MIDDLEWARE / SERVICE ORIENTED COMPUTING

LET'S GO BIKING !

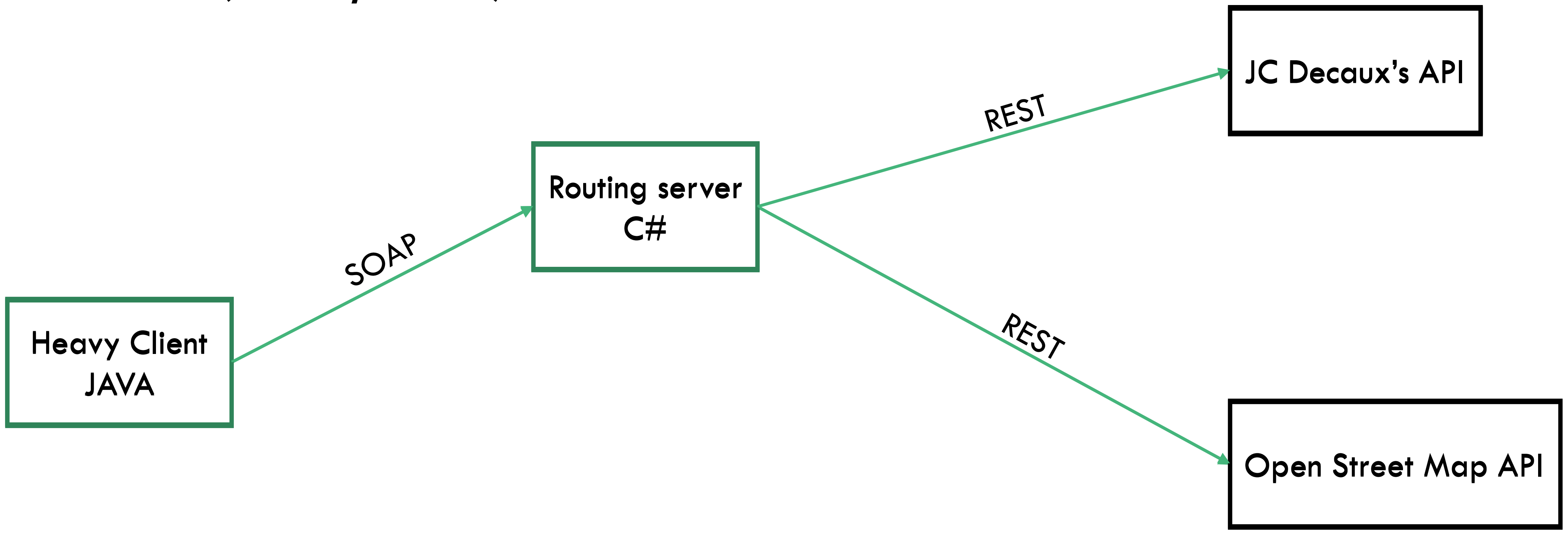
CONCEPT

The goal of the Let’s Go Biking! project is to develop a small app to compute itineraries by reducing as much as possible the distance to cover by foot (by using bikes instead).

EXTERNAL RESOURCES

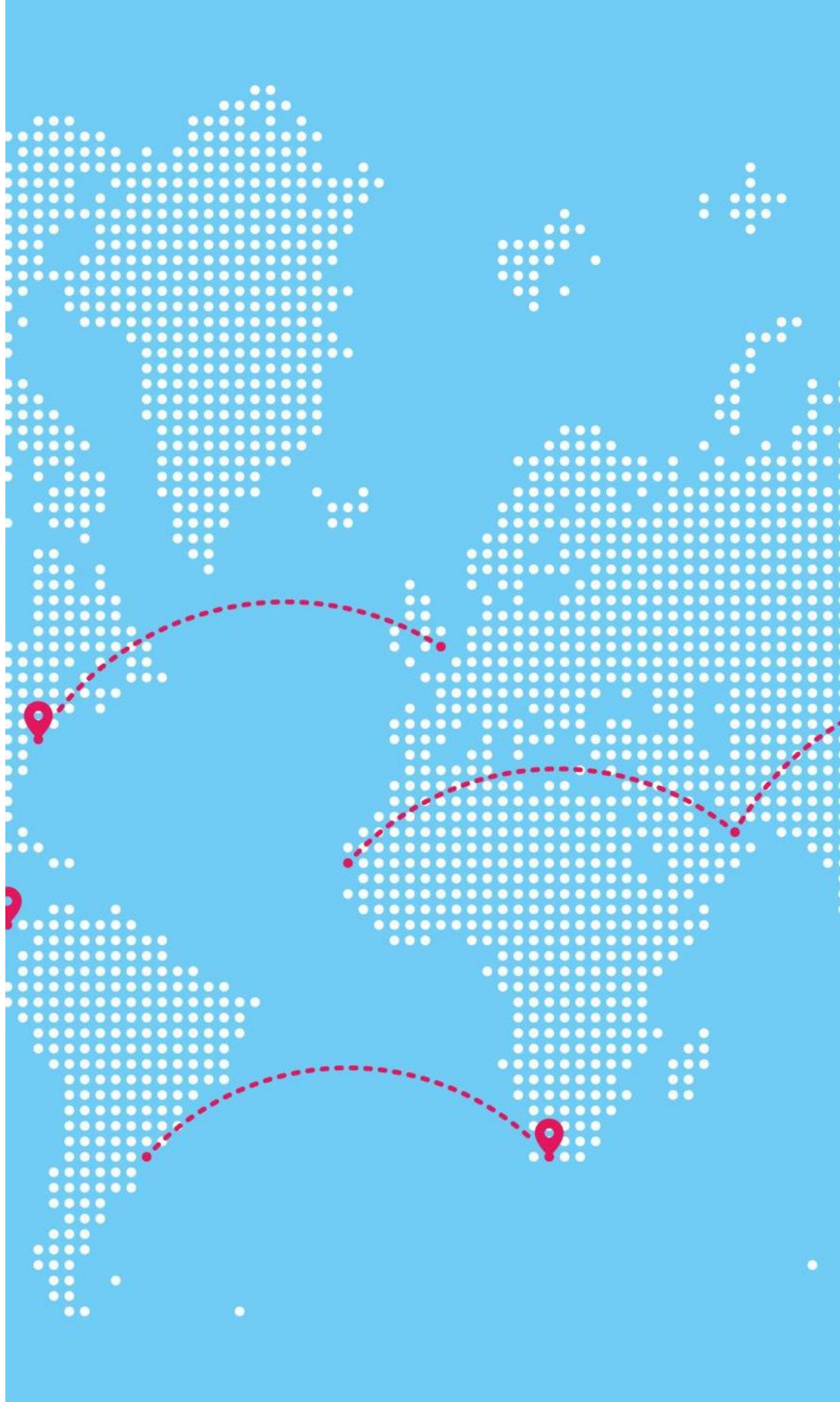
- One of JC Decaux’s business is renting bikes. They have stations in many cities where anyone can get or drop a bike (like Vélo bleu), and they developed an Open API offering real-time information about their stations.
- OpenStreetMap is an open map project, offering services such as GPS itineraries.

MVP (10/20)


Heavy Client
JAVA
SOAP
Routing server
C#
REST
JC Decaux's API
REST
Open Street Map API

MVP – JAVA HEAVY CLIENT

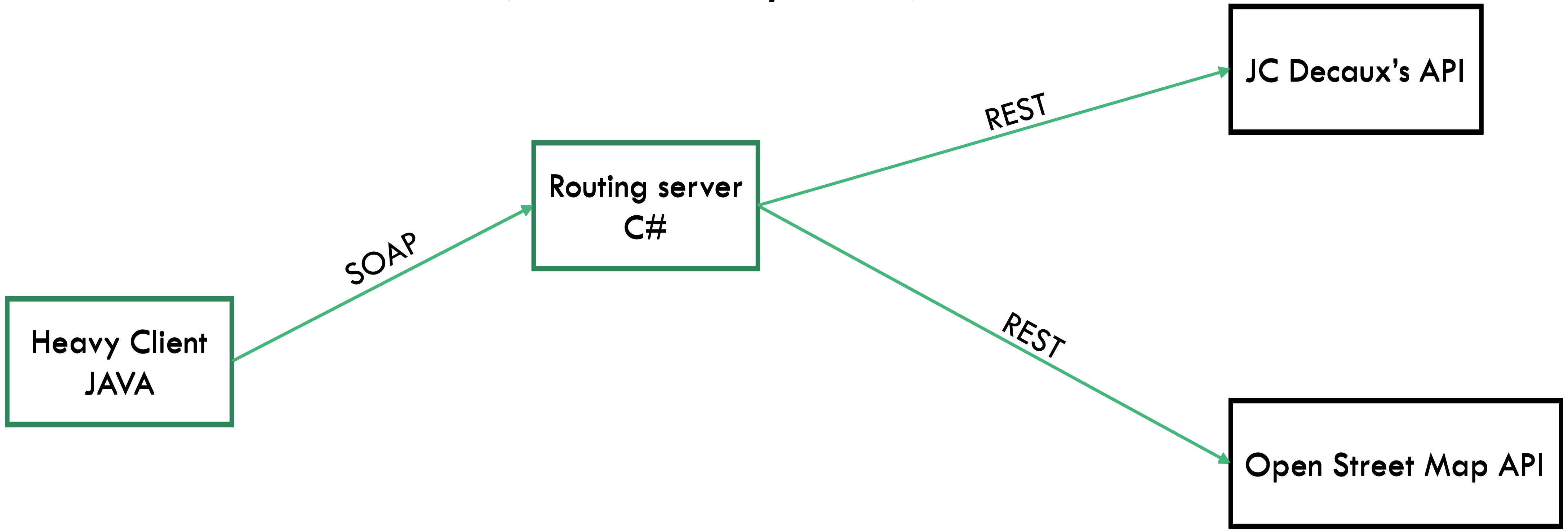
1. Asks the user for the origin and destination.
2. Calls the routing server to get the itinerary between those points.
3. Displays the itinerary (ideally on a map, or at least the detailed instructions).



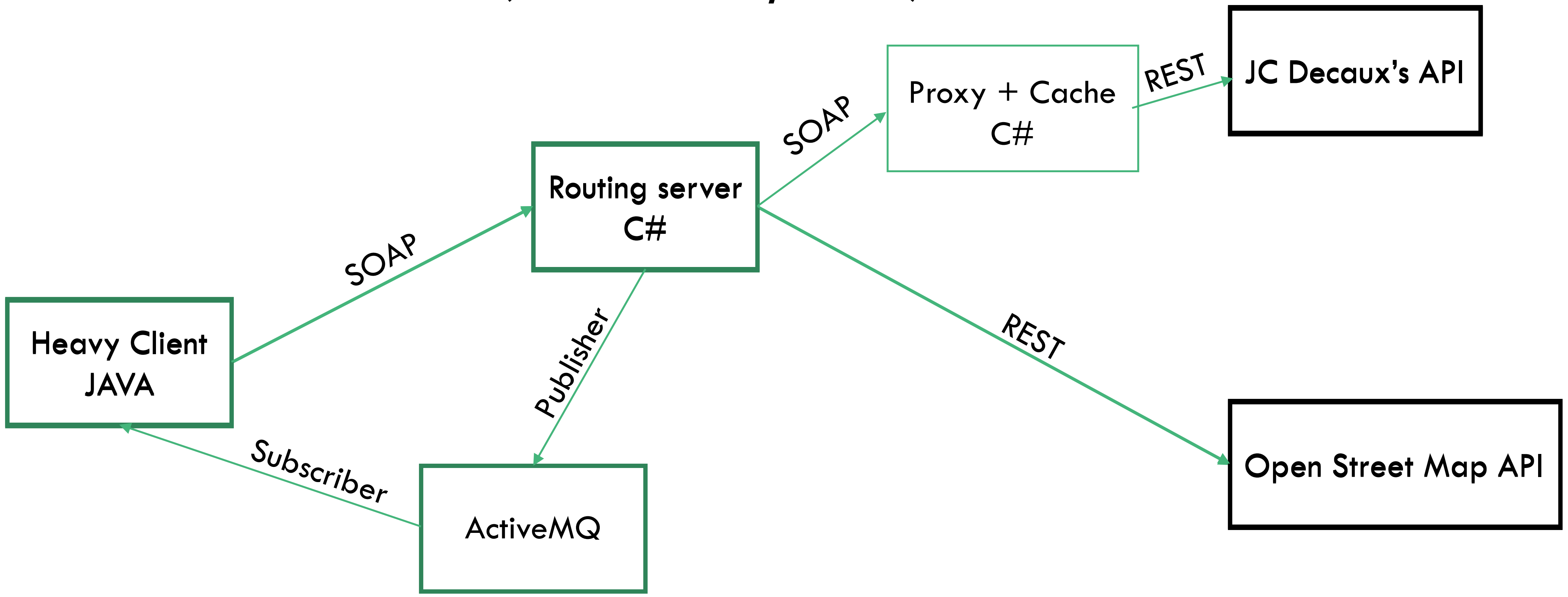
MVP – ROUTING SERVER

- Has one endpoint and one function GetItinerary(origin, destination)
    1. Calls OpenStreetMap to retrieve information about the given address.
    2. Calls JCDecaux to retrieve the stations and find: the closest one from the origin with available bikes, and the closest from the destination with places to drop the bike.
    3. If there are stations matching the conditions, calls OpenStreetMap for 3 itineraries: Origin to Station1, Station1 to Station2, Station2 to Destination

ADDITIONS (11-20/20)



ADDITIONS (11-20/20)



ADDITIONS – PROXY / CACHE

- JC Decaux’s stations status can be stored for a given amount of time, as there should not drastically change in a few seconds (or minutes), so to avoid overloading the server, an addition is to implement a cache inside a proxy which will manage the requests to JCDecaux.

ADDITIONS – ACTIVEMQ V1

- Getting all the itinerary information right at the start is not smart, firstly because it may imply the transfer over the network of a lot of information, and secondly because those information may not stay up-to-date.
- The idea is to include a messaging queue, in which the server will send the itinerary step by step, and the client will consume the information at its own pace.
- NB: the base feature (getItinerary) must still be available in case the client cannot communicate with the queue.

ADDITIONS – ACTIVEMQ V2

- In v1 we simulate the use of the queue by sending the entire itinerary directly in the queue.

  In v2, we want the client to tell the server when it reached a given step, so the server can check if the itinerary is still OK (number of available bikes, spots to drop the bike, …) and publish to the queue the new information.

- We'd also like, in addition to the itinerary steps, some meta information you consider useful to the user.

DETAILS

- Evaluation method: live demo + questions (~ nothing to prepare for the demo, we will tell you what to do during the interview).
  NB: we may (and most probably will) ask you to explain some bits of code you wrote.
  - This is not a C# / Java course, so we won't evaluate how good your code is. However, we reserve the right of adding a few bonus or malus points for code quality.
- This project can be done by group of 2 people max, but the **demo is alone.**
  NB: We let you work in pairs to be faster, not to only work on half the project → "I'm not the one who developed this feature" won't be accepted during the demo.
- Duration: 3 labs (~4 weeks)
  - The last lab of the semester will be for the demo.