Performer, Eventee

1> User: User relationship (Eventee : Performer)

In our project, an eventee follows performers and a performer has a list of eventees that follow him.

Thus we have a many-to-many relationship between Eventee and Performer.

To test this relationship do:-

In Fron End:

1) Eventee -> Performer
    a. Go to Eventee Page.
    b. You will see list of Eventees
    c. Click on one eventee to populate his profile
    d. In profile you will see a list of All Performers
    e. Select one perform, an eventee wishes to follow
    f. Click “+” to follow that performer
    g. Check more performers later and click “+” to follow more performers one by one
    h. You can click on “-“ to remove the performer from following list
    i. Now you can see List Of Performers a particular eventee follows on the Eventee Page being updated. s
2) Performer -> Eventee
    a. Go to Performer Page by clicking on .
    b. Click on one Performer to see a list of All Eventees that follow the performer.

In Backend:

RelationShip	Api EndPoint	Examples

Eventee follows Performer	PUT MAPPING:
	/api/eventee/{eid}/performer
		eid: Eventee ID
		
		Send Performer Object in Body:
		
		{
		
		}

Eventee unfollows Performer	PUT MAPPING:
	
	/api/eventee/{eid}/unfollowperformer
		eid : EventeeId
		Send Performer Object in Body:
		{
		}

Eventee views List Of Performers he follows	GET MAPPING /api/eventee/{eid}/performer	Eid: EventeeId
Performer views List Of Eventee following him	GET MAPPING /api/performer/{pId}/followers	pId: Performer ID.

2> User Searches for list of domain objects based on criteria (Any User can search for an event based on city and/or category)

In our project, any user be it logged in or anonymous or user of any user type can query a search for an event based on category and/or city.

Over here we are quering our database as well as third party api (eventfull) to get data of events.

To test this follow:-

In Front End:

a. Navigate to homepage
b. There are two text boxes one for city and the other for category
c. Fill any, either or None to get the data
d. If you fill both it will do a query based on both the parameters. If you input one field it will query based on that parameter. If no input provided it will return a list of all events in our database.

In Backend:

RelationShip	Api EndPoint	Examples

Search for Event Based on city and category	Get MAPPING: /api/host/event?cityName={cityName}&category={cateGory}	cityName: Boston category: music
Search for Event Based on city	Get MAPPING: /api/host/event?cityName={cityName}	cityName: Boston
Search for Event Based on category	Get MAPPING: /api/host/event? category={cateGory}	cateGory: music
Search for all Events	Get MAPPING: /api/host/event	pId: Performer ID.

3> A user views details of a particular domain object listed in the search results

In our project, user can view more details of event by clicking on it. We have provided this feature in front end as well as backend.
However for other domain objects like review, venue and ticket we are providing backend apis to view details of them.

To test this follow:-

In Front End:

a. Navigate to homepage
b. Search By Query Parameters as mentioned in the above use case.
c. Click on any Event To view Details about the Event.

In Backend:

RelationShip	Api EndPoint	Examples
Search for Event Based on Id	Get MAPPING: /api/event/{eventId}	eventId:
Search for Review Based on Id	Get MAPPING: /api/review/{reviewId}	reviewId:
Search for Venue Based on Id	Get MAPPING: /api/venue/{venueId}	venueId:

Search for Ticket Based on Id	Get MAPPING: /api/ticket/{ticketId}	ticketId:

4> A user views all domain objects related to the user,

In our project, host can view more details of himself by clicking on his name. One of the values in it is List of Events which he can view in his details page.

To test this follow:-

In Front End:

a. Navigate to host page

b. Click on any Host To view Details about the Host.

c. In the details page you will see list of events.

In Backend:

RelationShip	Api EndPoint	Examples
Search for Event Based on HostId	Get MAPPING: /api/event/{hostId}	hostId:

5) A user views all other users related to the user,

This is the same as usecase 1. Details are provided to satisfy this usecase in the first point.

6) A user related to a domain object

We have the following relationships between user and domain object. Host CRUDS event. The relationship between Host and event is OneToMany. Performer can perform for many events. Similarly an event can have many performers. Thus the relationship between Performer and Event is many to many. In front end we have created so that a Host can create a event. Once created host can view list of events he created.

FrontEnd Testing Steps:

a. Go to Host Page.
b. You will see a list of hosts with multiple symbols.
c. Click on “+” symbol next to any host to create an event for that host.
d. On event page you will create the event.

e. Once the event is created navigate back to the host page.
f. Click on the host name for which you created the event.
g. This will open a details page of Host with List Of Events Created By Host.

In Backend:

Relationship	Api EndPoint	Examples

Host creates event	Post MAPPING:
	/api/host/{userId}/venue/{venueId}/event
		hostId:
		
		Send Event Object in Body:
		
		{
		}

Host adds performer to Event	PUT Mapping
	
	/api/host/event/{eventId}/performer
		eventide:
		
		Send Performer Object in Body:
		
		{
		}

Host Removes Performer for an Event	PUT Mapping /api/host/event/{eventId}/performer/{performerId}	eventId: performerId:
Host Views List Of Events he hosted	GET Mapping /api/host/event?hostId={hostId}	hostId:
Performer gets events he performs in	GET Mapping /api/host/event?performerId={performerId}	performerId:
List Of Performers is shown who perform in some event	GET Mapping /api/event/{eventId}/performers	eventide:

7) A domain object related to another domain object

In our application, we are assigning a venue to an event. Thus there is OneToMany Mapping. There can be many events occurring on a venue. Thus an event has a venue but a venue has a list of events. This is implemented and shown in Front End. Other domain objects that we have implemented which can be seen from backend are. One Review can be written, created for One Ticket. Thus it has a OneToOne Relationship. Also Ticket To Event is a ManyToOne Relationship. There can be many tickets for an event. But a ticket is valid for only one event.

FrontEnd Testing Steps:

a. Go to Event Page.
b. Either you can choose a venue from the dropdown when creating an event or
c. You can update venue from an existing event by clicking on the event and updating venue
d. Now click on the create or update button respectively.
e. When u click on the event name from event list you will see the venue associated with that event

In Backend:

Relationship	Api EndPoint	Examples
Add Venue To Event	Put MAPPING: /api/event/{eventId}/venue	eventId:
Create Ticket For an Eventee of an Event	POST Mapping /api/event/{eventide}/eventee/{eventeeId}/ticket	eventeeId:
List Tickets For an Event	GET Mapping /api/ticket?eventId={eventId}	eventId:

Create Review for Ticket	POST Mapping
	
	/api/ticket/{ticketId}/review
		ticketId:
		
		Send Review Object in Body:
		
		{
		}

Find All Reviews For Event	GET Mapping /api /event/{eventId}/review	eventId:

8, 9, 10, 11)

ADMIN does CRUD on Users.

Admin can create, view, update, delete any user. He can also view list of users.

Testing on Front End:

1. Navigate to “/admin”.
2. You will notice creation page and List of all users below
3. Fill in details of input groups and click on “create” to create a new user.
4. Click on a user from the list to select a specific user and change his details and click on update to update user details or click on delete to delete user.

Details page of user is not created in Front End but there is a facility in backend for it.

Relationship	Api EndPoint	Examples
Admin creates user	POST MAPPING: /api/admin/performer – [CREATE PERFORMER] /api/admin/host - [CREATE HOST] /api/admin/eventee - [CREATE EVENTEE]	
Retrieve List Of User or By Type	POST Mapping /api/admin/{type}	type:
List Tickets For an Event	GET Mapping /api/ticket?eventId={eventId}	eventId:
Update Performer for admin	PUT Mapping /api/admin/performer/{performerId}	performerId:
Update Performer for host	PUT Mapping /api/admin/host/{hostId}	hostId:
Update Performer for eventee	PUT Mapping /api/admin/ eventee /{ eventeeId}	eventeeId:
Delete user by userId	/api/admin/{type}/{userId}	Type: userId:

Below is a list of all backend apis with their usage in case if I missed some:

ADMIN SERVICE			
Add performer	/api/admin/performer	POST	
Add host	/api/admin/host	POST	
Add eventee	/api/admin/eventee	POST	
Get users by type or get all users	/api/admin/{type}	GET	
Update Performer for admin	/api/admin/performer/{performerId}	PUT	
Update Eventee for admin	/api/admin/eventee/{eventeeId}	PUT	
Update Host for admin	/api/admin/host/{hostId}	PUT	
Delete user by userId	/api/admin/{type}/{userId}	DELETE	
Add Event	/api/admin/event	POST	
Update Event	/api/admin/event/{eventId}	PUT	
Delete Event	/api/admin/event/{eventId}	DELETE	
Get Events	/api/admin/event	GET	
Add Ticket	/api/admin/ticket	POST	
Update Ticket	/api/admin/ticket/{ticketId}	PUT	
Delete Ticket	/api/admin/ticket/{ticketId}	DELETE	
Get Ticket	/api/admin/ticket	GET	
Add Venue	/api/admin/venue	POST	
Update Venue	/api/admin/venue/{venueId}	PUT	
Delete Venue	/api/admin/venue/{venueId}	DELETE	
Get Venue	/api/admin/venue	GET	
EVENTEE SERVICE			
Add eventee for eventee	/api/eventee	POST	
Get all eventees	/api/eventee	GET	
Get eventee by Id	/api/eventee/{eventeeId}	GET	
Update eventee	/api/eventee/{eventeeId}	PUT	
Follow performer for eventee	/api/eventee/{eid}/performer	PUT	
Unfollow performer for eventee	/api/eventee/{eid}/unfollowperformer	PUT	
Get list performers not followed	api/eventee/{eventeeId}/notfollowing	GET	

Get list performers followed	/api/eventee/{eventeeId}/performer	GET	
Delete eventee	/api/eventee/{eventeeId}	DELETE	
HOST SERVICE			
Create host for host	/api/host	POST	
Get all hosts	/api/host	GET	
Get host by id	/api/host/{hostId}	GET	
Get host by event id	/api/event/{eventId}/host	GET	
Update Host By id	/api/host/{hostId}	PUT	
Delete Host by id	/api/host/{hostId}	DELETE	
PERFORMER SERVICE			
Create Performer	/api/performer	POST	
Get all performers	/api/performer	GET	
Get performer by id	/api/performer/{performerId}	GET	
Get eventees following performer	/api/performer/{performerId}/followers	GET	
Get performers for an event by event id	/api/event/{eventId}/performers	GET	
Update performer	/api/performer/{performerId}	PUT	
Delete performer	/api/performer/{performerId}	DELETE	
REVIEW SERVICE			
Create review for a ticket	/api/ticket/{ticketId}/review	POST	
Get all reviews for an event	/api/event/{eventId}/review	GET	
Get event by Id	/api/review/{reviewId}	GET	
Update review for ticket	/api/ticket/review/{reviewId}	PUT	
Delete review	/api/ticket/review/{reviewId}	DELETE	
TICKET SERVICE			
Create ticket for an event and eventee	/api/event/{eventId}/eventee/{eventeeId}/ ticket	POST	
Get all tickets	/api/ticket	GET	
Get ticket by id	/api/ticket/{ticketId}	GET	
Update ticket	/api/event/ticket/{ticektId}	PUT	
Delete ticket	/api/ticket/{ticketId}	DELETE	
VENUE SERVICE			
Create venue	/api/venue	POST	

Get all venues	/api/venue	GET	
Get venue by id	/api/venue/{venueId}	GET	
Update venue	/api/venue/{venueId}	PUT	
Delete venue	/api/venue/{venueId}	DELETE	
EVENT SERVICE			
Create event for host	/api/host/{userId}/venue/{venueId}/event	POST	
Add performer to an event	/api/host/event/{eventId}/performer	PUT	
Remove performer from an event	/api/host/event/{eventId}/performer/ {performerId}	PUT	
Add venue to an event	/api/event/{eventId}/venue	PUT	
Get all events	/api/host/event	GET	
Get event by event id	/api/event/{eventId}	GET	
Update event	/api/host/event/{eventId}	PUT	
Delete event	/api/host/event/{eventId}		