COMP 3004 - Deliverable #3
System Architecture and Design
Brackit - Mobile Tournament Bracket Creation

Metadata

Team / App Name: Brackit

Team member names

Name	Student ID
Jaime Herzog	101009321
Suohong Liu	101002340
Xiyi Liu	101004577
Alex Trostanovsky	100984702

Contents

Architecture

1 Description

1.1 Functional & Non-Functional Requirements

In developing `Brackit`, we set out to address an urgent need by tournament organizers and attendants to visualize, manage, and interact with double elimination brackets on their mobile devices. At a high level, we committed to developing a product that will meet the following **functional requirements**:

1. Tournament Organizers (TO's) can create, host, maintain, and visualize double elimination brackets.
2. Registerd `Brackit` Users, as well as Guests, can use the application to join created tournaments.
3. `Brackit` will store and maintain user profiles that will describe users' history:
   - Matches won/lost
   - Tournaments entered/created

In terms of **non-functional requirements**, we believed `Brackit` should be *usable* on mobile devices. `Brackit` users should be able to:

- View and access all components (Brackets, Rounds, Matches) of a tournament on an Android device.
- Seamlessly enter tournament competitors to brackets on an Android device.

Conceptually, `Brackit` needed to support the creation and maintenance of the following *components*:

- *Tournament*: The highest level of abstraction utilized in Bracket creation. A tournament acts a *container* for brackets. `Brackit` supports double-elimination tournaments, where competitors cease to be eligible to win the tournament after losing two matches [1].
- *Bracket*: Given the number of entrants and their corresponding seeds (ranks), Double elimination brackets dictate competitor matchups and the progression of competitors through the Winners and Losers brackets. Brackets contain a dynamic list of Rounds.
- *Round*: Rounds contain a dynamic list of Matches.
- *Match*: Matches pair the strongest and weakest (according to rank) players in a Round.

2 Justification of Architectural Style Choices

2.1 Object-Oriented Architectural Style

As described above, a Double Elimination Tournament mobile management application must maintain a set of well-defined entities (i.e. a Tournaments, Brackets, Rounds, and Matches) with predetermined relationships. For example, given n competitors, a correct double elimination tournament will contain $\lceil \lg n \rceil$ rounds in the Winners bracket and $\lceil \lg n \rceil + \lceil \lg \lg n \rceil$ rounds in the Losers bracket. In addition, the progression of competitors can be calculated at the creation of a tournament, and handling this progression follows a deterministic approach (e.g. The winner of Match 1 of Round 1 in the Winners Bracket will always progress to Match 1 Round 2 in the Winners Bracket - see Figure 1 for an illustrative example).

Therefore, to encourage an efficient decomposition of the algorithm and entities associated with Double Elimination Tournament creation, we decided to model the architecture of `Brackit` using an **Object-Oriented** (OO) architecture. Specifically, we chose to model each of the components of our application as objects. This allowed us to encapsulate the expected behaviour of each of the tournament objects while maintain a valid separation of concerns. To further explicate the validity of the choice of an OO architecture for `Brackit` consider the dynamic nature of Tournament creation.

A tournament bracket acts as a container for rounds, which themselves act as containers for matches. To handle the progression of a competition, the data associated with each match (i.e. which competitor won or lost) should be self contained within the match object instantiation, but also must be accessible through attributes of that object. Therefore defining the Match construct as an object allows the definition of self-contained class methods and attributes that achieve these intended behaviours.

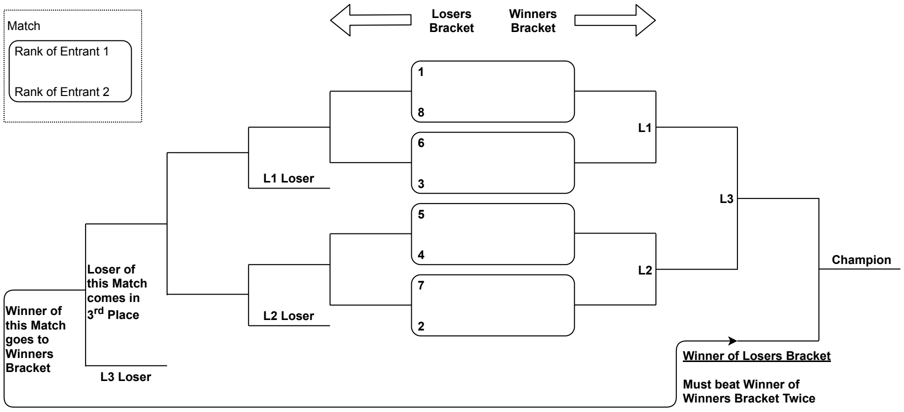


Figure 1: Seeded Double Elimination Tournament Chart for 8 competitors. (Adapted from [2])

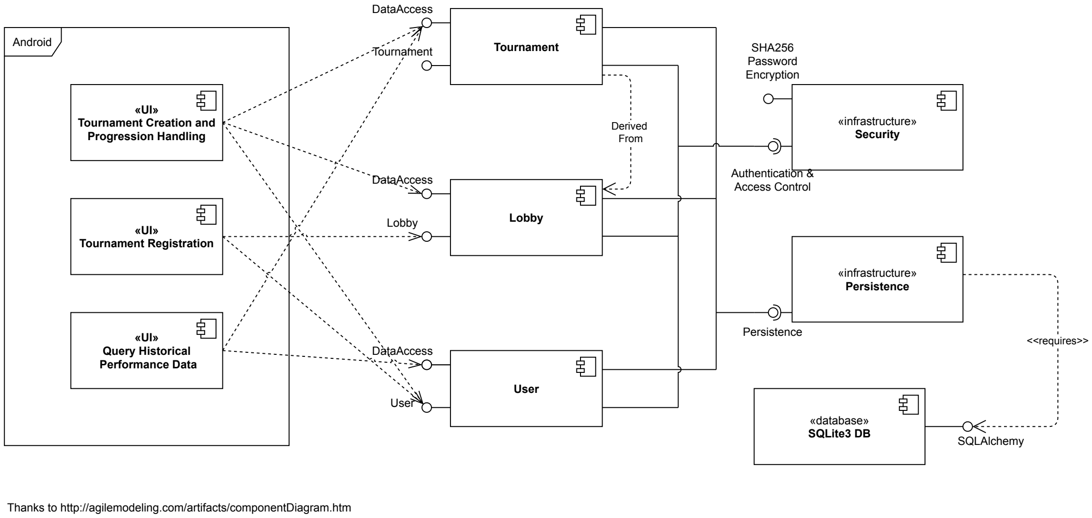
2.2 Client-Server Architectural Style

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

3 Architectural Diagrams



Thanks to http://agilemodeling.com/artifacts/componentDiagram.htm

Figure 2: `Brackit -` UML 2 Architectural Component Diagram

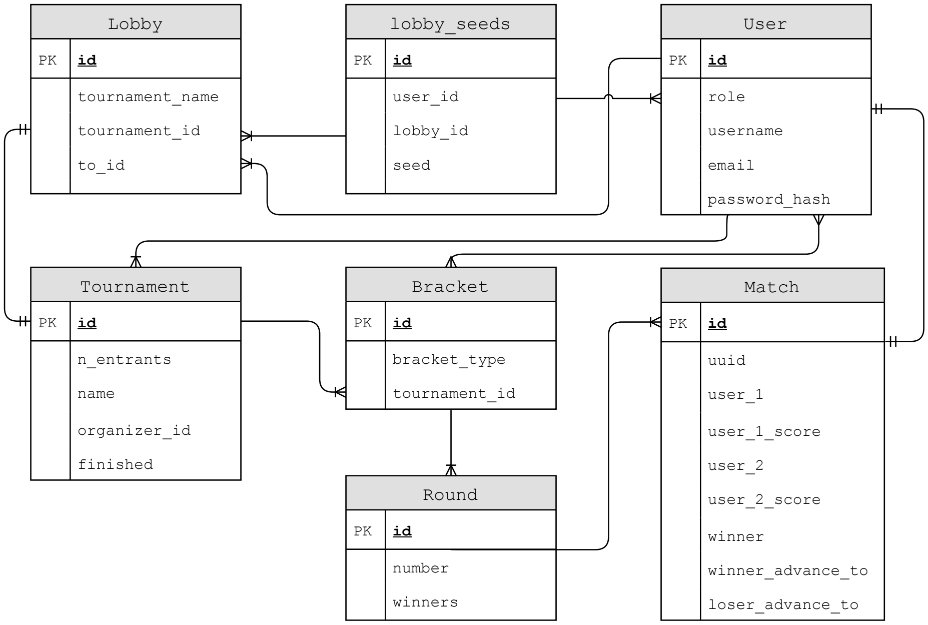


Figure 3: `Brackit -` Entity Relationship (ER) Diagram

Design

1 Description and Rationalization

- Use clear description of the structure of the components and its externally visible interfaces
- Clarify the physical location of where the classes will reside (e.g., on the client, on a server), as well as any external API
- Include references to your system's architecture (patterns, abstractions, data structures/ algorithms)
- An analysis of how your design minimizes coupling and accommodates changing requirements

Design Diagrams

Thanks to http://agilemodeling.com/artifacts/classDiagram.htm#CompositionAssociations

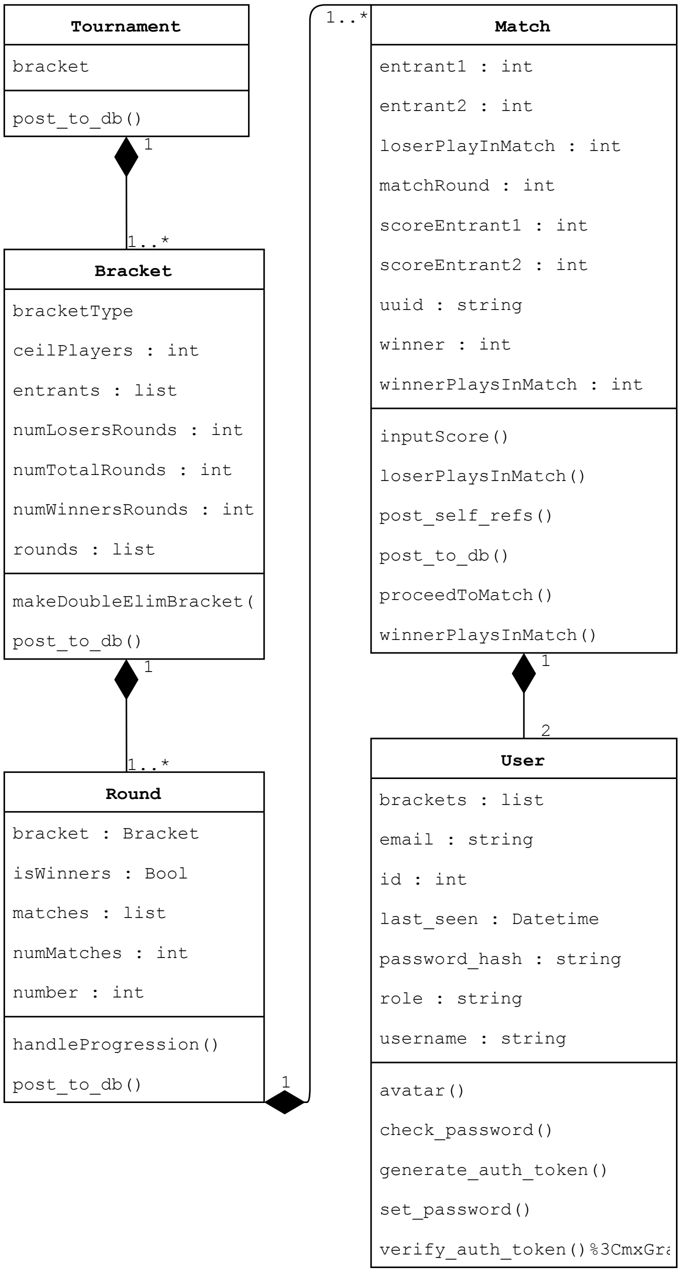


Figure 4: `Brackit` - UML Class Diagram

References

[1] Wikipedia. Double-elimination tournament — Wikipedia, the free encyclopedia, 6-October-2019. [Online; accessed 14-March-2020].

[2] candied-orange (https://softwareengineering.stackexchange.com/users/131624/candied orange). Tournament bracket algorithm. Software Engineering. [Online; accessed 14-March-2020].