User Guide

This is a guide to use the REST API to manage the logistics of an organization's fleet of spaceships.

I have implemented two endpoints for the travel functionality, one with payload and one without payload. Both carry out exactly the same functionality.

This table provides the functionalities of API

	Functionalities	Type	Endpoint
	Location related functions		
1	User can add a new location	POST	"/locations/{id}"
2	User can delete a location	DELETE	"/locations/{id}"
3	User can view a location	GET	"/locations/{id}"
4	User can view details of all locations	GET	"/locations"
	Spaceship related functions		
5	User can add spaceship	POST	"/spaceships/{id}"
6	User can remove spaceship	DELETE	"/spaceships/{id}"
7	User can edit status of spaceships	PUT	"/spaceships/{id}"
8	User can view spaceship	GET	"/spaceships/{id}"
9	User can view all spaceships	GET	"/spaceships"
	Travel related		
10	User can travel with payload	POST	“/travel”
11	Travel function without payload	POST	“/travel/{spaceship_id}/{location_id}”

1. User can add a new location:

Description: let the user add a new location to our database.

Type: POST

Endpoint: "/locations/{id}"

Payload: {
"city":"string",
“Planet": "EARTH",
"capacity": 0
}

Restriction in payload: Planet can only be one of
['EARTH', 'JUPITER', 'MARS', 'NEPTUNE', 'SATURN', 'VENUS', 'MERCURY', 'URANUS']

Response: 201 Created,
`'location added successfully'`

400 Bad request
`'location with id already exist in database'`

2. User can delete a location:

Description: let the user remove a location, given that there are no spaceships stationed at those locations

Type: POST

Endpoint: “/locations/{id}”

Response: 200 Success
`'spaceship removed successfully'`

404 Not found
`'spaceship with id do not exist in database'`

3. User can view details of a location:

Description: let the user view information about a location,
User can also see information of spaceships stationed at the location.

Type: GET

Endpoint: “/locations/{id}”

Response Codes: 200 Success,
404 Not found

Response example: {


```
"id": 0,
"city": "string",
"planet": "string",
"capacity": 0,
"availability": 0,
"spaceships": [
 {
  "id": "string",
  "name": "string",
  "model": "string",
  "status": "string"
 }
]
}
```


4. User can view details of all locations

Description: User can view details of all locations, along with details of spaceships stationed at those locations.

Type: POST

Endpoint: “/locations”

Response Code: 200 Success ,
400 Bad request

Response example: {

```
"locations": [
 {
  "id": 0,
  "city": "string",
  "planet": "string",
  "capacity": 0,
  "availability": 0,
  "spaceships": [
   {
    "id": "string",
    "name": "string",
    "model": "string",
    "status": "string"
   }
  ]
 }
]
}
```

5. User can add spaceship:

Description: User can add a new spaceship to an existing location .

Type: POST

Endpoint: “/spaceships/{id}”

Payload:{

```
"name": "string",
 "model": "string"
 "city": "string"
 "planet": "string",
  "status": "OPERATIONAL"
 }
```

Restriction in payload: status can only accept one of

['OPERATIONAL', 'MAINTENANCE', 'DECOMMISSIONED']

Response: 201 Created

```
'spaceship added successfully'
```

400 Bad request

```
'spaceship with id already exists in database'
'this location is at its maximum capacity'
'please enter a valid city and planet'
'this location is at its maximum capacity'
'Invalid status'
```

6. User can remove spaceship:

Description: User can remove a spaceship, given id .

Type: DELETE

Endpoint: “/spaceships/{id}”

Responses: 200 Success

```
'spaceship removed successfully '
```

404 Not found.

```
'spaceship with id do not exist in database'
```

7. User can edit status of spaceship:

Description: User can edit status of a spaceship, given id.

Type: POST

Endpoint: “/spaceships/{id}”

Payload model: {

"status": "string"

}

Responses: 200 Success.

```
'spaceship is already in given status'
```

400 Bad request

```
'spaceship is already in given status'
```

401 Not found
`‘spaceship with id do not exist in database'`

8. User can view spaceship:

Description: User can view information of a spaceship, including where it is located.

Type: GET

Endpoint: “/spaceships/{id}”

Response example: {

```
"id": "string",
"name": "string",
"model": "string",
"city": "string",
"planet": "string",
"status": "string"
}
```

Response: 200 Success.
404 Not Found.

9. User can view all spaceships:

Description: User can view information of all spaceships including where those are currently located.

Type: GET

Endpoint: “/spaceships”

Response example : {

```
"spaceships": [
 {
   "id": "string",
   "name": "string",
```

```
    "model": "string",
    "city": "string",
    "planet": "string",
    "status": "string"
   }
  ]
 }
```

Response: 200 Success.

400 Bad request.

10. User can travel with payload

Description: This is a function for spaceships to travel to different locations.

Type: POST

Endpoint: “/travel”

Payload Example:{

```
"name": "string",
"model": "string",
"destination_city": "string",
"destination_planet": "string"
}
```

Responses: 200 Success.

‘travel is successful’

'spaceship already at destination location'

400 Bad request.

'spaceship is not operational '

‘location is at maximum capacity, travel unsuccessful'

'location with id do not exist in database'

404 Not found

'spaceship with id do not exist in database'

11. User can travel without payload

Type: POST

Endpoint: “/travel/{spaceship_id}/{location_id}”

Responses: 200 Success.

```
'travel is successful'
'spaceship already at destination location'
```

400 Bad request.

```
'spaceship is not operational '
'location is at maximum capacity’
‘travel Unsuccessful'
```

404 Not found

```
'location not found,check city name and planet name'
'spaceship not found, check name and model of
  spaceship'
'location and spaceship not found'
```