

***Public Transport System Management***

**IS 301 Enterprise Integration**

***G2-3***

**Assignment**

***EUGENE CHOY WEN JIA***

***HO MIN KIT WINSTON***

***HO WEI HONG***

***SIM LI JIN***

***YIN YUKUN***

***YONG FU XIANG***

# Introduction

The Public Transport network in Singapore serves hundreds of thousand passengers every day. Any disruption in the service could be disastrous and mitigation actions must be undertaken to ensure minimal disruptions to the passengers. Integrations of various system in the Public Transport Network is therefore important to help enhance communication and automate processes if mitigation actions are required. On top of having to mitigate disruptions, there are also other processes such as checking the weather to determine the speed of travel and the bus schedules of bus drives which requires several steps before the message can be transferred from one end to the other. Using integration tools, we can automate a huge bulk of this process reducing the time needed for human intervention.

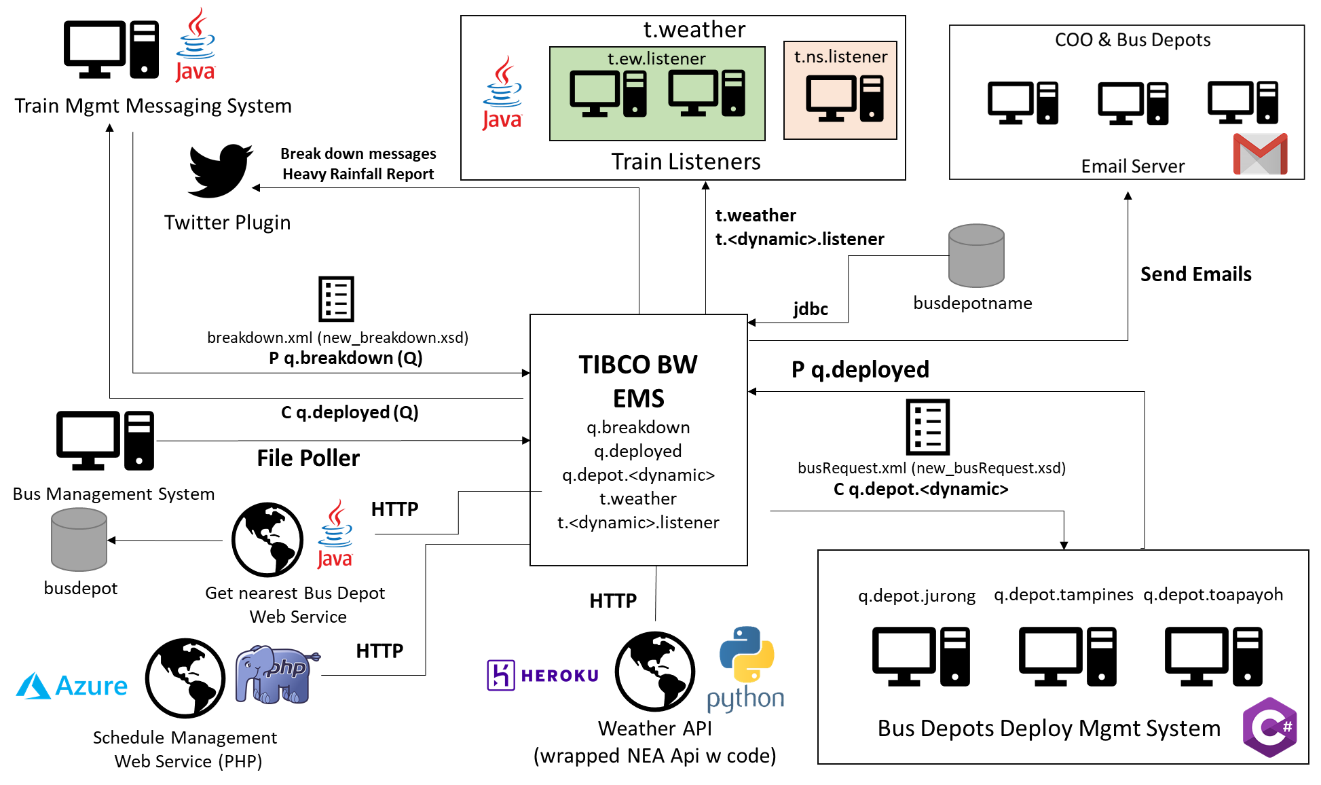
# Business Scenario

1. The Train Management System (TMS) would first receive a Train breakdown message
2. TMS would sent a message via the Integration Middleware (IM) to the other trains that are behind the train that broke down to stop
3. TMS Would then sent authenticate with the use of JWT Token to contact the Bus Depot Deploy Management System
4. Bus Depot Deploy Management System would then send the nearest Depot ID to TMS via the IM
5. TMS would then check with the correspond ID of the Depot to send the message to the nearest Bus Depot via the IM
6. The corresponding Bus Depot would then receive the message and start deploying buses and also acknowledge the message by sending back a message via the IM to TMS that buses have been deployed
7. The IM would then store the start time of the train breakdown for logging purpose
8. A tweet would be sent to inform the public that the train breakdown has occur at where and when
9. Once the train has resume service, a message would be sent to the Depot via the IM to stop sending buses and store the resume service time
10. A tweet would be sent to inform the public that the train service has been resume at where and when
11. A log file would be created to store the start and stop time of the breakdown and the coordinates of the breakdown
12. An email would also be sent to the Chief Operating Officer (COO) about the breakdown

If there are changes to the bus schedule, an email would be sent to the respective bus depots that the schedule has been changed

If the weather is currently heavy rain, it would be posted via twitter to inform the drivers to drive with care

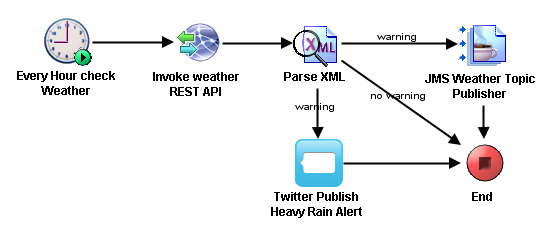
## Technical Overview Diagram

**

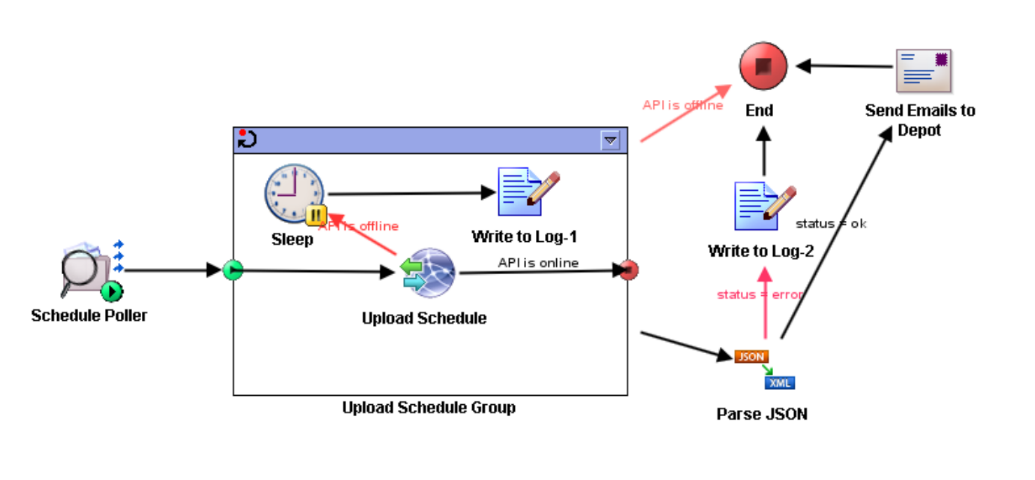
## Train Breakdown and Resumption of Service Process



## Weather Reporting Process



## Schedule Polling Process

**

# JMS Interactions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **JMS Integration** | **From** | **To** | **\*FF/**  **SRR/**  **ARR** | **Publish-Subscribe or Point-to-Point** | **JMS Queue/Topic (or Topic with durable subscription)** | **Queue/Topic Name used (case-sensitive)** |
| Integration 1 | Train Management System | Tibco IM | SRR | Point-to-Point | JMS Queue | q.breakdown |
| Tibco IM | Train Management System | Point-to-Point | JMS Queue | q.deployed |
| Integration 2 | Tibco IM | Bus Depot <Dynamic> | FF | Point-to-Point | JMS Queue | q.depot.<dynamic> |
| Integration 3 | Train Management System | Tibco IM | FF | Point-to-Point | JMS Queue | q.resumed |
| Tibco IM | Bus Depot <Dynamic> | Point-to-Point | JMS Queue | q.depot.<dynamic> |
| Integration 4 | Tibco IM | Train Listener | FF | Publish-Subscribe | Topic | t.weather |

**Legend** FF - Fire & Forget SRR - Sync Request/Reply ARR - Async Request/Reply

# Web Services

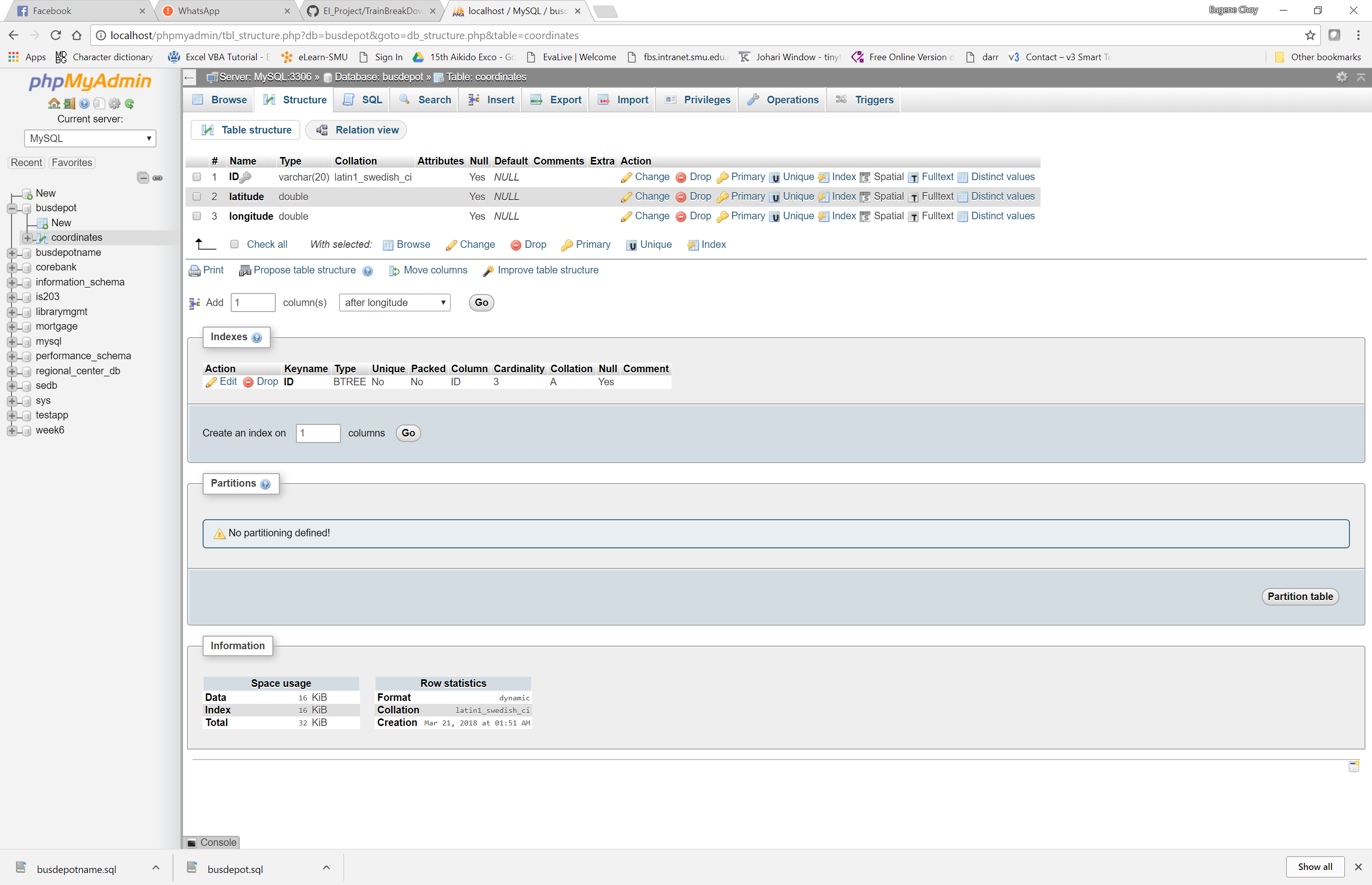
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service** | **Description** | **Protocol** | **Input** | **Output** |
| *Get Nearest Bus Depot Web Service (Java)* | *A transformed XML created from the breakdown.xml is sent into the web service as XML String to retrieve the depot ID. A JWT token is first generated from the same web service to ensure the user is allowed to access the information*  ***Self-Coded in Java*** | REST POST | *XML*  *Train id*  *Fault Type*  *Time stamp*  *Coordinates* | *Depot ID* |
| *Schedule Polling* | *Upload a txt file onto the Schedule Poller service which can be downloaded by other users.* ***Self-Coded in PHP*** | REST POST | *Text File*  *Multi Part* | *JSON*  *URL of the file*  *Status Code* |
| *Weather API* | *Check the rainfall in Singapore from NEA web service.* ***NEA API + Self-Coded wrapper in Python*** *(Self-coded wrapper randomize the chance of getting a rain fall if it is not having a rainfall in Singapore now – For Demo purpose)* | REST GET | *-* | *XML*  Heavy Rain Warning |

# Design/Schema and Content of Data

## Database

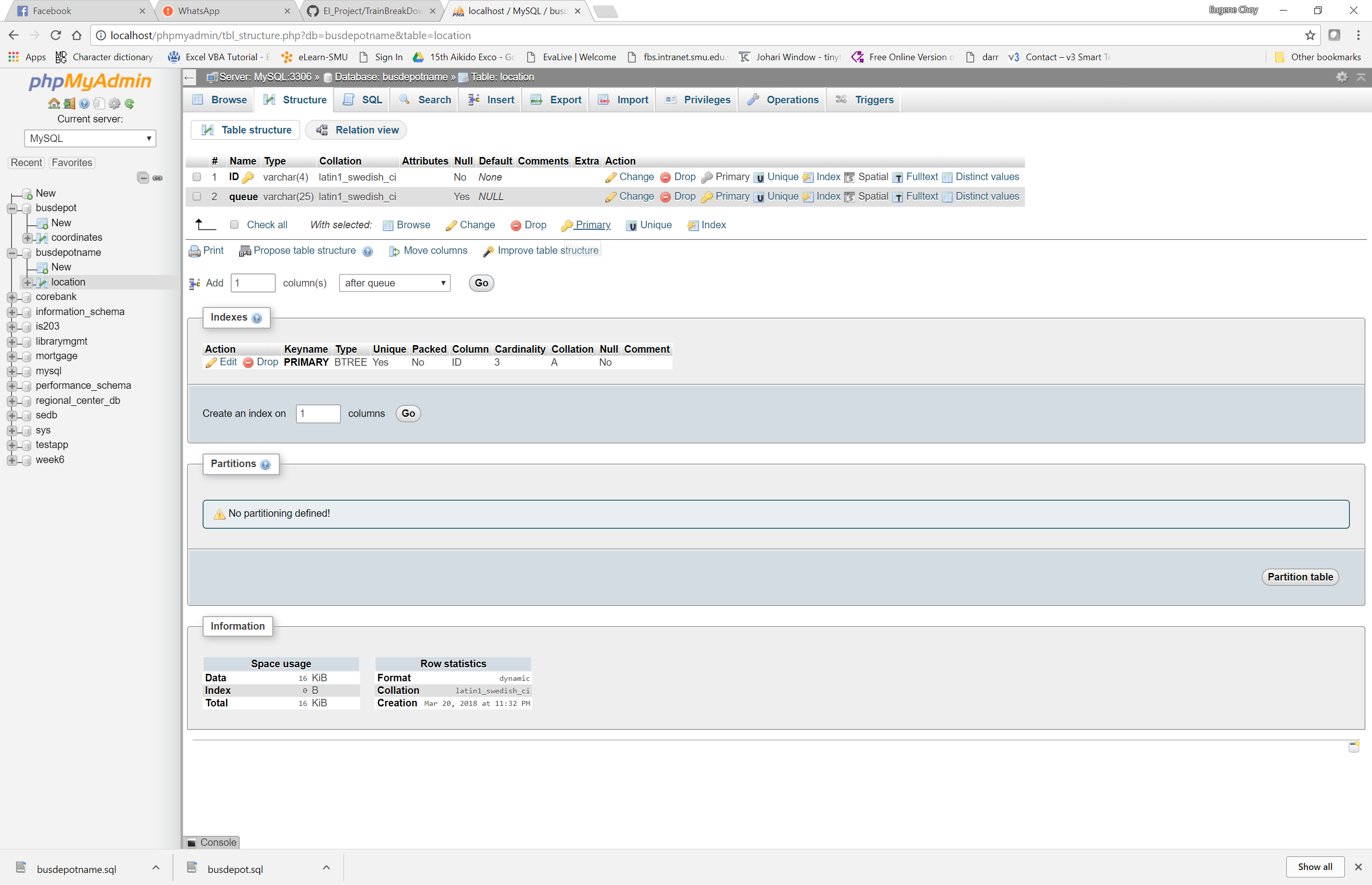
### Bus Depot

**coordinate table**



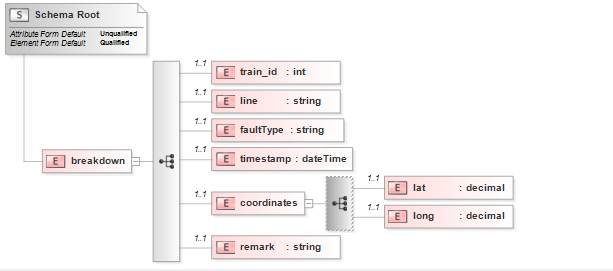
### Bus Depot Name

**location table**



## XSD

### new\_breakdown.xsd



*Sample:*

<?xml version = **"1.0"** encoding = **"UTF-8"**?>

<breakdown xmlns:xsi = **"http://www.w3.org/2001/XMLSchema-instance"**

xsi:noNamespaceSchemaLocation = **"..\\new\_breakdown.xsd"**>

<train\_id>**3**</train\_id>

<line>**ew**</line>

<faultType>**Train Fault**</faultType>

<timestamp>**2018-02-20T09:00:00**</timestamp>

<coordinates>

<lat>**1.3272383**</lat>

<long>**103.9443528**</long>

</coordinates>

<remark>**Tanah Merah Station**</remark>

</breakdown>

### new\_breakdown\_formatted.xsd

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<breakdown>

<train\_id>**3**</train\_id>

<faultType>**Train Fault**</faultType>

<timestamp>**2018-02-20T09:00:00**</timestamp>

<coordinates>**1.3272383,103.9443528**</coordinates>

<remark>**Tanah Merah Station**</remark>

</breakdown>



### busRequest.xsd

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<busRequest>

<station>**Tanah Merah Station**</station>

<reason>**Train Fault**</reason>

<timestamp>**2018-02-20T09:00:00**</timestamp>

<coords>

<lat>**1.3272383**</lat>

<long>**103.9443528**</long>

</coords>

</busRequest>

### weather.xsd

<channel>

<title>**Heavy Rain Warning**</title>

<source>**Meteorological Service Singapore** </source>

<item>

<title>**HEAVY RAIN WARNING**</title>

<issue\_datentime>**-**</issue\_datentime>

<warning> **TRUE**</warning>

</item>

<rain\_area\_image><metadata>**null**</metadata></rain\_area\_image>

<satellite\_image><metadata>**null**</metadata></satellite\_image>

</channel>

## JSON

### JSON from Schedule Polling

This JSON is returned upon successful upload of a file to the server

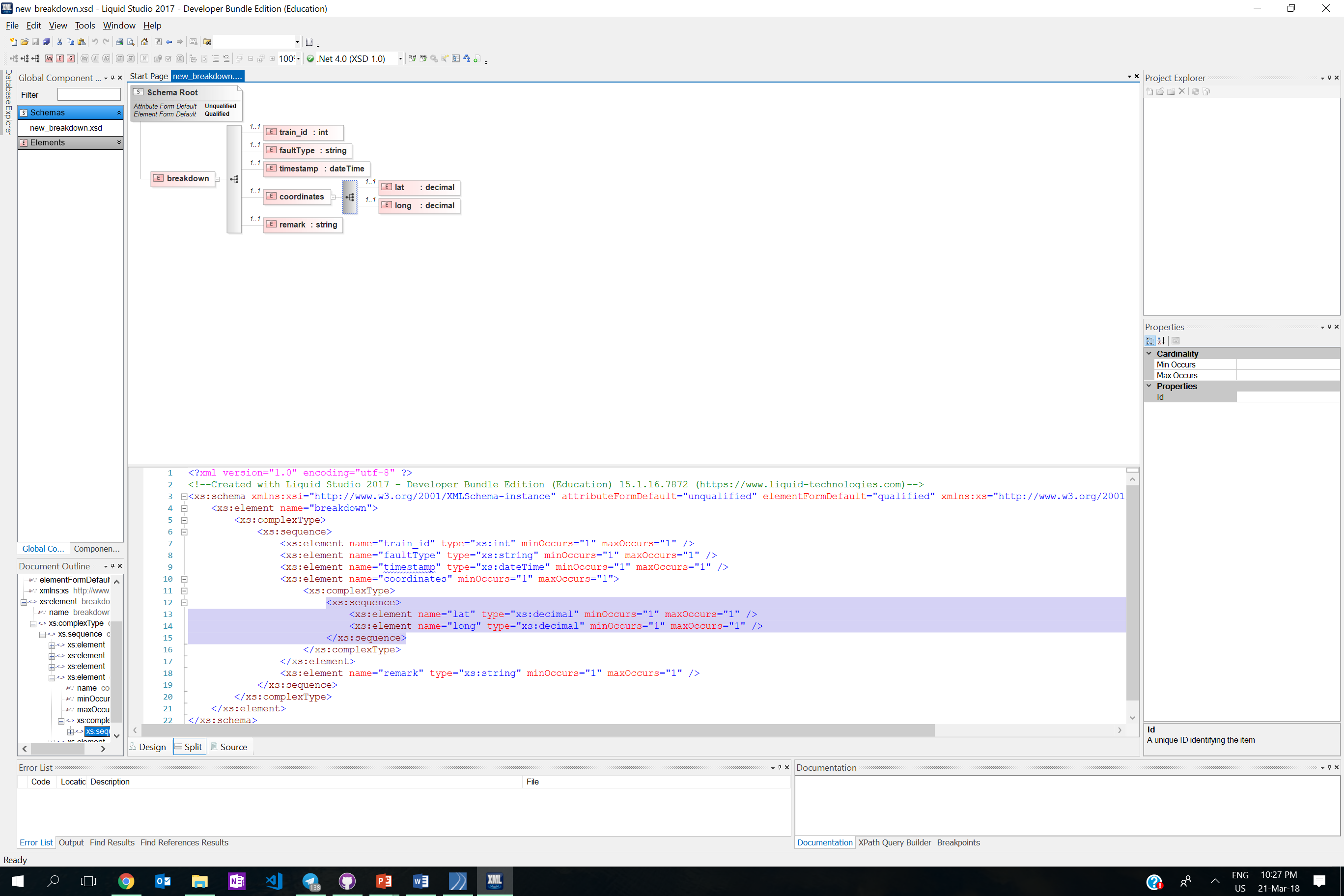
{"status":"ok",   
"statusCode":"200",   
"link":”https://eieio.blob.core.windows.net/schedule/January2018Schedule.txt" }

### JSON from getting nearest Bus Depot

{"message":"TMPN","status":"success"}

# Data Transformation

***Between Train Management System(new\_breakdown.xsd) and Get nearest Bus Depot web service (new\_breakdown\_formatted.xsd)***



The XML from new\_breakdown\_formatted expects the coordinates to be displayed in lat,long format in the coordinates tag instead of having a lat and long tag by itself. Therefore, the lat and long from new\_breakdown has to be concatenated with a comma to form the coordinates tag.

# C:\Users\i-am-\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Train Breakdown and Resumption of Service.jpgContent-based Routing

Upon receiving the break down message, the REST API (nearest bus depot) webservice will be invoked. The Depot ID will be returned to the IM. The IM will then query the database to get the destination name for the nearest Depot. The depots are all MsgConsumers on a queue with their name in this format q.depot.<dynamic> . With the queue name that is queried from the database, the deploy request will be sent to the relevant depot.

# Beyond the Labs

## Waiting for JMS Message

In the Train Breakdown and Resumption of Service process, upon deploying the bus as a mitigation action, the process will wait for a second message (With for JMS Message) from TMS upon successful resumption of service.

## Assigning a Variable

The Assign action is used to store a variable in TIBCO BW. The datetime of breakdown and datetime of resumption of service is stored in a variable for further processing and usage in the later part of the process.

## Twitter Plugin

We used Twitter plugin in Tibco BW for both the Train Breakdown and Resumption of Process as well as the Weather Reporting Process. The plugin is configured with the Oauth 1.0 for the usage of twitter. Some java properties have to be added in the designer.tra and the bwengine.tra to ensure that Twitter can be used. A tweet is sent when: service is down, service has resumed, heavy rainfall.

## Java Code

The Java Code action in Tibco BW is used in the Train breakdown and Resumption of Service process to check if the log file exists in the directory. If the file does not exist, it will return a false to prompt the IM to create a file.

## Send Mail

The Send Mail action is used to send email to the various stakeholders for the breakdown process and Schedule Polling process. Emails are sent out to the COO upon the end of an incident, to keep the COO updated the downtime of the incident. For schedule polling process, email is sent out to notify the bus depot of the URL to download their bus schedule for the month with the schedule attached as an attachment as a backup.

## Timer

We used the Timer function to run processes which require to be repeated in a short amount of time. For the weather reporting process, the timer function is used to restart the process based on the interval time specified. There is a need for constant monitoring of the weather therefore the process is put on a timer.

## File Polling

File polling in the schedule polling process listens to a certain directory for any new file added into the directory. The process will then kickstart a process to upload that new file into the server and generate a URL which allows other depot to download the file.

## Grouping Repeat-on-Error-Till-True & Sleep

If the webservice that uploads the file is down, we will sleep the process for about 10 seconds before re-attempting to upload it again. This process will repeat for a set amount of times before it ends.

## Hosting on Cloud

The wrapped version of NEA API is hosted on Heroku Cloud while the schedule polling website is hosted on Microsoft Azure.

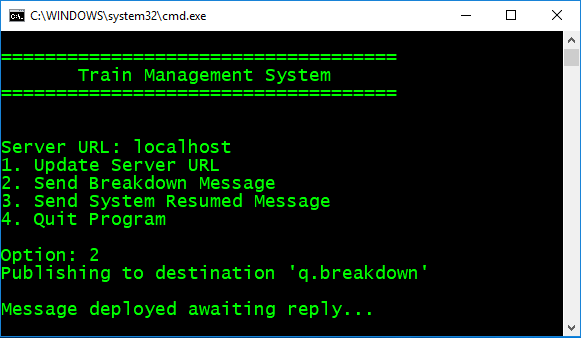
## Multi-Language Integration

Message Producers are written in java while Message Consumers were written in Java and C#. The webservices were also write in various language such as: Java, PHP and Python to simulate integration of systems in different language. In short, as much as possible, efforts have been made to simulate a real-life scenario where different system hosted on different platforms and coded in various programming languages are integrated together via TIBCO BW.

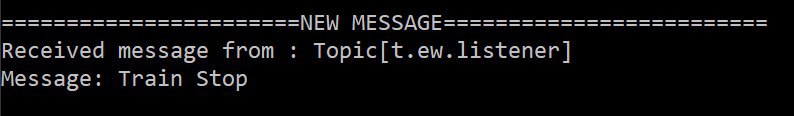
# Scenario Walkthrough (Max 10 Pages)

## Train Breakdown

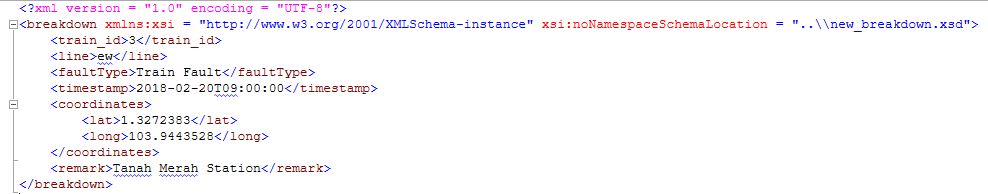
The scenario starts off when a train breaks down in one of the stations. The train would send a breakdown message to the Train Management System. (In this case, we are simulating it with the use of a command prompt to illustrate that the breakdown message is sent.)



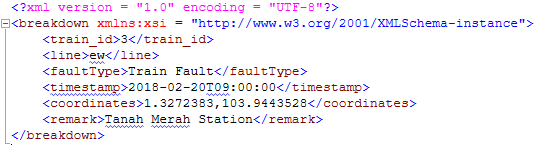
At the same time, a new message would be sent to the other trains as well to stop



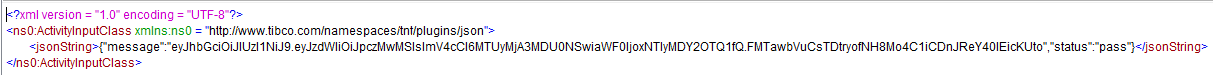
Sample of the breakdown message that is being sent to the Train Management System.

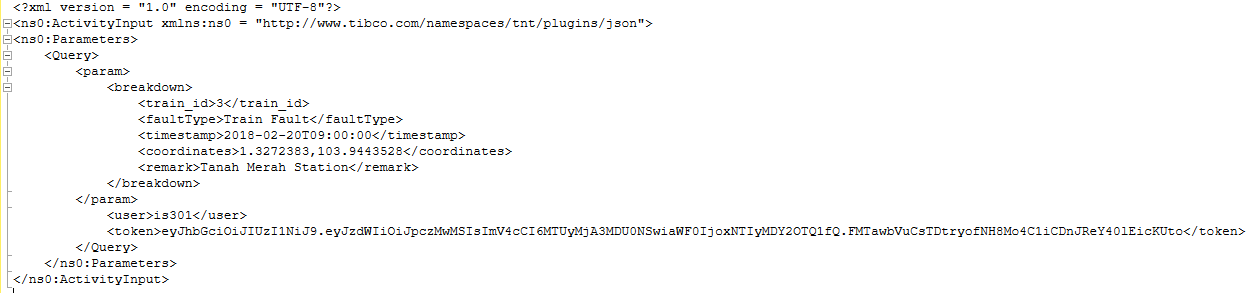


Before sending it to the Bus Depot Deploy Management System, there is a data transformation done that combines the lat and long into coordinates.

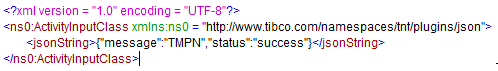


When it is communicating with the Bus Depot Deploy Management System, it will obtain a JWT token to invoke the REST API to the web service to get the nearest Depot

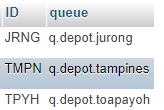




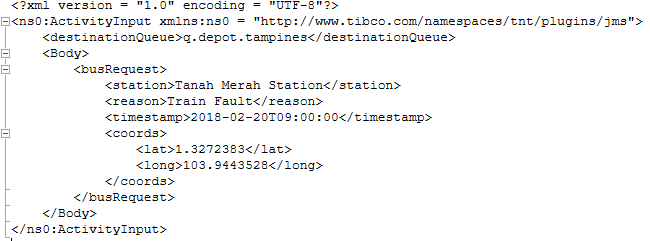
The web service would return the closest Bus Depot in acronyms in JSON format



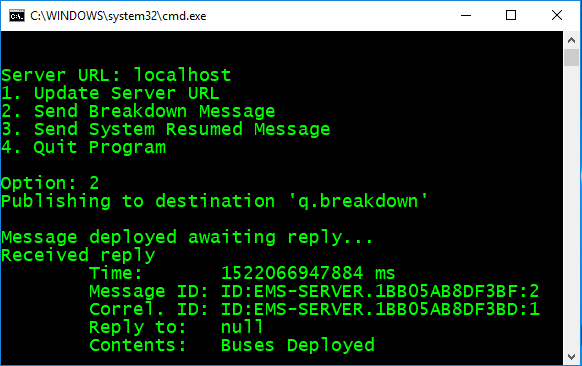
From there it would check with the database to send to which queue



Before sending the message out, it would generate a XML with the necessary information to perform the bus request and sent it to the respective queue.



Once the depot has received the message, it would reply with a message indicating the buses has been deployed.

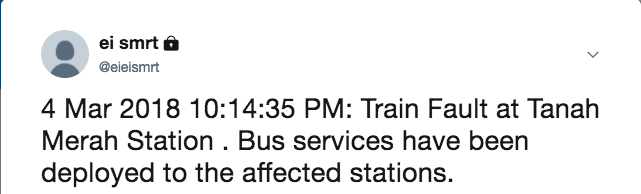


The start time of the breakdown would be stored in a assigned variable to be later on.

  
  
Before broadcasting the message onto social media, the time stamp would be formatted to the following:

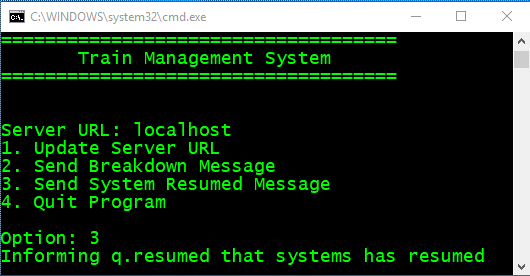
*concat(tib:format-dateTime("F MMM y hh:mm:ss a",$BreakdownDateTime/root/Datetime),": ",$Parse-XML/breakdown/faultType, " at ", $Parse-XML/breakdown/remark, " . Bus services have been deployed to the affected stations.")*

On twitter, it would reflect the following:



## Train Resume Service

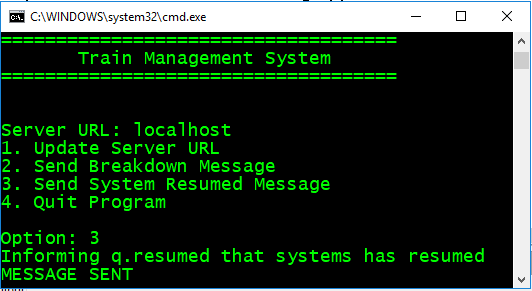
When the train service has resume, The Train Management System would send a message to the Bus Depot Management Service to stop deploying buses. (In this case, we are simulating it with the use of a command prompt to illustrate that the resume of train service message is sent.)



It would sent the following message to the depot to inform them to stop deploying buses.



After the message has been sent, The Train Management System would inform that the message have been sent.



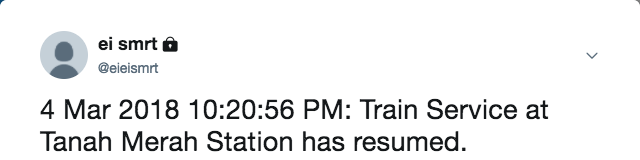
The end time of the breakdown would be stored in a assigned variable to be later on.

## 

Before broadcasting the message onto social media, the time stamp would be formatted to the following:

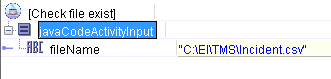
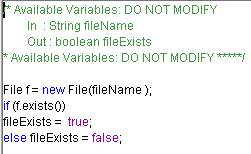
*concat(tib:format-dateTime("F MMM y hh:mm:ss a",$ResumeFromBreakdownDateTime/root/Datetime),": Train Service at ", $Parse-XML/breakdown/remark, " has resumed.")*

On twitter, it would reflect the following:

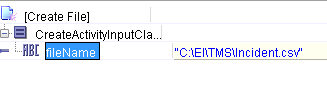


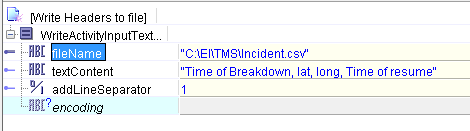
The incident would then be recorded in a csv log file of all the breakdowns that happened, with its start time and end time.

First it would run a java code to check if the file exist in the particular folder



If the file doesn’t exist, it would then check if the file exist in the particular folder. It will also create the folder if the file doesn’t exist and also writes the headers of the file.





Once the file has been created or the file exist in that directory, it will log the time of breakdown, lat, long and time of resume in the following format. (Time of breakdown and time of resume are obtained from the assigned variable during the process)

*concat(tib:format-dateTime("F MMM y hh:mm:ss a",$BreakdownDateTime/root/Datetime), ",", $Parse-XML/breakdown/coordinates/lat,",", $Parse-XML/breakdown/coordinates/long,",",tib:format-dateTime("F MMM y hh:mm:ss a",$ResumeFromBreakdownDateTime/root/Datetime))*

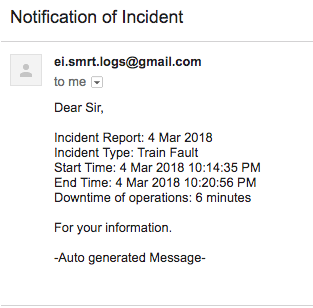
It will display the following in the document.



An email will also be sent to the COO, it would consist of information of when it happen, what fault is it, the start time and end time of the breakdown and the duration of the breakdown in the following format:

*concat("Dear Sir,&lf; &lf;Incident Report: ",tib:format-dateTime("F MMM y", $ResumeFromBreakdownDateTime/root/Datetime) ,"&lf;Incident Type: ",$Parse-XML/breakdown/faultType, "&lf;Start Time: ",tib:format-dateTime("F MMM y hh:mm:ss a", $BreakdownDateTime/root/Datetime),"&lf;End Time: ",tib:format-dateTime("F MMM y hh:mm:ss a", $ResumeFromBreakdownDateTime/root/Datetime), "&lf;Downtime of operations: ",(tib:get-day-from-dateTime($ResumeFromBreakdownDateTime/root/Datetime) - tib:get-day-from-dateTime($BreakdownDateTime/root/Datetime)) \* 1440 +(tib:get-hours-from-dateTime($ResumeFromBreakdownDateTime/root/Datetime) - tib:get-hours-from-dateTime($BreakdownDateTime/root/Datetime)) \* 60+tib:get-minutes-from-dateTime($ResumeFromBreakdownDateTime/root/Datetime) - tib:get-minutes-from-dateTime($BreakdownDateTime/root/Datetime)," minutes&lf;&lf;For your information.&lf;&lf;-Auto generated Message-")*

The following is a sample of the output of the email:



## Weather Reporting Scenario

|  |  |
| --- | --- |
|  |  |

Every hour, TIBCO BW will invoke a REST API via HTTP GET (<https://ei-weather-api.herokuapp.com/key/781CF461BB6606AD48001FDD2657FAF0D8CE9493DE977503>) to check for heavy rain warning alert. The Rest API will then return either one of the following XML:

|  |  |
| --- | --- |
| **Heavy Rain Warning** | **No Warning** |
|  |  |

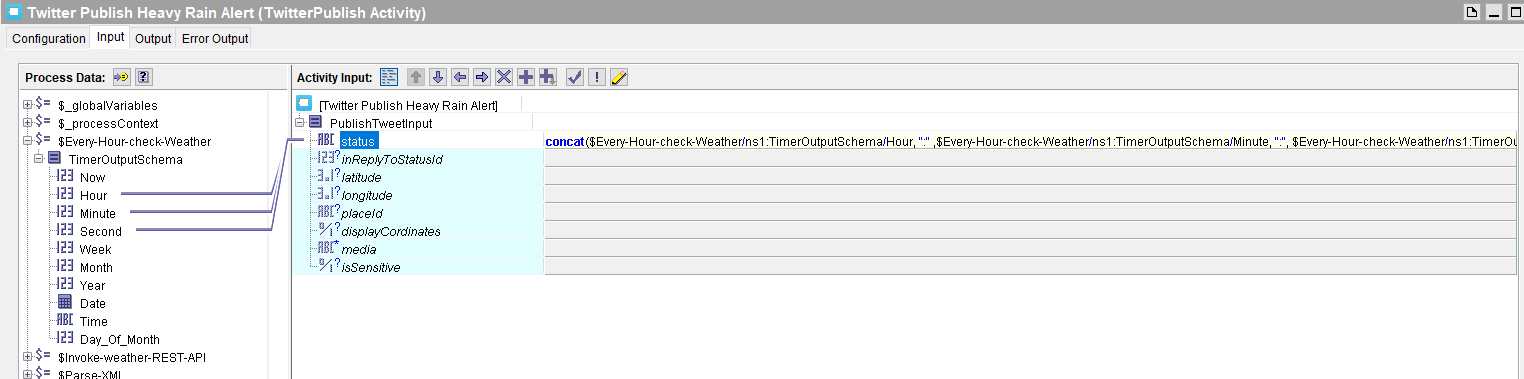
In “Parse XML”, it will use the XSD from “weather.xsd” to perform the transformation.

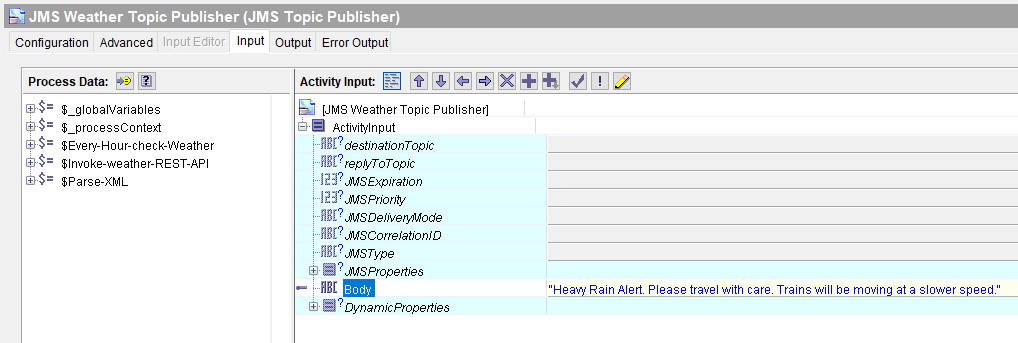
|  |  |
| --- | --- |
|  |  |

The “Transition” arrow will then perform XPATH to query for the content of “Warning” element

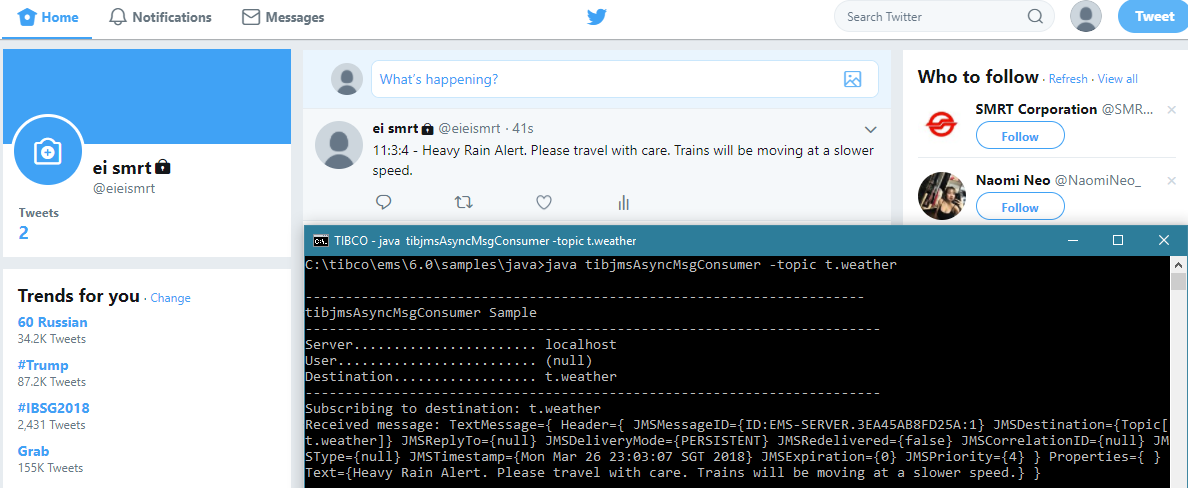
|  |  |
| --- | --- |
| **Warning** | **No warning** |
|  |  |

If “Warning” != “NIL”, TIBCO BW will route the message to “Twitter Publish Heavy Rain Alert” and “JMS Weather Topic Publisher” listening on destination “t.weather” as shown in the below screenshot.





The follow will then be displayed on Twitter and JSM listening to “t.weather”:



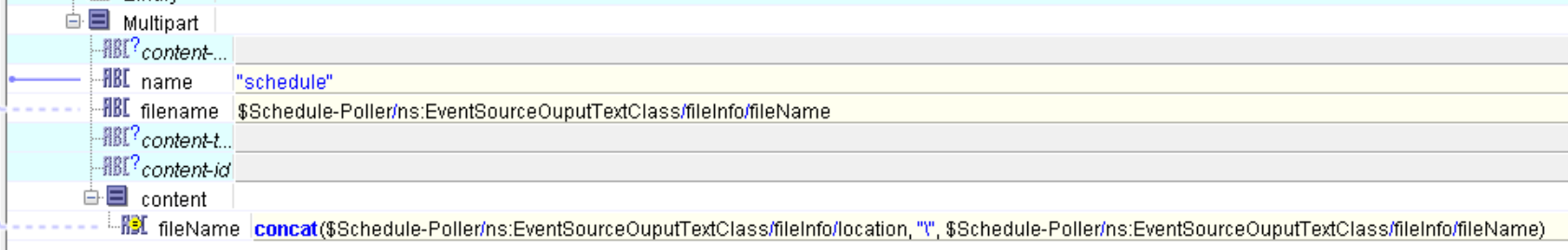
If “Warning” = “NIL”, The process will end here and no message is expected on Twitter and JMS listening to “t.weather”. The process will repeat again in an hour time.

## Schedule Polling

The scenario begins with the “Schedule Poller” activity checking the schedule folder (as indicated in the **File Name)** every 5 seconds for new text file added to the folder.

|  |  |
| --- | --- |
|  |  |

When a new file is added to the folder, the “Schedule Poller” activity will pick up the file and invoke a REST API via HTTP POST with “Upload Schedule” activity to <https://eieio.azurewebsites.net/schedulepolling/uploadschedule>. The file will be uploaded with the param key of “schedule”.



If the process faced difficulty reaching the API (local or remote is down), it will attempt to upload again for 10 times at an interval of 10 seconds. For every failed attempt, the timestamp will be logged in a log file in the below format:

*2018-03-20T19:36:09.272+08:00 Failed to connect API. Resource may be offline or not available*

The log file can be found (automatically created) at: “C:\EI\Project\log.txt”

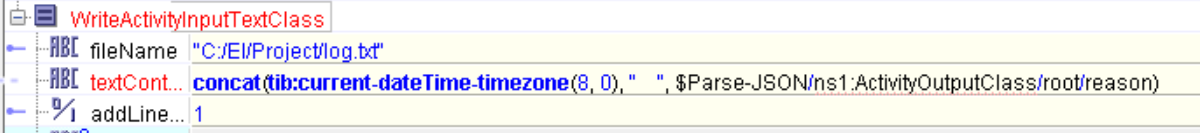
If at any point within the 10 tries, it manage to successful upload, the process will continue. Otherwise, it will end the process instance instantly after the 10th tries.

If the upload is successful, the “Upload Schedule” activity will return a JSON response as follows:

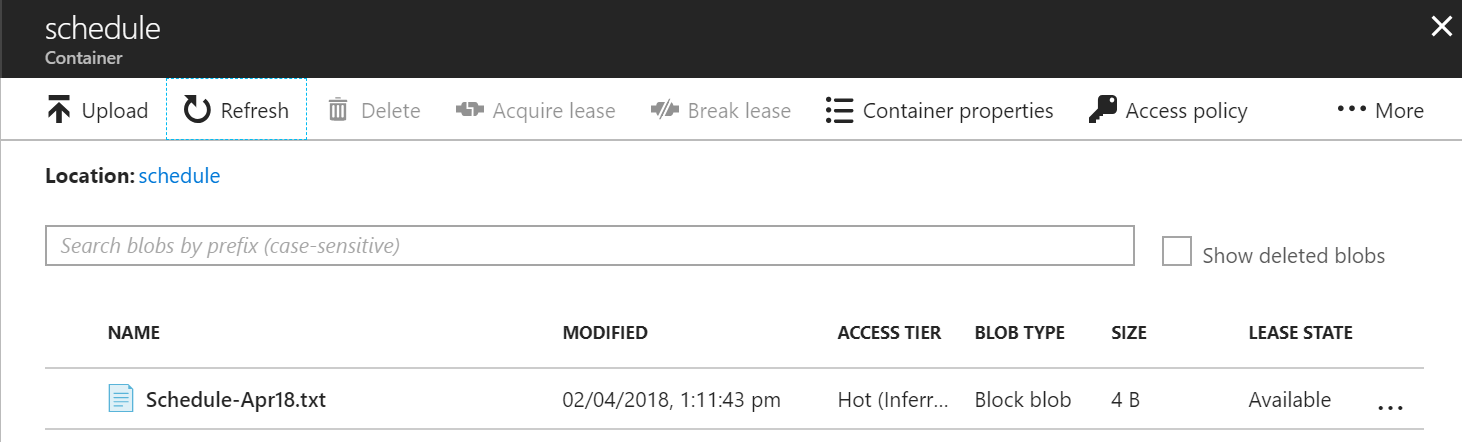
|  |  |
| --- | --- |
| {  "status": "error",  "statusCode": "503",  "reason": "Error opening file to convert to blob for uploading"  } | {  "status": "ok",  "statusCode": "200",  "link": "<https://eieio.blob.core.windows.net/container/schedule>.txt"  } |
| Example of JSON Response with error | Example of a successful upload of the schedule |

The JSON response will be parse by “Parse JSON” activity. If the JSON’s status returns “error”. It will go to “Write to Log 2” activity where the reason for the error will be stored in a log file in this format:

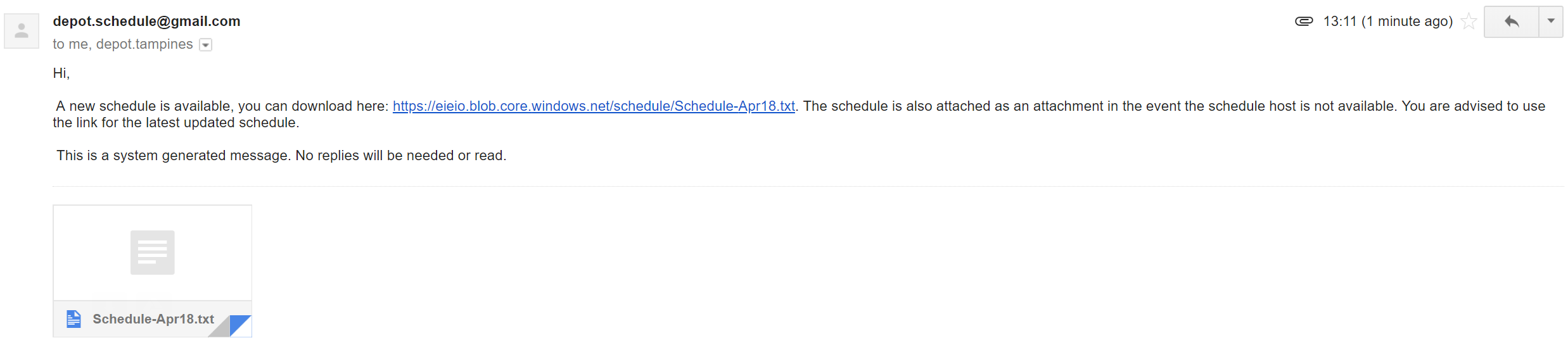
*2018-03-20T19:42:01.273+08:00 Error opening file to convert to blob for uploading*

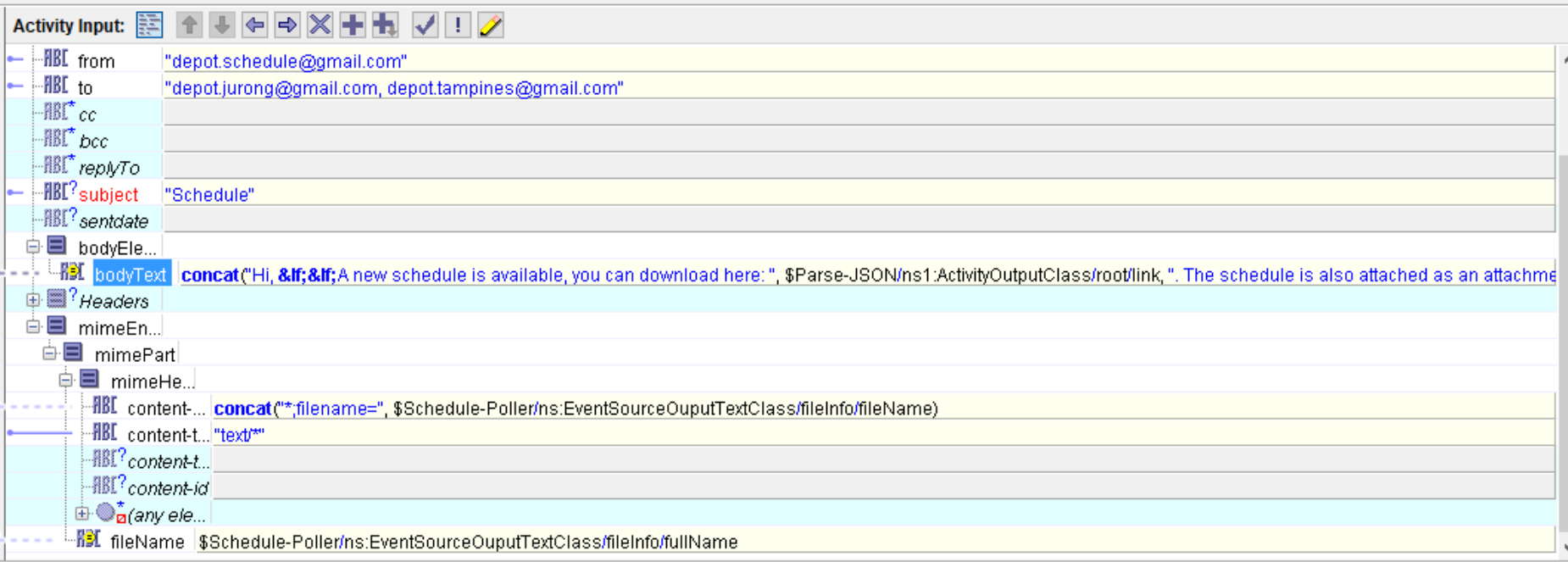


If the JSON’s status returns “ok”, it means the file has been successfully uploaded into Azure Blob Storage.



It will continue to “Send Email to Depot” activity where an email will be sent to 2 different emails ([depot.jurong@gmail.com](mailto:depot.jurong@gmail.com) and [depot.tampines@gmail.com](mailto:depot.tampines@gmail.com)) via [depot.schedule@gmail.com](mailto:depot.schedule@gmail.com) containing the link to the recently uploaded schedule with the message content as follows:





The process instance for this file will end but another instance will begin for every new file added to the schedule.