UNIVERSITY OF PISA

DISTRIBUTED SYSTEMS AND MIDDLEWARE TECHNOLOGIES

PISA-EAT
PROJECT DOCUMENTATION

ACCADEMIC YEAR 2020-2021

ANDREA TUBAK, FRANCESCO RONCHIERI, ALESSANDRO MADONNA

SUMMARY

INTRODUCTION

OVERALL IDEA

Pisa-Eat is an online platform that offers a new way of interacting between customer and restaurateur.

Restaurateurs can register their restaurant on the platform and make it available to all users.

A user can browse the site in search of his favorite restaurant and consult the available tables, once he has found the one that best meets his needs, in terms of number of seats and position, he can book it and share it with his friends.

At the table, users can consult the menu and start filling in the order autonomously; they can add, modify, or remove the dishes as they wish. At the same time users can communicate directly with the kitchen, specifying, for example, allergies or special requests that cannot be expressed in the standard order procedure.

Once all users are ready, each of them will have to confirm the order so that it can be processed by the kitchen.

The restaurateur will have access to a reserved area, where he can see and interact with every aspect of the restaurant (e.g., consult the status of the tables, change their arrangement, check orders, etc.)

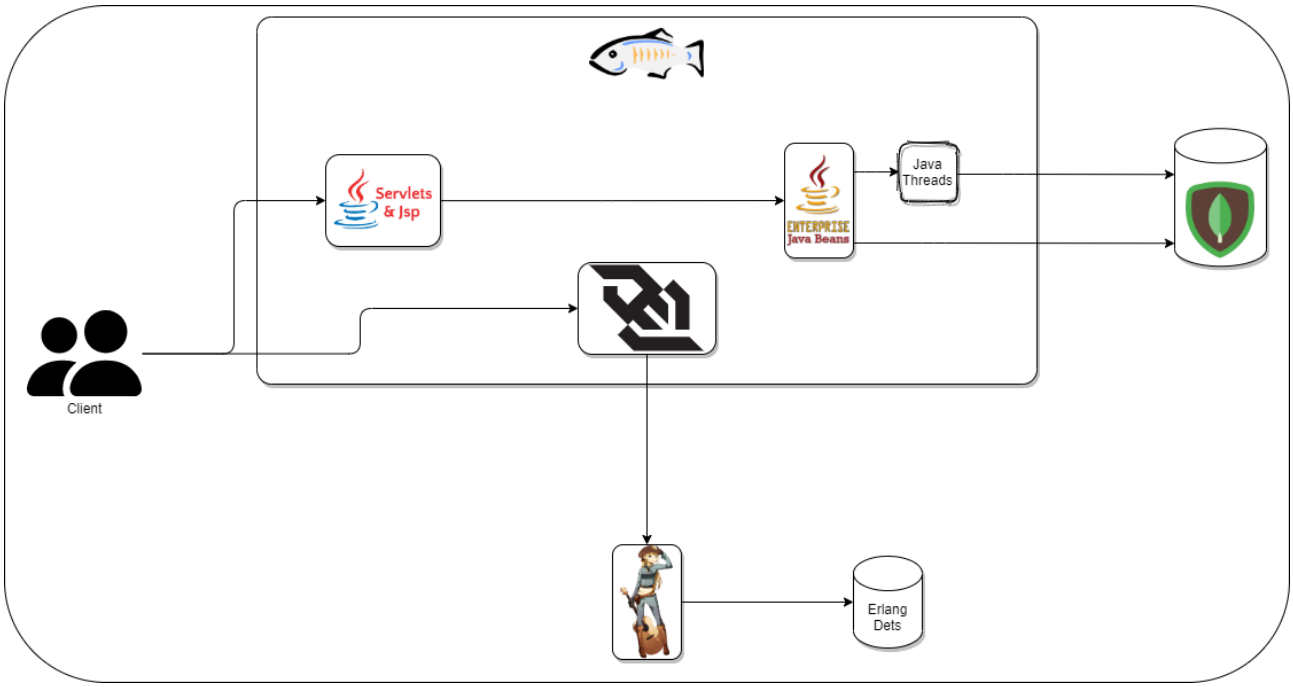
Always under the control of the restaurateur, access to the reserved area can also be given to staff with different privileges related to the job (e.g., waiter, cook, etc.) to speed up interaction between staff and improve the quality of service.

ASSUMPTIONS

To avoid spending time implementing superfluous functionalities we decided, instead, to focus on the topics discussed in class; the workload will be reduced by limiting us to the creation of a reduced version of the overall idea, in particular these assumptions will be considered:

- All the reserved area will not be considered
- Only one restaurant will be present, and the arrangement of the tables cannot be changed
- The order page for the user will be limited to only allow communications with the kitchen

INFRASTRUCTURE



APPLICATION SERVER

Having in mind to make massive use of java EE technologies, to develop PisaEat, GlassFish 6.0 was chosen as the Application Server that implements the specifications of Jakarta EE 9 (Java EE 9)

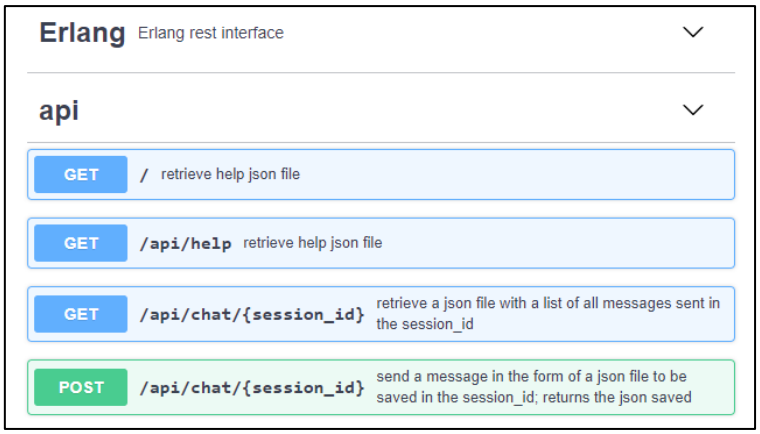
DATABASE

to guarantee the persistence of the information, it has been chosen MongoDB as document database which stores all information relating to the application (tables & booking sessions)

COWBOY REST SERVER

To save and retrieve all the messages sent in each booking session, a rest server was developed. This web server is based on cowboy an http server for Erlang/OTP. The message will be saved on a Dets table called “messages.dets”.

The overall A.P.I. provided is the following:



EXAMPLES

To save a message in the session with id “3” you should send a POST request like the following:

```
POST http://localhost:8081/api/chat/3 HTTP/1.1
Content-Type: application/json

{"username": "Tizio",
"message": "I’m intolerant to Gluten"}
```

The response message will look like this:

```
{
  "username": "Tizio",
  "message": "I'm intolerant to Gluten",
  "datetimeUTC": "2021-02-08T14:56:13Z",
  "bookSessionId": "3"
}
```

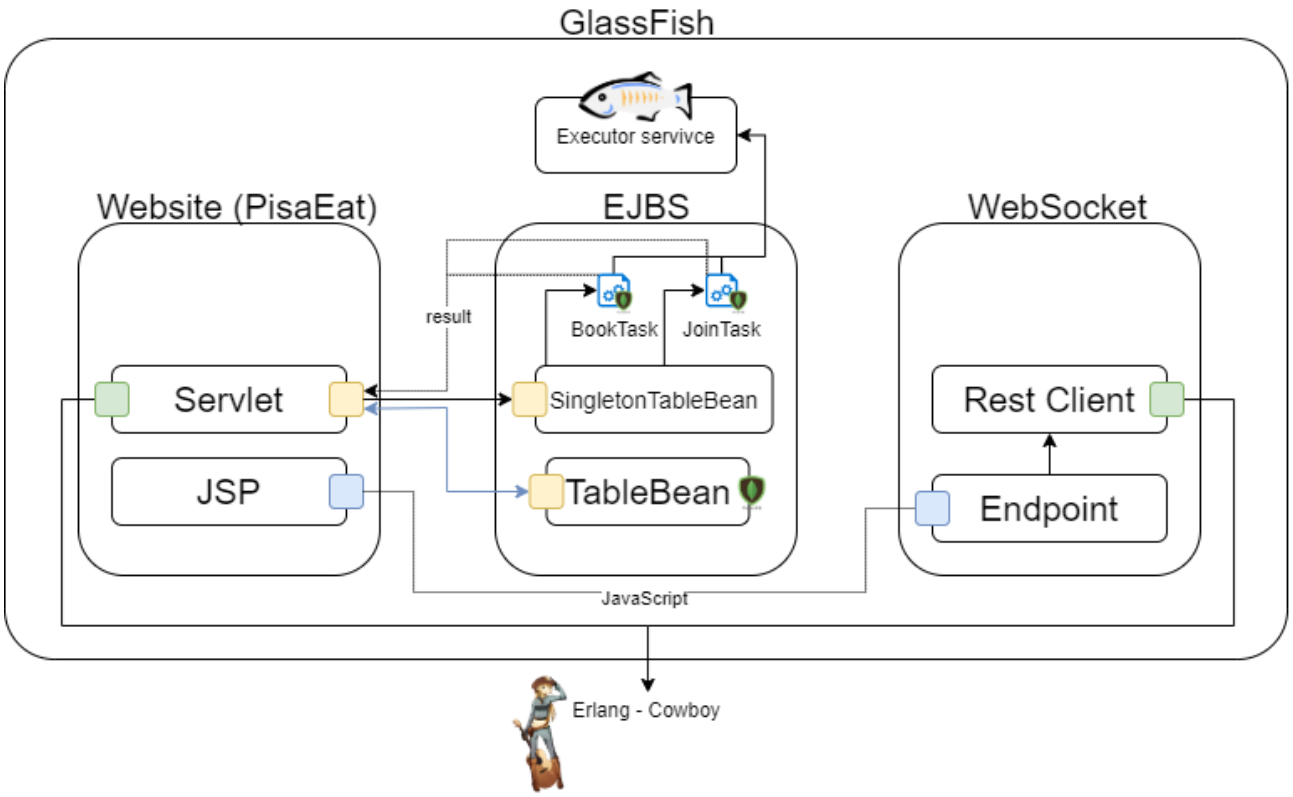
To retrieve all messages from session “3” you should send a GET request like the following:

```
GET http://localhost:8081/api/chat/3 HTTP/1.1
```

And the result will look like this:

```
{
    "list": [
      {
        "bookSessionId": "3",
        "username": "Tizio",
        "message": "I'm intolerant to Gluten",
        "datetimeUTC": "2021-02-08T14:56:13Z"
      },
      {
        "bookSessionId": "3",
        "username": "kitchen",
        "message": "Alright Tizio, I'll keep it in mind!",
        "datetimeUTC": "2021-02-08T15:09:08Z"
      }
    ]
  }
```

APPLICATION ARCHITECTURE



WEBSITE

Contains the presentation layer of the application. It contains the java servlet pages (jsp) with jstl and the servlets. Moreover, the servlets use the ejb-interfaces module, to perform rmi to complete the operations needed to perform the client request.

EJBS

There are two ejbs, one of type stateless, and one of type singleton.

Both ejbs implement a remote interface, one specific interface for the stateless ejb, and one for the singleton ejb. These two interfaces are placed in a module called ejb-interfaces, the same used by the website.

TABLEBEAN

This is the stateless ejb and it is used for non-critical operations, such as create a table or getting operation. It has been chosen a stateless ejb to maximize the performance.

SINGLETONTABLEBEAN

This is the singleton ejb and it is used for all the operations that must be thread safe such as booking a table or joining a table.

Here the schema is the same for both critical function:

- the singleton ejb obtains, by dependency injection, the default executor Service of glassfish, and through that it executes a task.

- Depending on the function that the caller wants to perform it spawns a task of *BookTableTask*, or *JoinBookSessionTask*, both classes implement the interface Callable, to let the task return an object.
- The spawned task is submitted to the executor service that returns a Future object which is immediately sent back to the caller to serve soon as possible another request.

This solution it is necessary because when using singleton ejb the lock is taken on the whole instance, creating a bottleneck

BOOKTABLETASK - JOINBOOKSESSIONTASK

Now inside the thread class, we have a Striped object; this is a class imported from the library guava, of google.

```
public class BookTableTask implements Callable<Table> {
   private static final Striped<Lock> tableLocks = Striped.lock(1024);
   ...

   @Override
   public Table call() throws Exception {
      //not synchronized code
      tableLocks.get(tableId).lock();

      try {
         //synchronized code

         tableLocks.get(tableId).unlock();
         return returnTable;
      }catch (Exception e){
         tableLocks.get(tableId).unlock();
         throw e;
      }

   }
}
```

With this class, we can create an array of Lock (or whatever synchronization structure) with a dimension, that must be a trade-off between memory usage and performance; this because the Striped, get a hash of the element that it receives, and the do a simple modulo (%) operation.

So, long story short, if we are trying to book two different table, we can get blocked even if the hash(tableId1) % Striped_DIM = hash(tableId2) % Striped_DIM.

In this way, this represent a trade-off, if we choose, for example 1 as the dimension, it is simple an always lock, even in the case we are booking two different table, on the other hand, a too large dimension, can result in wasting space. So, finally, the thread, take the lock, based on the id of the table, perform the operation, and then unlock.

WEBSOCKET

This is another module, it is used to implement the chat as it lets the server to notify clients, who already joined the table, about the arrival of a new message.

Basically, when a client opens a chat, a jQuery script opens a WebSocket with the server, and uses it to send and receive chat messages in real time.

The WebSocket, before sending the messages back to the other client, uses the ERWS (erlang web service), to persist the messages.

CRITIAL AREAS

SYNCHRONIZATION/COORDINATION

A critical area for synchronization/coordination is table reservations, since a user must have the mutual exclusion on the table that he wants to book (this also applies to joining a table)

PROPOSED SOLUTION

The SingletonTableBean is our component that handle this problem

COMMUNICATION

An area where communication is involved is in the order section, where users can communicate with the kitchen

PROPOSED SOLUTION

The WebSocket & ERLANG REST web service have been implemented to carry out the communications with the kitchen

ERLANG

To implement a part of the project in Erlang, the web service for communication with the kitchen has been implemented using Erlang