REST API interactive tool
-Scholarly HTML-

Authors:
1. Bogdan Volosincu
2. Andrei Avram
3. Teodora Hoamea

Project description:

This web application is a framework that implements a tool that interacts by text with a REST API. For this,we used modeling vocabulary, taxonomies, thesaurus with RDF schema and SKOS and a database based on models.

Requirements:

- **Business requirements:**
  - R.A.T. should be able to recognize words in common language
  - R.A.T should be able to send a request to the API and search in the clients databases
  - R.A.T. should be able to return all the related results to the user search
- **User requirements:**
  - The user should be able to enter a word in different languages (romanian/english) and receive results to his search
  - The user should be able to click on a result an the he/she should be able to access the URL behind that result
- **Software requirements:**
- **Performance:**
  - The page should be able to sand a feedback to the user in less than 0.4 sec
- **Usability:**
  - This app should be able to be access from any browser
  - The operating systems shouldn't affect the functionality of the framework
- **Other requirements:**
  - Frontend: HTML, Css, JavaScript, REACT
  - Backend: NodeJs, Sails
  - Testing: java, Selenium, Postman, K6

Preliminary Considerations:

- Internal data structures/models:
  The internal data is organised in models based on sails v1.x. They represent a set of structured data, called records and correspond to a collection in the database.

Examples of models in this application:
- Actor has the following attributes: id {string}, name {string}, surname {string}, birthday {string}, movies {json}, plays {json}, foto {json}, prizes {json}.
- Movie has the following attributes: id {string}, title {string}, cast {json}, isReleased {boolean}, isPlaying {boolean}, isSeries {boolean}, releasedTimestamp {string}
- Review has the following attributes: id {string}, starts{number}, comment {string}, movieId {string}
- User has the following attributes: emailAddress{string} (this attribute is required and unique), emailStatus{string} (default value = confirmed), emailChangeCandidate {string}, password {string} (this attribute is also required), fullName {string}(required), isSuperAdmin {boolean} (this attribute indicates if the user has extra permissions), passwordResetToken {string} (A unique token used to verify the user identity), passwordResetTokenExpiresAt {number}, emailProofToken {string}, emailProofTokenExpiresAt {number}, stripeCustomerId{string} (The id of the customer entry in Stripe associated with this user), hasBillingCard {boolean}, billingCardBrand {string}, billingCardLast4 {string}, billingCardExpMonth {string}, billingCardExpYear {string}, lastSeenAt {number}

The models are accessed from within the controller, in this way the call model methods communicate with the database.

● External data sources:
Because this application is an OpenAPI schema for movies, an external data source is Internet Movie Database, IMDb which is a massive online website directory housing tons of information related to films, TV programs, video games, internet streams and more.

The list of movie database APIs such as the Open Movie Database and Internet Movie Database API can be accessed at the link:
https://rapidapi.com/collection/omdb-imdb-apis

Application flow:
The Proof of concept that we build exposes an OpenAPI schema for movies, similar to imdb.com
From the graphic interface the user is able to query the data using natural language.
The application will read the input from the user through an input text field.
Behind the scenes the input will be sent first to a syntactic and semantic module that will analyze the text and with the results from the parser the application will create a query to map user's words with the OpenApi schema as best as it can.

Example of scenarios:

When user enters inputs like:

1. Movie with Angelina Jolie. => extract Subject, Object and map to movies titles, movies cast or actor name.

2. Plays of Shakespeare. => map play to themes from OpenAPI schema like drama, movies, tragedy, theatre (query synsets from WordNet)

3. Best movies from last year. => map adjectives to Reviews, to prizes won by movie, actors

To map the words from user input to OpenApi schema we also create a dictionary of semantic terms based on OpenAPI endpoints, parameters and data models.

Linked Data Principles:

In this section it will be to develop how this web application incorporates the Linked Data Principles. This principles refers to URIs:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information.
4. Include links to other URIs so that they can discover more things.

These four rules help keeping data interconnected and they represent the unexpected re-use of information which is the value added by the web.

In this case, the response that the user is going to receive will be one or more URIs, based on how many results will be found in the database. Those URIs will contain the URL address for a significant page for that keyword.

The first and the second rules are respected by providing the user HTTP URIs responses that respect semantic web technology.

The third rule is applied by formatting every URI using RDF and the fourth rule is used if a user search a keyword, for example an actor, he/she will receive all the links related to that actor:

When the user sends "john" a RDF file will be created<http://RAT.org/john> then you it will be used the local identifiers within the file #smith, #Alan etc.

This is how the RDF file will look like:

<rdf:Description about="#john"
<fam:child rdf:Resource="#smith">
 <fam:child rdf:Resource="#Alan">
</rdf:Description>

The WWW architecture is going to return a global identifier "http://RAT.org/john#Alan" to Alan.