A RESTful API for Online Movie Service

Project report

* Abraham Alabi (x17342373)
* Jegan Francis (x18131212)
* Tijesu Olalekan (x17347773)
* Ting Hao Chang (x16370076)

Group k

December 18, 2020

Contents

[Introduction 3](#_Toc59178740)

[Problem Domain 3](#_Toc59178741)

[Customer 3](#_Toc59178742)

[Account 3](#_Toc59178743)

[Movie 3](#_Toc59178744)

[Create 3](#_Toc59178745)

[Add Movie 3](#_Toc59178746)

[Transfer Movie 3](#_Toc59178747)

[Remove Movie 3](#_Toc59178748)

[List Movie 3](#_Toc59178749)

[Entity Relationship Diagram 4](#_Toc59178750)

[Security Concerns 5](#_Toc59178751)

[Hard Code - Rule 49 - MSC03-J - Never hard code sensitive information 5](#_Toc59178752)

[Private Variables - Rule 49 - MSC07-j – Prevent multiple instantiations of singleton objects 5](#_Toc59178753)

[Password Encryption – Rule 49 - MSC62-J – Store passwords using a hash function 5](#_Toc59178754)

[Sanitization - Rule 00 - IDS03-J – Do not log unsanitized user input 5](#_Toc59178755)

[Team Members contribution section 6](#_Toc59178756)

[Abraham – Contribution 25% 6](#_Toc59178757)

[Jegan – Contribution 25% 6](#_Toc59178758)

[Tijesu – Contribution 25% 6](#_Toc59178759)

[Ting Hao – Contribution 25% 6](#_Toc59178760)

[RESTful API 7](#_Toc59178761)

[1.) Customer Resource 7](#_Toc59178762)

[Method: getAllCustomer 7](#_Toc59178763)

[Method: getCustomer 7](#_Toc59178764)

[Method: addCustomer 7](#_Toc59178765)

[Method: removeCustomer 8](#_Toc59178766)

[Method:getCustomerAccounts 8](#_Toc59178767)

[Method:getCustomerAccountMovies 8](#_Toc59178768)

[2.) Account Resource 9](#_Toc59178769)

[Method: getAllAccount 9](#_Toc59178770)

[Method: getAccount 9](#_Toc59178771)

[Method: getAccountNickname 9](#_Toc59178772)

[Method: getMovies 10](#_Toc59178773)

[Method: removeAccount 10](#_Toc59178774)

[3.) Movie Resource 11](#_Toc59178775)

[Method: getAllMovies 11](#_Toc59178776)

[Method: getMovie 11](#_Toc59178777)

[Method: addMovie 11](#_Toc59178778)

[Method: removeMovie 12](#_Toc59178779)

[Method: transferMovie 12](#_Toc59178780)

[Web Service Operation Section 13](#_Toc59178781)

[1.) Creating new customer using POST request 13](#_Toc59178782)

[2.) Adding a new movie to an account using POST request 14](#_Toc59178783)

[3.) Transfer a movie to another account hold by the customer 15](#_Toc59178784)

[4.) Remove a movie from account by using Id 16](#_Toc59178785)

[5.) Making a GET request to retrieve a movie list of a customer’s account by Id 17](#_Toc59178786)

Web Services and API

# Introduction

This main idea of this document is to determine what solution we have implemented to solve this problem domain for Online Movie Service. It showcase the textual description of the problem, an Entity-Relationship diagram describing the problem domain and security concern across the whole choosen API in relation to the project.

## Problem Domain

Our chosen name of the project is called AJTTMovie, and the main focus of this project is to design, document and develop of an API and to build a Client that would utilize the API which would grant the customer to run Online Movie Services.

For an Online Movie system, the requirement would be these:

## Customer

We would create a class that would grant us to have the customer as the individuals with such details: Name, Address, Email and Unique customer ID and could also hold one or more accounts. That would allow us to retrieve the customer ID to construct a new customer, update an existing customer and grant us to delete the customer’s account too.

## Account

For the Account class, we implemented the following functions that would grant user to construct a new account and that could delete customer account too. That includes its own unique account ID number, Nickname and login password. For the customer account class, there would be allow to add and remove movies to the list, transfer a specific movie from one account to another known account, and list of movies that already exist.

## Movie

In the Movie Class, we would start by generate the Get all movies method then the Get movies by account number, followed by the Get watched by customer account number, then by recommended by the customer account number and then the create movie and the watched or is recommended.

Customers will able to do the following:

Create – Customers are permitted to be able to make an account for a family member. A minimum of one account per customer plus supplementary accounts such as a family account that usually have one account per member of the family e.g Adults and Childrens.

Add Movie – The customers must be able to add a new recommended movie to the list of moves that links to a customer account.

Transfer Movie – Customers can specify a distinct movie to be moved from one (current) account to another account that was maintain by the customer.

Remove Movie – The customers can withdraw which particular movie they so choose from their own list of movies from an account.

List Movie – Customers should be able to request a list of movies on any account at any time.

## Entity Relationship Diagram

## Security Concerns

As we continue developing this Online Movie Service project, we ensure that the appropriate java access modifers will implement the best practice for secure coding, correct any existing non-compliant code. Following SEI CERT Coding Standard for Java such as these:

* Rule 49 - MSC03-J
* Rule 49 - MSC07-J
* Rule 49 - MSC62-J
* Rule 00 - IDS03-J

### Hard Code - Rule 49 - MSC03-J - Never hard code sensitive information

In order to maintain data integrity our encryption keys or password of our customer account will never be hard coded which could led to expose these sensitive information to external attacks. Anyone who has entry to the executable or dynamic library files will be cross-check for any vulnebilites as these role have higher authority that could be used in malicious intention.

### Private Variables - Rule 49 - MSC07-j – Prevent multiple instantiations of singleton objects

To mitigate any potential damage of an attacker being able to access to unauthorized code area and modify the source code of the website through the front-end by setting specific methods to a class. Making sure that constructor private, ensuring the class is not serializable and

### Password Encryption – Rule 49 - MSC62-J – Store passwords using a hash function

By storing password as cleartext or unencrypted text data would have vastly risky exposure of those passwords which would invite unnecessary trouble that would result in leaking delicate information.

In order to avoid this, through the use of hashing the passwords that ultilizes a mathematical algorithim to encode the password to a hash value which decoding remains infeasible as it would be unrecognizable. Even at the event of a data breach, the password would still remain undecipherable since without the original hash value.

For extra security and as a good practice is that, a salt can be supplementary to the password being hashed. A salt is a unique randomly produce part of data that is stored and used to create the hash value along with the password.

We ran out of time to implement this to our code.

### Sanitization - Rule 00 - IDS03-J – Do not log unsanitized user input

To avoid log injection vulnerability or SQL Injection attacks in general, where the attacker might inject a script into a log file or input data like a text-box to consequently deceive system administrators bypass to gain control as a user or as the administrator

# Team Members contribution section

## Abraham – Contribution 25%

Abraham’s role was a backend developer. He aided in implementing models, resources, services and databases. He worked on base structure of the code and fixed connection issues with Postman and this document and presentation.

## Jegan – Contribution 25%

Jegan’s role was a backend developer. He aided in implementing models, resources, services and databases. He worked on testing all connections on Postman are working as intended and base structure of the code, and this document and presentation.

## Tijesu – Contribution 25%

Tijesu’s role was a backend developer. He aided in implementing models, resources, services and databases. He worked extensively on optimizing the codes, logics and this document and presentation.

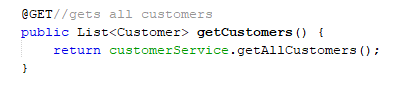
## Ting Hao – Contribution 25%

Ting Hao’s role was a backend developer. He aided in implementing models, resources, services and databases. He worked on base structure of the code, creating the presentation and finalizing this document.

# RESTful API

## Customer Resource

### Method: getAllCustomer

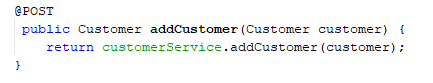


|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | Gets all customers made from a customer |
| URI | /customer |
| HTTP VERB | @GET |
| Parameters | none |
| Resource Content | 204 no content |
| Pre-Conditions | The system is in a wait state |
| Post-Conditions | Returns customerService object to getAllcustomers() |

### Method: getCustomer

|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | Allows the retrieval of a customer by an Id number |
| URI | /customer/{customerId} |
| HTTP VERB | @GET |
| Parameters | customerId (Integer, Required) |
| Resource Content | Example customerId: “8” |
| Pre-Conditions | A customer with the specified customer Id must exist |
| Post-Conditions | The customerService Object with customerId number is returned |

### Method: addCustomer

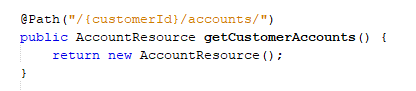


|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | Add customer to account |
| URI | /customer |
| HTTP VERB | @POST |
| Parameters | Customer customer (Customer Object) |
| Resource Content | Customer details (name,address,email,customerId) |
| Pre-Conditions | The system is in a wait state |
| Post-Conditions | A customer is created in the database with the customer object specfied |

### Method: removeCustomer

|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | remove customer by customer Id |
| URI | /customer/{customerId} |
| HTTP VERB | @PUT |
| Parameters | Id (Integer Required) |
| Resource Content | Example customerId: “3” |
| Pre-Conditions | The customer must exist in the database |
| Post-Conditions | An customer is remove from database with the customer Id |

### Method:getCustomerAccounts



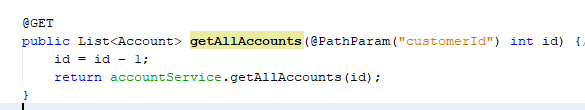
|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | Get customer accounts |
| URI | /customer/{customerId}/accounts |
| HTTP VERB |  |
| Parameters |  |
| Resource Content |  |
| Pre-Conditions | The account must exist in the database |
| Post-Conditions | Return AccountResource |

### Method:getCustomerAccountMovies

|  |  |
| --- | --- |
| API Name | CustomerResource |
| Description | Get customer account movies |
| URI | /customer/{customerId}/accounts/{accountId}/movies |
| HTTP VERB |  |
| Parameters |  |
| Resource Content |  |
| Pre-Conditions | The customer must exist in the database |
| Post-Conditions | Return MovieResource |

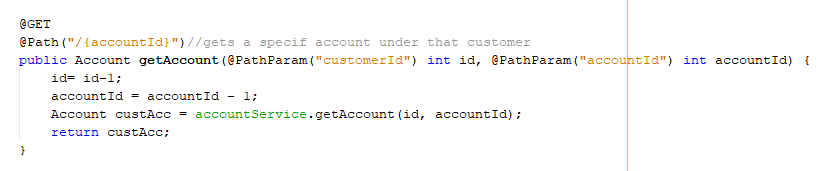
## Account Resource

### Method: getAllAccount



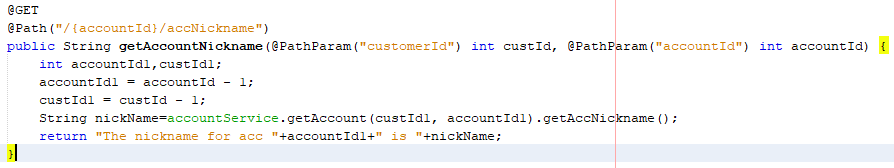
|  |  |
| --- | --- |
| API Name | AccountResource |
| Description | Gets all accounts by each specified customer Id |
| URI | /account |
| HTTP VERB | @GET |
| Parameters | customerId(Integer,Required) |
| Resource Content | Example customer Id: 111 |
| Pre-Conditions | A customer must exist with the specified customer Id |
| Post-Conditions | Returns accountService object to getAllAccounts() |

### Method: getAccount



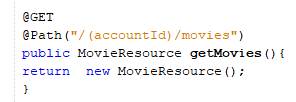
|  |  |
| --- | --- |
| API Name | AccountResource |
| Description | Allow to retrieve an account with a specified customerId and accountId |
| URI | /account/{accountId} |
| HTTP VERB | @GET |
| Parameters | customerId and accountId (Integer,Required) |
| Resource Content | Example customer Id: 111 and accountId: 222 |
| Pre-Conditions | A customer with the specified customerId with an account that have an unique accountId must exist |
| Post-Conditions | Return accountService object to getAccounts (id, accountId) as custAcc |

### Method: getAccountNickname



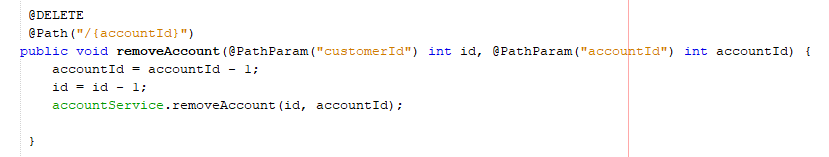
|  |  |
| --- | --- |
| API Name | AccountResource |
| Description | Allows to retrieve the Account Nickname from a specifed customerId and accountId |
| URI | /account/{accountId}/accNickname |
| HTTP VERB | @GET |
| Parameters | customerId and accountId (Integer,Required) |
| Resource Content | Example customer Id: 111 and accountId: 222 |
| Pre-Conditions | A customer with the specified customerId with an account that have an unique accountId must exist |
| Post-Conditions | Returns the account nickname |

### Method: getMovies



|  |  |
| --- | --- |
| API Name | AccountResource |
| Description | Allow to retrieve the movie from the account |
| URI | /account/{accountId}/movies |
| HTTP VERB | @GET |
| Parameters | Movie details (movie Id, movieName,isWatch, Summary) |
| Resource Content |  |
| Pre-Conditions | An account with a specifed account Id must exist/ |
| Post-Conditions | Returns the movie |

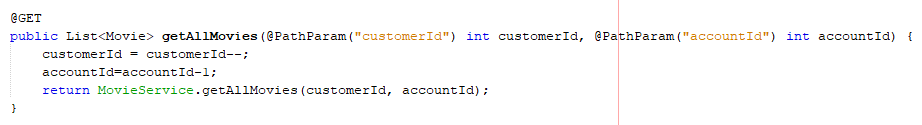
### Method: removeAccount



|  |  |
| --- | --- |
| API Name | AccountResource |
| Description | To remove account with a specified customer Id, and account Id |
| URI | /account/{accountId}/ |
| HTTP VERB | @DELETE |
| Parameters | CustomerId (Integer, Required) accountId (Integer, Required) |
| Resource Content |  |
| Pre-Conditions | Customer Id and its account Id must exist in the database already |
| Post-Conditions | An account is deleted from the database with customer Id and account Id |

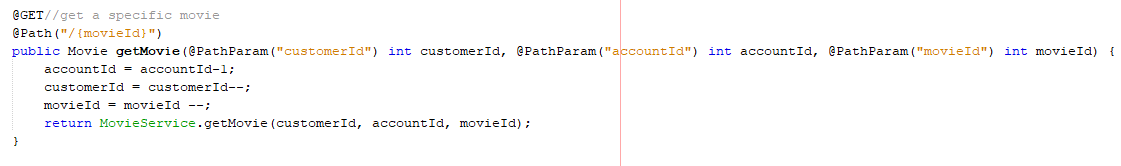
## Movie Resource

### Method: getAllMovies



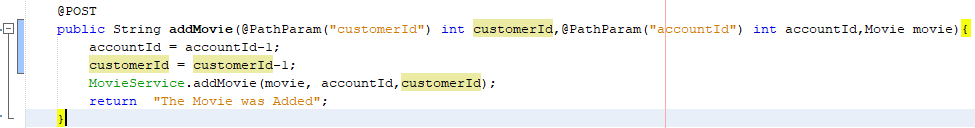
|  |  |
| --- | --- |
| API Name | MovieResource |
| Description | Gets all movies with a specfied customerId and accountId |
| URI | /movie |
| HTTP VERB | @GET |
| Parameters | customerId(Integer Require), accountId(Integer Require) |
| Resource Content | customerId = “13” accountId = “3” |
| Pre-Conditions | A customer and that customer’s account must exist in the database |
| Post-Conditions | Returns all the movies of that specified customer’s account |

### Method: getMovie



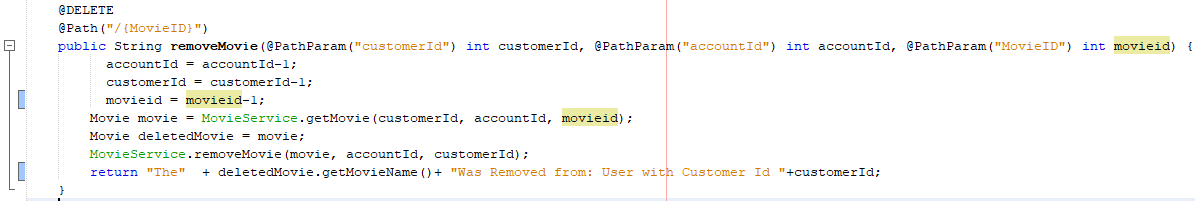
|  |  |
| --- | --- |
| API Name | MovieResource |
| Description | Get a specific movie from a particular customer and its account |
| URI | /movie/{movieId} |
| HTTP VERB | @GET |
| Parameters | All customerId, accountId, and movieId (Integer Require) |
| Resource Content | customerId = “13” accountId = “3” movieId = “10” |
| Pre-Conditions | A customer and that customer’s account must exist in the database |
| Post-Conditions | Returns the specifed movie from that customer and its account |

### Method: addMovie



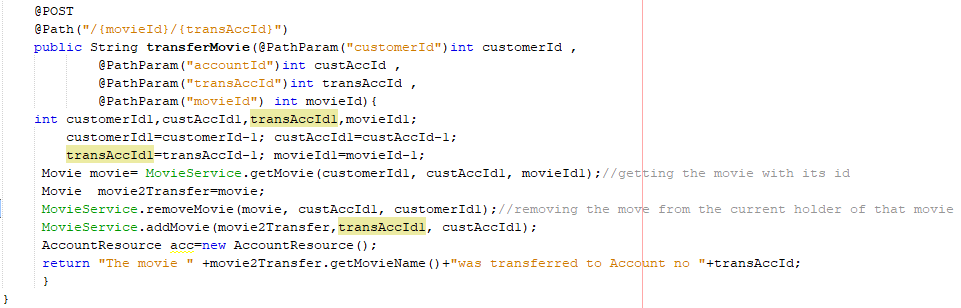
|  |  |
| --- | --- |
| API Name | MovieResource |
| Description | Add a movie to the account of the customer |
| URI | /movie |
| HTTP VERB | @POST |
| Parameters | All customerId, accountId (Integer Require) movie (Object) |
| Resource Content | customerId = “5” accountId = “1” |
| Pre-Conditions | A customer and that customer’s account must exist in the database |
| Post-Conditions | Returns the message “The Movie was Added” |

### Method: removeMovie



|  |  |
| --- | --- |
| API Name | MovieResource |
| Description | Remove a movie from the customer |
| URI | /movie/{movieId} |
| HTTP VERB | @DELETE |
| Parameters | All customerId, accountId, and movieId (Integer Require) |
| Resource Content | customerId = “12” accountId = “2” movieId = “3” |
| Pre-Conditions | A customer and that customer’s account of a movie must exist in the database |
| Post-Conditions | Returns a message “The (deleted movieId) was removed from:User with CustomerId (sender customerId) |

### Method: transferMovie



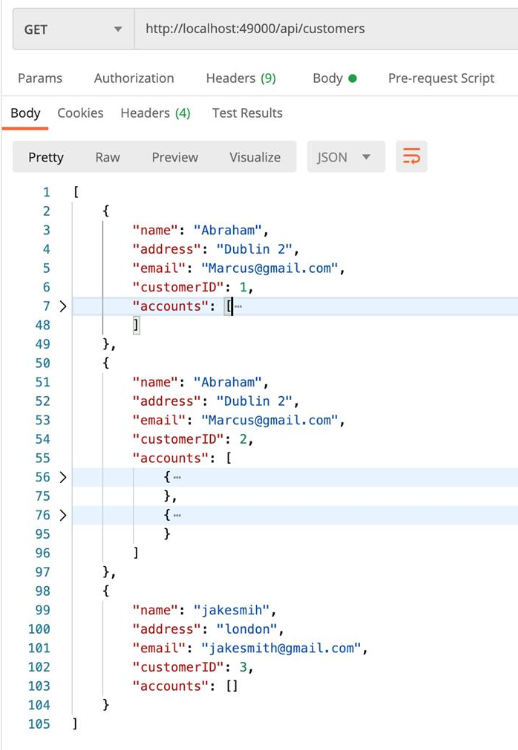
|  |  |
| --- | --- |
| API Name | MovieResource |
| Description | Transfer one movie to another account hold by the same customer |
| URI | /movie/{movieId}/{transAccId} |
| HTTP VERB | @POST |
| Parameters | All customerId, accountId, transAccId and movieId (Integer Require) |
| Resource Content | I.E. customerId = “12” accountId = “2” transAccId = “3” movieId = “3” |
| Pre-Conditions | That targeted account must exist in the database |
| Post-Conditions | Return a message “The movie (that movie name being transfer) was transferred to Account Id (targeted account Id) |

# Web Service Operation Section

## 1.) Creating new customer using POST request

Figure 1.2: Adding new customer

Figure 1.1: Before adding new customer to DB (only 2)

Using Postman I use the POST request to the send the data to the database

* Name
* Address
* Email

Figure 1.3: After added a new customer to the DB (3 now)

## 2.) Adding a new movie to an account using POST request

Figure 2.1: Before adding a new movie (2 reminds)

Figure 2.3: After successfully added a movie (now 3)

Figure 2.2: Adding new movie (Orange Wedding)

Using Postman I use the POST request to send the data to the database

* Movie Name
* Account Id
* Customer Id

## 3.) Transfer a movie to another account hold by the customer

Figure 3.1: Before transfering account (2) to account (1)

Figure 3.2: Before transfering (Red Wedding) movie to account (1)

Using Postman I use the POST request

* Movie Id
* MovieName
* isWatch
* Summary

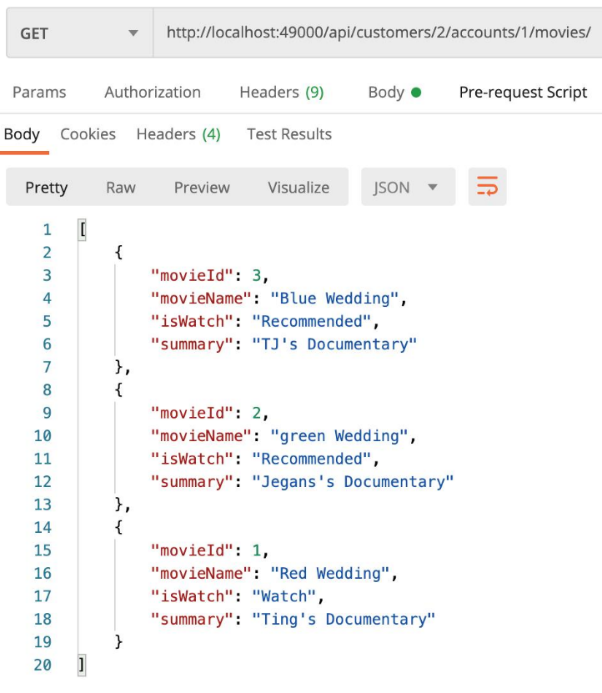
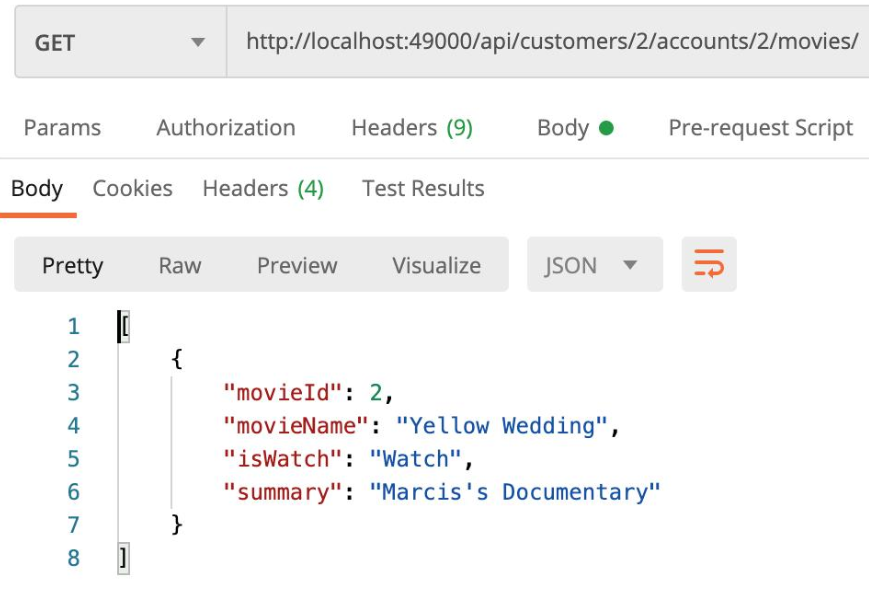


Figure 3.4: After successfully transfer (Red Wedding) to account (1)

Figure 3.3: Transfer process from account (2) to account (1)

Figure 3.5: After (Red Wedding) movie was removed from account (2)

## 4.) Remove a movie from account by using Id

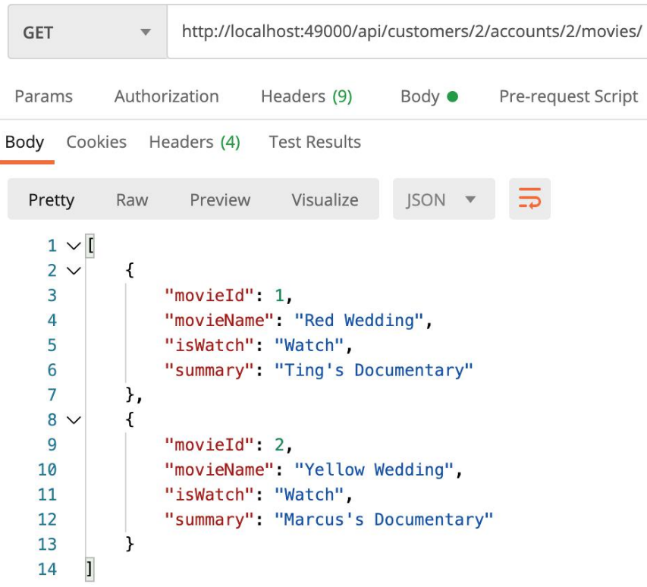
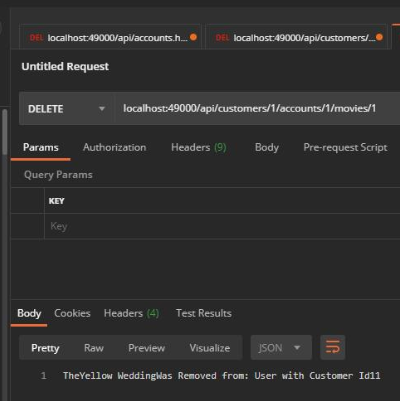
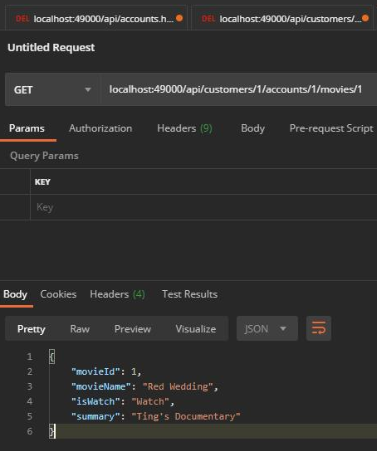


Figure 4.2: Removing a movie from account

Figure 4.1: Before remove a movie (2 reminds)



Using Postman I use the DELETE request

* Return message that its has been removed from Database

Figure 4.3: After successfully deleted (Yellow Wedding) only (Red Wedding) movie left

## 5.) Making a GET request to retrieve a movie list of a customer’s account by Id

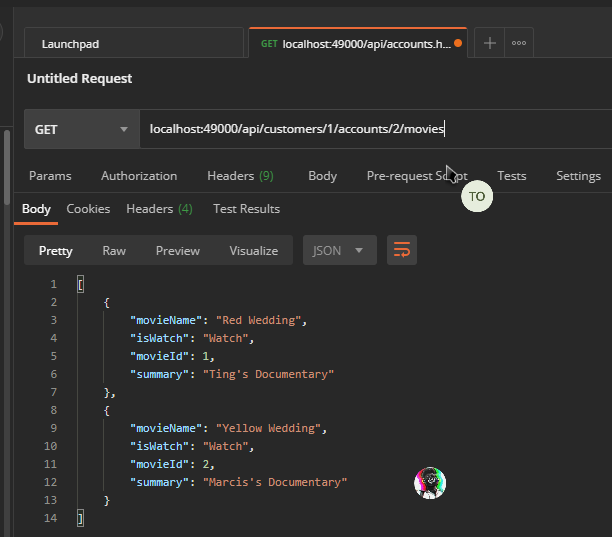


Figure 5: List of accounts's movies

Using Postman I use the GET request to retrieve:

* Movie Name
* Watch or Recommended
* Movie Id
* Movie Summary