Assignment 4: Intermediate REST Implementation

CS 493: Cloud Application Development

Fall 2019

Oregon State University

Ryan Wallerius

URLs

<https://wallerirassignment4.appspot.com/boats>

<https://wallerirassignment4.appspot.com/loads>

# Create a Boat

Allows you to create a new boat.

|  |
| --- |
| POST /boats |

## Request

### Request Parameters

None

### Request Body

Required

### Request Body Format

JSON

### Request JSON Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| name | String | The name of the boat. | Yes |
| type | String | The type of the boat. E.g., Sailboat, Catamaran, etc. | Yes |
| length | Integer | Length of the boat in feet. | Yes |

### Request Body Example

|  |
| --- |
| {  "name": "Sea Witch",  "type": "Catamaran",  "length": 28  } |

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 201 Created |  |
| Failure | 400 Bad Request | If the request is missing any of the 3 required attributes, the boat must not be created, and 400 status code must be returned.  You don’t need to validate the values of the attributes and can assume that if the request contains any of the listed attributes, then that attribute’s value is valid.  You can also assume that the request will not contain any extraneous attribute (i.e., the request JSON will never contain any attribute other than the ones that are listed). |

### Response Examples

* Datastore will automatically generate an ID and store it with the entity being created. This value needs to be sent in the response body as shown in the example.
* Loads will be initialized to zero.
* The self attribute will contain the live link to the REST resource corresponding to this boat. In other words, this is the URL to get this newly created boat. You must not store the self attribute in Datastore

#### Success

|  |
| --- |
| Status: 201 Created  {  "id": "abc123",  "name": "Sea Witch",  "type": "Catamaran",  "length": 28,  "self": "https://<your-app>/boats/abc123"  } |

#### Failure

|  |
| --- |
| Status: 400 Bad Request  {  "Error": "The request object is missing at least one of the required attributes"  } |

# Get a Boat

Allows you to get an existing boat by id number

|  |
| --- |
| GET /boats/:boat\_id |

## Request

### Request Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| boat\_id | String | ID of the boat | Yes |

### Request Body

None

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK |  |
| Failure | 404 Not Found | No boat with this boat\_id exists |

### Response Examples

#### Success

|  |
| --- |
| Status: 200 OK  {  "id": "abc123",  "name": "Sea Witch",  "type": "Catamaran",  "length": 28,  "self": "https://<your-app>/boats/abc123"  } |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No boat with this boat\_id exists"  } |

# List all Boats

List all the boats.

|  |
| --- |
| GET /boats |

## Request

### Request Parameters

None

### Request Body

None

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK |  |

### Response Examples

#### Success

|  |
| --- |
| Status: 200 OK  [  {  "id": "abc123",  "name": "Sea Witch",  "type": "Catamaran",  "length": 28,  “loads”: []  },  {  "id": "def456",  "name": "Adventure",  "type": "Sailboat",  "length": 50,  “loads”: [],  },  {  "id": "xyz123",  "name": "Hocus Pocus",  "type": "Sailboat",  "length": 100,  “loads”: [],  }  ],  “next”: “https://<your-app>/boats?cursor..... |

This project had us implement pagination. So, after showing 3 results, it will show a “next” value along with a link to get the user to the next page that holds more results.

# Edit a Boat

Allows you to edit a boat.

|  |
| --- |
| PUT /boats/:boat\_id |

## Request

### Request Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| boat\_id | String | ID of the boat | Yes |

### Request Body

Required

### Request Body Format

JSON

### Request JSON Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| name | String | The name of the boat. | Yes |
| type | String | The type of the boat. E.g., Sailboat, Catamaran, etc. | Yes |
| length | Integer | Length of the boat in feet. | Yes |

### Request Body Example

|  |
| --- |
| {  "name": "Sea Witch",  "type": "Catamaran",  "length": 99  } |

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK |  |
| Failure | 400 Bad Request | If the request is missing any of the 3 required attributes, the boat must not be updated, and 400 status code must be returned.  You don’t need to validate the values of the attributes and can assume that if the request contains any of the listed attributes, then that attribute’s value is valid.  You can also assume that the request will not contain any extraneous attribute (i.e., the request JSON will never contain any attribute other than the ones that are listed). |
| Failure | 404 Not Found | No boat with this boat\_id exists |

### Response Examples

* The self attribute will contain the live link to the REST resource corresponding to this boat. You must not store this attribute in Datastore.

#### Success

|  |
| --- |
| Status: 200 OK  {  "id": "abc123",  "name": "Sea Witch",  "type": "Catamaran",  "length": 99,  "self": "https://<your-app>/boats/abc123"  } |

#### Failure

|  |
| --- |
| Status: 400 Bad Request  {  "Error": "The request object is missing at least one of the required attributes"  } |

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No boat with this boat\_id exists"  } |

This is not going to update the boats load as that is separate than what this function can provide you. Send request twice to see updated boat in response body.

# Delete a Boat

Allows you to delete a boat. By deleting a boat, if there is a load on it, then that load will become unassigned. That means the load will still exist in the datastore, just not assigned to that boat.

|  |
| --- |
| DELETE /boats/:boat\_id |

## Request

### Request Parameters

None

### Request Body

None

## Response

No body

### Response Body Format

Success: No body

Failure: JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 204 No Content |  |
| Failure | 404 Not Found | No boat with this boat\_id exists |

### Response Examples

#### Success

|  |
| --- |
| Status: 204 No Content |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No boat with this boat\_id exists"  } |

# Create a Load

Allows you to create a new load.

|  |
| --- |
| POST /loads |

## Request

### Request Parameters

None

### Request Body

Required

### Request Body Format

JSON

### Request JSON Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| weight | Integer | The weight of the boat | Yes |
| content | String | What is contained in the load | Yes |
| Delivery date | String | The expected delivery date of the load | Yes |

### Request Body Example

|  |
| --- |
| {  “weight”: 5,  “content”: “LEGO Blocks”,  “delivery\_date”: “1-1-2020”  } |

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 201 Created |  |
| Failure | 400 Bad Request | If the request is missing the number attribute, the slip must not be created, and 400 status code must be returned.  You don’t need to validate the values of this attribute and can assume that if the number attribute is specified, then its value is valid.  You can also assume that the request will not contain any extraneous attribute (i.e., the request JSON will never contain any attribute other than number). |

### Response Examples

* Datastore will automatically generate an ID and store it with the entity being created. This value needs to be sent in the response body as shown in the example.
* The value of the attribute self is a live link to the REST resource corresponding to this load. In other words, this is the URL to get this newly created load.

#### Success

|  |
| --- |
| Status: 201 Created  {  "id": "123abc",  "weight": 5,  "content": “LEGO Blocks”,  "delivery date": “1-1-2020”,  "self": "https://<your-app>/loads/123abc"  } |

#### Failure

|  |
| --- |
| Status: 400 Bad Request  {  "Error": "The request object is missing the required number"  } |

# Get a Load

Allows you to get an existing load by its id number.

|  |
| --- |
| GET /loads/:load\_id |

## Request

### Request Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required?** |
| load\_id | String | ID of the slip | Yes |

### Request Body

None

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK |  |
| Failure | 404 Not Found | No slip with this slip\_id exists |

### Response Examples

#### Success

|  |
| --- |
| Status: 200 OK  {  "id": "123abc",  "weight": 1,  “content”: “Amazing content”,  “delivery date”: “As soon as possible”,  "self": "https://<your-app>/slips/123abc"  } |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No slip with this slip\_id exists"  } |

My program does not return the id of the boat that this load is on. I was not able to get that implemented in time.

# List all Loads

List all the loads

|  |
| --- |
| GET /loads |

## Request

### Request Parameters

None

### Request Body

None

## Response

### Response Body Format

JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK |  |

### Response Examples

#### Success

|  |
| --- |
| Status: 200 OK  [  {  "id": "123abc",  "weight": 1,  “content”: “Amazing content”,  “delivery date”: “As soon as possible”,  "self": "https://<your-app>/slips/123abc"  },  {  "id": "456abc",  "weight": 1,  “content”: “Amazing content”,  “delivery date”: “As soon as possible”,  "self": "https://<your-app>/slips/456abc"  },  {  "id": "456abc",  "weight": 1,  “content”: “Amazing content”,  “delivery date”: “As soon as possible”,  "self": "https://<your-app>/slips/456abc"  }  ],  “next”: “https://<your-app>/loads?cursor=..... |

We also needed to implement pagination for showing all the loads that have been created. Needs to show them in groups of 3 and then a link to the next set.

# Delete a Load

Allows you to delete a load.

|  |
| --- |
| DELETE /loads/:load\_id |

## Request

### Request Parameters

None

### Request Body

None

## Response

No body

### Response Body Format

Success: No body

Failure: JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 204 No Content |  |
| Failure | 404 Not Found | No load with this load\_id exists |

### Response Examples

#### Success

|  |
| --- |
| Status: 204 No Content |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No load with this load\_id exists"  } |

# Edit a Load

Allows you to delete a load.

|  |
| --- |
| PUT /loads/:load\_id |

## Request

### Request Parameters

None

### Request Body

None

## Response

No body

### Response Body Format

Success: No body

Failure: JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 204 No Content |  |
| Failure | 404 Not Found | No load with this load\_id exists |

### Response Examples

#### Success

|  |
| --- |
| Status: 204 No Content |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "No load with this load\_id exists"  } |

# Assign a Load to a Boat

A load has been assigned to a boat

|  |
| --- |
| PUT /boats/:boat\_id/loads/:load\_id |

## Request

### Request Parameters

None

### Request Body

None

## Response

No body

### Response Body Format

Success: No body

Failure: JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 No Content | Succeeds when a load is successfully assigned to a boat |
| Failure | 404 Not Found | No boat with this boat\_id exists, and/or no load with this load\_id exits. |

### Response Examples

#### Success

|  |
| --- |
| Status: 204 No Content |

#### Failure

|  |
| --- |
| Status: 404 Not Found  {  "Error": "The specified boat and/or slip don’t exist"  } |

There does seem to be a bug in my code where the status 404 does not occur. Have not found the reason why the 404 feature is not working properly.

# Get a Boat’s Load

A load has been assigned to a boat

|  |
| --- |
| GET /boats/:boat\_id/loads |

## Request

### Request Parameters

None

### Request Body

None

## Response

No body

### Response Body Format

Success: No body

Failure: JSON

### Response Statuses

|  |  |  |
| --- | --- | --- |
| **Outcome** | **Status Code** | **Notes** |
| Success | 200 OK | Succeeds when there is a load on the boat |

### Response Examples

#### Success

|  |
| --- |
| Status: 200 OK  [  {  “content”: “asdf3”,  “weight”: 10,  “delivery date”: “1-1-2020”,  “id”: “123abc  }  ] |

I was unable to implement an error check for if the boat doesn’t have a load for this particular test. It is caught by other tests however.