Design Document

CS 4503

Heartland Gaming Expo

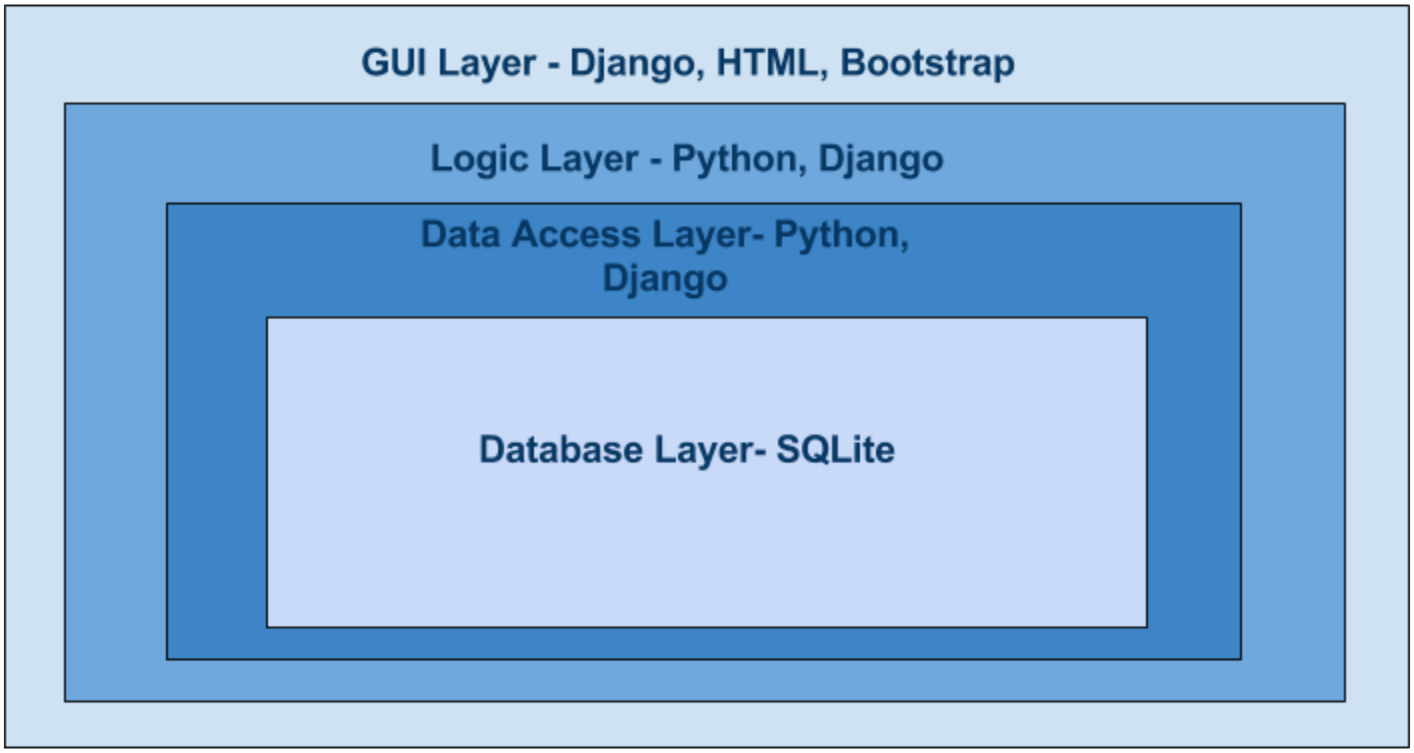
Team Gambit:
Bailey Smith, Brooke Peterson, Tyler Matthews, Ethan Robards, Aziz Al-Mahfoudh

Table of Contents

1. Introduction

This document gives an overview of the architecture and module relationships for the updated Heartland Gaming Expo system defined in the Phase 1 Gambit SRS. The document primarily makes use of diagrams and clarifying text where necessary to present its information. The intended audience of this document is members of the University of Tulsa who will be setting up and managing the Heartland Gaming Expo, however, presented information is in a format which is understandable to anyone with a limited background.

2. Architecture

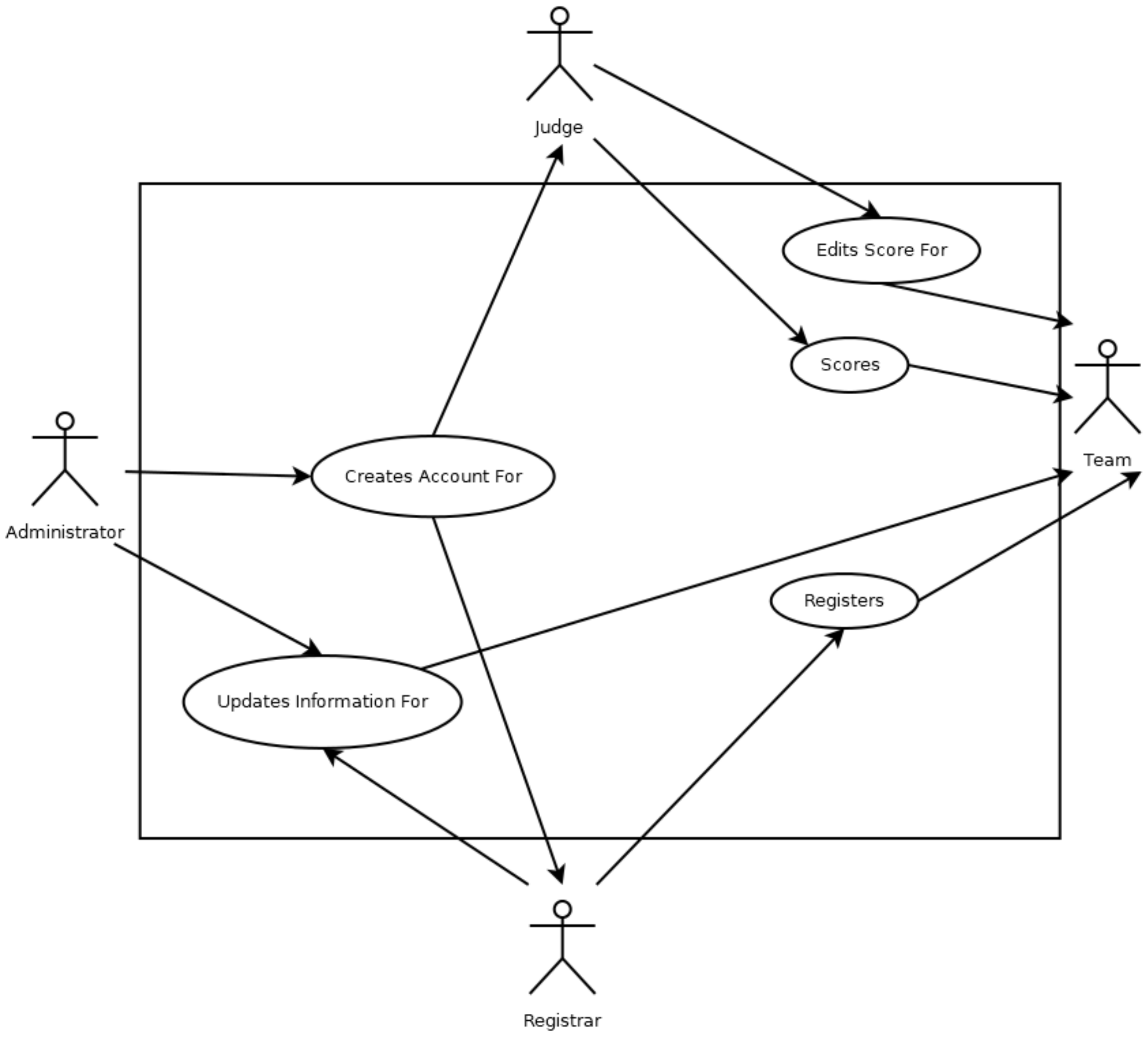

GUI Layer - Django, HTML, Bootstrap
Logic Layer - Python, Django
Data Access Layer- Python, Django
Database Layer- SQLite

2.1. Introduction

The layered model will be used to represent high level software architecture design. This architecture was chosen because the division of the system based on the services each layer provides made sense for the project implementation in supporting the incremental development of subsystems. The highest level, the GUI, will provide functionality for users to interact with the data they have permission to access. The logic layer will perform translation of data between the data access and GUI layers using functions which work between the two layers. The data access layer will use python and django scripts handle selection, data manipulation, and storage of data which is sent to and from the final layer, the database layer. The database layer will utilize SQLite to store tables of scoring and judging information.

2.2. Modules

2.2.1. GUI Layer



This layer will be the layer where the users, registrars, admins, and judges, will interact with the system. In this layer, a judge will be able to score or edit scores for a team. An administrator will be able to create user accounts in this layer as well as update team information. The registrars will be able to register and update information for the teams. This information will be sent through the layers to be stored in the database. The GUI layer will be created using a combination of Django, HTML, and Bootstrap. The use case diagram above summarizes the functions the GUI will have to support for the users.

2.2.2. Logic Layer

This layer will act as an intermediate link between the data access and GUI layers. High level commands given by the user within the GUI layer are translated to lower level commands to the data access layer. This translation can be achieved by calling functions between two layers. Once data is retrieved from the data access layer, decision-making and calculations are performed as required by user commands. Finally, the result of the processing is returned either to the GUI layer, if data was to be retrieved, or to the data access layer, if the user requested changes to be made to the database. Both Python and Django can be used to implement this layer. Access to the GUI layer can be implemented using Python, while accessing the Data Access layer can be done using Django.

2.2.3. Data Access Layer

The data access layer will have the functionality of receiving and manipulating data from the logic layer as well as the database layer. This layer will be responsible for the translation of data from the SQLite database's tables into objects which can be acted upon by the logic layer and through the logic layer, the GUI layer. Said translation will occur with any modifications to surrounding layers in order to preserve and store those changes. Implementation will be executed through the Django framework and Python scripts.

2.2.4. Database Layer

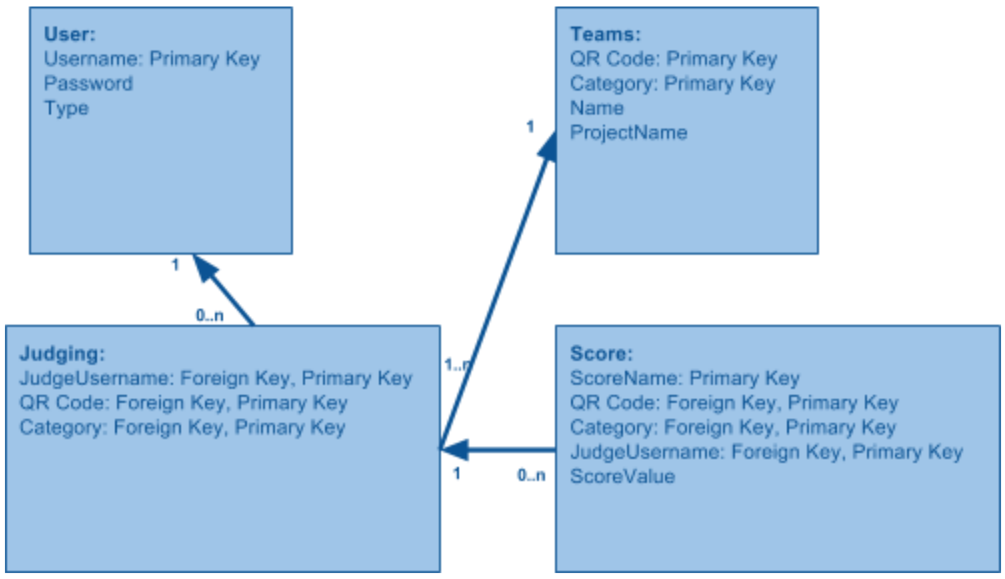
The persistent data is stored in the database layer which uses SQLite. SQLite is a relational database which allows us to relate the different tables to each other. The relations of tables are discussed in more detail in section 3.1.1 where a database schema is given. The database will be incorporated into the Heartland project using Django.

3. Class Diagrams

3.1. Data Table Classes

For storing persistent data, our implementation will be using an SQLite database. The data that will be stored in the database includes the user information, judging information for each team, the teams, and the scores given to the teams by the judges. Our database schema is outlined below showing the different tables for users, teams, judging, and scores, and the relationships between the tables. Also specified are the primary keys and foreign keys for the tables.

3.1.1. Schema


User:
Username: Primary Key
Password
Type
Teams:
QR Code: Primary Key
Category: Primary Key
Name
ProjectName
Judging:
JudgeUsername: Foreign Key, Primary Key
QR Code: Foreign Key, Primary Key
Category: Foreign Key, Primary Key
Score:
ScoreName: Primary Key
QR Code: Foreign Key, Primary Key
Category: Foreign Key, Primary Key
JudgeUsername: Foreign Key, Primary Key
ScoreValue
1
0..n
1
1..n
1
0..n

3.1.2. Schema Information

User:

The user table includes the login for the user, username being a unique key for each user, password, as well as the type of the user: admin, registrar, or judge.

Teams:

This table includes the name of the team as well as their project's. The QR code is unique for each team and will be used for judging and scoring purposes. While the category is not unique for each team, it will be referenced by the judging and scoring processes, and thus it will serve as a primary key.

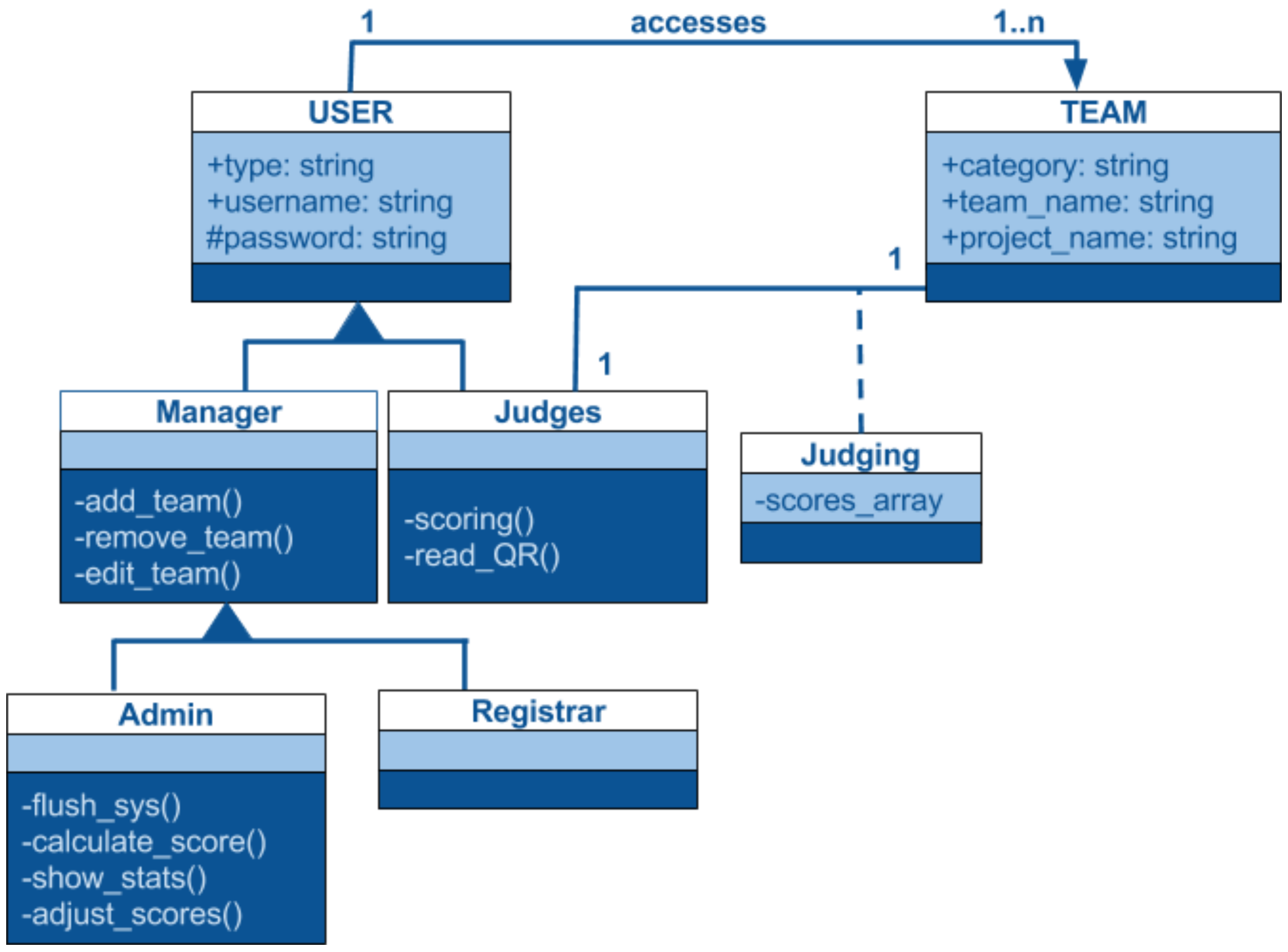
Judging:

For the judging table, all the attributes serves as both foreign and primary keys. JudgeUsername is a unique key for each user, it references the user table for the username. QR code and category are used to identify the team being judged, as well as being referenced as an identification for what the user has judged.

Score:

This table includes ScoreName, a unique name assigned to each scoring process. QR code and category are used to identify the team and project category. JudgeUsername is used to identify the user who judged the project. Finally, ScoreValue is the value assigned by the user to the project.

3.2. Class Information



User:

The user is a superclass having general attributes for all users of the system. Type of the user is a string that specifies which subclass it belongs to. The username is a string representing the unique identifier of the user. Password is also a string, but a protected attribute of the user. A user also has access to multiple teams.

Team:

Team consists of public attributes all of which are strings. It includes the team name, the project name, and the category of the project.

Manager:

Manager is a subclass as well as being a superclass itself. It includes functions related to team management.

- *add_team()* adds a team to the database.
- *remove_team()* removes a team from the database.
- *edit_team()* edits a team and saves changes to the database.

Judges:

Judges is a subclass of the user and includes the exclusive functions of scoring() which assigns a score, and reading QR codes of the teams. Score assignment functions as a relation to the Team class, with the relation having one attribute, the value of the score.

Admin:

Admins is a subclass of the manager and it has exclusive functions.

- *flush_sys()* completely erases all the data in the system.
- *calculuate_score()* does all necessary calculations for scoring and determines the winners.
- *show_status()* shows the current status of scores, teams, judges, and all the current relations between them.
- *adjust_score()* enables the admin to edit scores based on a scheme to avoid bias or other related issues.

Registrar:

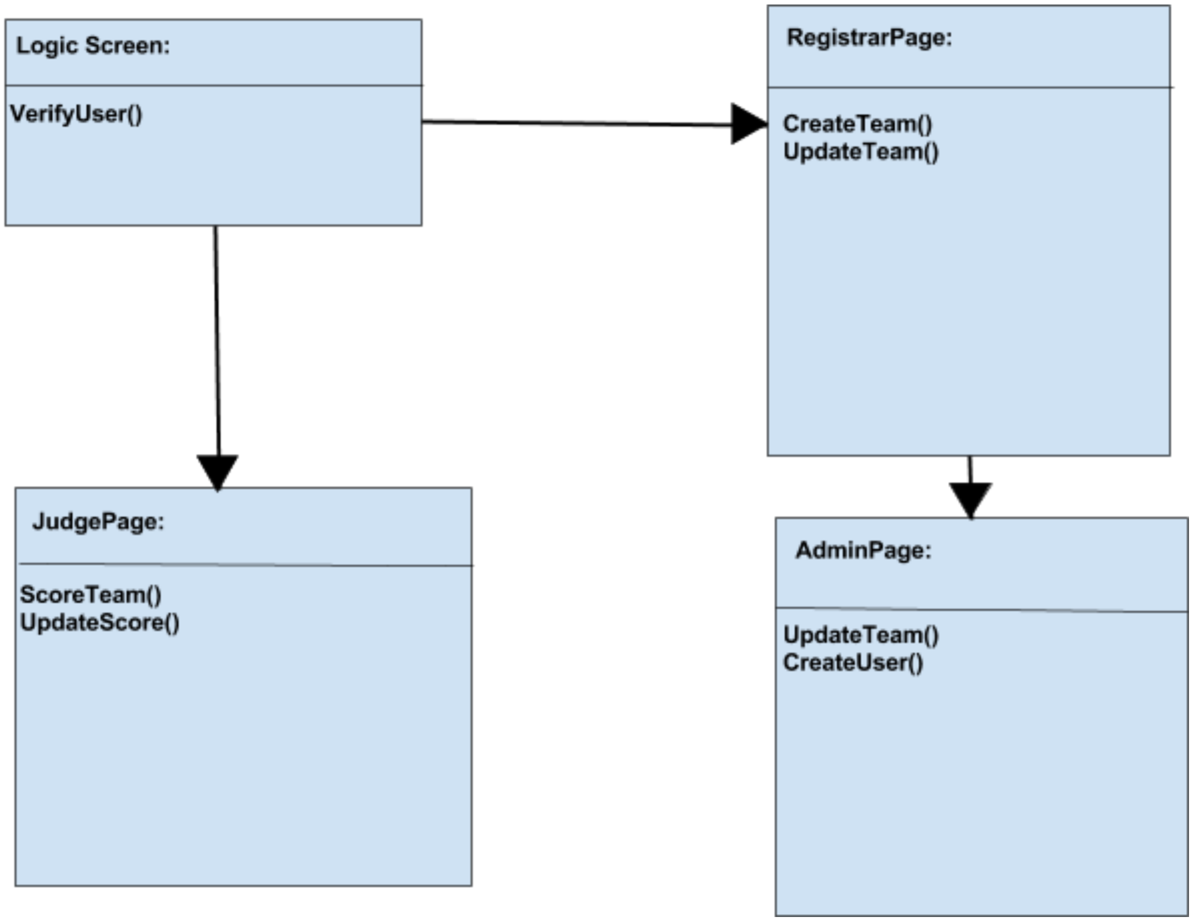
While registrars inherit everything from the Manager class, they have no exclusive attributes or functions.

3.3. GUI Layer

The GUI Layer will be comprised of three modules, each corresponding to a different type of staff: administrators, registrars, and judges. Administrators will be provided an administrative site through Django's built-in capabilities, which will display all models and allow the admin to create, update, and delete any models/model fields, as well as specify permissions for users (for example, creation permissions for team models should only be delegated to registrar users and the administrator). The data entered into the administration site will be prescreened by the logic layer before being committed to the database. The administrator will also want to display current/final winners for awards, which can be scripted through Python and ran through a django-admin shell. All website styling will be automatically generated for the administration site, and any additional form widgets implemented in the actual website will also be used in the admin site, for ease of use.

Registrars will login with a registrar account on a website through a judge/registrar login page, which takes them to the team check-in page. The team check-in page lists all pre-registered teams, and will have all non-checked-in teams displayed. If a team shows up and checks in, the registrar can simply press a button associated with said team which takes them to a check-in confirmation splash page. This double-checks if all the information is correct, then if the registrar and team member(s) both agree the information submitted is accurate, the registrar can commit the team to the database as checked in. Registrars can contact the administrator to remove a team from the checked-in list and/or edit team names or team-category associations to correctly register teams for the correct categories/spell team names correctly. Registrar accounts will be created by Administrators and usernames/passwords will be distributed by the administrator just before use.

Judges use the same login process, but through an android app. A list of unjudged team-category associations (and their location in the venue, possibly a numbering system paired with a map) will appear upon successful web login that they are qualified to judge. This is available immediately after registration closes. Login credentials will be randomly generated and stored on each android tablet. Then, the user can go offline and start judging teams by scanning printed QR codes. Each scan will bring up the team category's metrics if the logic layer verifies that both the judge and the team are in the category designated by the QR code. The scores are then entered and saved locally upon completion. There will be no confirmation splash page as judges will be able to edit their past scores until the competition ends, accessible through a separate menu much like the unjudged teams menu. On both of these menus, an "upload" button will be present, allowing the judge to commit their current scores to the database. When the competition finishes or the judges have finished their judging, the administrator will cut off access to editing the scores as well as auto-upload all locally stored scores to the database so that the django-admin python shell scripts can be run to compute results.


Logic Screen:
VerifyUser()
RegistrarPage:
CreateTeam()
UpdateTeam()
JudgePage:
ScoreTeam()
UpdateScore()
AdminPage:
UpdateTeam()
CreateUser()