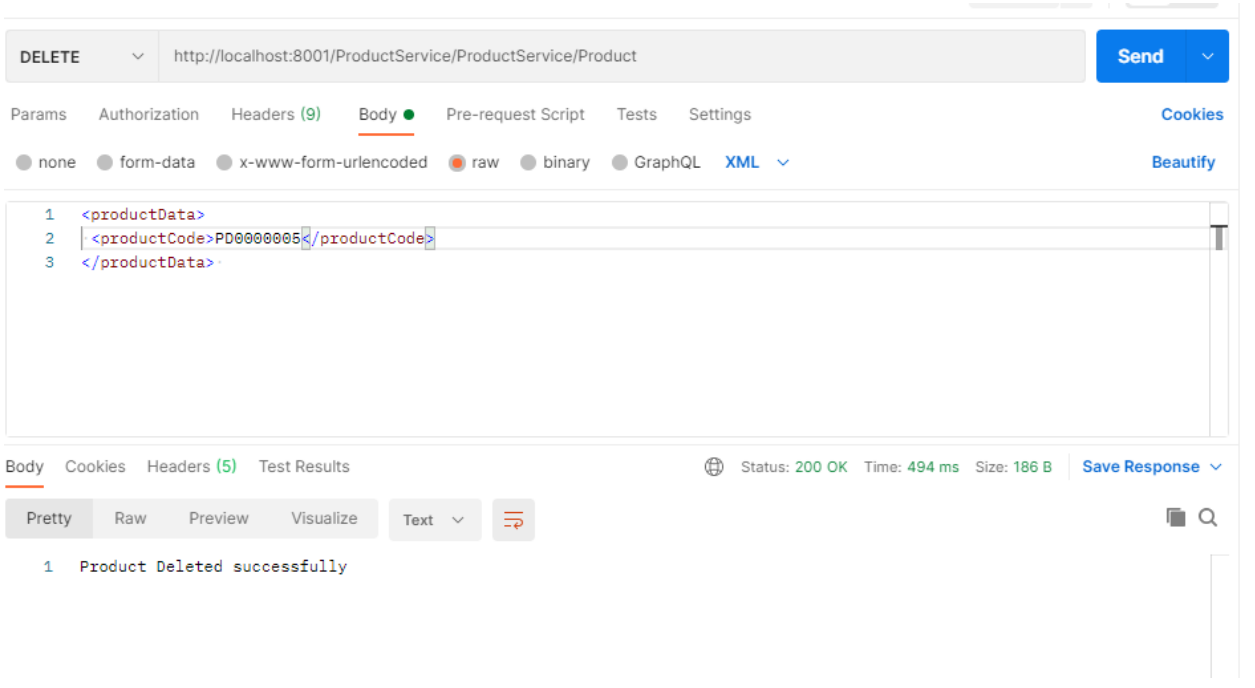


Figure 63: Delete Product by product Code Successful Test Result

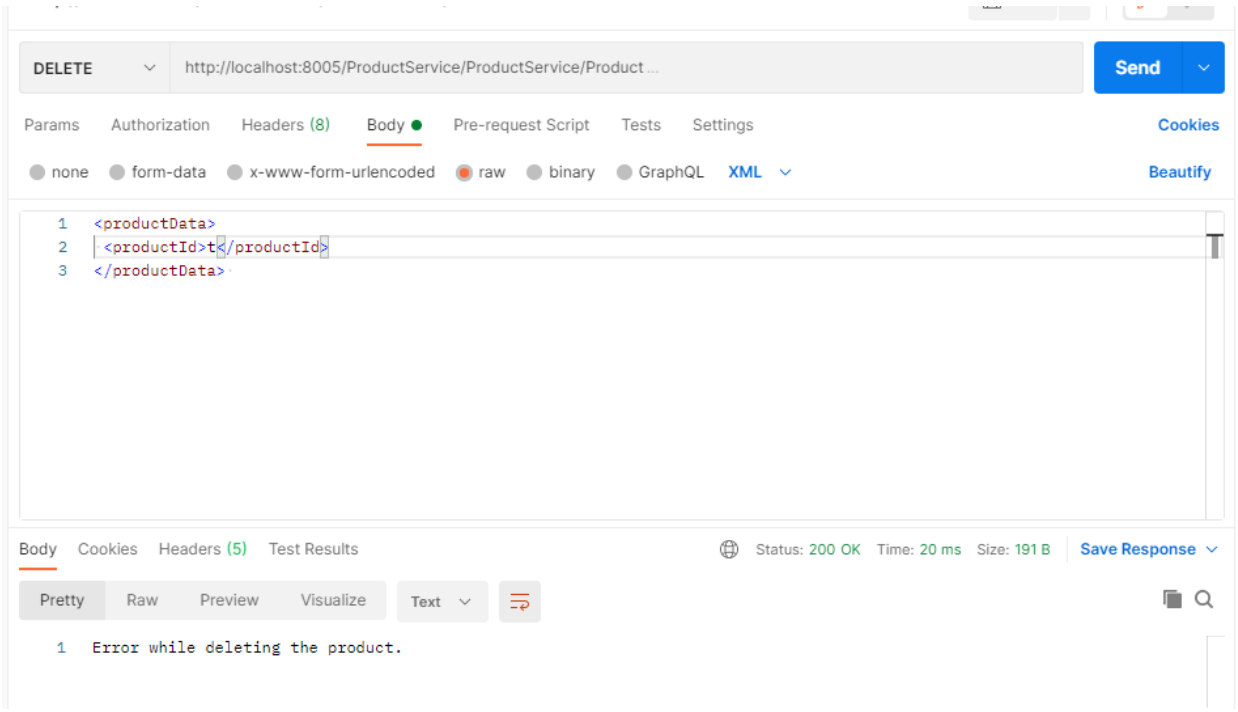


Figure 64: Invalid Delete entries Prompt unsuccessful error Test Result

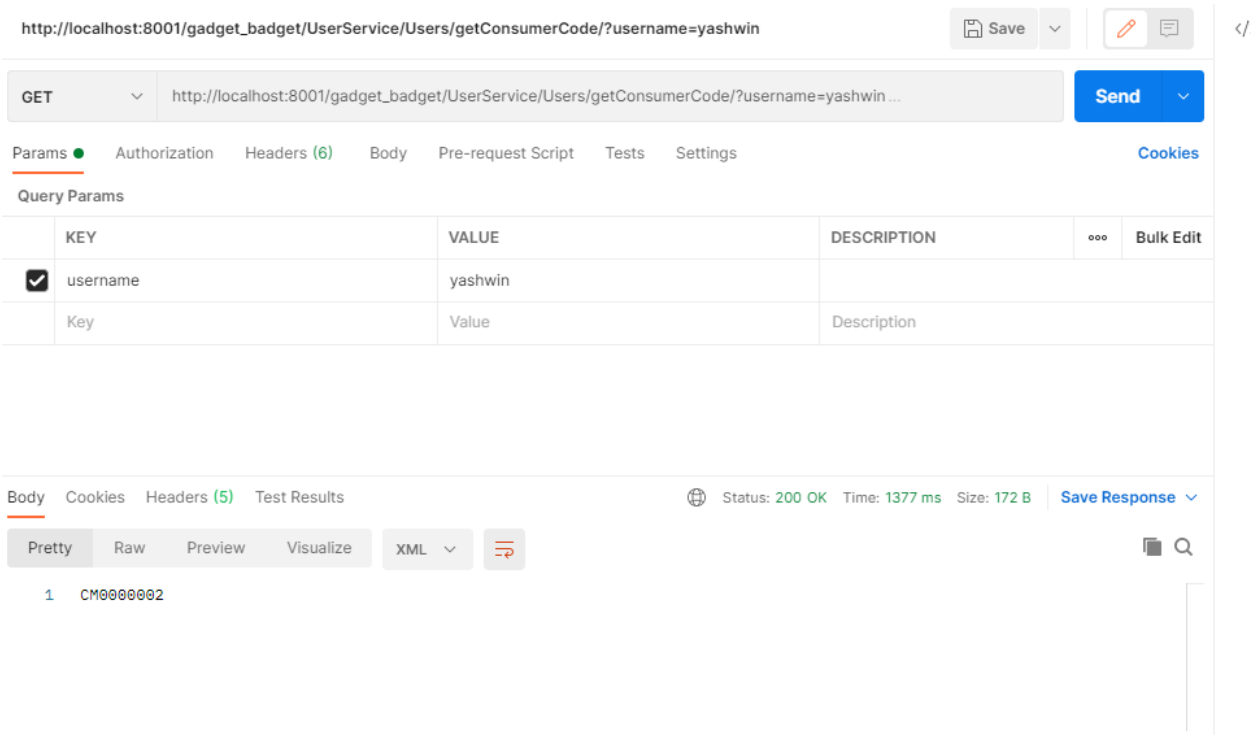


Figure 65: Retrieve consumer code from user service successful Test Result

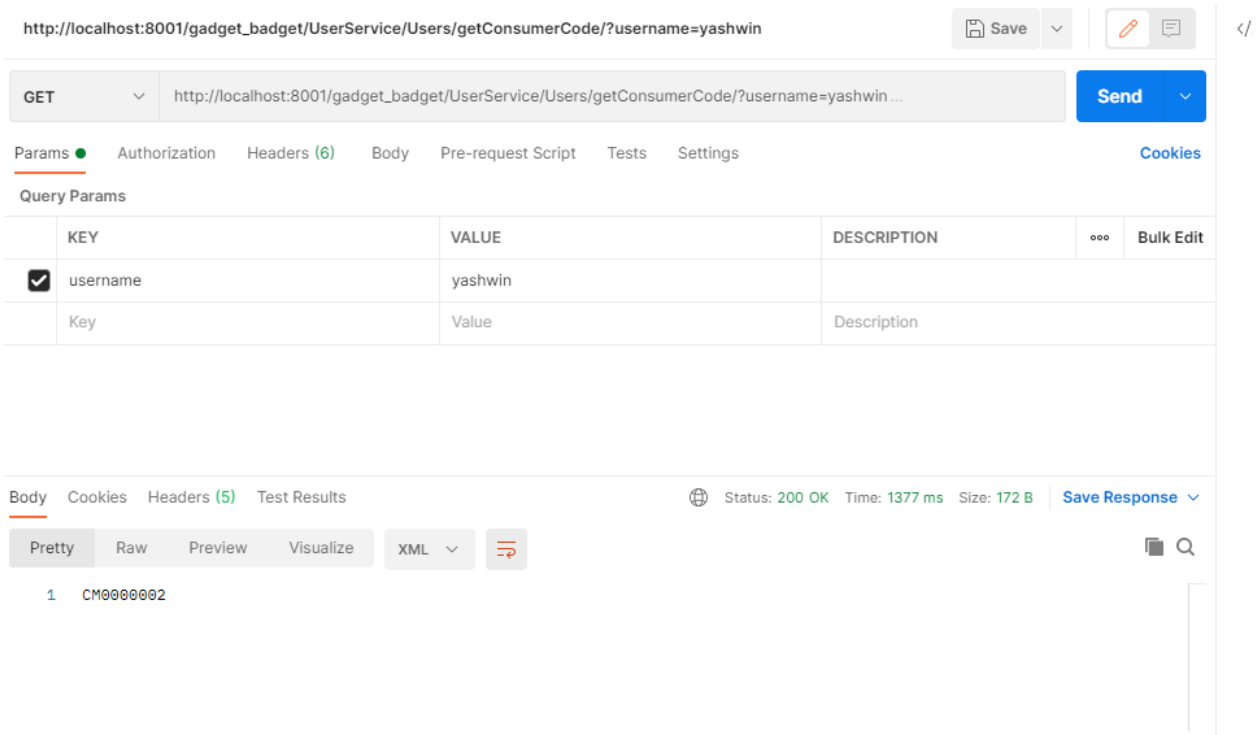


Figure 66: Prompt error while retrieving user code from user service for invalid username passed Test Result

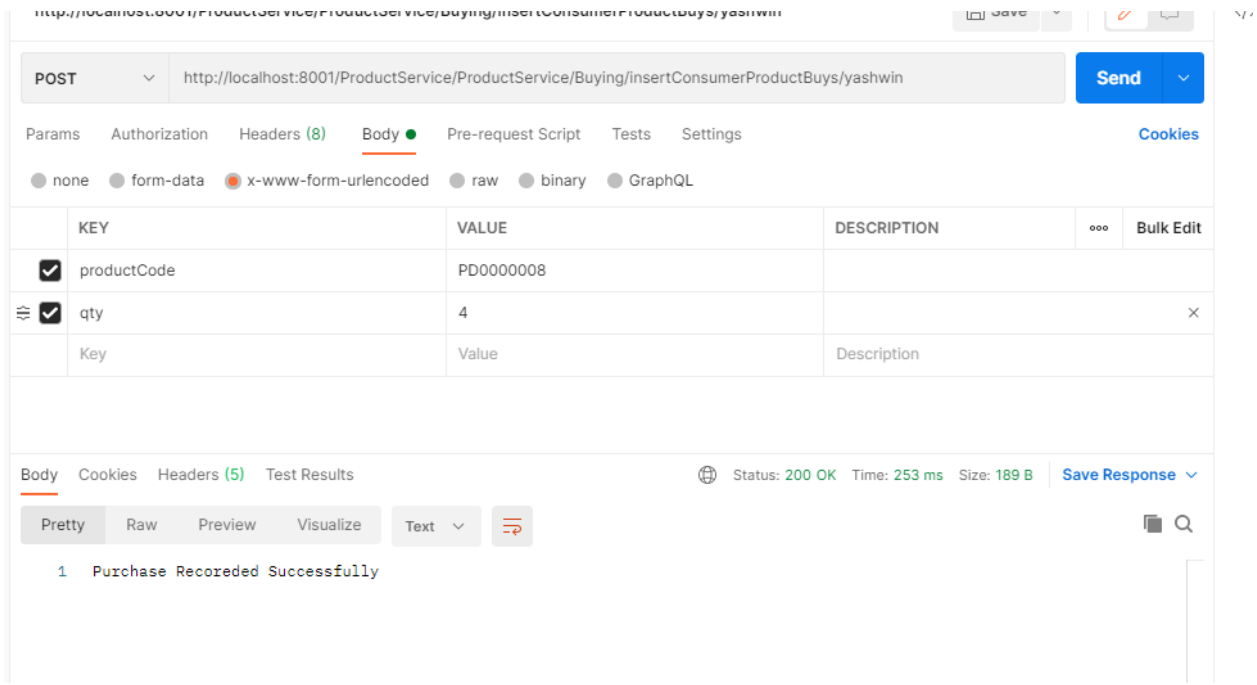


Figure 67: Retrieve user code from User Service, Insert Purchase Records Successful Test Result

Appendix E: Payment Management service - Kovishwakarunya K. IT19080840

Appendix E.1: Internal Logic diagrams

Appendix E.1.1: Class diagram

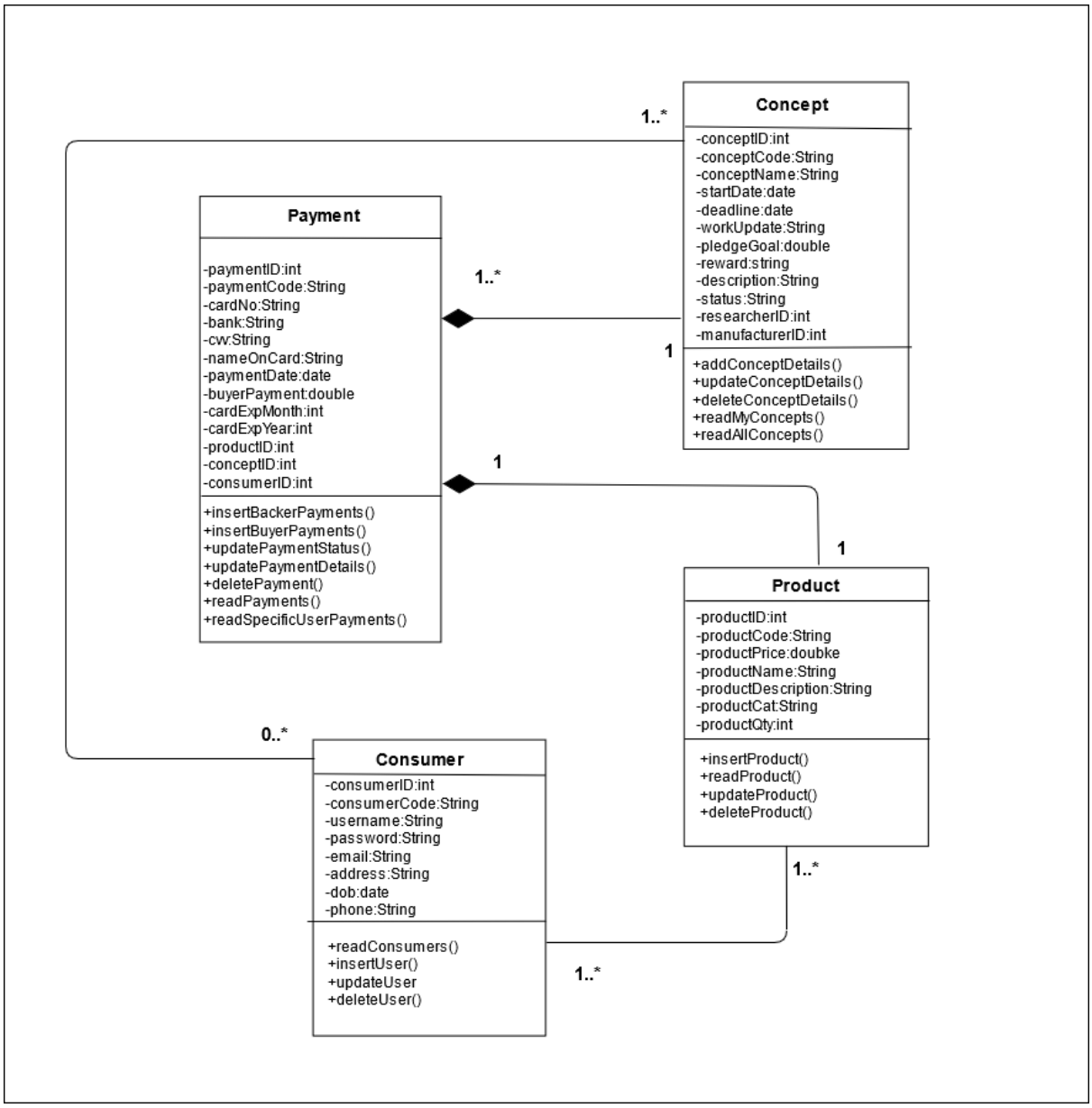


Figure 68: Payment Service Class Diagram

Payment service is operated only when a consumer buys a product or when a consumer backs a concept. Therefore, payment is a part of backing transaction or buying transaction and so is illustrated by a composition relationship to a concept and a product. Payment does not exist if buying or backing transaction does not occur. One concept can have many pledge transaction payments until the concept accomplishes its pledge Goal and to buy a particular launched project a single payment per user can be done in order to buy a quantity of projects.

Appendix E.1.2: Architecture diagram

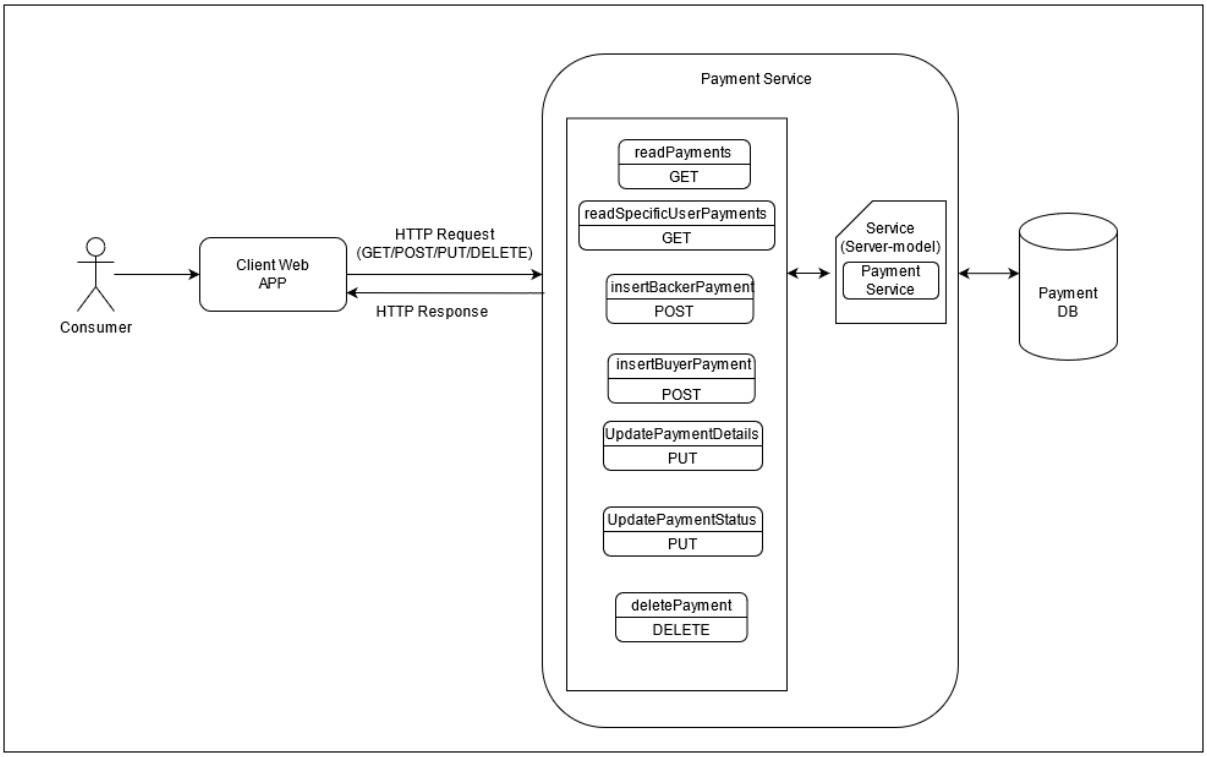


Figure 69: Payment service architecture diagram

The Payment service operations are implemented in the SERVICE section which serves as a server model to communicate with the database which stores payment data. The main resource manipulated using the HTTP verbs in this service is “Payments”. The communication on the resource within the service follows a request response pattern where client sends an HTTP request to perform CRUD operations on the resource ultimately receiving an HTTP response after being processed by the Service Server model, respectively.

Appendix E.2: Database design of service – ER

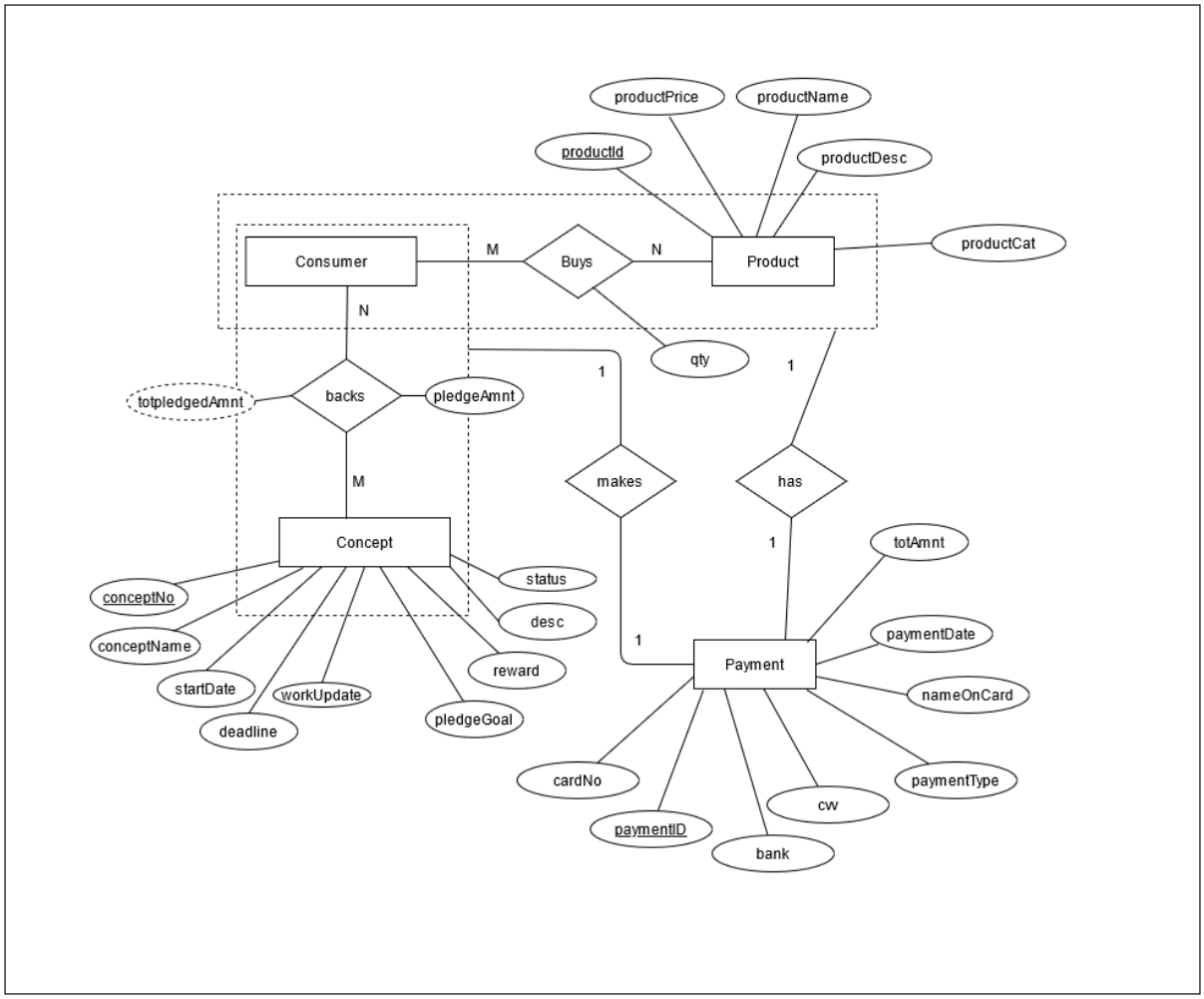


Figure 70: Payment service ER Diagram

A backer can back a concept added by a researcher by providing a pledge amount. This transaction is handled by the payment service by keeping track of the pledge Goal, pledge deadline of the concept along with total pledge amount collected for the concept up to date.

On the other hand, once a concept is completed and is launched a buyer can buy the projects. This transaction is also handled by the payment service by calculating the total buying cost considering price of product and quantity bought.

Appendix E.3: Other relevant design diagrams

Appendix E.3.1: Activity Diagram

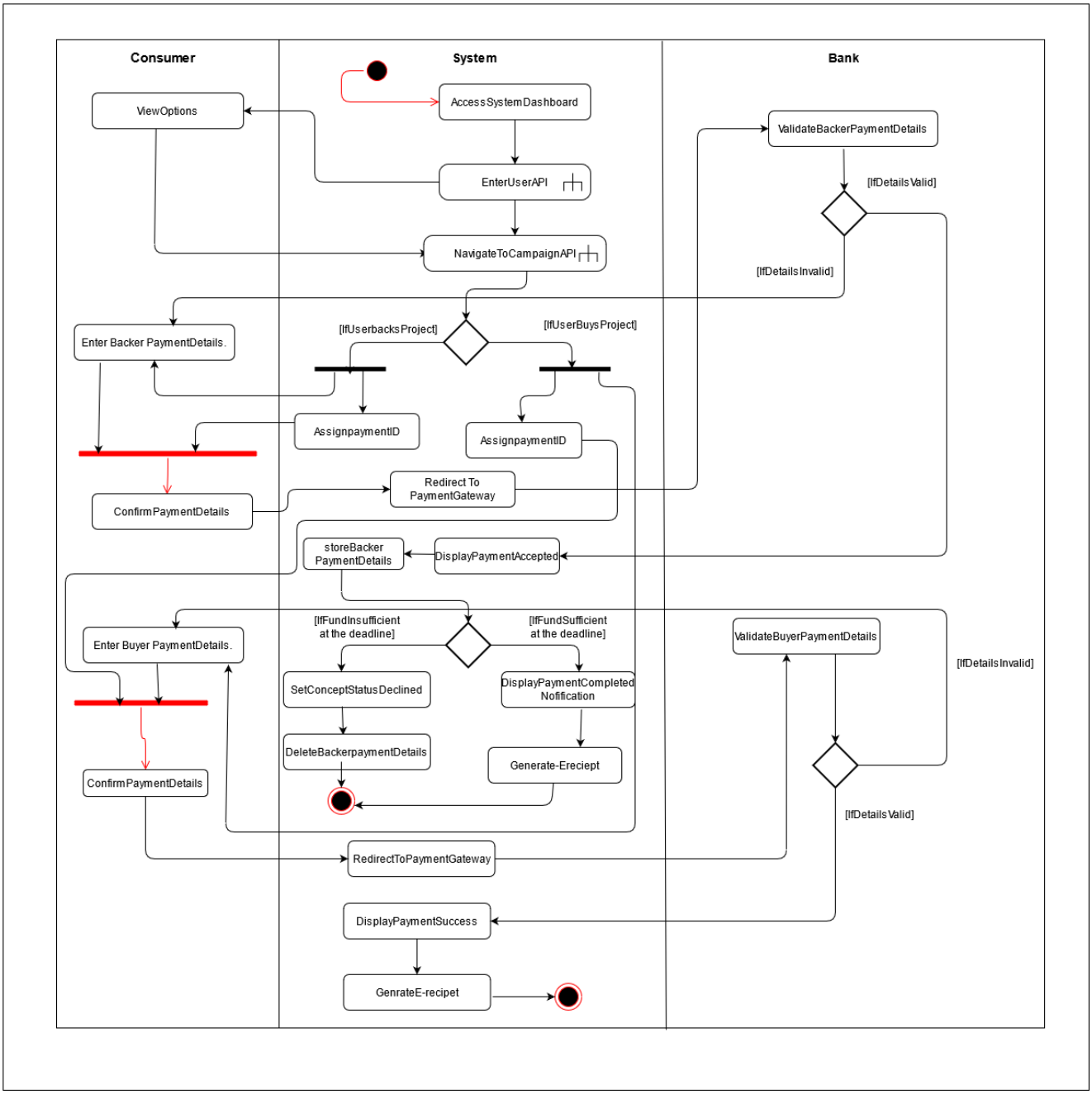


Figure 71: Payment service Activity diagram

- Two types of user payments as backer and buyer payments are handled
- Once a concept added by a researcher is backed by a user, the user will be prompted to enter the payment details where the backer can enter the card details of the payment.
- If the user payment details are valid, the user will be considered as a backer for the concept and the payment details will be stored in the database, but the transaction will not be completed until the pledge amount required for concept is collected completely.
- Time to time the total pledge amount for a particular concept is tracked.
- If the total pledge amount required by the researcher for a concept is collected within the deadline indicated by the researcher, the backer payments corresponding to that concept is validated via the payment gateway and once the transaction is complete backers will be able to view a digital receipt in their profile.

- If the pledge amount collected is not sufficient at the deadline period of the concept, the backer payment details stored will be deleted from the database informing the backers about the project incomplete status. The concept status will be updated as 'declined' ensuring no backer can back the concept further.
- Once a project is launched to be sold, buyers can buy the project by entering their transaction details finally receiving a digital receipt at the end of the transaction.

Appendix E.3.2: Use Case Diagram

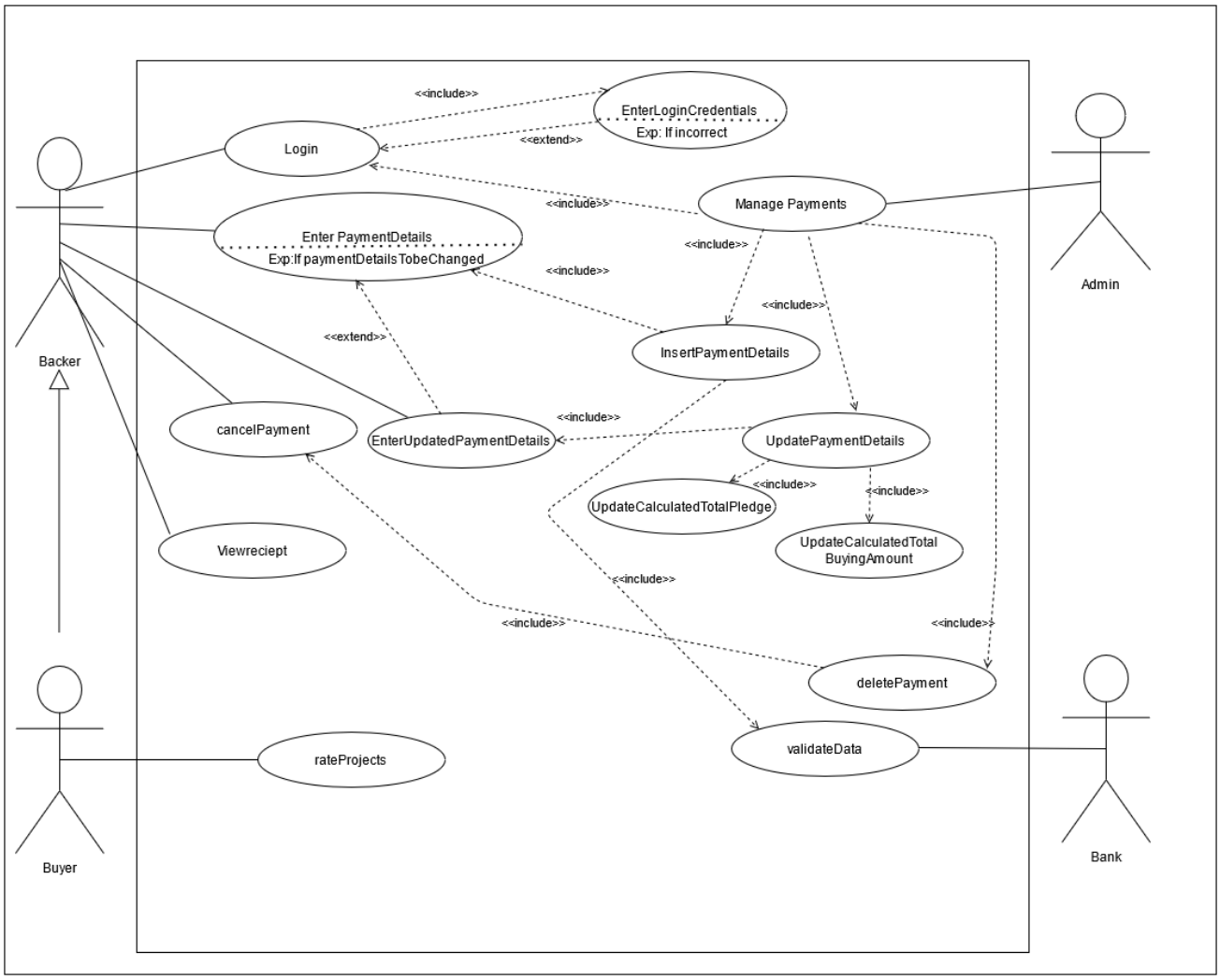


Figure 72: Payment service Use case diagram

- The actors involved in the payment service are backer, buyer, admin, and bank.
- The above diagram illustrates the various functions performed by the actors involving in the payment service.
- Buyer is generalized from the backer as all actions done by a backer can be done by a buyer too. In addition, a buyer can rate the project bought.
- The buyer and backer deal with the payment service as front-end users whereas this data is validated and handled by the bank and system admin at the backend of the system.

Appendix E.4 Testing Results

Test case ID	Description	Test input	Expected Output	Test output	Test status
1	Null values can't be inserted for any column field	Payment type = ""	"Error in inserting payment details"	"Error in inserting payment details"	Pass
2(a)	Update concept status depending on total pledge amount and pledge deadline	pledge Deadline < system date && Total pledge amount < pledge Goal	Concept status updated as "Declined"	Concept status updated as "Declined"	Pass
2(b)	Update concept status depending on total pledge amount and pledge deadline	pledge Deadline < system date && Total Pledge Amount = pledge Goal	Concept status updated as "Completed"	Concept status updated as "Completed"	Pass
2(c)	Update concept status depending on total pledge amount and pledge deadline	Pledge Deadline > system date && Total Pledge Amount = pledge Goal	Concept status updated as "Completed"	Concept status updated as "Completed"	Pass
2(d)	Update concept status depending on total pledge amount and pledge deadline	Pledge Deadline > system date && Total Pledge Amount < pledge Goal	Concept status updated as "Processing"	Concept status updated as "Processing"	Pass
3	Delete payment details of declined concepts	Concept statuses updated to "Declined"	"Backer payments deleted successfully"	"Backer payments deleted successfully"	Pass
4	Delete payment details of processing or completed concepts	Concept statuses updated to "Processing" or "Completed"	"Error in deleting payment"	"Error in deleting payment"	Pass
5	Buyer payment calculation	buyer product price= 12500 && quantity bought = 2	Insert Total Buying Cost as 12500* 2 = 25000/=	25000/= added to Buying cost column of payment	Pass
6	Update Consumer payment details	Input existing payment id details to be updated	"Payment updated successfully"	"Payment updated successfully"	Pass

7	Update Consumer payment details	Input non-existing payment id details to be updated	“Error in updating payment details”	“Error in updating payment details”	Pass
8	Buying Amount cannot be a String	Input “abc” as buying amount	“Error in inserting payment details”	“Error in inserting payment details”	Pass
9	Payment type can only be credit / debit but not cash	Input payment type = “cash”	“Please enter valid details”	“Please enter valid details”	Pass
10	No of digits in card number should be 16	Input card Number = “58167”	“Please enter valid details”	“Please enter valid details”	Pass

Appendix E.5: Postman API test result screenshots

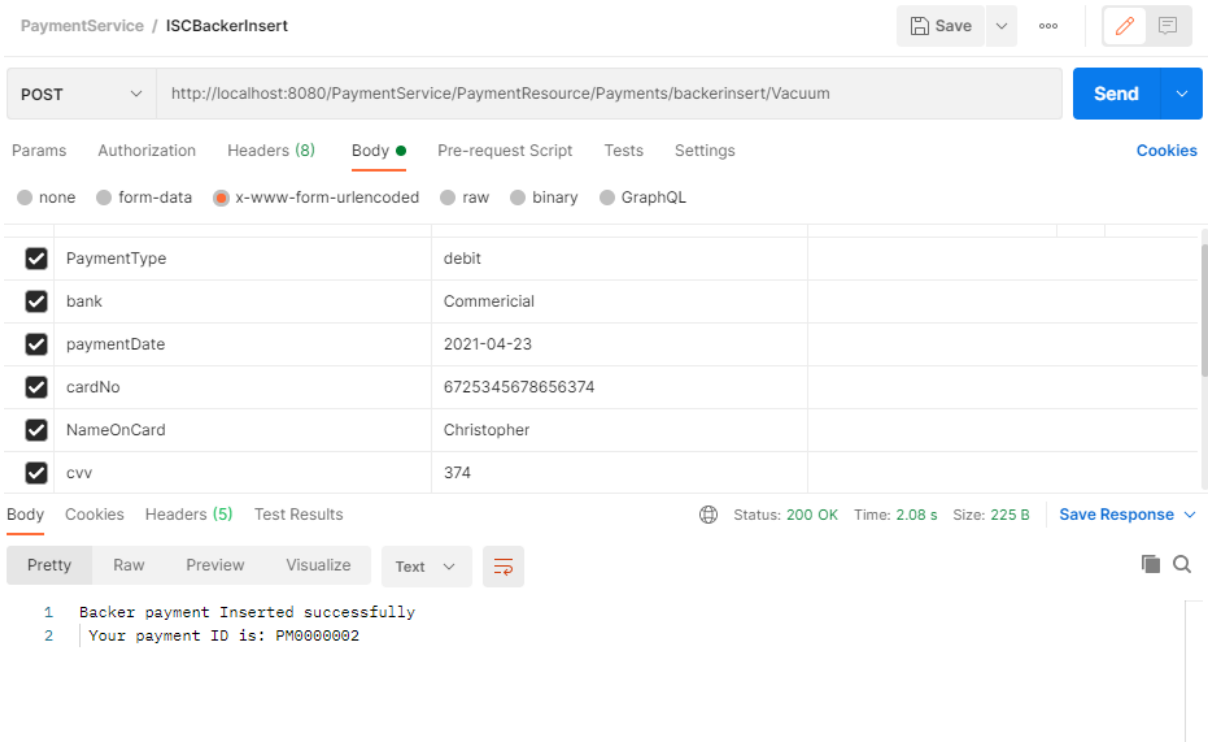


Figure 73: Insert Backer Payment Test Result

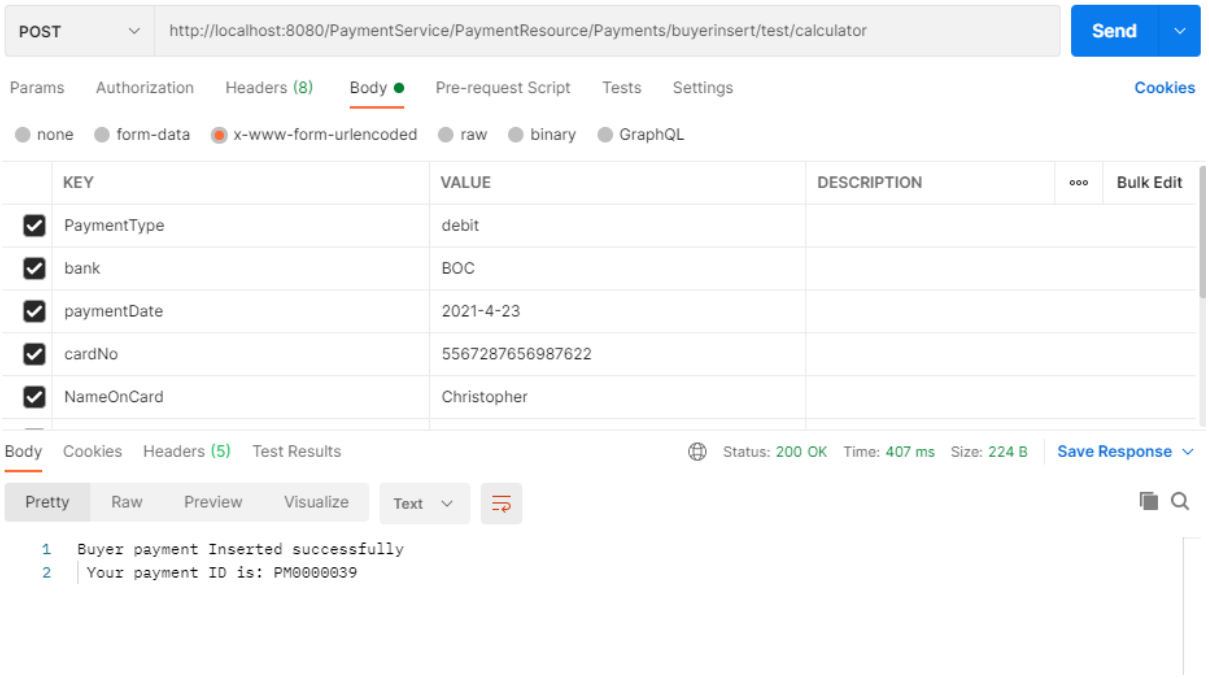


Figure 74: Insert Buyer Payment Test Result

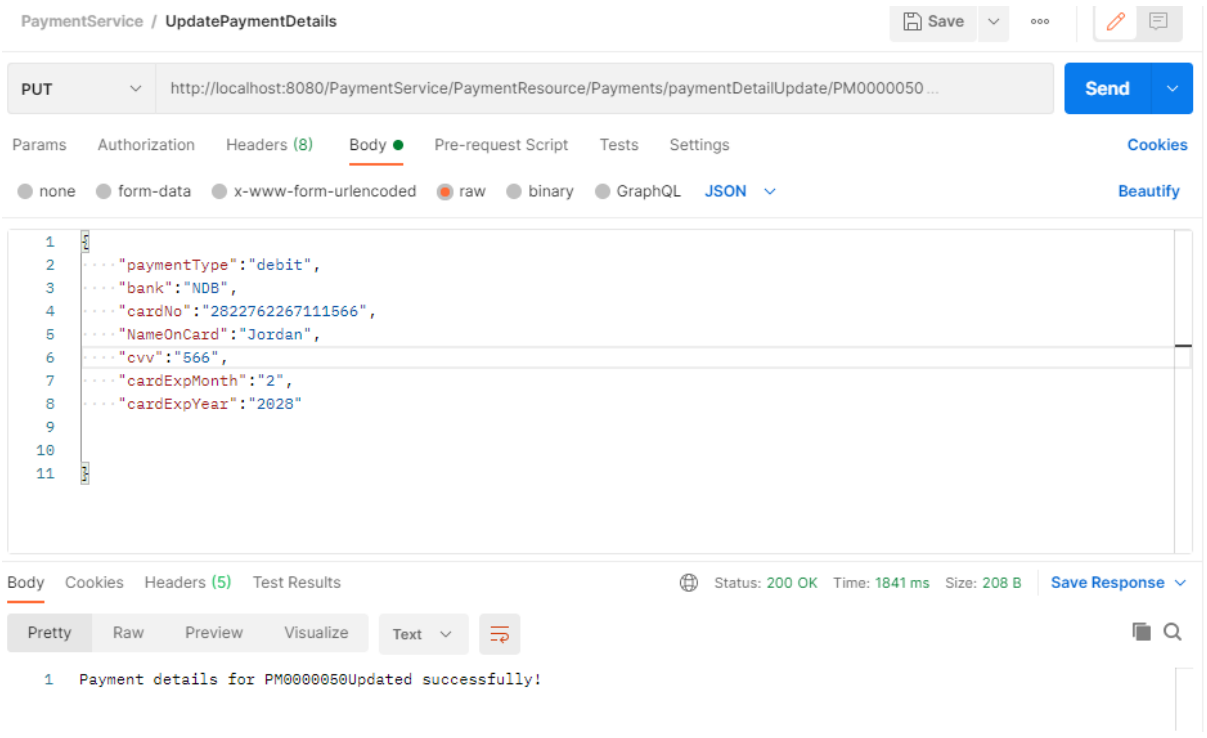


Figure 75: Update payment details update Test Result

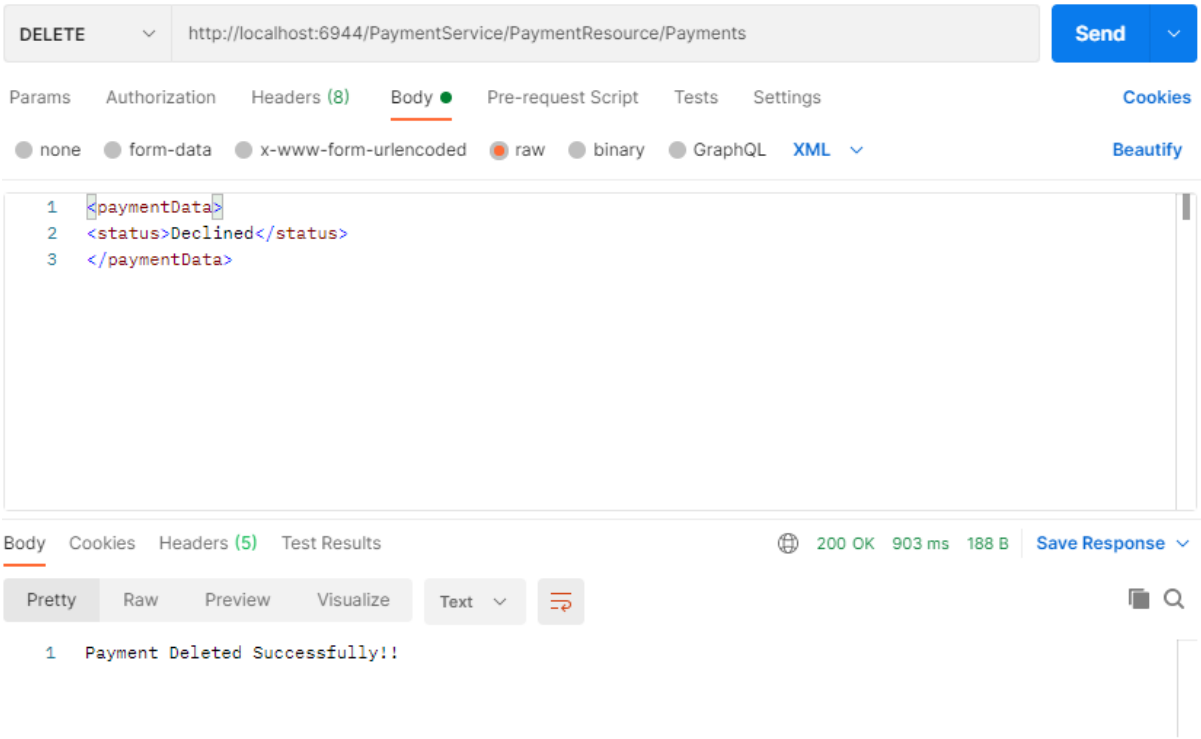


Figure 76: Delete declined concept payments Test Result

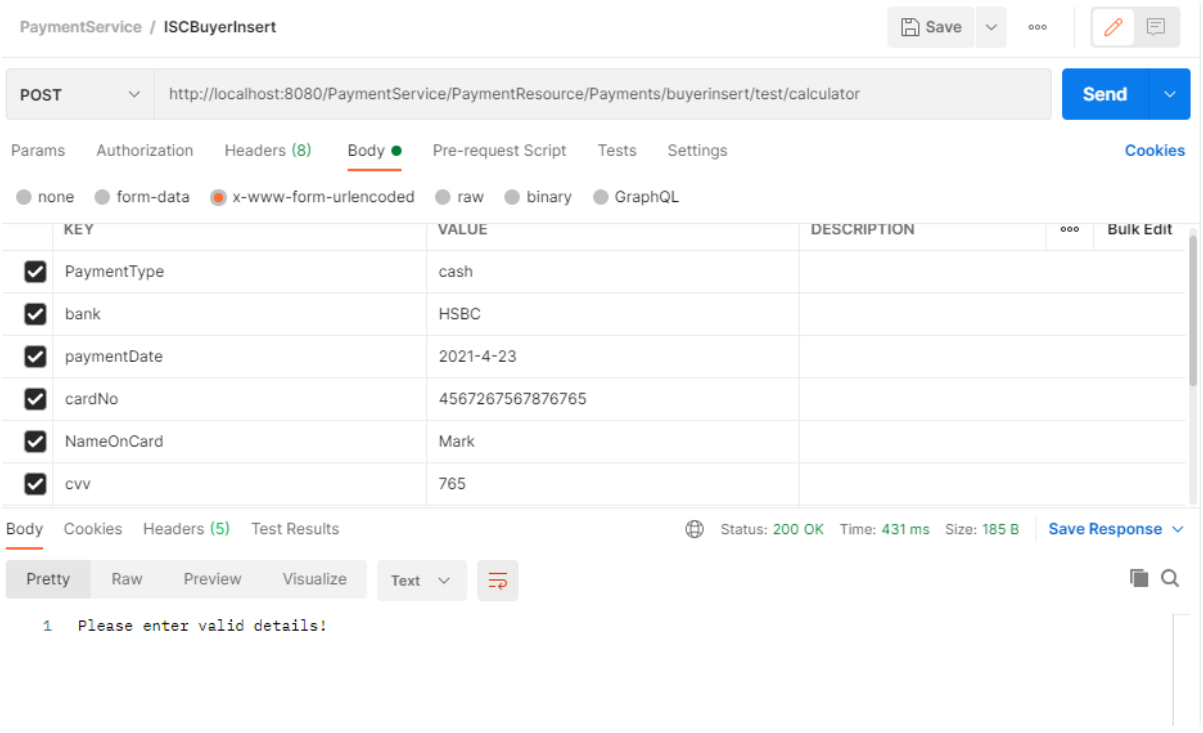


Figure 77: Validate invalid details Test Result

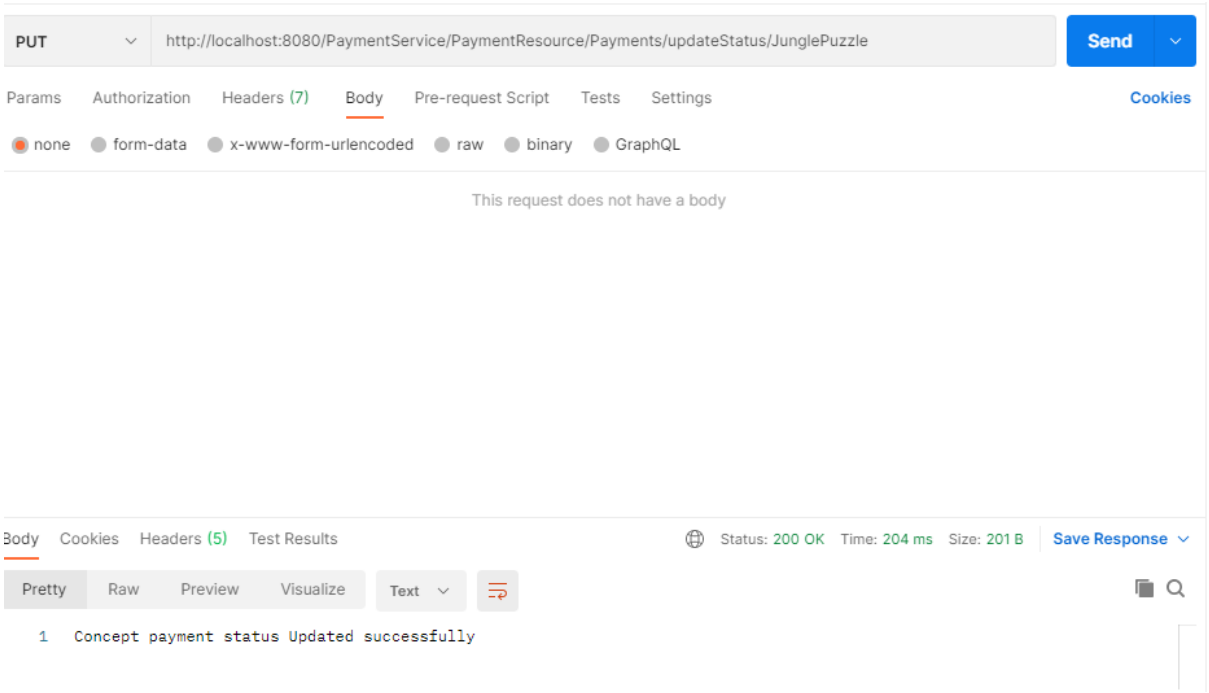
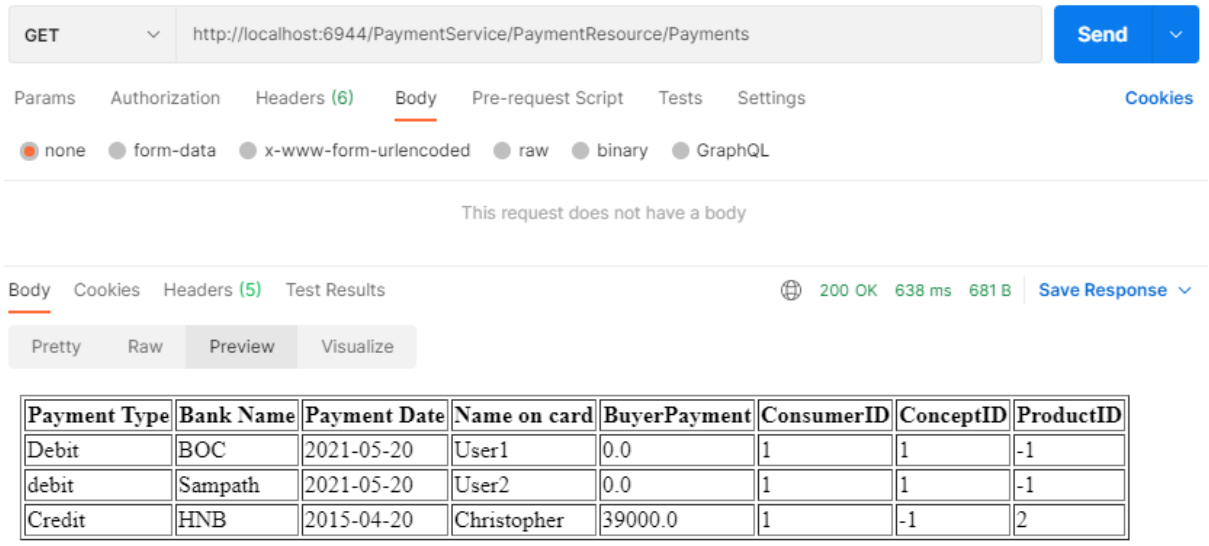
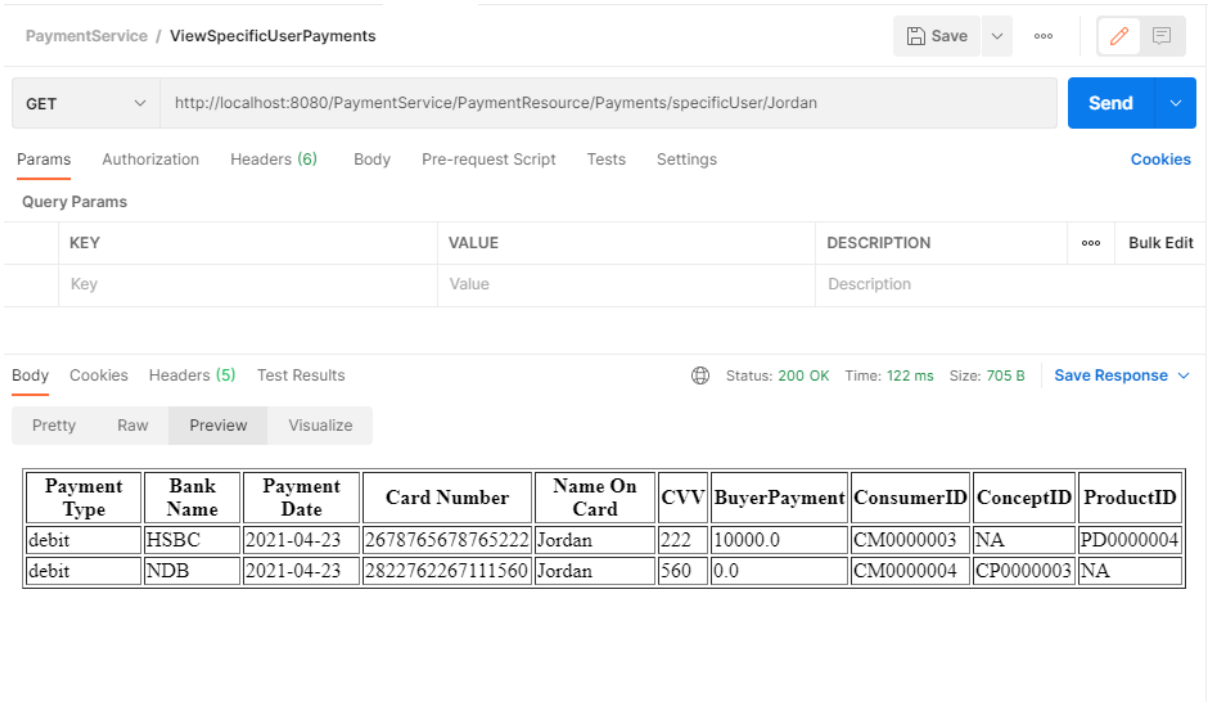


Figure 78: Update Concept status Test Result



Payment Type	Bank Name	Payment Date	Name on card	BuyerPayment	ConsumerID	ConceptID	ProductID
Debit	BOC	2021-05-20	User1	0.0	1	1	-1
debit	Sampath	2021-05-20	User2	0.0	1	1	-1
Credit	HNB	2015-04-20	Christopher	39000.0	1	-1	2

Figure 79: View all user payment details Test Result



Payment Type	Bank Name	Payment Date	Card Number	Name On Card	CVV	BuyerPayment	ConsumerID	ConceptID	ProductID
debit	HSBC	2021-04-23	2678765678765222	Jordan	222	10000.0	CM0000003	NA	PD0000004
debit	NDB	2021-04-23	2822762267111560	Jordan	560	0.0	CM0000004	CP0000003	NA

Figure 80: View Specific user payment details Test Result

```
CREATE DEFINER=`root`@`localhost` FUNCTION `insertProductAmount`(productID varchar(9) , consumerID varchar(9)) R
    DETERMINISTIC
BEGIN

    declare totalProductCost double;
    declare productqty double;
    declare productPrice double;
    declare maxPayID int;

       select b.qty into productqty
       from productservice.buying b
       where b.productID=productID AND b.consumerID = consumerID;

       select p.productPrice into productPrice
       from productservice.product p
       where p.productCode=productID ;


       set totalProductCost = productqty * productPrice ;

        return totalProductCost;

END$$
```

Figure 81: Buyer payment calculation function implementation

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `updateStatus`(conceptID varchar(9))
BEGIN
    /*Declaring variables to hold campaign values*/
        declare totalPledgeAmount double;
        declare pledgeGoal double;
        declare pledgeDeadline datetime;
        declare trackingDate datetime;
        SET trackingDate = now(); /*Getting presentDate*/

        select sum(b.pledgedAmnt) into totalPledgeAmount
        from concept_service.backs b
        where b.conceptID = conceptID;/*Summing collected pledgeAmt for given campaignID*/

        select c.pledgeGoal into pledgeGoal
        from concept_service.concept c
        where c.conceptCode = conceptID;/*Obtaining actual goal amount of researcher*/

         select c.deadline into pledgeDeadline
        from concept_service.concept c
        where c.conceptCode = conceptID;/*Obtaining actual deadline set by researcher*/

        /*Tracking deadline and pledge amount to update status*/
       IF(trackingDate < pledgeDeadline) THEN
         IF(pledgeGoal = totalPledgeAmount)THEN
        BEGIN
```

Figure 82: Update Concept Status based on pledge amount calculation Stored procedure IC1 Test Result

```
        BEGIN
            update concept_service.concept c
            set c.status = "Completed"
            where c.conceptCode = conceptID;
        END;

        elseif(pledgeGoal > totalPledgeAmount) THEN
        BEGIN
            update concept_service.concept c
            set c.status = "Processing"
            where c.conceptCode = conceptID;
         END;

        END IF;/*end of first IF*/

        elseif(trackingDate > pledgeDeadline)THEN
        IF(pledgeGoal = totalPledgeAmount)THEN
        BEGIN
            update concept_service.concept c
            set c.status = "Completed"
            where c.conceptCode = conceptID;
        END;

        elseif(pledgeGoal > totalPledgeAmount) THEN
        BEGIN
```

Figure 83: Update Concept Status based on pledge amount calculation Stored procedure IC2 Test Result

```
            set c.status = "Processing"
            where c.conceptCode = conceptID;
         END;

        END IF;/*end of first IF*/

        elseif(trackingDate > pledgeDeadline)THEN
        IF(pledgeGoal = totalPledgeAmount)THEN
        BEGIN
            update concept_service.concept c
            set c.status = "Completed"
            where c.conceptCode = conceptID;
        END;

        elseif(pledgeGoal > totalPledgeAmount) THEN
        BEGIN
            update concept_service.concept c
            set c.status = "Declined"
            where c.conceptCode = conceptID;
         END;
        END IF;
        END IF;
END$$
DELIMITER ;
```

Figure 84: Update Concept Status based on pledge amount calculation Stored procedure IC3 Test Result

References

Anon., 2021. *10 Reasons to Use Agile Software Development.* [Online]
Available at: https://www.qualitylogic.com/2019/07/18/10-reasons-to-use-agile-software-development/#:~:text=Well%20executed%20Agile%20software%20development,results%20in%20a%20working%20product.

docs.oracle.com, 2021. *Building RESTful Web Services with JAX-RS.* [Online]
Available at: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp1v/index.html#:~:text=JAX%2DRS%20is%20a%20Java,development%20of%20RESTful%20web%20services.
[Accessed 25 04 2021].

Guru99, 2021. *Agile Methodology: What is Agile Software Development Model?.* [Online]
Available at: https://www.guru99.com/agile-scrum-extreme-testing.html

TATVASOFT, 2021. *Introduction to SonarQube & SonarLint.* [Online]
Available at: https://www.tatvasoft.com/blog/introduction-to-sonarqube-sonarlint/
[Accessed 25 04 2021].

wrike.com, 2021. *Why Use Agile Project Management?.* [Online]
Available at: https://www.wrike.com/project-management-guide/faq/why-use-agile-project-management/
[Accessed 25 04 2021].