RDF Lib

RDFlib or the RDFlib.js library is created to handle RDF data related to the RDF store.

Setting up RDFlib

Importing the rdflib library:

```
const $rdf = require('rdflib');
```

Data types

RDF data types can be

- IRIs
- Blank nodes
- Literals

Formatting IRIs

When using IRIs when matching triples and inserting data, the IRIs will have to be formatted correctly.

IRIs extend URIs by using the Universal Character Set, where URIs were limited to ASCII, with far fewer characters. IRIs may be represented by a sequence of octets but by definition are defined as a sequence of characters, because IRIs may be spoken or written by hand.

IRIs are mapped to URIs to retain backwards-compatibility with systems that do not support the new format.

Use the $rdf.sym syntax for setting it as an IRI.

```
$rdf.sym('https://sjoforsgata36.hovedbygg/room1');
```

Blank nodes

Blank nodes *are disjoint from IRIs and literals.*

Blank nodes represent a node without setting a permanent IRI. This is often used so that the Trinity system can generate the IRIs for the nodes.

The VUE local store keeps track of the blank node numbers. To increase the number you can use following method:

```
store.commit('blankNodeIncrement');

tempBlank = $rdf.sym(store.state.blankNodeString);
```

The blank node must be modified in the Turtle so it looks like:

:t_1 and not like <:t_1>

The brackets have to be removed for Trinpod to understand it.

Literals

`literal(value: string, languageOrDatatype?: string | NamedNode)`

Literals are used as object in triples for describing a text or value. Formatting a literal with language tag:

First parameter: The value of literal formatted as String

Second parameter: The language tag in 'brackets'

```
$rdf.literal("Site circulation "+sFacility, 'en'));
```

Data typed literals:

First parameter: Value of the literal (String)

Second parameter: (Can be undefined)

Third parameter: The type of the literal (Often XSD ontology standard)

```
$rdf.literal(sNetarea+'', undefined, ns.XSD('decimal'))
```

Prefixes

Turtle uses URIs to identify concepts all the time, so it has a way of abbrevating them. You declare a prefix say at the top of the URI, and then use `foaf:` prefix with a colon:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<profile.ttl#me>    a    foaf:Person   .
```

We create a prefixes file that keeps all the prefixes. Prefixes.js:

```
  NEO: $rdf.Namespace("https://neo.graphmetrix.net/node/"),
  FOAF: $rdf.Namespace('http://xmlns.com/foaf/0.1/'),
  RDF: $rdf.Namespace('http://www.w3.org/1999/02/22-rdf-syntax-ns#'),
```

To use a prefix in RDF lib, you can do like:

```
ns.RDF("type")
ns.NEO("x_door")
```

Adding triple to RDF store

Initializing an RDF store. Example is for a temp store:

```
tempRdfStore = $rdf.graph();
```

The RDF store is in the VUE store. This can be used to add a triple:

```
store.state.rs.rdfStore.add(subject, predicate, object, $rdf.default);
```

The $rdf.default defines that the default store is used.

Getting data from RDF store

Matching triples

You can use the match function for filtering triples from the RDF store. The match function takes in: Subject, predicate and object as parameters.

```
store.state.rs.rdfStore.match(item.subject, ns.FOAF("image"), null).forEach(foundItem => {
   imageuri = foundItem.object.value;
});
```

Remember that URIs will have to be coded with $rdf.sym

Using SPARQL query

Function for running a SPARQL query in RDF lib:

```
ur.runSparqlQuery = function(queryString = 'SELECT ?s ?p ?o WHERE { ?s ?p ?o } .'){
  const query = $rdf.SPARQLToQuery(queryString, false, store.state.rs.rdfStore);
  store.state.rs.rdfStore.query(query, function(result) {
      console.log('Query ran');
      console.log(result);
      return result;
  });
}
```

Serializing RDF store to Turtle

Turtle is the RDF exchange format used to send data to Trinpod. RDFlib provides method to generate Turtle from a store:

```
    $rdf.serialize(undefined, tempRdfStore, buildingPODurl, 'text/turtle', function(err, str) {

        console.log("Turtle serialize error: ", err);

        result = str;

    });
```

Parameter description:

```
serialize(target: null | NamedNode | BlankNode | Formula, kb: Formula, base?: unknown, contentType?: string,
callback?: ((err: undefined | null | Error, result?: string) => any), options?: {
    flags?: string;
    namespaces?: Record<string, string>;
}): string | undefined
```