CS-411 (Section QG)
Project Track-1

ARC Management System Project Report
Team #28 (Team SRKC)

Table of Contents

Team Details

- Team Name – SRKC
- Team Number – 28
- Team Members

Net Id	Name	Email Id
csaraf2	Chinmay Nandkishor Saraf	csaraf2@illinois.edu
takwane2	Kedar Mukund Takwane	takwane2@illinois.edu
saketsj2	Saket Jajoo	saketsj2@illinois.edu
rg18	Rishabh Garg	rg18@illinois.edu

Project Details

Project Summary

This project is a management system which provides a comprehensive view of the facilities at the Activities and Recreation Center at UIUC. The system has an interactive web User Interface which will allows the students to book courts for sports and equipment (gym or otherwise) on a slot-wise basis. This process also optimizes the booking of the sports equipment as well as the locations (e.g. courts, pools, etc.) so that there isn't a burden on one sport. Admin interface of the application has all the booking options available like users. Furthermore, the admin also has option to create sporting events (for example a football match at the Memorial Stadium).

Detailed Description

The ARC Management System sorts out the entire booking process and help optimize the usage of the facilities. The system has multiple components to it like the users, sports, courts (or locations), booking and equipment. Only the students studying at UIUC, and the staff can make a booking. The sports category lists all the available sporting activities (including the gym) available at the ARC. This category also includes various gym facilities like the sauna or the spa room. Based on the number of slots available, students can book that sport (or gym) along with its associated court (or location). The system also allows students to rent out equipment like badminton shuttles, table tennis bats, etc.

Another page in the system enables admin to create an event and make it available for booking for users. This interface allows the administrators to perform CRUD operations on the back-end tables (users, sports, equipment, locations, sporting events, slots, etc.). In the future, we also plan to integrate the existing system with Illini Cash which can allow for a seamless booking experience for the UIUC members.

Detailed Functionality Description

The ARC Management System sorts out the entire booking process and help optimize the usage of the facilities. The application has two sets of users- administrator and the facility users.

1. Administrator –
   This type of user will have the access control rights to perform the following actions:
   a. Adding, updating, and deleting users in/from the system (with the current scope of project, as we have not integrated the system with university's database).
   b. Adding, updating, and deleting the slots for the court's booking.
   c. Renting out the sports equipment to the users and adding their details to the system. Updating the record once user returns the equipment.
   d. Creating events (like a football game).

2. Users –
   This type of user will have access to perform the following actions:
   a. Booking slots for courts.
   b. Book/rent equipment for sports.

Usefulness

1. User based views
   a. The ARC Management System provides user-specific views. The administrator will be able to see and perform functionalities like managing users, sports, equipment and slots for badminton, volleyball, sauna, etc. Second, the users of this application will be able to book slots for various sports and equipment.
2. Slot booking
   a. It will allow the users to view the available slots for different sports like badminton and basketball and book them. The web app will provide this information which will help many users schedule their activities better and save time.

Realness

Our ARC Management Database will have data related to Sports, Equipment, Slots, and Users. We distribute the data as follows-

1. Slot booking- 70% synthetic and 30% application generated. We have used a python script to generate data and we have ensured its validity by creating slots that are an hour each. Furthermore, we also guarantee the slots will lie within the ARC operational hours.
2. For Sports, Equipment and Room tables we visited the ARC to get the information of the data.

Finally for Users table, we have used the Twitter Username Alias Dataset (https://www.usna.edu/Users/cs/nchamber/data/twitternames/). We have randomly assigned these users roles- users, administrators, and advisor to each user in the dataset.

Project Setup Instructions and Demo

Project Setup

There are 3 steps to initialize the project and running it as a web application (frontend + backend + database connection):

1. Frontend: In the Frontend/ directory run
    a. *npm install* - This would install the npm/react packages and compile the frontend code
    b. *npm start* - to start the frontend (this would serve the webpage on port 3000).
2. Backend:
    a. *pip3 install requirements.txt* - This has python-based dependencies that would be required to server the APIs. Install these dependencies using
    b. *python3 -m src.server* - To create the Flask server, execute from the Backend/ directory itself. This will spin up the Flask server on port 5000.
3. Database: Setting up the database is a bit tricky. Although we have database files (*.sql) in the database/ directory, if one needs to create this file, the steps are mentioned in the 4th point. The current point only covers the creation of a database that already has been populated. To do so, go to the database/ directory and select the latest .sql file (based on date [db-YYYYMMDD.sql]). At this point in time, it is "db-20221125.sql".
    a. Login to mysql shell and execute: *source db-20221125.sql*. This will create the database (called 'SRKC' [our team name]) and will create and populate various tables. The schema is present in the sql/DDL.sql file in the database/ directory.
4. Creating the .sql file: Once the database called 'SRKC' has been created (manually) navigate to the database/scripts/ directory, tables needs to be initialized and the static data must be loaded. To do so, navigate to the database/sql/ directory and execute:
    a. *source DDL.sql*
    b. *source static_data.sql*
5. To populate the synthetic data in other tables, navigate to the database/scripts/ directory and execute: *python3 main.py*. This will create, initialize and populate the tables in the 'SRKC' database. Once this process is complete, the database can be exported into a ".sql" file using:
6. *mysqldump -u <DATABASE_USERNAME> -p<DATABASE_PASSWORD> --databases SRKC > db-$(date '+%Y%m%d').sql*. This will create the 'db.YYYYMMDD.sql' file that has been referred to in point 3.

Project Demo Video

https://drive.google.com/file/d/1xQHQJCbAfXUmUguyhuHygJMdyi5nenTr/view?usp=sharing

Reflection Report

Difference of final project from the original proposal

- There are no changes since the submission of the original proposal.
- Our design in the Stage-2 had accounted for all the use-cases that we planned to implement.
- After the final stage submission, we can say that we had modelled the schema to cover all the functionalities, so we didn't need to make any changes to the design after stage 2.

Application achievements and failures related to its Usefulness

- Our web application aimed at all the critical functionalities of ARC like user-specific view dashboard, booking a facility, renting equipment, booking events, and more. In retrospect, we were able to implement all the functionalities that are deemed to be useful for both Administrators and Users. In fact, towards the end we implemented several statistics like average number of people visiting the ARC per weekday, which could be useful for the management team to know busy days. Based on this information they can decide when to have more workforce.
- Moreover, another statistic related to most revenue generating events could give an insight into popular events. This information can help them plan a much bigger event in a bigger space. So, we incorporated such statistics which were initially not a part of our scope. Thus, improving the overall usefulness of the system.

Changes in schema or source data for the application

- The schema was finalized during Stage-2, which included DDL commands for creating all the tables. However, based on the requirements of Stage-5 (implementation of triggers and stored procedures), for triggers specifically, we needed to update our *Users* table to include the trigger in the schema itself. Only the new trigger as added to the table's definition and rest of the schema- attribute names, data types, keys, and indexes, remained the same.
- As part of Stage-3, we generated synthetic data using a python script, but the source of data did not change. We used this dataset to populate the 'Users' table - https://www.usna.edu/Users/cs/nchamber/data/twitternames/. Overall, there was no change to the source data in general.

Changes in ER diagram/table implementation as compared to the original design

- During Stage-2, we approached our project using a UML diagram instead of an ER diagram and we put in much thought and time into designing the architecture of our system. And things went as we planned so we didn't need to change our initial design in the later stages of the project.
- Having said this, at times we felt that having a couple of additional attributes could have decreased the overall complexity of the queries we wrote for fetching some of the statistics mentioned previously in the usefulness section. Although, the queries became a bit complicated, we managed to implement them using the original schema.

Functionalities added or removed

- The ARC management system provided several functionalities from User summary to display personal information and past bookings, to ARC revenue statistics and booking insights.
- Due the lack of time we could not implement the staff-related functionalities like management of on-duty staff members that would be useful for the administrator to track their schedules, working hours, payroll, and so on. For these functionalities we had the backend ready but not the User Interface. So, we pushed these features as part of the future scope.

Advanced database programs complement the application

- Advanced database programs such as stored procedures, and triggers are convenient to execute a sequence of actions.
- In our ARC management application, we wrote a stored procedure to extract details related to a user. It included user's basic information like contact information, birthdate, etc. to fetch details like which facilities were booked by the user and when. Moreover, we also calculated how much time did the user spend on a sport and money on renting equipment and events. This helped us with the creative component portion of Stage-5.
- This provided valuable information for the management staff to get insights into who is regular at ARC and maybe they could use this information to target advertisements to a specific group of people or something on similar lines. Additionally, it increased flexibility in our application and provided better performance overall.

Technical challenge encountered by each team member

Challenge faced by Rishabh (rg18):

- For the development of stored procedures, we faced a challenge in returning two separate tables as an output. We wanted to fetch these two tables for updating tables in the user interface. However, the first table that was returned was always over-written by the second table.
- To overcome this challenge, we created a cursor over the base table (the table which derived the first output table) and updated a temporary table after a matching condition. The output was the temporary table that had aggregate information from the first table and data from the second. This was a quick workaround.

Challenge faced by Saket (saketsj2):

- For implementing the API, flask's restx framework (https://flask-restx.readthedocs.io/en/latest/) was used. One of the advantages of it is that provides a swagger UI to visualize all the written APIs. There were a couple of challenges to implement the API at the earlier stages like connecting to the database and refactoring the API code to create different namespaces for each logical group of endpoints. Earlier all the API code was written in a single file which made the API look monolithic and difficult to understand and

modify. Refactoring the API code into modules consumed some time but was successfully implemented which made it super easy to add/update endpoints as and when needed.

- Furthermore, hosting the application on GCP was something that we couldn't wrap our heads around. We thought to host the DB in a private VPC that would only be accessible to the application (GCE VM) to make our system more secure (with respect to access at the network layer). However, we encountered connectivity issues that we could not resolve in time.

Challenge faced by Chinmay (csaraf2):

- For the creative component part of Stage-5, we displayed a bar chart which indicated the number of tickets booked on each day of the week. The idea behind this was that it would help the administration team know which days of the week are busy so that they can accordingly adjust their workforce.
- We are using ReactJs for front-end and initially we tried implementing https://devexpress.github.io/devextreme-reactive/react/chart/ library but we kept getting errors while trying to embed data fetched from our API. So, we decided to go with the https://recharts.org/en-US/ library which then worked.

Challenge faced by Kedar (takwane2):

- While starting the development phase of our project, we decided to use Redux (https://react-redux.js.org/), which is an open-source JS library for managing the state of our application. It is kind of a central storage space which can be accessed from all the pages of the web app.
- The initial process to set it up was a little time consuming and tedious, but after it was setup things went smoothly.

Any changes comparing final application with original proposal

- As mentioned previously, due to lack of time we could not implement the staff-related functionalities like management of on-duty staff members that would be useful for the administrator to track their schedules, working hours, payroll, and so on.
- On the other hand, our project met all the other requirements especially the ones related to usefulness.
- The User-Interface (UI) was upgraded to include dynamic functionalities such as dashboard for user and ARC statistics while keeping the basic operations same as proposed.

Future work

- Integrate IlliniCash with the ARC management system for renting equipment and booking tickets.
- Having coupon codes (discounts / schemes) for our most popular customers (from whom we get maximum revenue).
- Onboarding referral system for event booking and club membership.
- Expansion of the concept to the larger system of University of Illinois i.e., Springfield and Chicago campus.

- Although, we figured out the network connectivity issues in GCP, we couldn't host our application on cloud. This is a ‘good to have’ feature that can be implemented in future.

Division of labor

- All team members contributed equally towards designing database architecture for the project. The entire project was a collaborative effort and there was high synergy among the team members in achieving the deliverables of each stage.
- Chinmay (csaraf2) and Kedar (takwane2) were responsible for building the front-end for the system using ReactJS.
- Saket (saketsj2) and Rishabh (rg18) were responsible for architecting the backend. More specifically, Saket worked on building the APIs (using Flask/Python). All end-to-end SQL queries for database were written by Rishabh including advanced queries for stored procedure, trigger, and statistics.