# COMP3006L: Software Engineering II

Jerry

Team Member 1 -
R M J C Madawala
**17209393**

## Summary of Work Done

Implemented two separate APIs for internal communication and external communications.

Completed the Order generation service according to probability.

- Used UUID as a random unique ID for orders.
- Probability calculated using the random generation of numbers 1-100.
- Generated orders were wrapped into Objects with item Ids and frequency lists .

Completed the Order Management service.

- Built internal Communication with order generation service to get new orders.
- Built internal Communication with Worker Simulator Service to get available workers.
- Mapped Order UUIDs and worker Ids using HashMap.

Completed the Simulator service with endpoints.

- Worker, Item, Map Models ,and endpoints implemented.
- Communication with Worker Service implemented.

Partially completed Worker Service

Implemented intercommunications through RestTemplate.

Used MySQL database to store Workers and Items.

Used Java Persistence API (**JPA**).

## Order Processing Strategy

Order Generation service generates orders according to given probabilities

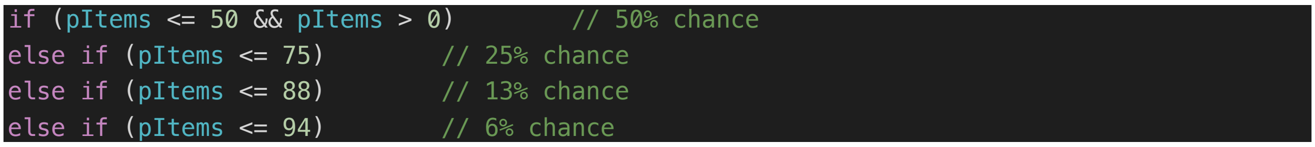
Probability --

- `Math.Random()` gives numbers 0 to 1.

```
    public int generateRandomInt0To100(){
        Double dn = Math.random()*100.0;
        return dn.intValue();
    }
```

- Generates random numbers 0 to 100 using `Math.Random()*100` function.
- There is a 25% chance to get 1 to 25 (25)numbers.
- 10% chance of getting 26 to 36 (10)numbers.

- Likewise we can generate any probability.
- According to that probability Orders and item Frequencies are generated.



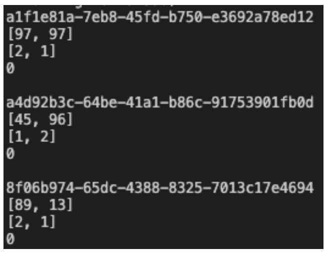
```
if (pItems <= 50 && pItems > 0)          // 50% chance
else if (pItems <= 75)        // 25% chance
else if (pItems <= 88)        // 13% chance
else if (pItems <= 94)        // 6% chance
```

tested stable implementations

https://github.com/janithongithub/random_order_generator_test
https://github.com/janithongithub/Probabilityorder

Generated Orders --

- Item frequencies and items are Wrapped in a class with unique UUID of 36 characters.
- Static list is used to keep the record of all created objects.
- The object reference and UUIDs are mapped using a hash map to pass through RestTemplate.
- For REST API communications Hash Map is wrapped with objects for convenience.



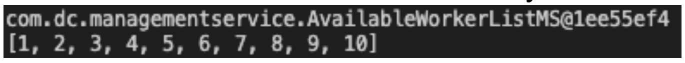
```
a1f1e81a-7eb8-45fd-b750-e3692a78ed12
[97, 97]
[2, 1]
0

a4d92b3c-64be-41a1-b86c-91753901fb0d
[45, 96]
[1, 2]
0

8f06b974-65dc-4388-8325-7013c17e4694
[89, 13]
[2, 1]
0
```

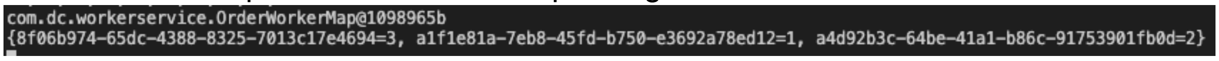
Management service

- Communicate through REST API with OrderGeneration service and Get Order UUID list.
- Communicate with Simulator Service to get available workers.
- Available workers are identified if they don't have holding items.



```
com.dc.managementservice.AvailableWorkerListMS@1ee55ef4
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Checks static Order UUID list and identify new orders and assign them to new workers.
- Create Order UUID and Worker ID hash map.
- OrderUUID, WorkerID Hash map is sent to Worker Brain through RestTemplate.
- One Worker completes the Order until packing.



```
com.dc.workerservice.OrderWorkerMap@1098965b
{8f06b974-65dc-4388-8325-7013c17e4694=3, a1f1e81a-7eb8-45fd-b750-e3692a78ed12=1, a4d92b3c-64be-41a1-b86c-91753901fb0d=2}
```

Worker Service

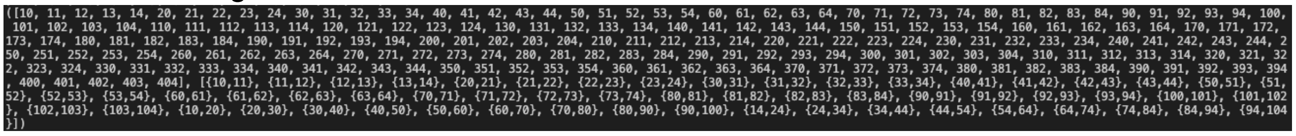
- Gets OrderUUID,WorkerID Hash map from management service through RestTemplate.
- Gets Worker Id, URI, location, capacity from Simulator service.
- Gets Item Id, location, weight from simulator service.
- Computes start and stop using the  Dijkstra algorithm.
- Sends path to Simulation Service using workers unique URI created with a name.
- Computes path to packing area using Dijkstra algorithm.
- Every last point in a row has a packing area.
- If weight is more than worker capacity drops items to packing area .
- Computes path from packing area to Item Location.

- Complete actions step by step.
- Finally set Order Status to COMPLETED in OrderGenerator service.

Simulator Service

- Sends all required models to worker service , Order generator  and Management Service through Rest Template.
- Converter is used to store map vertices.

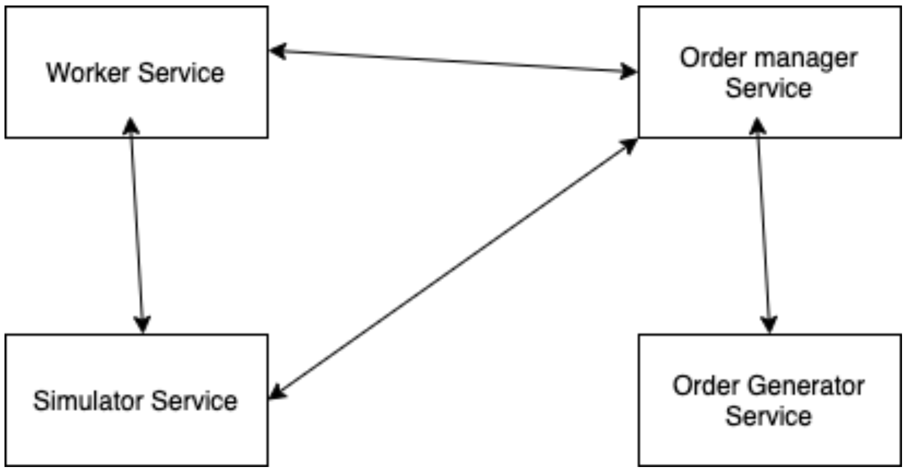
  Customizable Map – implemented using JGraphT
  vertices and edges



```
([10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44, 50, 51, 52, 53, 54, 60, 61, 62, 63, 64, 70, 71, 72, 73, 74, 80, 81, 82, 83, 84, 90, 91, 92, 93, 94, 100,
 101, 102, 103, 104, 110, 111, 112, 113, 114, 120, 121, 122, 123, 124, 130, 131, 132, 133, 134, 140, 141, 142, 143, 144, 150, 151, 152, 153, 154, 160, 161, 162, 163, 164, 170, 171, 172,
173, 174, 180, 181, 182, 183, 184, 190, 191, 192, 193, 194, 200, 201, 202, 203, 204, 210, 211, 212, 213, 214, 220, 221, 222, 223, 224, 230, 231, 232, 233, 234, 240, 241, 242, 243, 244, 2
50, 251, 252, 253, 254, 260, 261, 262, 263, 264, 270, 271, 272, 273, 274, 280, 281, 282, 283, 284, 290, 291, 292, 293, 294, 300, 301, 302, 303, 304, 310, 311, 312, 313, 314, 320, 321, 32
2, 323, 324, 330, 331, 332, 333, 334, 340, 341, 342, 343, 344, 350, 351, 352, 353, 354, 360, 361, 362, 363, 364, 370, 371, 372, 373, 374, 380, 381, 382, 383, 384, 390, 391, 392, 393, 394
, 400, 401, 402, 403, 404], [{10,11}, {11,12}, {12,13}, {13,14}, {20,21}, {21,22}, {22,23}, {23,24}, {30,31}, {31,32}, {32,33}, {33,34}, {40,41}, {41,42}, {42,43}, {43,44}, {50,51}, {51,
52}, {52,53}, {53,54}, {60,61}, {61,62}, {62,63}, {63,64}, {70,71}, {71,72}, {72,73}, {73,74}, {80,81}, {81,82}, {82,83}, {83,84}, {90,91}, {91,92}, {92,93}, {93,94}, {100,101}, {101,102
}, {102,103}, {103,104}, {10,20}, {20,30}, {30,40}, {40,50}, {50,60}, {60,70}, {70,80}, {80,90}, {90,100}, {14,24}, {24,34}, {34,44}, {44,54}, {54,64}, {64,74}, {74,84}, {84,94}, {94,104
}])
```

  tested stable implementations

  [https://github.com/janithongithub/MapFullFunctional](https://github.com/janithongithub/MapFullFunctional)



## Reflections

What I learnt

- Microservices architecture.

- Rest communication using Rest Template and specific methods.
  getForObject()
  getForEntity()

postForObject()
public getters and setters are a must for Rest communications as object fields are initialized using setters.
When passing objects RestTemplate uses getters to fetch data.

- Using spring boot annotations effectively.
  @Autowired
  @Component
  @RestController
  @ResquestMapping
  .
  .
  .

- Finding the shortest path.
- Using JGraphT to generate vertices and edges and to find shortest the path.
- Functional Programming.
- Streams API.
- JPA CRUD repository.
- Using Data Structures.
- Testing independent Services using Junit.
- Automation.

Challenges

- Generating probabilities and generating Order according to probability.
  Used nested conditions and Math.Random() function to overcome.
- Communications within microservices using Rest Template and Web Client.
  Empty constructor getters and setters are always needed to communicate Rest Template.
  Object conversion issues when highly coupled.
- Using Akka HTTP.
- Process automation.
- Using Docker.

## Content to be Submitted

| | |
|---|---|
| ***README*** | https://gitlab.com/comp3006l/2020/jerry/dcapp/-/blob/master/README.txt |
| ***WALKTHROUGH VIDEO*** | *https://youtu.be/9C-wn3E1f6I* |
| ***REPORT*** | https://gitlab.com/comp3006l/2020/jerry/dcapp/-/tree/master |
| ***CODE*** | https://gitlab.com/comp3006l/2020/jerry/dcapp |