Towards an Open, Collaborative REST API for Recommender Systems

Iván García
Universidad Autónoma de Madrid
ivan.garciar@estudiante.uam.es

Alejandro Bellogín
Universidad Autónoma de Madrid
alejandro.bellogin@uam.es

ABSTRACT

Recommender Systems aim to suggest relevant items to users, however, for this they need to properly obtain/serve different types of data from/to the users of such systems. In this work, we propose and show an example implementation for a common REST API focused on Recommender Systems. This API meets the most typical requirements faced by Recommender Systems practitioners while, at the same time, is open and flexible to be extended, based on the feedback from the community. We also present a Web client that demonstrates the functionalities of the proposed API.

CCS CONCEPTS

• **Information systems → Users and interactive retrieval**; **Recommender systems**;

1 INTRODUCTION

Nowadays, Recommender Systems (RS) are an important component in the interactive internet world. The goal of these systems is to offer a comfortable and intuitive experience, being able to suggest useful recommendations and, at the same time, avoiding being lost in the Web. However, the services provided by these systems have increased exponentially in recent years; therefore, it is important to understand how to build the different layers of an RS in an efficient, useful way for the target users.

In this work, we focus on the definition of a common REST API that could be exploited in many different environments, similar to previous works that also provide REST endpoints to RS [3]. Our inspiration starts from the proposal made some time ago in [4], that, to the best of our knowledge, was never implemented or tested in any public library. We propose an adaptation of that REST API and, based on that, we present a system that instantiates such proposal, considering only open source libraries for its development. We also include a Web client that demonstrates how such a REST API could be used by the end users.

By making this (rather simple but very generic) API public, our main contribution is to open up discussions about how to improve it, while, at the same time, we provide a working implementation that supports such definition and makes easier to test any modification proposed by the community.

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 SYSTEM FRAMEWORK

2.1 Backend

The model stored in the backend of our system is based on three main entities: users, items, and events – these entities are at the core of any recommendation system [5], which allows us to provide a very generic API. The users, items, and events contain an identifier; besides, the users include some demographic information (age, gender, location, email) in case the system supports this type of data, items contain a name and other features so that content-based recommenders could be used [1], and each event stores the (user, item) pair that generated such event, together with the timestamp when it was created and two optional fields: value and type, to generalise common events such as ratings, clicks, or purchases, depending on the domain of the recommender system being used. At the moment, this information is stored in memory, but it can be easily transferred into a relational database or to a document-oriented database such as MongoDB (since, as described in the next sections, most of the data is converted from/to JSON format).

2.2 Open source libraries

We use open source libraries based on Java to develop the REST backend, mostly Dropwizard[1] and RankSys[2]. Dropwizard is a Java framework composed of many libraries, most of them already mature, that encapsulates creating an HTTP server, parsing the requests, transforming JSON data into objects, and other aspects allowing to create Web services that are maintainable and easy to configure very quickly.

Regarding the generation of recommendations, we used RankSys because it is one of the most popular libraries nowadays. Nonetheless, the recommendation component in our system can be easily modified so that other Java-based libraries such as Lenskit or Librec could be used instead – even libraries in other languages (such as MyMediaLite) could be plugged in via triggers or other Web services if necessary.

2.3 API endpoints

Table 1 shows some of the most representative endpoints defined in our proposed API for recommendation. Based on the proposal from [4], we analysed each task that may be needed or used by an end user in a recommendation system and defined a set of coherent (from a semantic point of view of the endpoints) URLs, following standard REST principles and patterns [2].

As we show in this table, we use the HTTP methods POST, GET, and DELETE (DEL) to discriminate between similar endpoints. We decided to provide some redundancy in the names of the URLs

[1] Available at https://www.dropwizard.io. Accessed June 2018.
[2] Available at http://ranksys.org. Accessed June 2018.

URL	Method	Description
user/add	POST	Add a user
user/get/{uid}	GET	Returns a user
user/get	GET	Returns all users
user/delete/{uid}	DEL	Removes a user
user/get/{uid}/events	GET	Returns the events of a user
item/add	POST	Add an item
item/get/{iid}	GET	Returns an item
item/get	GET	Returns all items
item/delete/{iid}	DEL	Removes an item
item/get/{iid}/events	GET	Returns the events of an item
event/add	POST	Add an event
event/get/user/{uid}/item/{iid}	GET	Returns an event
user/get/{uid}/recommendations	GET	Returns recommendations of a user
train	GET	Train the recommender
statistics/get	GET	Returns statistics about the system

Table 1: Selection of most representative API endpoints.

(for instance, most of the GET methods include *get* in the URL) to make it more human-readable. In general, endpoints using the GET method obtain information from the backend, sometimes based on a parameter received in the URL; a JSON object is returned if a complex data structure is required. On the other hand, those endpoints using the POST method send information to the backend as JSON data. Finally, the DELETE method allows to remove information from the backend model.

It is worth noting that the endpoints can be grouped according to the three entities described before: those related to users (add, return, remove), items, and events. There is a fourth group of requests related to the recommender system: by using the `train` endpoint, we can setup a specific recommender with some parameters and train it based on the data received up to that point. Then, the recommendations generated for a particular user will be based on such technique. Additionally, we provide a REST method that returns the statistics of the system, such as the number of users, items, average number of events per user/item, etc.

Other endpoints not presented here but implemented in the proposed REST API are related to the different types of events (so that we can return the ratings or clicks of a particular user or item) and redundant methods included to provide completeness in terms of the three entities – namely, there is a method `event/get/user/{uid}` that is completely equivalent to `user/get/{uid}/events`.

2.4 Web client

When building a REST service, it is important for the end user to obtain an easy-to-use, intuitive system. Because of this, we provide a Web page where most of the API functionalities can be tested.

More specifically, the implemented Web client allows to add users, items, and events into the system, while, at the same time, it shows the existing data in the server regarding users, items, and events. It also lets the user to load a Movielens-like dataset. For this, we used the FreeMarker[3] templates from DropWizard, an extension of classical Java JSPs that use the FreeMarker Template Language (FTL) to interact with the data.

[3] Available at https://freemarker.apache.org/index.html. Accessed June 2018.

3 DEMONSTRATION

In the demonstration, we will be showing how to instantiate the proposed REST API, emphasising that several APIs can coexist in the same server by using different ports. The Web client will then be used to illustrate how the data is added to the backend model. At any moment, we can show the current status of the model, which, in particular, allows us to check how the model is modified once users, items, or events have been added or removed. Additionally, we will show the techniques supported to generate recommendations according to different parameters, comparing the obtained outputs and analysing aspects such as coverage (nearest neighbours algorithms suffer more from coverage than matrix factorisation) or personalisation capabilities (a random recommender looks more personalised than a popularity-based technique).

One of the goals of the demonstration is to evidence how easy our system can be configured to satisfy any requirement from the end users, e.g., creating new API endpoints or changing how the parameters are received (from POST to GET or as URL parameters).

The source code for our demo can be found in the following GitHub repository: abellogin/REST4RecSys.

4 FUTURE WORK

In this work, we present our view of a REST API for Recommender Systems, however, we would like to offer the community the possibility of agreeing on a common API useful in different scenarios. Because of this, we expect the repository where this API is defined could be used as a forum to discuss other features to be added or any modifications the community may find interesting.

Besides this, our implementation is just a proof-of-concept of how it can be implemented. As mentioned before, other recommendation libraries could be integrated or an actual database could be used to make the events persistent every time the server is restarted, just to name a few straightforward modifications. Additionally, the Web client presented could be further extended to provide more functionality, and even other types of clients could be implemented, such as mobile apps or wrappers in any programming language, so that the API would be easier to use by end users.

ACKNOWLEDGMENTS

This work was supported by the project TIN2016-80630-P (MINECO).

REFERENCES

[1] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. 2015. Semantics-Aware Content-Based Recommender Systems. In *Recommender Systems Handbook*, Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). Springer, 119–159. https://doi.org/10.1007/978-1-4899-7637-6_4

[2] Thomas Erl, Benjamin Carlyle, Cesare Pautasso, and Raj Balasubramanian. 2013. *SOA with REST - Principles, Patterns and Constraints for Building Enterprise Solutions with REST*. Pearson Education. http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0137012519,00.html

[3] Emanuel Lacic, Matthias Traub, Dominik Kowald, and Elisabeth Lex. 2015. ScaR: Towards a Real-Time Recommender Framework Following the Microservices Architecture. In *Proceedings of the Workshop on Large Scale Recommender Systems (LSRS2015) at RecSys 2015.*

[4] RecSysWiki. [n. d.]. Common Recommender REST API. https://web.archive.org/web/20160324042313/http://www.recsyswiki.com:80/wiki/Common_Recommender_REST_API. Accessed April 2018.

[5] Francesco Ricci, Lior Rokach, and Bracha Shapira (Eds.). 2015. *Recommender Systems Handbook*. Springer.