How I reverse engineered my bus stop service to make my own app: Part 2: Structure the data

Before reading this article, you can check out the previous part [where I found all the necessary data](https://therolf.fr/anchor/index.php/posts/optymo-reverse-engineering-part-1).

Now time for us to structure the data around classes. I discovered many things during this journey. One thing that I truly experienced is that the needs define the classes. This is due to the fact that [Object-Oriented Programming](https://en.wikipedia.org/wiki/Object-oriented_programming) is focused on the data approach when [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming) is focused on the processing approach.

What I have currently is data. What I need is to structure them in a practical way so that is serves what I am going to do.

For each class I will have what is relevant to store, then what is relevant to get.

# A stop

First, I need the most basic thing in my network. I need stops. And what does a stop have? A **name** of course and here it’s crucial because the name gives us the [**slug**](https://en.wikipedia.org/wiki/Slug_(publishing)) which allows us to identify each stop. The last thing we could add is a **list of lines** that are coming to this stop.

What I need to know is the slug for a given stop name. What I deduced last time that a slug has no accents and contains only lowercase alphanumeric characters. From this description we can write the corresponding Java code:



*Example: “Techn'hom 1/UTBM” => “technhom1utbm”*

Then what I need to get from a stop is its next times, but this method is too complicated to be explained [right now](#_How_to_retrieve).

# A line

Second naturally comes the line. Each line is described with a **number** and the **destination of the line**. And inside a line there is a **list of stops**. I will need these lists for the favorite part of my application in order to filter by line and then select a stop in the corresponding line. Please notice that when I say, “a line”, I am talking of something that has the same number and destination name. The “problem” is that a line has minimum 2 directions.



*Here we have a typical example with a separation. Here we will have 3 lines: “[3] Valdoie, [3] Châtenois and [3] Gare TGV”*

# A direction

I will explain what I mean when I say a direction. For me a direction is what compose a favorite. What you put as a favorite is a stop but also what is the direction you are interested in. Which means it’s a stop with its **name**, its **slug** but also a **line number** and a **direction**. What I need to get then is those informations so [getters](https://en.wikipedia.org/wiki/Mutator_method) will do the job.



# A next time

This is a crucial data here. Here the next time is the next time a bus is coming for a given direction described above. So, it is a **direction** with a “**next time**” string:



# How to retrieve next times for a given stop

Last time, we saw where I found the data and a webpage. This web page [URL](https://en.wikipedia.org/wiki/URL) contains a [stop slug](#slugtransformation). What I did is that I downloaded the source code of the page.



*The interesting part of the code*

Next I used a [HTML](https://en.wikipedia.org/wiki/HTML) [parser](https://en.wikipedia.org/wiki/Parsing#Parser) [library](https://en.wikipedia.org/wiki/Library_(computing)) ([JSOUP](https://mvnrepository.com/artifact/org.jsoup/jsoup)) in order to convert the downloaded string into proper “html document” java object.

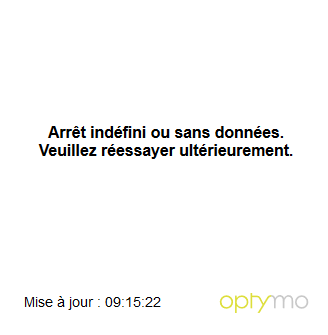
This code was written in a static method so that you don’t need a stop object to access it. It takes as a parameter a [stop slug](#slugtransformation).

Given the example, we can extract our informations by getting the text of each [element](https://en.wikipedia.org/wiki/HTML#Markup) with the corresponding classes:

* **f1**: provides the line number
* **f2**: provides the line direction
* **f3**: provides the “next time” string

Finally, I can construct my next time object.

*How do you do if you have an error?* This is a good question. If there is an error, this page is shown:



The source code of this page does not contain any HTML element with the previous classes. This however does not mean that the stop doesn’t exist. This response is shown when the Optymo service is not available **at night** or when there have **no more buses** coming.

# The network

This class is bigger than all previously described classes before. She’s the one that regroups all the data with every stop and every line. What it means is that in a network we have is a **list of stops** and a **list of lines**. What we need to get is the whole network or “how to generate an entire network with stops and lines”. Also, I need to fetch a **stop** by a given **slug** or a **line** by its **line number** and its **direction**.

# Network generation process

The way we will generate this network is thanks to the [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language) file we found last time. However, this file does not only include urban bus stops. It also includes sub-urban stop buses, bike sharing stations and self-service car locations.

I then used the native java XML parser to get a java object I can work with.

Now I have to loop through all the “locations” and clean their name and send them to the method where I [get the next times](#_How_to_retrieve). *How do you know this is a real stop?* That’s the thing, it requires to be generated during a weekday in daytime (~ en journée en semaine). If I get a result, then it’s a bus stop, I add it and for each next time that I get it’s a potential new line where I add my fresh stop. This process however is very long and takes up to 20 seconds (fair enough there is more than 500 “locations” in this file). At the end, I use a [JSON](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation) parser in order to generate a string that I can export to a file.

This generation time is too important for the first launch of the app, so I must arrive with a pre-generated JSON network file.

# Improve the network

However, that’s not all folks. Some “big” stops are real hubs in the network, and they have more lines coming though than the number of lines for a next time request. So, I need to improve my network by doing a similar procedure to generate the network.

Eventually, we made a structure that we will use in our application. I “only” have to make the app then.

Thanks for reading. See you later for the next part about the app.