Conference Organization App: Getting Started

PDF Download

In Project 4, you will develop a cloud-based API server to support a provided conference organization application using Google Cloud Endpoints. It is the same conference organization app that you have worked on in the course, which ends up being functional but somewhat limited. In that app state, a conferences just has a name, description and date when the conference happens. But usually conferences have more than that - there are different sessions, with different speakers, maybe some of them happening in parallel!

Your task in this project is to add this functionality. Some of the functionality will have well defined requirements, some of it will be more open ended and will require you both design the specification and implement the solution. You do not have to do any work on the frontend part of the app to successfully finish this project. All your added functionality will be testable via APIs Explorer.

The functionality to be added to the app can be broken down into 4 tasks:

Task 1: Add Sessions to a Conference

A session can have speakers, start time, duration, type of session (as in workshop, lecture, forum, etc.), and a location. You will need to define the `Session` class and the `SessionForm` class, as well as appropriate Endpoints shown below. You are free to choose how you want to define speakers (just as a string or as a full fledged entity).

You must define the following methods:

`getConferenceSessions(websafeConferenceKey)` - Given a conference, return all sessions

`getConferenceSessionsByType(websafeConferenceKey, typeOfSession)` - Given a conference, return all sessions of a specified type (eg lecture, keynote, workshop)

`getSessionsBySpeaker(speaker)` - Given a speaker, return all sessions given by this particular speaker, across all conferences

`createSession(SessionForm, websafeConferenceKey)` - Create a session, open only to the organizer of the conference

For the respective `Session` and `SessionForm` class, pass in the following arguments: session name, highlights, speaker, duration, type of session, date, and start time (in 24 hour notation so it can be ordered). Ideally, you would create the session as a child of the conference, though you are not bound to this app structure design decision. Either way, you must explain in a couple of paragraphs in your project `README` file your design choices for session and speaker implementation.

Task 2: Add Sessions to User Wishlist

Users should be able to mark some sessions that they are interested in and retrieve their own current wishlist. You are free to design the way this wishlist is stored.

You must define the following methods:

`addSessionToWishlist(SessionKey)` -- adds the session to the user's list of sessions they are interested in attending. You can decide if they can only add conference they have registered to attend or if the wishlist is open to all conferences.

`getSessionsInWishlist()` -- query for all the sessions in a conference that the user is interested in

Task 3: Work on indexes and queries

For this task, there are three main objectives you need to target. First of all, make sure the indexes support the type of queries required by the new Endpoints methods.

Index Viewer in Admin Console

To get to the Index Viewer in the Admin Console, go to the Developer's Console at https://console.developers.google.com/, select your project, go to **Cloud Datastore**, then click **Indexes**.

Auto generated index configuration file

When you run a query on localhost, if the query needs a composite index, then that index is written to the auto-generated index configuration, which is in:

target > artifactname-1.0 > WEB-INF > appengine-generated > datastore-indexes-auto.xml

For example:

target > conference-1.0 > WEB-INF > appengine-generated > datastore-indexes-auto.xml

Note: If you don't see the appengine-generated folder, run your application on localhost, then go to the "Show Conferences" page in the browser. Then refresh your project in Eclipse. Now you should see the appengine-generated folder containing datastore-indexes-auto.xml.

When you deploy your application to appspot, the index definitions defined in the auto-generated index config files are deployed to appspot too, and will start building on appspot. Apps on appspot cannot serve queries that require index definitions that either do not exist or are not ready to start serving.

The contents of the auto-generated index file is transient, it exists until you deploy the app to appspot. Some other things could cause the file to be cleared out, so it's a good idea to check the auto-generated index file before deploying to appspot if you are relying on it to define required indexes.

Read more at Using Automatic Index Configuration

Manual Index Configuration Files

You can also add index definitions to a manual index configuration called datastore-index.xml in the WEB-INF folder in src > main > webabb. You will need to create this file the first time you want to use it. When you deploy your application to appspot, all indexes defined in either the auto-generated index config file and the manual config file will be deployed and will be built (if they don't already exist).

Read more at About datastore-indexes.xml

Resources

Developer Documentation for Datastore Indexes
https://developers.google.com/appengine/docs/java/datastore/indexes

Java Datastore Index Configuration
https://developers.google.com/appengine/docs/java/config/indexconfig

Queries

As your second objective, you need to think about other types of queries that would be useful for this application. Describe the purpose of 2 new queries in your README file and write the code that would perform them.

Lastly, you should solve the following query related problem. Let's say that you don't like workshops and you don't like sessions after 7 pm. How would you handle a query for all non-workshop sessions before 7 pm? What is the problem for implementing this query? What ways to solve it did you think of, prior to implementing your solution in code? The response to this question should also be included in the `README` file.

Task 4: Add a Task

When a new session is added to a conference, check the speaker. If there is more than one session by this speaker at this conference, also add a new Memcache entry that features the speaker and session names. You can choose the Memcache key. The following single Endpoint should be defined:

`getFeaturedSpeaker()` - returns the featured speaker, if any.

Those are the 4 tasks you should set out to finish to complete this project. The following table summarizes the 4 tasks explained above:

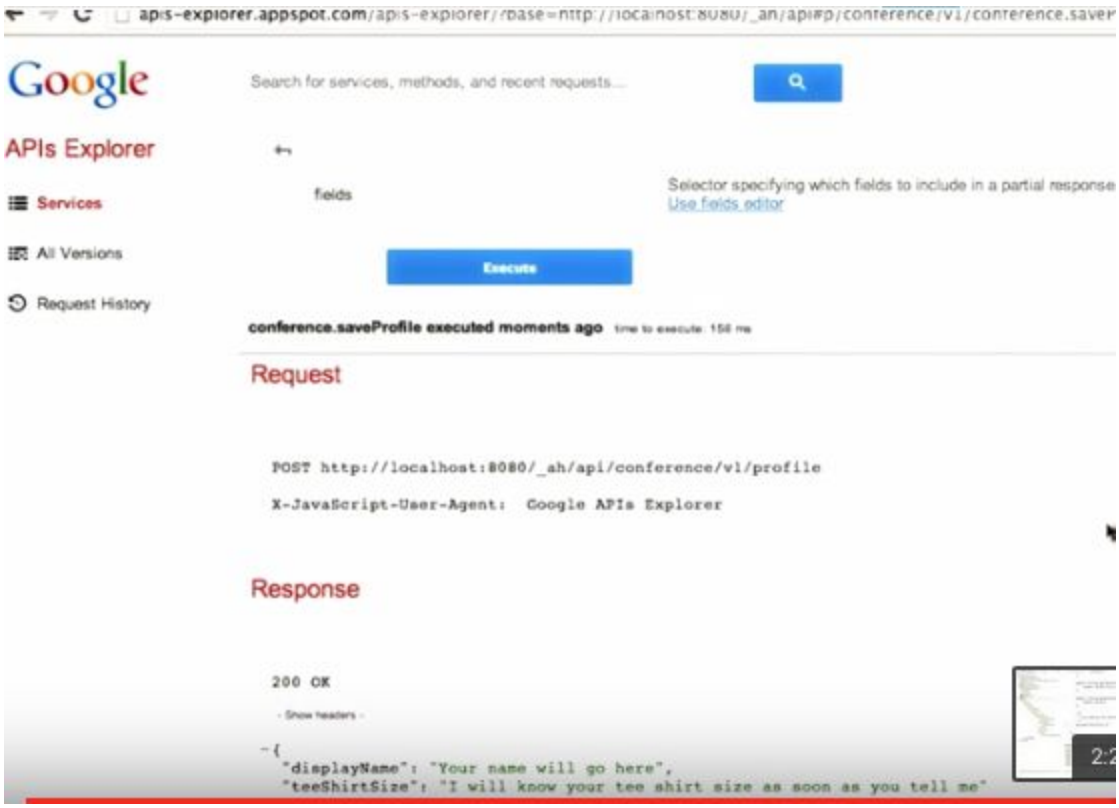
Task 1	*Endpoint methods:*
	
	`getConferenceSessions(websafeConferenceKey)`
	
	`getConferenceSessionsByType(websafeConferenceKey, typeOfSession)`
	
	`getSessionsBySpeaker(speaker)`
	
	`createSession(SessionForm, websafeConferenceKey)`
		Include design choices explanation in your README file

Task 2	Endpoint methods:
	
	`addSessionToWishlist(SessionKey)`
	`getSessionsInWishlist()`	

Task 3	Indexes check and two new queries	Include design choices explanation in your README file
Task 4	*Endpoint methods:* `getFeaturedSpeaker()`	

Testing your project

As you work on each one of the task, you should be testing them with the APIs Explorer.



Useful links

- Project rubric
- P4 Webcast
- Project Description