

CS 7340 Team 1 Lab 2

API documents

Screenshot of routes file in backend:

The screenshot shows the IntelliJ IDEA interface with the 'Lab-2-backend' project open. The 'routes' file is selected in the editor tab bar. The code in the editor is as follows:

```
1 # Routes
2 # This file defines all application routes (Higher priority routes first)
3 #
4
5 # An example controller showing a sample home page
6 GET / controllers.HomeController.index
7
8 # An example controller showing a sample home page
9 GET /findAll controllers.PubController.findAll()
10
11 # An example controller showing a sample home page
12 POST /query1Response controllers.PubController.partOneQueryOne()
13
14 POST /query2Response controllers.PubController.partOneQueryTwo()
15
16 POST /query3Response controllers.PubController.partOneQueryThree()
17
18 POST /query4Response controllers.PubController.partOneQueryFour()
19
20 POST /query5Response controllers.PubController.partOneQueryFive()
21
22 POST /query21Response controllers.PubController.partTwoQueryOne()
23
24 POST /query22Response controllers.PubController.partTwoQueryTwo()
25
26 POST /query23Response controllers.PubController.partTwoQueryThree()
27
28
29
30
31
32
33
34
```

The project structure on the left shows files like application.conf, ReadMe.txt, Conference.java, pub_info.java, and various controller and model classes. The 'test' directory is also visible.

Screenshot of routes file in frontend:

The screenshot shows the IntelliJ IDEA interface with the 'Lab-2-frontend' project open. The 'routes' file is selected in the editor tab bar. The code in the editor is as follows:

```
1 # Routes
2 # This file defines all application routes (Higher priority routes first)
3 #
4
5 # An example controller showing a sample home page
6 GET / controllers.HomeController.querySelectionHandler()
7
8 GET /query1 controllers.HomeController.query1()
9
10 GET /query2 controllers.HomeController.query2()
11
12 GET /query3 controllers.HomeController.query3()
13
14 GET /query4 controllers.HomeController.query4()
15
16 GET /query5 controllers.HomeController.query5()
17
18 GET /query21 controllers.HomeController.query21()
19
20 GET /query22 controllers.HomeController.query22()
21
22 GET /query23 controllers.HomeController.query23()
23
24 GET /querySelection controllers.HomeController.querySelectionHandler()
25
26 GET /query1Response controllers.HomeController.queryOneHandler()
27
28 GET /query2Response controllers.HomeController.queryTwoHandler()
29
30 GET /query3Response controllers.HomeController.queryThreeHandler()
31
32 GET /query4Response controllers.HomeController.queryFourHandler()
33
34 GET /query5Response controllers.HomeController.queryFiveHandler()
35
36 GET /query21Response controllers.HomeController.queryTwoOneHandler()
37
38
```

The project structure on the left shows files like Publication.java, Conference.java, HomeController.java, and various controller and model classes. The 'test' directory is also visible.

Lab-2-frontend / conf / routes

```

Project Lab-2-frontend [client] ~/Desktop/Lab2_test/submit/CS573
  app
    controllers
      Conference
      HomeController
      Publication
    conf
      application.conf
      logback.xml
      routes
    logs
    project [client-build] sources root
    public
    target
      _DS_Store
      build.sbt
  External Libraries
  Scratches and Consoles

16
17 GET /query5 controllers.HomeController.query5()
18
19 GET /query21 controllers.HomeController.query21()
20
21 GET /query22 controllers.HomeController.query22()
22
23 GET /query23 controllers.HomeController.query23()
24
25 GET /querySelection controllers.HomeController.querySelectionHandler()
26
27 GET /query1Response controllers.HomeController.queryOneHandler()
28
29 GET /query2Response controllers.HomeController.queryTwoHandler()
30
31 GET /query3Response controllers.HomeController.queryThreeHandler()
32
33 GET /query4Response controllers.HomeController.queryFourHandler()
34
35 GET /query5Response controllers.HomeController.queryFiveHandler()
36
37 GET /query21Response controllers.HomeController.queryTwoOneHandler()
38
39 GET /query22Response controllers.HomeController.queryTwoTwoHandler()
40
41 GET /query23Response controllers.HomeController.queryTwoThreeHandler()
42
43
44
45
46 # Map static resources from the /public folder to the /assets URL path
47 GET /assets/*file controllers.Assets.at(path="/public", file)
48

```

Event Log

Info: sbt compilation for play framework 2.x disabled by default (6 minutes ago)

Screenshot of some functions in frontend relate to each query:

Query 1.1 to 1.4 and 2.1 to 2.2:

Lab-2-frontend / app / controllers / Publication

```

Project Lab-2-frontend [client] ~/Desktop/Lab2_test/submit/CS573
  app
    controllers
      Conference
      HomeController
      Publication
    conf
      application.conf
      logback.xml
      routes
    logs
    project [client-build] sources root
    public
    target
      _DS_Store
      build.sbt
  External Libraries
  Scratches and Consoles

176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214

public CompletionStage<WSResponse> checkAuthorizedQ1() {
    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    WSRequest request = ws.url("http://localhost:9005/query1Response");
    ObjectNode res = Json.newObject();
    res.put( "fieldName", "title" );
    res.put( "value", "this.title" );
    return request.addHeader( name: "Content-Type", value: "application/json" )
        .post(res)
        .thenApply((WSResponse r) -> {
            return r;
        });
}

public CompletionStage<WSResponse> checkAuthorizedQ2() {
    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    WSRequest request = ws.url("http://localhost:9005/query2Response");
    ObjectNode res = Json.newObject();
    res.put( "fieldName", "journal" );
    res.put( "value", "this.journal" );
    res.put( "fieldName", "volume" );
    res.put( "value", "this.volume" );
    res.put( "fieldName", "pub_number" );
    res.put( "value", "this.pub_number" );
    System.out.println(res);
    return request.addHeader( name: "Content-Type", value: "application/json" )
        .post(res)
        .thenApply((WSResponse r) -> {
            return r;
        });
}

public CompletionStage<WSResponse> checkAuthorizedQ3() {
    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    WSRequest request = ws.url("http://localhost:9005/query3Response");
    ObjectNode res = Json.newObject();
    res.put( "fieldName", "author" );
    res.put( "value", "this.author" );
    res.put( "fieldName", "pub_year" );
    res.put( "value", "this.pub_year" );
    System.out.println(res);
}

```

Event Log

Info: sbt compilation for play framework 2.x disabled by default (8 minutes ago)

Lab-2-frontend app controllers Publication

```

214
215     System.out.println(res);
216
217     return request.addHeader( name: "Content-Type", value: "application/json")
218         .post(res)
219         .thenApply((WSResponse r) -> {
220             return r;
221         });
222
223 }
224
225 public CompletionStage<WSResponse> checkAuthorizedQ4() {
226
227     WSClient ws = play.test.WSTestClient.newClient( port: 9005);
228     WSRequest request = ws.url("http://localhost:9005/query4Response");
229     ObjectNode res = Json.newObject();
230     res.put( fieldName: "Title", this.title);
231
232     return request.addHeader( name: "Content-Type", value: "application/json")
233         .post(res)
234         .thenApply((WSResponse r) -> {
235             return r;
236         });
237
238 public CompletionStage<WSResponse> checkAuthorizedQ21() {
239
240     WSClient ws = play.test.WSTestClient.newClient( port: 9005);
241     WSRequest request = ws.url("http://localhost:9005/query21Response");
242     ObjectNode res = Json.newObject();
243     res.put( fieldName: "author", this.author);
244     res.put( fieldName: "pub_year", this.pub_year);
245     System.out.println(res);
246
247     return request.addHeader( name: "Content-Type", value: "application/json")
248         .post(res)
249         .thenApply((WSResponse r) -> {
250             return r;
251         });
252
253 public CompletionStage<WSResponse> checkAuthorizedQ21() {
254
255     WSClient ws = play.test.WSTestClient.newClient( port: 9005);
256     WSRequest request = ws.url("http://localhost:9005/query21Response");
257     ObjectNode res = Json.newObject();
258
259     return request.addHeader( name: "Content-Type", value: "application/json")
260         .post(res)
261         .thenApply((WSResponse r) -> {
262             return r;
263         });
264
265
266
267
268 }

```

Event Log

Info: sbt compilation for play framework 2.x disabled by default (8 minutes ago)

12:14 LF UTF-8 4 spaces

Query 1.5 and 2.3:

```

public void setY (double y) { this.y = y; }

public CompletionStage<WSResponse> checkAuthorizedQ5() {
    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    WSRequest request = ws.url("http://localhost:9005/query5Response");
    ObjectNode res = Json.newObject();
    res.put( fieldName: "name", this.name);
    res.put( fieldName: "year", this.year);
    System.out.println(res);

    return request.addHeader( name: "Content-Type", value: "application/json")
        .post(res)
        .thenApply((WSResponse r) -> {
            return r;
        });
}

public CompletionStage<WSResponse> checkAuthorizedQ23() {
    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    WSRequest request = ws.url("http://localhost:9005/query23Response");
    ObjectNode res = Json.newObject();
    res.put( fieldName: "name", this.name);
    res.put( fieldName: "yearFrom", this.yearFrom);
    res.put( fieldName: "yearTo", this.yearTo);

    return request.addHeader( name: "Content-Type", value: "application/json")
        .post(res)
        .thenApply((WSResponse r) -> {
            return r;
        });
}

```

Query 1.1 – 1.5:

frontend => publication.java:

```

177     public CompletionStage<WSResponse> checkAuthorizedQ1() {
178         WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
179         WSRequest request = ws.url("http://localhost:9005/query1Response");
180         ObjectNode res = Json.newObject();
181         res.put( fieldName: "title", this.title);

182         return request.addHeader( name: "Content-Type", value: "application/json")
183             .post(res)
184             .thenApply((WSResponse r) -> {
185                 return r;
186             });
187     }
188

```

frontend => HomeController.java:

```

49     public CompletionStage<Result> queryOneHandler() {
50         Form<Publication> publicationForm = formFactory.form(Publication.class).bindFromRequest();
51         if (publicationForm.hasErrors()){
52             String authorizeMessage = "Invalid publication Title";
53             List<String> wrongMessage = new ArrayList<>();
54             wrongMessage.get(0).add(authorizeMessage);
55             return (CompletionStage<Result>) badRequest(views.html.query1.render(wrongMessage));
56         }
57
58         return publicationForm.get().checkAuthorizedQ1().thenApplyAsync((WSResponse r) -> {
59             System.out.println("Body: "+ r.getBody());
60             if (r.getStatus() == 200 && r.asJson() != null && !r.getBody().equals("false")) {
61                 JsonNode res = r.asJson();

```

Backend => PubController.java

```

public Result partOneQueryOne() {
    System.out.println("In PartOneQueryOne");
    JsonNode req = request().body().asJson();
    String title = req.get("title").asText();
    System.out.println("receive title: "+title);
    try {
        List<ObjectNode> query1_1=getMetadataUsingPaperName(title);
        if (title.equals("")){
            return ok(content: "false");
        } else return ok(query1_1.toString());
    }
    catch (Exception e) {
        return ok(content: "false");
    }
}

public Result partOneQueryTwo() {
}

```

Backend => pub_info.java

```

public static List<SqlRow> findByTitle_1_1(String title) {
    try{
        List<SqlRow> query1_1 = Ebean.createSqlQuery("select * from pub_info where title like '"+replacePunctuation
                +"%"+title+"%'"
                .limit(1)
                .findList();
        return query1_1;
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}

```

For frontend, we receive a form from front end webpage, then save the frontend information into a JSON object “res” . Then send a post request from front end to back end by using an API url: [http://localhost:9005/query\(1-5\)Response](http://localhost:9005/query(1-5)Response) which is the base route url and our Json Object is our passing parameter.

For backend, we receive the request from frontend, we parse the API url. We get the JsonNode by using function request().body().asJson();

Then we call another function findByTitle_(1_1 – 1_4) in model.pub_info or findByConference in model.Conference to search result in our mysql Database.

We save the answer as String and use post request to send our data from backend to frontend.

When the front end receives the result from backend, then we translate the result from String to Json Object, then parse the Json Object and save the result into a List<List<String>> res and pass the res into front webpage.

```
return publicationForm.get().checkAuthorizedQ23().thenApplyAsync((WSResponse r) -> {
    if (r.getStatus() == 200 && r.asJson() != null && !r.getBody().equals("null")) {
        JsonNode res = r.asJson();
        session("name", publicationForm.get().getName());
        session("yearFrom", publicationForm.get().getYear());
        session("yearTo", publicationForm.get().getYear());

        System.out.println("The metadata of conference you are looking for:");
        List<Conference> confList = new ArrayList<>();
        for (int i = 0; i < res.size(); i++) {
            Conference curConf = new Conference();
            JsonNode row = res.get(i);
            curConf.setName(row.get("name").asText());
            curConf.setYear(row.get("year").asText());
            curConf.setLocation(row.get("location").asText());
            curConf.setX(row.get("x").asDouble());
            curConf.setY(row.get("y").asDouble());
            confList.add(curConf);
        }
        return ok(views.html.response.render(confList));
    }
}
```

For query 1.5, the send API from front to back end is the same as before 4 queries. But when we receive the result from backend, we translate it as a Json Object, then we save the information into a List<Conference> to save all the information about the conference location and other information (coordinate). Then show the result in the query5.scala.html

Query 2.1-2.3:

Frontend:

GET	/query21	controllers.HomeController.query21()
GET	/query22	controllers.HomeController.query22()
GET	/query23	controllers.HomeController.query23()
GET	/query21Response	controllers.HomeController.queryTwoOneHandler()
GET	/query22Response	controllers.HomeController.queryTwoTwoHandler()
GET	/query23Response	controllers.HomeController.queryTwoThreeHandler()

```

public Result query21() {
    return ok(views.html.query21.render(new ArrayList<>()));
}

public Result query22() {
    return ok(views.html.query22.render(new ArrayList<>()));
}

public Result query23() {
    return ok(views.html.query23.render(new ArrayList<>()));
}

```

Query 2.1:

From “query21” “queryTwoOneHandler” and “CheckAuthorizedQ21” , we are able to send the Json Values of “author” and “pub_year” from frontend to backend with OK object. Also, using them with “GET” in routes, the frontend can receive the “WSResponse” type of objects which store all the metadata for specific publication. We can change the “ WSResponse” objects into Json objects.

Moreover, the API url from backend is “<http://localhost:9005/query21Response>” .

```

public CompletionStage<Result> queryTwoOneHandler() {
    Form<Publication> publicationForm = formFactory.form(Publication.class).bindFromRequest();
    if (publicationForm.hasErrors()){
        String authorizeMessage = "Form has errors";
        List<List<String>> wrongMessage = new ArrayList<>();
        List<String> curMess = new ArrayList<>();
        curMess.add(authorizeMessage);
        wrongMessage.add(curMess);
        return (CompletionStage<Result>) badRequest(views.html.query21.render(wrongMessage));
    }

    return publicationForm.get().checkAuthorizedQ21().thenApplyAsync((WSResponse r) -> {
        if (r.getStatus() == 200 && r.asJson() != null && !r.getBody().equals("null")) {
            JsonNode res = r.asJson();
            List<List<String>> res2_1 = new ArrayList<>();
            String query2_1 = new String();
            // add author and pub_year to session
            session("author", publicationForm.get().getAuthor());
            session("pub_year", publicationForm.get().getPub_year());
            System.out.println(res);
            List<String> curList = new ArrayList<>();
            for (int i = 0; i < res.size(); i++) {
                JsonNode row = res.get(i);
                List<String> curNode = new ArrayList<>();
                //System.out.println(row.findValue("pid").asText());
                curNode.add(""+row.findValue("title"));
                curNode.add(""+row.findValue("mdate"));
                curNode.add(""+row.findValue("author list"));
            }
        }
    });
}

```

```

public CompletionStage<WSResponse> checkAuthorizedQ21() {

    WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
    //add Title
    WSRequest request = ws.url("http://localhost:9005/query21Response");
    ObjectNode res = Json.newObject();
    res.put( fieldName: "author", this.author);
    res.put( fieldName: "pub_year", this.pub_year);
    System.out.println(res);

    return request.addHeader( name: "Content-Type", value: "application/json")
        .post(res)
        .thenApply((WSResponse r) -> {
            return r;
        });
}

```

Query 2.2:

From “query22” “queryTwoTwoHandler” and “CheckAuthorizedQ22”, we are able to send the execution command from frontend to backend with OK object. Also, using them with “GET” in routes, the frontend can receive the “WSResponse” type of objects which store all the co-authors for specific authors. We can change the “WSResponse” objects into Json objects.

Moreover, the API url from backend is http://localhost:9005/query22Response.

```

public CompletionStage<Result> queryTwoTwoHandler() {
    Form<Publication> publicationForm = formFactory.form(Publication.class).bindFromRequest();
    if (publicationForm.hasErrors()){
        List<List<String>> wrongList = new ArrayList<>();
        wrongList.get(0).add("Author form has errors");
        return (CompletionStage<Result>) badRequest(views.html.query22.render(wrongList)); // 
    }

    return publicationForm.get().checkAuthorizedQ22().thenApplyAsync((WSResponse r) -> {
        if (r.getStatus() == 200 && r.asJson() != null && !r.getBody().equals("null")) {
            JsonNode res = r.asJson();
        }
    });

    public CompletionStage<WSResponse> checkAuthorizedQ22() {

        WSClient ws = play.test.WSTestClient.newClient( port: 9005 );
        //add Title
        WSRequest request = ws.url("http://localhost:9005/query22Response");
        ObjectNode res = Json.newObject();
        System.out.println(res);

        return request.addHeader( name: "Content-Type", value: "application/json")
            .post(res)
            .thenApply((WSResponse r) -> {
                return r;
            });
    }
}

```

Backend:

```

POST      /query21Response           controllers.PubController.partTwoQueryOne()
POST      /query22Response           controllers.PubController.partTwoQueryTwo()
POST      /query23Response           controllers.PubController.partTwoQueryThree()

```

Query 2.1

With routes and request in “partTwoQueryOne” to get the JsonNode of “author” and “pub_year” from frontend.

```

public Result partTwoQueryOne(){
    System.out.println("In PartTwoQueryOne");
    JsonNode req = request().body().asJson();
    String author = req.get("author").asText();
    System.out.println(req);
    int pub_year = req.get("pub_year").asInt();
    System.out.println("receive researcher name = "+author);
    System.out.println("pub_year = "+pub_year);
}

```

Then we use these two functions below as API to handle the inputs. Firstly, it gets author and pub_year and outputs the String List of titles. Then, it uses titles to output list of ObjectNode which contains the metadata of specific publication.

And the URL is “<http://localhost:9005/query21Response>” .

```

301   public List<ObjectNode> getTitlesUsingAuthorAndYear (String author, int pub_year) {
302     List<ObjectNode> reslist = new ArrayList<>();
303     try {
304       List<SqlRow> pub = pub_info.findByTitle_1_3(author, pub_year);
305       System.out.println(pub.get(0)!= null);
306       if (pub.get(0)!= null) {
307         int i=0;
308         ...
309       }
310     }
311
312   public List<ObjectNode> getMetadataUsingPaperName(String title) {
313     List<ObjectNode> reslist= new ArrayList<>();
314     try {
315       List<SqlRow> pub = pub_info.findByTitle_1_1(title);
316
317       if (pub.size()!= 0) {
318         for (SqlRow pubs : pub) {
319           ObjectNode res = Json.newObject();
320
321         }
322       }
323     }
324   }

```

Finally, we use ok to “POST” the ObjectNode list to frontend.

```

        System.out.println("result: "+query2_1_metadata);
        return ok(query2_1_metadata.toString());
      }

    catch (Exception e) {
      return ok(content: "false");
    }
  
```

Query 2:

With routes and “partTwoQueryTwo” to get the execution command from frontend.
Then we use the function below as API to get the productive author by using ObjectNode list.

```
/*
public List<ObjectNode> getAuthorsWhosePaperMoreThanTen() {
    List<ObjectNode> reslist = new ArrayList<>();
    try {
        List<SqlRow> pub = pub_info.findByAuthor_1_4();
        System.out.println(pub.get(0)!= null);

        if (pub.get(0)!= null) {
            int i=0;
            for (SqlRow pubs : pub) {
                i++;
                ObjectNode res = Json.newObject();
                res.put( fieldName: "author", (String) pubs.get("author"));

                System.out.println(res);

                reslist.add(res);
            }
        }
    }
}
```

Next, we get the List of data and apply them into function below. Then we can get the coauthor list with ObjectNode type.

```
/*
public List<ObjectNode> getCoauthorForProductiveAuthor(String name) {
    List<ObjectNode> reslist = new ArrayList<>();
    try {
        List<SqlRow> pub = pub_info.findCoAuthor_2_2(name);
        System.out.println(pub.get(0)!= null);

        if (pub.get(0)!= null) {
            int i=0;
            for (SqlRow pubs : pub)
                ObjectNode res = Json.newObject();
                res.put( fieldName: "author", (String) pubs.get("author"));


```

Finally, we use OK and routes to send the data back to the frontend. And the URL is
<http://localhost:9005/query22Response>.

```

public Result partTwoQueryTwo(){
    System.out.println("In PartTwoQueryTwo");

    try {
        List<SqlRow> pub = pub_info.findByAuthor_1_4();
        System.out.println(pub.get(0)!= null);
        List<ObjectNode> query2_2_Coauthor=new ArrayList<>();
        List<ObjectNode> query2_2_Author = getAuthorsWhosePaperMoreThanTen();
        for(int i=0;i<query2_2_Author.size();i++){
            List<ObjectNode> temp=new ArrayList<>();
            String author=query2_2_Author.get(i).findValue( fieldName: "author").asText();
            temp=getCoauthorForProductiveAuthor(author);
            List<String> coauthor=new ArrayList<>();
            for(int k=0;k<temp.size();k++) {
                coauthor.add(temp.get(k).findValue( fieldName: "author").asText());
            }
            List<ObjectNode> temp=new ArrayList<>();
            String author=query2_2_Author.get(i).findValue( fieldName: "author").asText();
            temp=getCoauthorForProductiveAuthor(author);
            List<String> coauthor=new ArrayList<>();
            for(int k=0;k<temp.size();k++) {
                coauthor.add(temp.get(k).findValue( fieldName: "author").asText());
            }
            ObjectNode authorWithCoauthor=Json.newObject();
            authorWithCoauthor.put( fieldName: "author",author);
            authorWithCoauthor.put( fieldName: "coauthor",coauthor.toString());
            query2_2_Coauthor.add(authorWithCoauthor);
        }

        System.out.println("result: "+query2_2_Coauthor);

        return ok(query2_2_Coauthor.toString());
    }
    catch (Exception e) {
        return ok( content: "false");
    }
}

```

Query 2.3:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
      integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLV9ocqUKLsCC5CXdbqCmbLAsh0MAS6/keqq/sMZMZ19scR4PsZChSR7A==" 
      crossorigin="" />
    <style>
      #mapid {
        height: 600px;
      }
    </style>
    <!-- Make sure you put this AFTER Leaflet's CSS -->
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"
      integrity="sha512-XQoYMqMTK8LydxXYG3nZ448h0EQiglfqkJs1NOQV44cWnUrBc8PkA0cXy20w0vlaXaVUearIOBhiXZ5V3ynxwA==" 
      crossorigin="">
    </script>
  </head>
  <body>
    <div id="mapid"></div>
  </body>
</html>

```



The screenshot shows a browser window displaying a map. A single red marker is placed on the map at coordinates [51.7, -0.89]. A small blue info box (popup) is open at this location, containing the text "I am a standalone popup.". The browser interface includes tabs, a search bar, and a status bar.

```
id="mapid">></div>
<script type="text/javascript">
var mymap = L.map('mapid').setView([51.505, -0.09], 3);

L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1IjoiYmVpY2hlbmgiLCJhIjoiY2tsdjl3dT5MHE2bDMwczU3emY1OW5vMyJ9.VVfIHxS7zO04N1trHAnEmw', {
    attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, Imagery &copy; <a href="https://www.mapbox.com/">Mapbox</a>',
    maxZoom: 18,
    id: 'mapbox/streets-v11',
    tileSize: 512,
    zoomOffset: -1,
    accessToken: 'your.mapbox.access.token'
}).addTo(mymap);

@for(conferenceList <- conferences){
    L.marker([@conferenceList.getX(), @conferenceList.getY()]).addTo(mymap).bindPopup("<b>@conferenceList.getYear(), @conferenceList.getName(), @conferenceList.getLocation()</b>").openOn(mymap);
}

<!-- var popup = L.popup()-->
<!-- .setLatLng([51.7, -0.89])-->
<!-- .setContent("I am a standalone popup.")-->
<!-- .openOn(mymap); -->
```

The screenshot is an external Map API which is leaflet API. So, when we receive the result from query 5 and get the conference metadata. Then we call the external Map APIs, from the screenshot, we can see the API URL should be

https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token=pk.eyJ1IjoiYmVpY2hlbmgiLCJhIjoiY2tsdjl3dT5MHE2bDMwczU3emY1OW5vMyJ9.VVfIHxS7zO04N1trHAnEmw where the parameter is the access_token which is created by my account on Internet.