Dance Card

Application specification

The Dance Card app provides an electronic version of a traditional dance card. Dance Cards normally have youth and parent information, a dance schedule, a place for an authorized signature, and dance rules and etiquette.

# User Scenarios

## Stake/Region administrator

* Administrators need to create/update dance schedules for their region.
* Administrators need to create/update information for their region, such as wards
* They manage those who have signing authoring for their stake
* There can be multiple administrators per region

## Leaders

* Be able to security sign a youths dance card for their region
* Read the schedule and the rules for their region
* Stretch: leaders can see youth information for their region

## Youth

* Create their account, update
* Read the schedule
* Read the rules
* Ask for a signature from a leader
* STRETCH: RSVP to a dance

## Parent (STRETCH)

* Who from my ward has RSVP’d.
* Willing to drive?

# Schema

|  |  |
| --- | --- |
| User | Name |
|  | Stake |
|  | Ward |
|  | Parent Name \* |
|  | Parent Phone \* |
|  | Card is signed Y/N |
|  | Expiration Date |
|  | Email |
|  | Phone \* |
|  | Auth GUID |
|  | Region Administrator y/n |
|  | RegionId |
|  |  |
| Dance | RegionId |
|  | Stake host |
|  | Theme |
|  | Location |
|  | Date/time |
|  |  |
| Region | Name |
|  | Standards |
|  | RegionId |
|  | Signing Password |
|  | Stakes [{Name: , Id:, Wards [{Name:, Id:} ]}] |
|  |  |

\*Note that we had 6 collections, but after review, were able to reduce to 3.

# Routes

**HOME**

PUBLIC GET /: EJS template that provides a link to login/logout

**USER**

PRIVATE (ADMIN/YOUTH) **GET** /USER/:<guid>

RETURNS JSON { Name:, Region Id, Region, Stake, Ward, Signed/Y/N, Phone Number }

PRIVATE (ADMIN/YOUTH) **POST** /USER

RETURNS JSON { Name:, Region Id, Region, Stake, Ward, Signed/Y/N, Phone Number }

PRIVATE (ADMIN/YOUTH) **PUT** /USER

RETURNS JSON { Name:, Region Id, Region, Stake, Ward, Signed/Y/N, Phone Number }

PRIVATE (ADMIN/YOUTH) DELETE /USER:<guid>

**REGION**

PUBLIC **GET /**REGION

[{REGION\_NAME, REGIONID}]

PRIVATE (ADMIN) **GET** /REGION/:<regionid>

RETURN JSON {Name, Standards, [{Stake Name}], [{Ward Name, Stake Name}, Signing Password]

PRIVATE (ADMIN) **POST** /REGION

{Name, Standards, [{Stake Name}], [{Ward Name, Stake Name}, Signing Password]

PRIVATE (ADMIN) **PUT** /REGION

{Name, Standards, [{Stake Name}], [{Ward Name, Stake Name}, Signing Password]

PRIVATE (ADMIN) **DELETE** /REGION:<regionid>

**SCHEDULE**

PUBLIC **GET** /SCHEDULE/:<regionid>

RETURNS JSON [{DATE/TIME, STAKE HOST, THEME, LOCATION}, …]

PUBLIC **GET** /FUTURESCHEDULE/:<regionid>

RETURNS FUTURE (today -1 ) dances JSON [{DATE/TIME, STAKE HOST, THEME, LOCATION}, …]

PRIVATE (ADMIN) **POST** /SCHEDULE

[{DATE/TIME, STAKE HOST, THEME, LOCATION}, …]

PRIVATE (ADMIN) **PUT** /SCHEDULE

[{DATE/TIME, STAKE HOST, THEME, LOCATION}, …]

PRIVATE (ADMIN) **DELETE** /SCHEDULE/:<regionid>

**STAKE**

PUBLIC **GET /**STAKE

[{STAKE\_NAME, STAKE\_ID}]

**WARD**

PUBLIC **GET /**WARD

[{WARD\_NAME, WARD\_ID}]

**OPEN ITEMS**

How do you create an admin?

# Authentication

The app will leverage Google and Microsoft OAuth 2 through Auth0.com.

# Authorization

The app will implement user roles defined in Mongo or in Auth0. Roles will be the following.

* Region Administrator
* User

# Example UI

Text, letter

Description automatically generatedGraphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generated