**Tournament Management Software for UA Badminton Club**

Status Report 2

*Junfeng Xu, Enfa George, Rupal Jain, Urvika Gola*

# **Table of Content**

[**Table of Content 2**](#_4z7ivippkriu)

[**1. Introduction 4**](#_ax0u1pmm5z55)

[Highlights 4](#_vkshb35u0h6s)

[Plan for this iteration 4](#_8u7cjpfh62bu)

[Team accomplishments for this iteration 4](#_euiye8rkhd27)

[Team accomplishments according to the plan: 4](#_8v4oguyy4bem)

[Team other accomplishments beyond the plan: 5](#_3csvkrp6crgi)

[Changes 5](#_k9yx1m55l0im)

[**2. Customer Need 5**](#_vsfrgjxbmkd4)

[**3. Project Goals 5**](#_n4zyenl8zwb1)

[**4. System Description 6**](#_iksdgrw8wxyr)

[System Overview & Main Challenges 6](#_djo90daf0b1s)

[Rationale for the Design: 6](#_mfs7ogfq2g3w)

[Context Diagram and External Interactions: 7](#_n1duo3604pqx)

[Guides to the Main Architectural Views 8](#_keekaftibvih)

[Logical View: 9](#_pllswy782c25)

[Primary Diagram: 9](#_pszzqb3z2b8d)

[Element Catalog: 9](#_ikgxjjytyz22)

[Development View: 11](#_ormtj29qgw22)

[Primary Diagram: 11](#_jguzgfry22zw)

[Element Catalog: 11](#_xqk7vjievcd6)

[Dynamic Views 13](#_65mkdkucj1xx)

[Primary Diagram: 13](#_rc09d8huax3l)

[Element Catalog: 13](#_xfcsfzian90x)

[Deployment Views 14](#_wa69zg7oejay)

[Primary Diagram: 14](#_jk7jar6ir3yl)

[**5. Current Status 15**](#_laq79r5bvjjj)

[Goals for this Iteration: 15](#_6jj0ry53znue)

[Screenshots of What’s Working: 15](#_xmd23kwqko2b)

[Public View web pages: 15](#_i5k61bpqrz0p)

[Login and Sign up functions: 19](#_suunfwydsb4)

[Player Dashboard: 20](#_gwhazdg8iq6n)

[Admin Dashboard: 21](#_gw6zkiq2v5xn)

[Database Setup: 25](#_zbzg0rfg0kdn)

[Challenges we are facing: 28](#_xv2va89qxism)

[Challenges with the database setup: 28](#_nn2em0feupxy)

[Challenges with the tournament bracket implementation: 29](#_11m7akrciza8)

[Block diagrams for the current state of the system: 29](#_j3zxmrmbnmmr)

[System overview block diagram: 29](#_vxcn1dwkwm4b)

[Modular block diagram: 30](#_z4sy2loxmi68)

[Registration module: 30](#_j6ymqkwfwue6)

[User Management Module: 31](#_gl6uu3eth8ek)

[Seed & Draw Module: 32](#_5zw3ekw4co0r)

[Match Management Module: 32](#_xs1qbcxoihl1)

[Test Performed: 33](#_fyxox59ltq6v)

[Total number of lines of code we collectively written: 33](#_zffwirc2u8xk)

[**6. Project Management 34**](#_l7pbjq149c61)

[Change Log 34](#_ersopnc7k1sk)

[Goals for next iteration: 38](#_yvkz04zfstf1)

[Goals for rest of the semester: 38](#_2zo6llj2dbxp)

[**7. Team 39**](#_mj157npc0orp)

[Team roles for this iteration? 39](#_cedx068brotp)

[Team members contribute? 39](#_5htcg6psdte1)

[Rough estimates of each team member contributions to the project (in percentage) 40](#_vqhnpx6g521u)

[8. Reflection 41](#_1ki6qz1znjej)

[What went well? 41](#_jxuvnfhgsorz)

[What didn't go well? 42](#_alox7b1fimh9)

[Goals that were not met, what were the issues? 42](#_k1d3zzn3ldfv)

[Plan to overcome the issues? 42](#_cef8dqctyzy)

[Plan to do differently in the next iteration? 42](#_pfyish6bib8f)

# **1. Introduction**

## Highlights

### Plan for this iteration

* Completing the backend support and integrating it with the front end UI for the user login/signup functions.
* Completing the database setup on both local and a distributed server and database tables creation.
* Completing the player dashboard functionalities including the player dashboard UI web pages, the backend support for player dashboard, retrieving and storing profile information from and to the player database table, retrieving and storing event registration information from and to the event database table.
* Performing functional testing, unit testing and user testing against the functions mentioned above.
* Meeting with the customer for a showcase and collecting feedback.

### Team accomplishments for this iteration

#### Team accomplishments according to the plan:

* We completed the backend support and integrated it with the front end UI for the user login/signup functions. Test has been performed to verify the user login and signup functions.
* We completed the database setup and database table creations. Test has been performed to verify that we can retrieve from the data tables and store data into the data tables.
* We completed most of the player dashboard functionalities including the player dashboard UI web pages, the backend support for player dashboard, retrieving and storing player information from and to the player database table. The event registration portion of the player dashboard remains to be completed. Test has been performed to verify that a player user account can use it to display their current profile and to modify the profile.
* Functional test has been performed (refer to the bullet points above). Unit testing has been performed during the implementation of each function. User testing has been performed during the customer review meeting.
* Customer review meeting was conducted for a showcase on 4/2/2023

#### Team other accomplishments beyond the plan:

* We completed the front end support for all the public view UI web pages. Unit testing has been performed accordingly.

## Changes

There has been no change made since Status Report 1. Please refer to Status Report 1 for more details.

# **2. Customer Need**

No change has been made to the customer's desired overall experience since Status Report 1. Please refer to Status Report 1 for more details.

# **3. Project Goals**

No change has been made to the project goal since Status Report 1. Please refer to Status Report 1 for more details.

# 

# **4. System Description**

## System Overview & Main Challenges

No change has been made to the system overview and the main challenges since Status Report 1. Please refer to Status Report 1 for more details.

## Rationale for the Design:

The MVC architecture was very attractive to us due to the following reasons.

1. **Separation of concerns.**  
     
    We wanted to ensure that the functionalities implemented, changes made or needed at any level were independent of/ had minimal dependency on other components, inshort separation of concerns.
2. **Easily modifiable.**   
     
   The focus of the project, especially given the time frame, was to have working software that can meet the customer’s needs and requirements. In future, they may want to say give the website a new look to match their branding, or want to introduce new subevents. We wanted to make sure that such requirements could be met with minimal overhead.
3. **Faster Development Process**The separation of views, from models and that from controller ensured that we are able to build separately and faster.

### Context Diagram and External Interactions:

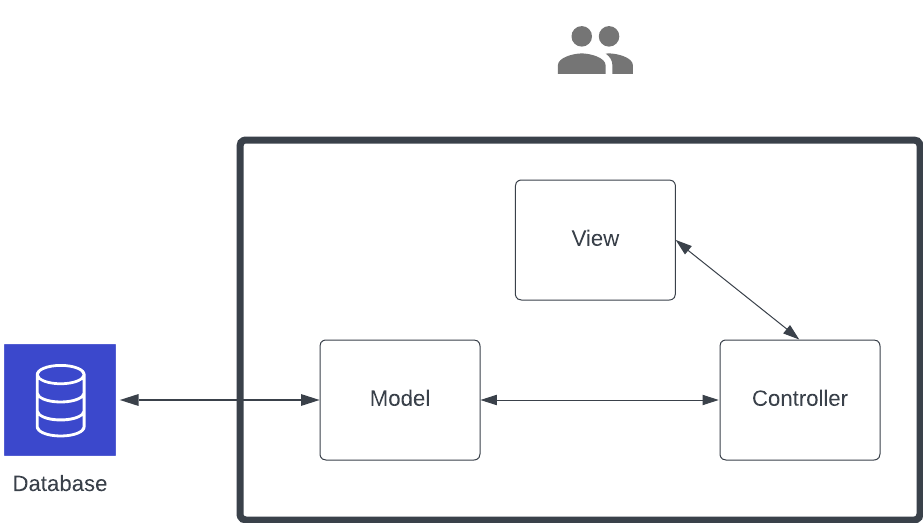


Figure: Context diagram for the application. The thick black border marks the boundary of the system. The system interacts externally only with the database.

**Model:** The Model represents the business logic and data of the system. It will be responsible for storing and retrieving data from the database, as well as the processing and manipulating data. The Model will interact with the database layer to store and retrieve data.

**View:** The View represents the user interface and presentation of the system. It will be responsible for displaying data to the user and receiving input from the user. This is a web interface that the user can visit at a given URL. This is independent of any implementation logic.

**Controller:** The Controller acts as an intermediary between the Model and the View. It will receive input from the View and pass it to the Model for processing, and it will receive data from the Model and pass it to the View for presentation.

**External Services:**

**Database:** This stores all relevant data needed for the application to perform well. The database interacts with the Model only. The database is hosted using AWS cloud services.

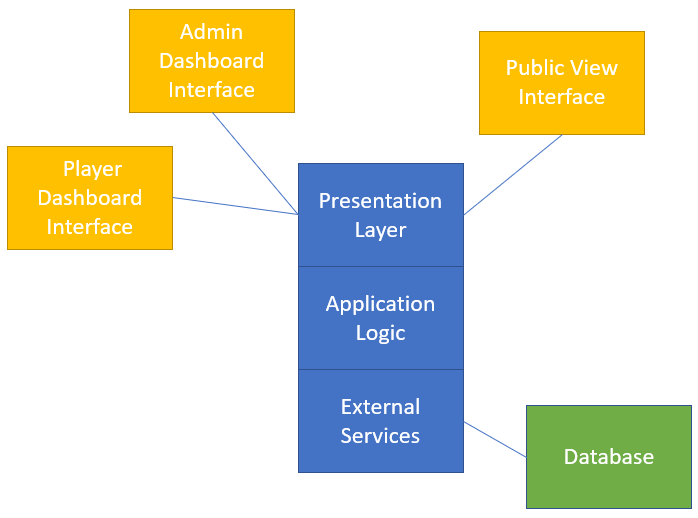
The boundaries of the system are clearly defined, with the Model representing the business logic and data, the View representing the user interface and presentation, and the Controller acting as an intermediary between the Model and the View.

## Guides to the Main Architectural Views

We have employed a logical view and a development view for the architecture for our system.

### Logical View:

#### Primary Diagram:



#### Element Catalog:

Admin Dashboard Interface (Owner: Rupal): this provides the interactive web pages for the tournament organizer user which includes functionalities including tournament management, player roster management, Seeding and drawing management, and match management. The tournament organizer may edit and publish a tournament, edit player information, assign seeding to event entries, make draws for each event, and operate with the matches on the tournament day.

Player Dashboard Interface (Owner: Junfeng): this provides the interactive web page for the tournament player users. A player may edit their profile information and register for the events they want to participate in for the current active tournament.

Public View Interface (Owner: Junfeng): this provides the static web pages for the general public (tournament spectators, tournament participants, etc) to view all the tournament related information including tournament details, announcements, players, event brackets, match progress, and much more. Authorized users may login to the system on this interface.

Presentation Layer (Owner: Enfa): this takes care of the rendering for all the web page interfaces.

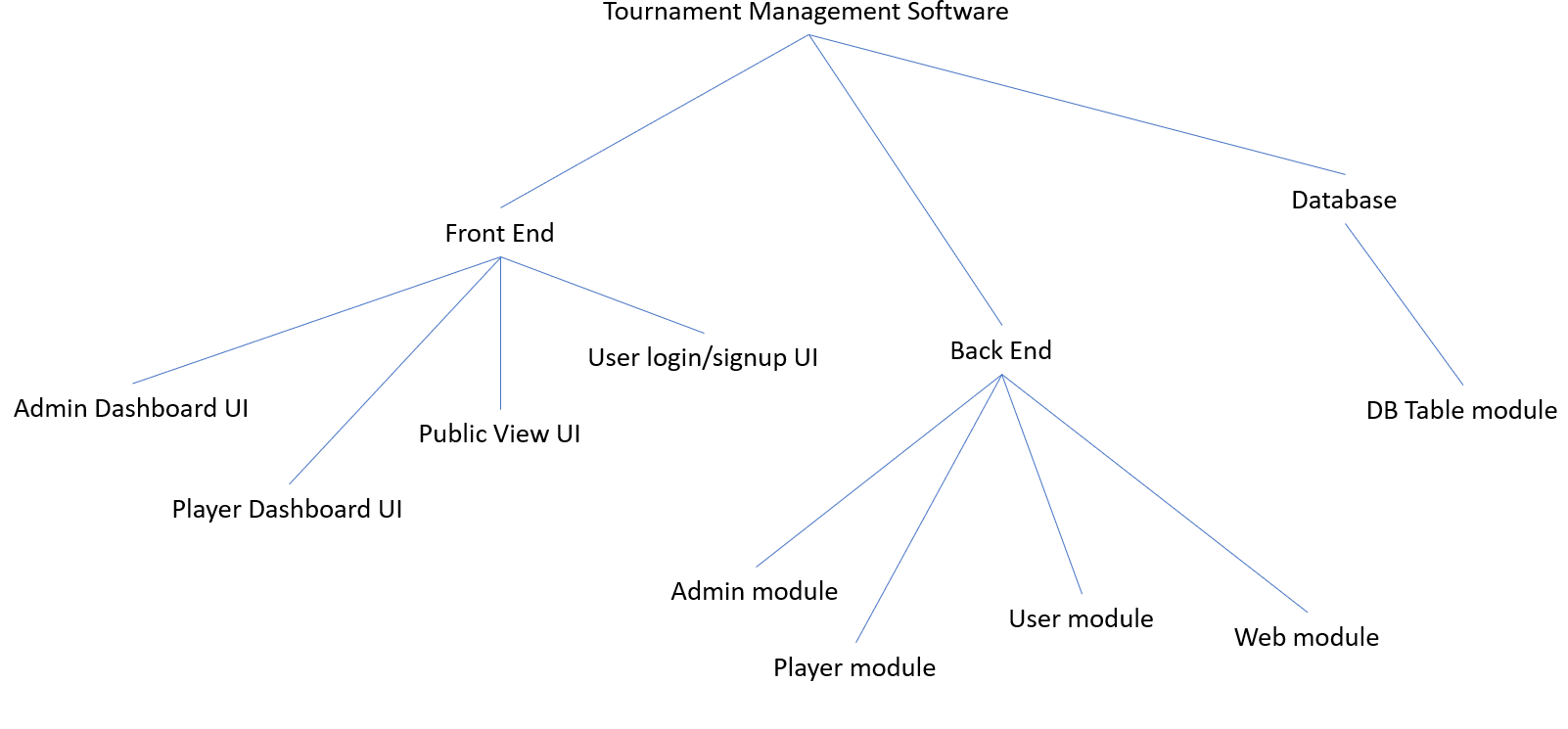
Application Logic (Owner: Enfa): this provides the support of communication between the Presentation Layer and the External Service. Includes requesting the information from the External Service (database) and sending to the Presentation Layer to display, storing the information passed in from the Presentation Layer and storing to the External Service (database).

External Services (Owner: Urvika): this provides the support for setting up the external database for our application, deploying the database on a distributed server, creating all the database tables.

Database (Owner: Urvika): this provides the storage space for our application needs including player profiles, tournament details, events, draws, and matches.

### Development View:

#### Primary Diagram:



#### Element Catalog:

User login/signup UI (Owner: Junfeng): this includes the implementation of the user login and sign up pages. Users may sign up for a new account via it. Authentication will be performed when users try to gain access to the application.

Admin Dashboard UI (Owner: Rupal): this includes the implementation of tournament management page, player roster page, events management page, and match management page. The admin user may use it to interact with all the admin capabilities for the application.

Player Dashboard UI (Owner: Junfeng): this includes the implementation of the player profile management and tournament registration page. The player user may use it to interact with all the player user capabilities for the application.

Public View UI (Owner: Junfeng): this includes the implementation of the public web pages including home page, login/signup page, player roster display page, event display page, draw display page, and match display page. The general public user may use it to browse all the tournament related public information.

Admin module (Owner: Enfa): this module processes the data received from the admin front end UI and also sends the requested data to the admin front end UI to display. This module interacts with the database. This also routes all the admin dashboard web pages.

Player module (Owner: Enfa): this module processes the data received from the player front end UI and also sends the requested data to the player front end UI to display. This module interacts with the database. This also routes all the player dashboard web pages.

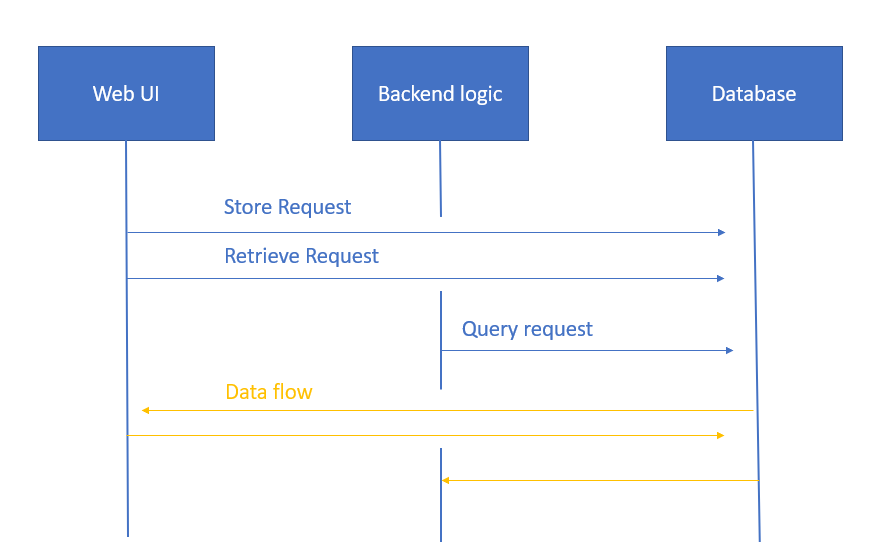
User module (Owner: Enfa): this module processes the user log in data received from the login front end UI and authenticates the user. This module processes the user sign up data received from the account sign up front end UI and validates it. This module interacts with the database. This also sends the requested data to the front end UI to display. This also routes all the login/sign up web pages.

Web module (Owner: Enfa): this module sends all the requested data for the public view page UI to display. This module interacts with the database. This also routes all the public view pages.

DB Table module (Owner: Urvika): this module creates all the required database tables for this application including user login table, permission table, player table, event table, tournament table, draw table, match table, etc.

### Dynamic Views

#### Primary Diagram:



#### Element Catalog:

Store request (owner: Junfeng): player profile information and event registration information will be stored to the database from the player dashboard UI. Tournament information and event/match information will be stored to the database from the admin dashboard UI. Player account information will be stored to the database from the user sign up UI.

Retrieve request (owner: Enfa): all the data displaying on different UI’s we have will need to retrieve the data from the database

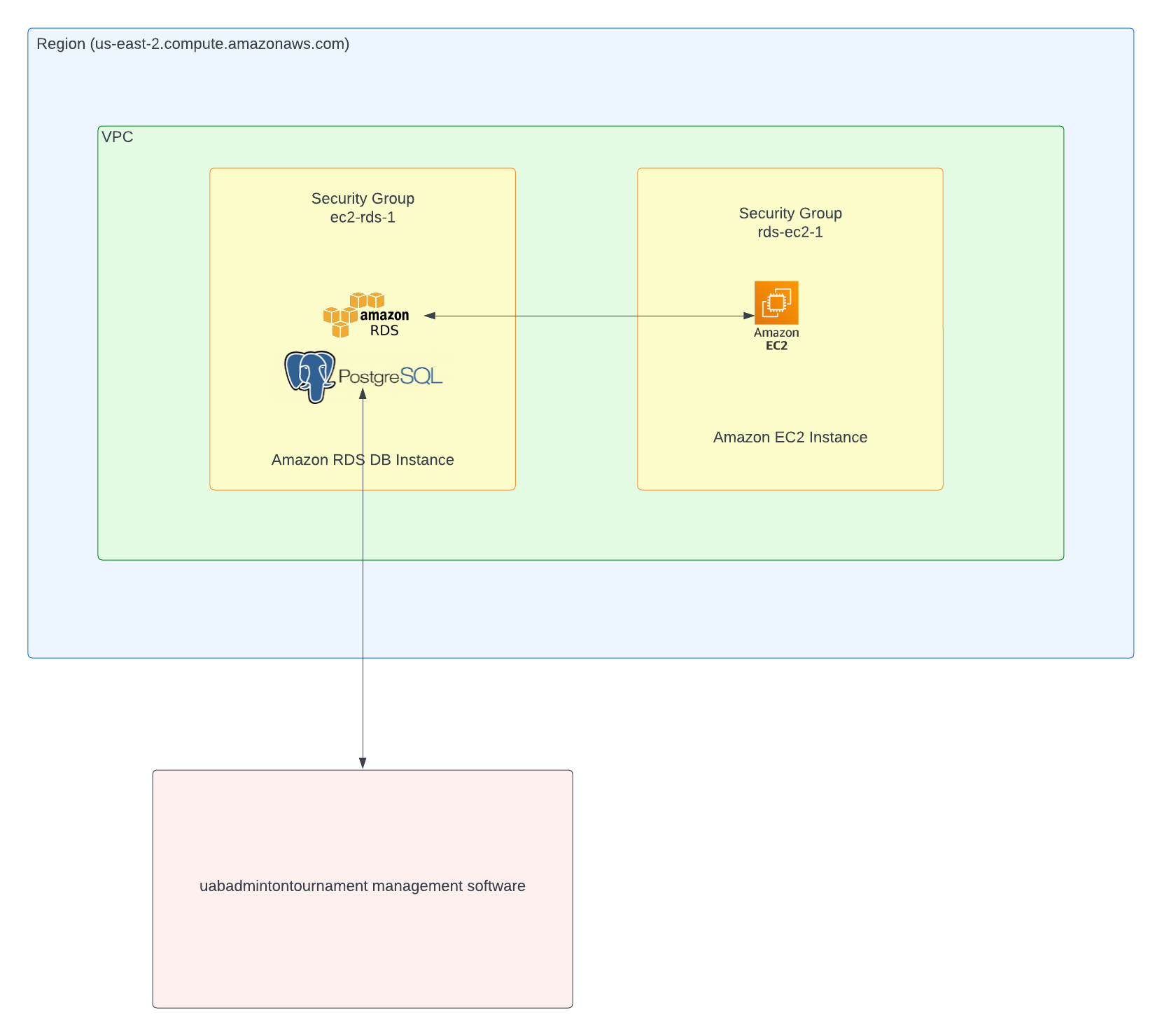
Query request (owner: Enfa): the user login function will need to query the user table from the database and authenticate the user login credentials.

Dataflow (Owner: Urvika): the data flow will be corresponded to the request types listed above.

### Deployment Views

For the current scope of our project, the deployment view consists of deployed servers running on Amazon RDS and Amazon EC2. Our software runs locally but it communicates to the servers on Amazon cloud. The below diagram despite the geographical boundaries of the servers. Amazon RDS automatically takes care of replication internally for scalability, performance, and availability.

#### Primary Diagram:

****

# **5. Current Status**

## **Goals for this Iteration:**

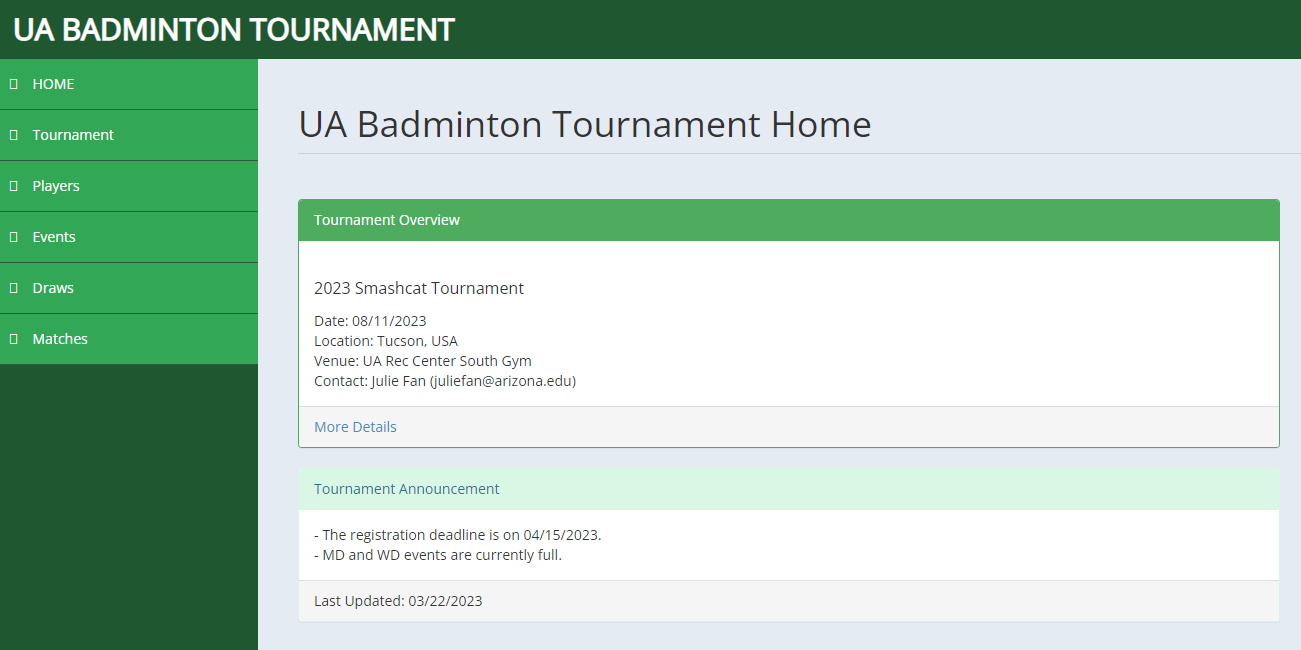
Refer to the Introduction section. No additional goals beyond what are described in the Introduction section.

## **Screenshots of What’s Working:**

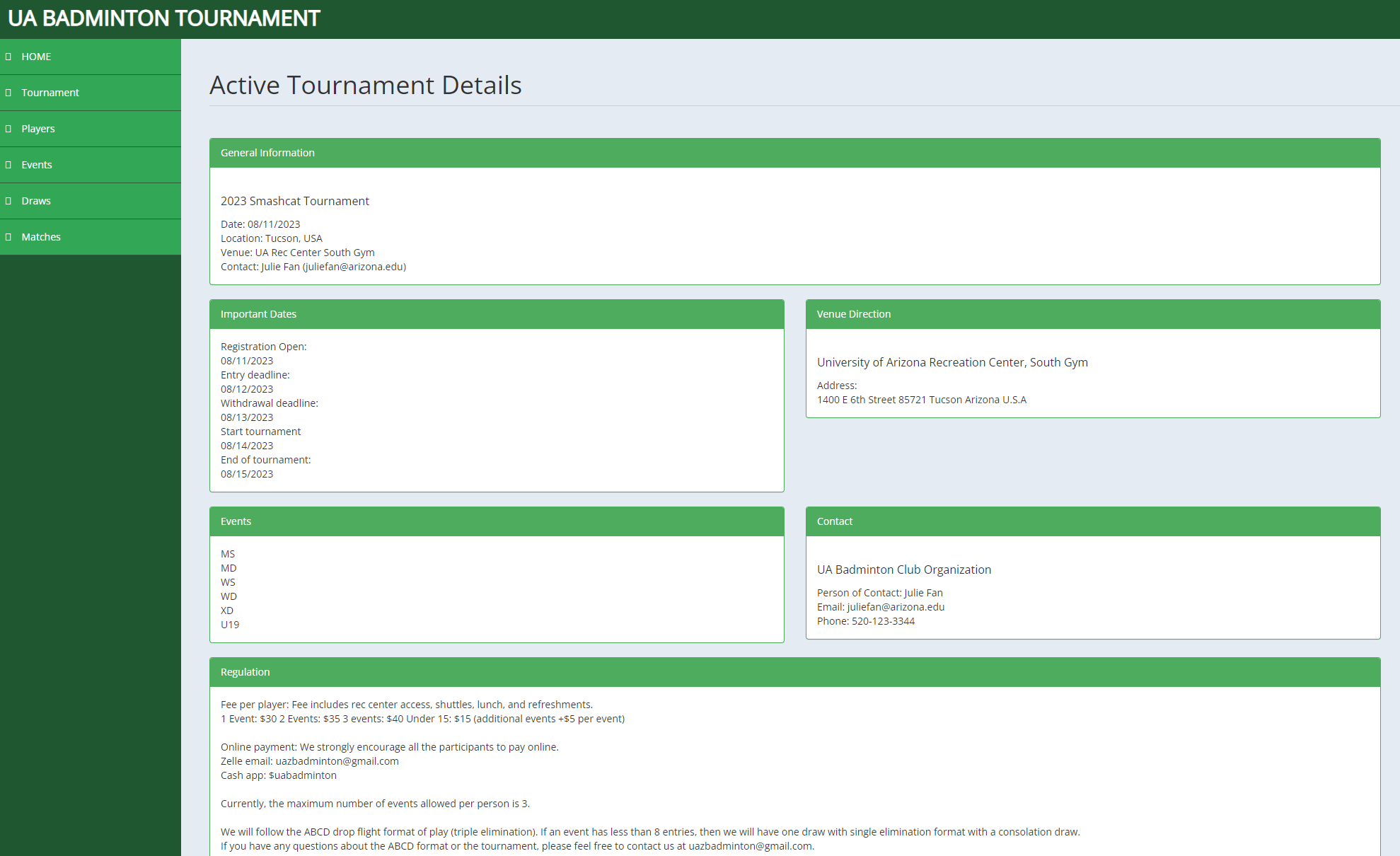
### Public View web pages:

Public users can use the following web pages to browse any tournament related public information.

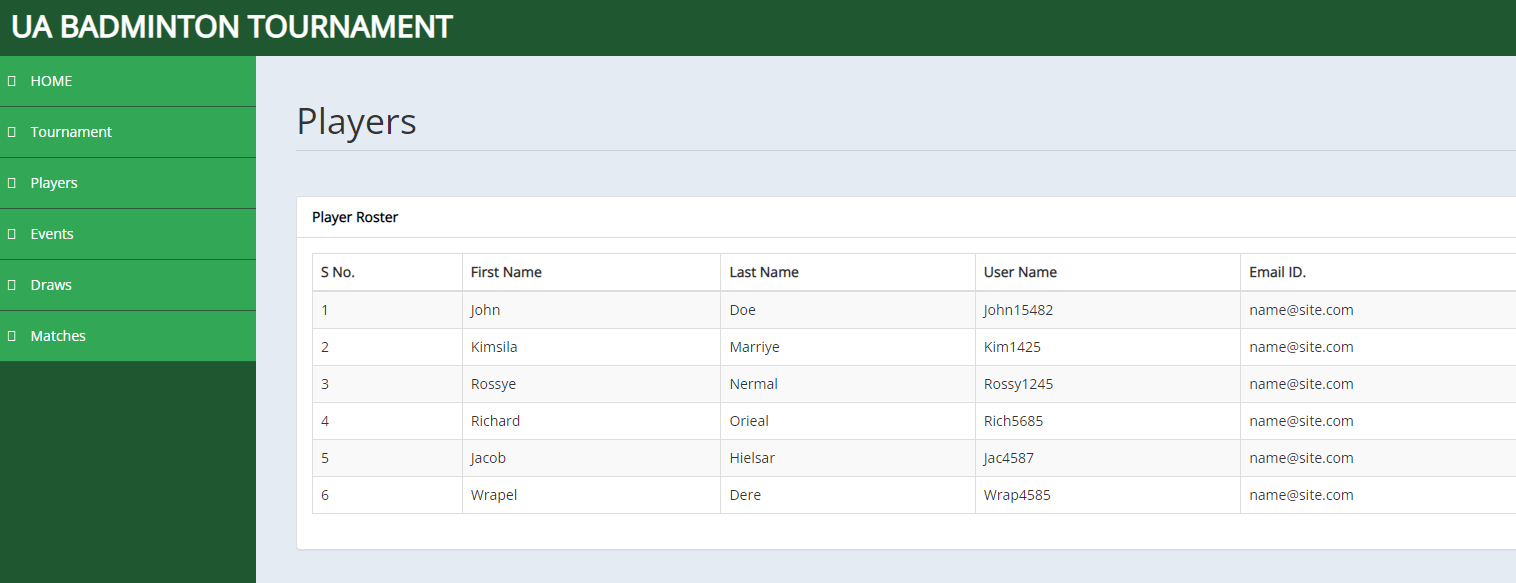
The Home page is able to display Tournament Overview and Announcements information:



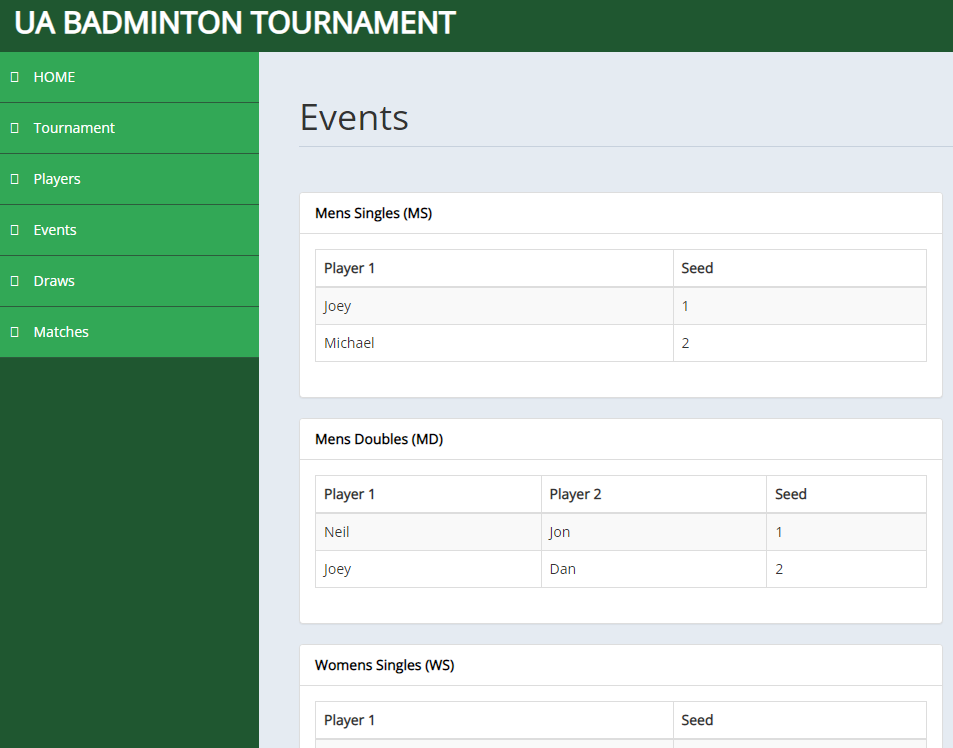
The Tournament page is able to display the detailed tournament information:



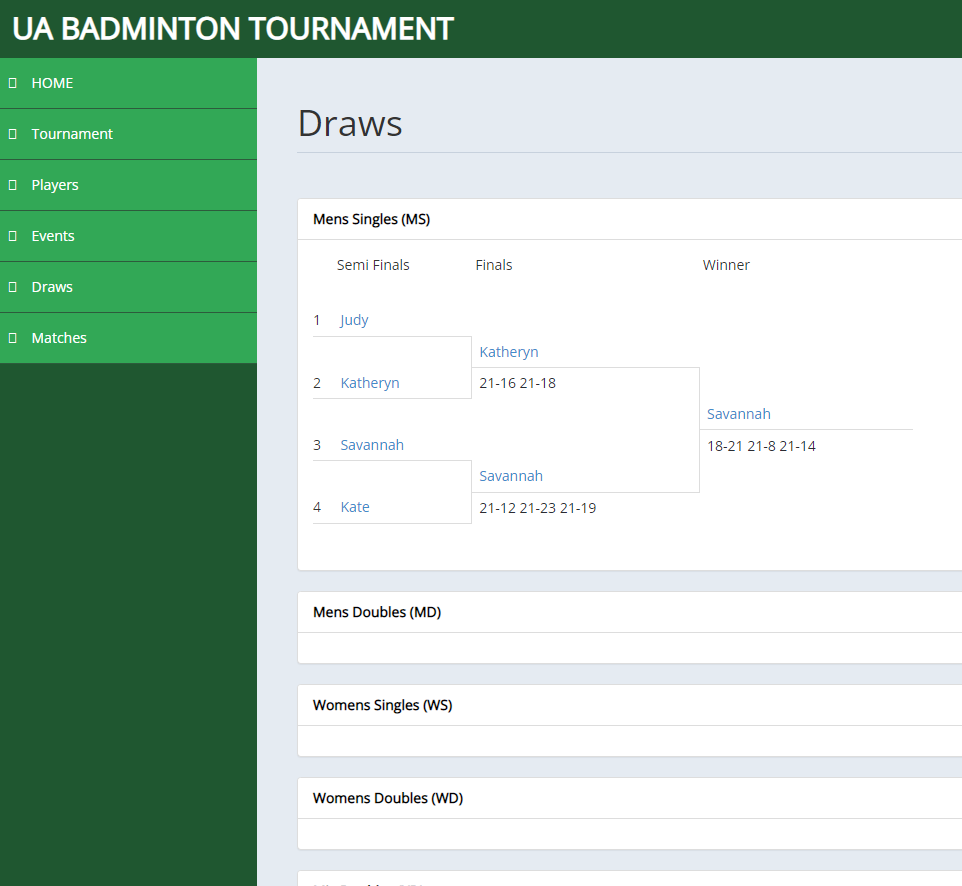
The Players page is able to display the player rosters:



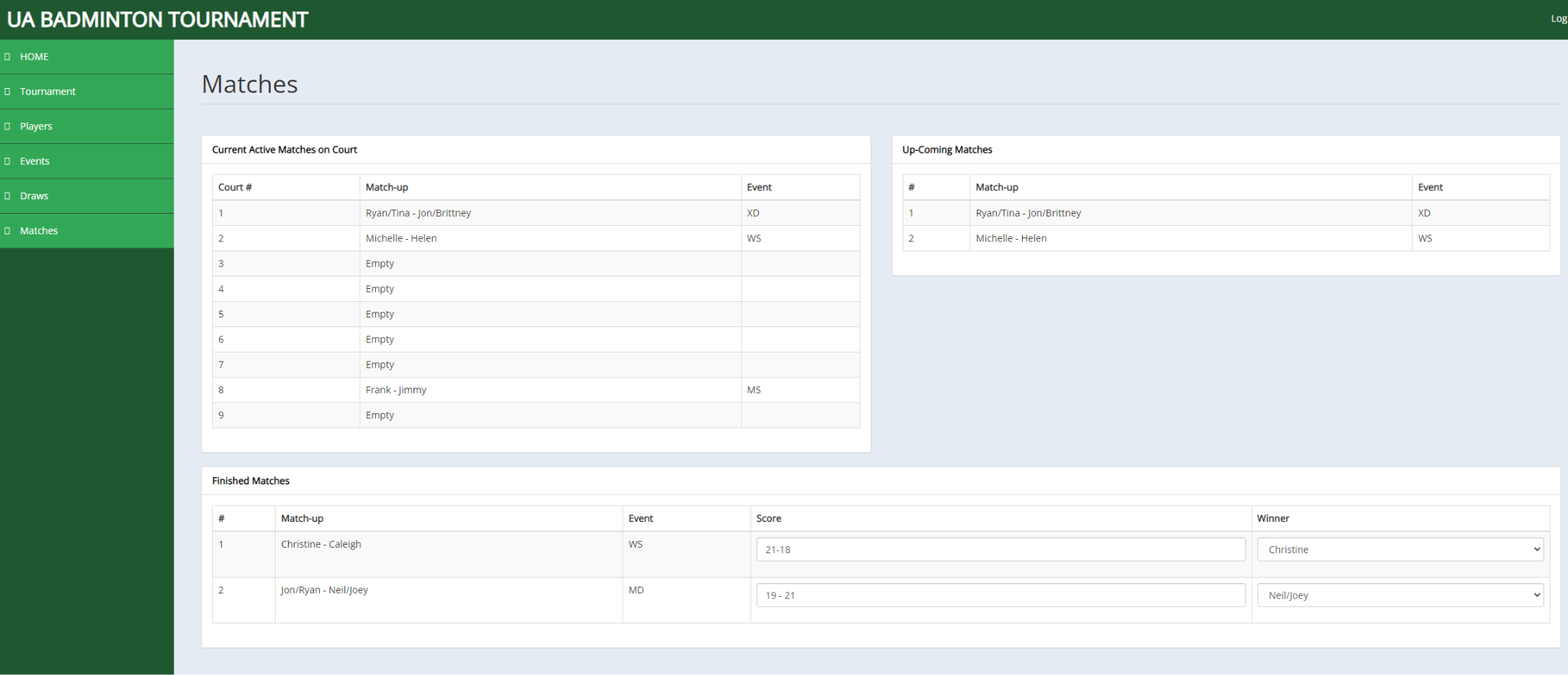
The Events page is able to display entries and seed information for each event in a tournament:



The Draw page is able to display event bracket for each event in a tournament:



The Match page is able to display court status, up-coming matches, and completed matches live on the tournament day:



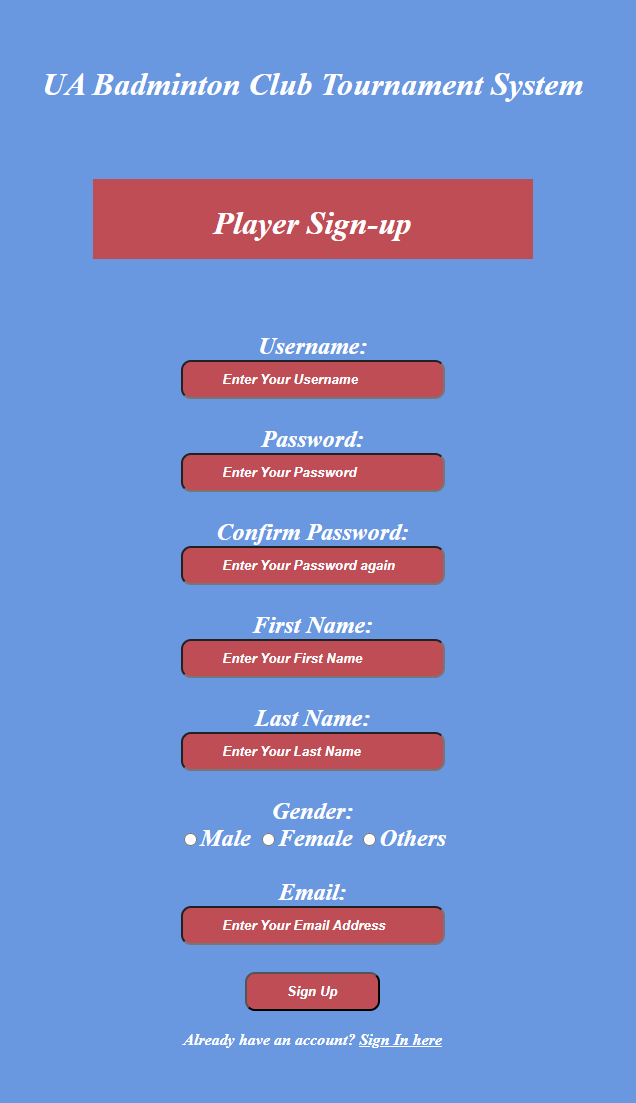
### Login and Sign up functions:

The admin user can log into their admin dashboard via the User Login page below. A player user can sign up an account and log into their player dashboard via the interfaces shown below.

Login page:



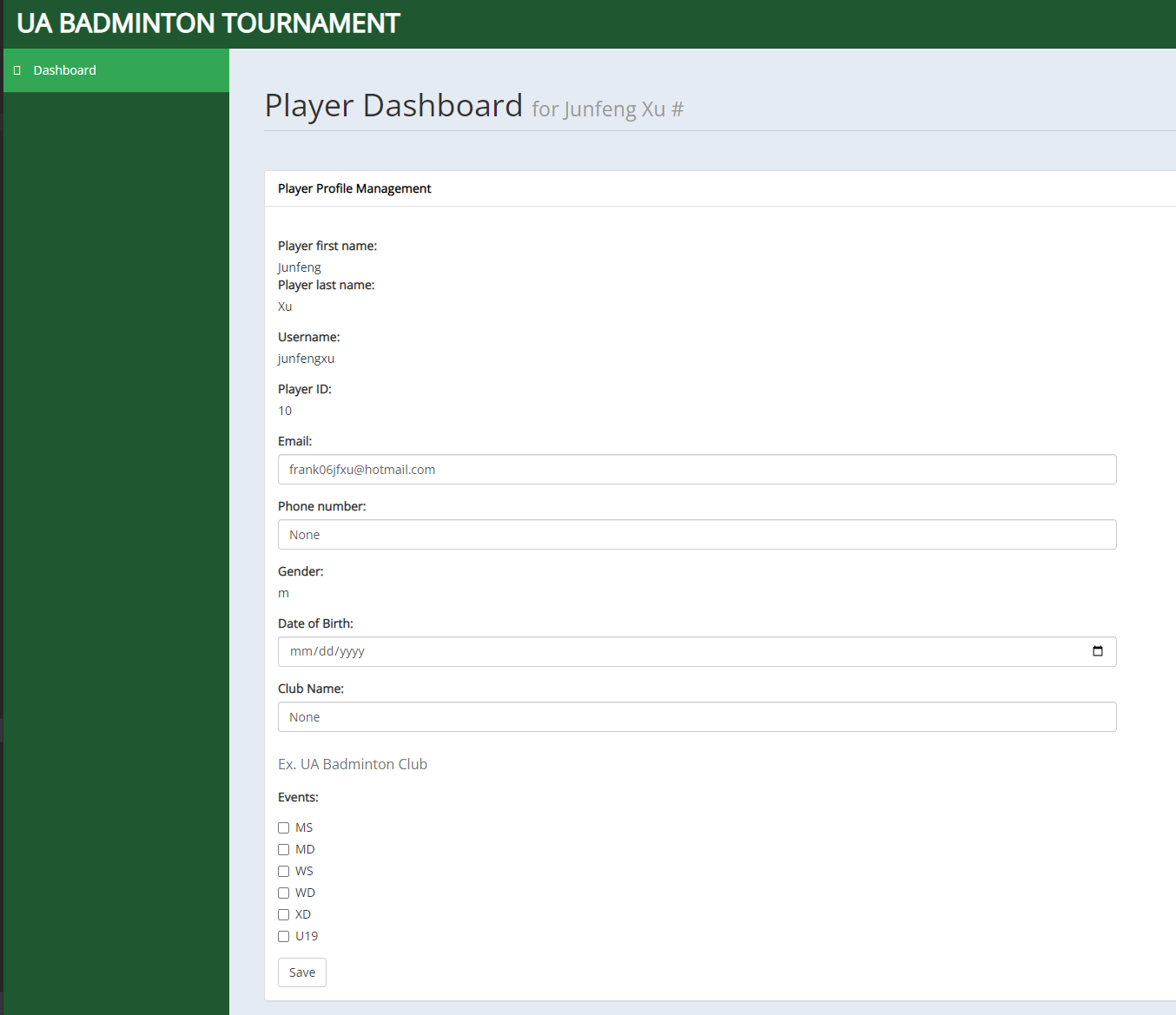
Player Sign Up Page:



### Player Dashboard:

A player user can use the player dashboard to edit their profile information.

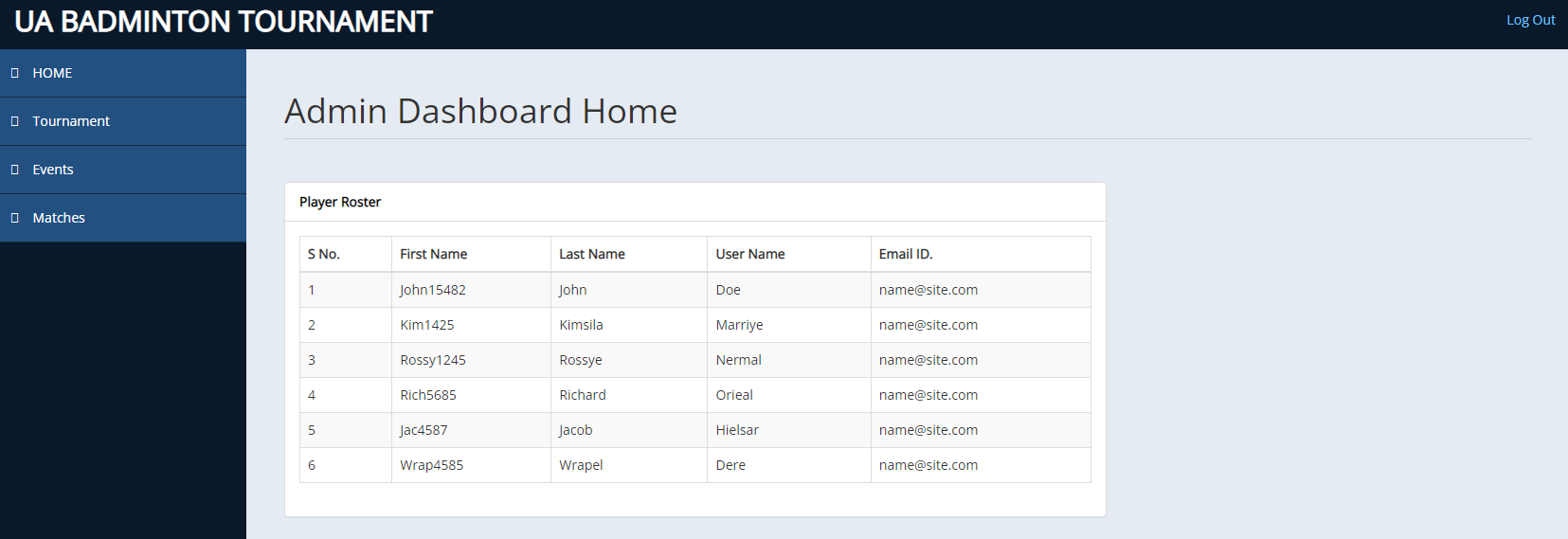
Player dashboard web page:



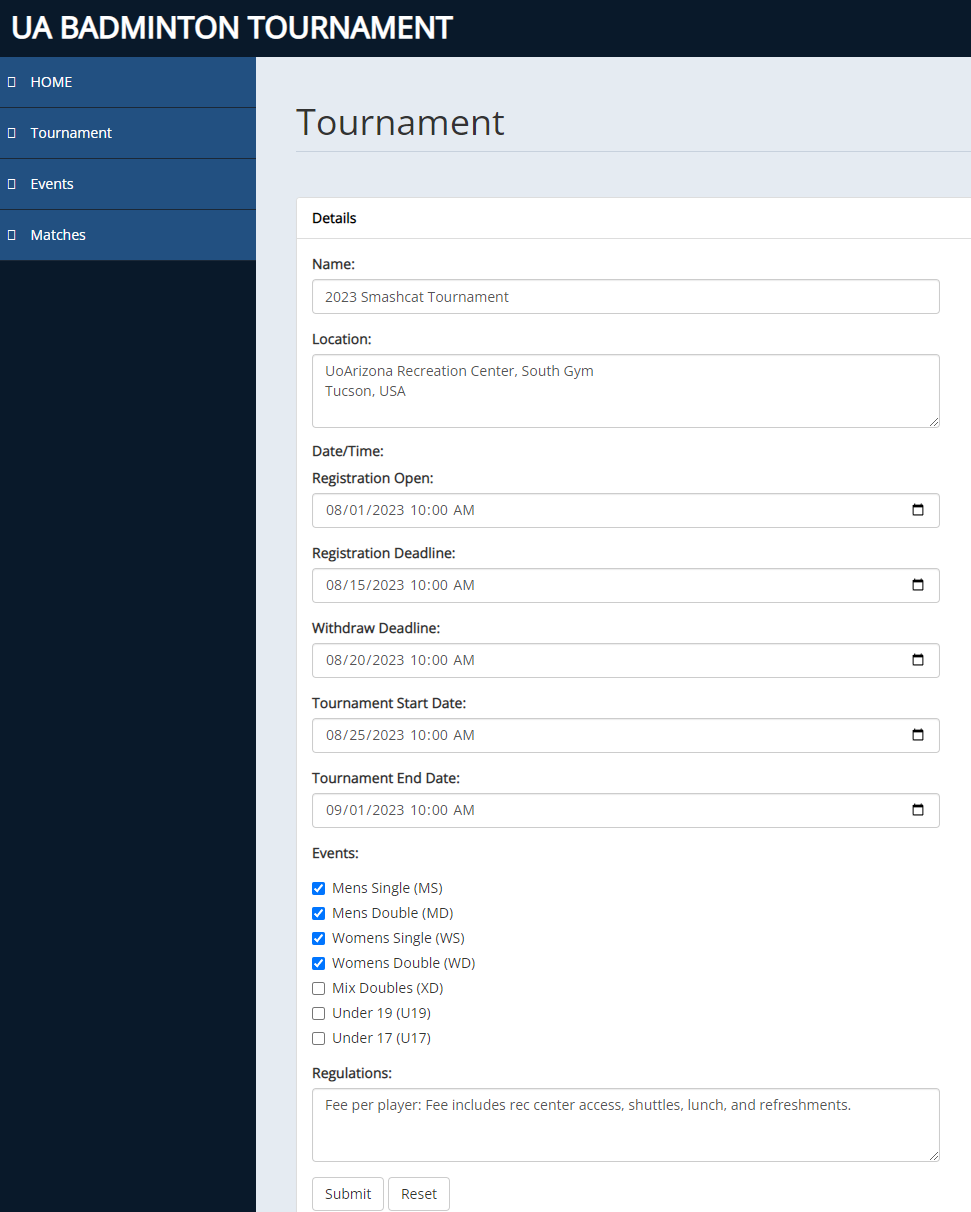
### Admin Dashboard:

The admin dashboard only has the front end interface completed. The backend support is still to be completed.

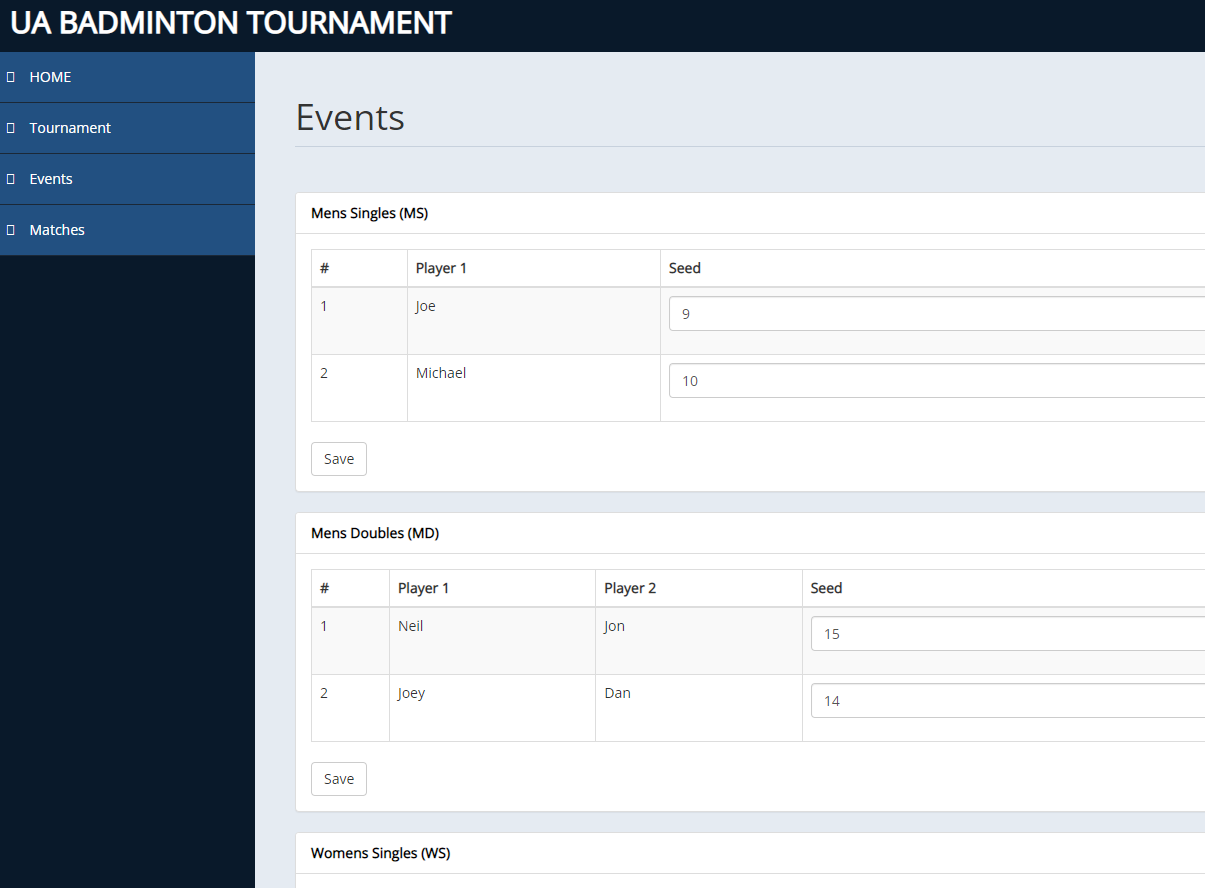
Admin dashboard Home page:



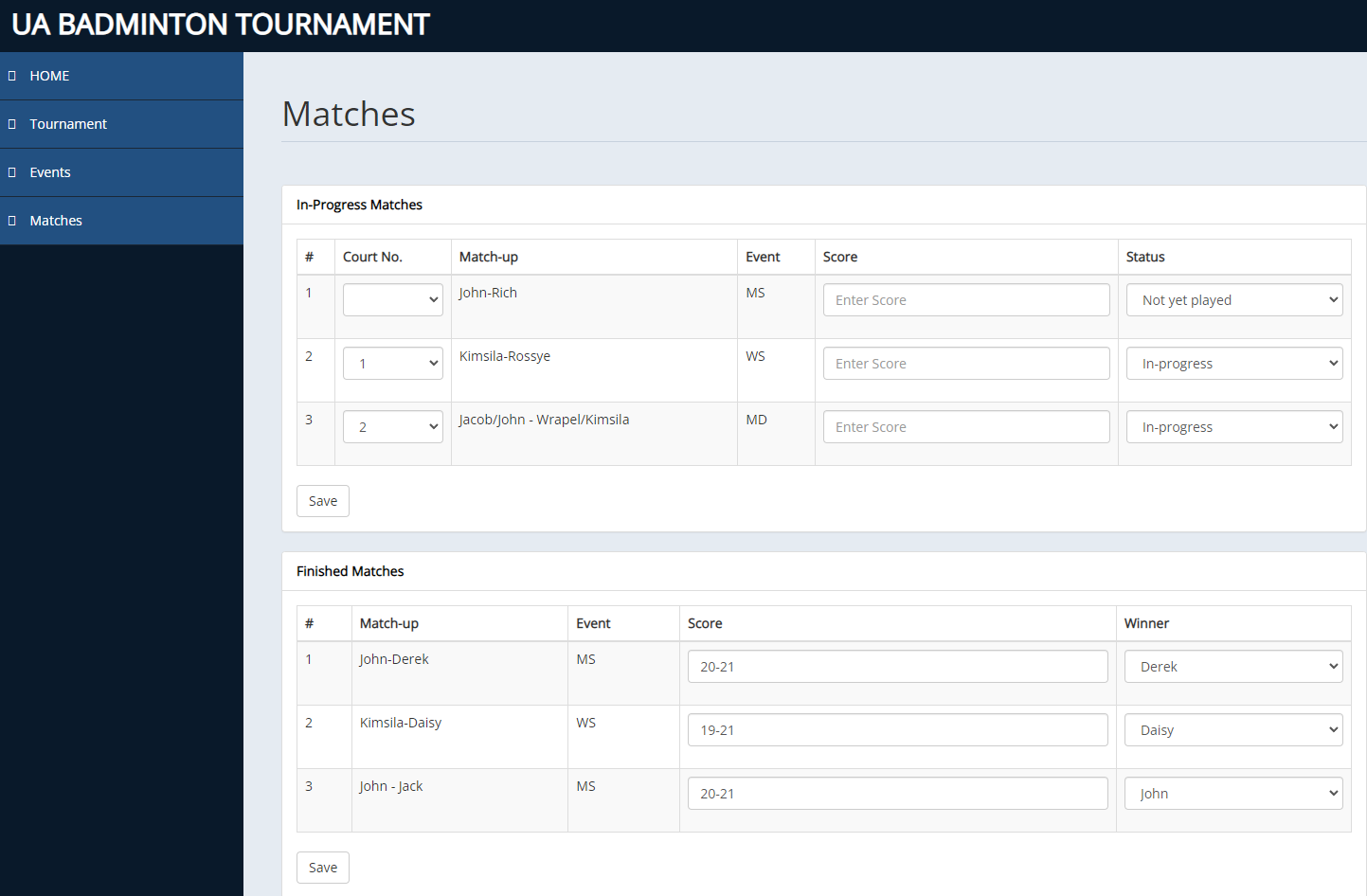
Admin dashboard Tournament Setup page:



Admin dashboard Event management page:



Admin dashboard Match management page:



### Database Setup:

Our production database is hosted on Amazon RDS and uses PostgreSQL; it is linked to an EC2 instance. Our database is open to the public, and the production database is used by all team members for development. This avoided the installation of database software on each team member's local PC. We have one system with a local setup of PostgreSQL that helps to debug issues locally without touching the golden production database. The following factors influenced our decision to use Amazon RDS:

1. It allows us to follow the Principle of Separation of Duty.

Amazon RDS enables database administration activities to be separated from

other development activities. One of us acts as the database administrator.

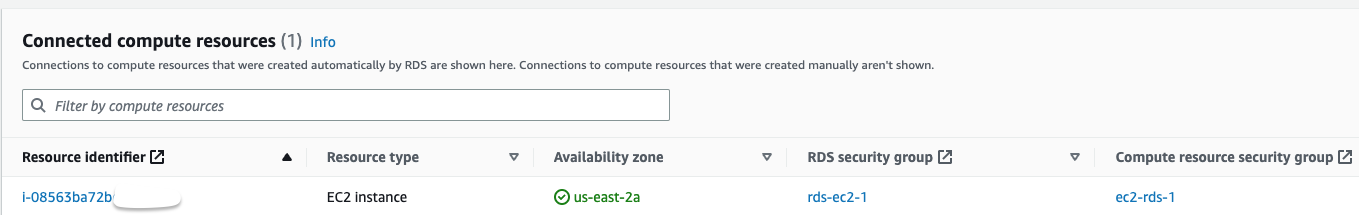
1. We get 20GB of storage for free on our student account.
2. Scaling and Performance: Amazon RDS offers simple scaling choices to meet a tournament management software's shifting workload demands. It enables automatic scaling of compute and storage resources, ensuring optimal performance during peak usage periods, i.e. during and around a live tournament.



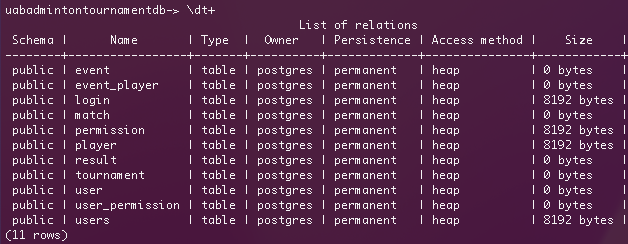
**Amazon RDS connection with EC2 instance:**

We connected the Amazon RDS to EC2 due to the following reasons:

1. **Security**: By connecting our Amazon DB instance to an Amazon EC2 instance, we ensured that our tournament management software app's database is protected from unauthorized access. This is important for any app that deals with sensitive information, such as user data or financial information. By setting up security groups and configuring our database to only accept connections from our EC2 instance, we minimized the risk of data breaches or other security threats.
2. **Scalability**: If our tournament management software app experiences a surge in traffic during peak periods (e.g. during a major tournament), we need to be able to scale our infrastructure to handle the increased load. By connecting our Amazon DB instance to an Amazon EC2 instance, we ensured that we can easily add more EC2 instances to handle the increased traffic. This helps ensure that our app remains responsive and available, even during periods of high demand.

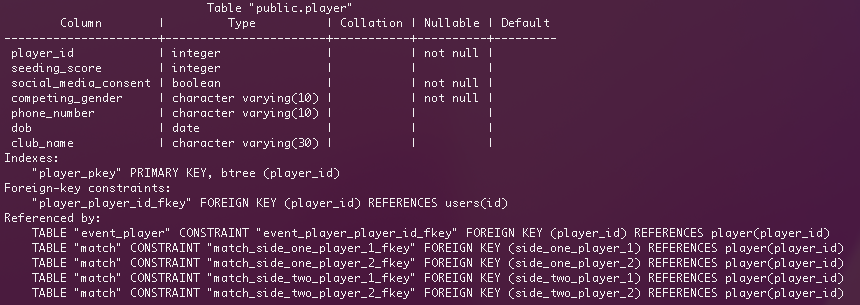


Our database contains the following tables:



The schema of few crucial tables:

1. The Player table:



1. The Tournament table:



1. The Match table:



## **Challenges** w**e are facing:**

#### Challenges with the database setup:

What didn't go well was the initial attempt to set up the database using Oracle on Amazon EC2. We broke down and examined our issue with this timeline:

Challenge - Difficulty starting Oracle on Amazon EC2 instance due to configuration issues after installation.

Attempted Solution: Tried a few configuration changes, none of them helped, proceeded to a fresh reinstall of Oracle Database software on Amazon EC2, but issue persisted.

Impact: Delayed timeline by 3 days, resulting in increased development time.

Solution: Leveraged alternative database system - Postgres. Installed and familiarized with Postgres on Mac locally,

Next Steps: Installed Postgres on Amazon RDS and successfully connected it with EC2, resolving the Oracle configuration challenges.

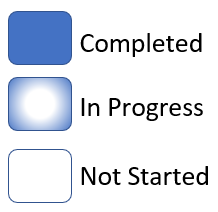
Results: Connected the product database setup using SQLAlchemy to our Flask Application, and the project timeline was back on track.

#### Challenges with the tournament bracket implementation:

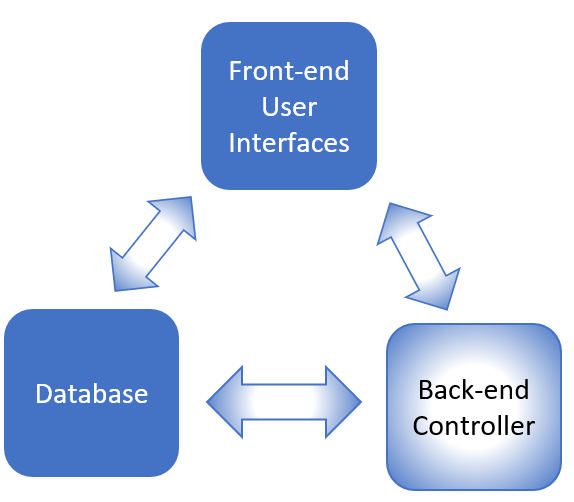
Initially the developer that implemented the public view front end web pages had no knowledge on how to build the tournament bracket on a web page using HTML. He did some research and also seeked help from a subject matter expert on web application front end development. He figured out the bracket is just a table under the cover and was able to implement the tournament bracket front end in this iteration.

## **Block diagrams for the current state of the system:**

Note: our progress for this iteration can be clearly seen by comparing the updated block diagrams here to the ones in Status 1 Report.

(Block color code:)

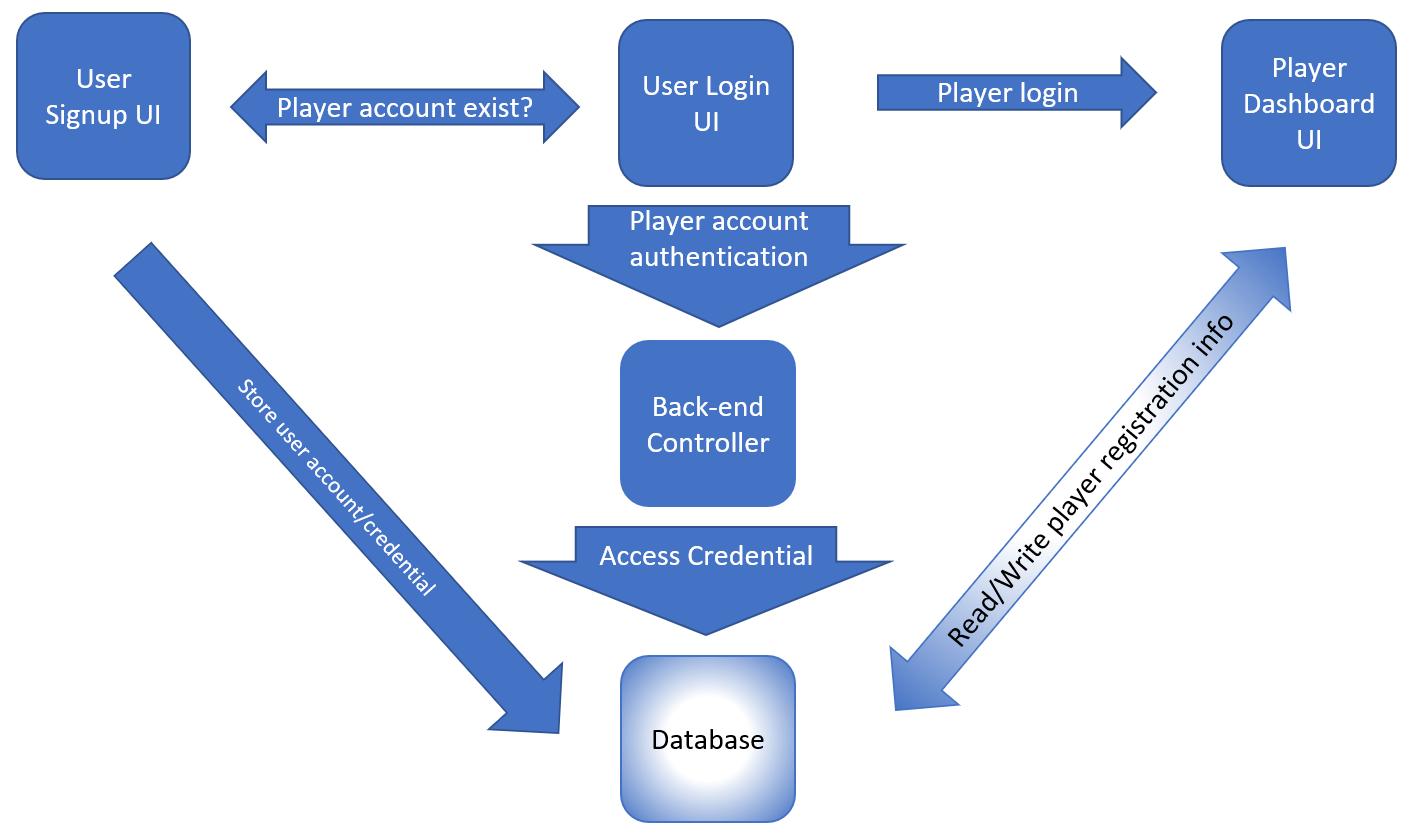
### System overview block diagram:



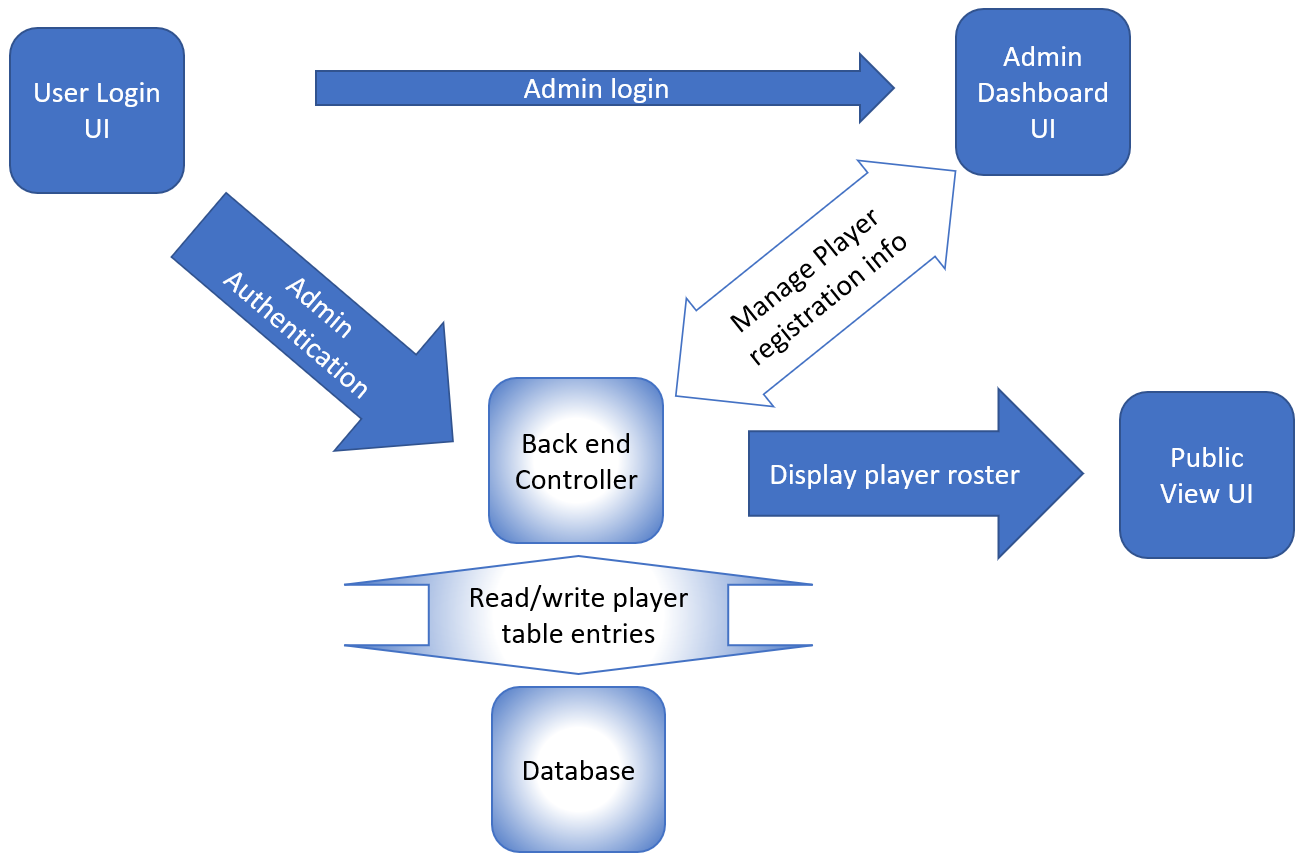
### Modular block diagram:

Note: System modules are Registration, User Management, Seed & Draw, and Match Management.

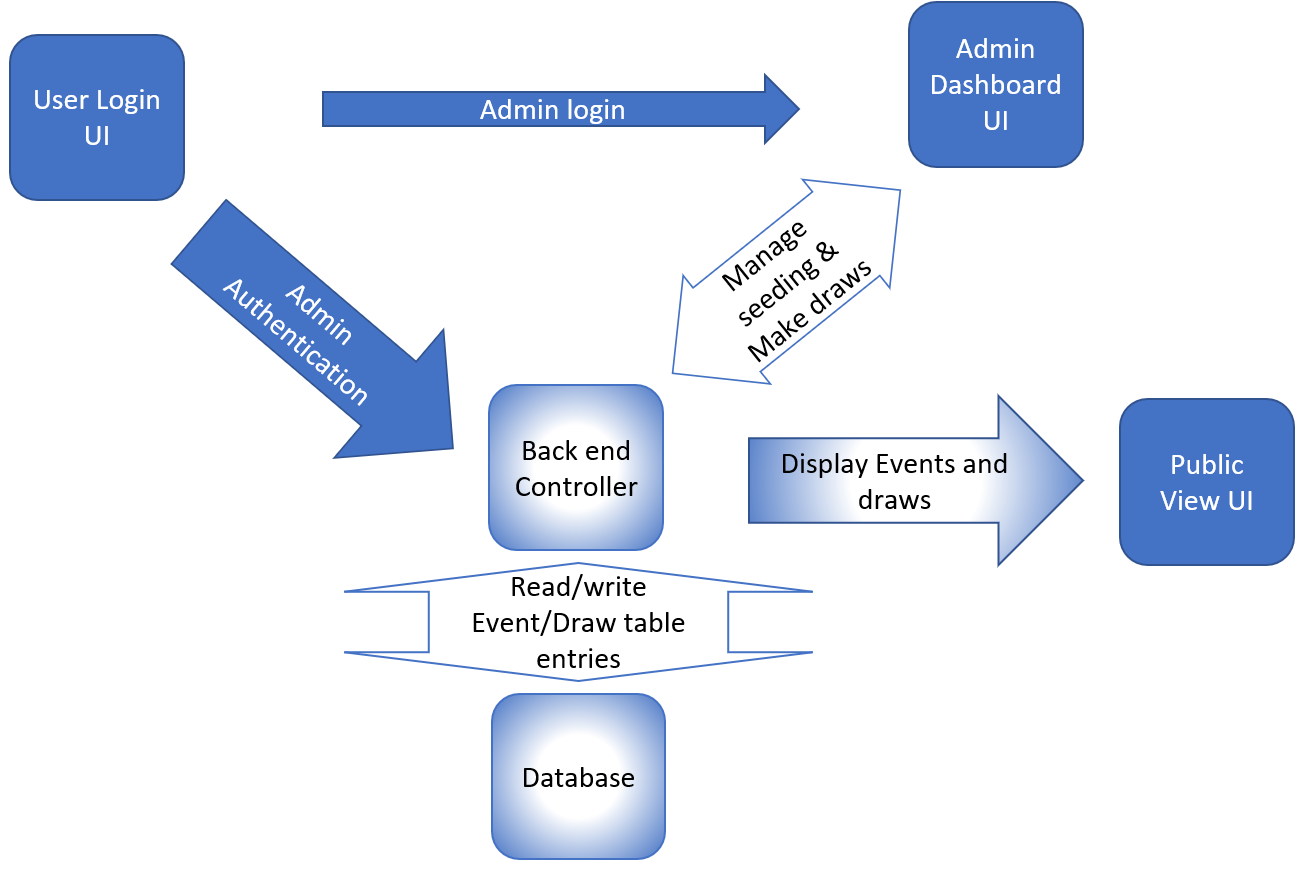
#### Registration module:



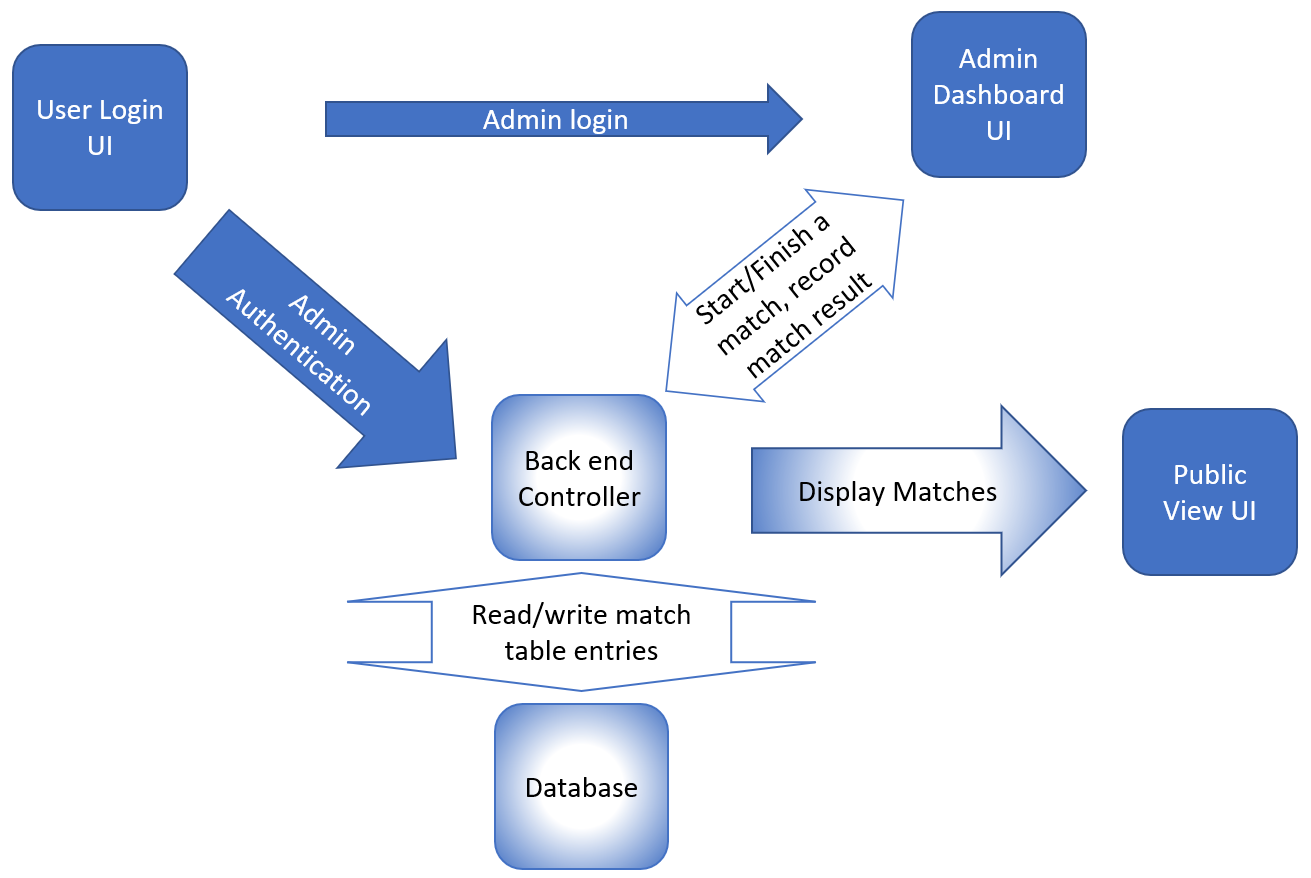
#### User Management Module:



#### Seed & Draw Module:



#### Match Management Module:



## **Test Performed:**

1. Manually **functional testing** with dummy values on a local server.
2. **Unit testing** for front end UI receiving data from or sending data to the back end controller.
3. **Unit testing** using the **Pytest framework.** An example: Added a test to add a permission object to the permission table in the database and verify that it was successfully committed to the database using the app context and database session. This test helps ensure that our app's database functionality is working as expected and can handle adding new data to the database.
4. **Function verification testing** for each functionality we implemented this sprint.

## **Total number of lines of code we collectively written:**

~ 4050 loc

# 

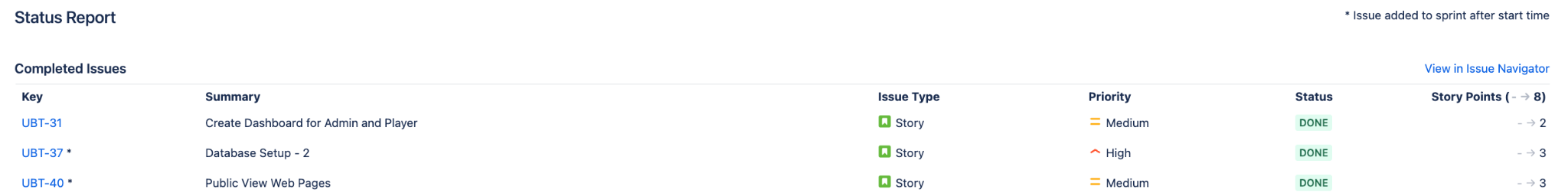
# **6. Project Management**

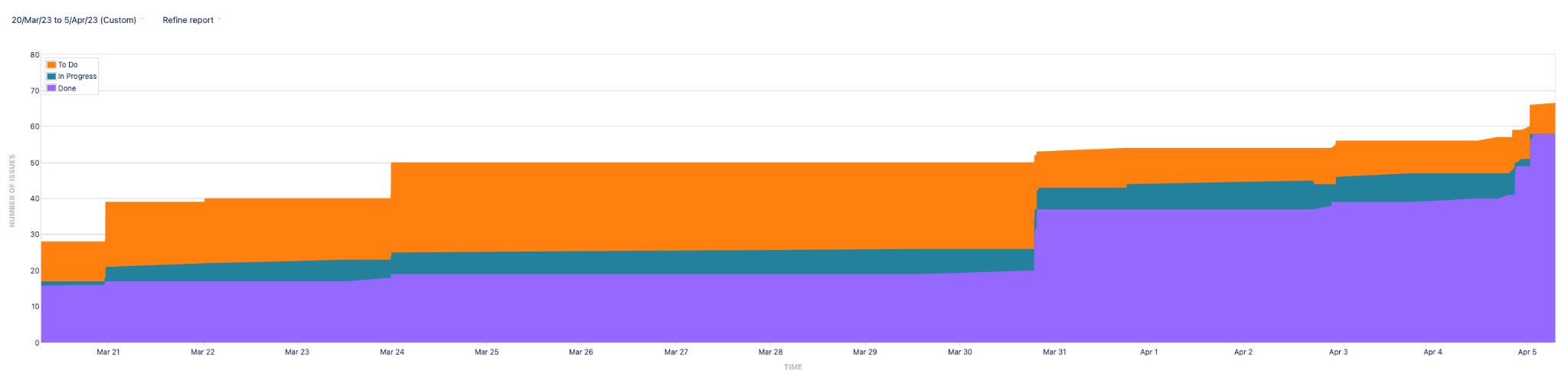
## **Change Log**

| Date | Description | Motivation | Implementation |
| --- | --- | --- | --- |
| 03/02/2023 | UI Design Change | To make the admin dashboard more user-friendly and easier to update. | Updated UI design to have an admin dashboard in a single screen instead of multiple screens. This will make it easy for the Admin to make updates. |
| 03/07/2023 | Database redesign | To fix errors with primitive data types in initial design. | Reworked the structure of the database as previously we found some design error, and created a new ER diagram. |
| 03/12/2023 | Customer desired over all experience change | To satisfy the customer’s updated requirements that we have learned from the customer’s feedback. | Given our current design for the project, it is still able to address customer’s new surfaced problems. |

Jira Sprint:

1. Sprint Duration - 2 weeks 2 days (20th March - 5th April)
2. 3 Stories
3. 40 Subtasks

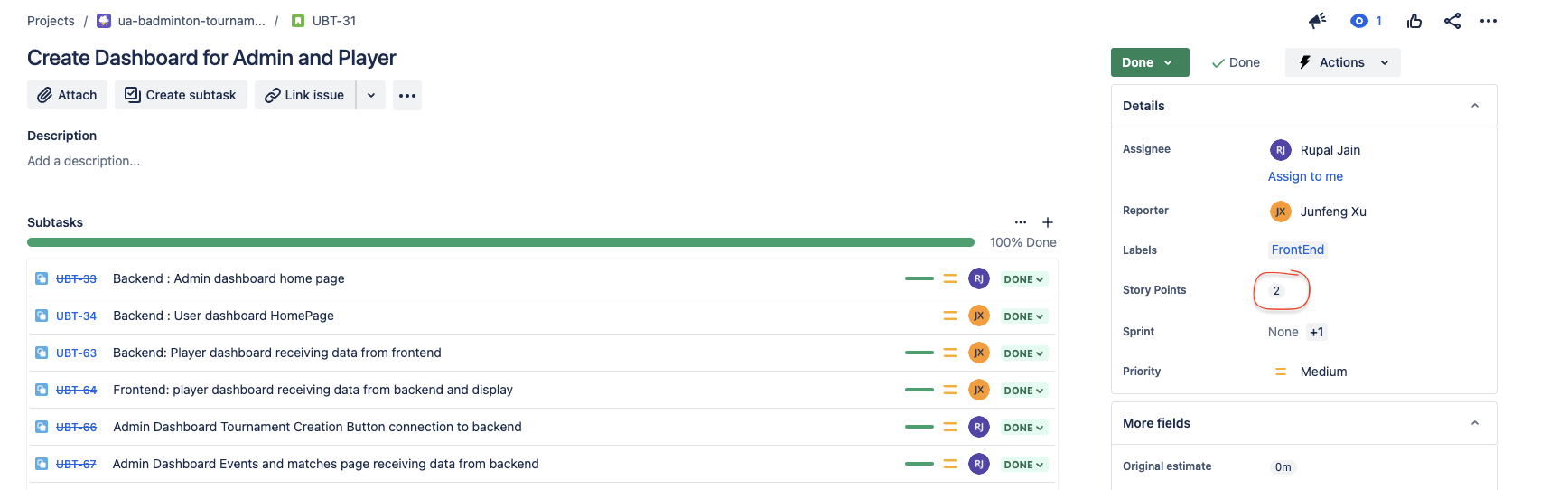


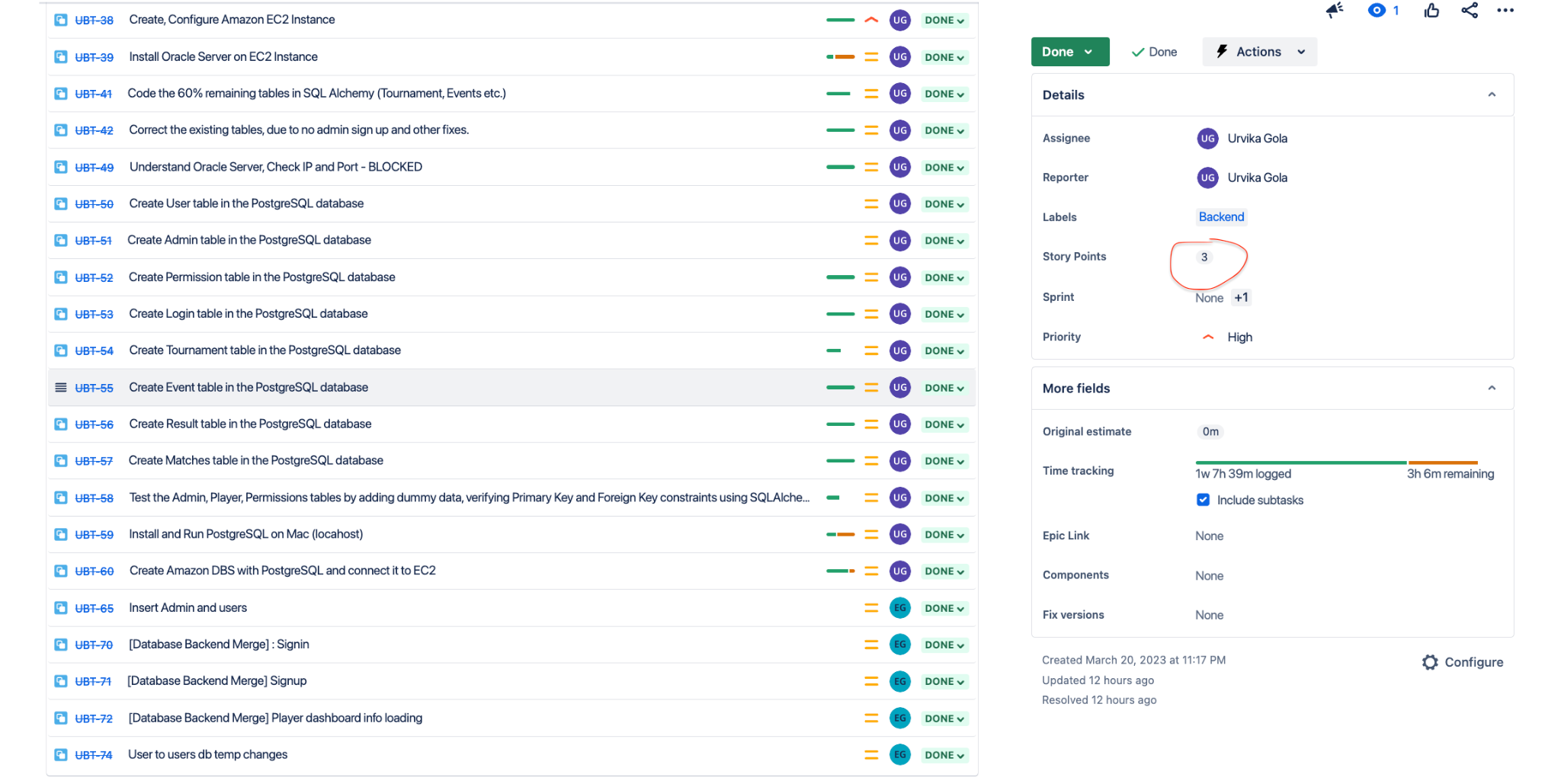
Cumulative Flow Diagram

**Incorporated 1, 2, 3 Story Points:**

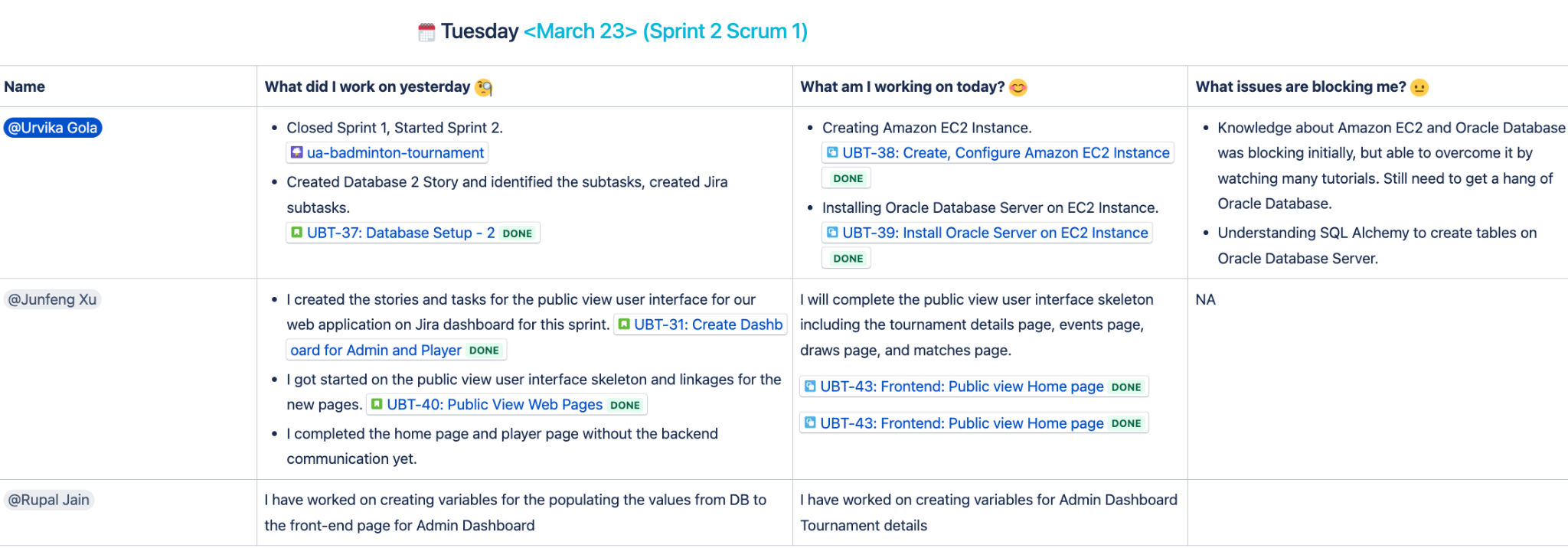
After learning about story points in class, we practiced assigning story points to user stories in our Jira dashboard. In our sprint, we assigned three stories with story points. Two of the stories were assigned 3 story points because we determined that they were more complex than the other user stories in the sprint and would require more effort to complete. We chose to assign 2 story points to the third story as it was less complex and would require less effort to complete. By assigning story points to our user stories, we were able to better estimate how much work can be done in a sprint and plan our resources accordingly.

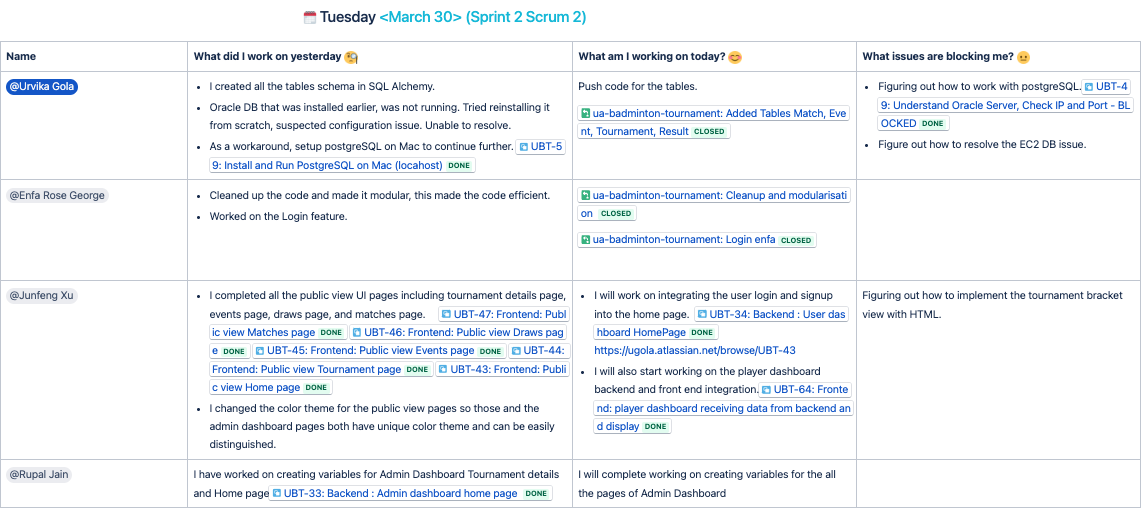
Snapshots of stories highlighting the story points:





Daily Dashboard







## 

## 

## Goals for next iteration:

1. Replace dummy values in the admin dashboard with SQL queries to load data from the database. Dummy values were added when the database was facing challenges.
2. Develop the logic needed to create a tournament, including creating a tournament entry in the database, adding events marked as included by the admin as separate events in the database, and displaying these events on the player dashboard.
3. Implement draw functionality, including allowing the admin to add a seeding score, creating the initial draw based on the list of participants and seeding score, and updating matches live with score, venue, and status (ongoing/finished) - This will ensure that tournaments can be started and managed seamlessly.
4. Implement data validation for all form entries in the frontend
5. Apply the theme to the sign in pages and home page.
6. Match management, implementing a feature that can display all first round matches in a clear and user-friendly format.

## Goals for rest of the semester:

Additionally, towards the end, we will brainstorm on optimizing the application's performance and scalability to ensure it can handle large volumes of data and traffic and hosting the application on cloud. Ultimately, our goal is to deliver a fully functional application that meets and exceeds customer requirements.

# 

# **7. Team**

## **Team roles for this iteration?**

During this iteration, our team had a well-defined set of roles and responsibilities.

Junfeng Xu served as the Product Owner and oversaw the product to stay in line with the requirements. He also served as a developer and contributed to the implementation of the public view front end interface web pages and changes to the player dashboard front end interface.

Urvika Gola acted as the scrum master, ensuring that the team stayed on track and met customer goals. In addition to configuring the sprint on Jira board, maintaining the daily dashboard, Urvika is a backend developer, she designed, created, and configured the entire database on cloud using Amazon RDS. She installed, configured PostgreSQL on Amazon RDS and connected them to Amazon EC2. She is acting as database administrator, managing the Amazon Cloud Portal for the previous tools being used.

Rupal Jain is one of the developers and contributed to the backend part of the Admin Dashboard UI. She was responsible for connecting the backend to the front-end page and populating the dummy information structured in the same way as the original tables in db for the Admin Dashboard UI. This includes completing the development of intermediate between python and html.

Enfa was the backend developer this iteration. Her main responsibility was in integrating the database with different functionalities in the project. She also set up the system to ensure that unauthorized operations are not possible.

## **Team members contribute?**

1. Junfeng Xu contributed to the communication with the customer. He met with the customer and showcased our progress and collected feedback. He also contributed to building the complete front end interface for the public view web pages and the player dashboard front end communication piece to the back end. He also used the Jira dashboard to keep track of his work and progress. He performed code reviews for the peers.
2. As the scrum master, Urvika was managing the Jira sprint. Urvika is a backend developer, she programmed the ER diagram she had designed in previous sprint to tables using SQLAlchemy. She created, configured (allowed access with security groups) the entire database on cloud using amazon RDS. She installed, configured postgreSQL on Amazon RDS. She also set up an Amazon EC2 instance that communicates with the Amazon RDS. Finally, she connected the PostgreSQL running on Amazon RDS with the application using SQLAlchemy, Python. She also reviewed code reviews for the peers.
3. Rupal played a key role in connecting the front end of the Admin Dashboard to the backend. To connect the front end of the Admin Dashboard to the backend, Rupal first created variables with dummy data, which were then parsed and displayed on the front end. This involved designing and implementing the necessary functions and features that would enable the system to mark new information and update it in real-time. She ensured that the front-end and back-end of the system were seamlessly integrated.
4. Enfa’s focus was on integrating the db setup with the existing backend codebase. Earlier, dummy data coded into the python program was being used to populate the website. Her focussing was on replacing these variables with actual values reading directly from db. The main areas where she completed this are sign in, signup, loading and updating player dashboard details. Outside of this, she worked on modularising the code better to support the increasing codebase. She also implemented the functionalities needed to ensure that an unauthorized user cannot access any dashboard pages.

## **Rough estimates of each team member contributions to the project (in percentage)**

Our team's estimates suggest that every team member made noteworthy contributions to the success of the application. According to our rough calculations, each team member made an equal contribution of 25% to the overall progress of the project. This exemplifies the cooperative and collaborative spirit of our team, as well as each member's dedication to ensuring the project's success. By working together, supporting one another, and sharing ideas, we accomplished our goals and delivered a top-quality outcome during the sprint.

# **8. Reflection**

## What went well?

1. **Backend Integration:** During this iteration, our team made significant progress in developing the user login/signup functions by completing the backend support and integrating it with the front-end UI. Most of the player dashboard functionalities, including the player dashboard UI web pages, backend support, and retrieving and storing player information from and to the player database table, were completed. Although the event registration portion of the player dashboard remains to be completed, we performed testing to ensure that a player user account can use it to display their current profile and modify it.
2. **Testing:** The team performed multiple types of testing to ensure the functionality of the user login and signup functions. Unit testing was carried out during the implementation of each function, while functional testing was performed in conjunction with other completed tasks. Additionally, user testing was conducted during the customer review meeting to gather feedback and ensure the user interface was intuitive. The front end support for all the public view UI web pages was completed, and corresponding unit testing was carried out to verify their functionality.
3. **Database Configuration:** Despite facing a roadblock due to Oracle configuration issues on an Amazon EC2 instance, our team was able to successfully resolve the issue and continue with the project. We found an alternative solution by installing Postgres on Amazon RDS and connecting it with EC2. This allowed us to connect the production database setup using SQLAlchemy to our Flask application.
4. **Project Management:** The usage of numerous tools, such as the Jira, Atlassian Confluence, Github contributed to the effectiveness of our project management process. These tools assisted us in staying organized and kept everyone on the team up to date on our progress. Each team member gave their updates and any issues they were facing during weekly sprint meetings. The Scrum dashboard was used to track our progress, identify areas that need our attention, and prioritize our efforts. We also connected Jira and Github to Confluence, which allows us to see all project management tools under one umbrella. We use a Whatsapp group to communicate on a regular basis.

## What didn't go well?

One thing that didn’t go particularly well is our time management as a team. If you see our Cumulative Flow Diagram in the Project management section, it demonstrates that we were going slow at the beginning of this iteration in terms of the number of issues we closed on. Then towards the end of iterations, the number of issues we closed on got ramped up. Ideally, this diagram should have a linear curve instead of a step-type of curve like ours. In the next iteration, we need to pay more attention to time management. We will try to more evenly distribute the issues throughout the whole iteration.

## Goals that were not met, what were the issues?

One of the goals was to complete event registration in the player dashboard which is yet to be completed. The issue we faced was the backend support for the player dashboard is more complicated than we anticipated. We underestimated the scope of work for it. So we only finished the player profile portion of the player dashboard, and moved the event registration place to the next iteration.

## Plan to overcome the issues?

It's worth mentioning that since this project involves the use of new technologies that were not familiar to everyone right from the beginning, we have made it a priority to maintain open communication and support each other throughout the development and learning process. We believe that our current success is largely due to our collaborative approach, and we intend to stick with it into the upcoming iteration.

## Plan to do differently in the next iteration?

While we were able to achieve our goals for this sprint, we believe there is always room for improvement. We plan to implement more rigorous testing procedures to catch any issues before they get logged as bugs. Additionally, we will continue to look for ways to optimize the application's performance to manage heavy traffic, something that we expect during an ongoing tournament.