Project Notes

Assumptions – Registrant Uniqueness

It is assumed that the registering participants have already been registered if their email OR phone number exists within the database. This not only provides for a faster database (using three unique key indexes total) but also provides for a greater degree of accuracy in weeding out duplicate registrants. For example, this might be from the same company with one person trying to register everyone with singular contact information.

Assumptions – Data Values

It is assumed that the various variables in the database will be of the specified (in schema) type and within acceptable lengths. The existence and length of variables is checked in code, but information that must be stored beyond these limits is not supported. For example, the phone number +(678) 248 330 4138 x8765 would not be supported because it is over 20 characters in length at 25 total. However the sanitized value (67824833041388765) is supported. The downside is that the x for extension is lost. See the "Limitations – Input Sanitization" section for more details.

Limitations – Security

This applications includes little to no security. This is mostly a product of my relatively lacking full stack experience. For instance, all information is available through GET and POST parameters, which is not secure, at all.

Limitations – Input Sanitization

The input for an application like this can be sanitized in many ways depending on various factors. For instance, are international phone numbers supported? This alone creates many possible valid formats. Some (basic) sanity checking is done, but nothing beyond that. Depending on how much is done, this alone could require code of larger size than this entire project. Normally, the basics are done with Regular Expressions however, as it is done in this project to strip phone numbers of non-numerical characters.

Limitations – Testing

Due to my inexperience and lack of time for this project, no rigorous testing of the database or full stack was executed. Normally, PyTest, JUnit, and/or Selenium should be used to rigorously test such an application. Testing was however conducted for all aspects as the application was developed.

Strengths – JQuery and Bootstrap

JQuery and Bootstrap represent excellent strengths of the chosen application. This is because JQuery is excellent for gathering information and Bootstrap is excellent for responsive and well designed layouts, particularly for mobile devices.

Weaknesses – System Design

As I am new to most full stack technologies, I am unsure if the design follows standard best practices. From what I learned online I think it does follow standard design patterns but I cannot be sure without the ability (since it is an individual assignment) to code review with subject matter experts.

Scalability - MySQL

The project (as designed) is quite scalable in size. Intentionally few MySQL calls are executed from code to limit database access delays. Since the database can be access concurrently (up to 131 simultaneously) from multiple served copies of the website, it is unlikely that server overload could become an issue except in the most extreme cases. I doubt this case could ever be reached with a maximum of only 5000 participants. For such cases, utilizing a split/mirrored database and/or a more robust database technology in future might be prudent, ie – PostGreSQL, MSSQL, Oracle, etc

Scalability – HTML/JavaScript/CSS

I am new to the "full stack" as it is called, so I can't speak (very well) to the scalability of the web stack, however the scalability of the site would largely be dependent on the traffic level and the system specs of the web server. Also, a more experienced full stack engineer could probably increase the speed with which the HTML and JavaScript is executed/arranged.

Design Decisions and Future Adaptability – Flask vs Django

I chose Flask because (for the given use cases) it was the right balance of easy, simple, and stable. The other option is the much more robust Django. Plans for this project to integrate into a larger ACME Summit portal (perhaps a website for the entire conference operations) might warrant the migration to Django. This would require significant reworking of the routing code.

Design Decisions and Future Adaptability – Database

The database (in MySQL) is designed for the use cases provided only. Because the data provided for these use cases is not related to other records, a single table is the most efficient design. Using the Database (even though it isn't really needed) grants faster performance time anyway. In future, if the project were to include rows that do need to relate to each other, for instance which attendees are presenting

sessions, then additional tables would need to be created and the current "Registration" table might need to be re-worked or migrated.

Design Decisions and Future Adaptability – Users and Permissions Levels

Currently, the organizer login functionality is implemented with a username and password hard coded (user:"bugsbunny" pass:"whatsupdoc") into the Python Source Code. This is terrible practice but works for the use cases provided. In future, it should (more properly) be implemented with a users table or database to allow for different levels of access to information, as well as hashing and securely handling the username and password fields. This login functionality would also be required for any further conference portal like application features to be added, like RSVP or room scheduling, etc.

Installation

Dependencies

Python 2.7+ but NOT Python 3+ as Flask is not happy with Python 3+ yet
The following Python packages are required:

from flask import Flask, render_template, request
from flaskext.mysql import MySQL
import re
import csv
import MySQLdb
import getpass
import os
import time

JQuery (provided)
JavaScript files (provided)
HTML5 files (provided)
CSS files (provided)
Bootstrap (linked in to files)

Installation Steps

To install, first install and meet the dependencies listed above, then:

1. Create or use an existing account on a MySQL server
2. On said server, run the provided createdb.sql script.
3. Run the app.py file from the Python shell.
4. When the program prompts, input the location, username, and password of the appropriate MySQL server.