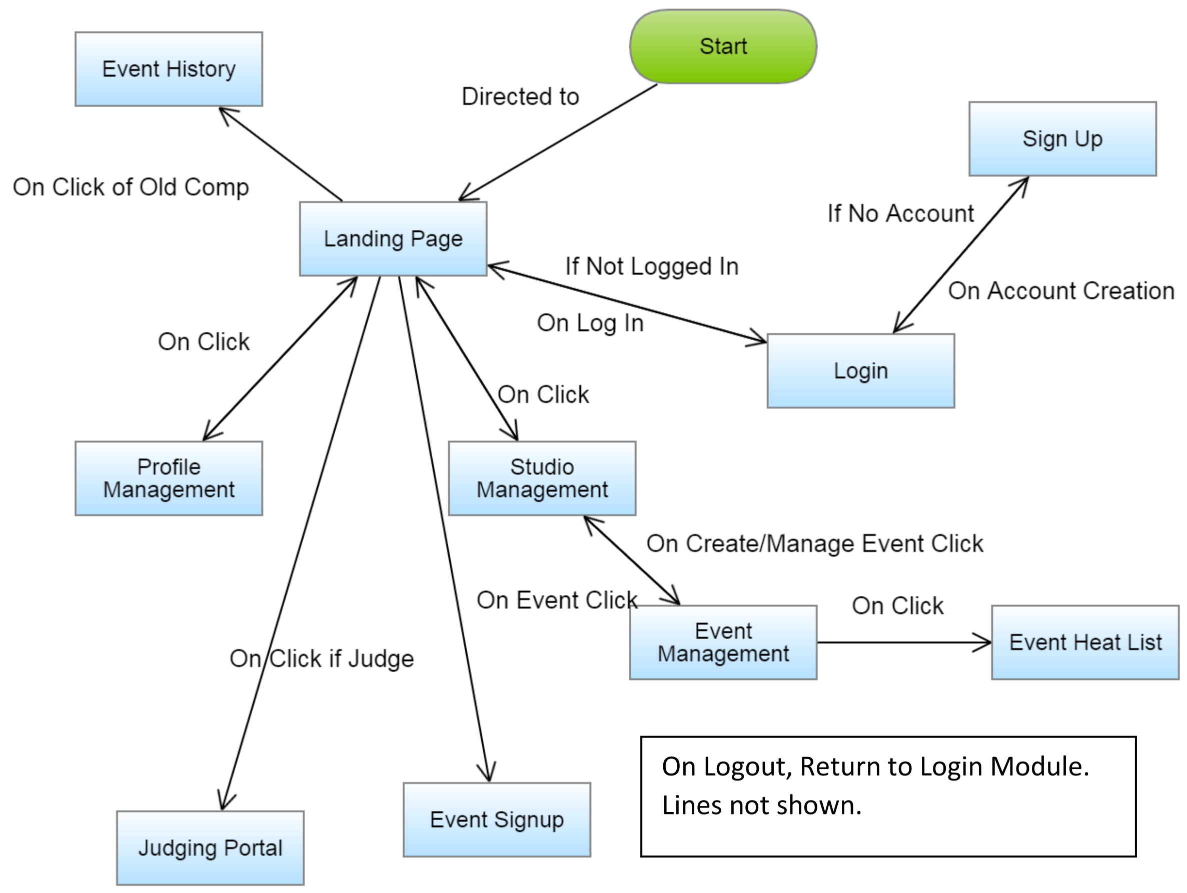
Ballroom Comp Manager Design Document

Site Map



On Logout, Return to Login Module. Lines not shown.

Modules

Landing Page

Features

List of Upcoming Events which can be selected for sign up or for judging (two separate links/buttons)

Receives from Server

Events – array of JSON serialized events in the following format

```
{
   id : Integer.
   date : String. ISO 8601 format "YYYY-MM-DD"
   registration_close_date: String. ISO 8601 format "YYYY-MM-DD"
   name :String.
}
```

Sends to Server

GET request to /event/(event_id: (0-9)+)/

Response:

redirect to Event Sign Up Module if future event

redirect to Event History Module if past event

GET request to /event/(event_id: (0-9)+)/judge

Response:

Redirect to /event/(event_id: (0-9)+)/ if not an authorized judge

Redirect to /event/(event_id: (0-9)+)/judge if an authorized judge

GET request to /profile/

Response: redirect to Profile Module

GET request to /studio/

Response: redirect to Studio Module

Login Module

Template

registration/login.html

Features

A basic login form that accepts a username and password and validates the user.

Receives from Server

Login Form – a Django-to-HTML5 form that must be used in order to authenticate the user

CSRF token

Sends to Server

POST request to /login/

Data: Submitted Form

Response:

Refresh page with errors if bad auth.

Redirect to landing page (“/”) if good auth.

GET request to /signup/

Response: redirect to Signup Module

Signup Module

Template

session/signup.html

Features

Form for creating a new account.

Receives from Server

Django-to-HTML5 form for account creation

CSRF token

Sends to Server

POST request to /signup/

Data: submitted form

Response:

Redirect to /login/ on successful creation

Refresh with errors on erroneous submission

Profile Module

Features

A table with a filter set for searching up personal results from past competitions

A change password area

A Studio association area

Receives from Server

Profile information in JSON format

```
Format:  {
           Name: string
           Studio: string or null
         }
```

Change password Django-to-HTML5 form

CSRF token

Event results matching URL querystring filters

Array of events in JSON format client has participated in for populating a dropdown in the filter

```
Format: [
            {
               Name: String.
            }
         ]
```

Sends to Server

POST request to /profile/

Data: submitted form

Response:

Refresh Page with errors if change password was bad

Refresh Page without errors if change password was successful

POST request to /profile/associate

Data: association pin

Response:

Refresh Page with errors if pin does not match a school

Refresh Page with success message if association successful

GET request to /profile/ with querystrings

Parameters:

date: expected format = “YYYY-MM-DD”, competition date

name: competition name

Response:

Array of JSON serialized events with nested arrays of JSON serialized places in the format:

```
[{
   date: string, format: “YYYY-MM-DD”
   event_name: string
   placements:   {
                     place: integer
                     event name: string
                  }
}]
```

Event Signup Module

Features

Form for Competition that imitates the functionality of the existing O2CM competition signup form.

Receives from Server

Django-to-HTML5 form for event signup with prepopulated information for available dancers and events

CSRF token

Sends to Server

POST request to /event/(event_id: (0-9)+)/

Data: submitted form

Response:

Redirect to "/" on successful submission

Refresh with Errors on erroneous submission

Judging Module

Features

Replicates the current judging system for marking contestants divided by heats.

Asynchronously checks for next event+round then loads next event+round after judge has submitted the prior event's round form

Receives from Server

Django-to-HTML5 form for marking contestants, prepopulated with the dancers for that heat

CSRF token

Sends to Server

POST request to /event/(event_id: (0-9)+)/judge

Data: submitted form

Response:

Refreshes with next event's round on success.

Refreshes without losing data on same event on failure

Event History Module

Features

Replicates the current results system of O2CM

Receives from Server

Array of JSON serialized events in the following format:

```
Format: [{
            date: ISO 8601 Format, YYYY-MM-DD, string
             name: String
        }]
```

Sends to Server

GET request to /event/(event_id: (0-9)+)/ with querystring arguments

division
age
skill
style
competitor

Response:

Array of event and round information in JSON format:

```
[{
      event_name : string
      rounds: [{
            round_number: Integer,
            results: [{
                  couple: {
                        lead: String
                        follow: String
                  }
            Marked: Boolean
            Total_marks: Integer
            Place: Integer
            }]
      }]
}]
```

Studio Management Module

Features

List of events this studio has or will host.
Association Pin generator

Receives from Server

Information of studio in JSON format:

```
{
    name: String
    pin: Integer
    events: [
          {
              Name: string
              Date: String, ISO 8601, YYYY-MM-DD
               Last_date_of_registration: String, ISO 8601, YYYY-MM-DD
          }
    ]
}
```

Sends to Server

POST request to /studio/pin
    Data: new pin
    Response:
        refresh with updated pin on success
        refresh with error on erroneous submission
GET request to /studio/competition

Response:
Redirect to competition management loading in the data of the event clicked
GET request to /studio/competition/new
Response:
Redirect to new competition creation form

Competition Management Module

Features

Event creation form if navigated to /studio/competition/new
If navigated to /studio/competition/(competition_id [0-9]+)/ List of clickable events which bring up their list of rounds, with a list of dancers in the rounds. The event admin can change the number of dancers in a heat, can DQ dancers, can select which heat is active, and can select which event is active.

Receives from Server

If its event creation:
A Django-to-HTML5 form for creating a new competition
If event already exists:
An array of JSON serialized events

```
[{
    Name: string
    Heats: [
        Heat_number: Integer
        Dancers: [{
            Number: Integer
            Lead: String
            Follow: String
        }]
    ]
}]
```

Sends to Server

POST to studio/competition/(competition_id [0-9]+)/event/(event_id [0-9]+)
Data: CSRF Token
Response:
Activates event selected in URL
Front end should be updated by SPA framework.
POST to studio/competition/(competition_id [0-9]+)/event/(event_id [0-9]+)/round/(round_id [0-9]+)
Data: CSRF Token
Response:
Activates round selected in URL
Front end should be updated by SPA framework.
POST to studio/competition/(competition_id [0-9]+)/event/(event_id [0-9]+)/round/(round_id [0-9]+)/update_max
Data:
CSRF Token
Max: Integer
Response:
Changes max for that round.
Front end should be updated by SPA framework.
POST to studio/competition/(competition_id [0-9]+)/event/(event_id [0-9]+)/round/(round_id [0-9]+)/disqualify
Data:
CSRF Token
Couple_id: Integer
Response:
DQ’s couple.
Front end should be updated by SPA framework.

POST to studio/competition/(competition_id [0-9]+)/event/(event_id [0-9]+)/enlist
    Data:
        CSRF Token
        Couple_id: Integer
    Response:
        Adds Couple to event
        Front end should be updated by SPA framework.
GET to studio/competition/(competition_id [0-9]+)/heat_list
    Data: none
    Response:
        Returns HTML page of heat list that auto updates.

Heat List

Features

Recreates O2CM heat list which auto-updates.

Receives from Server

Array of JSON serialized events and heats in order of chronology;

```
Format:
    [{
        Name: String. Event and Heat combined.
        Heat_number: Integer
        Couples: [{
            Number: Integer,
            Lead: String
            Follow: String
        }]
        isFinal: boolean
    }]
```

Sends to Server

Nothing

Algorithm for Calling Back Dancers

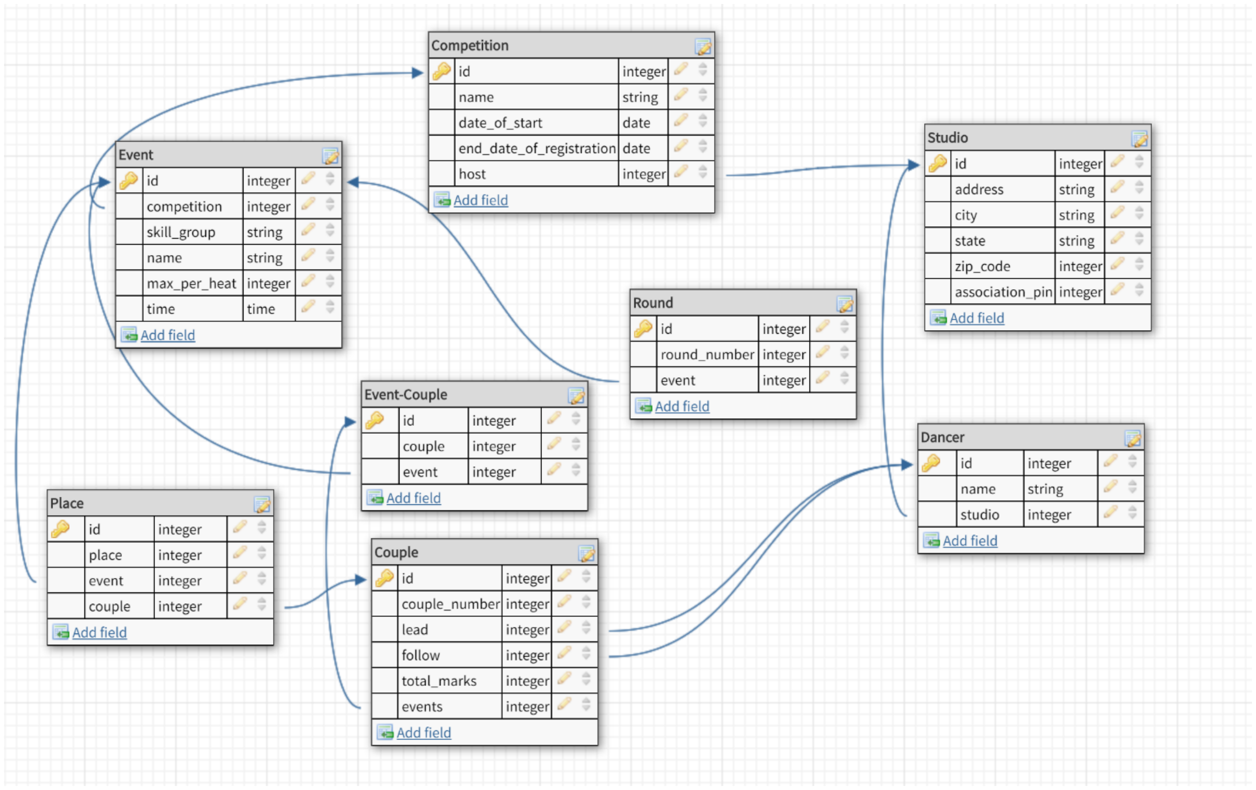
Non-Final Rounds:

Judges are given X number of marks. At the end of the round, the X number of dancers with the most marks are called back to the next round. Ties result in the tied couples being brought to the next round automatically.

Final Rounds:

Judges assign 1-X (where X is the number of finalists) to each dancer. A majority of judges must assign the same number to a dancer for the dancer to receive that place. If there is no majority, then all the 1-2s are counted up, and those with the highest are then compared. If a winner exists than that winner receives 1st place. Else all the 1-2s are added per dancer and the lower total wins first place. If a tie persists than the process is repeated, but including 1-3. Only once all X places are included is a tie declared. Then the process starts again with 2-3 to find the second place, and so on down.

Database Schema



Competition

🔑	id	integer
	name	string
	date_of_start	date
	end_date_of_registration	date
	host	integer

Add field

Studio

🔑	id	integer
	address	string
	city	string
	state	string
	zip_code	integer
	association_pin	integer

Add field

Event

🔑	id	integer
	competition	integer
	skill_group	string
	name	string
	max_per_heat	integer
	time	time

Add field

Round

🔑	id	integer
	round_number	integer
	event	integer

Add field

Event-Couple

🔑	id	integer
	couple	integer
	event	integer

Add field

Dancer

🔑	id	integer
	name	string
	studio	integer

Add field

Place

🔑	id	integer
	place	integer
	event	integer
	couple	integer

Add field

Couple

🔑	id	integer
	couple_number	integer
	lead	integer
	follow	integer
	total_marks	integer
	events	integer

Add field