Backpacker Recommender

Echipa:
Aghebanoae Oana, MSD2
Andries Simona, MSD2
Gabor Silvia, MSD2
Minea Iuliana, MSD2

1. Introduction

The purpose of ”Backpacker Recommender” application is to help the backpackers who travel a lot through foreign countries, often get lost or don't know how to get around in an unfamiliar place.[1]

The user has the posibility to create an account. He/she can specify his/her interests, preferred cuisines and if needs a wheelchair. All this information will be used when we are searching for places.

Fig 1 User preference within Bapr application

The application offers the user possibility to choose if he/she wants to search restaurant/hospitals/hotels/shops/museums nearby its current location or another selected city.

The application offers the posibility to monitor the travel progress. This includes the places visited by the user. User can also mark a location as favourite.

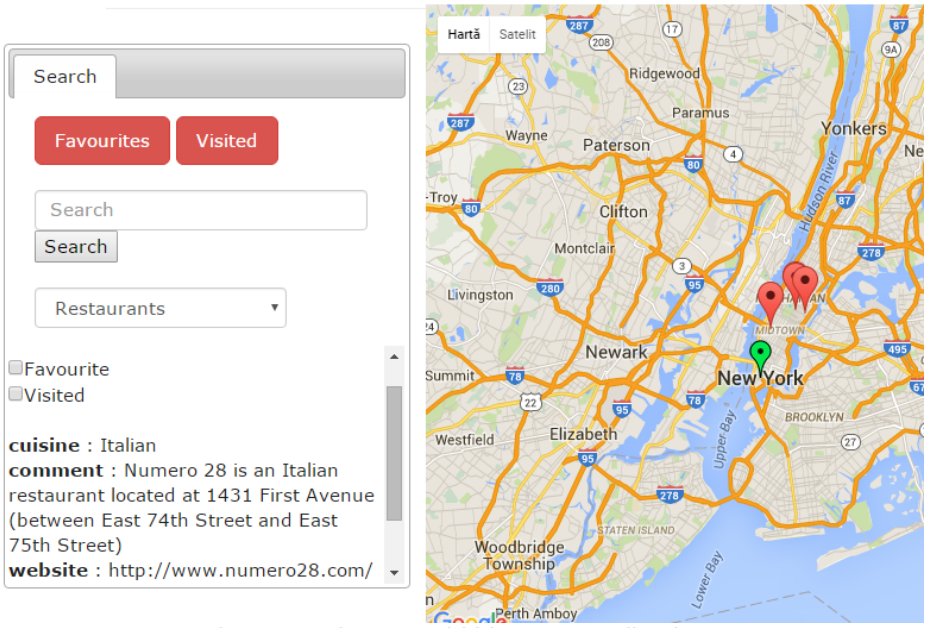

Fig 2 Search page whithin Bapr application

2. Used technologies

2.1 Web API

Asp.Net Web API is a framework for building HTTP services that can be consumed by a broad range of clients including browsers, mobiles, iphone and tablets. It is very similar to ASP.NET MVC since it contains the MVC features such as routing, controllers, action results, filter, model binders, IOC container or dependency injection. But it is not a part of the MVC Framework. It is a part of the core ASP.NET platform and can be used with MVC and other types of Web applications like Asp.Net WebForms. It can also be used as an stand-alone Web services application.[2]

In order to create the specification of the API we used RAML[3] because it offers a simple and intuitive way to describe the resources, methods, parameters and responses. The API will be accessed using the following methods:

- /Locations/Get?text=”park”&&latitude=42.6574&&longitude=23.5643&&userEmail=””
- /Hospitals/GetAllHospitalsNearby?latitude=42.6574&&longitude=23.5643&&userEmail=””
- /Hotels/GetByCoordinates?latitude=42.6574&&longitude=23.5643&&userEmail=””
- /Museums/GetMuseumsNearby?lat=42.6574&&lng=23.5643&&userEmail=””
- /Restaurants/GetRestaurantsNearby?lat=42.6574&&lng=23.5643&&userEmail=””
- /Shops/GetShopsNearby?lat=42.6574&&lng=23.5643&&userEmail=””
- /UserPreference/GetUserPreference?user=”email1@gmail.com”
- /UserPreference/SaveUserPreference?user=”email1@gmail.com”&&userPreference=”{’needWheelchair’ : ’true’}”

2.2 Data sources

The data sources that we will use are listed below. They can provide information about major points of interest for the user by their SPARQL endpoints. The namespace “geo” enables us to search by location latitude and longitude. This fulfills a major need of the application, since all the searches are based on geographical coordinates or based on the id of a resource.

- DBPedia
- LinkedGeoData

2.3 BrightstarDB

BrightstarDB is a native .NET NoSQL semantic web database. It can be used as an embedded database or run as a service.

BrightstarDB supports the W3C RDF and SPARQL 1.1 Query and Update. standards, the data model stored is triples with a graph context (often this is called a quad store). [4]

The RDF Client API provides a simple set of methods for creating and deleting stores, executing transactions and running queries. It should be used when the application needs to deal directly with RDF data. An RDF Client can connect to an embedded store or remotely to a running BrightstarDB instance.[5]

The BrightstarDB Entity Framework is a powerful and simple to use technology to quickly build a typed .NET domain model that can persist object state into a BrightstarDB instance.To use this you create a set of .NET interfaces that define the data model. The BrightstarDB tooling takes these definitions and creates concrete implementing classes. These classes can then be used in an application. The flexibility of the underlying storage makes evolving the model very easy and straight forward. BrightstarDB is optimized for associative data which provides a high performance when working with objects. As this is a fully typed domain model it also provides LINQ and OData support.[6]

BrightstarDB is used to keep users preferences and the travel progress. See below a sample from the application that shows how the preferred/visited location is saved :

```
var marker = ctx.Locations.Create();
marker.name = markedLocation.name;
marker.latitude = markedLocation.latitude;
marker.longitude = markedLocation.longitude;
marker.attributes = markedLocation.attributes;
if(type == "fav")
   marker.IsFavorite = markedLocation.IsFavorite;
else
   marker.IsVisited = markedLocation.IsVisited;
usr.MarkedLocations.Add(marker);
```

2.4 dotNetRdf

SPARQL is the standard query language for the Semantic Web and can be used to query over large volumes of RDF data. dotNetRDF provides support for querying both over local in-memory data using it's own SPARQL implementation and for querying remote data using SPARQL endpoints or through other stores SPARQL implementations.[7]

In our application we use dotNetRdf for doing sparql queries to http://dbpedia.org/sparql and http://linkedgeodata.org/sparql.

The query from Fig 3 will retrieve restaurants from DBpedia, which have the cuisine selected by the user. The search returns places existing in an area of 55 kilometers around.

```
string queryRestaurantsDBpedia = "SELECT DISTINCT ?name ?lat ?long ?cuisine ?comment ?website ?address ?dress_code ?head_chef ?reservations \n"
                    " WHERE { \n" +
                     "?f rdf:type dbo:Restaurant .\n" +
                     "?f rdfs:label ?name .\n" +
                     "?f geo:lat ?lat .\n" +
                     "?f geo:long ?long .\n" +
                     "Optional { ?f dbo:cuisine ?cuisine .}\n " +
                     "Optional { ?f rdfs:comment ?comment . FILTER(langMatches(lang(?comment ), \"en\"))}\n " +
                     "Optional { ?f foaf:homepage ?website .}\n " +
                     "Optional { ?f dbo:address ?address .}\n" +
                     "Optional { ?f dbp:dressCode ?dress_code  .} \n" +
                     "Optional { ?f dbp:headChef ?head_chef .} \n" +
                     "Optional { ?f dbp:reservations ?reservations .} \n" +
                     " FILTER ( langMatches(lang(?name), \"EN\")\n" +
                     " && ?lat > " + lat + " - 0.5 && ?lat < " + lat + " + 0.5\n" +
                     " && ?long > " + lng + " - 0.5 && ?long < " + lng + " + 0.5\n" +
                     (!string.IsNullOrEmpty(cuisine) ? " && lcase(?cuisine) in (" + cuisine + ") " : string.Empty) +
                     ")}\n" +"LIMIT 30";

//Create a Parameterized String
SparqlParameterizedString queryString = new SparqlParameterizedString();
//Add a namespace declaration
queryString.Namespaces.AddNamespace("geo", new Uri("http://www.w3.org/2003/01/geo/wgs84_pos#"));
queryString.Namespaces.AddNamespace("rdf", new Uri("http://www.w3.org/1999/02/22-rdf-syntax-ns#"));
queryString.Namespaces.AddNamespace("rdfs", new Uri("http://www.w3.org/2000/01/rdf-schema#"));
queryString.Namespaces.AddNamespace("dbo", new Uri("http://dbpedia.org/ontology/"));
queryString.Namespaces.AddNamespace("dbp", new Uri("http://dbpedia.org/property/"));
queryString.Namespaces.AddNamespace("foaf", new Uri("http://xmlns.com/foaf/0.1/"));
queryString.CommandText = queryRestaurantsDBpedia;

return BaprAPI.Utils.Utils.ParseSparqlQuery(queryString, @"http://dbpedia.org/sparql");
```

Fig 3. Sparql query to retrive restaurants from dbpedia with dotNetRdf

The query from Fig 4 will retrieve hotels from LinkedGeoData. If the user specified that he needs wheelchair, then the query will return only the hotels that are wheelchair accessible. The search returns places existing in an area of 55 kilometers around.

```
string searchHotelsQuery = "SELECT DISTINCT ?lat ?long ?name ?website ?phone ?address ?rooms ?internet_acces"
    + " ?opening_hours ?wheelchair ?stars ?operator"
    + " WHERE { ?s a lgd:Hotel .\n"
    + " ?s rdfs:label ?name . \n"
    + " ?s geo:lat ?lat .\n"
    + " ?s geo:long ?long . \n"
    + "OPTIONAL {?s foaf:homepage ?website .}\n"
    + "OPTIONAL {?s foaf:phone ?phone. }\n"
    + " OPTIONAL {?s lgd:address ?address .}\n "
    + " OPTIONAL {?s lgd:rooms ?rooms.} \n"
    + " OPTIONAL {?s lgd:internet_access ?internet_access.} \n"
    + " OPTIONAL {?s lgd:opening_hours ?opening_hours.} \n"
    + (userPreference.NeedWheelchair ? "?s lgd:wheelchair ?wheelchair. FILTER (?wheelchair =" + BaprAPI.Models.Constants.xsdBooleanIsTrue + ")\n"
                                     : " OPTIONAL {?s lgd:wheelchair ?wheelchair.} \n")
    + " OPTIONAL {?s lgd:stars ?stars. } \n"
    + " OPTIONAL {?s lgd:operator ?operator. } \n"
    + " FILTER ( ?lat > " + latitude + " - 0.5  && ?lat < " + latitude + " + 0.5"
    + " && ?long > " + longitude + " - 0.5 && ?long < " + longitude + " + 0.5 )\n"
    + "} LIMIT 10";

SparqlParameterizedString sparqlQueryString = new SparqlParameterizedString();
sparqlQueryString.Namespaces.AddNamespace("rdf", new Uri("http://www.w3.org/1999/02/22-rdf-syntax-ns#"));
sparqlQueryString.Namespaces.AddNamespace("rdfs", new Uri("http://www.w3.org/2000/01/rdf-schema#"));
sparqlQueryString.Namespaces.AddNamespace("lgd", new Uri("http://linkedgeodata.org/ontology/"));
sparqlQueryString.Namespaces.AddNamespace("geo", new Uri("http://www.w3.org/2003/01/geo/wgs84_pos#"));
sparqlQueryString.Namespaces.AddNamespace("foaf", new Uri("http://xmlns.com/foaf/0.1/"));
sparqlQueryString.CommandText = searchHotelsQuery;

return BaprAPI.Utils.Utils.ParseSparqlQuery(sparqlQueryString, @"http://linkedgeodata.org/sparql");
```

Fig 4. Sparql query to retrive hotels from LinkedGeoData with dotNetRdf

RDFS Reasoner

RDFS is an RDF vocabulary for expressing schemas for RDF data specified by the W3C. It allows for the definition of class and property hierarchies and specifying things like the domains and ranges of properties. We provide both a StaticRdfsReasoner which has to be initialised with

one/more schemas and uses only those to make inferences and a RdfsReasoner which is dynamic in that every Graph you apply reasoning to can extend the set of rules that the reasoner uses.[8]

The RDFS reasoner does not apply the full range of possible RDFS based inferencing but does do the following:

- Asserts additional type triples for anything which has a type which is a sub-class of another type
- Asserts additional triples where the property (predicate) is a sub-property of another property
- Asserts additional type triples based on the domain and range of properties[8]

2.5 Schema.org and RDFa

Schema.org vocabulary can be used with many different encodings, including RDFa, Microdata and JSON-LD. These vocabularies cover entities, relationships between entities and actions, and can easily be extended through a well-documented extension model.[9]

We use schema.org construct when we display options for searching. A sample from the application can be seen below:

```
<select class="form-control searchCategory" id="placeCategory" data-action="@Url.Action("GetPlacesByCategory", "Bapr")">
    <option value="Empty">Select a category</option>
    <option value="Restaurants" itemscope itemtype="https://schema.org/Restaurant">Restaurants</option>
    <option value="Museums" itemscope itemtype="https://schema.org/Museum">Museums</option>
    <option value="Hotels" itemscope itemtype="https://schema.org/Hotel">Hotels</option>
    <option value="Hospitals" itemscope itemtype="https://schema.org/Hospital">Hospitals</option>
    <option value="Shops" itemscope itemtype="https://schema.org/Store">Shops</option>
</select>
```

We use geo microformat for keeping user coordinates as it is shown below:

```
<div hidden class="geo" id="#userCoordinates">
    <span class="latitude" hidden title="@Model.latitude"></span>
    <span class="longitude" hidden title="@Model.longitude"></span>
</div>
```

In Contact page we use vCard for describing the contact details about application developers. An sample can be observed in the image below:

```
<div vocab="http://www.w3.org/2006/vcard/ns#"
     resource="http://example.com/me/Oana" typeof="Individual">
    <div>
        <span><b>Name : </b></span>
        <span property="formattedName">Aghebanoae Oana - Domita</span>
    </div>
    <div>
        <span><b>Telephone : </b></span>
        <span property="hasTelephone" typeof="Home Voice">
            <span property="telephone">61755555555</span>
        </span>
    </div>
    <div>
        <span><b>Email address : </b></span>
        <link property="email" href="mailto:oana.aghebanoae@info.uaic.ro" />
    </div>
    <div>
        <span> <b>Address </b></span>
        <div property="hasAddress" typeof="Home">
            <div>
                <span>      <b>Street :</b> </span>
                <span property="streetAddress">111, Unirii</span>
            </div>
            <div>
                <span>      <b>Locality : </b></span>
                <span property="locality">Iasi</span>
            </div>
```

Fig 5 vCard on the Contact page

2.6 Google Maps API

Google Maps API is used to geographically represent the places searched by the user and also to extract his current position.

We also used „*Place Autocomplete*”[10] service providing autocomplete functionality for text-based geographic searches.

3. Bibliography

1. http://profs.info.uaic.ro/~busaco/teach/courses/wade/projects/
2. http://www.dotnet-tricks.com/Tutorial/webapi/VG9K040413-What-is-Web-API-and-why-to-use-it-?.html
3. https://www.mulesoft.com/platform/api/api-designer
4. https://brightstardb.readthedocs.org/en/latest/Concepts/
5. https://brightstardb.readthedocs.org/en/latest/RDF_Client_API/#rdf-client-api
6. http://brightstardb.readthedocs.org/en/latest/Getting_Started/
7. https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/UserGuide/Querying%20with%20SPARQL
8. https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/UserGuide/Inference%20and%20Reasoning
9. https://schema.org
10. https://developers.google.com/places/web-service/autocomplete
11. http://profs.info.uaic.ro/~busaco/teach/courses/wade/web-film.html