BlablaMove Architecture - Mobile tracking app

Members :
Couvreur Alexis
Spinelli Aurélien
Swiderska Joanna
Wilhelm Andreina

Tutor :
M. Guilhem

September - February 2018/2019

Contents

1 System Description **2**

2 Mobile Tracking App **2**
2.1 Scope . . . 2
2.2 Scenarios . . . 3
2.2.1 Tracking with direct route . . . 3
2.2.2 Tracking with route combination . . . 4
2.3 Architecture: Component Diagram . . . 5
2.4 Technological Stack . . . 6
2.5 Roadmap . . . 6

3 BlablaMove : Project update **8**
3.1 Reminder . . . 8
3.1.1 Scaling up . . . 8
3.1.2 Solution . . . 9
3.2 New requirements . . . 9
3.3 New requirements : new solutions . . . 9
3.3.1 Chaos Broker : A Kafka proxy . . . 9
3.3.2 Refactoring services . . . 9
3.3.3 Kubernetes . . . 10
3.4 Experiments . . . 11

1 System Description

BlablaMove is an application that it is meant to help students move their goods or furniture for a much lower price by using the free space in other people's cars who are going to the same destination or doing part of the path.

2 Mobile Tracking App

As part of the entire system this app will allow students follow their goods along the way from start to finish and track all the possibles changes in between.

2.1 Scope

The general architecture of the app is shown in the Use Case Diagram of Figure 1 where there will be two main users: Student and Driver.

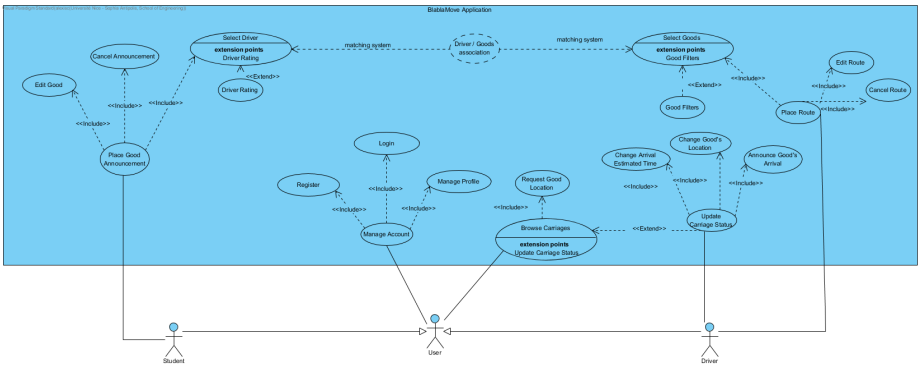

Figure 1: General architecture of the app

For the scope of this project the following functionalities of the app will be developed:

- **Account management:** Create, edit and delete an account. Log in and log out.
- **Announcement management:** Create, edit and delete the different types of announcement. For *students* it will be an announcement for moving goods and for *drivers* it will be a free space for transportation kind of announcement.
- **Tracking:**
  - Notification on received
  - Notification on start
  - Notification on checkpoints

  - Notification when there is a change of drivers (route combination)
  - Notification on incidents
  - Notification on arrival
  - Notification on delivered

On the other hand, the following functionalities will be mocked as they are not part of the scope, but are needed for the development of the app:

- **Matching system goods/routes:** Assign a driver to a good delivery.
- **Billing:** Distribute points according to service.
- **Volume assessment:** Estimate the volume of the goods to be moved.

2.2 Scenarios

Personas:

- *Lucas* is a student living in **Sophia** that needs to send his bike to his brother *Charles* in **Paris**
- *Charles* is *Lucas'* brother living in **Paris**
- *Austin* is a mechanic living in **Nice** who's traveling to **Lyon** soon
- *Mila* is a dancer living in **Lyon** who's traveling to **Paris** soon
- *Hope* is a student living in **Nice** who's traveling to **Paris** soon

2.2.1 Tracking with direct route

This scenario describes the tracking of the goods in a direct route (only one driver involved) with no incidents recorded.

1. *Lucas* create an announcement stating:
   - Bike 2 wheels, 8kgs
   - Departure: Sophia, 10km radius, before October the 12th
   - Arrival: Paris 10th arrondissement, before December the 24th
   - Picked up by: *Charles*
2. System finds a matching route with *Hope*
3. *Lucas* and *Hope* are notified and they agree. All the announcements change their status.
4. *Lucas* meets *Hope* and gives her the bike. She notifies on the app that she's received it.

5. *Hope* leaves the following day. She notifies when her trip begins.
6. *Hope* meets *Charles* and proceeds to give him the bike
    - *Hope* notifies that she has completed the delivery.
    - *Charles* notifies he has received the bike.
    - The system transfer points to *Hope's* account

2.2.2 Tracking with route combination

This scenario describes the tracking of the goods in a combination of routes (more than one driver involved) with no incidents recorded.

1. *Lucas* create an announcement stating:
    - Bike 2 wheels, 8kgs
    - Departure: Sophia, 10km radius, before October the 12th
    - Arrival: Paris 10th arrondissement, before December the 24th
    - Picked up by: *Charles*
2. System finds a matching route combination with *Austin* and *Mila*.
3. *Lucas*, *Austin* and *Mila* are notified and they agree. All the announcements change their status.
4. *Lucas* meets *Austin* and gives him the bike. He notifies on the app that he's received it.
5. *Austin* leaves the following day. He notifies when his trip begins.
6. *Austin* meets *Mila* and proceeds to give her the bike.
    - *Austin* notifies that he has completed his part of the delivery.
    - *Mila* notifies that she has received the bike.
    - The system transfer points to *Austin's* account
7. *Mila* leaves the following day. She notifies when her trip begins.
8. *Mila* drives through **Dijon** and notifies the checkpoint.
9. *Mila* arrives to **Paris**. She meets *Charles* and proceeds to give him the bike
    - *Mila* notifies that she has completed the delivery.
    - *Charles* notifies that he has received the bike.
    - The system transfer points to *Mila's* account

2.3 Architecture: Component Diagram

We decided to use an SOA approach, (with a kafka bus) + explain that some services might have to scale way more than others such as tracking ; For a better understanding of the system the Component Diagram on Figure 2 has been created. The components were separated server side and client side where most of our system will be on the server side.

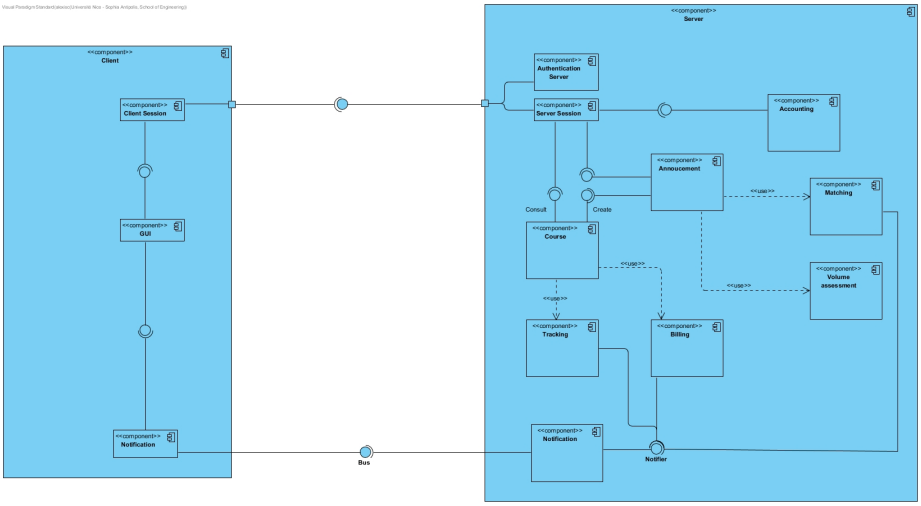

Figure 2: Component Diagram

The components are described below:

- Client's side components:
  - **Client Session** - contains informations on client session
  - **GUI** -graphical user interface
  - **Notification** - receiving notifications and displaying them
- Server's side components:
  - **Authentication Server** - allows user sign in/sign out
  - **Server Session** - contains informations about server session
  - **Course**- holds informations about the ride, created once both sides(student and driver) has accepted a ride
  - **Tracking** - gives the possibility to track a particular object
  - **Announcement** - contains informations about announcement
  - **Accounting** - holds informations about user (name, email, phone number etc.)

  - **Matching** - match student's offerts with driver's ones
  - **Billing** - billing details like points exchange between student and driver after transporting goods
  - **Volume assessment** - estimation of good's partition in order to fit it's volume into car's space
  - **Notification** - takes care of sending notifications

2.4 Technological Stack

- Service Development:
  - Server side: Java Spring Boot
  - Client side: Static pages with ajax calls / Mobile application
- Storage:
  - Database: Any SQL DB for persistant data
  - Cache Database : Kafka
- Deployment:
  - Docker Community Engine
  - Docker Compose
- Testing:
  - Acceptance testing: Cucumber (integration tests)
  - Stress testing: Gatling
  - Unit testing by services

2.5 Roadmap

The development of the app will be done in small iterations of one weak each. Different roles will be assigned, such as software architect, tester and developer. These will not be permanent and will change every week so everyone can experience each role.

The weeks are planned as follow:

- **Week 41:**
  - Choose technologies to be used
  - External and internal interfaces
  - Mock external systems

- **Week 42:**
  - Continuous integration
  - Walking skeleton
    - Student / Driver entities
    - Announcement creation
    - Matching system *(mock)*
    - Course creation
    - Basic notifications
- **Week 43:**
  - Main risk mitigated
  - Verification / tests of the system
  - Billing system *(mock)*
  - Account management
  - Initiate front interfaces
- **Week 44:**
  - Coding enough of the rest for the POC
  - User Interface
  - Notification front to back
  - Front tests
- **Week 45:**
  - POC complete

3 BlablaMove : Project update

3.1 Reminder

We previously put forward our microservices architecture and demonstrated how and why it was an adapted solution to this project. Our main argument was because such an application would need to scale up if it is deployed even coutry-wide for students only.
See a basic scenario our application must provide through APIs :

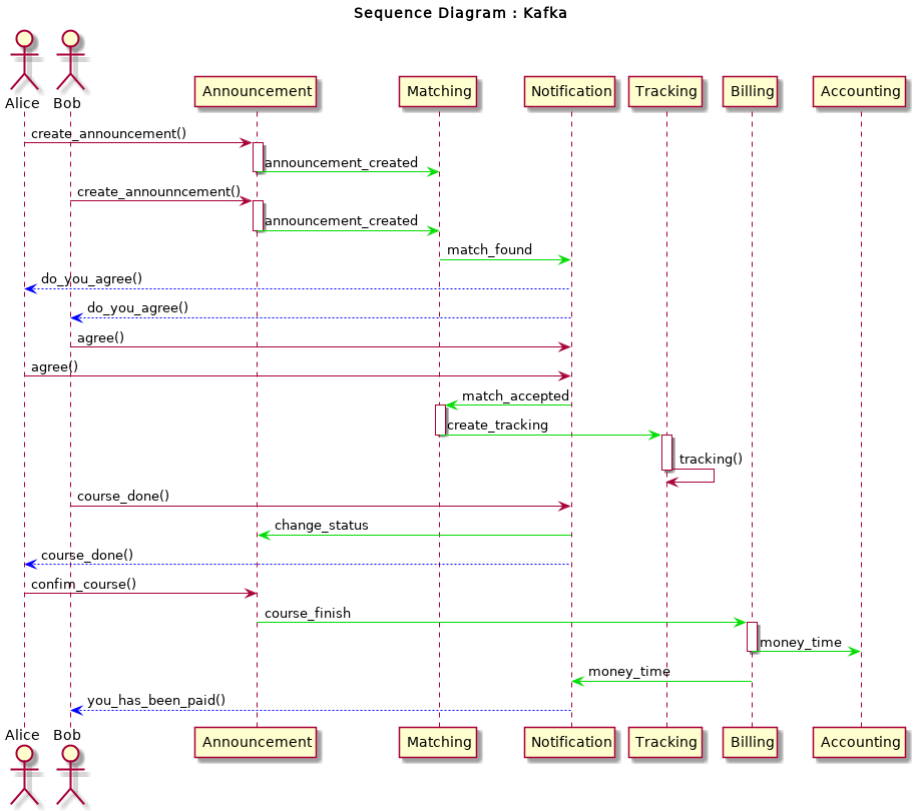

Figure 3: Scenario

3.1.1 Scaling up

In fact we strongly supported the fact that some services might be used a lot more than some others, their API being requested a lot more we would need to be able to scale this up independantly. A monolithic application would have scaled up services that are less requested than others, giving less flexibility and resource managing.

3.1.2 Solution

We built a microservices application with a Kafka message bus between these microservices, allowing async communication, duplication of services behind a load balancer. This gives more flexibility, and a more fine resource managing.

3.2 New requirements

We need to proves that the architecture is able to scale up to 400k simultaneous users. But we also need spread chaos among services by the kafka bus.

Hypothesis : Microservices allows a fine resource management which at a very high scale can be very efficient.

3.3 New requirements : new solutions

3.3.1 Chaos Broker : A Kafka proxy

Chaos Broker is an ad-hoc solution to provide disorder inside the kafka message bus. The Chaos Broker can :

- duplicate events
- delete events
- slow down events
- input salt

It will act as a proxy in front of the kafka bus and will handle messages to be send to the bus by applying randomly actions into it. This will allow us to identify weak spots inside our API, by forcing the application to be in a chaos state and having scenario we did not expect.

Each services that needs to send messages through kafka will now use the Chaos Broker instead of Kafka directly.

3.3.2 Refactoring services

Previous services are naive and do not handle these kinds of scenario. Actually they only handle input validation which should be enough for a small scale application.

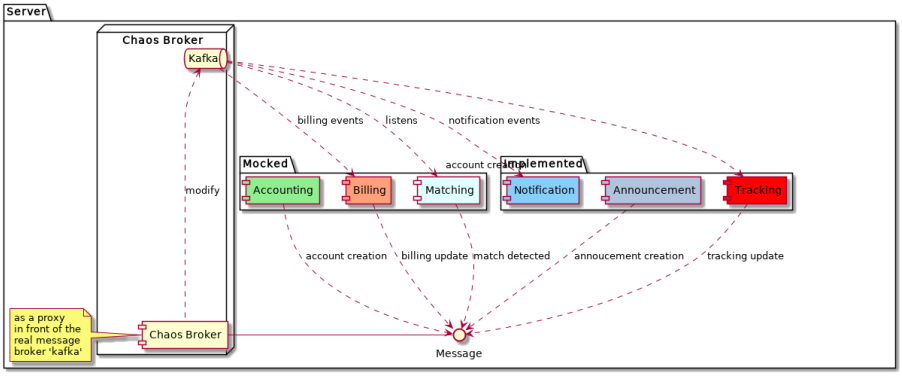

Figure 4: Architecure with Chaos Broker and Kafka

As we want to keep a coherent state even with the chaos broker system up, we will have to add more information to ensure some scenario might not corrupt the state (e.g. duplication could contains an id and thus duplication could be identified and executed once).

3.3.3 Kubernetes

As we used Docker to isolate all of our services, an answer to scaling up is to use Kubernetes.

Kubernetes (commonly stylized as k8s) is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.

It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It works with a range of container tools, including Docker, since its first release.

Many public-cloud service providers (e.g. IBM), provide Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service (e.g. IBM Cloud Kubernetes Service).

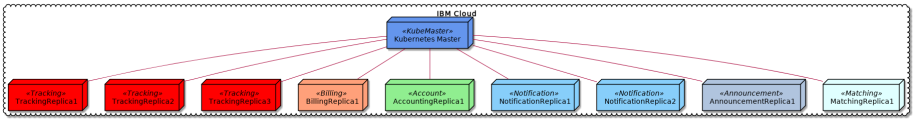

Figure 5: Kubernetes deployment with horizontal service scaling

We can use the IBM Cloud Kubernetes Service for this, using student accounts but we might be limited by our offer. Another solution would be to build a raspberry pi cluster and start a small size experiment and based on the results we could approximate resources needed for the real number of users (400k).

3.4 Experiments

In matter of testing an utilization by 400k simultaneous users we will use Gatling and design a simple pattern (based on our scenarios) to conduct the experiment.

This already have a few resources problems as a local testing:

- hardware limitation (server)
- network interface saturated (client: gatling)

Thus we will do small incremental experiments, and do the critical experiments on the cloud so we can see the Kubernetes resources management in action, and shows the microservices architecture efficiency.

We will then have multiple metrics from Gatling and others from the IBM CLoud Kubernetes Service (e.g. instsance creation of a service).