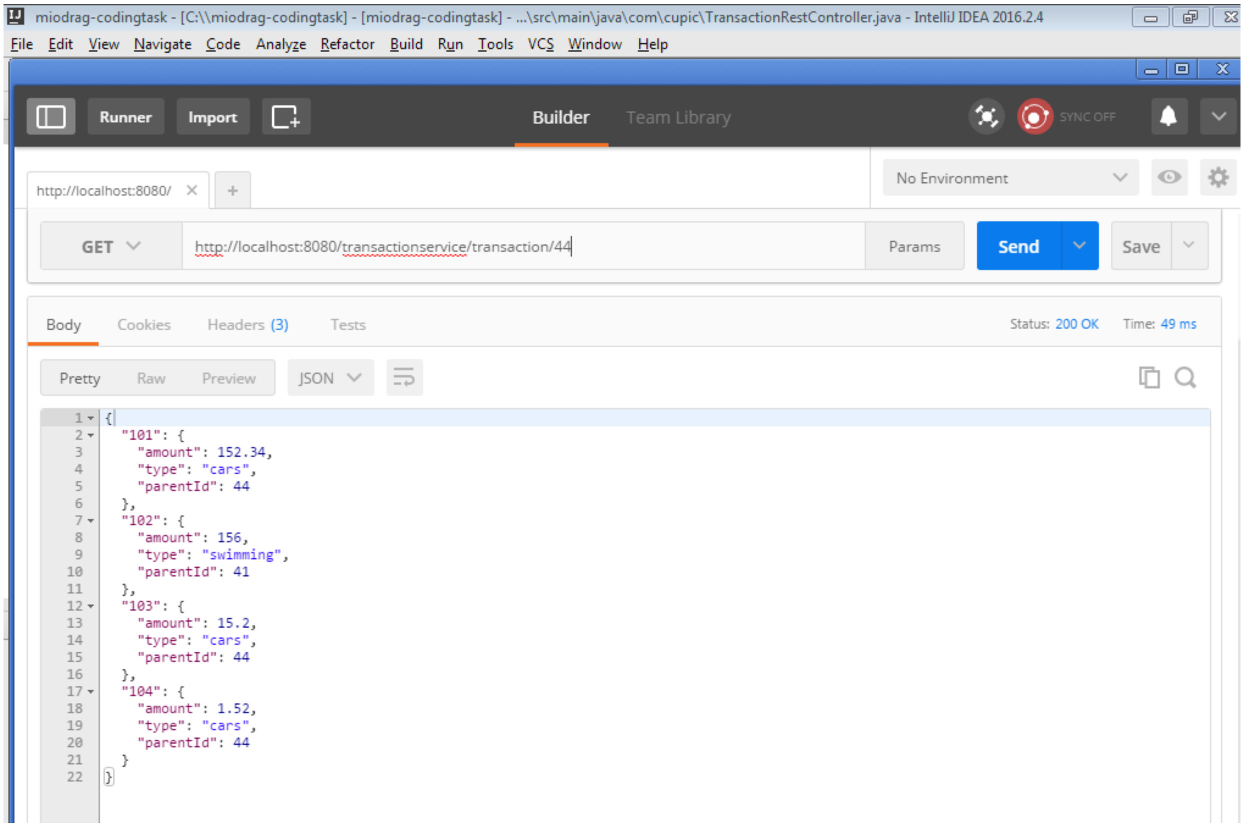
Miodrag Cupic - Coding Task

In this document I will briefly describe some of the actions I performed and some edge cases that could have been seen with the application.

The project was coded on Windows 7 using Java, with IntelliJ IDE. The project was built with Maven and Spring. It showed no problem when it was executed from the console nor from the IDE kit.

When booting from the console I was using the usual maven commands such as: mvn clean compile install, afterwards I would start the service with the following command: mvn spring-boot:run.

The app was tested using the POSTMAN plugin for Chrome:


miodrag-codingtask - [C:\\miodrag-codingtask] - [miodrag-codingtask] - ...\src\main\java\com\cupic\TransactionRestController.java - IntelliJ IDEA 2016.2.4
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Runner
Import
Builder
Team Library
SYNC OFF
http://localhost:8080/
No Environment
Params
Send
Save
Body
Cookies
Headers (3)
Tests
Pretty
Raw
Preview
JSON

GET http://localhost:8080/transactionservice/transaction/44

Status: 200 OK Time: 49 ms

```
{
  "101": {
    "amount": 152.34,
    "type": "cars",
    "parentId": 44
  },
  "102": {
    "amount": 156,
    "type": "swimming",
    "parentId": 41
  },
  "103": {
    "amount": 15.2,
    "type": "cars",
    "parentId": 44
  },
  "104": {
    "amount": 1.52,
    "type": "cars",
    "parentId": 44
  }
}
```

Picture 1 – Postman Plugin

The Application contains 4 Java classes: Transaction, TransactionUtility, TransactionRestController, Application.

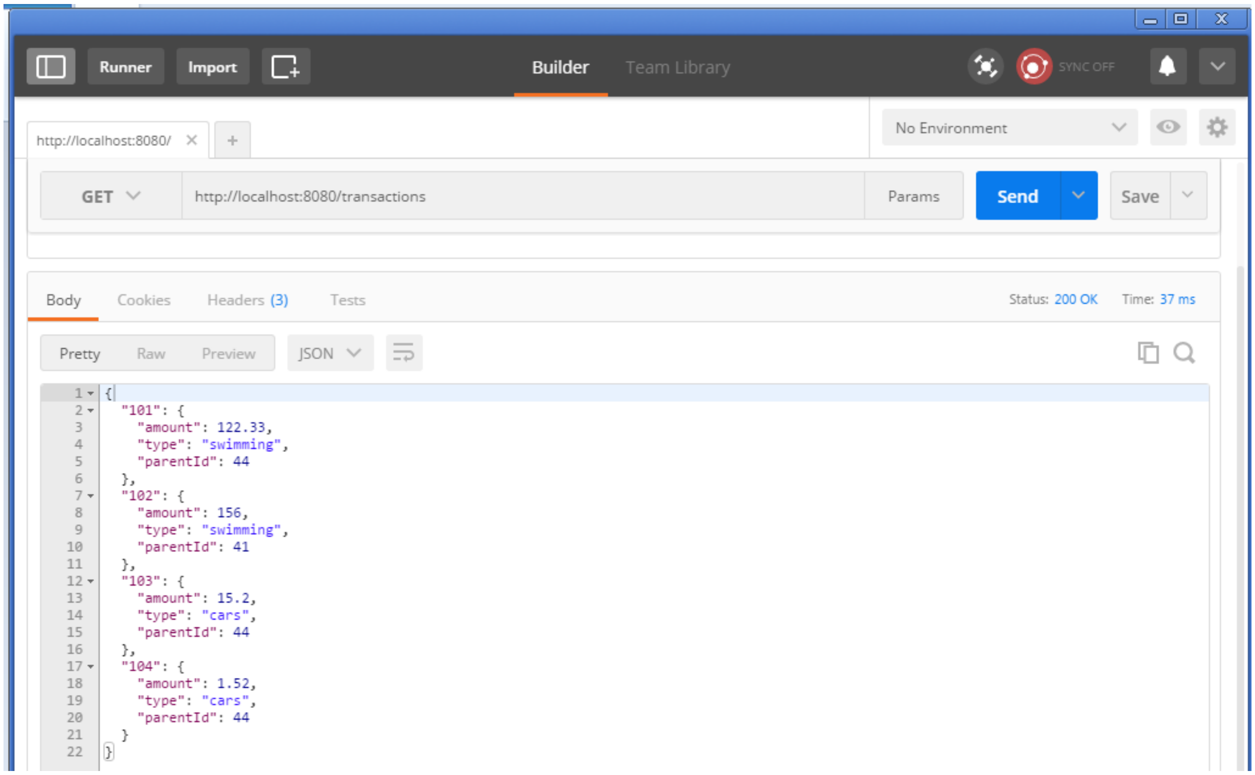
Transaction class is the class where we define the Object that stores the parameters of each transaction. The Transaction class has three attributes: parent_id, amount and type.

TransactionRestController is where the communication with the server happens. In order to make a simple service, I have used the Spring framework to help me construct GET and PUT petitions with corresponding responses. When the user runs the app and adds transactions on the server, each transaction is added to a HashMap, and it is stored there for future manipulation. The key of this HashMap is the transaction_id and the value is an instance of the Transaction class.

Transaction Utility class is a class where certain calculations and manipulation happen. It was constructed in order to move unnecessary code from TransactionRestController class.

Application is the main class that starts the service.

Border cases: One of the possible drawbacks in this application is that we can overwrite transactions. For example, let us say we have a transaction that has a transaction_Id -> 101



http://localhost:8080/

No Environment

GET http://localhost:8080/transactions

Status: 200 OK Time: 37 ms

```
{
  "101": {
    "amount": 122.33,
    "type": "swimming",
    "parentId": 44
  },
  "102": {
    "amount": 156,
    "type": "swimming",
    "parentId": 41
  },
  "103": {
    "amount": 15.2,
    "type": "cars",
    "parentId": 44
  },
  "104": {
    "amount": 1.52,
    "type": "cars",
    "parentId": 44
  }
}
```

Picture 2: List of transactions

If we were to make another request where we would state the same transaction_id: 101, we would end up overwriting the transaction and create a new one in its place, and loose the former. This could evaded if we were to keep track and compare if a given transaction_id is already located in the database and give it another identifier, or create the transaction_id as an autoincrement attribute.