Conference Organizer

April 07, 2017

Overview

The Conference Organizer application is intended to allow small conference / event organizers to create and schedule ad-hoc events, sign up participants, and also allowing participants to propose and view content for each event.

Total time for completion: 3-4 weeks, 1 FTE.

Goals

1. Allow self-registration of users into the system, each user having a unique username and a password.
2. Allow events to be created, scheduled, deleted and automatically archived once the event is past. Events should have:
    a. Owners, the only ones allowed to change event time, description, venue or modify anything but scheduled content for the event.
    b. Scheduled dates and durations (eg. “starts 2019/04/03 at 10am and ends on 2019/04/04 at 5pm”).
    c. Venues: An address field where the event will occur
    d. Description: What the event is about. This can optionally include a picture - an event logo or banner or other descriptive graphic.
3. Allow existing events to have ad-hoc sessions proposed by anyone who has signed up for an event. A session should have a title, a basic description, a speaker (by default, the user proposing the event) and a duration.

4. Sessions should be scheduled into the available time for the event in a first-come, first scheduled basis. No session can overlap another session. If all available time slots have been claimed for the event, no additional sessions should be schedulable unless the event owner or a session owner deletes or changes the necessary time for a session to make room.

Specifications

1. All user accounts will be stored in an accounts database and managed by a microservice who's sole purpose is to handle user registration, lookup, and authentication.
2. All events and sessions in events will be stored in an events database and managed by a microservice who's sole purpose is to manage events.
3. User interface written in a suitable Web Framework and available as a web app which allows basic CRUD operations on users and events. An administrative super-user should also be able to CRUD any user or event, whereas only owners should be allowed to delete their own objects.
4. Services code should build with buck or gradle. Web UI should build with whichever build harness the chosen web framework prefers.
5. Services and web UI should be hostable using standard docker-compose mechanisms on a standard Linux host. For the purpose of this exercise, all 3 containers can run on the same host, though they need not do so for architectural reasons.

Milestones

1. End of Week 1 (0+7 days)

   User database and corresponding user service implemented with REST API. Self-tests allow service to be tested for all basic user CRUD operations.

2. End of Week 2 (0+14 days)

Event database and corresponding events service implemented with REST API. Self-tests allow events to be created, looked up and deleted, as well as session CRUD operations against an existing service to be tested.

3. End of Week 3 (0+21 days)

   Basic Web UI is up and running to allow users to register themselves, log in to the events service, create events, view events and delete (if owner) events. Users should also be able to register for one or more events as attendees.

4. End of Week 4 (0+28 days)

   Web UI complete. Sessions can now be registered for events by attendees, scheduled, and viewed in an overall event agenda UI. When an event's ending date/time has passed, events are also closed for further editing or scheduling.

Implementation Languages

- All microservice code to be written in Go
- All web application code to be written in Angular.js or React and Typescript

Note: If you truly wish to do so, you are welcome to use Elixir on the backend and Phoenix for the web UI, but this will be considerably more challenging