INTERDISCIPLINARY PROJECT (IDP) 2016

BAGGAGE HANDLING SYSTEM

(March,2016 – July,2016)

Akash Manjunath, ga42cej@mytum.de
Prateek Bagrecha, ga63geq@mytum.de
Advisor: Mr. Christian Ruf

Lehrstuhl für Operations Management
Technische Universität München

ABSTRACT

This interdisciplinary project (IDP) was realized in collaboration with Mr. Christian Ruf belonging to the department of "Lehrstuhl für Operations Management". This documentation is intended to serve both as a reference for the understanding and usage of the 'Baggage Handling System' as well as for future enhancements of the same.

The system essentially consists of a series of Optimizations and Simulations which effectively regulate the load (number of bags) on the carousels of airport terminals. The primary objective of the project was to develop a web application which would effectively present the results from the 'Baggage Handling System' in an intuitive manner to the observers. The challenge was to develop a Graphical User Interface to present the complex variations of load on the carousels in a simplistic manner.

This documentation consists of a basic review of the literature and research behind the idea of the System in question and the events related to it. Additionally, it consists of the Methodologies and techniques incorporated to achieve the necessary results. The methodologies also act a reference in order to for future development. Finally, it consists of the results which were achieved during the course of the development period.

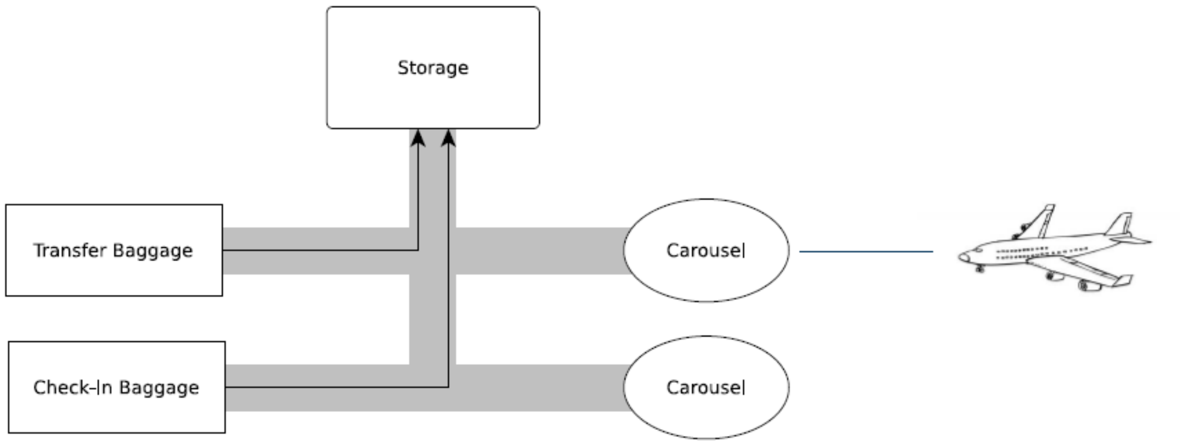
TABLE OF CONTENTS

REVIEW OF LITERATURE AND RESEARCH

1.1 Overview of Baggage Handling System(BHS)

Outbound baggage is the baggage that is brought in from check-in passengers (check-in baggage) or incoming flights (transfer baggage) and which has to be loaded into departing airplanes. The check-in baggage and transfer baggage are fed in into the Baggage Handling System (BHS) which is a conveyor belt network automatically transporting the bags to their destinations. To load the bags into bulk-containers, all departing flights are scheduled and assigned to handling facilities, called carousels. During a flight's baggage handling period, workers at the carousels load incoming bags from the carousel's conveyor belt into bulk-containers. At the end of the baggage handling period, usually 10 to 15 minutes before the flight's departure, all containers are transported to the corresponding airplane, where they are loaded into the airplane's cargo hold.

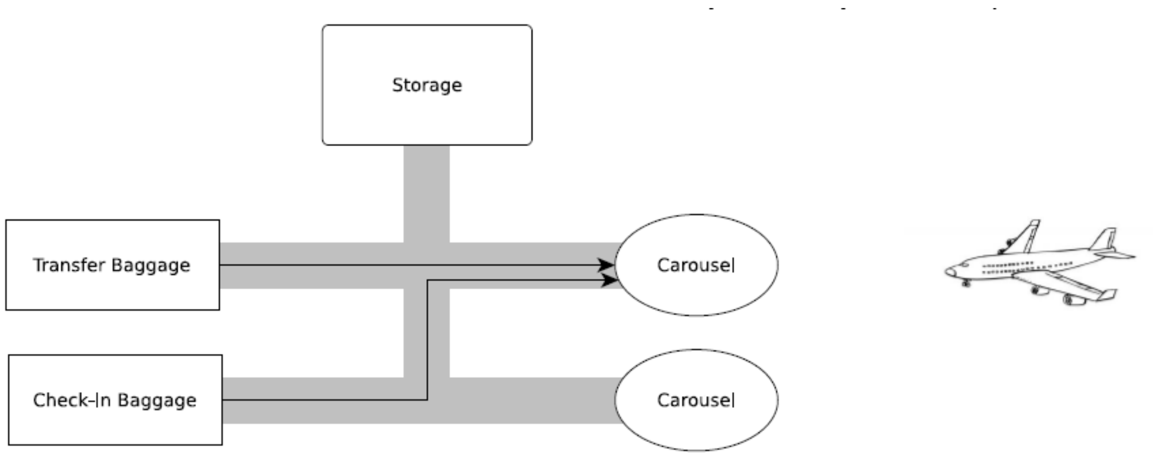
At initial stages, each flight is assigned to a carousel and a number of its workstations. all check-in baggage and transfer baggage are rerouted to a central storage.



(**Fig 1.1.1** Overview of BHS: Initial Stage)

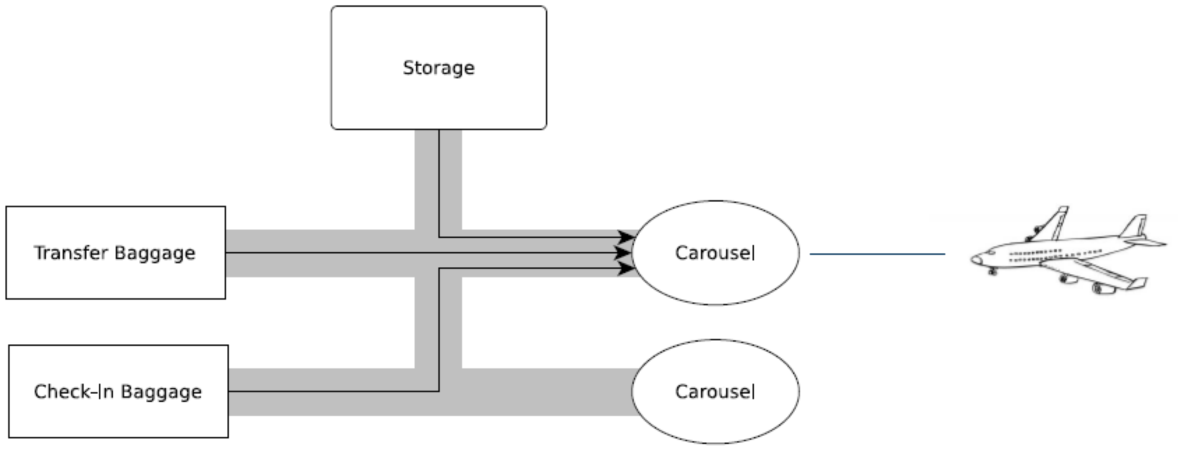
1.2 Overview of Events in BHS

For each flight, start of baggage handling, henceforth called SH (Start Handling), and start of storage depletion, henceforth called SD (Storage Depletion), have to scheduled. Baggage handling has to finish 10-15 minutes before the flight's scheduled take off and is marked as EH (End Handling).

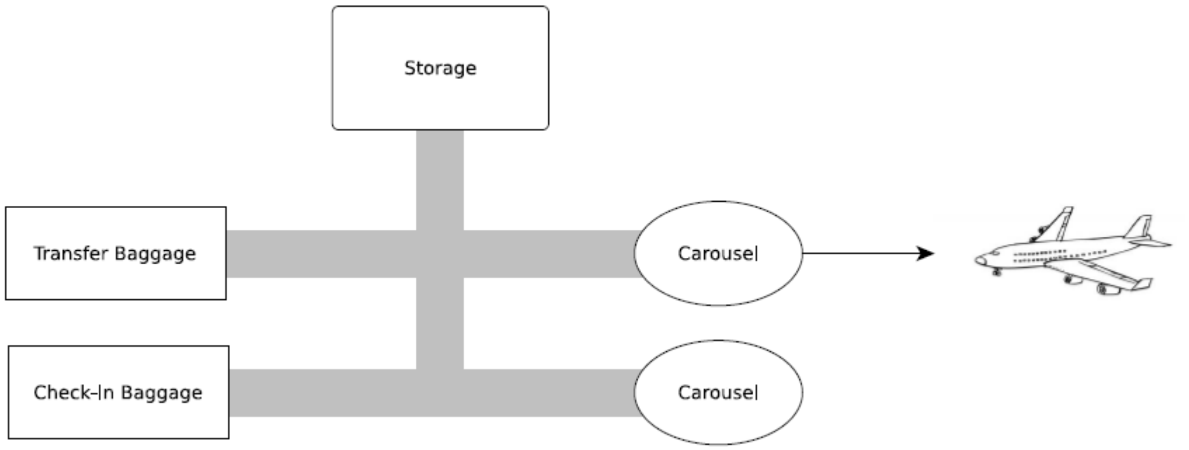


(**Fig 1.2.1**: Overview of BHS: scheduling SH, SD and SE)

The SD can be scheduled between SH and EH so that all bags for that particular flights are loaded. The arrival of a baggage on carousel or baggage being removed from the carousel will henceforth be called BA (Baggage Arrival).



(**Fig 1.2.2** Overview of BHS: Baggage Flow since SD start)



(**Fig 1.2.2** Overview of BHS: Baggage Flow at EH)

Apart from these events, the BHS also has worker start to handle(WH) event and this occurs in between SH and SD.

REQUIREMENTS AND FEASIBILITY STUDY

2.1 MongoDB for Storing and Retrieving Events

2.1.1 What is MongoDB?

MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through the MongoDB query language. MongoDB uses dynamic schemas, meaning that you can create records without first defining the structure, such as the fields or the types of their values. You can change the structure of records (which we call documents) simply by adding new fields or deleting existing ones. This data model gives the ability to represent hierarchical relationships, to store arrays, and other more complex structures easily. Documents in a collection need not have an identical set of fields and de-normalization of data is common. MongoDB was also designed with high availability and scalability in mind, and includes out-of-the-box replication and auto-sharding.

2.1.2 Terminology and Concepts

Many concepts in MySQL have close analogs in MongoDB. This table outlines some of the common concepts in each system.

MySQL	**MongoDB**
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

(**Table 1.1.1**: Comparison of Terms b/w MySQL and MongoDB)

2.1.3 Why use MongoDB instead of MySQL?

Development is simplified:
MongoDB documents map naturally to modern, object-oriented programming languages. Using MongoDB removes the complex object-relational mapping (ORM) layer that translates objects in code to relational tables.

Flexible Data Model
MongoDB's flexible data model also means that your database schema can evolve with business requirements.

Scalability
MongoDB can also be scaled within and across multiple distributed data centers, providing new levels of availability and scalability previously unachievable with relational databases like MySQL. As your deployments grow in terms of data volume and throughput, MongoDB scales easily with no downtime, and without changing your application. In contrast, to achieve scale with MySQL often requires significant, custom engineering work.

2.1.4 Use of Mongo DB in the Project

1. Simulation events are stored in mongo as documents (JSON format) in MongoDB leading to fast storage access and retrieval: Avoiding latency
2. Event structure (i.e. parameters of simulation) can be dynamically set according to type of event
3. Since the events are retrieved as JSON strings which can be easily parse for UI table and chart development
4. AJAX handling are simplified as web components can be separately refreshed for reflecting latest events.
5. Historical events can be maintained and tracked easily.

2.2 Backend Application - PLAY Framework

2.2.1 What is PLAY Framework?

PLAY is a Framework which is used for developing Java/Scala based application in an extremely easy manner. The "Baggage Handling System" is built using PLAY 2 which is the version of PLAY available in the market.

2.2.2 Reasons for the use of PLAY Framework

It is built for asynchronous programming
The "Baggage Handling System" deals with concurrent real-time data. Hence, the need for a framework which supports a full asynchronous HTTP programming model was high. Since PLAY provides such a capability, it was one of the primary factors behind its usage.

Native support for Java
The Optimization and Simulation code was previously written using Java. In order to ensure compatibility, PLAY was chosen as it provides inherent support for Java.

Powerful build system
PLAY provides better integration with Maven projects out of the box and the ability to package and publish projects as a simple set of JAR files to any repository. Live compiling and reloading at development time of any depended project for standard Java library projects is provided.

Datastore and Model Integration
With the emergence of various different sort of databases, PLAY offers full support for application based on different types of databases and not just RDBMS. Considering that we are using MongoDB as our database and not any of the traditional databases like MySQL, Oracle etc. the use of PLAY Framework is justified.

2.3 Frontend Application - AngularJS with Material Design, HighCharts for Graphs

2.3.1 AngularJS

It provides structure to JavaScript
JavaScript framework like AngularJS provides a more structured way of building complete JavaScript applications as compared to previous ways of development using jQuery and others JavaScript libraries.

Two-way data-binding
AngularJS provides a more declarative way of connecting models to your view. The views and models are interconnected and a change in the view reflects in the model and vice versa.

Great for SPA (Single-Page Applications)
Considering the application consists of multiple components which have to be displayed on a single page, AngularJS provides the perfect solution to develop each of the components in an independent manner and finally aggregate all the components and display them on a single page.

Modular development
AngularJS provides an easier way to load modular pieces of code and views dynamically into the application. It also makes it easier to understand the code and to maintain it.

2.3.2 Material Design
The application makes use of Angular Material which implements Material Design components as AngularJS directives and services making development of GUI fast and easy to understand.

2.3.3 HighCharts
The usage of charts for visualization of the data forms the most important aspect of the front end application. For this purpose, multiple solutions were tries. Amongst the alternatives, angular Chart.js, nvd3 and HighCharts were tried. HighCharts was used in the final version due to the following aspects:

Chart.js
Advantages
1. Graphically appealing
2. Easy assignment of data to the charts

Disadvantages
1. No support for stacked charts
2. No export feature

nvd3
Advantages
1. Provides support for stacked charts

Disadvantages
1. Complex Data handling
2. Not very aesthetic visually
3. No export feature

HighCharts
Advantages
1. Visually aesthetic
2. Easy assignment of data
3. Support for stacked charts
4. Provides facility to export data

METHODOLOGY

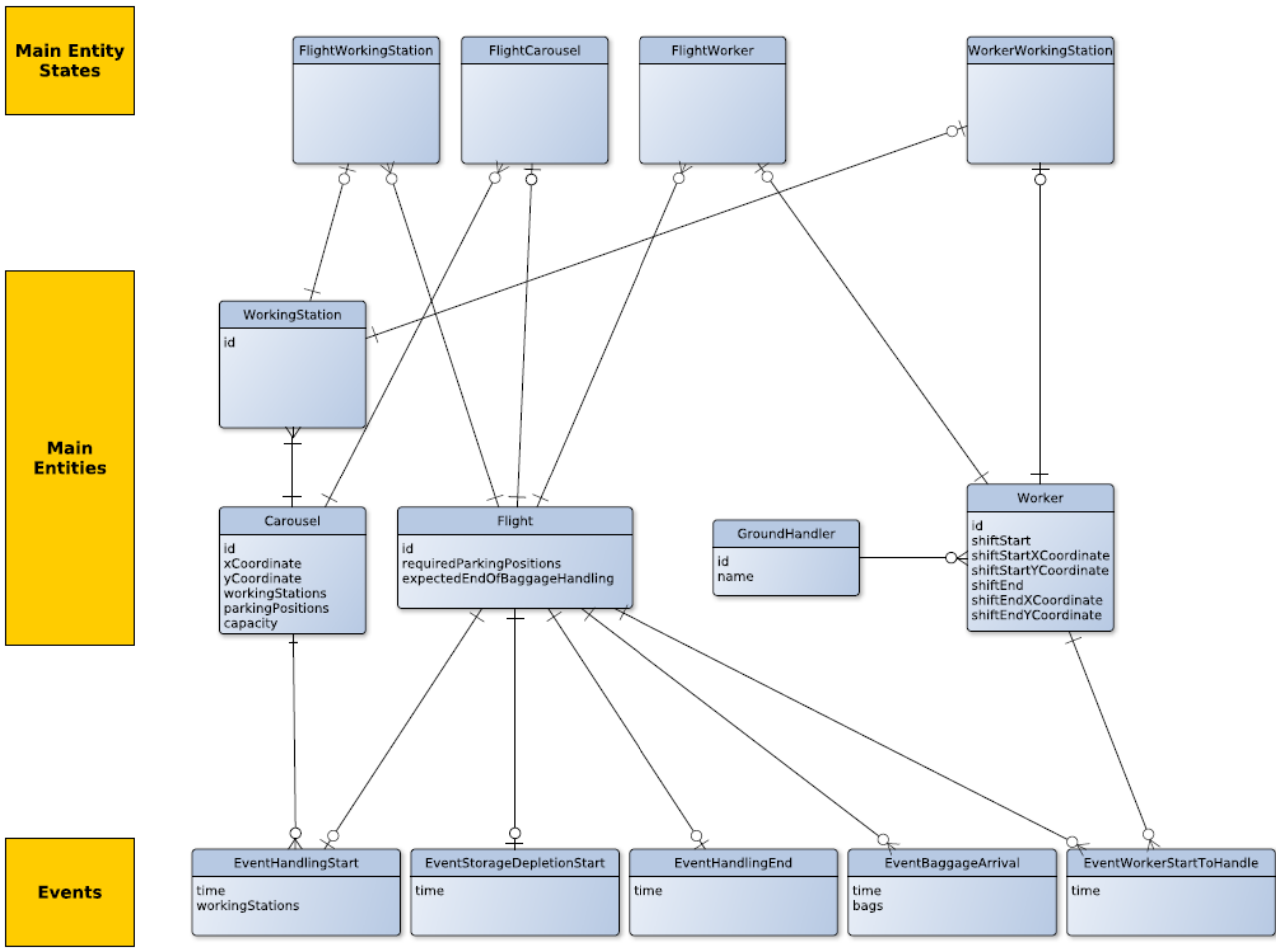
The project was developed using Agile Methodology with iterative development. The life cycle of the project development was divided into the following phases:

3.1 Design Phase

The Design Phase formed the primary phase for the project implementation.

Firstly, it consisted of designing the mockups which would act as templates for the development of the GUI. For this purpose, used the popular mockup tool Balsamiq. An iterative approach was followed in order to review the mockups with the guide in charge so that all the necessary features would be incorporated. The results from this phase can be seen in the Results section of the document under the section 3.1.

Secondly, the Entity-Relationship Model for handling the various events related to the "Baggage Handling Systems" was designed. The ER Model is as shown below:



(Fig. 3.1)

Finally, for the view point of separation of concerns, speed of development and ease of code maintainability, the overall project was split into multiple sub-projects for implementation during the Development phases.

3.2 Development Phase

Following the Design Phase, the Development Phase involved the development of a series of prototypes based on different technology stacks in order to come up with a solution which would be most suitable for the use cases involved. From the results of the Design Phase, separate GitHub repositories were created to handle the three different Use Cases that were present.

The following section gives a detailed description of each project along with the technology stack that was used for its implementation.

3.2.1 Overview of Backend Project

GitHub URL
https://github.com/Akash-M/idp-backend

Description
The scope of this project is to retrieve the data from the MongoDB and convert the unstructured data in to structured format such that it can be used by the frontend project. The structured data is presented in the form of JSON objects which is then consumed by the front end application to construct the graphical views.

Technology Stack
PLAY 2 Framework using Spring

REST API Documentation
Representational State Transfer (**REST**) is a software architectural style for distributed hypermedia systems like world wide web. It has been used to guide the design and development of the architecture for the modern web.
Play implements REST using a built-in HTTP router that translates each incoming HTTP requests to an action call. An HTTP request is seen as an event by the MVC framework. This event contains two major pieces of information:

- the request path (such as /clients/1542, /photos/list), including the query string.
- the HTTP method (GET, POST, …).

Routes are defined in the **conf/routes** file, which is compiled. This file lists all of the routes needed by the application. Each route consists of an HTTP method and URI pattern associated with a call to an action method. Each route starts with the HTTP method, followed by the URI pattern. The last element of a route is the call definition. The HTTP method can be any of the valid methods supported by HTTP (GET, PATCH, POST, PUT, DELETE, HEAD, OPTIONS).

Ex. ***GET /clients/:id controllers.Clients.show(id: Long)***

Following are the routes that are defined and available in this projects. Each of this returns JSON data that is processed by the frontend components and displayed as required

Resource URL
/api/v1/getCarouselDetailsById/:id
Type
GET
Parameter
ID of the carousel
Response

Code	Description	Schema

200	Details of a particular Carousel	{
		    _id : *Number*,
		    xCoordinate: *Number*,
		    yCoordiante: *Number*,
		    parkingPostions: *Number*,
		    maxCapacity: *Number*,
		    currentCapacity: *Number*,
		    Flight_id: [*Number*],
		    No_of_Workers: *Number,*
		    Ground_Handler_Name: *String*
		}

Example usage
/api/v1/getCarouselDetailsById/16
Example response
```
{
   "_id" : 16,
   "xCoordinate" : 4,
   "yCoordinate" : 1,
   "parkingPositions" : 1,
   "maxCapacity" : 40,
   "currentCapacity" : 5,
   "flight_id" : [
      14, 21
   ],
   "No_of_Workers" : 1,
    "Ground_Handler_Name" : "AeroHandler"
}
```

Resource URL
/api/v1/getCarouselsList
Type
GET
Parameter
None
Response

Code	Description	Schema

200	Details of all carousels with status	[
		    {
		        _id : *Number*,
		        xCoordinate: *Number*,
		        yCoordiante: *Number*,
		        parkingPostions: *Number*,
		        maxCapacity: *Number*,
		        currentCapacity: *Number*,
		        Flight_id: [*Number*],
		        carouselStatus: *String*

		}]

Example usage

/api/v1/getCarouselsList

Example response

```
[
{
   "_id" : 16,
   "xCoordinate" : 4,
   "yCoordinate" : 1,
   "parkingPositions" : 1,
   "maxCapacity" : 40,
   "currentCapacity" : 5,
   "flight_id" : [
      14, 21
   ],
  "carouselStatus" : "OK"
  },
{
   "_id" : 14,
   "xCoordinate" : ,
   "yCoordinate" : 41,
   "parkingPositions" : 31,
   "maxCapacity" : 40,
   "currentCapacity" : 15,
   "flight_id" : [
      19, 212
   ],
  "carouselStatus" : "OK"
  }
]
```

Resource URL

/api/v1/getFlightsDetailsById/:id

Type

GET

Parameter

ID of the Flight

Response

Code	**Description**	**Schema**
200	Gets All Details of a flight using its ID	{ _id : *Number*, requiredNumberOfParkingPositions : *Number*, expectedEndofBaggageHandling: *Numbers*, carousel_id: *Number*,

		worker_id: [*Number*], workingstations: [*Number*] }

Example usage

/api/v1/getFlightsDetailsById/

Example response

```
{
   "_id" : 14,
   "requiredNumberOfParkingStations" : 1,
   "expectedEndOfBaggageHandling" : 35,
   "carousel_id" : 16,
   "worker_id" : [
      44
   ],
   "workstations" : [ 12, 13]
}
```

Resource URL

/api/v1/getAllCarouselsId

Type

GET

Parameter

None

Response

Code	**Description**	**Schema**

200	Gets All Carousels IDs	{
		Carousel_ids: [*Numbers*]
		}

Example usage

/api/v1/getAllCarouselsId/

Example response

```
{
"Carousel_ids" : [ 1, 2, 6, 9 ,16, 14]
}
```

Resource URL

/api/v1/getAllCarouselEvents/:id

Type

GET

Parameter

ID of the carousel

Response

Code	**Description**	**Schema**

200	Gets All Events Related to a Carousel	{ carousel_id: *Number*, Flights: [flight_id: *Number*, Events: [{ time: *Number*, name: *String*, bags: *Number* }]] }

Example usage

/api/v1/getAllCarouselEvents/

Example response

```
{
  "carousel_id":14,
  "Flights":[
    {
      "flight_id":2,
      "Events":[
        {
          "time":17,
          "name":"SH"
        },
        {
          "time":17,
          "name":"WH"
        },
        {
          "time":17,
          "name":"SD"
        },
        {
          "time":22,
          "name":"BA",
          "bags":4
        },
        {
          "time":23,
          "name":"BA",
          "bags":3
        },
        {
          "time":28,
          "name":"BA",
          "bags":-5
        }
      ]
    },
    {
      "flight_id":11,
      "Events":[
        {
          "time":21,
          "name":"SH"
        },
        {
          "time":21,
          "name":"SD"
        },
        {
          "time":22,
          "name":"BA",
          "bags":6
        }
      ]
    }
  ]
}
```

3.2.2 Overview of Frontend Project

GitHub URL
https://github.com/Akash-M/idp-frontend

Description
The front end project consists of the Graphical User Interface which presents the data from all the sub-projects in a manner which is simple and easy to understand for the viewers. One cannot judge the performance changes that take place from Optimization and Simulation Phase by simply looking at the log files. The front end project consumes the data provided by the backend project in the form of REST APIs in order to graphically display the change in the load on the carousels to the eyes of naïve end users.

The front end application is built using the AngularJS framework in order to divide the Javascript code into MVC components for simplicity and ease of code maintenance. It consists of two major views with each view consisting multiple components.

The first view consists of a table which represents the status of all the carousels at any given point of time. The second view consists of the details of the carousels such as flights and work stations assigned. In addition to the raw data, it also consists of a graph which shows the load on the carousel versus time stamps which reflects the effectiveness of the Optimization and Simulation processed.

Technology Stack
AngularJS using Material Design, Bootstrap and Highcharts.

3.2.3 Overview of the Event Scripts Project

GitHub URL
https://github.com/Akash-M/idp-eventscripts

Description
As mentioned before, the current BHS experiment has two parts: Optimization and Simulation. BHS experiment is run in iterations. Each iteration has multiple runs of optimization in discreet time period, with intervals set before iteration, based on current status of BHS. The results of optimization in BHS are set of logs with indicated time periods of run and iteration. Logs represent the simulation phase of the BHS experiment.

For the purpose creating a new simulation system of BHS, a stochastic discreet-event model (Appendix) based on the simulation logs was created. Logs that match the events mentioned in the previous section were extracted in order to create input values. The output of model is a set of events for the given iteration stored in a database. Model perfectly represented the current simulation system however it was made more powerful by storing the events in the database rather in log files. This makes output of the model much more readable, helping in improving the speed of output analysis of the experiment.

The model itself can be integrated in the current BHS system and used as tool to insert events into the database for output analysis. The model is written in JAVA as maven project and can be imported into other projects as jar or as maven dependency.

Technology Stack
Maven project using Java and MongoDB

Overview of the Database

GitHub URL
https://github.com/Akash-M/idp-db

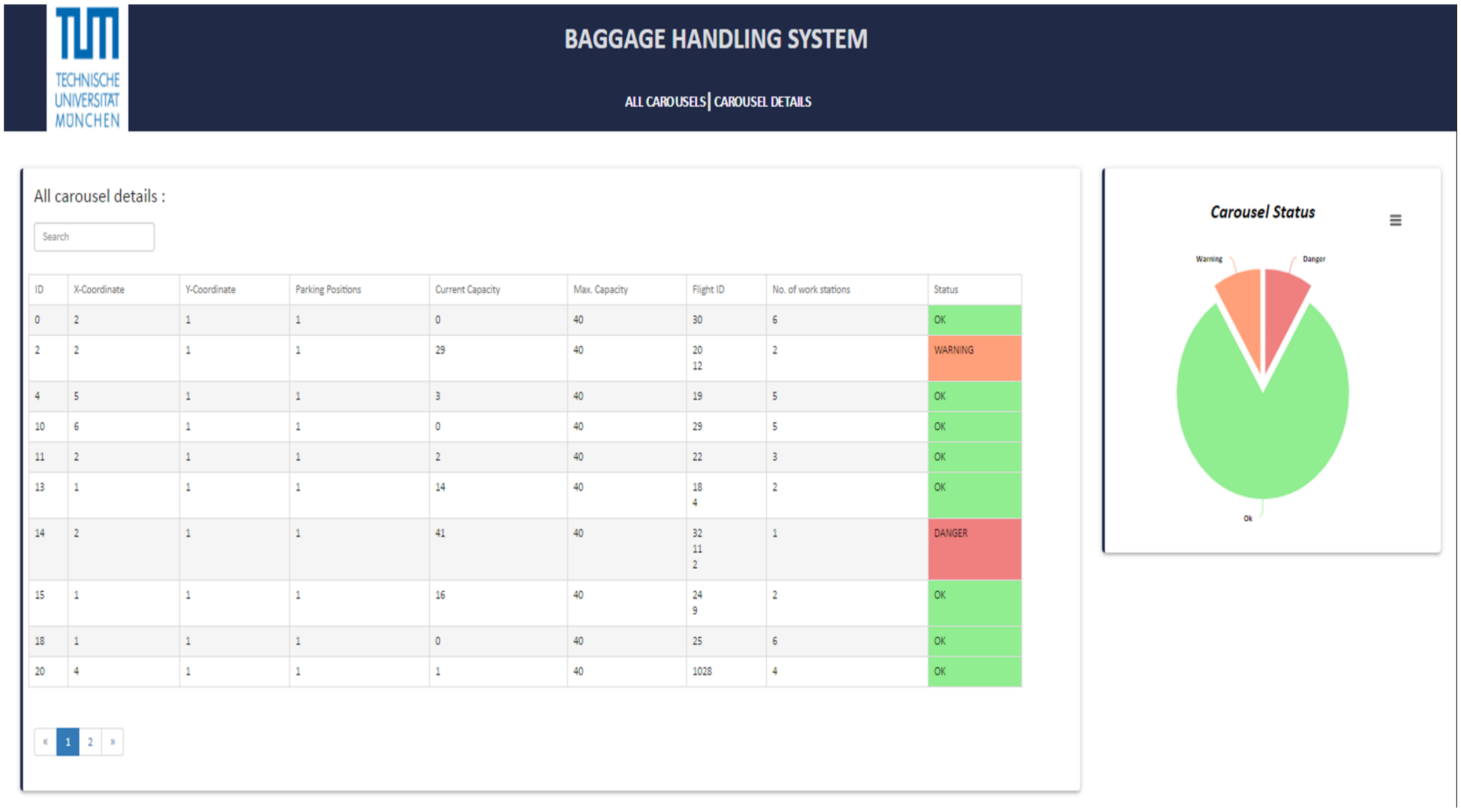
Description
This projects consists of all the events related data resulting from the Optimization and Simulation in an unstructured format.

The data is the foundation based on which information is extracted using the backend project and fed to the frontend project for graphical representation.

Technology Stack
Mongo Database

RESULTS

4.1 All Carousel View


TECHNISCHE UNIVERSITÄT MÜNCHEN
BAGGAGE HANDLING SYSTEM
ALL CAROUSELS | CAROUSEL DETAILS
Carousel Status
Warning
Danger
Ok
« 1 2 »

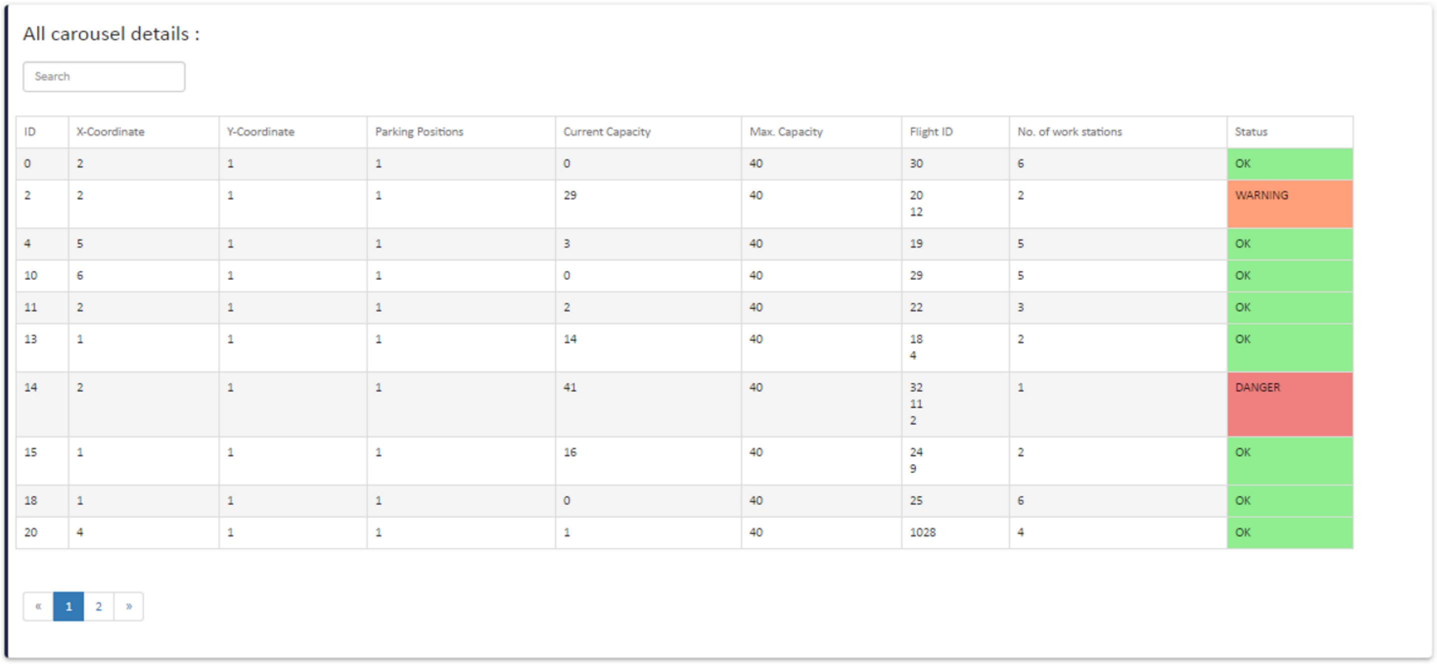
All carousel details :

Search

ID	X-Coordinate	Y-Coordinate	Parking Positions	Current Capacity	Max. Capacity	Flight ID	No. of work stations	Status
0	2	1	1	0	40	30	6	OK
2	2	1	1	29	40	20 12	2	WARNING
4	5	1	1	3	40	19	5	OK
10	6	1	1	0	40	29	5	OK
11	2	1	1	2	40	22	3	OK
13	1	1	1	14	40	18 4	2	OK
14	2	1	1	41	40	32 11 2	1	DANGER
15	1	1	1	16	40	24 9	2	OK
18	1	1	1	0	40	25	6	OK
20	4	1	1	1	40	1028	4	OK

This view provides the user with on overall overview of all the carousels in the systems. It consists of two components:

4.1.1. Table consisting of details of all the carousels

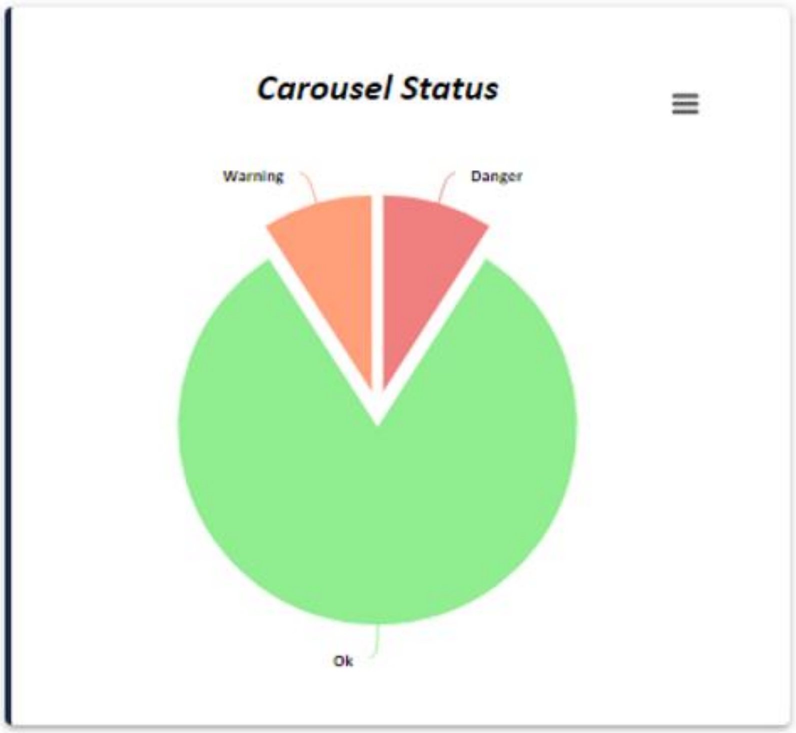


All carousel details :

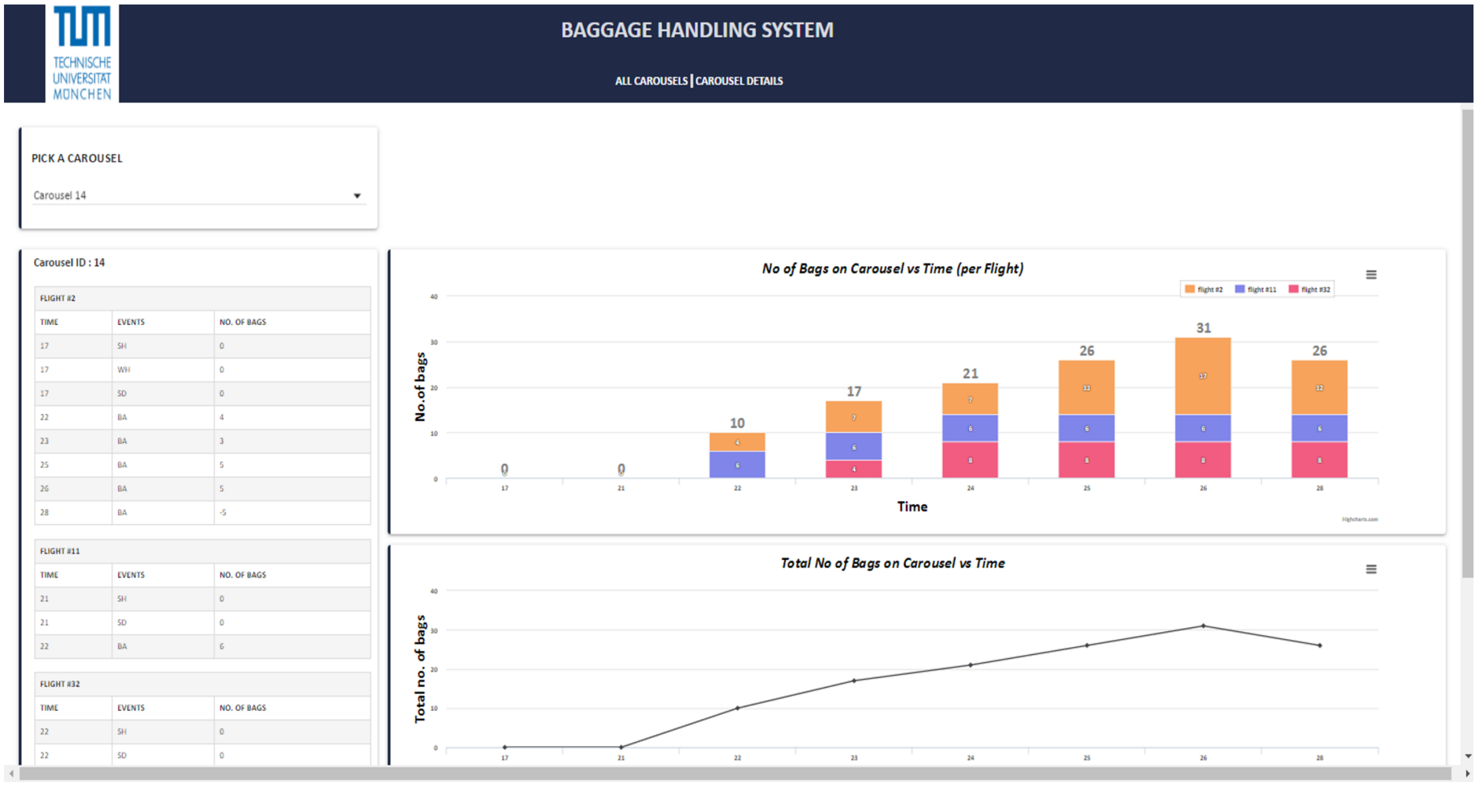
Search

ID	X-Coordinate	Y-Coordinate	Parking Positions	Current Capacity	Max. Capacity	Flight ID	No. of work stations	Status
0	2	1	1	0	40	30	6	OK
2	2	1	1	29	40	20 12	2	WARNING
4	5	1	1	3	40	19	5	OK
10	6	1	1	0	40	29	5	OK
11	2	1	1	2	40	22	3	OK
13	1	1	1	14	40	18 4	2	OK
14	2	1	1	41	40	32 11 2	1	DANGER
15	1	1	1	16	40	24 9	2	OK
18	1	1	1	0	40	25	6	OK
20	4	1	1	1	40	1028	4	OK

4.1.2 Pie chart representing the number of carousels in different states



4.2 Carousel Detail Page


BAGGAGE HANDLING SYSTEM
ALL CAROUSELS | CAROUSEL DETAILS
PICK A CAROUSEL
Carousel 14
No of Bags on Carousel vs Time (per Flight)
flight #2
flight #11
flight #32
No.of bags
Time
Total No of Bags on Carousel vs Time
Total no. of bags

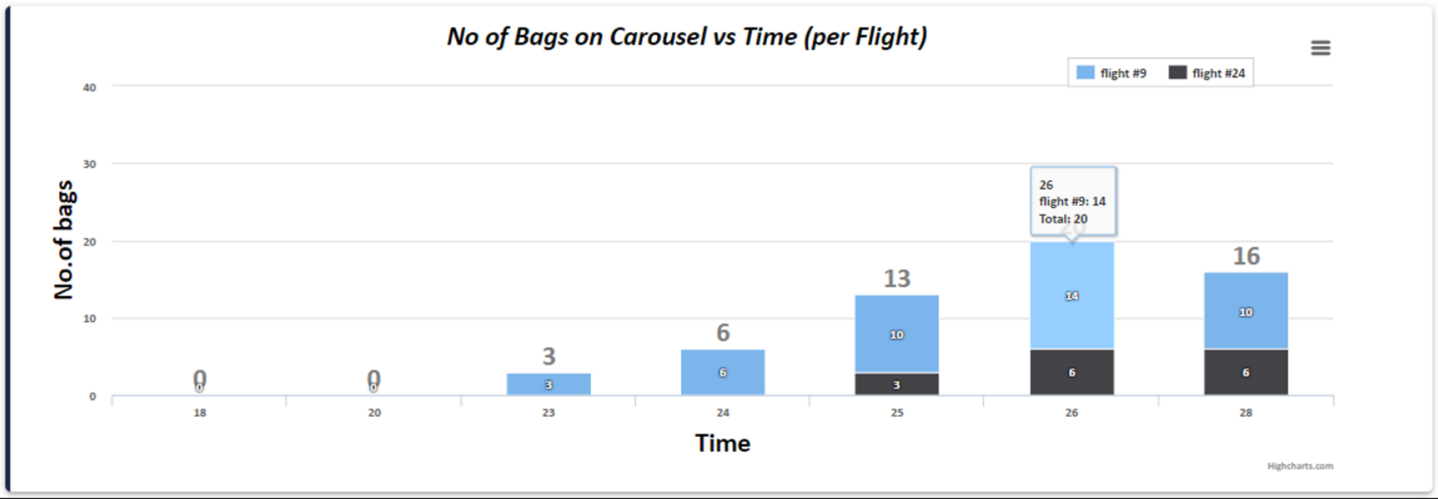
Carousel ID : 14

FLIGHT #2		
TIME	EVENTS	NO. OF BAGS
17	SH	0
17	WH	0
17	SD	0
22	BA	4
23	BA	3
25	BA	5
26	BA	5
28	BA	-5

FLIGHT #11		
TIME	EVENTS	NO. OF BAGS
21	SH	0
21	SD	0
22	BA	6

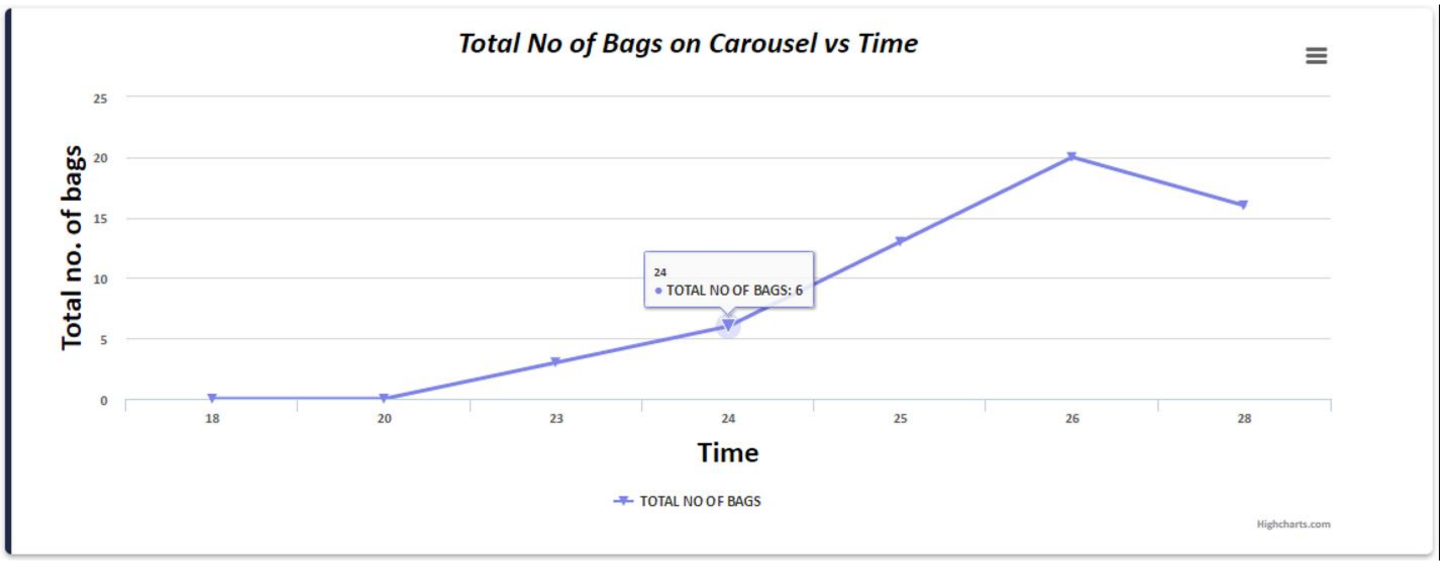
FLIGHT #32		
TIME	EVENTS	NO. OF BAGS
22	SH	0
22	SD	0

4.2.1 Graph of Number of Bags on the Carousel versus Time



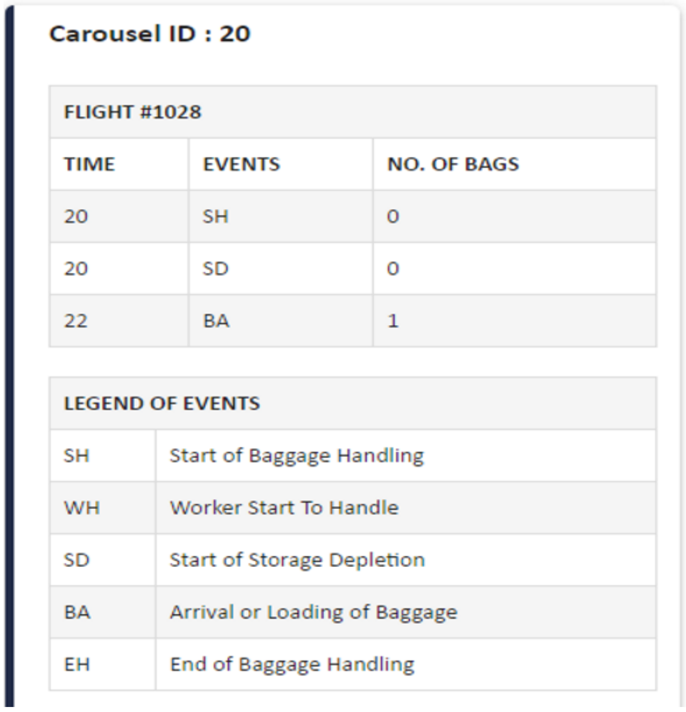
Graph is a stacked bar chart which shows the number of bags assigned per flight at a particular time period.

4.2.2 Graph of Total Number of Bags on the carousel versus Time



This graph is a line chart which shows the total number of bags at any point on the carousel. One can infer how the load on the carousel varies over time using this graph.

4.2.3 Tables of Flight Details



Carousel ID : 20

FLIGHT #1028		
TIME	EVENTS	NO. OF BAGS
20	SH	0
20	SD	0
22	BA	1

LEGEND OF EVENTS	
SH	Start of Baggage Handling
WH	Worker Start To Handle
SD	Start of Storage Depletion
BA	Arrival or Loading of Baggage
EH	End of Baggage Handling

This table shows all the details related to each flight assigned to the selected carousel. The details include the time stamp, event names and number of bags affected by that event at the time stamp.

REFERENCE LIST

1. Disruption Management for Outbound Baggage Handling with Work Group Pairings by Christian Ruf, Markus Frey, Rainer Kolisch, TUM School of Management, Technische Universität München
2. MongoDB Architecture Guide, June 2016 by MongoDB
3. Top 5 Considerations When Evaluating NoSQL Databases, June 2016 by MongoDB
4. Storing Log Data in MongoDB: https://docs.mongodb.com/ecosystem/use-cases/storing-log-data/
5. MongoDB Driver for Java: http://mongodb.github.io/mongo-java-driver/3.2/
6. Play for Java Developers: https://www.playframework.com/documentation/2.5.x/JavaHome
7. Angular Material: https://material.angularjs.org/latest/
8. Balsamiq: https://balsamiq.com