Interview Now

From experts to experts

Software Specifications and System Implementation

Del.1.2

Version 1.0

Χρυσικός Χρήστος, christosc@ece.auth.gr

Φώλας Δεμίρης Δημήτριος, folasded@ece.auth.gr

Γούναρης Γιώργος, ggounaris@ece.auth.gr

Σαπουντζής Ανδρέας, spandreas@ece.auth.gr

June 2022

Modification Log

Name	Date	Modification	Version
Α. Συμεωνίδης	17/05/2007	Δημιουργία εγγράφου. Προσαρμογή των προτύπων του K. E. Wiegers και του M. Smialek's.	0.1
Α. Συμεωνίδης	29/3/2014	Μικρή αναθεώρηση – τροποποίηση ενοτήτων	0.1.3
Χ. Ζολώτας	10/4/2020	Μεγάλη αναθεώρηση – αφαίρεση ενοτήτων	0.4
Χ. Ζολώτας	15/4/2020	Μεγάλη αναθεώρηση – προσθήκη ενότητας REST προδιαγραφών	0.5.3
Κ. Παναγιώτου	25/4/2020	Μεγάλη αναθεώρηση – προσθήκη ενότητας Nodered περιγραφής	0.5.7
Α. Συμεωνίδης	30/4/2020	Αναθεώρηση και τελική δομή προτύπου	0.6

Development Team Members

Name	DT	Email
Α. Συμεωνίδης	*	asymeon@issel.ee.auth.gr
Χ. Χρυσικός	7	folasded@ece.auth.gr
Δ. Φώλας Δεμίρης	7	christosc@ece.auth.gr
Γ. Γούναρης	7	ggounaris@ece.auth.gr
Α. Σαπουντζής	7	spandreas@ece.auth.gr

Table of Contents

Figures' List

1. Implemented Design Patterns

1.1. Proxy Pattern

Proxy prototype is a structural prototype, a class functioning as an interface to another. It is used to secure user's data and establish a connection between the database and the app. Specifically, in the proxy pattern, we create an object based on the original object to interface its functionality to the outer world. In this way, the proxy satisfies the NFR-5 GDPR requirement by hiding the original object's complexity from the client. Moreover, the database proxy has an important role in the app's functionality as it forwards connections to the database.

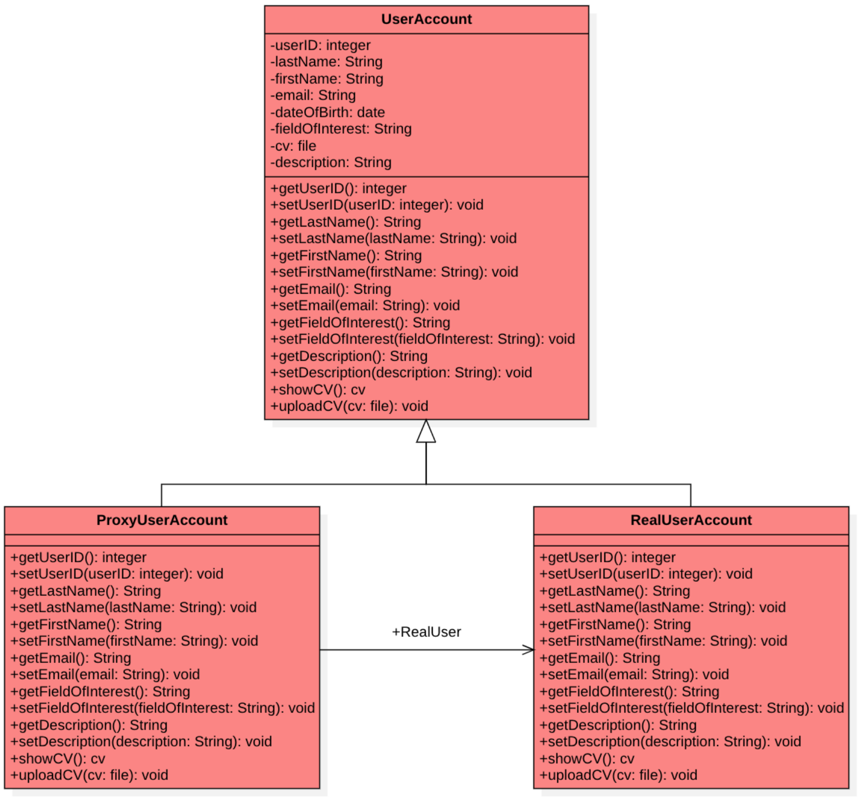

UserAccount
-userID: integer
-lastName: String
-firstName: String
-email: String
-dateOfBirth: date
-fieldOfInterest: String
-cv: file
-description: String
+getUserID(): integer
+setUserID(userID: integer): void
+getLastName(): String
+setLastName(lastName: String): void
+getFirstName(): String
+setFirstName(firstName: String): void
+getEmail(): String
+setEmail(email: String): void
+getFieldOfInterest(): String
+setFieldOfInterest(fieldOfInterest: String): void
+getDescription(): String
+setDescription(description: String): void
+showCV(): cv
+uploadCV(cv: file): void
ProxyUserAccount
+getUserID(): integer
+setUserID(userID: integer): void
+getLastName(): String
+setLastName(lastName: String): void
+getFirstName(): String
+setFirstName(firstName: String): void
+getEmail(): String
+setEmail(email: String): void
+getFieldOfInterest(): String
+setFieldOfInterest(fieldOfInterest: String): void
+getDescription(): String
+setDescription(description: String): void
+showCV(): cv
+uploadCV(cv: file): void
+RealUser
RealUserAccount
+getUserID(): integer
+setUserID(userID: integer): void
+getLastName(): String
+setLastName(lastName: String): void
+getFirstName(): String
+setFirstName(firstName: String): void
+getEmail(): String
+setEmail(email: String): void
+getFieldOfInterest(): String
+setFieldOfInterest(fieldOfInterest: String): void
+getDescription(): String
+setDescription(description: String): void
+showCV(): cv
+uploadCV(cv: file): void

Figure 1

1.2. Memento Pattern

Memento Prototype is a behavioral prototype that represents the ability to restore an object to its previous state (undo). Memento pattern is used in order to satisfy NFR-3 requirement. User's progress is autosaved every time a user action is performed. It uses three actor classes, Memento, Originator and CareTaker. Memento contains the state of an object to be restored. Originator creates and stores states in Memento objects and CareTaker object is responsible to restore the object state from Memento.

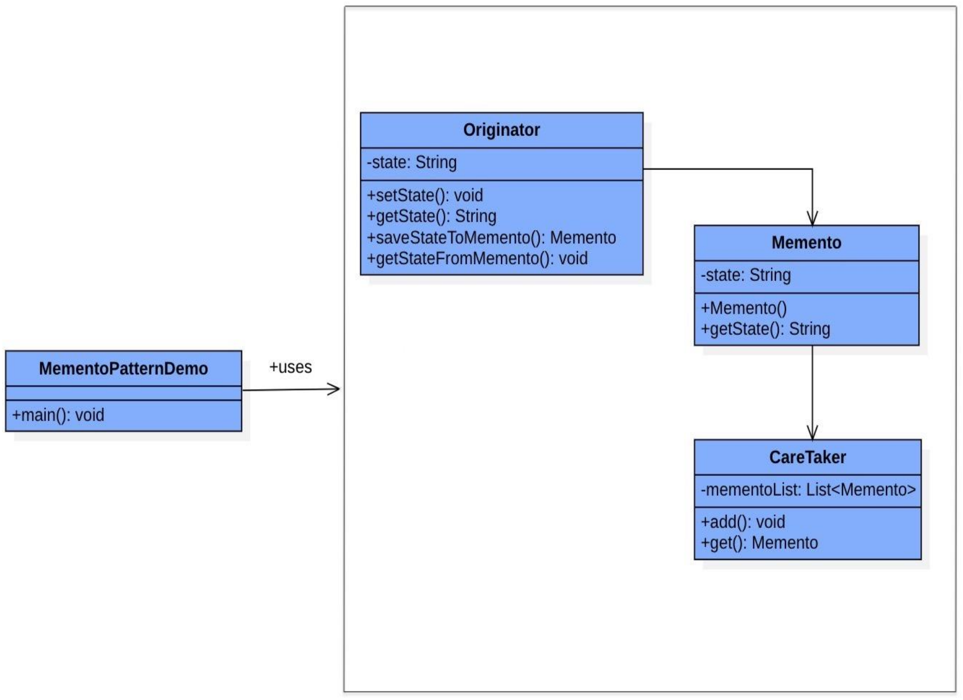


Figure 2

1.3. Composite Pattern

Composite pattern is a structural prototype, a tree structure of objects where every object has the same interface. It is used when we need to treat a group of objects in a similar way as a single object. In our case, the user's assignments are grouped by the manager's id. Each manager can manage and control multiple assignments for different users.

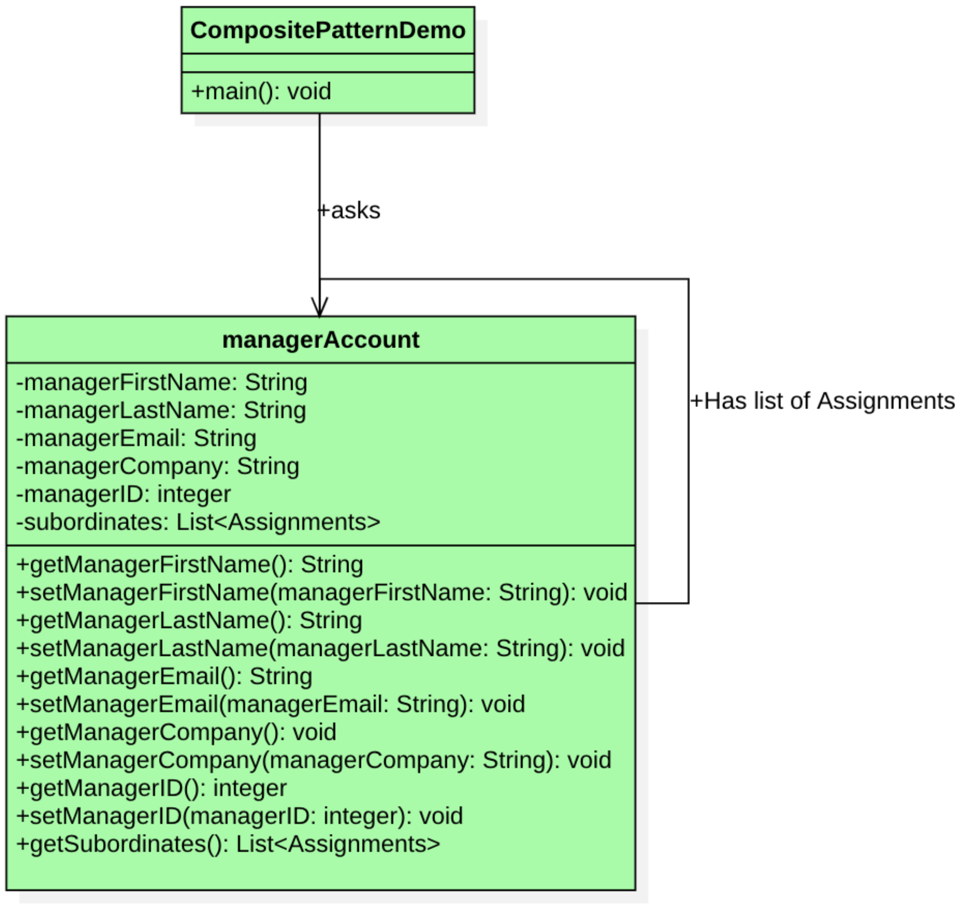

CompositePatternDemo
+main(): void
+asks
+Has list of Assignments
managerAccount
-managerFirstName: String
-managerLastName: String
-managerEmail: String
-managerCompany: String
-managerID: integer
-subordinates: List<Assignments>
+getManagerFirstName(): String
+setManagerFirstName(managerFirstName: String): void
+getManagerLastName(): String
+setManagerLastName(managerLastName: String): void
+getManagerEmail(): String
+setManagerEmail(managerEmail: String): void
+getManagerCompany(): void
+setManagerCompany(managerCompany: String): void
+getManagerID(): integer
+setManagerID(managerID: integer): void
+getSubordinates(): List<Assignments>

Figure 3

2. System Architecture

SwaggerHub was used to write the API.

1. Link to the JSON file with the API specifications.
2. Link to the zip file with the code for the application server creation.
3. Link to API on SwaggerHub.

2.1. System Resource Identification

BEC Class	REST Resource	Endpoints (HTTP Verbs)
user	/user	POST
user	/user/{userID}	GET, PUT, DELETE
manager	/manager	POST
manager	/manager/{managerID}	GET, PUT, DELETE
assignment	/assignment/{ual}	GET, DELETE
assignment	/manager/{managerID}/assignment	POST, GET
assignment	/manager/{managerID}/assignment/{ual}	PUT, DELETE
assignment	/user/{userID}/assignment	GET

2.2. REST Interface Justification

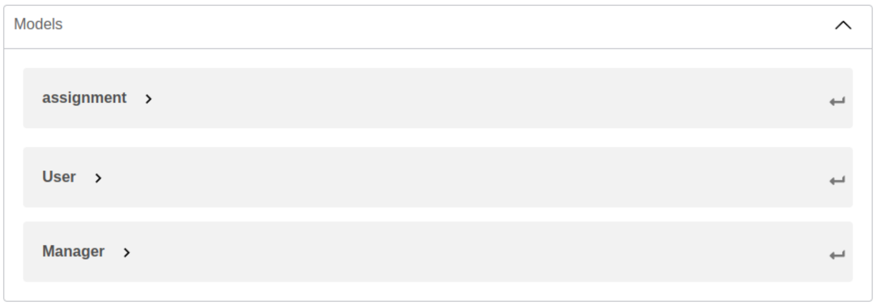
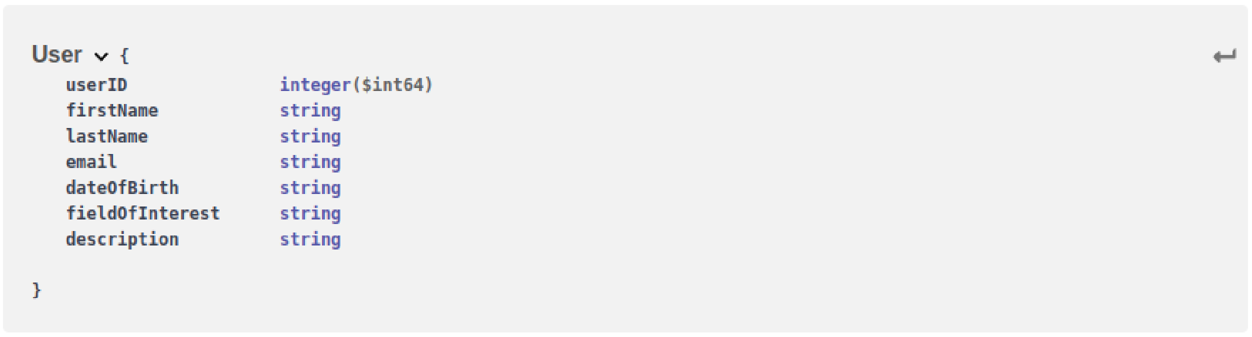


Figure 4

2.2.1. User Resource

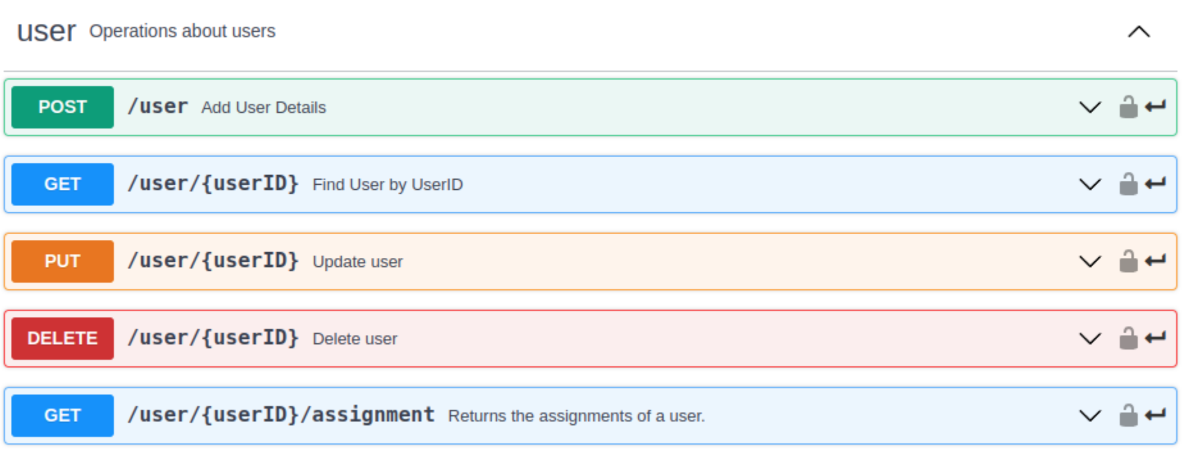
2.2.1.1. User Data Model



```
User {
   userID             integer($int64)
   firstName          string
   lastName           string
   email              string
   dateOfBirth        string
   fieldOfInterest    string
   description        string

}
```

Figure 5

2.2.1.2. Sum of User Resource Endpoints


user Operations about users

POST /user Add User Details

GET /user/{userID} Find User by UserID

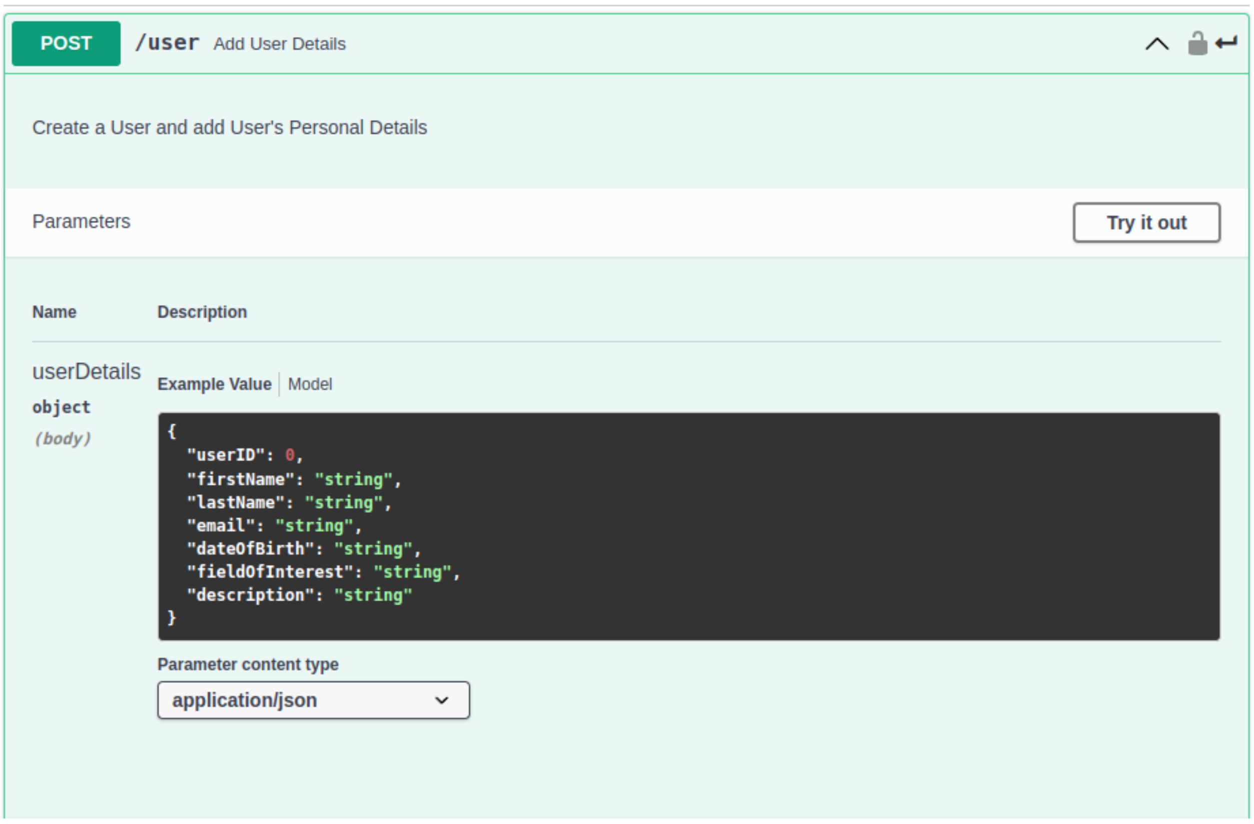
PUT /user/{userID} Update user

DELETE /user/{userID} Delete user

GET /user/{userID}/assignment Returns the assignments of a user.

Figure 6

2.2.1.3. User resource POST Endpoint


POST /user Add User Details

Create a User and add User's Personal Details

Parameters

Try it out

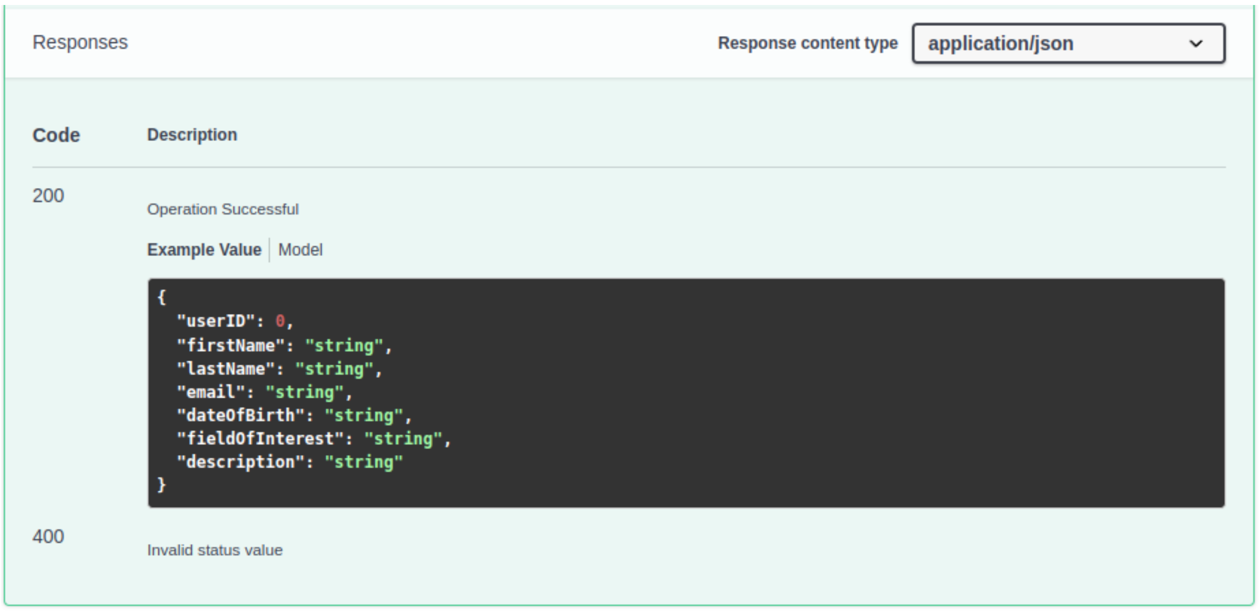
Name	Description
userDetails object *(body)*	Example Value | Model

```
{
  "userID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "dateOfBirth": "string",
  "fieldOfInterest": "string",
  "description": "string"
}
```

Parameter content type

application/json

Figure 7


Responses

Response content type: application/json

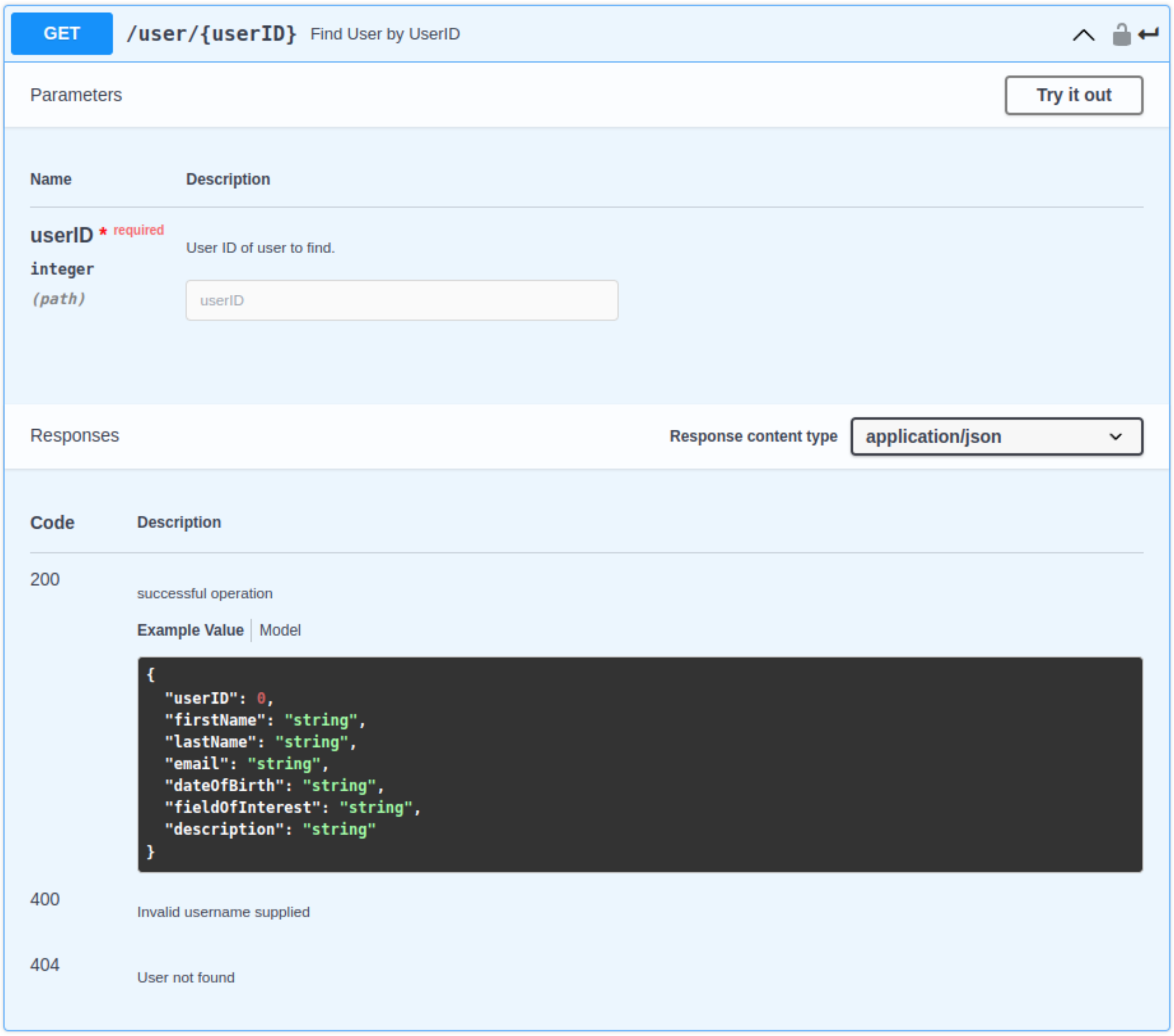
Code	Description
200	Operation Successful
400	Invalid status value

Example Value | Model

```
{
  "userID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "dateOfBirth": "string",
  "fieldOfInterest": "string",
  "description": "string"
}
```

Figure 8

2.2.1.4. User GET Endpoint, w/ specified userID



/user/{userID} Find User by UserID

Parameters

Name	Description
userID * required integer (path)	User ID of user to find. userID

Responses

Response content type: application/json

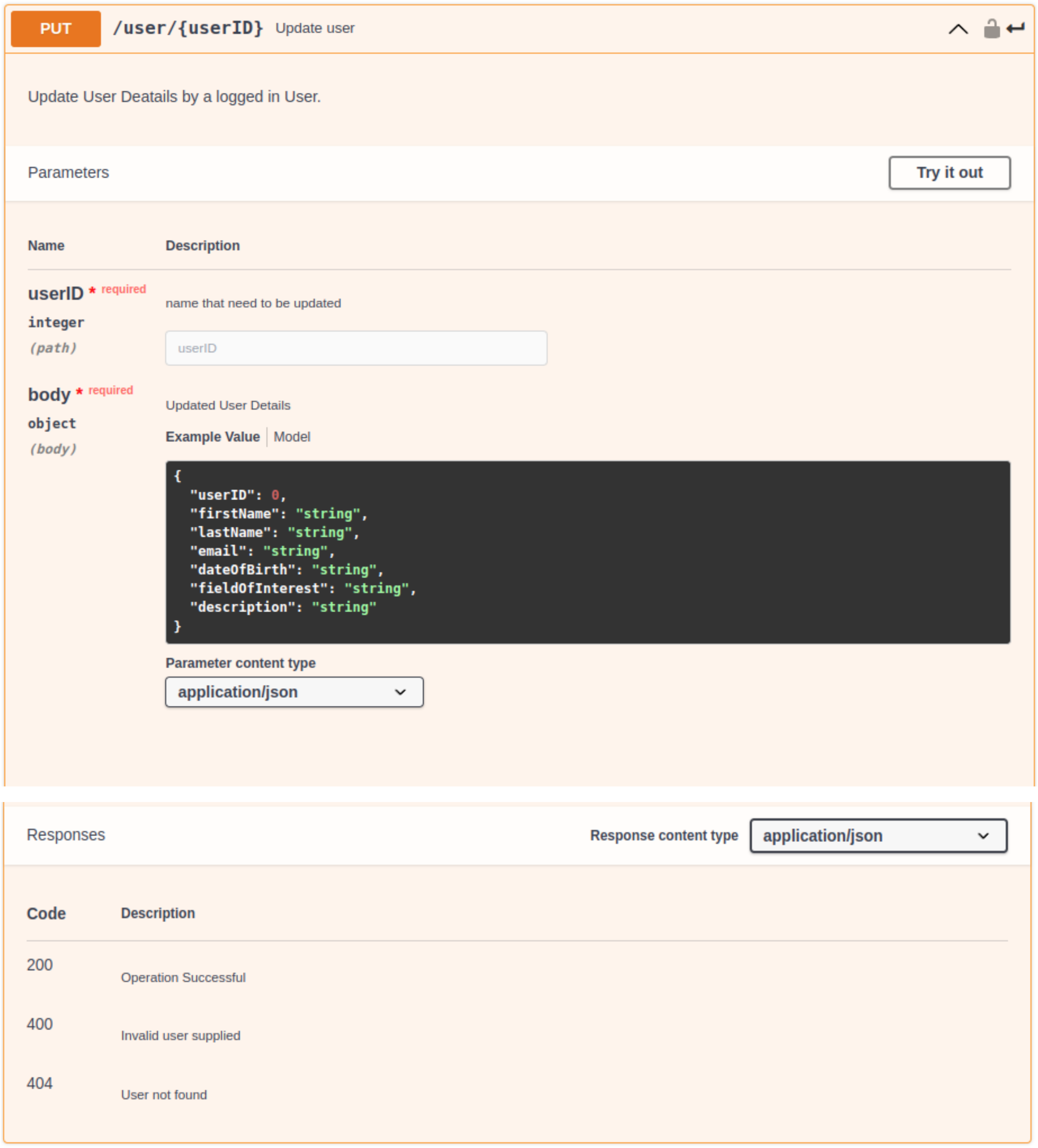
Code	Description
200	successful operation
400	Invalid username supplied
404	User not found

Example Value | Model

```
{
  "userID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "dateOfBirth": "string",
  "fieldOfInterest": "string",
  "description": "string"
}
```

Figure 9

2.2.1.5. User PUT Endpoint, w/ specified userID



PUT /user/{userID} Update user

Update User Deatails by a logged in User.

Parameters

Name	Description
userID * required integer (path)	name that need to be updated
body * required object (body)	Updated User Details

Example Value | Model

```
{
  "userID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "dateOfBirth": "string",
  "fieldOfInterest": "string",
  "description": "string"
}
```

Parameter content type: application/json

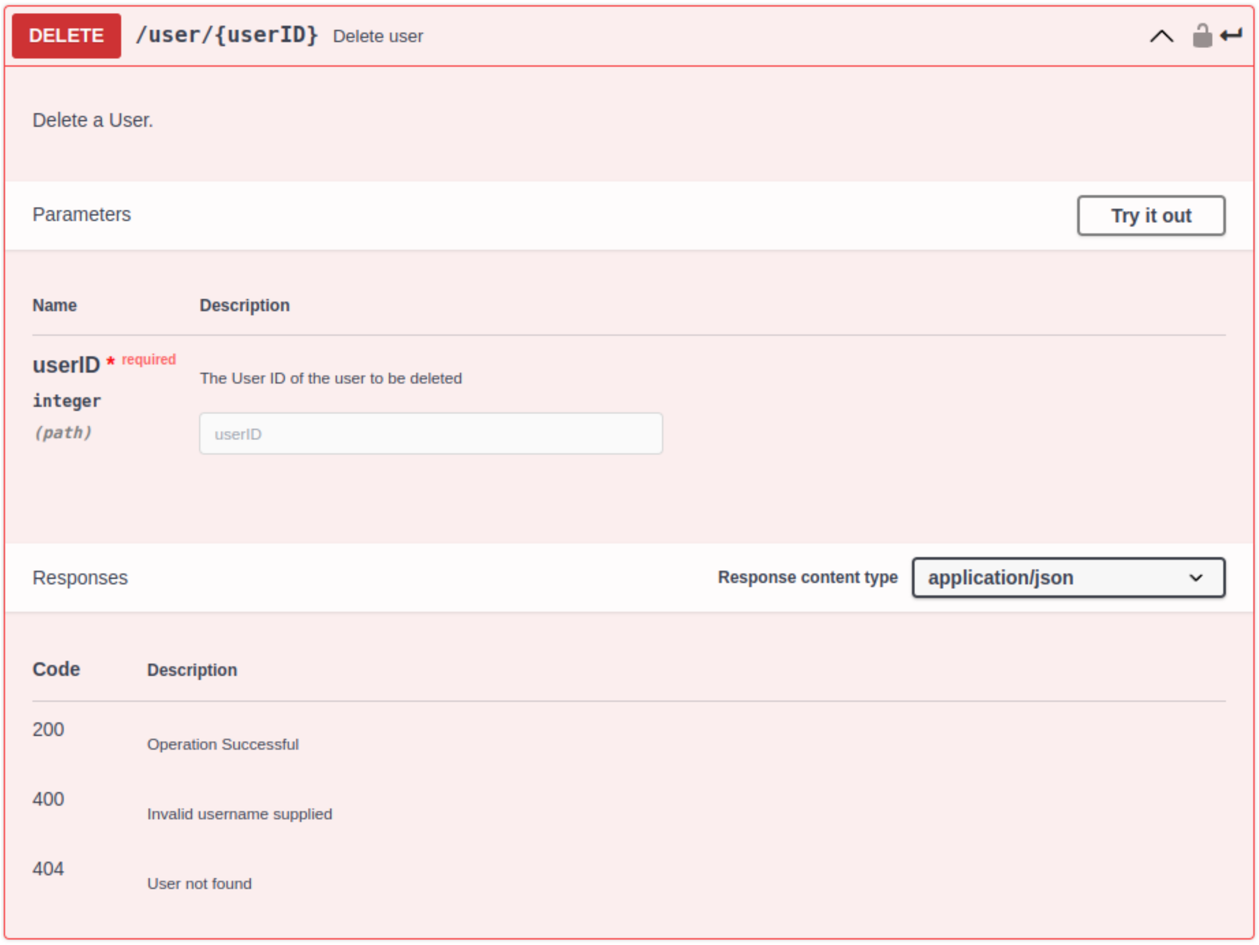
Responses

Response content type: application/json

Code	Description
200	Operation Successful
400	Invalid user supplied
404	User not found

Figure 10

2.2.1.6. User DELETE Endpoint, w/ specified userID


DELETE /user/{userID} Delete user

Delete a User.

Parameters

Name	Description
userID * required integer (path)	The User ID of the user to be deleted

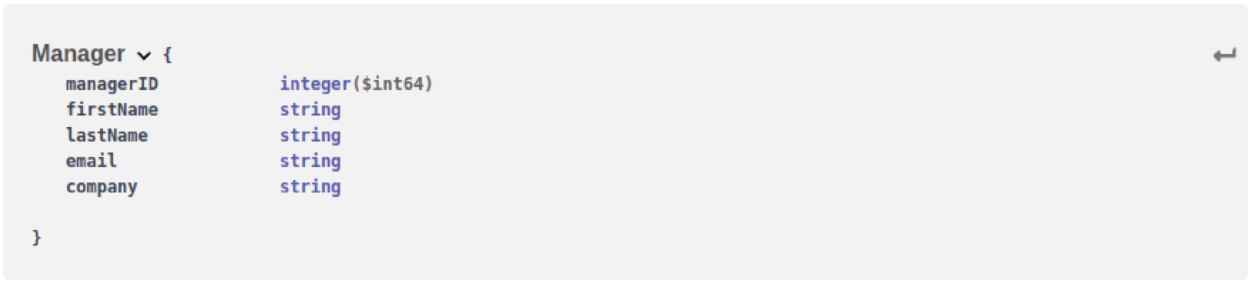
Responses — Response content type: application/json

Code	Description
200	Operation Successful
400	Invalid username supplied
404	User not found

Figure 11

2.2.2. Manager Resource

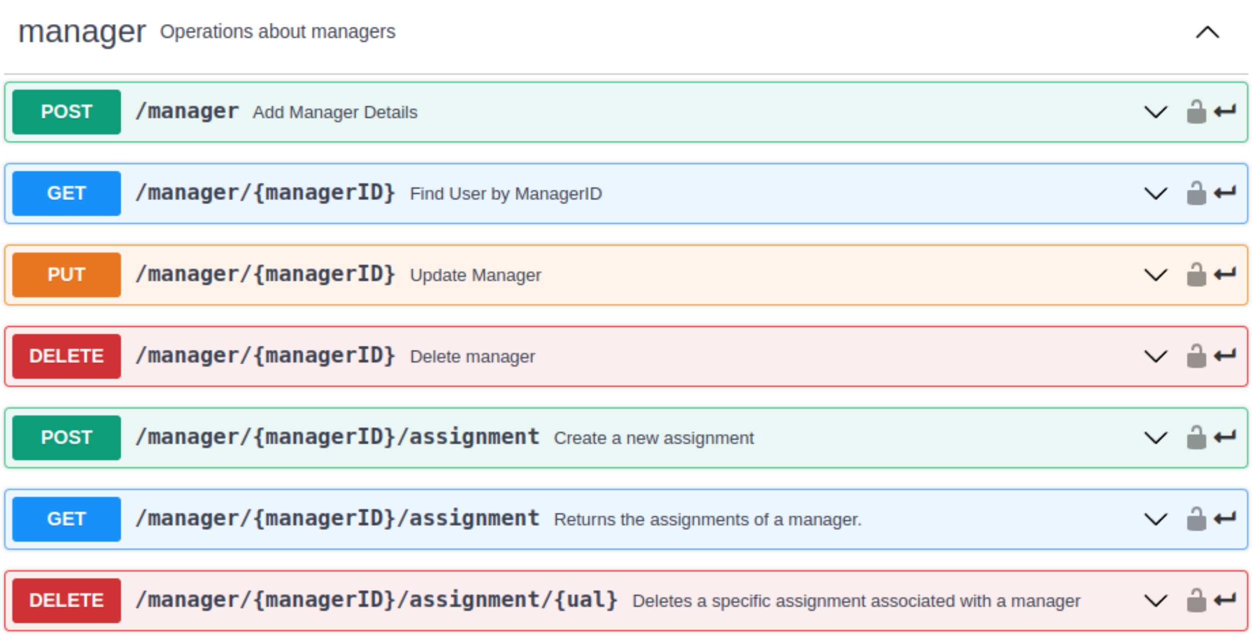
2.2.2.1. Manager Data Model



```
Manager ⌄ {
    managerID           integer($int64)
    firstName           string
    lastName            string
    email               string
    company             string

}
```

Figure 12

2.2.2.2. Sum of Manager Endpoints

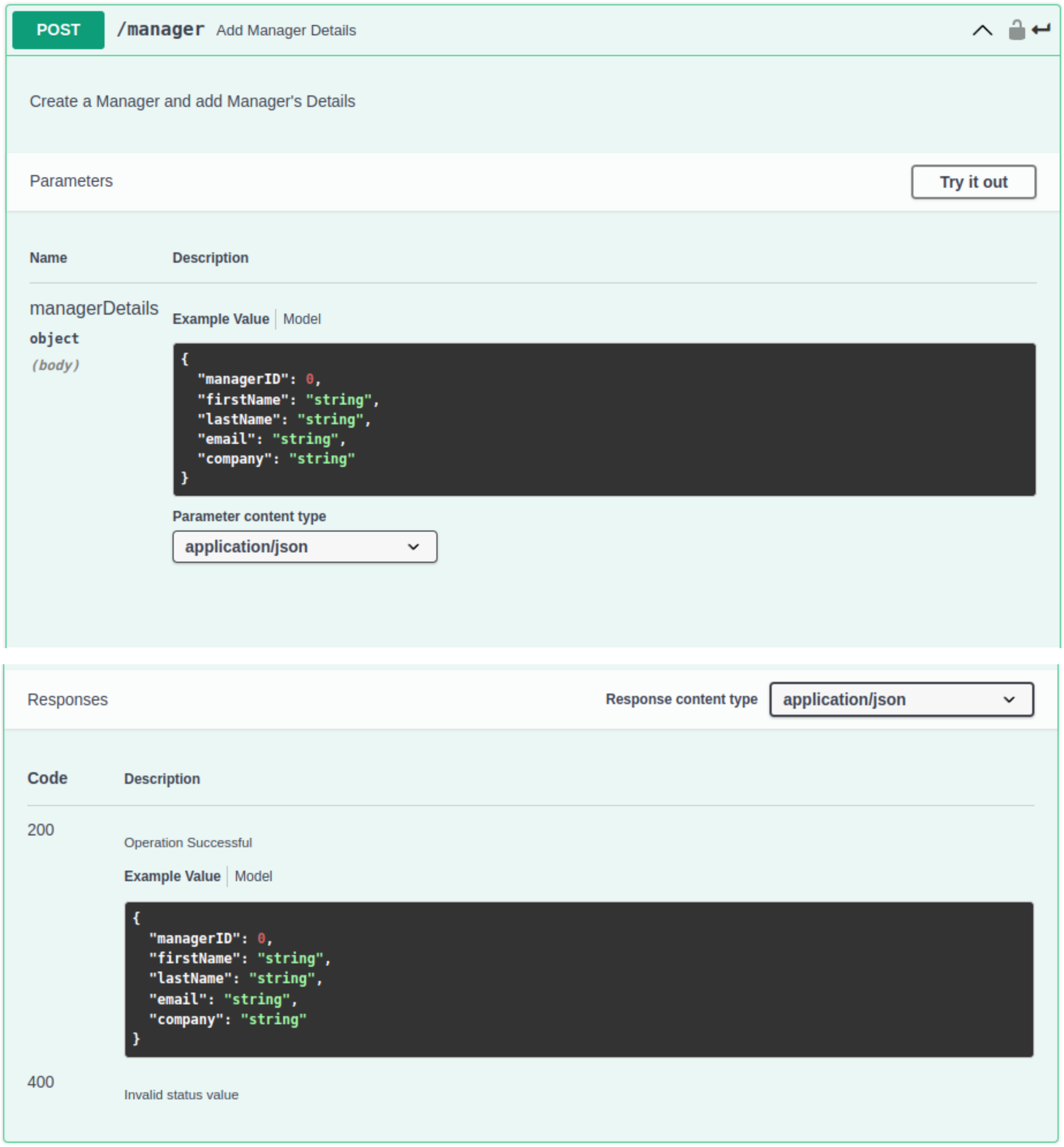


manager Operations about managers

```
POST    /manager                                       Add Manager Details
GET     /manager/{managerID}                           Find User by ManagerID
PUT     /manager/{managerID}                           Update Manager
DELETE  /manager/{managerID}                           Delete manager
POST    /manager/{managerID}/assignment                Create a new assignment
GET     /manager/{managerID}/assignment                Returns the assignments of a manager.
DELETE  /manager/{managerID}/assignment/{ual}          Deletes a specific assignment associated with a manager
```

Figure 13

2.2.2.3. Manager POST Endpoint



Create a Manager and add Manager's Details

Parameters

Name	Description
managerDetails object (body)	Example Value | Model

```
{
  "managerID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "company": "string"
}
```

Parameter content type: application/json

Responses — Response content type: application/json

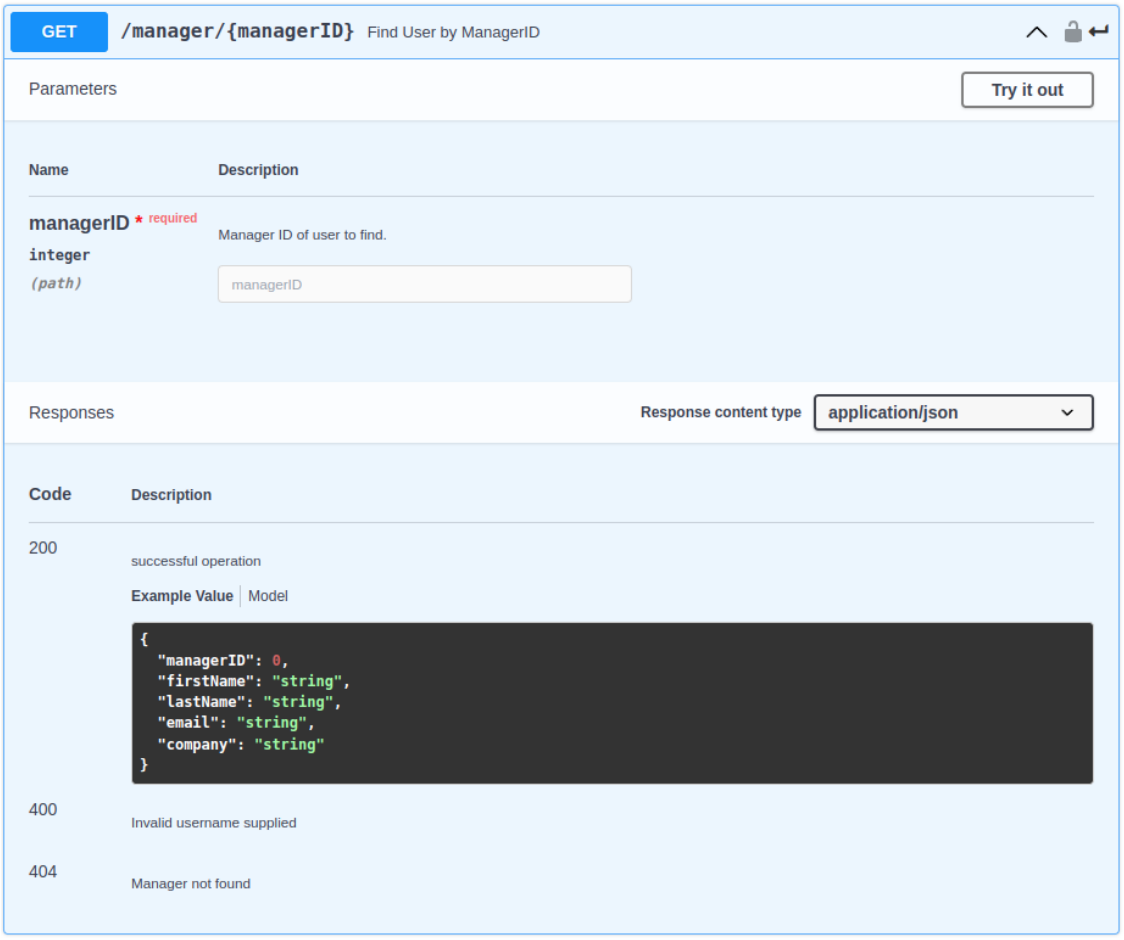
Code	Description
200	Operation Successful
400	Invalid status value

Example Value | Model

```
{
  "managerID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "company": "string"
}
```

Figure 14

2.2.2.4. Manager GET Endpoint, w/ specified managerID


GET /manager/{managerID} Find User by ManagerID

Parameters

Name	Description
managerID * required integer (path)	Manager ID of user to find. managerID

Responses — Response content type: application/json

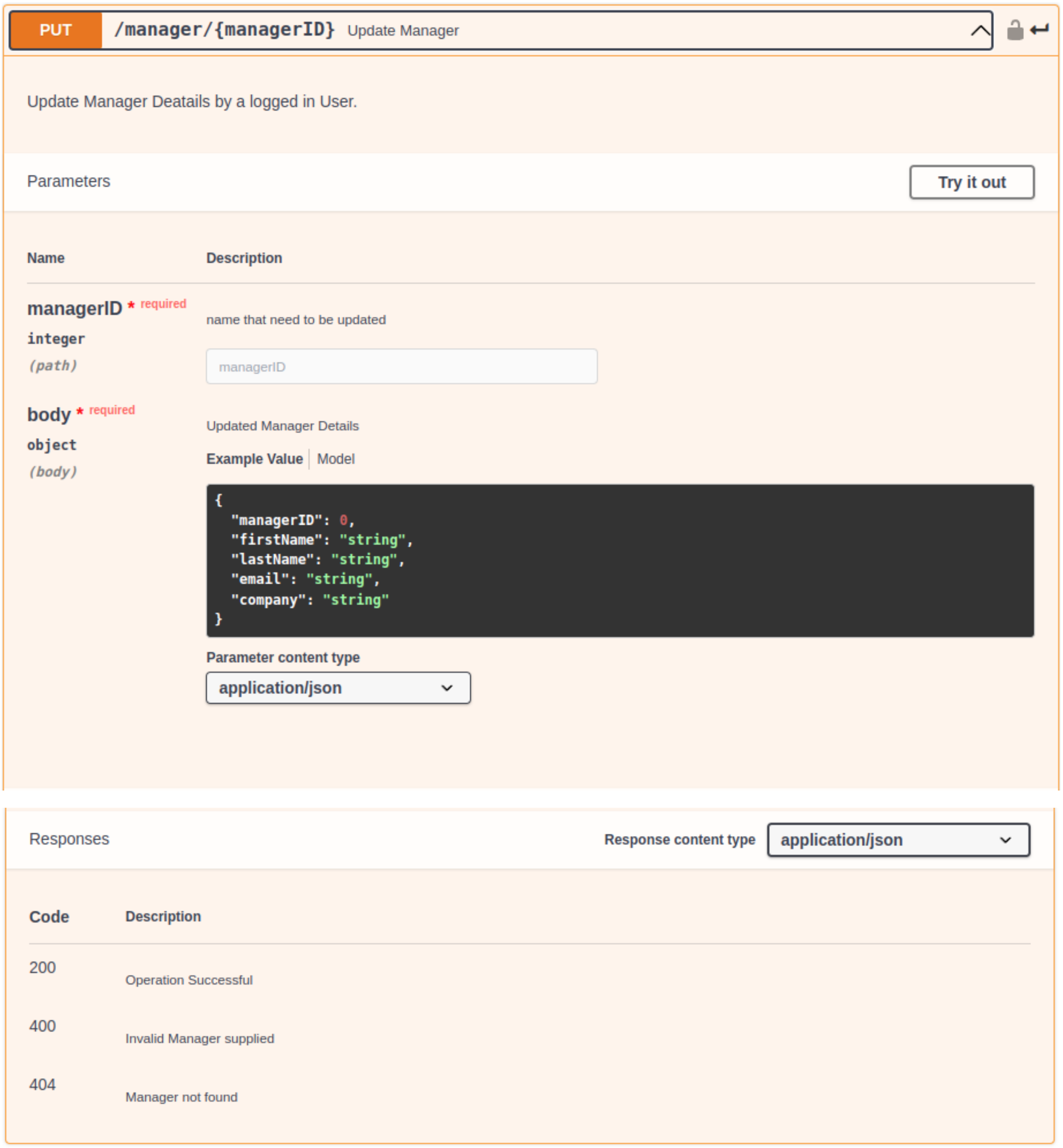
Code	Description
200	successful operation
400	Invalid username supplied
404	Manager not found

Example Value | Model

```
{
  "managerID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "company": "string"
}
```

Figure 15

2.2.2.5. Manager PUT Endpoint, w/ specified managerID



Update Manager Deatails by a logged in User.

Parameters

Name	Description
managerID * required integer (path)	name that need to be updated (managerID)
body * required object (body)	Updated Manager Details

Example Value | Model

```
{
  "managerID": 0,
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "company": "string"
}
```

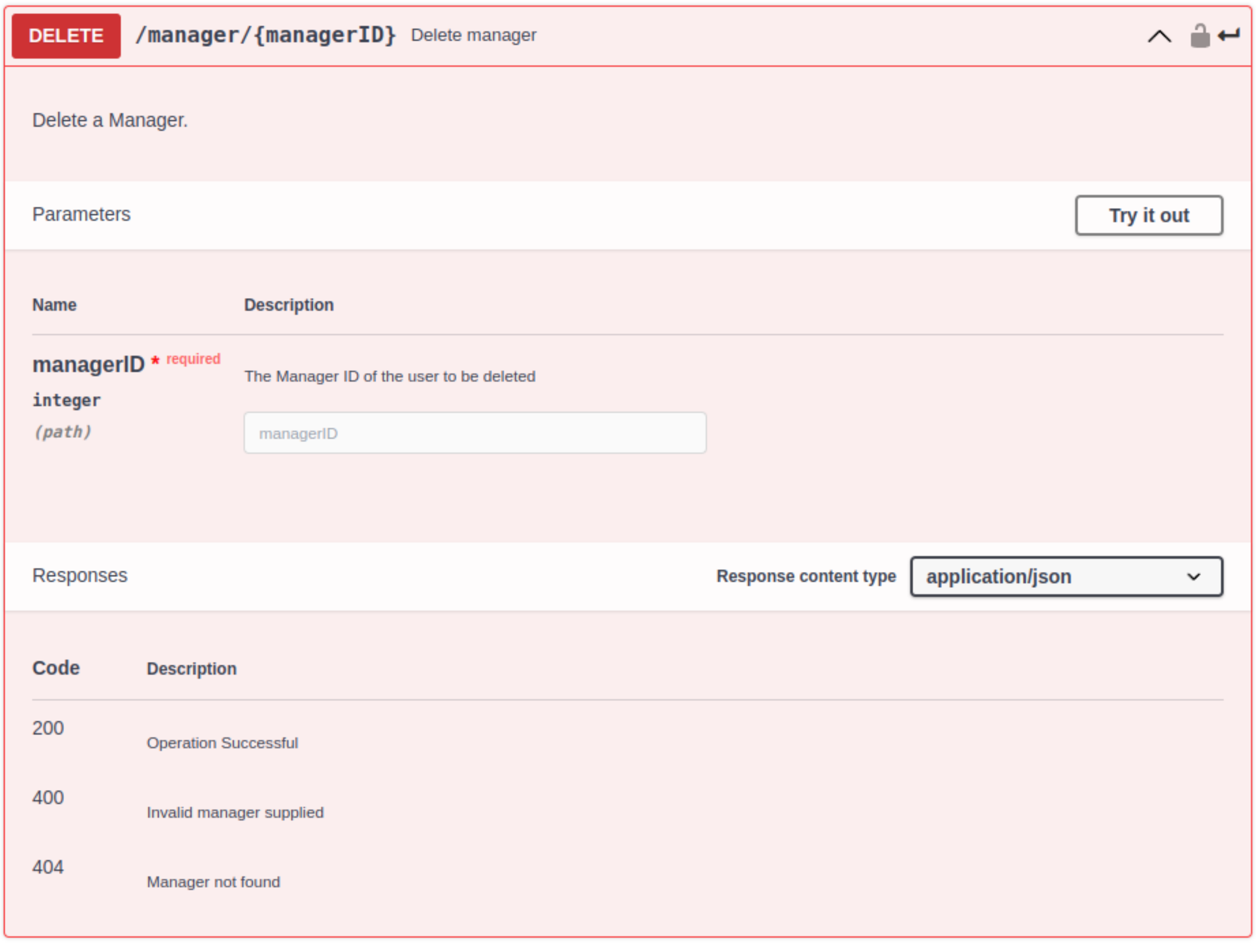
Parameter content type: application/json

Responses — Response content type: application/json

Code	Description
200	Operation Successful
400	Invalid Manager supplied
404	Manager not found

Figure 16

2.2.2.6. Manager DELETE Endpoint, w/ specified managerID



DELETE `/manager/{managerID}` Delete manager

Delete a Manager.

Parameters

Name	Description
managerID * required integer *(path)*	The Manager ID of the user to be deleted managerID

Responses

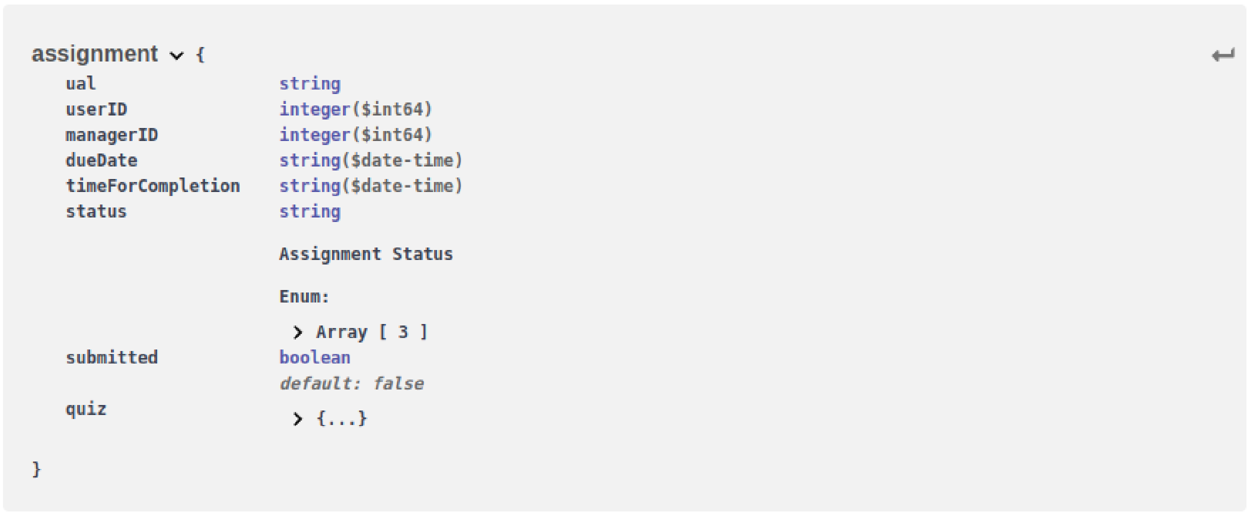
Response content type: application/json

Code	Description
200	Operation Successful
400	Invalid manager supplied
404	Manager not found

Figure 17

2.2.3. Assignment Resource

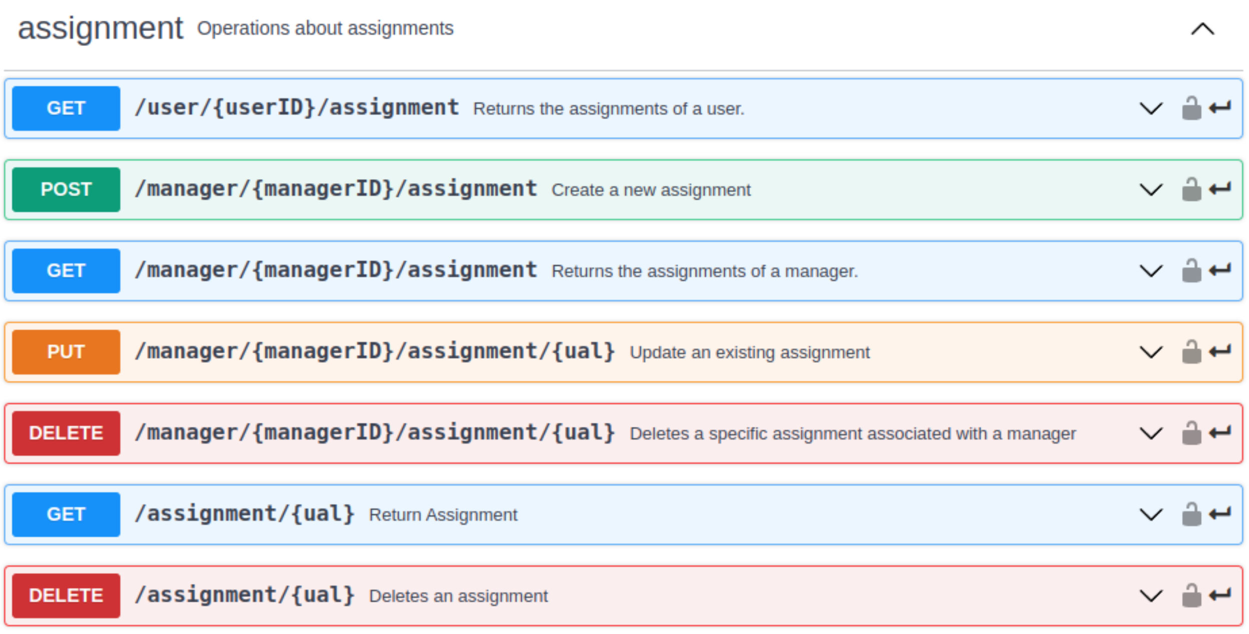
2.2.3.1. Assignment Resource Data Model



```
assignment ⌄ {
    ual                  string
    userID               integer($int64)
    managerID            integer($int64)
    dueDate              string($date-time)
    timeForCompletion    string($date-time)
    status               string

                         Assignment Status

                         Enum:

                          > Array [ 3 ]
    submitted            boolean
                         default: false

    quiz                  > {...}

}
```

Figure 18

2.2.3.2. Sum of Assignment Resource Endpoints



assignment Operations about assignments

GET /user/{userID}/assignment Returns the assignments of a user.

POST /manager/{managerID}/assignment Create a new assignment

GET /manager/{managerID}/assignment Returns the assignments of a manager.

PUT /manager/{managerID}/assignment/{ual} Update an existing assignment

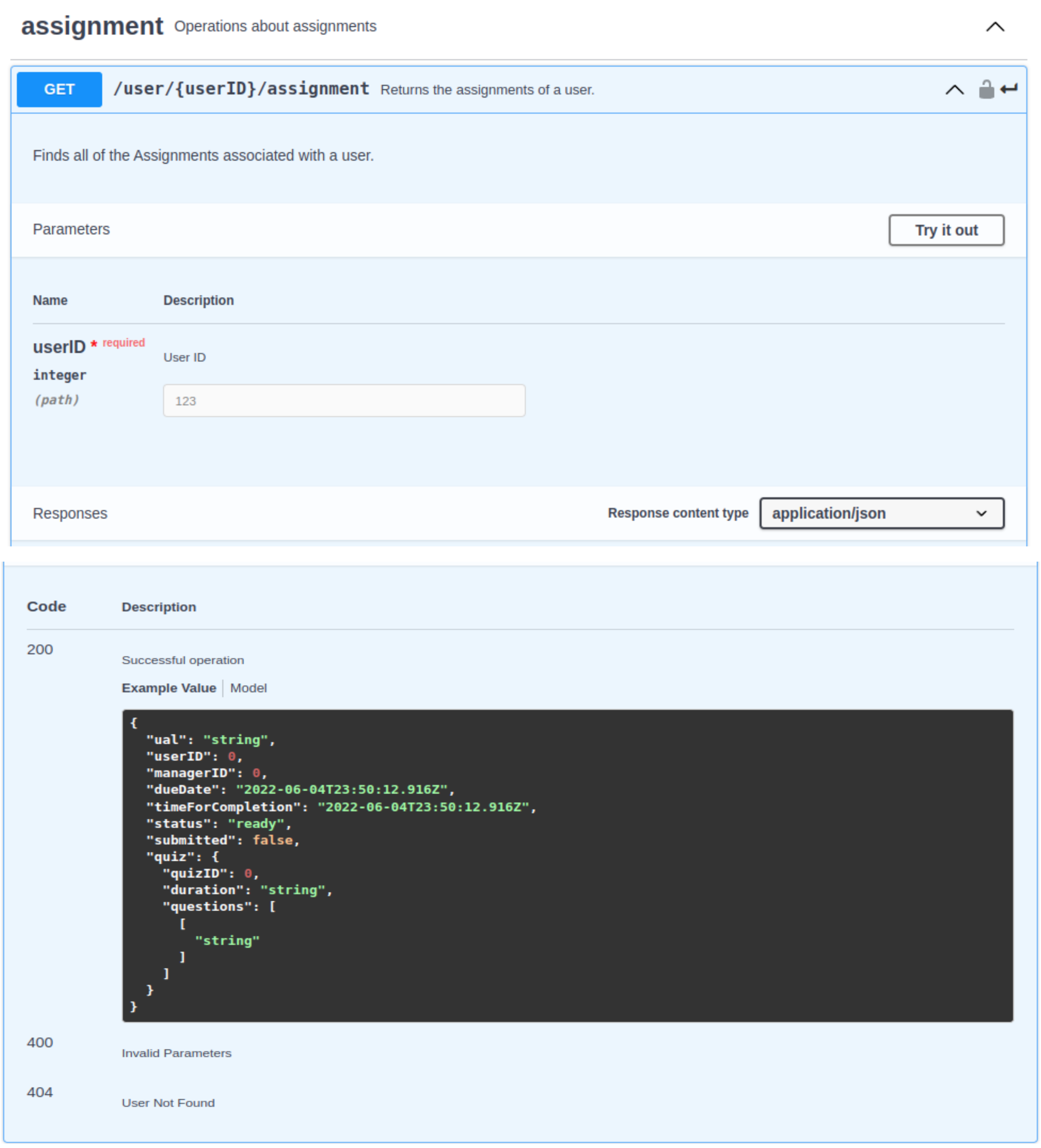
DELETE /manager/{managerID}/assignment/{ual} Deletes a specific assignment associated with a manager

GET /assignment/{ual} Return Assignment

DELETE /assignment/{ual} Deletes an assignment

Figure 19

2.2.3.3. Assignment GET Endpoint, for a User w/ specified userID



assignment Operations about assignments

GET `/user/{userID}/assignment` Returns the assignments of a user.

Finds all of the Assignments associated with a user.

Parameters

Try it out

Name	Description
userID * required integer *(path)*	User ID 123

Responses

Response content type: application/json

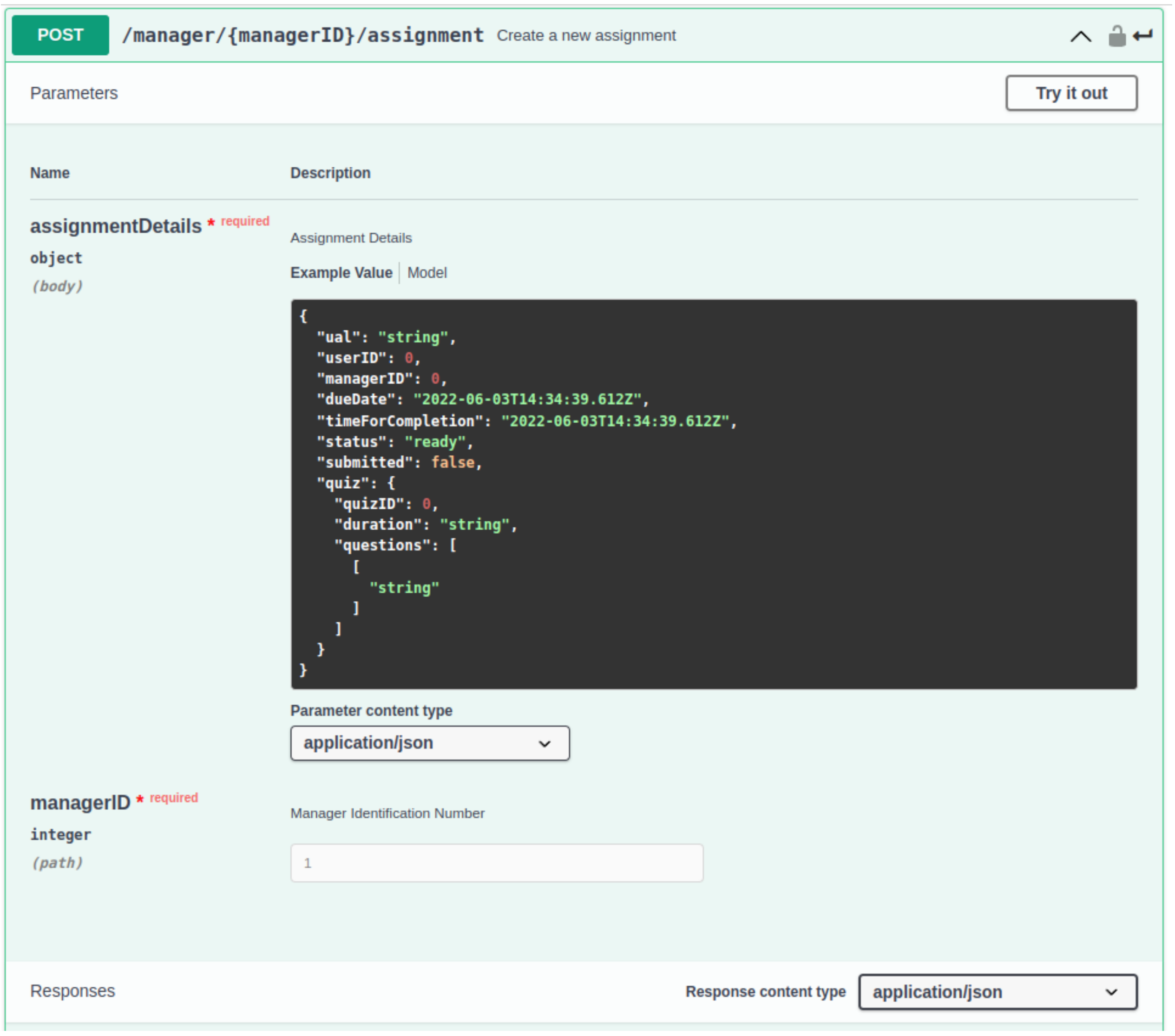
Code	Description
200	Successful operation
400	Invalid Parameters
404	User Not Found

Example Value | Model

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-04T23:50:12.916Z",
  "timeForCompletion": "2022-06-04T23:50:12.916Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Figure 20

2.2.3.4. Assignment POST Endpoint, for a manager w/ specified managerID



POST /manager/{managerID}/assignment Create a new assignment

Parameters — Try it out

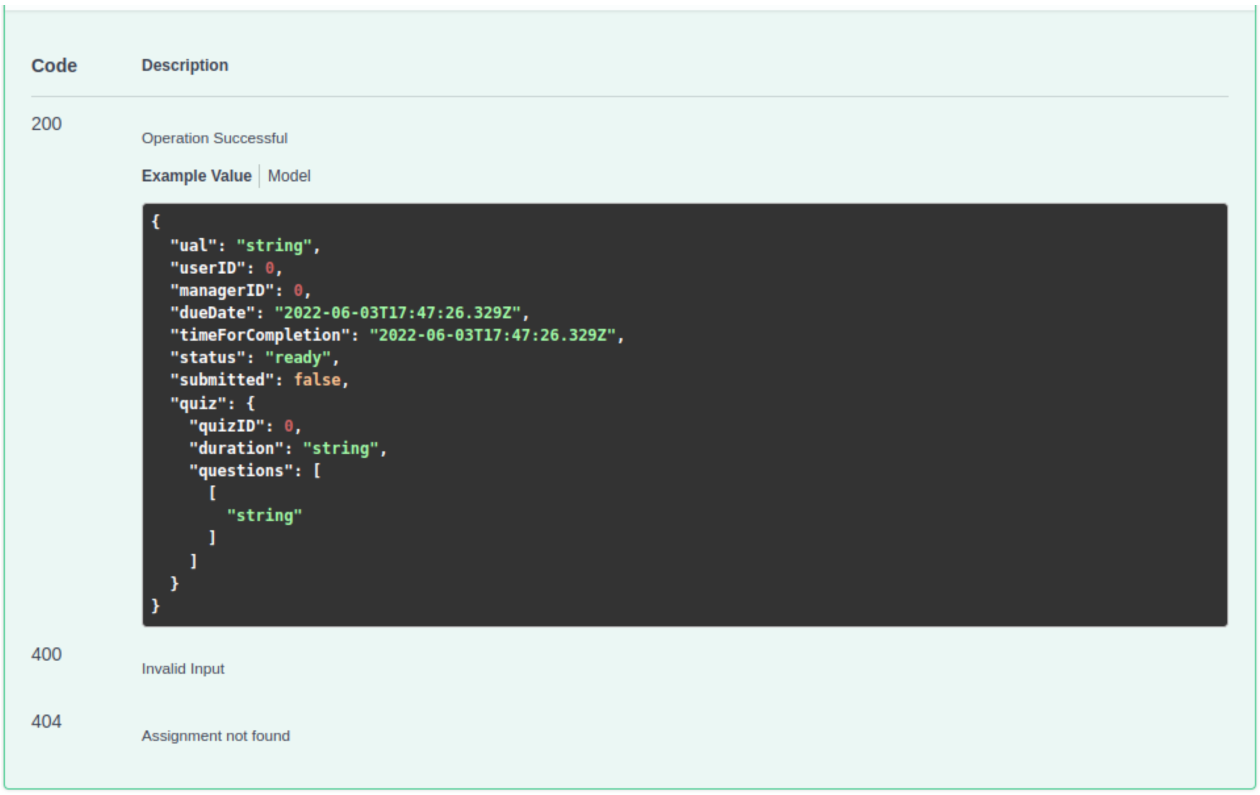
Name	Description
assignmentDetails * required object *(body)*	Assignment Details Example Value | Model
managerID * required integer *(path)*	Manager Identification Number 1

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-03T14:34:39.612Z",
  "timeForCompletion": "2022-06-03T14:34:39.612Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Parameter content type: application/json

Responses — Response content type: application/json

Figure 21



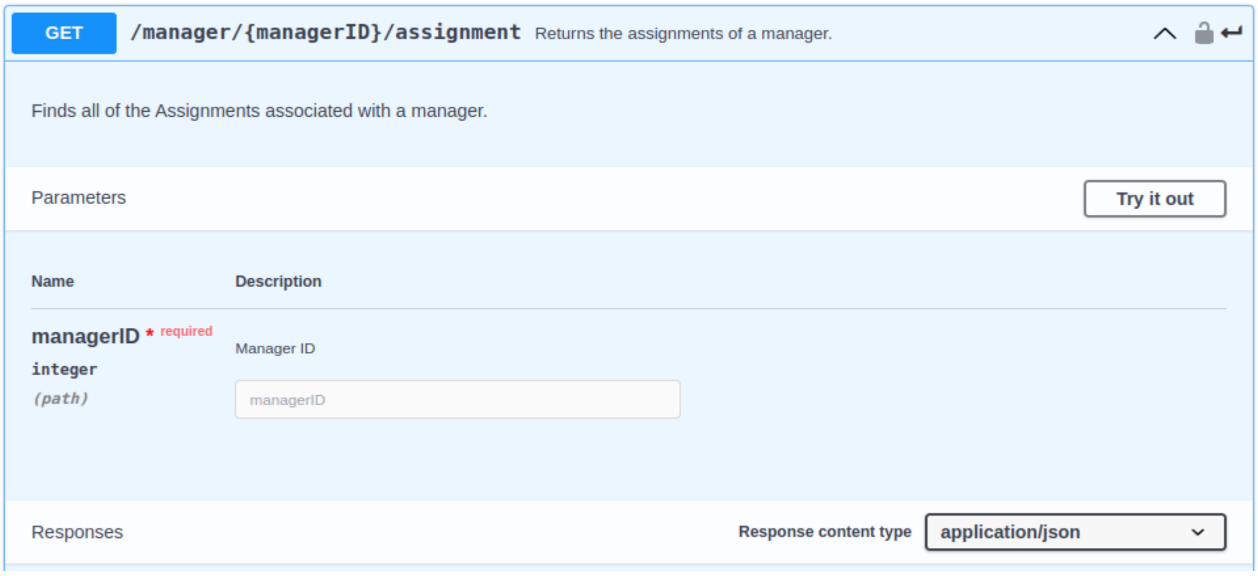
Code	Description
200	Operation Successful
400	Invalid Input
404	Assignment not found

Example Value | Model

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-03T17:47:26.329Z",
  "timeForCompletion": "2022-06-03T17:47:26.329Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Figure 22

2.2.3.5. Assignment GET Endpoint w/ specified ual, for a manager w/ specified managerID



GET `/manager/{managerID}/assignment` Returns the assignments of a manager.

Finds all of the Assignments associated with a manager.

Parameters

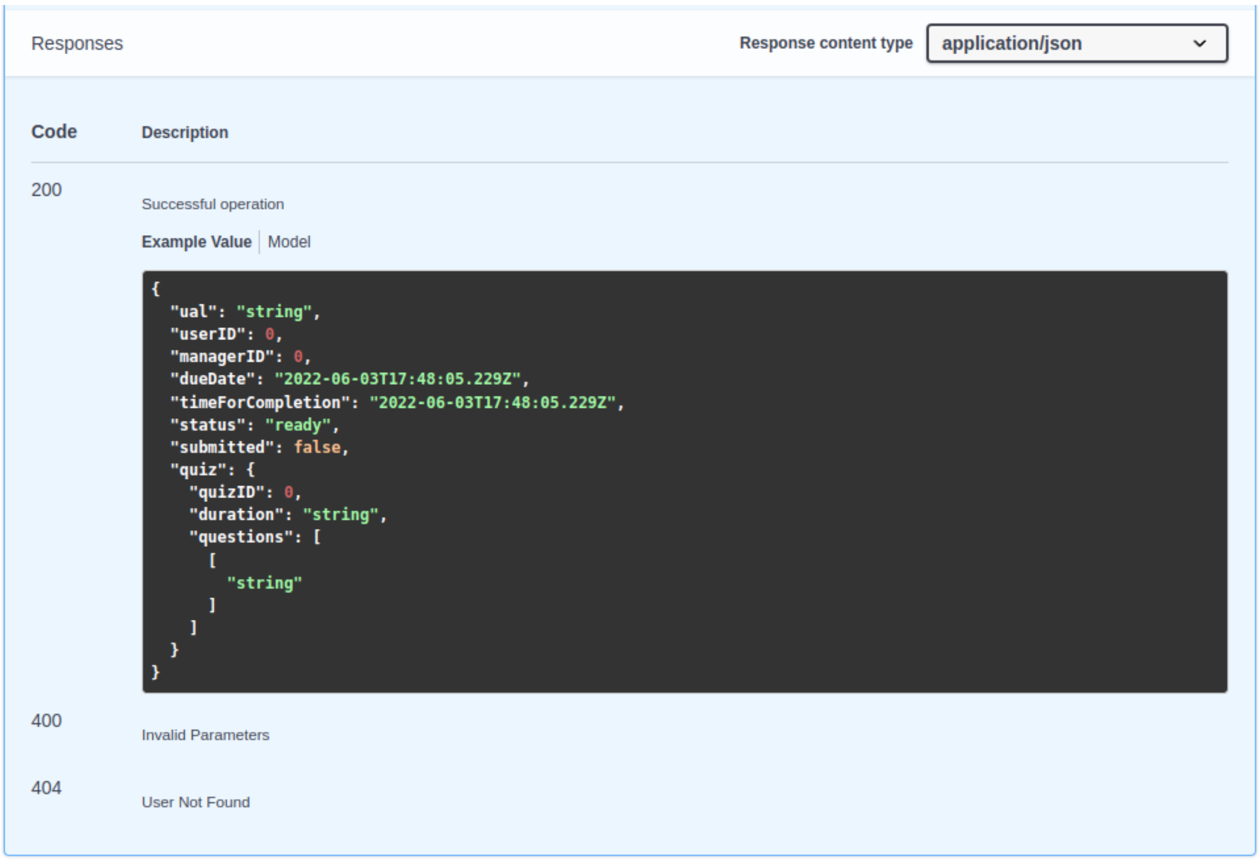
Try it out

Name	Description
managerID * required integer *(path)*	Manager ID managerID

Responses

Response content type: application/json

Figure 23


Responses

Response content type: application/json

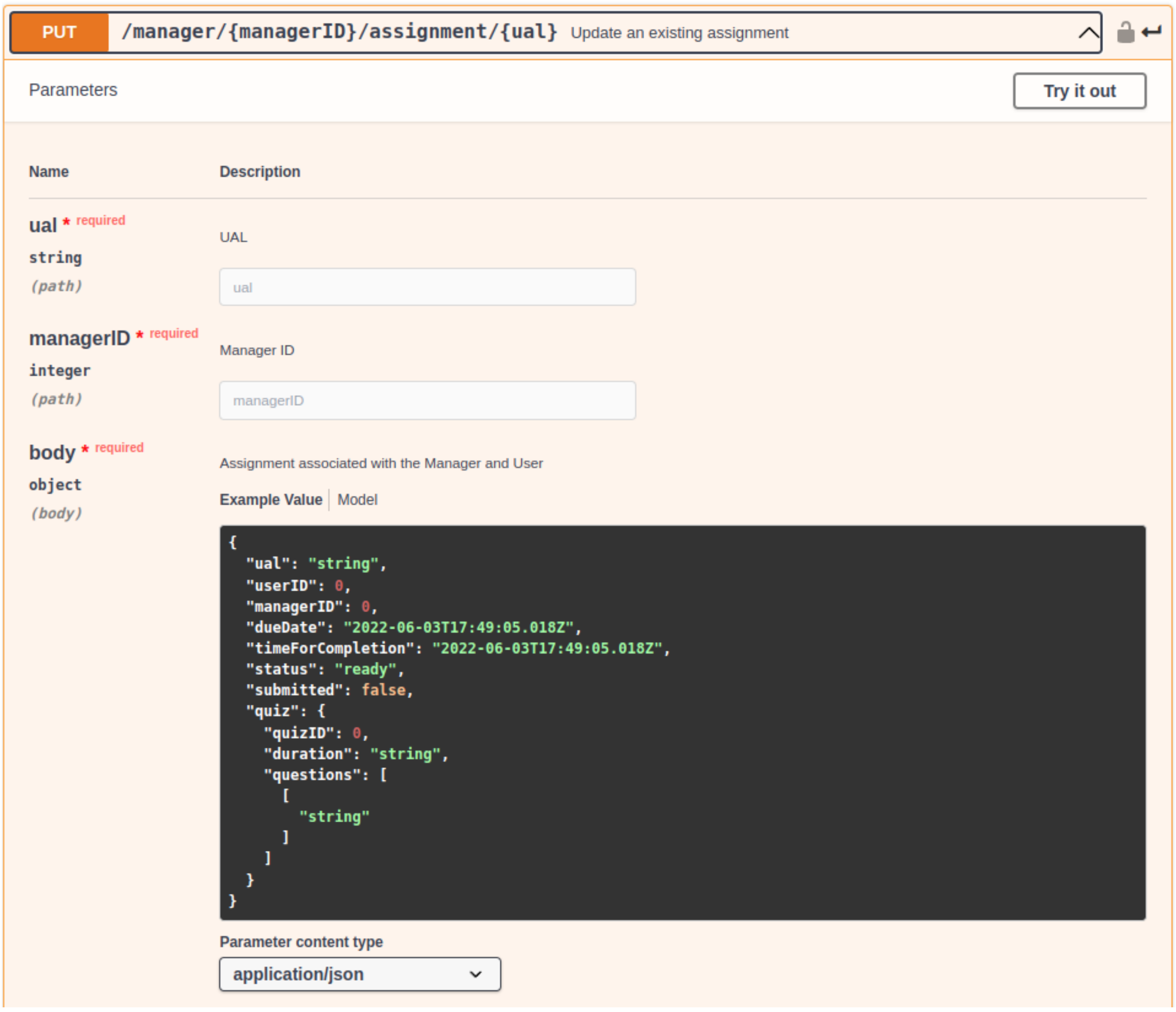
Code	Description
200	Successful operation
400	Invalid Parameters
404	User Not Found

Example Value | Model

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-03T17:48:05.229Z",
  "timeForCompletion": "2022-06-03T17:48:05.229Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Figure 24

2.2.3.6. Assignment PUT Endpoint w/ specified ual, for a manager w/ specified managerID


PUT /manager/{managerID}/assignment/{ual} Update an existing assignment

Parameters — Try it out

Name	Description
ual * required string (path)	UAL ual
managerID * required integer (path)	Manager ID managerID
body * required object (body)	Assignment associated with the Manager and User

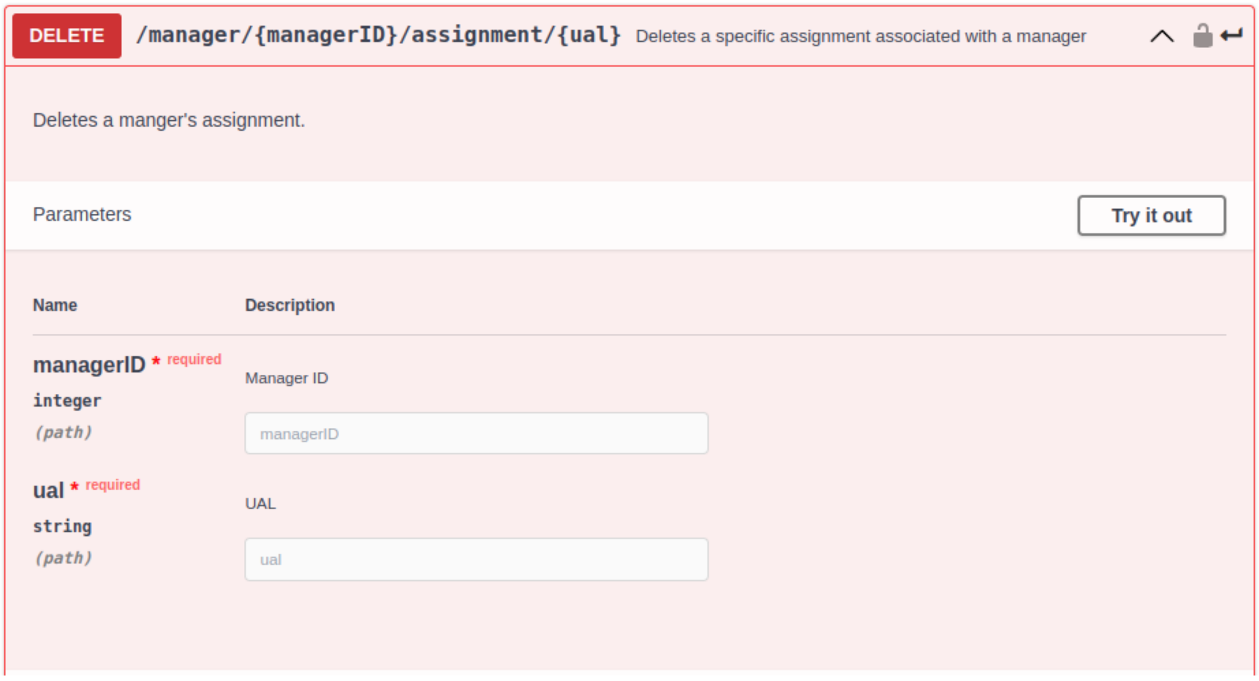
Example Value | Model

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-03T17:49:05.018Z",
  "timeForCompletion": "2022-06-03T17:49:05.018Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Parameter content type: application/json

Figure 25

2.2.3.7. Assignment PUT Endpoint w/ specified ual, for a manager w/ specified managerID


DELETE `/manager/{managerID}/assignment/{ual}` Deletes a specific assignment associated with a manager

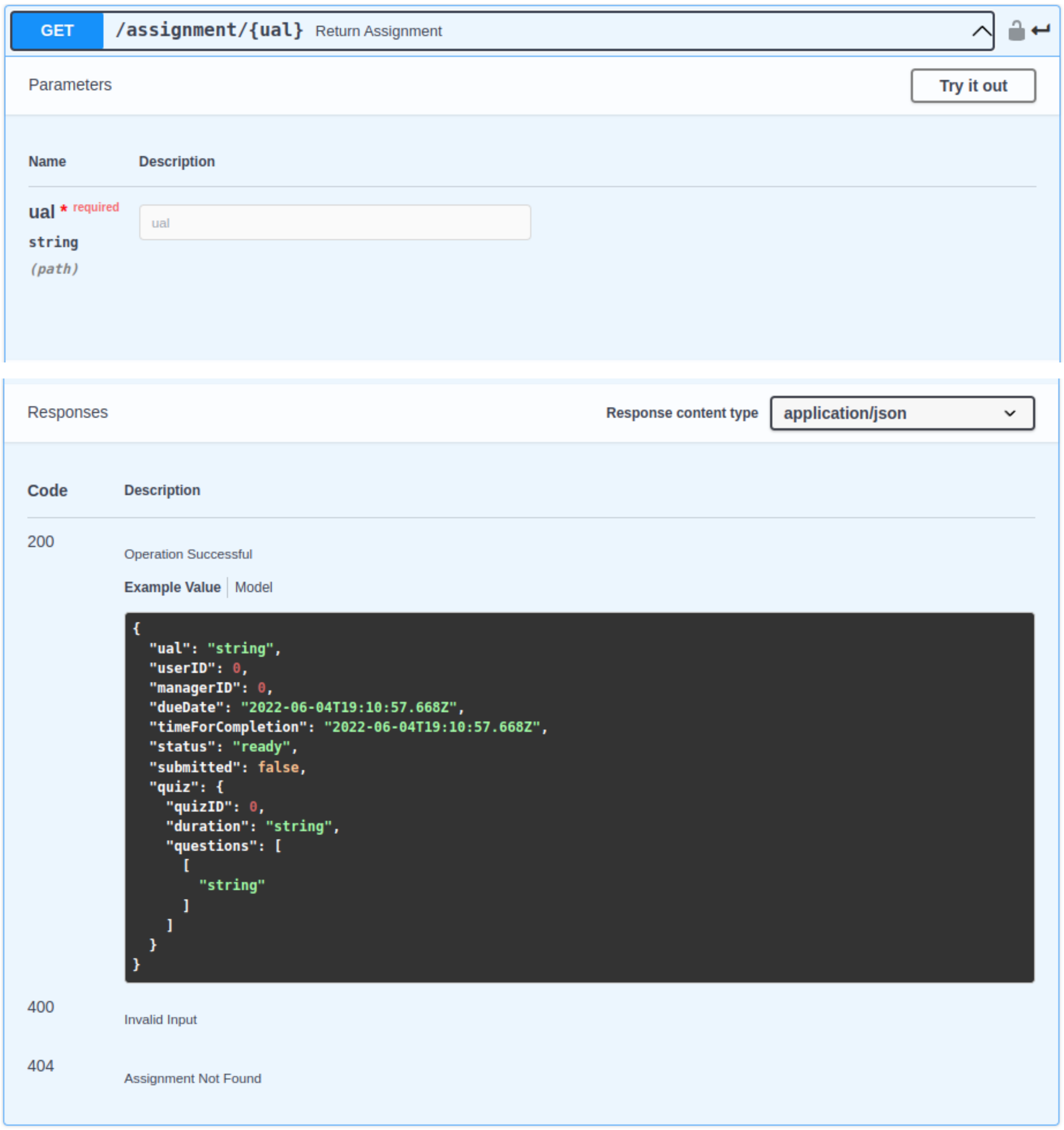
Deletes a manger's assignment.

Parameters

Name	Description
managerID * required integer (path)	Manager ID managerID
ual * required string (path)	UAL ual

Figure 26

2.2.3.8. Assignment GET Endpoint, w/ specified ual

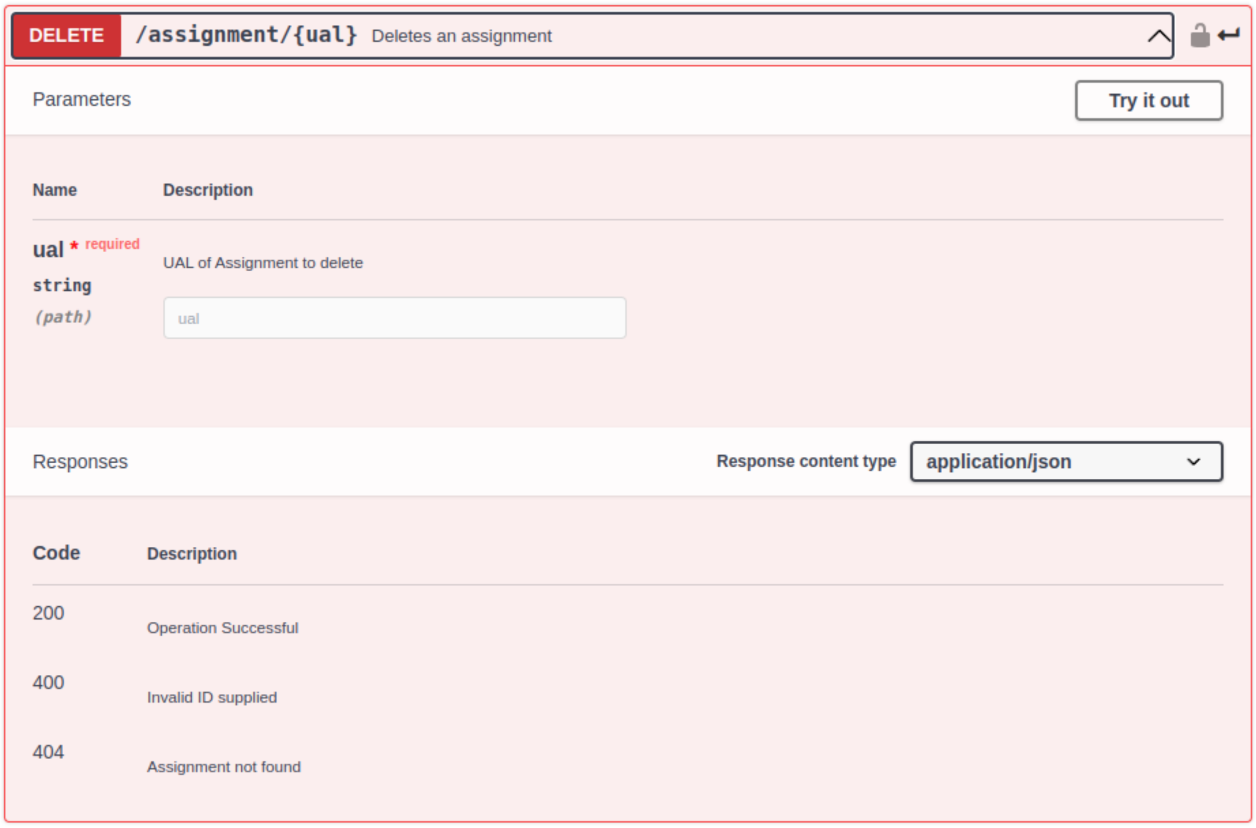


Code	Description
200	Operation Successful
400	Invalid Input
404	Assignment Not Found

```
{
  "ual": "string",
  "userID": 0,
  "managerID": 0,
  "dueDate": "2022-06-04T19:10:57.668Z",
  "timeForCompletion": "2022-06-04T19:10:57.668Z",
  "status": "ready",
  "submitted": false,
  "quiz": {
    "quizID": 0,
    "duration": "string",
    "questions": [
      [
        "string"
      ]
    ]
  }
}
```

Figure 27

2.2.3.9. Assignment DELETE Endpoint, w/ specified ual


DELETE /assignment/{ual} Deletes an assignment

Parameters

Try it out

Name	Description
ual * required string *(path)*	UAL of Assignment to delete ual

Responses

Response content type: application/json

Code	Description
200	Operation Successful
400	Invalid ID supplied
404	Assignment not found

Figure 28

3. System Implementation with Node-RED

3.1 Correlation of Rest Services with Node-RED flows

- Url for the zip file which contains the Node-RED flows

3.1.1. Node-RED Flows User

Flow endpoint GET /user/{userID}

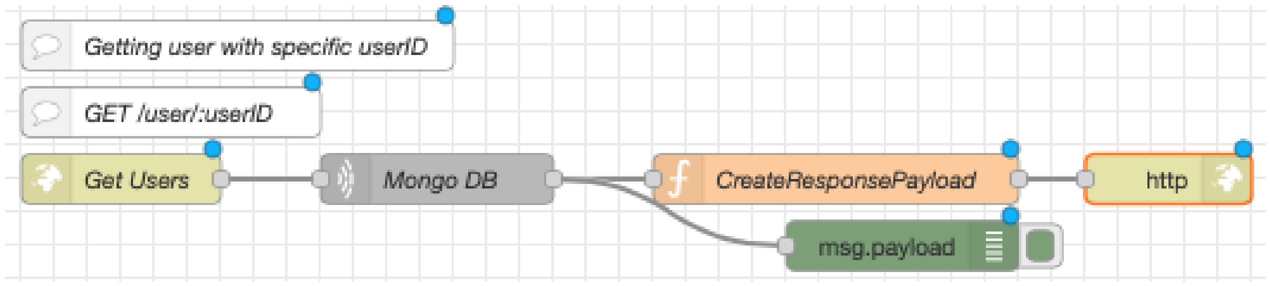


Figure 29

The flow that implements the service which is responsible for returning a user by using his unique ID.

Flow endpoint PUT /user/{userID}

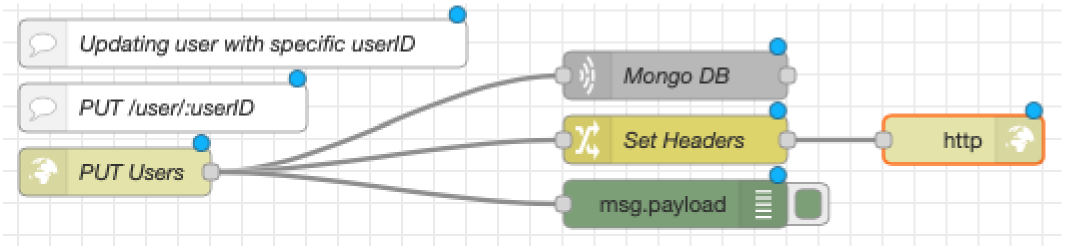
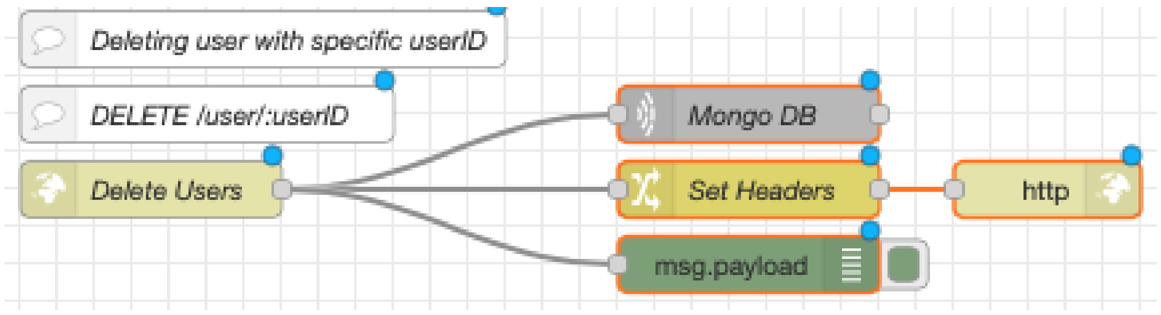


Figure 30

This flow implements the service, which is responsible for updating a specific user in the system using his unique ID.



Flow endpoint DELETE /user/{userID}

Figure 31

This flow implements the service, which is responsible for deleting a specific user in the system using his unique ID.

Flow endpoint POST /user

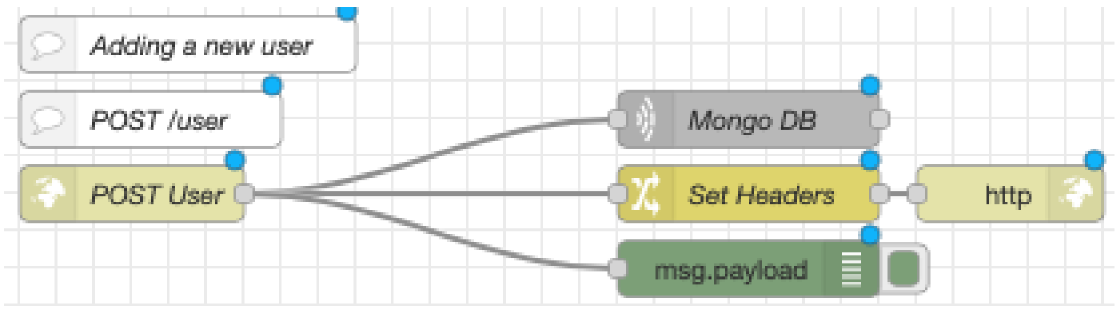


Figure 32

This flow implements the service, which is responsible for adding a new user in the system.

Flow endpoint GET /user/{userID}/assignment

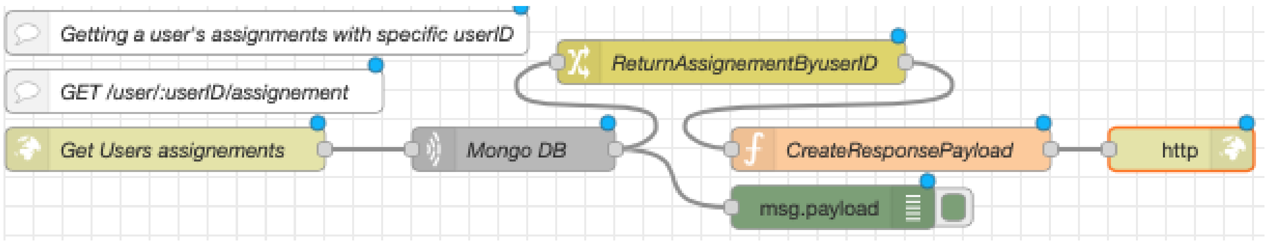

Getting a user's assignments with specific userID
GET /user/:userID/assignement
Get Users assignements
Mongo DB
ReturnAssignementByuserID
CreateResponsePayload
http
msg.payload

Figure 33

This flow implements the service, which is responsible for returning a list of assignments to the user, giving his unique ID as an entry. The function ReturnAssignementbyuserID returns the results from the user's request, when the function CreateResponsePayload, brings the http response to the required state.

3.1.2. Node-RED Flows Manager

Flow endpoint POST /manager

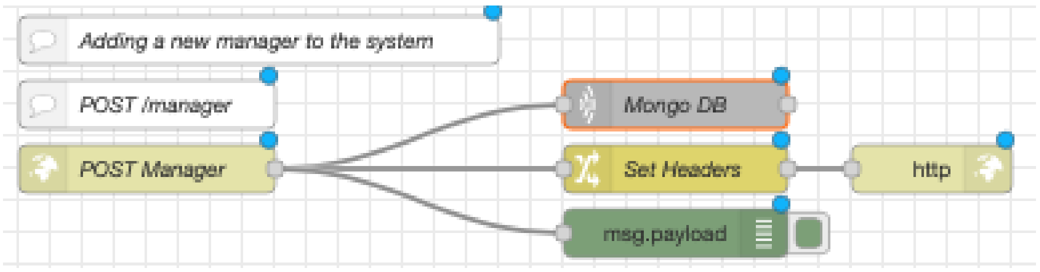


Figure 34

This flow implements the service, which is responsible for adding a new manager in the system.

Flow endpoint GET /manager/{managerID}

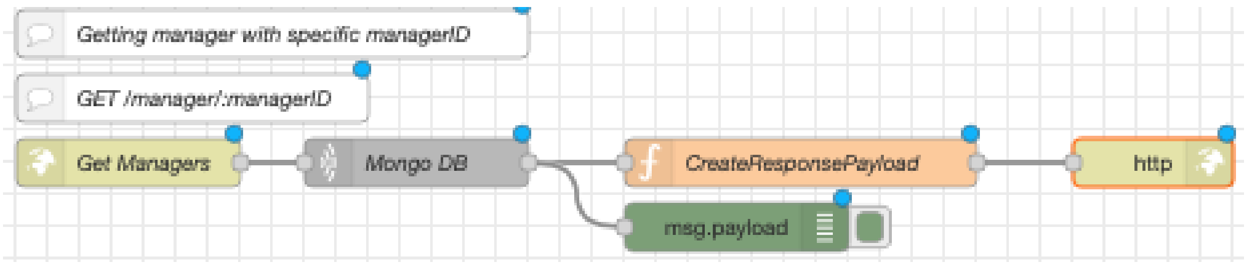


Figure 35

This flow implements the service, which is responsible for returning a specific manager in the system using his unique ID.

Flow endpoint PUT /manager/{managerID}

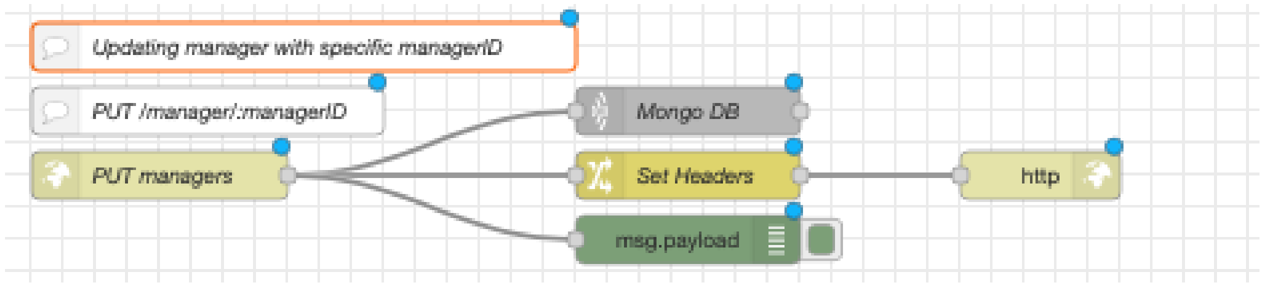


Figure 36

This flow implements the service, which is responsible for updating a specific manager in the system using his unique ID.

Flow endpoint DELETE /manager/{managerID}

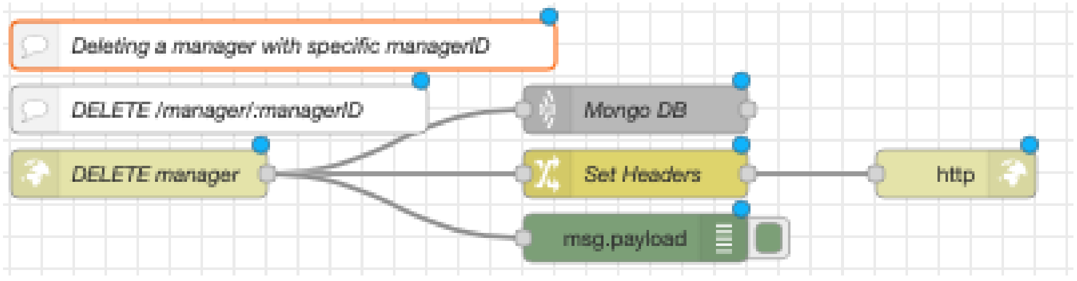

Deleting a manager with specific managerID
DELETE /manager/:managerID
DELETE manager
Mongo DB
Set Headers
msg.payload
http

Figure 37

This flow implements the service, which is responsible for deleting a specific manager in the system using his unique ID.

Flow endpoint GET /manager/{managerID}/assignment

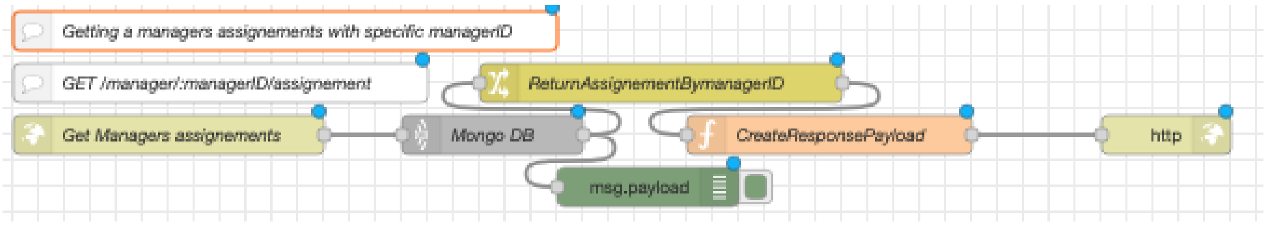

Getting a managers assignements with specific managerID
GET /manager/:managerID/assignement
Get Managers assignements
Mongo DB
ReturnAssignementBymanagerID
CreateResponsePayload
msg.payload
http

Figure 38

This flow implements the service, which is responsible for returning a list of assignments to the manager, giving as an entry his unique manager ID. The function ReturnAssignementbymanagerID returns the results from the manager request, when the function CreateResponse Payload, brings the http response to the required state.

Flow endpoint POST /manager/{managerID}/assignment

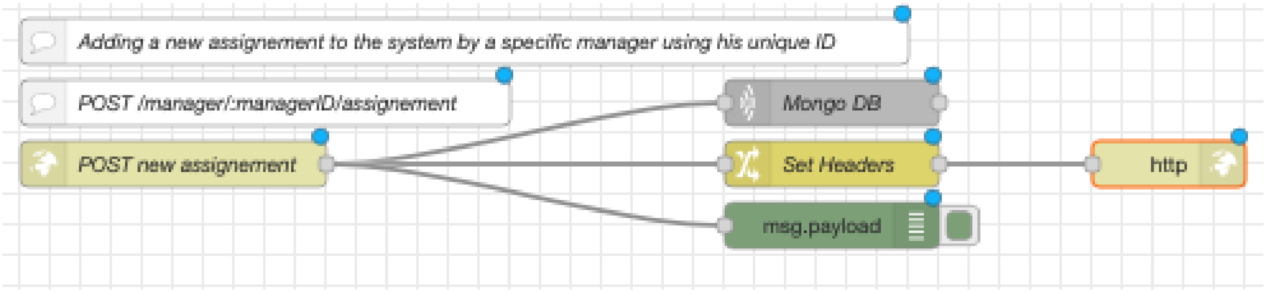

Adding a new assignement to the system by a specific manager using his unique ID
POST /manager/:managerID/assignement

Figure 39

This flow implements the service, through which it is possible for a manager to add a new assignment in the system using his unique ID.

Flow endpoint PUT /manager/{managerID}/assignment/{UAL}

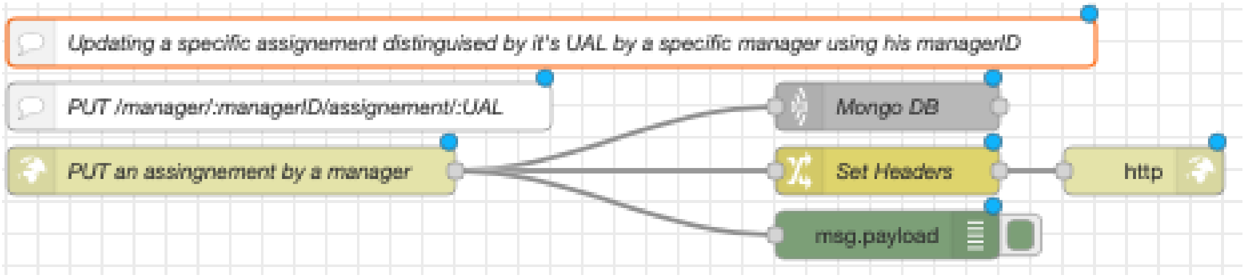

Updating a specific assignement distinguised by it's UAL by a specific manager using his managerID
PUT /manager/:managerID/assignement/:UAL

Figure 40

This flow implements the service, through which is possible for a manager to update an assignment in the system using his unique ID and the assignment's UAL.

Flow endpoint DELETE /manager/{managerID}/assignment/{UAL}

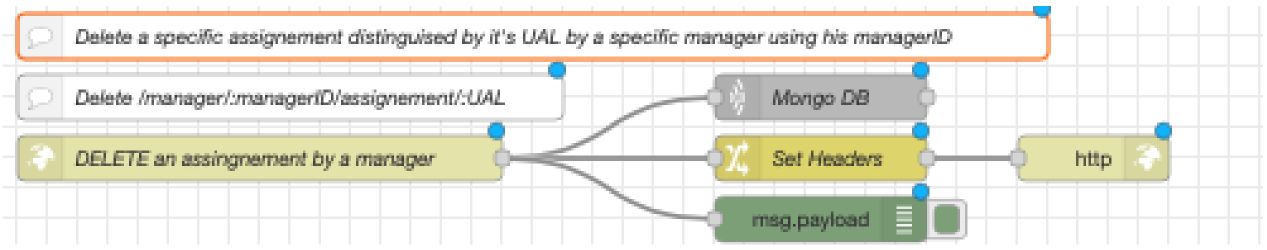

Delete a specific assignement distinguised by it's UAL by a specific manager using his managerID
Delete /manager/:managerID/assignement/:UAL
DELETE an assingnement by a manager
Mongo DB
Set Headers
http
msg.payload

Figure 41

This flow implements the service, through which is possible for a manager to delete an assignment in the system using his unique ID and the assignment's UAL.

3.1.3. Node-RED Flows Assignment

Flow endpoint GET /assignment/{UAL}

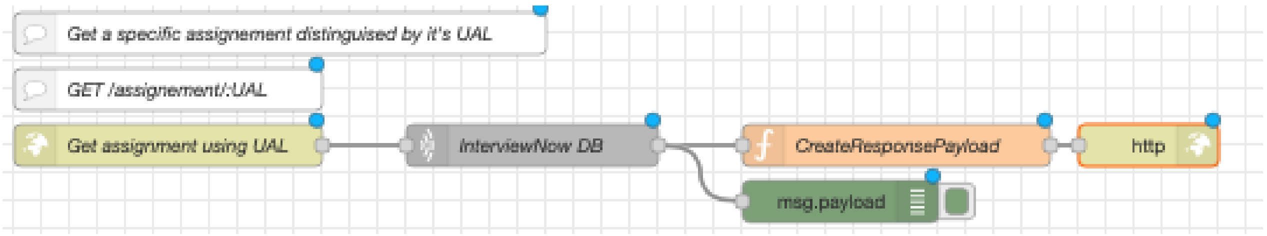


Figure 42

The flow that implements the service which is responsible for returning an assignment using its UAL.

Flow endpoint DELETE /assignment/{UAL}

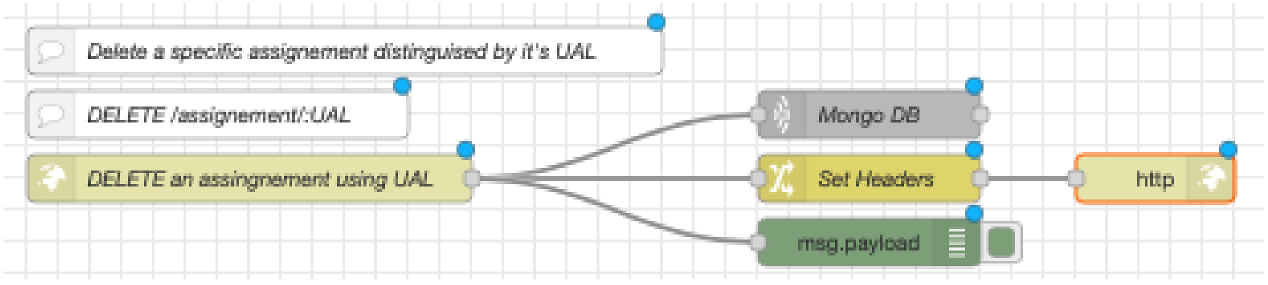

Delete a specific assignement distinguised by it's UAL
DELETE /assignement/:UAL
DELETE an assingnement using UAL
Mongo DB
Set Headers
http
msg.payload

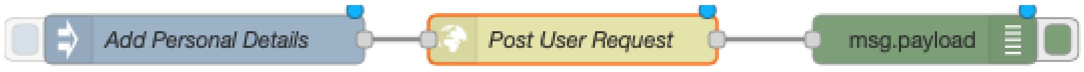
Figure 43

The flow that implements the service which is responsible for deleting an assignment using its UAL.

3.2. Use Case Implementation

3.2.1. Use Case Scenario User Add Personal Details

Flow through which the user can add his personal details.

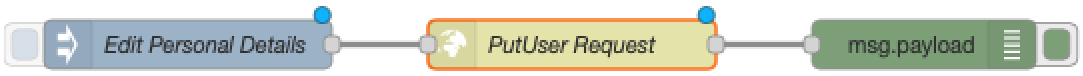


Node Name	Node Type	Description
AddPersnoalDetails	Inject	It's used to start the execution of the flow
PostUserRequest	http-request	It calls the service which is responsible for altering a user..
Msg.payload	debug	Prints in the console the new details of the user.

Figure 44

3.2.2. Use Case Scenario User Edit Personal Details

Flow through which the user can add his personal details.

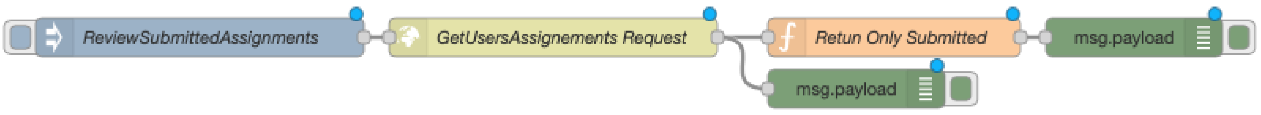


Node Name	Node Type	Description
EditPersnoalDetails	Inject	It's used to start the execution of the flow
PutUsersRequest	http-request	It calls the service which is responsible for altering a user..
Msg.payload	debug	Prints in the console the new details of the user.

Figure 45

3.2.3. Review Submitted Assignments

Flow through which the user can review his Submitted assignments details.



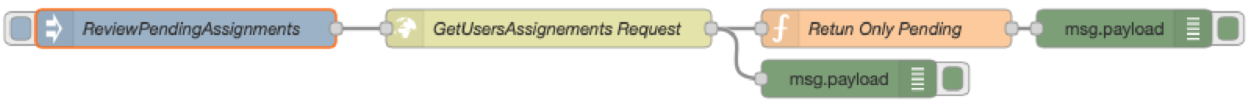
Node Name	Node Type	Description
ReviewSubmitted Assignments	Inject	It's used to start the execution of the flow
GetUsers Assignments Request	http-request	It calls the service which is responsible for viewing all of the user's assignments.

Node Name	Node Type	Description
Return Only Submitted	Function	This function filters all the assignments attached to the user's ID and shows only the submitted ones.
Msg.payload	debug	Prints in the console a list of users assignments
Msg.payload	debug	Prints in the console a list of user's submitted assignments

Figure 46

3.2.4. Review Pending Assignments

Flow through which the user can review his Pending assignments details.

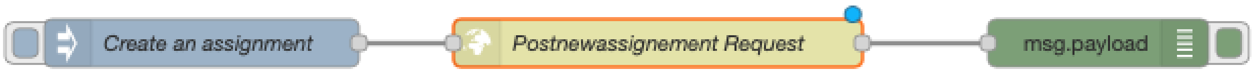


Node Name	Node Type	Description
ReviewPendingAssignments	Inject	It's used to start the execution of the flow
GetUsers Assignments Request	http-request	It calls the service which is responsible for viewing all of the user's assignments.
Return Only Pending	Function	This function filters all the assignments attached to the user's ID and shows only the submitted ones.
Msg.payload	debug	Prints in the console a list of users assignments
Msg.payload	debug	Prints in the console a list of user's pending assignments

Figure 47

3.2.5. Create an Assignment

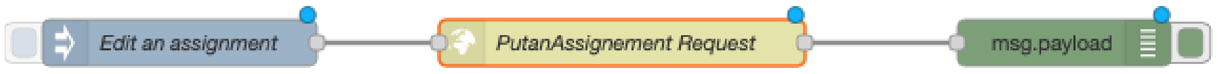
Flow through which the manager can create a new assignment.



Node Name	Node Type	Description
Create an assignment	Inject	It's used to start the execution of the flow
Postnewassignement Request	http-request	It calls the service which is responsible for creating a new assignment.
Msg.payload	debug	Prints in the console the newly created assignment

Figure 48

3.2.6. Edit an Assignment



Flow through which the manager can create a new assignment.

Node Name	Node Type	Description
Edit an assignment	Inject	It's used to start the execution of the flow
PutanAssignement Request	http-request	It calls the service which is responsible for altering an assignment by a specific manager who uses the UAL.
Msg.payload	debug	Prints in the console the newly altered assignment

Figure 49

Appendix - Open Issues

The flows regarding the user scenarios with the involvement of the Notification system are not included, since they are not a part of the main system.

There would be a major overhaul of the class diagrams and design from the last deliverable in order to facilitate the design patterns.