# Fuel consumption management application – RESTful API documentation

## Assumptions

* Users can register consumptions in past, present or future dates
* When required to group or filter by month, the assumption is that month means month of the current year
* *Statistics for each month, list fuel consumption records grouped by fuel type (each row should contain: fuel type, volume, average price, total price)*
  + Volume = should be an aggregate value, so here is interpreted as **total volume**

## Endpoints

### Create new consumption

|  |  |
| --- | --- |
| URL | /consumptions |
| Method | POST |
| URL params |  |
| Request headers | Content-type: application/json |
| Data params | *{*  *“****fuelType****”:* string (required, accepted values: D, 95 or 98),  *"****price****”:* number (required, positive),  *"****volume****”:* number (required, positive),  *"****date****”:* string (required, format: yyyy-MM-dd),  *"****driverId****":* string (required)  *}* |
| Success response | Status: 201 Created  *Returns the id of the newly created consumption as location header.*  Headers:   * **Location**: /consumptions/{id} |
| Error response | Status: 400 Bad Request  *Returns details how the request is not correct in the response body as JSON document.*  *{*  *“****message****”:* string, (human readable error message)  *“****fieldsWithError****”:* array of strings (fields that are missing, malformed or in invalid format)  *}* |
| Example | *{*  *"fuelType": "D",*  *"price": 4.56,*  *"volume": 5,*  *"date": "2014-04-18",*  *"driverId": "mario"*  *}* |

### Create consumptions batch

|  |  |
| --- | --- |
| URL | /consumptions/batch |
| Method | POST |
| URL params |  |
| Request headers |  |
| Data params | Body: form-data  Key = file value=<consumptions file>  <consumption file> is a JSON formatted file with list of consumptions to create |
| Success response | Status: 200 Ok  Returns array of objects containing URI of the objects created  *[*  *{*  *"consumptionURI": "http://localhost:8080/consumptions/84"*  *},*  *…*  *]* |
| Error response | Status: 400 Bad Request  Returns details how the request is not correct in the response body as JSON document.  *{*  *"message": "Error while parsing or mapping the input to a list of consumptions"*  *}* |
| Example | See project test resources |

### Get consumptions by month and driver id

|  |  |
| --- | --- |
| URL | /consumptions |
| Method | GET |
| URL params | *month* (required valid values: numbers in the range [1-12])  *driverId* (optional) |
| Request headers |  |
| Data params |  |
| Success response | Status: 200 OK  Array of objects:  [  {  "***fuelType***": string,  "***volume***": number,  "***date***": string,  "***price***": number,  "***totalPrice***": number, (calculated as volume x price)  "***driverId***": string  }  ]  If there are no consumptions for the requested month/driverId returns empty array |
| Error response | Missing or invalid format month: 400 Bad Request |
| Example |  |

### Get statistics

|  |  |
| --- | --- |
| URL | /statistics |
| Method | GET |
| URL params | *driverId* (optional) |
| Request headers |  |
| Data params |  |
| Success response | Status: 200 OK  Object  {  "***consumptionsTotalAmountByMonth***": object,  "***consumptionsStatisticsByMonth***": object  }  ***consumptionsTotalAmountByMonth***  Map: each entry has as key the year month and as value the amount of money spent.  {  “yyyy-MM”: number  }  ***consumptionsStatisticsByMonth***  Map: each entry has as key the year month and as value an array of objects. Each object is a fuel type group with aggregated values: total volume, average price and total price.  {  “yyyy-MM”: [{  "***fuelType***": string,  "***totalVolume***": number,  "***averagePrice***": number,  "***totalPrice***": number  }]  }  If there are no consumptions for the requested driverId, the API returns empty objects:  {  "consumptionsTotalAmountByMonth": {},  "consumptionsStatisticsByMonth": {}  } |
| Example response | {  "consumptionsTotalAmountByMonth": {  "2018-03": 7.81,  "2018-02": 52.37,  "2018-04": 83.4  },  "consumptionsStatisticsByMonth": {  "2018-05": [  {  "fuelType": "D",  "totalVolume": 22,  "averagePrice": 1.24,  "totalPrice": 1.24  },  {  "fuelType": "98",  "totalVolume": 32,  "averagePrice": 5.44,  "totalPrice": 5.44  }  ],  "2018-03": [  {  "fuelType": "D",  "totalVolume": 21,  "averagePrice": 1.19,  "totalPrice": 2.37  },  {  "fuelType": "98",  "totalVolume": 2,  "averagePrice": 5.44,  "totalPrice": 5.44  }  ],  "2018-02": [  {  "fuelType": "D",  "totalVolume": 22,  "averagePrice": 1.14,  "totalPrice": 3.41  },  {  "fuelType": "95",  "totalVolume": 65,  "averagePrice": 9.29,  "totalPrice": 37.16  },  {  "fuelType": "98",  "totalVolume": 6,  "averagePrice": 5.9,  "totalPrice": 11.8  }  ],  "2018-04": [  {  "fuelType": "D",  "totalVolume": 50,  "averagePrice": 20.85,  "totalPrice": 83.4  }  ]  }  } |
|  |  |

## Application Architecture

### Framework used

Java 8

Spring Boot

Spring Data JPA

H2 embedded database

Model mapper for basic mapping of DTOs to Entity and viceversa

JUnit for unit tests

Maven

### Modules

The project is structured in:

* use cases/resources: application specific business rules; each use case correspond to a package with controller resource, service and DTOs
  + **consumptions**: save a consumption, save consumptions in batch mode, query consumptions
  + **statistics**: get aggregated statics around consumptions, contains most of the business logic, ConsumptionsAggregator interface has the aggregation logic responsibility
* core: aspects that are common to the use cases
  + abstract controller: contains cross cutting exception handling of controllers
  + domain entity
  + common DTOs across the use cases
  + validation
* repository: contains interface of the repository and implementation with Spring Data JPA; the implementation can be swapped with a new one providing another implementation of ConsumptionRepository interface

Notes:

* the application is structured in a layered way: controller -> service -> repository
* each resource (use case) has its own controller and service
* controllers handle DTOs only (no entities are passed outside the service layer)
* the application has JSON resource controllers which can be easily changed for a different interface such as XML
* both storage technology and repository implementation can be changed
* data validation is performed with javax bean validation
  + malformed JSON
  + non-valid consumption creations
  + invalid query parameters
* creation in batch mode is done flushing the persistence context every N entities