UNIVERSITÀ DI PISA

Department of Engineering

Real Time Information and Earthquake Notification services

Distributed Systems and Middleware Technology

Supervised by :
prof. Alessio Bechini

Authors :

Aida Himmiche

Bashar Hayani

Farzaneh Moghani

A.Y. 2021-2022

Contents

1 Introduction **1**

2 Web-Client **2**
2.1 Client Request Flow 3

3 Web-Servers **4**
3.1 Regional Server 4
3.2 Central Server 4

4 Implementation **5**
4.1 Erlang 5
4.2 EJBs: Regional Servers 7
4.3 Servlets 9
4.4 Web Application 9
4.5 Central Server 11

5 Deployment **11**

1 Introduction

Earthquake Tracker is an application which provides information about the latest earthquakes in various regions. The platform gives access to data about the magnitude, longitude, latitude, date and time of the earthquakes, provided by different seismic sensor networks. In addition to simple earthquake listing, a notification service is implemented in order to send alerts to users about high magnitude earthquakes in particular.

The network is organized in a hierarchy of two layers:

- One central server
- Three regional servers

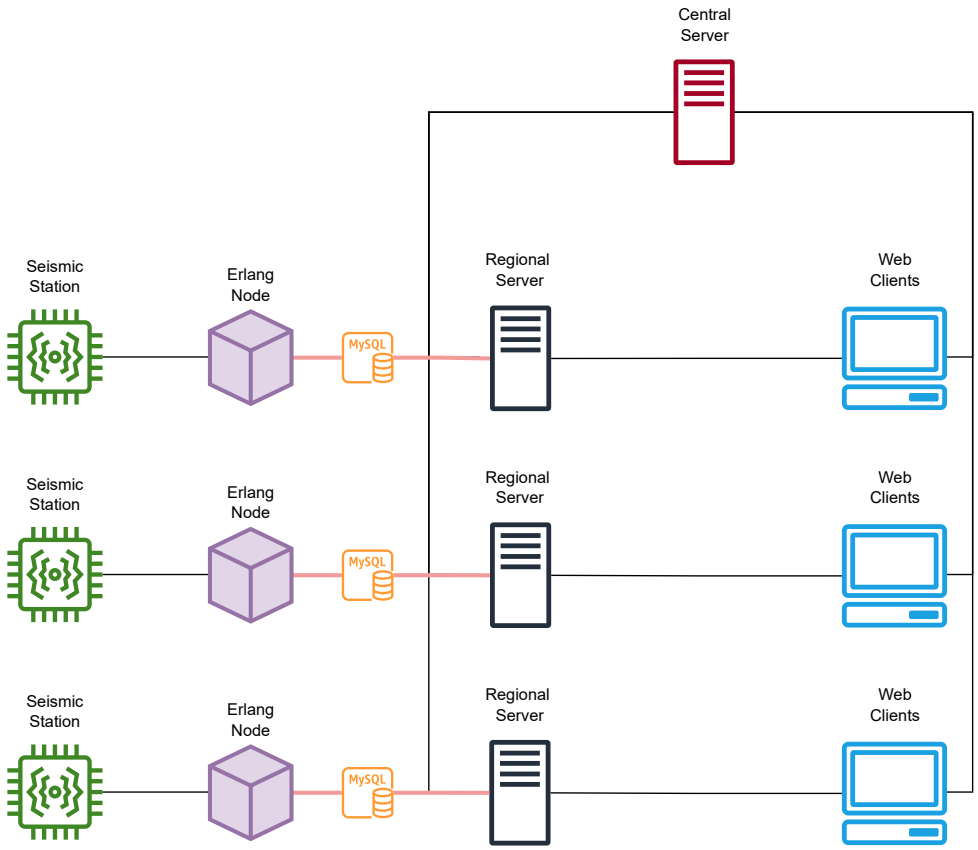
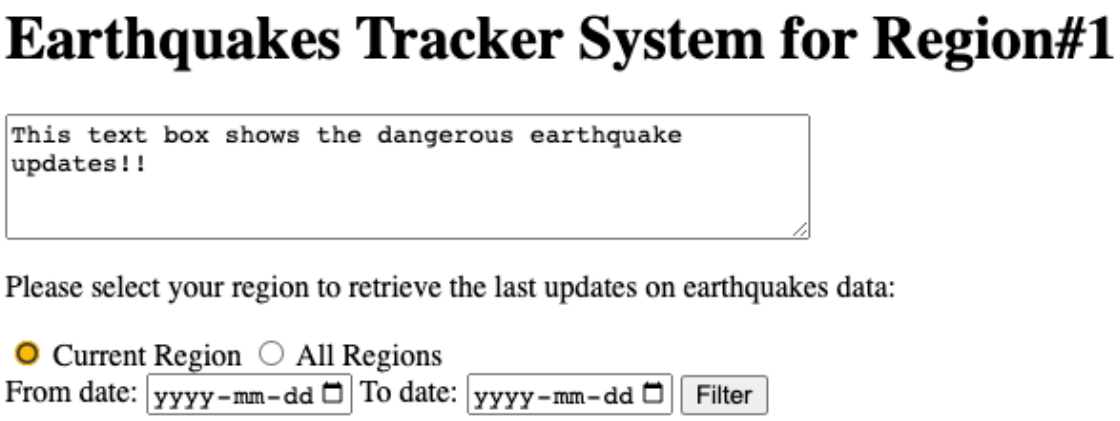


Figure 1: Architecture of the whole system

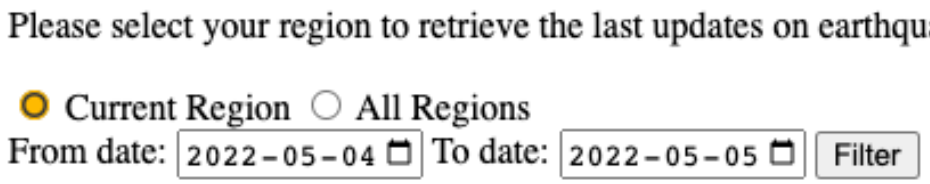
2 Web-Client

When using the web interface, the client will automatically be connected to one of the regional servers available based on their IP address. One regional server can handle many client connections at a time.



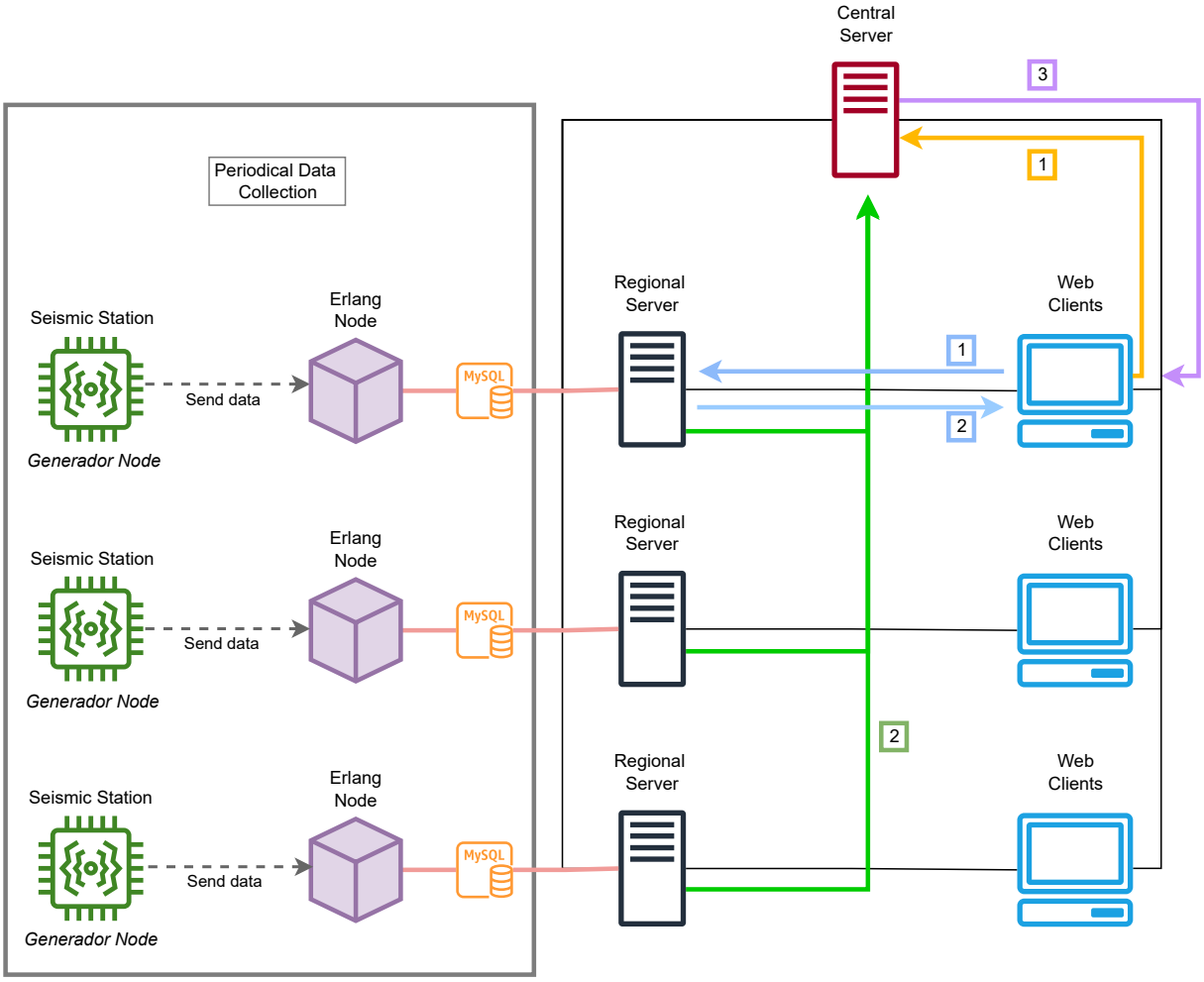
The web interface allows the user to perform the following actions:

- Choose the region they want to fetch the earthquake data from: The user can either chose to view the earthquakes in the region from where they are connected, or all the regions available in the system at once. The default option selected is "Regional Server" but it can be modified by clicking the preferred radio button.
- Choose a date range to filter results: The user can either provide both a start and end date for their query, or either one in case they want every earthquake **since** a certain date, or the earthquakes **up until** a certain date.



When querying earthquakes from all available regional servers, an additional column is introduced, specifying the region this listing was fetched from.

2.1 Client Request Flow



Querying data use case:

A user would like to retreive the list of earthquakes recorded in the "Current Region"

1 The web client sends a request to the regional server it is connected to

2 The regional server responds with the list of earthquakes from its satabase.

A user would like to retrieve the list of earthquakes recorded in all available regions

1 The client request is sent to the central server through the Web app which is hosted on the regional server.

2 The Central server collects all records in each regional database into one list

3 The central server responds to the request sending the complete list to the regional Web app which in turns respond to the Web client client

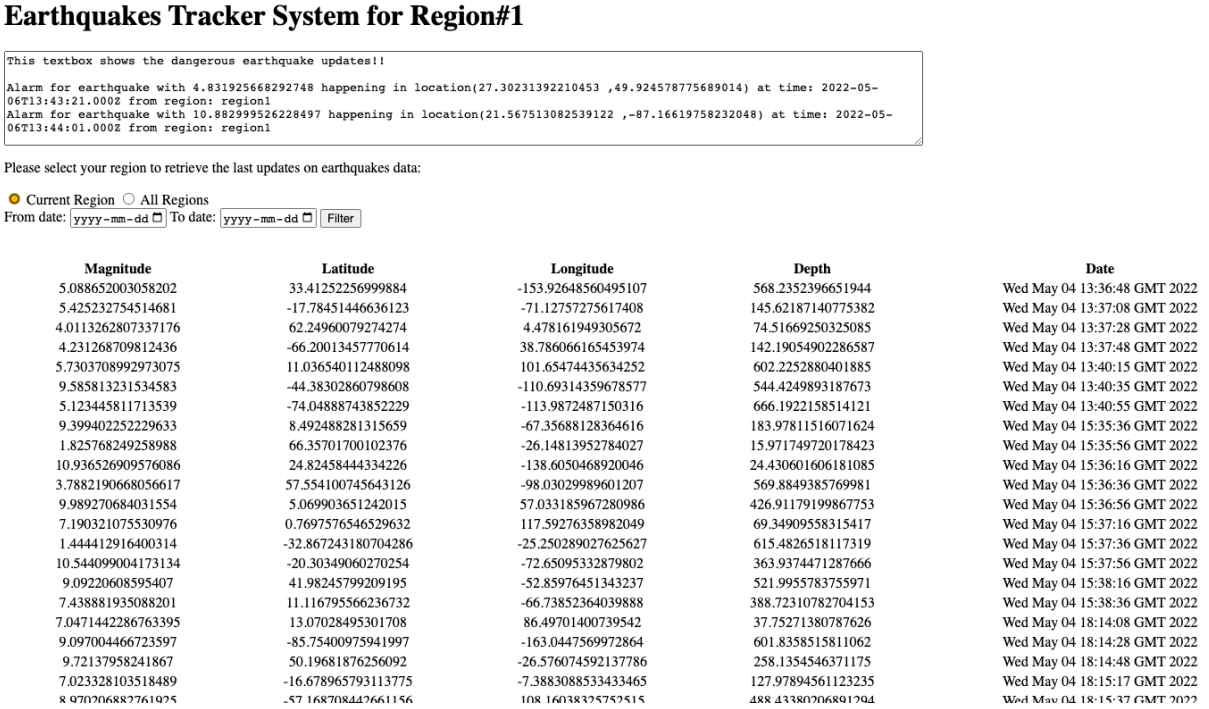
Figure 2: Client Request flow Diagram

3 Web-Servers

3.1 Regional Server

This category of servers is connected to MySQL databases that are tasked with the storage of earthquake information received from the designated seismic sensor stations. This server is used in two cases:

- If the user wants to retrieve a list of earthquakes in the region: The regional server receives all requests from the clients connected to it exclusively.
- When an earthquake of particularly high magnitude has been detected: in which case it sends a notification (alert message) to all connected clients in an alert box, including all information about said earthquake.



Earthquakes Tracker System for Region#1

```
This textbox shows the dangerous earthquake updates!!

Alarm for earthquake with 4.831925668292748 happening in location(27.30231392210453 ,49.924578775689014) at time: 2022-05-
06T13:43:21.000Z from region: region1
Alarm for earthquake with 10.882999526228497 happening in location(21.567513082539122 ,-87.16619758232048) at time: 2022-05-
06T13:44:01.000Z from region: region1
```

Please select your region to retrieve the last updates on earthquakes data:

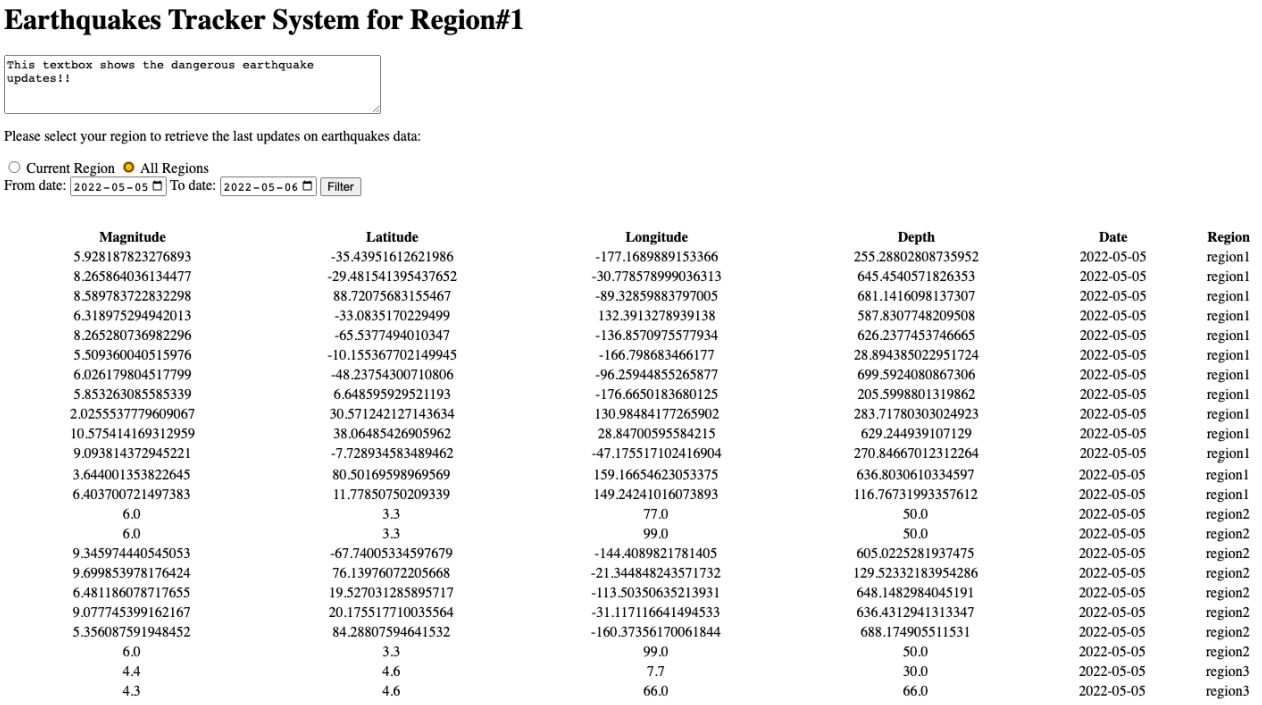
Current Region All Regions
From date: yyyy-mm-dd To date: yyyy-mm-dd Filter

Magnitude	Latitude	Longitude	Depth	Date
5.088652003058202	33.41252256999884	-153.92648560495107	568.2352396651944	Wed May 04 13:36:48 GMT 2022
5.425232754514681	-17.78451446636123	-71.12757275617408	145.62187140775382	Wed May 04 13:37:08 GMT 2022
4.0113262807337176	62.24960079274274	4.478161949305672	74.51669250325085	Wed May 04 13:37:28 GMT 2022
4.231268709812436	-66.20013457770614	38.786066165453974	142.19054902286587	Wed May 04 13:37:48 GMT 2022
5.7303708992973075	11.036540112488098	101.65474435634252	602.2252880401885	Wed May 04 13:40:15 GMT 2022
9.585813231534583	-44.38302860798608	-110.69314359678577	544.4249893187673	Wed May 04 13:40:35 GMT 2022
5.123445811713539	-74.04888743852229	-113.9872487150316	666.1922158514121	Wed May 04 13:40:55 GMT 2022
9.399402252229633	8.492488281315659	-67.35688128364616	183.97811516071624	Wed May 04 15:35:36 GMT 2022
1.825768249258988	66.35701700102376	-26.14813952784027	15.971749720178423	Wed May 04 15:35:56 GMT 2022
10.936526909576086	24.82458444334226	-138.6050468920046	24.430601606181085	Wed May 04 15:36:16 GMT 2022
3.7882190668056617	57.554100745643126	-98.03029989601207	569.8849385769981	Wed May 04 15:36:36 GMT 2022
9.989270684031554	5.069903651242015	57.033185967280986	426.91179199867753	Wed May 04 15:36:56 GMT 2022
7.190321075530976	0.7697576546529632	117.59276358982049	69.34909558315417	Wed May 04 15:37:16 GMT 2022
1.444412916400314	-32.867243180704286	-25.250289027625627	615.4826518117319	Wed May 04 15:37:36 GMT 2022
10.544099004173134	-20.30349060270254	-72.65095332879802	363.9374471287666	Wed May 04 15:37:56 GMT 2022
9.09220608595407	41.98245799209195	-52.85976451343237	521.9955783755971	Wed May 04 15:38:16 GMT 2022
7.438881935088201	11.116795566236732	-66.73852364039888	388.72310782704153	Wed May 04 15:38:36 GMT 2022
7.0471442286763395	13.07028495301708	86.49701400739542	37.75271380787626	Wed May 04 18:14:08 GMT 2022
9.097004466723597	-85.75400975941997	-163.0447569972864	601.8358515811062	Wed May 04 18:14:28 GMT 2022
9.72137958241867	50.19681876256092	-26.576074592137786	258.1354546371175	Wed May 04 18:14:48 GMT 2022
7.023328103518489	-16.678965793113775	-7.3883088533433465	127.97894561123235	Wed May 04 18:15:17 GMT 2022
8.970206882761925	-57.168708442661156	108.160383257525 15	488.433802068912 94	Wed May 04 18:15:37 GMT 2022

Figure 3: Earthquakes from one Region with Alerts

3.2 Central Server

This server is used if the user wants to retrieve earthquake information from all available regions: in which case it fetches a compiled list of all earthquakes stored in each of the regional Databases and sends them back to the client.



Earthquakes Tracker System for Region#1

This textbox shows the dangerous earthquake updates!!

Please select your region to retrieve the last updates on earthquakes data:

Current Region All Regions
From date: 2022-05-05 To date: 2022-05-06 Filter

Magnitude	Latitude	Longitude	Depth	Date	Region
5.928187823276893	-35.43951612621986	-177.1689889153366	255.28802808735952	2022-05-05	region1
8.265864036134477	-29.481541395437652	-30.778578999036313	645.4540571826353	2022-05-05	region1
8.589783722832298	88.72075683155467	-89.32859883797005	681.1416098137307	2022-05-05	region1
6.318975294942013	-33.0835170229499	132.3913278939138	587.8307748209508	2022-05-05	region1
8.265280736982296	-65.5377494010347	-136.8570975577934	626.2377453746665	2022-05-05	region1
5.509360040515976	-10.155367702149945	-166.798683466177	28.894385022951724	2022-05-05	region1
6.026179804517799	-48.23754300710806	-96.25944855265877	699.5924080867306	2022-05-05	region1
5.853263085585339	6.648595929521193	-176.6650183680125	205.5998801319862	2022-05-05	region1
2.0255537779609067	30.571242127143634	130.98484177265902	283.71780303024923	2022-05-05	region1
10.575414169312959	38.06485426905962	28.84700595584215	629.244939107129	2022-05-05	region1
9.093814372945221	-7.728934583489462	-47.175517102416904	270.84667012312264	2022-05-05	region1
3.644001353822645	80.50169598969569	159.16654623053375	636.8030610334597	2022-05-05	region1
6.403700721497383	11.77850750209339	149.24241016073893	116.76731993357612	2022-05-05	region1
6.0	3.3	77.0	50.0	2022-05-05	region2
6.0	3.3	99.0	50.0	2022-05-05	region2
9.345974440545053	-67.74005334597679	-144.4089821781405	605.0225281937475	2022-05-05	region2
9.699853978176424	76.13976072205668	-21.344848243571732	129.52332183954286	2022-05-05	region2
6.481186078717655	19.527031285895717	-113.50350635213931	648.1482984045191	2022-05-05	region2
9.077745399162167	20.175517710035564	-31.117116641494533	636.4312941313347	2022-05-05	region2
5.356087591948452	84.28807594641532	-160.37356170061844	688.174905511531	2022-05-05	region2
6.0	3.3	99.0	50.0	2022-05-05	region2
4.4	4.6	7.7	30.0	2022-05-05	region3
4.3	4.6	66.0	66.0	2022-05-05	region3

Figure 4: Earthquakes from all Region with Filter

4 Implementation

The application is hosted on a GlassFish server (version 5.1). In this application communication is Message Driven and exploits JMS.

4.1 Erlang

A portion purely in Erlang was written to simulate the activity of a seismic station sensor. This code creates earthquake records with information on the latitude, longitude, magnitude, depth and Date-time, with randomly generates numerical data.

The process sends one sample periodically every 40 seconds (non-blocking send). The receiving has been handled in java (blocking receive) with a timeout set to 30s.

```
periodic_readings_loop(ServerMailBox,ServerNodeName,Period) ->
  io:format("I am ~w", [self()]),
  random:seed(now()),
  Magnitude=1+(rand:uniform()*10),
  Latitude=scale(90,-90,rand:uniform()),
  Longitude=scale(180,-180,rand:uniform()),
  Depth=scale(0,700,rand:uniform()),
  TS = {_,_,Micro} = os:timestamp(),
{{Year,Month,Day},{Hour,Minute,Second}} = calendar:now_to_universal_time(TS),
{ServerMailBox,ServerNodeName} ! {{Magnitude,Latitude,Longitude,Depth,Year,Month,Day,Hour,Minute,Second}, self()},
  timer:sleep(Period),

  receive   %in case a stop has been received, no further executions are triggered.
    stop -> ok
  after 0 ->
    periodic_readings_loop(ServerMailBox, ServerNodeName,Period)
  end.
```

Figure 5: Erlang - Generating Earthquake Data

We deployed on each of the containers a Java executable file containing code which fetches data from Erlang nodes. When executed, it performs two tasks:

- Creating on the server itself another Erlang node which receives the data from the generator: The data "generator" node sends the data to the "main" Erlang node. Upon receiving the data this new node inserts the records into MySQL database on the regional server it is connected to.

```java
for(int i=0; i<numOfReceives;i++) {
    //while (true) {
    try {

        OtpErlangObject message = otpMbox.receive( i: 30000);

        if (message instanceof OtpErlangTuple) {
            OtpErlangTuple erlangTuple = (OtpErlangTuple) message;
            OtpErlangPid senderPID = (OtpErlangPid) erlangTuple.elementAt( i: 1);
            OtpErlangTuple earthquakeTuple = (OtpErlangTuple) erlangTuple.elementAt( i: 0);

            OtpErlangDouble magnitudeEr = (OtpErlangDouble) earthquakeTuple.elementAt( i: 0);
            OtpErlangDouble latitudeEr = (OtpErlangDouble) earthquakeTuple.elementAt( i: 1);
            OtpErlangDouble longitudeEr = (OtpErlangDouble) earthquakeTuple.elementAt( i: 2);
            OtpErlangDouble depthEr = (OtpErlangDouble) earthquakeTuple.elementAt( i: 3);

            double magnitude = magnitudeEr.doubleValue();
            double latitude = latitudeEr.doubleValue();
            double longitude = longitudeEr.doubleValue();
            double depth = depthEr.doubleValue();
            OtpErlangLong year = (OtpErlangLong) earthquakeTuple.elementAt( i: 4);
            OtpErlangLong month = (OtpErlangLong) earthquakeTuple.elementAt( i: 5);
            OtpErlangLong day = (OtpErlangLong) earthquakeTuple.elementAt( i: 6);
            OtpErlangLong hour = (OtpErlangLong) earthquakeTuple.elementAt( i: 7);
            OtpErlangLong minute = (OtpErlangLong) earthquakeTuple.elementAt( i: 8);
            OtpErlangLong second = (OtpErlangLong) earthquakeTuple.elementAt( i: 9);
```

Figure 6: Receiving data from the Erlang Generator Node

- The "main" Erlang node also checks for high magnitude earthquakes, for the sake of frequent results, we chose the magnitude threshold to be 2, but in a more concrete context it would be magnitudes of 4 and above. If the match is made and an earthquake record is found to have magnitude higher than 2, the "main" Erlang node writes the record into the GlassFish JMS queue.

When inserting data into the SQL databases we realized there might be some latency, so to deal with that we introduce the use of threads. A thread is created to insert the data.

4.2 EJBs: Regional Servers

The regional servers are hosted on the GlassFish server and each use an EJB responsible for collecting data from the regional server database, and returns the collected data with filtering options.

```
connection=dataSource.getConnection();
StringBuilder sqlStringBuilder=new StringBuilder();
sqlStringBuilder.append("select " );
sqlStringBuilder.append("  e.magnitude,  ");
sqlStringBuilder.append("  e.latitude,  ");
sqlStringBuilder.append("  e.longitude,  ");
sqlStringBuilder.append("  e.depth,  ");
sqlStringBuilder.append("  e.date  ");
sqlStringBuilder.append("  from earthquakedata e ");
sqlStringBuilder.append("  where 1 = 1  ");
if(startDate!=null) {
    sqlStringBuilder.append(" and e.date >= ? ");
    params.add(startDate);
}if (endDate !=null) {
    sqlStringBuilder.append(" and e.date <= ? ");
    params.add(endDate);
}
```

Figure 7: Retrieving data from the mySQL database

The EJB has one interface "EarthquakeRemoteEJB" and one implemented method "listEarthquakes()" which returns a Data Transfer Object (DTO) list of earthquakes collected.

The DTO class called "EarthquakeDTO" maps mySQL records to objects of this DTO class. The class's attributes match the columns of the MySQL records (magnitude, latitude, etc).

```
rs = pstm.executeQuery();
while (rs.next()) {
    EarthquakeDTO earthquakeDTO=new EarthquakeDTO();
    earthquakeDTO.setMagnitude(rs.getDouble( columnIndex: 1));
    earthquakeDTO.setLatitude(rs.getDouble( columnIndex: 2));
    earthquakeDTO.setLongitude(rs.getDouble( columnIndex: 3));
    earthquakeDTO.setDepth(rs.getDouble( columnIndex: 4));
    earthquakeDTO.setDate(new Date(rs.getTimestamp( columnIndex: 5).getTime()));
    earthquakeDTO.setRegion(region);
    earthquakeDTOS.add(earthquakeDTO);
}
```

Figure 8: Building a DTO object

4.3 Servlets

The application makes use of a servlet called "earthInfo" and it is included in the "Webapp-test" module. This servlet gets the request from the web interface, containing all parameters entered by the user (Region selection, date range) and uses the EJB methods to collect the data accordingly, then filters the records as requested.
"earthInfo" provides one other method to initialize a connection to the JMS Queue.

```
public void initializeConnection() throws NamingException, JMSException {
    Properties env = new Properties();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
            "org.apache.activemq.jndi.ActiveMQInitialContextFactory");
    env.put(Context.PROVIDER_URL, "tcp://localhost:8080");
    env.put("queue.queueSampleQueue","myQueue2");

    try{
        Context ic = new InitialContext();
        ConnectionFactory qcf = (ConnectionFactory)ic.lookup( name: "jms/__defaultConnectionFactory");
        Queue queue = (Queue)ic.lookup( name: "jmsmyQueue2");
        javax.jms.Connection qc = qcf.createConnection();
        Session qs = qc.createSession( b: false, Session.AUTO_ACKNOWLEDGE);
        MessageConsumer qprod = qs.createConsumer(queue);
        earthquakeConsumerBean asyncReceiver = new earthquakeConsumerBean();
        qprod.setMessageListener(asyncReceiver);
        qc.start();
    }
```

Figure 9: Initializing a connection to the JMS Queue

4.4 Web Application

The web application is hosted on the GlassFish server and makes use of 2 EJBs. The first EJB's mission is to collect data from the "local" regional server (the one the web client is connected to) if they select "Current Region" in the radio button section. The second one serves to collect data from all regional servers (going through the central server) if the client selects "All Regions" in the radio button section.

In any case, a connection with JMS queue "jmsmyQueue2" is established, and a new object with "onMessage" listener is created:

- With the Java class "MessageSocket", the app preserves a list of clients connected through the web sockets as a list of sessions, in order to forward the dangerous earthquakes upon receiving them from the JMS queue.

```java
@ServerEndpoint("/MessageSocket1")

public class MessageSocket {

    static List <Session> socketSessions = Collections.synchronizedList(new ArrayList<Session>());

    @OnOpen
    public void open (Session session) throws IOException {
        session.getBasicRemote().sendText( s: "This textbox shows the dangerous earthquake updates!!\n");
        socketSessions.add(session);
    }

    @OnClose
    public void close (Session session) { socketSessions.remove(session); }

    public static void broadcast (String message) throws IOException {
        for (Session session: socketSessions){
            session.getBasicRemote().sendText(message);
        }
    }
```

Figure 10: Message Socket Class

```java
public class earthquakeConsumerBean implements MessageListener {


    @Override
    public void onMessage(Message msg) {
        System.out.println("On   Message event");
        if (msg instanceof TextMessage) {
            try {
                String text = ((TextMessage) msg).getText();
                String outMsg = "Received: " + text;
                String bar = "++++++++++++++++++++++++++++++++++++++++++++";
                outMsg = "\n" +bar +"\n" + outMsg + "\n" + bar +"\n";
                System.out.println(outMsg);
                MessageSocket.broadcast(text);
            } catch (final JMSException | IOException e) {
                throw new RuntimeException(e);
            }
        }
```

Figure 11: Message Listener Class

- JMS Queue (namely "jmsmyQueue2" in this project) triggers an "onMessage" event in the web app. The web app forwards the received message to all sessions connected through the web socket. The client in the JSP file has a listener for "onMessage" event which writes the received data into the text box.

```
<html>
<head>
    <title>Hi ...</title>
    <script type = text/javascript>
        var ws;
        var host = document.location.host;
        var pathname = document.location.pathname;

        const url = "wss://" + host  + "/webapp-test/MessageSocket1";
        ws = new WebSocket(url);

        ws.onmessage = function(event) {
            console.log(event.data);
        let textarea = document.getElementById("AlertBox");
        textarea.value+= event.data +"\n";
        };
    </script>
</head>
```

Figure 12: JSP on Message Listener

4.5 Central Server

This server is hosted on the GlassFish server (version 5.1) To implement the central server, we used an EJB which makes an SQL connection to three databases in 3 "Regional Server" containers, and returns a list of compiled rows. This is done through the EJB interface containing the method "collectServersData()" it returns a DTO list of earthquakes.

5 Deployment

This application was first implemented on a local machine, then deployed onto virtual containers provided by the Information Engineering Department. We made use of 4 containers, one for each of the 3 Regional servers, and another for the central server.

- The following software was added to the containers: GlassFish 5.1, Mysql, Erlang.
- GlassFish JDBC Resources were configured for 3 "regional server" containers and JMS Resources for the "central server" container.
- A MySQL database was created on each of the "regional server" containers.

- The executable JAR file where the Erlang potions were written was exported to all "regional server" containers. Upon execution, the following prompts are displayed:

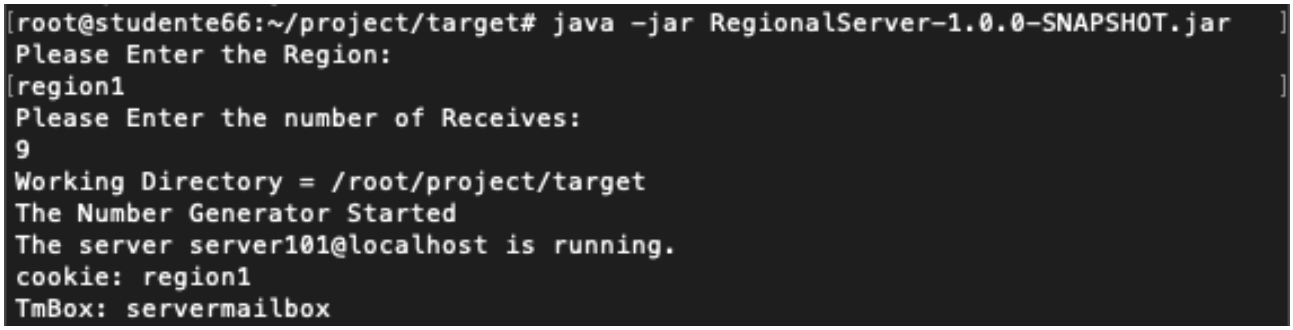


Figure 13: Erlang Jar Executable