CITS4401

Requirements Analysis Document

Badminton Automated Game Scheduling System (BAGSS)

Sarah Connor 21707394
Nerces Kahwajian 21592645
Damon van der Linde 21506136

1st June 2018

Table of Contents

1. Introduction

1.1. Purpose of the system

The purpose of BAGSS is to manage the scheduling of games for a badminton club. Having an online, automated system will make it easier and faster for the club members to organise their games. The system also has the role of managing teams, players and their details. Players need to be able to access the system to reschedule their games. The system also updates an ordered ladder by receiving the game outcomes from the players.

1.2. Scope of the system

The following project will involve the development of a Badminton Automated Game Scheduling System (BAGSS) that is controlled by members of a badminton club. The system automatically schedules badminton games and books courts based on player availability. The functionality of the system will enable its users to view a ladder showcasing each team's position on the leaderboard, while providing them with updated ladders every week. This system will also enable the participants to define their availability, reschedule matches accordingly as well as notifying other affected players if the reschedule is successful. The system will manage 4 courts and 32 teams that are made up of a total of 64 members, however it will not handle registrations of new users as this is conducted by a third party. Each team will input their game's score after each match into the system through the web interface on a computer or mobile device.

1.3. Objectives and success criteria of the project

The success of the project will be evaluated through the completion of the project deliverables that will be defined below in further detail. Other aspects that will also be considered in evaluating the success of the project is the adoption rate of the system amongst badminton clubs as well as through conducting informal surveys across its user base.

1.3.1. Deliverables

- Requirements Analysis Document (RAD)
- System Design Document
- Base implementation of prototype
- Iteration on the base implementation to accommodate more of the requirements
- Prototype display and presentation

1.4. Definitions, acronyms, and abbreviations

BAGSS = Badminton Automated Game Scheduling System
Mbps = Megabits per second
MB/s = Megabytes per second
DNA = Did Not Arrive
MTBF = Mean Time Between Failures
OS = Operating System

1.5. Assumptions

-If a player selects a date for availability, it is assumed they are available for all 24 hours
-BAGSS has limitless API access to the 3rd party court booking system
-Players can only set and edit availability for the current and following month

1.6. Overview of document

The remainder of this document will cover the existing implementation as well as breaking down the proposed system's functional and non-functional requirements. There will also be a series of scenarios which describe the basic user interaction with the system as well as which functional and non-functional requirements they involve.

2. Current System

The current system implementation focuses on a predominantly manual based approach whereby the scheduling, team assignment and availability is handled in person. This system has many drawbacks, as it is time-consuming and makes it very difficult to effectively organise all the club members, as for example, each member will need to be contacted separately to determine whether they consent to a reschedule.

3. Proposed System

3.1. Overview

The proposed system (BAGSS) will provide its users with an easy to use portable management system for their badminton tournaments. Allowing users to reschedule games based on the affected members' given availability.

3.2. Functional Requirements

3.2.1. System Context

- BAGSS will provide its users with an updated ladder every week
- The different teams will be ranked on the “ladder” according to their number of wins. The team that features the most wins will be ranked at the top and the team with the least at the bottom
- The teams consist of two players (doubles)
- Matches are conducted against a team that is either above or below them on the leaderboard
- Matches are determined by whether the week is odd or even
    - If the week is odd, then the team plays against the team above it
    - If the week is even, then the team plays against the team below it
    - The team that is ranked first does not play when it is an odd week
    - The team that is ranked last does not play when it is an even week

3.2.2. Schedule

- The match schedule will be determined by Monday 17:00
- The matches can be played anytime between Tuesday 12:00 and Sunday 23:00
- Teams can be removed from the ladder if required
- Teams can also be added into the ladder; however, they will assume the bottom position in doing so
- The matches are 90 minutes long
- BAGSS will schedule the matches
- BAGSS will book the courts for each of the matches
- If there are any issues the system administrator will be automatically notified

3.2.3. Reschedule

- Players can reschedule up until Tuesday 17:00 if it is more than 24 hours before their match
- BAGSS will present the player with a list consisting of alternative times which are subject to the availability of the remaining players in the two teams
- The player will choose a new suitable match time for them
- The other players will be notified of the requested change and prompted to accept or reject
- The match will only be rescheduled if every player from the affected teams have confirmed the new match details by 17:00 on the day prior to the original match
- Otherwise the reschedule is automatically cancelled
- No limits are placed on the number of rescheduling attempts

3.2.4. Forfeit

- Any player can forfeit the match which they were participating in
- All the other players are notified of this occurrence
- Forfeits are handled as a loss for the forfeiting team and a win for the opposing team

3.2.5. Editing of Player details

- Players can alter their availability schedule each week
- Default availability is assigned to new players until altered
- Current availability carries over to the following week if no changes are made

3.2.6. Administrator

- The ability to manage teams by creating a new one, adding additional players or removing them from existing teams
- Manage player details
- Manage ladder alongside the schedules
- Can run report and audit functionality

3.2.7. Match Outcomes

- The players will be able to fill out the scores of the match
- Players can report (DNA) for teams that did not arrive at the match

3.3. Non-functional Requirements

3.3.1. Usability

The system needs to be usable by people with little to no technical knowledge. They should also be able to navigate around the system without needing any additional documentation. Tasks such as requesting a reschedule of their game and updating their availability should not take any user more than 10 minutes to complete. When using the system for the first time, we expect that users will make a maximum of 5 errors within a 10-minute period.

To aid usability, the interface of the system will be designed to look like programs the users have likely encountered before. For example, having an initial login page that takes you to a home screen, similar to websites such as Facebook and Netflix. Thus, users will find the familiar layout easy to navigate.

3.3.2. Availability

According to Microsoft "the metric used to measure availability is the percentage of time that a system is capable of serving its intended function.". [1] For BAGSS, the availability of the system is the percentage of time that the service is up and running. The aim for the BAGSS is to have 99.99% availability which equates to 52.56 minutes of downtime for a system running 24/7/365 (24 hours per day/seven days a week/365 days a year). BAGSS' metric for downtime is when more than half of its users cannot access the system. In cases of software issues this will be solved by a system restart. Assuming the average SSD takes thirty seconds to restart and boot into the OS, this means that BAGSS can be restarted 105 times per year while still maintaining a 99.99% availability.

3.3.3. Reliability

Reliability is a form of measure to calculate the chances of a system failure. A commonly used measurement is mean time between failures (MTBF). MTBF is the average amount of time it takes for a system to fail. For BAGSS the MTBF can be calculated using the respective equation where:

$$MTBF = \frac{total\ elapsed\ time - sum\ of\ downtime}{number\ of\ failures}$$

Assuming a worst-case scenario, total elapsed time will be 8,760 hours ($24\ hours \times 365\ days$), sum of downtime would be 0.876 hours ($\frac{52.56}{60} \times 100$) and number of failures would be 105 (52.56×2).

Plugging this into the equation gives us:

$$MTBF = \frac{8760 - 0.876}{105}$$

Hence $MTBF = 83.42\ hours$

A best-case scenario will have almost zero failures hence a MTBF of 8,760 hours which is the equivalent of 365 days.

Aside from the MTBF, BAGSS will incorporate systems to reduce data loss during failures to zero. Such examples would be redundancy. When BAGSS encounters a failure it will attempt to return to a stable state before the failure and immediately notify a system administrator of the exception.

3.3.4. Performance

The tasks users will perform on the system are relatively short, simple tasks. However, some user tasks may be time critical. An example of such an instance involves a user having to submit a reschedule request near the deadline of when all such changes must have been handled. Therefore, they need to be able to complete tasks quickly. This means there needs to be rapid response times and the loading of each web page should not take more than 5 seconds, when used with an average internet connection speed of around 24Mbps.

The leader board will refresh once a week, on Monday at 17:00. This allows each team enough time to submit their match outcomes. The system will also be able to support a minimum of 65 users concurrently to account for the possibility of all 64 players and 1 administrator using the system at the same time.

Other factors such as response time, high throughput, low resource utilisation, low bandwidth consumption and short data transmission times should be taken into consideration when designing the system and choosing a server. If not considered, all the above can collectively heavily hinder a system's performance and cause issues and frustration for the end user.

3.3.5. Supportability

BAGSS needs to be able to support modern browsers and mobile platforms such as Google Chrome, Opera, Firefox, Microsoft Edge, iOS and Android. The web interface should implement a framework that allows for the screen elements to be adjusted and optimised according to the accessing device's screen resolution. An example of such a framework is Bootstrap.

3.3.6. Scalability

Scalability, within the context of this document, is the measure of how effectively an application can grow and manage an increased demand in performance. [1] Scalable systems tend to be advantageous as they are more adaptable to a user's changing needs and show that the system is stable and competitive as it can grow with demand. [2]

BAGSS will be built with Python 3, using locally stored lists and CSV files. Assuming an average 32-bit computer system will be running BAGSS, a standard Python list for that particular system will be able to store over 500,000,000 elements [3] while a CSV file is essentially limitless. The biggest hurdle for scalability would be the element access and append time. Since new players will be added and court availability checked via iteration, the time complexity of these two need to be taken into consideration. According to the official Python documentation, append and iteration have a time complexity of (O)1 and (O)n respectively. [4] Since BAGSS is a localised system and will be made for members of only one club, the time complexity would be negligible hence the system can be considered to have no issues regarding scalability in terms of software.

Resources are not the only thing that make a system scalable. Any system that can have extra resources added onto it through add-ons or any other downloadable content is also classified as scalable. [2] Hence, BAGSS is classified as scalable as it can have extra features added to the system without requiring major changes in architecture.

3.3.7. Portability

BAGSS will be accessible via a website where all the work will be carried out by the back-end server. The website will essentially be used to show the results returned by the server to the user. The only issues regarding portability will be whether the front-end supports a particular web browser and device. As discussed in section 3.3.4, it can be seen that different browsers and devices have already been taken into consideration which indirectly reinforces the system's portability.

3.3.8. Implementation

BAGSS does not have any major implementation constraints. The largest constraint would be the requirement to use Python 3 for the system. As BAGSS is a relatively small-scale project, there is not any red tape to navigate such as company policy and physical, electrical, and software interfaces. As BAGSS will most likely be hosted on a Linux or Windows OS, we have ensured that any 3rd party libraries are readily available on both OSs.

3.3.9. Security

The system deals with confidential information such as people's email addresses and phone numbers hence security is of utmost importance. According to the IEEE, "Software security is the idea of engineering software so that it continues to function correctly under malicious attack." [5] which means that aside from the basics, we need to ensure that the system will be able to function while under attack and concurrently prevent any data leaks. This requirement can be fulfilled by practicing correct programming techniques including but not limited to, setting correct access privileges, limiting the number of administrator accounts and ensuring there are no backdoors available to be taken advantage of.

Other security methods can be implemented to compliment the system's security such as:

- Forcing all accounts with administrative privileges to register for a 2FA service (2 Factor Authentication) such as Google Authenticator.
- Setting a maximum number of failed login attempts for all users within a certain time frame. Exceeding the set number of failed logins will cause your account to be automatically locked for a set period of time. This will prevent user accounts from being brute-force attacked.

- Installing an SSL certificate using a service such as Let’s Encrypt which will help keep interactions between the system and the user private.

3.4. Scenarios

Each scenario has been designed to demonstrate at least one functional and one non-functional requirement, as listed at the beginning of each scenario.

3.4.1. Scenario 1: Rescheduling game

Functional Requirement:	Rescheduling a game
Non-functional Requirement:	Usability

Scenario name	rescheduleGame
Participating actor instances	Peter: User Robert, Norma, Gladys: Player
Flow of Events	1. Peter logs onto the system on Monday and requests a reschedule for his Wednesday game. 2. The system provides him with a list of alternative game times from which Peter selects Thursday at 11:00. 3. The system then contacts Robert, Norma and Gladys who are involved in the original game and asks them to confirm the change. 4. Robert and Gladys accept but Norma declines. 5. Peter is then informed that the reschedule has been unsuccessful. 6. Peter wanting a change, starts the process again by requesting a new reschedule and chooses a different time slot of Friday at 14:00 from the list of alternative times provided by the system. 7. The system again contacts Robert, Norma and Gladys to confirm the change. 8. This time Robert, Norma and Gladys all agree, and the system reschedules the game and notifies Peter as well as Robert, Norma and Gladys.

3.4.2. Scenario 2: Editing details by user with little technical knowledge

Functional Requirement:	Editing of Players Details
Non-functional Requirement:	Usability

Scenario name	editDetails
Participating actor instances	John: User
Flow of events	1. John needs to update his availability but has not used the system before and has had minimal experience with computers. 2. John loaded the website and had access to the login screen almost immediately. 3. John then logged in without any issues and was directed to the post-login homepage. During this process, he twice failed to select the correct page for updating availability before making the correct choice on his third attempt. 4. He manages to successfully update his availability within 5 minutes

3.4.3. Scenario 3: Forfeiting a game

Functional Requirement:	Forfeit
Non-functional Requirement:	Supportability

Scenario name	forfeitGame
Participating actor instances	Ruth: User Edward, Earl, Gertrude: Player
Flow of events	1. Ruth has a game scheduling at 14:00 on Tuesday. She arrives at the court 10 minutes before her game and waits for her doubles partner to arrive. 2. She then gets a message 5 minutes before the game from Edward saying that he is unable to make the game. She has to forfeit the game right away and therefore she needs to be able to do this on her Android phone using the Google Chrome application. 3. She manages to log into her account and use the forfeit function. 4. The system then sends a message to Earl and Gertrude who are on the other team to inform them of the forfeit and records the game as a loss for the forfeiting team and a win for the other team.

3.4.4. Scenario 4: Addition of a new team

Functional Requirement:	Addition of a New Team
Non-functional Requirement:	Scalability

Scenario name	addTeam
Participating actor instances	Chris: Administrator James, Frank: Player
Flow of events	1. James and Frank recently became members of the Badminton club and decided to form a team together. Chris, the system administrator has been informed of this occurrence and logs into the system using the administrative login portal. 2. Despite already having 64 players in the tournament, Chris then adds James and Frank as a team into the system, which the underlying permissions allow as the system is capable of handling more than 64 players. 3. The system will then place James and Frank at the bottom of the ladder and they will be assigned matches from the next allocation session onwards.

3.4.5. Scenario 5: Recording the match results

Functional Requirement:	Recording Match Results
Non-functional Requirement:	Performance

Scenario name	recordResults
Participating actor instances	Bernard: User Jane: Player
Flow of events	4. It is a Thursday and Bernard and Jane have just won their badminton game with a score of 12-7. Bernard then logs into the system and selects the function to record their game. 5. Bernard inputs that they were the winners and inputs the score of the game. 6. Being very eager badminton players they each log back into the system on Monday evening to check the week's results on the leader board, knowing from experience that the leader board is updated once a week, on Monday at 17:00.

4. References

[1] Technet.microsoft.com. (2003). *Understanding Availability, Reliability, and Scalability*. [online] Available at: https://technet.microsoft.com/en-us/library/aa996704%28v=exchg.65%29.aspx?f=255&MSPPError=-2147217396 [Accessed 6 May 2018].

[2] Techopedia.com. (n.d.). What is Scalability? - Definition from Techopedia. [online] Available at: https://www.techopedia.com/definition/9269/scalability [Accessed 6 May 2018].

[3] Stack Overflow. (2009). How Big can a Python Array Get?. [online] Available at: https://stackoverflow.com/questions/855191/how-big-can-a-python-array-get [Accessed 7 May 2018].

[4] Wiki.python.org. (2017). TimeComplexity - Python Wiki. [online] Available at: https://wiki.python.org/moin/TimeComplexity [Accessed 9 May 2018].

[5] Mcgraw, G. (2004). Software security. IEEE Security & Privacy Magazine, [online] 2(2), pp.80-83. Available at: https://ieeexplore.ieee.org/document/1281254/ [Accessed 11 May 2018].