Rewards-based gamified polling system to provide song feedback to artists in a restaurant setting

Vikas Kumar Gupta

MSc in Computer Science
The University of Bath
2021

Rewards-based gamified polling system to provide song feedback to artists in a restaurant setting

Submitted by: Vikas Kumar Gupta

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

It is often quite challenging to place a song request of our likes in a social gathering, and more specifically, in a restaurant setting. Several software solutions are available to help users listen to songs in a more democratic way (O'Hara et al., 2004), but they are limited to just playing the songs based on the number of votes, which only benefits the restaurant's customers. This project introduces two new entities, the restaurant and the budding artists, into the equation and aims to benefit all the participants by solving their core problems. The restaurants find it challenging to regulate their offers and rewards and also to choose a cost-effective promotion strategy (Jackson, Titz and Defranco, 2004). The budding artists can get benefit by receiving feedback on their work and ultimately help the music industry in finding promising talents (Zwaan and ter Bogt, 2009). Finally, the solution provides more benefits to the customer's as they drive this solution by providing their inputs and getting rewards in return. This project aims to develop an ecosystem where the restaurant, its customers, and budding artists coexist and get benefited from each other. The idea is to motivate the customers to compete for offers and rewards in an event controlled by the restaurant, called the "Game-Mode". The game mode randomly selects a song from the list of new unreleased songs uploaded to the application by budding artists and runs a preview of the song. The customers listen to the preview and share their input using "Like" and "Dislike" votes. This activity registers the customer's votes and generates a feedback report for the artists. The application decides the game mode winner who is entitled to the rewards regulated by the restaurant. The customers get offers and rewards from the restaurant, the restaurant can regulate rewards through the game mode, and the artists get the much-needed feedback on their work.

Contents

List of Figures

List of Tables

Acknowledgements

Many thanks to Alessio Guglielmi (Reader, Dept. of Computer Science) for providing me with the opportunity to work on my project idea.

This project would not have been successful without the guidance and support of Dr Alan Hayes (Senior Lecturer, Dept. of Computer Science); thank you so much for providing the feedback and the much-needed encouragement throughout the dissertation period.

Finally, I would like to thank my wife and brother and the rest of my lovely family and friends for believing in me, supporting me, and encouraging me at each step of this project.

Chapter 1

Introduction

1.1 Document Map

This section will explain the flow of the document. Chapter-1 provides an introduction to the project, followed by the motivation and the model for the application. Chapter-2 starts with the scenario-based problem description followed by a literature and technology review. Chapter-3 looks into the related work and evaluates them with the proposed solution. Chapter-4 defines the aims, objectives, and deliverables of this project, and Chapter-5 covers the requirements analysis and requirements specification and further breaks it down into functional and non-functional requirements of the project, including the required resources. Chapter-6 discusses the technical and non-technical project risks involved in the development of this project. Chapter-7 discusses the design of the proposed application and consists of design goals, system overview, system architecture and user interface details. Chapter-8 talks about the implementation of the project in detail. Chapter-9 tells about the testing strategy adopted in this project. Chapter-10 has the details of the survey conducted for this application. Chapter-11 discusses the limitations and the future work, followed by Chapter-12, which concludes the dissertation report.

1.2 Background

Modern technology has revolutionized the music industry and made it possible to access the song of your choice anytime and from anywhere. However, in a social gathering (for example, restaurants or pubs), the song playlist is generally controlled by a disc jockey (DJ). The only way to play songs of your choice is to request a DJ. The primary aim of this application is to provide freedom of song selection to the audience in a social gathering.

The primary aim of this application can be leveraged to solve the other two closely related issues faced by artists and restaurants. For artists, new or established, getting feedback about their work is crucial. The feedback from the live audiences will further help them decide whether to release a particular song or work over it to improve. New artists will have proof and data to show to record companies how their songs have worked in live audiences and will ultimately benefit them in their careers.

Similarly, a restaurant has to decide on its promotion strategy to compete in the market. Restaurants try several promotional strategies to retain their customers or to get hold of new

ones. The promotion strategies range from loyalty programs to sharing coupons to happy hours, and sometimes they even provide discount offers on food and beverages. A solution that could help the restaurant control the offers or the number of coupons they release could be of great importance. The restaurant can decide how many offers or coupons to distribute in real-time and balance their profits and expenses.

The other part of this application is related to gathering song feedback for unreleased songs by engaging the live audience in a voting game motivated by rewards. The intent here is to leverage the primary aim of this project, i.e., by using the voting power of the live audience to provide feedback to the artists for their unreleased songs. The restaurant offers rewards for each voting round to keep the audience motivated, which will be a good promotion strategy. Artists can use this application as a testing ground for an unreleased song and get feedback from the live audience in real-time.

1.3 Motivation

Several studies in the area of songs and music try to decrease the dependency on the DJ and give more control to the audience. (O'Hara et al., 2004) is one of the early examples in this area that introduced the concept of voting for a song in a social gathering. The study demonstrates a mechanism to democratically select the songs. Mobile devices and mobile applications were very limited in use during the time of this study. The study involved placing a jukebox named "Jukola" (O'Hara et al., 2004) in a cafe bar along with a 15-inch public display and hand-held devices. The application uses a public display and a hand-held device for nominating and casting a vote.

The proposed solution derives its motivation from (O'Hara et al., 2004). Instead of using a public display in the proposed solution, the application will be accessible to its users by installing the application on their mobile device or using the web application. The host can control the whole event from his login and configure the game mode and settings. The users will be able to join the event and vote for their preferred song. The game mode allows users to compete for the surprise reward set up by the host. After listening to the song preview, the users have to vote for the song they liked. The user who voted first for the song which got the maximum votes is the winner.

1.4 Model

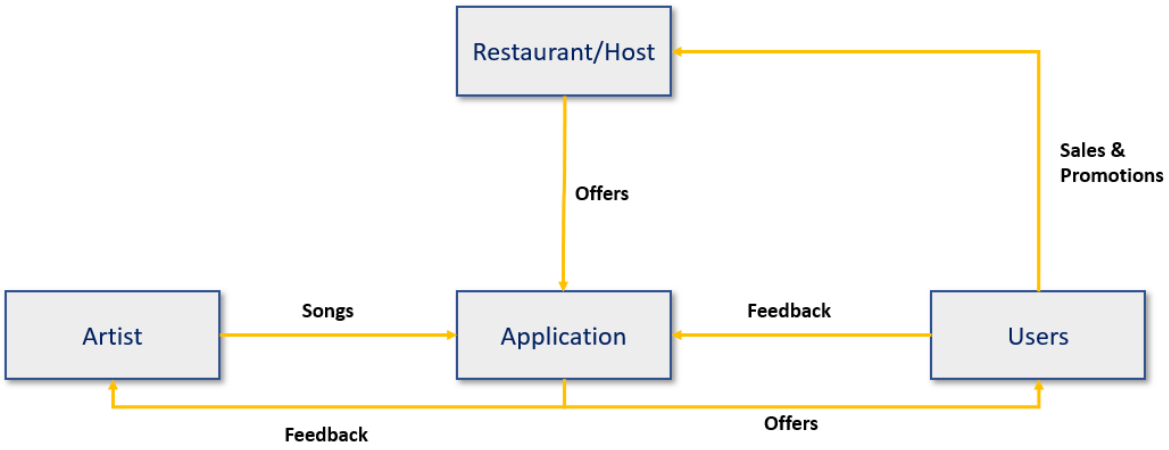


Figure 1.1: Business Model

The main stakeholders for the proposed solution are the restaurant (host), artist, and users. A restaurant hosts this application, the artists log in to the application and upload their songs. The users will join the hosted event and vote for their favourite songs. Users will also be able to compete for rewards by participating in the game mode of the application. The options provided to the users are the songs uploaded by the artists. Each round of the game mode will have some reward associated. These rewards will be the promotional activities set up by the restaurant (host). The application will gather data from the live audience from each game mode round to generate a report. The artist gets the report as feedback from the live audience. The restaurant makes promotions, the users get rewards for sharing their music taste, and the artist receives the feedback report.

The design of the proposed solution benefits each of its stakeholders.

Chapter 2

Literature and Technology Review

2.1 Problem Statement

Below mentioned scenarios explain the problem statement from the perspective of an application user, an artist, and a restaurant manager:

User Scenario:
I am in a social gathering (restaurant). I wish to request the song that I want to listen to.

- How can this be made possible without having any conflict?
- Do I get any benefit by sharing my personal music preference?

Artist Scenario:
As an artist, I have recently composed a song. I wish to get feedback from a live audience.

Restaurant Manager:
As a restaurant manager, I wish there could be some cost-effective way to regulate offers and rewards and have promotions for my restaurant.

To address the problem faced by the above stakeholders below points have been identified as the major components of the proposed solution:

- For Users: Polling votes to play music
- For Application: Gamification and surprise rewards for motivation
- For Restaurants: Rewards as promotion strategy in restaurants
- For Artists: Benefits of receiving song feedback

2.2 Literature and Technology Review

A literature and technology survey is conducted to gain more insight into the proposed application's above-identified significant components.

Polling votes to play music:

Various approaches have been tried to find a suitable way to play songs in a social setting. The below section will discuss some essential works in the domain related to collaboratively listening to songs in a social gathering. It also reflects the progressions made in the past two decades in terms of design and implementation.

Figure 2.1: Jukola public-display and hand-held device (O'Hara et al., 2004)

"Jukola" (O'Hara et al., 2004) creates a great sense of community but, it is heavily dependent on the user's input, right from queuing a song to voting and playing. The system implementation requires a display screen and hand-held devices for each table, which may not be feasible for every restaurant. It also lacked a mechanism to handle social conflicts related to song selection. Moreover, the host can skip any song and, users do not have any control in such an event. It raised questions about the transparency of the process and the limitations to the user actions.

Figure 2.2: Users interacting with MUSICtable tabeletop (Stavness et al., 2005)

(Stavness et al., 2005) performed a study that uses a tabletop display for getting user inputs. This study shows a map visualization of songs to the users, with each colour on the map corresponding to a style of music, and every grid square on the tabletop represented at most five songs. This study let users navigate around the map and explore music. The system randomly selects a song out of the five songs mapped to the grid square at the position of the

selection pointer. Although being an innovative concept, this study did not display any playlist of songs to select from; instead, the users select a category of music they were interested in and steered their way towards that category using the controls available on the tabletop display. There was no voting element involved in this study, and it required group effort in selecting the type of songs to be played.

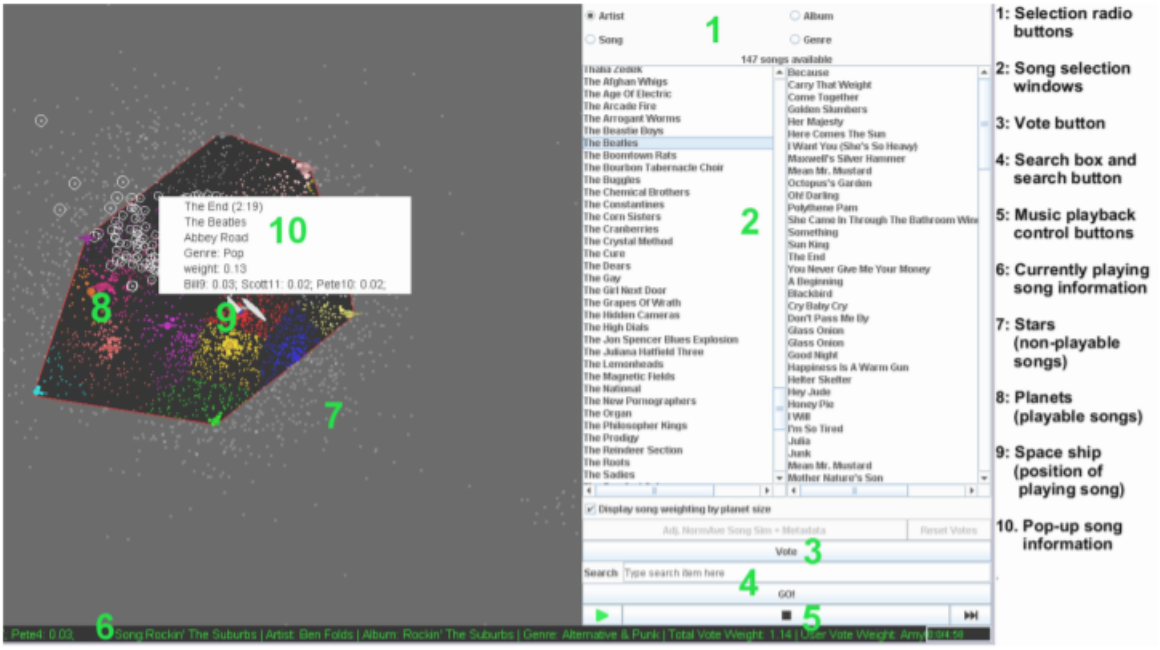


Figure 2.3: Screenshot of PartyVote system interface (Sprague, Wu and Tory, 2008)

The study conducted by (Sprague, Wu and Tory, 2008) provided a space-themed visualization of potential songs on the screen placed as a planet. Users were provided with an interface to search and vote for a song. The system provided transparency in the selection of songs. The users can see how other votes from other users affect the song selection and can accordingly cast their votes. This system uses a monitor to display the visualization and a keyboard and mouse to search songs and cast votes. This study is limited to providing a mechanism for the users to collaboratively vote for songs in a public place and visualizing the voting patterns. The system also guarantees the user that their song selection will get played within 2 hours. There are no song rankings generated in the process.

Figure 2.4: Community-based Music Voting Service interface (Koskela et al., 2009)

Another study performed by (Koskela et al., 2009) uses a tablet and a music voting service application. This study aimed to assess the regulatory factors such as attitude, social, and perceived behaviour of a user while using a music voting service in a social setting. It presented the users with a song playlist and a voting event. On clicking the voting event, it provided the users with options to vote for a song. This study maintained transparency throughout the voting process. The study considered some evaluating factors such as usage of images, playfulness and usefulness of the application, the price for the service, mobile experience, and internet experience. They projected the music service as a paid service. Users had to decide on the usefulness of the service and the actual value they are getting in return for the service charge. The results showed that the users well-received this concept. They well appreciated the usage of tablets. (Koskela et al., 2010) further extended this study, and the results showed that two crucial factors to be considered for community music playing applications as user's behaviour and user's experience with mobile application usage.

Figure 2.5: UbiRockMachine public interface (Kukka, Patino and Ojala, 2009)

(Kukka, Patino and Ojala, 2009) created a client-server-based mobile application with Bluetooth

connectivity for nearby devices. This application provides features such as searching, voting, and rating for songs, chatting, uploading songs by artists, downloading songs by users. The application covers a vast portion of the proposed solution. However, there are some shortcomings when compared to the aims of the proposed solution. The local artists provide all the songs. The restaurant is not involved in any of the processes apart from providing the premise for the study. The application also requires a public display to show the ongoing voting process. Not enough information is present in the article about sharing the song report with the artists. The study is quite detailed in terms of qualitative and quantitative aspects. The users appreciated the whole concept, especially the ability to download songs free of charge. The use of mobile devices was quite popular among the users.

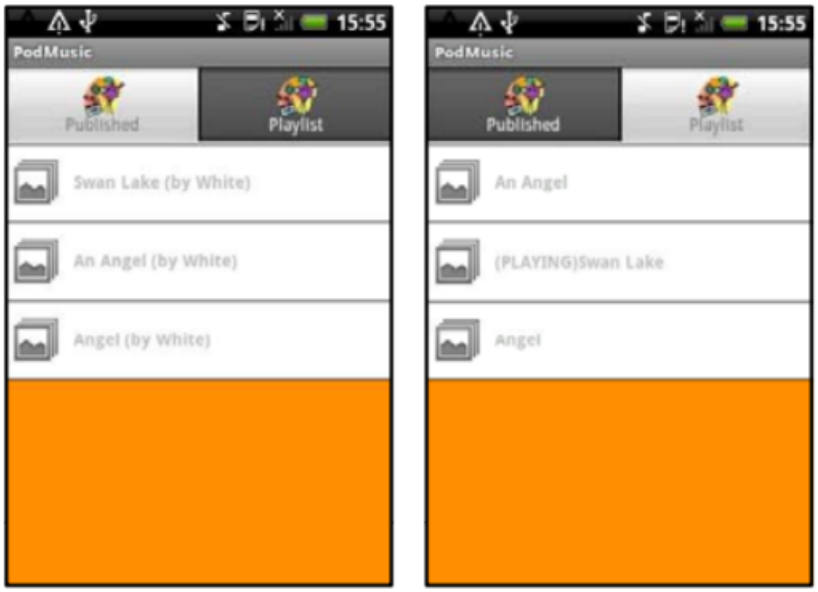


Figure 2.6: What a juke! interface (Chen, Yavuz and Karlsson, 2012)

(Chen, Yavuz and Karlsson, 2012) developed a content sharing (not exchanging) application based upon a publish-subscribe model with the intent of letting users access their media content and stream it to a connected jukebox. This way application handles any copyright issues. The application deploys on a mobile device or a computer that connects to a sound system. A manual override option is also available for the administrator to have overall control. Users can access all the audio files on other user's devices and allow other users to access their audio files. This application could cause some security issues as it can access all the audio files on the device. Also, there are chances that inappropriate content may get played by mistake. There is no voting mechanism, but the users can queue their songs in the playlist.

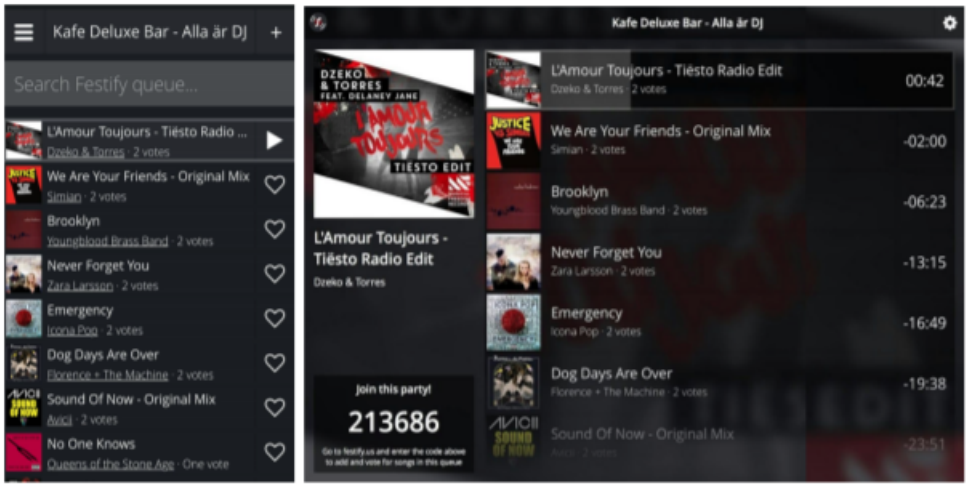


Figure 2.7: Festify - mobile and shared view interface (Müller, Otero and Milrad, 2016)

(Müller, Otero and Milrad, 2016) performed a study to capture the user experiences using collaborative music playing application in social settings, called Festify (Leo, 2014-2018). Festify is a web application that enables a user to create a party to which other users can join. The users can then vote for their song by searching the playlist. The user activities are visible on a shared public display. This study was performed at two different venues, and the results were collected to know "How and why do people interact with public music system?" and "How does the system affect the public space and social context it is situated in?" (Müller, Otero and Milrad, 2016). Results showed that the application increased group interaction and collaboration among the users. Users were also curious about sharing their taste of music with the group. This study provides a list of great suggestions for future work. Suggestions include photo-sharing of the event, a mechanism to express more than just casting a vote. Including user ranking for the tasks of adding songs to the playlist could make users compete even more. Although this application excels in voting and playing songs, it is only available as a web application, which could be a possible security threat. Also, it has limitations to support few internet browsers only. There is no song ranking generated in this application and no involvement of the restaurant.

The study performed by (Steininger and Gatzemeier, 2019) is an effort towards building a forecasting model for the music record companies to predict the success of a song (or new artist). Instead of using feedback from a live audience, this study crowd-sourced the rating task to the users on Amazon Mechanical Turk (Crowston, 2012). This study kept track of song ranking in the actual music charts and compared it with the crowd response to evaluate the effectiveness of their model. Forecasting the prediction of a song's success can be highly beneficial to both the artist and the record company. Results claim to improve the forecasting of success prediction by 30% using this study. This study is not based in a social setting although, it is closely related to the proposed solution in explaining the effects of voting to assess the success of a song before release. This study is comparatively recent among all the above studies mentioned in this section. The results are motivating to work on the proposed solution. Although this study shows promising results, it does not involve a live audience or takes any social setting into account.

This section looked into various studies related to democratic voting for a song in a social setting and evaluated the different voting methods in these studies. It is evident from the

evaluation of the above studies that users appreciate mobile applications compared to any public device for inputs. The idea of a democratic way of song selection was well appreciated. From a social point of view, these studies encouraged user interactions and group discussions among the users. The following section will look into the studies performed around gamification of applications and the effects of providing surprise rewards to the users.

Gamification and surprise rewards for motivation:

This section will look into some studies performed to assess the effects of gamification on applications and the effects of surprise rewards on users' motivation. The expected outcome from this section of the literature review is to help structure the proposed solution and evaluate if gamification and rewards should be considered in the solution to be developed.

(Cechanowicz et al., 2013) performed an empirical study to demonstrate the effects of gamification on three types of surveys of varying levels of gamification. The surveys include the three fundamental survey categories used in the market research industries: image identification, slogan matching, and a five-second quiz, with each survey, consists of 12 questions. These surveys correspond to the categories of the plain survey, partially-gamified survey, and fully-gamified survey (including theme, rewards, challenges, and user progress). A framework consisting of six elements was identified to design the surveys. The elements included core tasks and game mechanics along with the other elements of a fully gamified survey. This study aims to answer some of the fundamental questions related to the gamification of work-like applications. The main focus of this study was the questions such as the effect of gamification on motivation, demographics, experienced participants, and the quality of data produced by the users. Each respondent was provided with 36 questions to complete with a condition to answer at least one question to win the reward. The number of questions completed was the primary metric for analysis. The results were promising, as indicated by the increase in the number of questions completed with the increase in the surveys' gamification level, and the same effect was visible across all demographics. It was also observed that the motivation decreased as the game progressed to other blocks of questions in the fully-gamified surveys. This indicated a need for additional game elements such as varying the difficulty level, showing the overall progress, and mixing the question types. This study showed that gamification of applications leads to higher user engagement, and including new game elements will keep the user stay for a longer duration. The results indicate that the gamification of the application will motivate the users for using the application more as compared to a plain version of the application.

(Wu, Mattila and Hanks, 2015) performed a study to examine the effects of surprise rewards in a restaurant setting (as compared to the membership rewards) on the loyal customers in terms of three variables defined as delight, frustration, and satisfaction coupled with low and high values of cumulative customer satisfaction. Cumulative satisfaction refers to the overall satisfaction level that builds upon the member from the date of joining the membership plan. The variables were measured on a 7-point bipolar scale. The study was crow-sourced to Amazon Mechanical Turk (Crowston, 2012) for payment of USD 0.50, and the participants were briefed to pose as a member of the restaurant. The participants were required to read a dining experience and select the applicable value out of the four sets of semantic differential measures: dissatisfied–satisfied, unhappy–happy, poor job–good job, and poor choice–good choice. For performing the manipulation check, the users were asked two separate questions about assessing cumulative satisfaction and the effects of surprise rewards. The participants were asked to rate on a 7-point scale how they felt about receiving the surprise rewards and

how pleased they were with the services of this restaurant (based on their dining experience) to assess the changes in cumulative satisfaction. The results indicate that the customers with low cumulative satisfaction felt a significantly high level of satisfaction on receiving a surprise reward. There were no effects of reward types on a customer with a high level of cumulative satisfaction. The study successfully established that the surprise rewards are more effective for customers with low cumulative satisfaction and increases customer satisfaction. Thus, including surprise rewards in the proposed solution will motivate the users with low cumulative satisfaction, and it will work in favour of the restaurant by gradually converting the customers with low cumulative satisfaction to high cumulative satisfaction.

The paper review of 24 empirical studies performed by (Hamari, Koivisto and Sarsa, 2014) observes a mixed result in the context of gamification of applications. The study was performed to test three main areas, motivational affordance, psychological outcomes, and behavioural outcomes. The results indicate positive effects of gamification and mention the dependency on the context of the developed application and the users' demographics. The majority of the studies showed positive effects, but the results were varying for each of the three measurement metrics. Considering the positive effects of most of the studies, the gamification of the proposed solution seems promising.

This section covered a few studies performed to see the effectiveness of gamification of an application and what measures must be taken to keep the application more engaging for the users. Some of the noticeable suggestions were to include new game elements and surprise rewards which will keep the users curious and motivated towards more prolonged application usage. The following section will look into the various promotion strategies followed in restaurants.

Rewards as promotion strategy in restaurants:

Various studies have been performed on different types of promotion strategies followed by restaurants to retain their loyal customers and add new customers. Some of the common promotion strategies include loyalty programs, giving coupons, and monetary incentives. This section will look into the restaurant's promotion strategies, frequency, and customer effects.

The study by (Jang and Mattila, 2005) takes a closer look into the loyalty programs offered by restaurants to customers. It also examines the customer's enthusiasm and their expectations with such loyalty programs. The study was conducted through two rounds of focus group interviews (gender-specific) over six members (graduates from the hospitality industry). This study examined the various types of rewards strategies offered to a customer from the loyalty programs. These offers include immediate rewards, point-based rewards, necessary or luxury rewards, and monetary rewards. Most loyalty programs offer point-based rewards where the customers accumulate points that can be redeemed after reaching a particular value. These accumulated points are projected in comparison to the currency. Customers were more inclined towards getting immediate rewards instead of accumulating the points to get a bigger reward. The study also showed that the minimum threshold of points accumulated to redeem a reward plays an important role. The customer's preference increases for luxury rewards when the threshold is higher. Another type of reward that this study looked into was monetary and non-monetary rewards. Customers preferred monetary rewards over non-monetary rewards as they can be redeemed immediately. Finally, the study showed that the customer's motivation for loyalty rewards also depends on the quality of service, convenience, and entertainment. The results of this study will be helpful for the proposed solution in setting up the rewards

offered by the restaurants. The solution will let the restaurant decide on the type of reward they want to set up.

The study performed by (Jackson, Titz and Defranco, 2004) tries to find out details about the most common forms of advertising and promotions and the frequency of the promotions at restaurants. Standard promotions include radio advertisements, directory advertisements, newspapers, electronic/internet resources, direct mails, food samples, coupons, loyalty programs. The study was conducted through phone surveys and included 224 restaurants. The results indicated that the most common form of promotion was internet and electronic resources, followed by directory advertisements and food samples. The least preferred form of promotion was radio advertisements. Although the internet and electronics usage was limited to large restaurants, most respondents did not find any benefits from the advertisements. This study shows that internet and electronic advertisements work only for restaurants with larger capacity, but it will be interesting to see the results of deploying the proposed solution in a small-scale restaurant.

This section provided an insight into the restaurant promotion strategies and their effects on customers. The following section will examine the benefits of song raking for artists and artist and repertoire (A&R) managers.

Benefits of generating song ranking and feedback:

This section will look into the studies related to the music industry and the criteria for selecting a successful artist. This section is the final piece of the proposed application. It is more relevant for assessing the artist's performance and creating their profile to showcase to the A&R managers.

(Zwaan and ter Bogt, 2009) takes a closer look into the records and music industry and tries to identify the traits that an artist should have to be successful. It also discusses what artist and repertoire (A&R) managers look for in an artist. The study was conducted through interviews with the (A&R) managers. Two main factors were identified to explain an artist's success; first, related to earnings by being profitable for the record company. Second, the development of artistic skills and being recognized by other established artists. Apart from these, other factors included appearance, self-criticism, motivation, family background, and support system for the artist. From the point of view of an A&R manager, the most important factor was the potential of increasing sales. The results indicated that for an A&R manager to hire an artist, the uniqueness of the artist's skill and recommendations from other professional networks played an important role. The proposed solution will work upon reflecting these factors to display the potential of an artist in the report, which will help the artists and the A&R managers to make decisions.

Summary of Literature and Technical Review

To summarize, the literature and technology review section looked into various studies corresponding to identified components of the proposed solution.

- The studies related to voting applications designed for social settings bring along different levels of innovation. The studies ranged from simple voting mechanisms using hand-held devices to complex tabletop steering solutions to space-themed and map-themed visualizations. It was interesting to see and learn how different works approached solving similar problems.
- Some studies related to the gamification component showed mixed results, whereas some showed quite positive effects of gamification. It will be interesting to see the results of the gamification of the application.
- The studies related to restaurant promotions and customer responses also provided vital information to conduct social activities while keeping the customers engaged and increasing restaurants' sales and promotions.
- Finally, the studies identifying a thriving artist's traits will help shape the final report. The report will be shared with the artists to help to build a profile backed up by data.

Chapter 3

Related Work

This section discusses the pros and cons of some of the existing solutions to the problem of requesting song of your choice in a social gathering:

1. **Festify** (Leo, 2014-2018) [1]
   Pros:
   - Web-App and browser-based
   - Ability to create and join party using ID
   - Everyone can vote on songs
   - Fallback Playlist is a plus
   - Admin can control the app
   - Landscape view available for casting on TV

   Cons:
   - No mobile app. Only Web-app available and could be a threat to security.
   - Requires Spotify Premium account for the host
   - Only supports Playback on Google Chrome and Firefox browsers
   - Not enough information about casting the application.
2. **JukeStar** (Jukestar, 2020) [2]
   Pros:
   - Has a mobile app and a web-app
   - Users can join the party and vote for songs, serves the purpose
   - Share party link through SMS, Facebook, and Twitter

   Cons:
   - Parties can be hosted by only using the mobile app.

[1] https://festify.rocks/
[2] https://jukestar.mobi/

- Has a token system to request songs which restricts guests and limits their requests
- Requires a guest app to download to join the party and to purchase the Tokens
- Web-App do not support buying tokens
- No option for password-less public parties

3. **OutLoud** (OutLoud, 2020) [3]
   Pros:
   - Application Available for iOS, Android and Web
   - Users can create party and join party
   - Users can add songs easily with no restrictions
   - Users can easily vote up or down

   Cons:
   - No cross-fade between songs
   - Artist search not available
   - You cannot use your own local music.
   - Many bugs related to location tracking and casting the application

The applications mentioned above provide a varying range of functionalities. "Festify" is web-based and does not have any Android or iOS applications. "Jukestar" has a token-based system and forces the users to buy tokens for requesting songs. The playback option is also not supported by all browsers. "OutLoud" meets most of the requirements but does not support the local songs library. It makes using these applications difficult. The proposed solution will try to provide an elegant way to address most of these issues.

Above mentioned applications do not have any fun factor or surprise element and the only purpose they serve is to prioritize the playing queue based on votes from users.

Proposed solution adds a surprise element of incentives/ rewards by offering a ***game mode*** feature. This feature is fun and keeps the users engaged. Users can compete and win a prize in return for sharing their music choice. The game mode is required to be set up by the host. The game mode activates using a button click to simplify implementation and consider the project timelines. For future implementations, the settings can include game-mode frequency, polling duration, choosing song repository, setting up incentives, etc., to completely automate the game mode. The host will be able to control the rewards and their frequency. Game-mode provides the users with a song followed by its preview. The user can then vote for the song. The game-mode algorithm will decide the winner. The votes tell us about the popularity of the songs and can be used to generate song reports for the artist. The report captures information such as the number of times the songs appeared as an option, the number of times the song got played and the total number of likes and dislikes.

[3] https://outloud.dj/

Chapter 4

Aims, Objectives, and Deliverable

4.1 Project Aims

This application aims to create an ecosystem between a restaurant, its customers and artists, using which all the stakeholders are benefited.

- Artists may benefit by receiving feedback on their new songs.
- Restaurant owners may benefit by regulating the offers/ rewards/ coupons and controlling operating expenses.
- Customers receives offers and rewards by participating and voting for a particular song.

One aspect of this application lets the customers control the music they want to listen to by polling for the songs they like. The highest voted song gets played.

Another aspect is to leverage the customer polls and provide feedback for songs from new artists after listening to a 10 seconds preview. Overall, this activity can be classified as a surprise rewards-based gamified polling system. The rewards and offers are provided by the restaurant that helps the restaurant with its promotions and sales and helps regulate the operating cost involved with rewards.

4.2 Project Objectives

Below are the main objectives of this project for each stakeholder.

Host/ Restaurant can:

- Create event and add people to the event.
- Configure settings of the application.
- Start game mode.
- Control the frequency of game mode.
- Regulate the number of rewards and offers.

User/ Customer can:

- Join an event created by the host.
- Browse all the songs in the playlist.
- Place song request by voting for a song of his choice in the playlist.
- User can vote and compete for rewards in game mode.

Artist can:

- Upload songs to the application.
- View the application generated report in real-time.
- Share the report.

4.3 Project Deliverable

Below is the list of project deliverable:

- Dissertation Report
- Presentation Slides
- Maintenance Guide (Appendix D)
- User Guide (Appendix E)
- Source Code

Chapter 5

Project Requirements

5.1 Requirements Analysis

On carefully analysing the problem statement defined in section-2, the solution should have the capability to:

- let users pick their favourite songs to play in a restaurant setup
- help the restaurant (where the songs are being played) to regulate the rewards and offers and use them as a promotion strategy for the restaurant
- helps the budding artists by providing them song feedback from a live audience.

The proposed solution can be broadly divided into two parts, based on the problems that it is trying to address.

The first part of this application is called the "Normal Mode". The normal mode tries to solve a prevalent problem faced by almost every other person who visits a restaurant or some party or event where songs are being played. The problem is, *requesting the disc jockey (or the person controlling the music) to play your favourite song in a social gathering*. This problem statement itself highlights some of the major components required for the solution. The components include:

- **Host (restaurant staff):** represents the restaurant staff member who will be responsible for creating and conducting the event.
- **Event:** represents the premise where the application will be hosted. The application will provide the capability to the users to search and join the event.
- **Event Songs Upload Utility:** represents the utility provided by the application to upload songs by the Host of the event. This will create a repository of songs associated with that restaurant. The uploaded songs will be presented to the users as a songs playlist. The users will be voting on this playlist to place their requests.
- **Users (customers):** represents the people who are present in the location where this application is hosted.
- **Voting:** represents the mechanism to request for playing songs by the users.

Normal Mode

- The host creates an event and adds songs to the event.
- The users search and join the event.
- The users place their song request by voting for their favourite song.
- Highest voted song gets played.

From the normal mode, we have the functionality to upload music to the application, events and users, and a voting mechanism in place. The application can now be further enhanced by adding a new entity, the artists.

The second part of the application, called the "Game Mode", basically tries to link the restaurant, its customers and artists together and form an ecosystem where all the stakeholders are benefited. This is achieved by leveraging the voting activity of the users to benefit the restaurant and the artists. The Game Mode focuses on solving three main problem areas. The first problem area is related to the artists and getting feedback on their new unreleased songs from live-audience. The second problem area is to provide the users with offers in return for providing feedback. The third problem area is for the restaurant to enable them to regulate the rewards and offers. The additional components (apart from the Normal Mode) for this part of the application include:

- **Artists:** represents the new budding artists. They create original songs and need feedback on their song.
- **Artist Songs Upload Utility:** represents the utility provided by the application to upload the artist songs. The songs uploaded by the artists are accessible globally across all the events created by this application. The artist songs are stored separately, and it is different from the event songs.
- **Songs Feedback Mechanism:** represents the functionality using which the users can provide their feedback for the songs played in the game mode. The mechanism also includes the logic for selecting the winner of the game mode round.
- **Songs Report:** represents the report that is only accessible by the artist and provides the feedback statistics about the songs.

Game Mode

- The artists upload their songs into the application and get feedback from the users in a report.
- The users share their feedback, and the winner of the game mode receives offers from the restaurant.
- The restaurant can control the offers and rewards (by controlling the number of game mode rounds) and gets free promotions by using this application.

The normal mode is about providing the users with the freedom to request the songs they want to listen to. The game mode is about keeping the users engaged, get feedback on the artist's songs and help the restaurant with regulating their offers and rewards.

5.2 Requirements Specification

The overall requirements for this project can be categorised as functional, non-functional and the required resources for developing the application.

5.2.1 Functional Requirements

The functional requirements are defined for each of the stakeholders—the requirements code help in uniquely identifying the requirements. The code USR-REQ represents the user requirements, ART-REQ represents the artist requirements, RES-REQ represents the restaurant requirements, and APP-REQ represents the application requirements. To identify each requirement, a code **For Users**

The user should be able to:

- **USR-FREQ-1:** login the application using social media accounts.
- **USR-FREQ-2:** search an event by clicking on the search event button.
- **USR-FREQ-3:** join an event from the search result by clicking the join button.
- **USR-FREQ-4:** vote for their favourite songs by tapping on the vote button in the playlist.
- **USR-FREQ-5:** view the ongoing voting for playlist songs.
- **USR-FREQ-6:** vote in game mode and compete for rewards.

For Artist

The artist should be able to:

- **ART-FREQ-1:** login the application using social media accounts.
- **ART-FREQ-2:** upload songs by clicking on the upload button.
- **ART-FREQ-3:** view application generated report by clicking the view report button.
- **ART-FREQ-4:** share application generated report by clicking the share report button.

For Restaurant/Host

The host should be able to:

- **RES-FREQ-1:** login the application using social media accounts.
- **RES-FREQ-2:** create an event by clicking on the create event button.
- **RES-FREQ-3:** share the party link/ ID by clicking on the share event button.
- **RES-FREQ-4:** setup song repository by clicking the select song repository option.
- **RES-FREQ-5:** add/ remove users to the event.
- **RES-FREQ-6:** enable game mode by selecting the game mode option.
- **RES-FREQ-7:** setup Game Mode by clicking on the game mode settings button.
- **RES-FREQ-8:** setup the rewards as and when required by clicking on add offers/rewards button under game mode settings.

For Application

The application should be able to:

- **APP-FREQ-1:** play highest voted song
- **APP-FREQ-2:** resolve conflict in case of a tie
- **APP-FREQ-3:** display next songs in the queue
- **APP-FREQ-4:** play the game mode notification sound
- **APP-FREQ-5:** display songs with preview in game mode
- **APP-FREQ-6:** display the winner on screen

5.2.2 Non-Functional Requirements

- **NF-REQ-1:**This application is only for users who are in the same location as the host.
- **NF-REQ-2:**The songs list will be pulled from Firebase (Google, 2021b) database or Local Library.
- **NF-REQ-3:**To provide mobile (must) application and web (future) application.
- **NF-REQ-4:**To provide Android (must) and iOS (future) application.

5.2.3 Required Resources

Below is the list of resources required for the project:

- Flutter for building the User interface (Google, 2021c)
- Dart is the main coding language (Google, 2021a)
- Firebase for database (Google, 2021b)
- Cloud Storage for Music Repository (Storage, 2017)
- GitHub Repository setup (github, 2021)
- Visual Studio Code Editor (Microsoft, 2021)

Chapter 6

Project Risks

6.1 Technical Risks

The below list shows the possible technical challenges that could be faced during development.

- Since this project involves third-party API integration and pulling data from another repository, there may be some level of complexities involved, and the result may not be as expected. This could lead to slight modifications to the functional requirements.
- Accessing the local database to fetch songs list and make them available in the application could be another challenging task.
- For search functionality, the initial approach will be to implement the basic search as implementing a full-text search functionality could also be a challenging task.
- Implementing the game-mode algorithm to deduce the final report output and make it accessible to the owner of the song. Also, deciding on the level of details to cover in the report will take some time to get finalized.
- Google Firebase (Google, 2021b) integration to store songs which will be uploaded by the artists to be used in the game mode.

6.2 Non-Technical Risks

This section describes the non-technical risks that could be faced during development:

- Sickness: This could be a hurdle in meeting some of the intermediate task deadlines.
- Task prioritization: The initial plan has been build considering the deadlines and efforts required for the tasks. If required, more effort will be put to meet the deadlines.
- System crash: The code will be backed up on the GitHub (github, 2021) repository, and regular backups will be made to avoid this risk.

Chapter 7

Design

This chapter will elaborate on the design of this application in detail. It will cover the design goals that tell what this application wants to achieve. The system overview section discusses the strategy to achieve these goals. The following sub-section provides a deeper explanation of the system architecture implemented to build this application. Finally, the user interface section will show how the idea evolved during the storyboarding, low fidelity, and high fidelity design phases.

The literature review section looked into various studies that aim to provide users with more freedom to listen to the music of choice in a social gathering. Each of the studies has its core components which are orchestrated to deliver the desired solution. For example, Jukola (O'Hara et al., 2004), which is considered as one of the pioneering works in this field, has features like adding songs to a playlist, using a public display where the users can go and select songs to be added to the playlist. A hand-held device is used to capture the votes from the users in the restaurant setting. Songs are then played based on the votes, thereby making the whole experience more democratic. Another innovative solution to provide users with the freedom to choose the desired music was table-top design with levers to navigate a pointer towards a particular genre of songs. The journey from Jukola to Festify (Leo, 2014-2018) shows how these applications gradually evolved and how they addressed the problem in their unique ways.

These solutions were mainly focused on playing the songs in some order or based on the number of votes in a social setting. The analysis helped understand and identify the core components of these applications, which facilitated the users to play songs of their choice. The typical components across these applications include a host, a common or shared display screen, end-users and, a voting mechanism. The proposed application provides a solution to playing songs based on votes, but it also provides an ecosystem for the artists, restaurant, and its customers to work towards benefiting each other. The proposed application is named as ***Play Poll***.

7.1 Design Goals

This section explains the purpose of building the proposed application. Based on the literature review, the three required entities for building this application are the **artist** (mostly referred to fresh talent, or new budding artists), **restaurant** (also referred to as the host of the

application), and the **customers** (interchangeably called the end-users). The basic concept behind developing this application is to create an ecosystem by relating all the required entities so that they support and benefit from each other. The design of this application is explained more clearly by taking each of these entities into consideration.

For Restaurants

The restaurant is the central part of this application which provides the premise for executing and hosting this application where the customers can interact with the system. The main goal for the restaurant is to keep the audience involved by offering rewards and discounts during the game mode (explained later). The game mode is the key area of this application as it provides the restaurant to regulate the rewards, offers and discounts. It tries to solve a significant operational hurdle in the food industry by enabling the restaurant to control the rewards and offers fully. Based on the study performed by (Jang and Mattila, 2005), a restaurant needs to know what kind of rewards are suitable for their customers. Some customers prefer loyalty programs, whereas some are more inclined towards immediate rewards. This application provides the restaurant with a mechanism to choose the reward or offer they want to give to the customers. The study by (Jackson, Titz and Defranco, 2004) tries to explain the common forms of promotion strategies used by restaurants and also tells about setting up the right frequency of rewards. This application lets the restaurant decide the frequency of rewards they want to offer by initialising the game-mode desired number of times.

The design goals for restaurants are listed below:

- provide the restaurant with the choice of rewards that they want to offer
- give restaurants the capability to choose the right frequency of rewards considering their operational cost

Promotions are the positive side-effects of achieving these goals, which the restaurant can get for free by hosting this application.

For Users

The users are the driving force for this application as the whole process works based on their input. This application allows users to place their song requests by voting for their favourite songs. The playlist gets prioritised based on the number of votes received from the users.

The design goals for users are to:

- provide the users with the capability to place requests for their favourite songs using this application
- to compete for rewards in the game-mode initiated by the restaurant

This keeps the users motivated and engaged throughout the activity, and it is expected to help the restaurant with their promotional activities, retaining their loyal customers or even adding a new customer base. Since the rewards are ad hoc and there is no loyalty program involved, this seems fair for all users.

For Artists

The artists are directly linked with the game mode. The artists can upload their new songs into the application using the upload utility for artists. The restaurant conducts a round of the game mode, and the songs appearing in the options for game mode are chosen from the original work uploaded by the new artists. The customers see the options on their screen immediately, followed by a 10 seconds preview of the song. Customers listen to the preview, and a message to vote appears on the screen. The customer then provides their feedback using the voting buttons.

The design goals for an artist are to:

- create an upload utility in the application, where a new artist can upload their original work into the system
- to receive the feedback on their original work provided by the customers

There could be an option to share the report directly with the record companies or their A&R (artists and repertoire) representatives.

For Application

This application is designed to provide an elegant solution that effectively utilises the customer's votes to provide feedback to the artists. This whole activity is made even more interesting by gamifying the voting mechanism and adding surprise rewards.

The design goals for the application include:

- the simplicity in user interface and the ease of navigation between pages
- the application should be able to decide which song to play, in case of a tie between the number of votes

7.2 System Overview

This section explains how the proposed application will achieve the design goals listed in the previous section. The design goals suggest that the application will have two different modes of operation as illustrated in figures 7.1 and 7.2 (the icons used in the image are from the Icon Set provided by (Microsoft Corporation, n.d.))

Below are the details of the two modes - normal mode and the game mode. The images below show the leading entities of the application and their interaction with each other. The entities are colour-coded, with each colour representing an individual entity. The amber colours represent the application, the blue colour represents the restaurant, the maroon colour represents the customers, and the green colour represents the artists. The outgoing arrows signify what that particular entity provides. For a more precise understanding, a colour legend is also provided in the figures.

7.2.1 Normal Mode

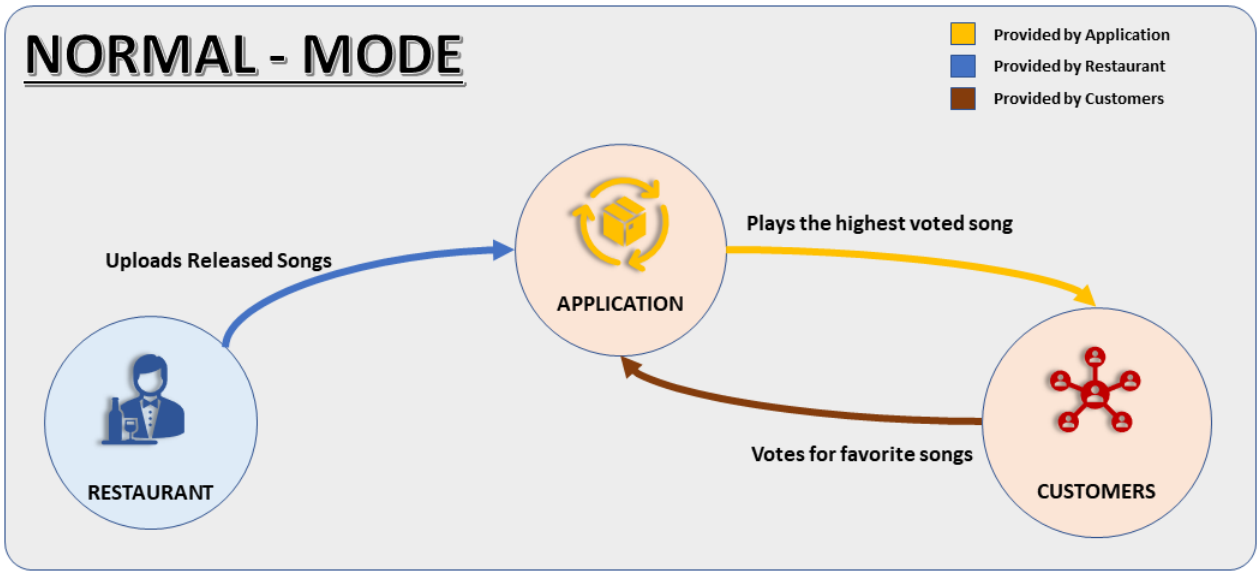


Figure 7.1: The normal mode diagram

The first mode is the ***Normal-Mode*** which involves only the restaurant and the customers. The restaurant is responsible for creating an event in the application to help customers identify the restaurant by providing the name and location of the restaurant. Once the event is created, the restaurant can now use the upload utility (Event Songs) and upload songs to the application, creating a song repository for the restaurant. The customer will use the search functionality of the application to find the restaurant's event. Once the event is displayed in the search results, the customer can click on the join button to join the event. On joining the event, the customer gets navigated to the event page. The event page comprises an audio player and a songs playlist. The audio player works in read-only mode for the customers and is mainly controlled by the restaurant. The customers can interact with the playlist. The customer can then scroll through the songs playlist and explore the available songs in the song repository provided by the restaurant. The customers have the freedom to place a request for the songs they want to listen to by voting for their favourite songs. This playlist will get prioritized based on the number of votes received from the customers. The song with the highest number of votes gets played. This is one aspect of the application that helps users place their songs requests, and this functionality is similar to applications discussed in the related work section.

7.2.2 Game Mode

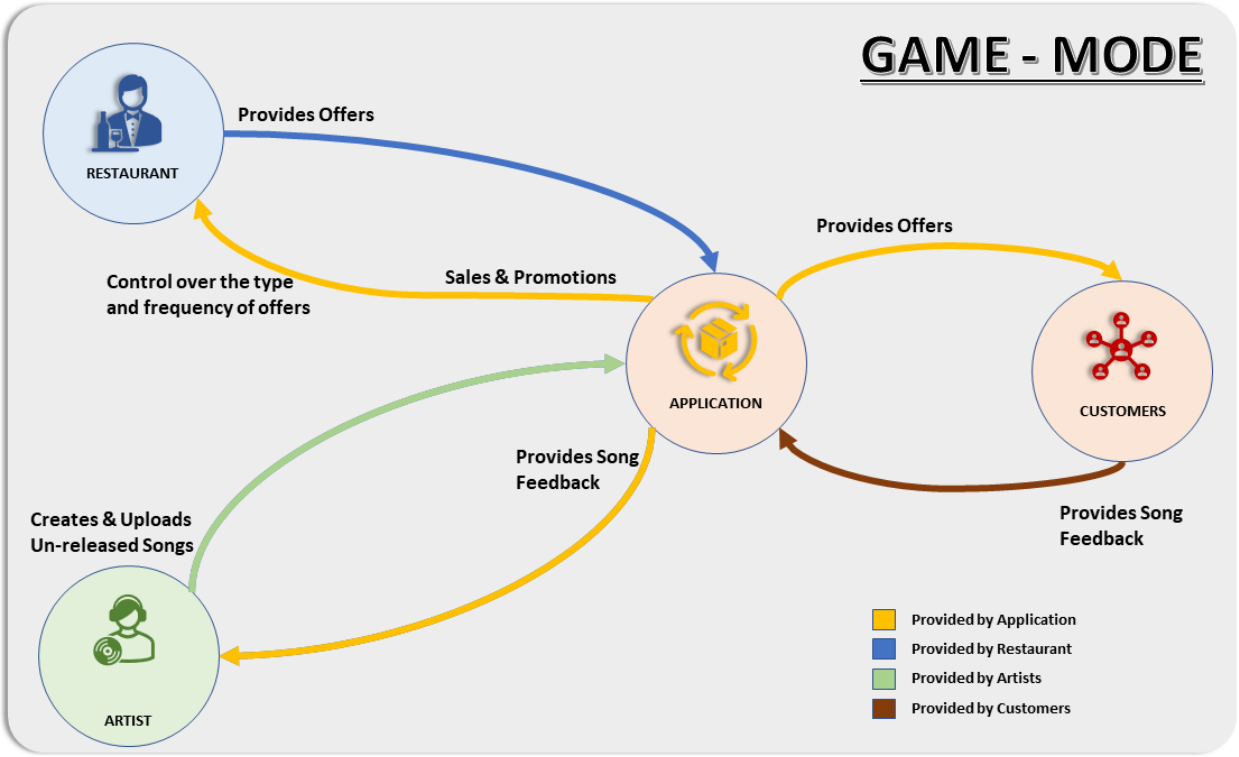


Figure 7.2: The game mode diagram

The second mode is the ***Game-Mode*** which involves the restaurant, the customers, and the artists. The artists can log in to the application and use the artist song upload utility to upload their songs to the central songs repository. The songs uploaded by the artist are accessible to all the restaurants, and this is different from the event songs which the restaurant uploads. Like the normal mode, the restaurant will create an event, and the customers will search and join the event. Once the event is started, the songs get played based on the number of votes. Now the restaurant can decide when to start the game mode, which gives the restaurant complete control of giving rewards and offers to the customers. The restaurant can click the 'BEGIN GAME-MODE' button, and the game mode will start with a 5 seconds drum-roll, and a new window will appear with a song displayed as an option. After the drum-roll, a 10 seconds preview of the song is played. Once the preview is over, the screen displays a message to vote, and the customers can choose the like or dislike button to vote for the song. After the voting window is closed, the application displays the votes count for both the voting button and the winner on the screen. Now the restaurant can choose to give the desired rewards and offers to the winner. The restaurant can now return to the event page by clicking the 'END GAME-MODE' button. The votes provided by the customers in the game mode create the artist report. The artist report is viewed on the artist homepage by clicking the view report button. On clicking the button, the artist navigates to the View Report page, where all the songs for the logged-in artist get listed as individual cards. The report shows how many times a particular song appeared as an option in the game mode and how many times it got played. The report also shows how many customers liked the song and how many customers did not

like it.

7.3 System Architecture

7.3.1 Domain-Driven Design

Software development project needs architecture to keep the code organized, maintainable, and easy to test. This project uses dart as the programming language for both front-end and back-end, so there is a high possibility that the addition of new functionalities will clutter the code and make it unreadable. In other words, follow an architecture to segregate the code to make it more readable, easy to understand, and easy to make changes. Thus, for any software development project, it is vital to follow a specific architecture. There are several architectures available to choose from, but, from prior experience, it becomes an easy decision to select Domain-Driven Design as the architecture for developing this mobile application. This section will discuss the various architectural layers of Domain-Driven Design, as shown in the figure 7.3.

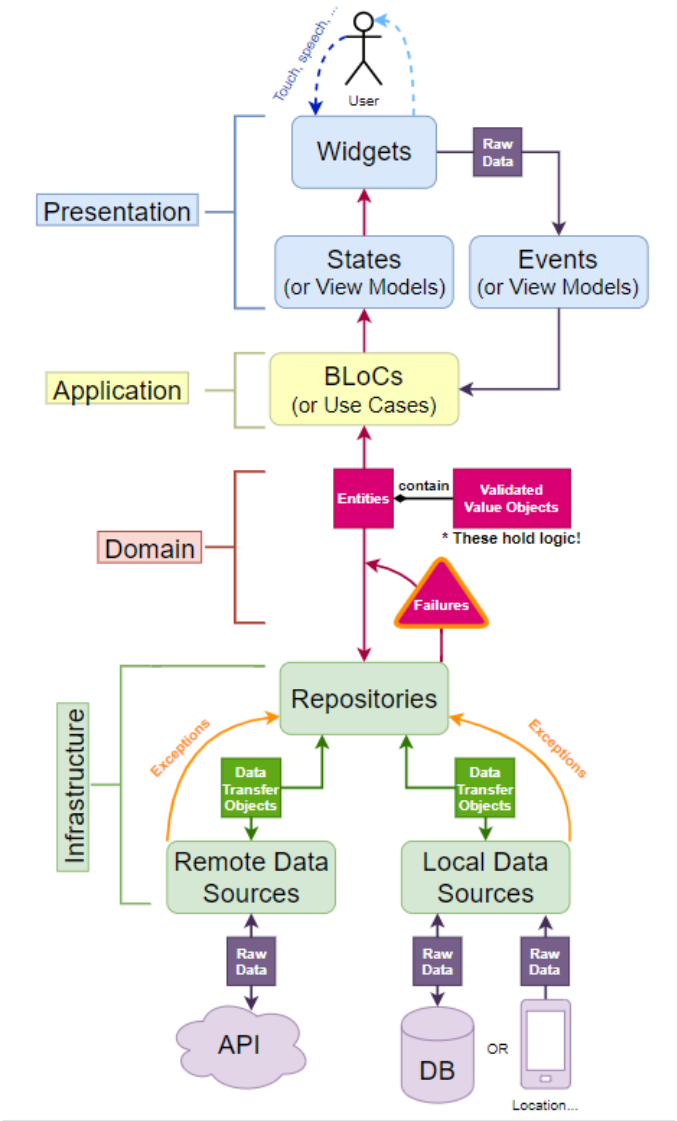


Figure 7.3: Domain Driven Design (Resetar, 2019)

The presentation, application, domain, and the infrastructure layers together form the architectural layers of Domain-Driven Design, as shown in the figure 7.3 above.

Presentation Layer

As the name suggests, this layer holds all the pages in the mobile application and their states. In flutter, anything visible to the user on the application screen is a widgets. Widgets are the basic building blocks of pages. A user can interact with the presentation layer by either touching the screen, typing, or speech. These interactions are called the events. Events can also carry some data along with it, such as some text entered into a field, or it could be as simple as a check-box, a button click. These events are then input to the BLoC (Business Logic Components) (Angelov, 2018) and it yields a state. The updated state is then presented to the user as a result of the earlier interaction (event triggered by the user). The presentation layer consists of only events and states, the BLoC is not a part of this layer.

Application Layer

This layer is where the BLoC resides. The user input coming from the events enters the BLoC and this is the place where the developer can decide to use the user provided data for further calculations, validations, or calling database to fetch more data based on the user input. This layer is responsible for interacting with other layers such as the domain and the infrastructure layers. Below figure 7.4 explains the working of a BLoC.

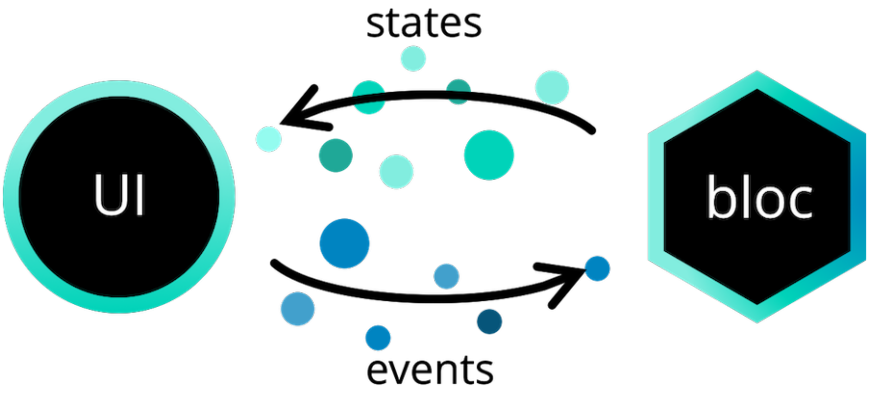


Figure 7.4: BLoC - Business Logic Components (Angelov, 2018)

A BLoC has the responsibility of mapping events with the corresponding states. It basically provides the new values of fields on the page based upon the action performed by the user. BLoC has a classic example of a counter application, where the user can click on the increment or decrement counter button to change the value of a counter that starts with initial value of zero. When the user clicks on the increment button, this event is sent to the BLoC and the bloc decides what state to yield and display it in the user interface.

Domain Layer

As the architecture name suggests, the domain layer is the main layer responsible for handling all the business logic. This layer is independent from all the other layers. Making changes to any other layer in the architecture doesn't affect the domain layer. It means that the implementation of the presentation layer or the infrastructure layer can be changed at any time without making any changes to the domain layer, and the application will work exactly the same. As a common practice, most of the validation is performed in the user interface itself. But in the case of domain-driven design, the user interface is only used for reflecting the state changes in the form of widgets, and no validation in performed in the user interface. This layers deals with all the entities involved in the application and performs the validation on these entities. The user input gets validated in this layer. Apart from validations, this layer is also used to handle exceptions, perform data transformations, and perform the business logic.

Infrastructure Layer

This layer consists of the repositories and the data sources. The infrastructure layer of domain-driven design is replaceable at any time without affecting the functionalities of the application. If the initial database was Firestore (Google, 2021b), then it can be easily replaced by any other database of choice. The same is true for the presentation layer as well, and it is also replaceable. Another thing to note in this layer is the use of data transformation objects

(DTOs). The purpose of the DTOs is to convert the data from the domain layer into the data types accepted and stored in the database. And in the same way, it is responsible for converting data from the database to domain objects. This application uses Cloud Firestore as the database and Cloud Storage for storing files.

7.3.2 Project Folder Structure

This section will discuss the folder structure followed for this project, in accordance with the Domain-Driven Design. Below figure shows the folder structure followed in the project:

Figure 7.5: Project Folder Structure

The folder structure in this project is aligned with the implemented architecture. The "lib" folder consists of four sub-folders and each folder represents a layer of the architecture.

The presentation folder contains all the user interface related files. This is the place from where the user is able to trigger events and view the updated state in return. This folder further contains sub-folders representing the various features of the application. Each sub-folder contains the pages and extracted widgets. For example, the "create_event" folder has the "create_event_page.dart" file. This is the page using which events are created in this application. Presentation folder contains all the pages that are visible to the user in this application, except for the routes folder. The routes folder is used for defining the initial page to start the application. The other pages are also listed in the "router.dart" file. This application uses the "auto_route: 2.2.0" package for defining the navigation between the application pages.

The application folder consists of the BLoCs created for each feature of this application. For example, the "create_event" folder represents the BLoC for creating a new event. The BLoC

folder consists of three files. The "create_event.dart" file defines all the possible events on the Create Event page. The "create_state.dart" file defines all the possibles states corresponding to the events. The "create_bloc.dart" file has the mapping defined between all the events and the states. BLoC is the place where the application decides what to do in response to an event.

The domain folder consists of all the entities required for a feature. Each folder represents a specific feature. This folder consists of entities, the validated value objects, the failure and the interface defined for the repositories. The Figure 7.5 shows the domain folder related to the events feature. The event entity is defined in "event.dart" file. The validated value objects related to an event (event name and event location) are defined in the "value_objects.dart" file. The validation failure are defined in the "event_failure.dart" file. The event repository interface is defined in the "i_event_repository.dart" file.

The infrastructure folder consists of the files that are responsible for interacting with the database and storage. This folder consists of the data transfer objects (DTOs), which are responsible for the data transformation to store the data in the database in the data types permitted by the database. Also, the DTOs are responsible for reading the raw data from the database and convert them to be used in the application. This folder has the concrete implementation of the interfaces defined in the domain folder. The "event_repository.dart" file is the concrete implementation for the methods defined in "i_event_repository.dart" file in the domain folder. This files contains all the APIs defined which are required to perform the create, read, update, and delete (CRUD) operations on the database.

7.3.3 Architecture In Action

This section will explain the system architecture as implemented in the development of the application. Below Figure 7.6 shows the system architecture in action:

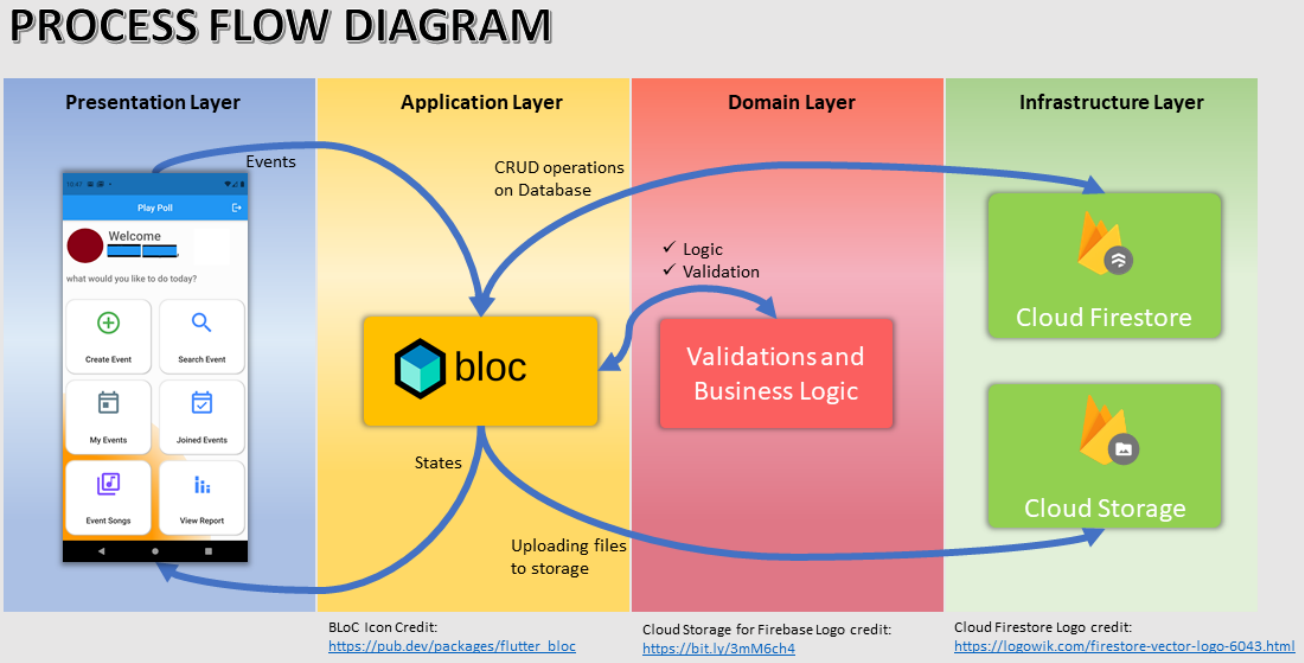


Figure 7.6: Architecture in action - Process flow diagram

The figure consists of four parts, each representing layers of the Domain-Driven Design architecture. It shows how a user will interact with the application user interface and the sequence of actions in the application. Creating an event is an example of interaction with the application.

Example: Creating an Event

For creating an event, restaurant staff can log in to the application. On the home page of the application, there is an option to create an event. Once the staff member selects that option, it generates an event. This event then goes into the BLoC, and the logged-in user sees the corresponding state. In this case, the yielded state will be the event creation page. On the event creation page, the user enters the name and location of the event, and the domain layer comes into action to validate the user input. After providing details, the user can select the create option. This action will again go into the BLoC and call the corresponding application programming interface (API). This API will create a new event in the "Cloud Firestore" database, as shown in the Figure 7.7. The new state yielded, in this case, will be a snack-bar message, telling the user that the event is available under "My Events" followed by navigating back to the homepage.

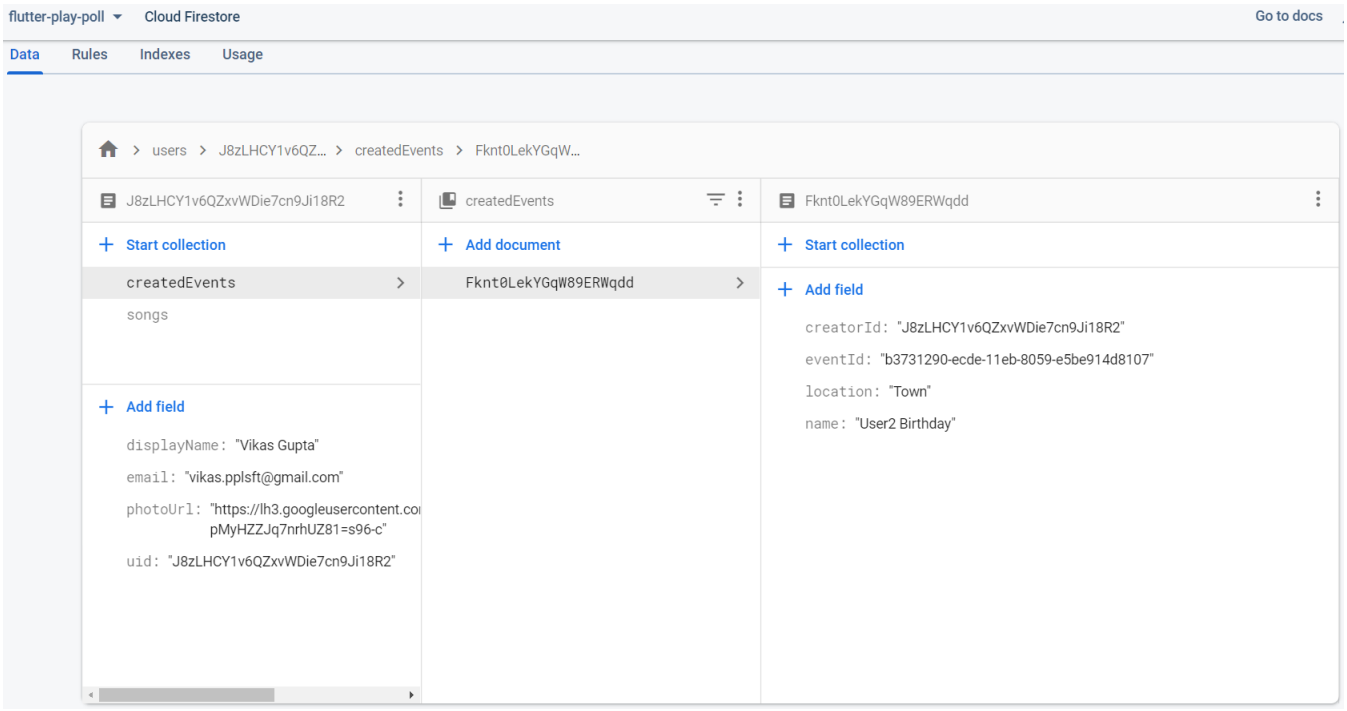


Figure 7.7: Event created in Cloud Firestore database

Example: Uploading Event Songs

Similarly, for uploading event songs, the logged-in user can select the "Event Songs" option on the homepage. This click event is sent to the BLoC and the user is presented with a new state, that is the upload event songs page. There are three options present on the upload song page. First option is to select the song to be uploaded. Selecting second option, uploads the song to the "Cloud Storage". On selecting the second option, this click event goes into the

BLoC. The BLoC then calls the API to upload the song to the cloud storage. A new folder hierarchy is created in the cloud storage with the root folder name as the user-id of the logged in user, and a sub-folder "songs". The uploaded song is stored in this folder, as shown in the Figure 7.8. The songs uploaded in this folder are only associated with the event created by the user. The user can see the upload progress on the page, after the upload button is clicked. The third option is to return to the home page, after the song is successfully uploaded.

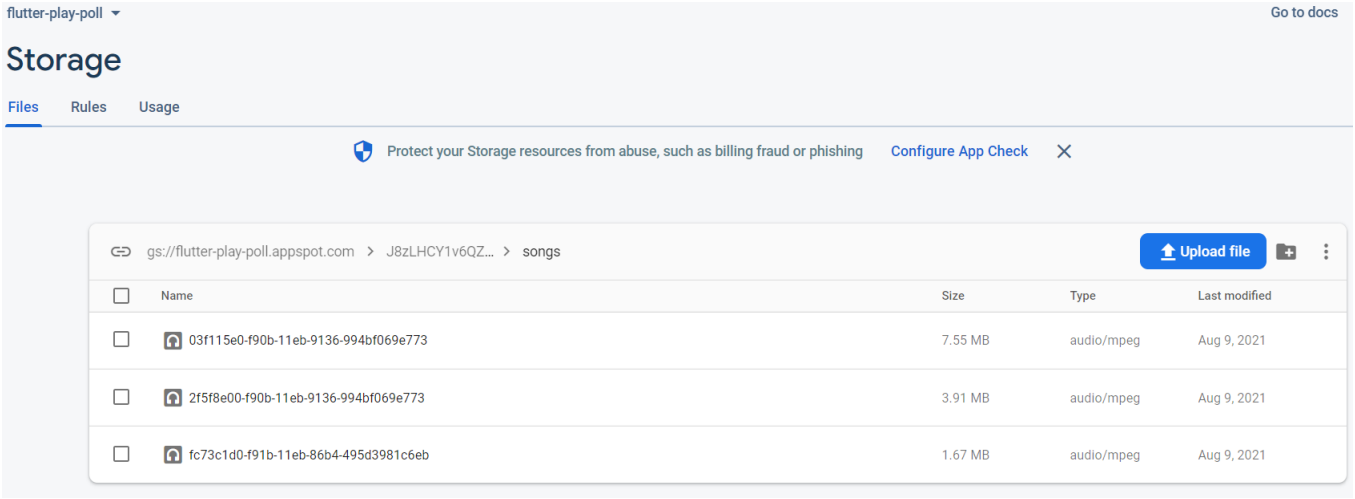


Figure 7.8: Cloud Storage Folder Structure

The two examples of creating an event and uploading songs to a created event gives a brief idea about the implementation of the architecture. More details will be discussed in the implementation section.

7.4 User Interface

This section will discuss the various pages required for each of the features of this application. It will also go through the various design phases of the user interface and produce the final design of the pages to be used in this application.

7.4.1 Low Fidelity Designs

Low fidelity designs (LFD) were created during the initial phase of this project. It only captured the information about the main features of this application. For example, a basic design for the login page, the home page, the search page, the create event page, the main event page, and the game mode page were created with the help of (Microsoft Corporation, n.d.). Not much time was devoted in this activity, but it was required to start the designing process of pages with some basic idea about the project. This activity provided an overview of the application. The missing functionalities will be explored and added in each iteration of this design.

Below Figure 7.9 shows the low-fidelity user interface designs:

Figure 7.9: Low Fidelity Page Designs

7.4.2 Medium Fidelity Designs

Medium fidelity design (MFD) takes a closer look into the requirements. This iteration also work towards story-boarding the requirements by deciding on the navigation flow and adds more details to the application design. Below Figure 7.10 shows the result of iteration over the low fidelity design:

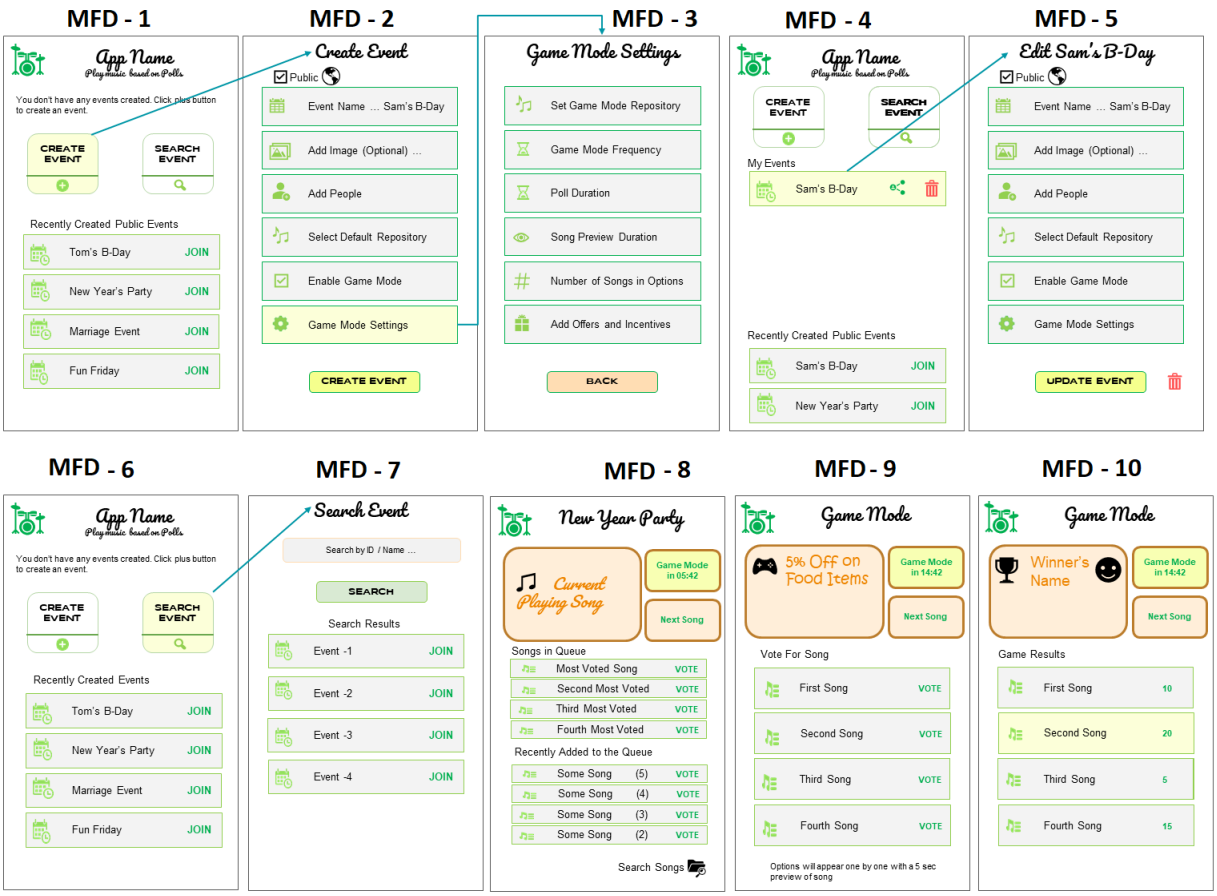


Figure 7.10: Medium Fidelity Page Designs

Home Page

This design is also made using (Microsoft Corporation, n.d.). User can achieve the functionalities by following the arrows in the above figure. The design covers the core functionalities of this application. Images MFD-1 and MFD-6 in the Figure 7.10 shows the home page that will appear after logging into the application. This page show the buttons to create an event and search for an event.

Creating an event

Image MFD-2 and MFD-3 shows the flow for creating an event from the home page. The user will click on the "Create Event" button and navigate to the "Create Event" page. The events can be made public or private. Private events will not appear in search results and users can be added to a private event only by the host. For creating an event, the user will provide the details of the event, add people to the event and select the repository from which the songs will be played.

Enabling and Configuring Game Mode

Image MFD-2 also displays an option to enable the game mode. If the user enables game mode then a new option "Game Mode Settings" should appear on the page. On clicking this new option, the user will get navigated to the "Game Mode Settings" page. The user will

be able to select a different repository for game mode (intended for new unreleased songs), then the user will define the frequency of game mode, which means after how many songs the game mode should appear and also poll duration can be defined. The song preview setting will let the host decide for how many seconds the songs appearing in the options should be previewed, for example, 5 seconds or 10 seconds, so that the users can listen to the preview and then vote for their favorite song among the given options. The next setting will let the host decide on the number of songs to display as options. In the last setting, the host will define the offers or incentives that the winner will get. The person who voted first for the most voted song is the winner.

Updating Event

After the event gets created, it will then display as a list item on the home page, as shown in image MFD-4. There will be options to share the URL of the event or delete that event. On clicking the list item, the user will navigate to the "Edit Event" page (MFD-5), where the host can make changes to the event. Once the host has made the changes, clicking on the "Update Event" button will save the changes and take the host back to the main page

Searching Event

Image MFD-7 shows the steps for searching and joining an event. The user will click on the "Search Event" button on the home page and get navigated to the "Search Event" page. The user searches for an event by entering the event ID or event name and clicking the search button. Search results will appear for the user to select the desired event to join.

Event Page

Images MFD-8,9 and 10, shows the screen that will be visible to the users joining an event. The top left section shows the currently playing song and on the right of it is an indicator that tells when the next game mode will come. The next item in the queue gets displayed below the game mode counter. The following section shows the list of songs in the queue with the highest voted song on the top and other songs list below in descending order of votes.The subsequent section shows the songs recently added to the playlist (the most recent one on the top). The last option on the page is to search for songs in the repository to add to the playlist. The user will get navigated to a new page to search for the desired song and add it to the current playlist.

Game Mode

Once the game mode is active, the current playing song section will display the offer for which the users are playing. This offer could also be a mystery box that will get revealed with the results, thereby adding a surprise element to the game mode. Songs in the queue section will be replaced by the voting section, displaying the songs on which the voting is to be performed. Songs will get added to the voting section followed by its preview (a 5-10 seconds preview defined by the host in "Game Mode Settings"). Song preview helps the user to decide their vote.

After all the choices are available to the users, they will get some indication to start voting. The poll duration will come from the "Game Mode Settings" as defined by the host. When the poll duration is over, the result will get displayed on the screen. The winning song will

display in the result section. The winner's name will display on the top left section and flash for 5-10 seconds. On completion of game mode, users will navigate to the previous page where the playlist is getting played for the event.

Game Mode Algorithm

The basic idea behind implementing the game mode is to get feedback from the live audience about the new-released or unreleased songs and generate a report to forecast the likeness of a song. The number of songs to present as the voting option is controlled by the admin. Song preview duration can also be set up as defined by the administrator.

For example, if the admin has set up the number of songs to be displayed in the game mode as 4, then the system will:

- Randomly selects 4 songs from the game mode repository to begin with.
- Once the audience has voted for their favorite song among the provided options, the vote count for each song in the first round will be recorded and will be later used in generating the report.
- These 4 songs will be removed from the subsequent list of options to be provided for the next round until all the songs in the repository have appeared at least once in the options.
- After each song has received some votes, the algorithm will try to form a group of 4 songs that have nearly the same votes and use these newly formed groups as options for voting in subsequent rounds of game mode.
- This process of forming a group of 4 songs will continue till all the songs have received different non-zero votes or till the event is closed for the day (whichever occurs later).

7.4.3 High Fidelity Designs

Below Figure 7.11 is the final high-fidelity design (HFD) for the application. Although the core functionalities and the design goals remain the same, there are many changes made to the medium-fidelity design along with some new pages introduced in the design.

For example, the medium-fidelity design does not discuss any page related to uploading the songs to the application either by the restaurant or artists. There is no page designed for the artist to login into the application and view the song's report. There is functionality to search and join an event but there is no page defined to view all the joined events. This section will discuss all the new additions and changes made to the existing design and explain the reasons for making those changes.

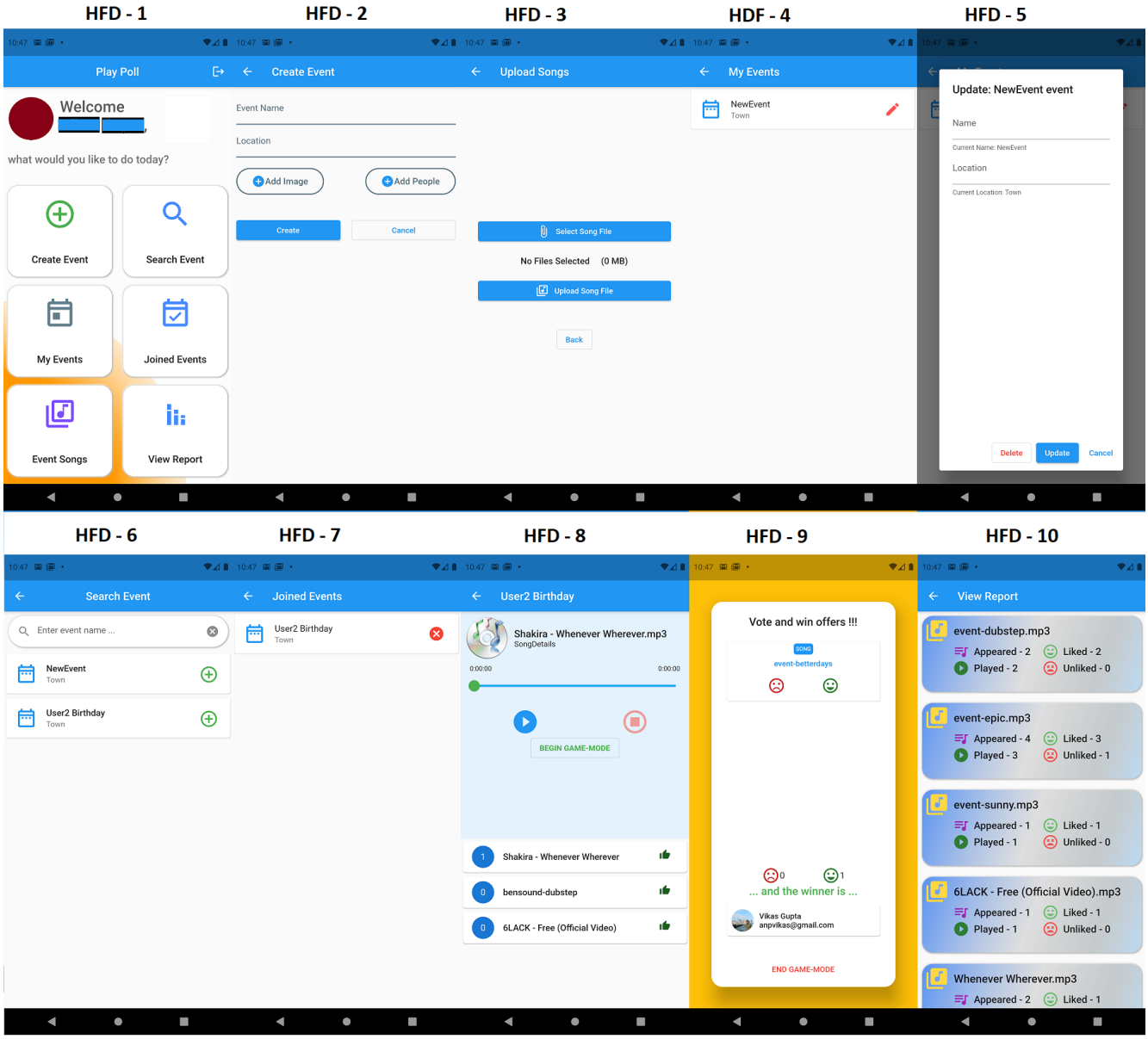


Figure 7.11: High Fidelity Page Designs

Home Page

The home page went under major changes, as shown in the Image HFD-1 of Figure 7.11. The final home page now has upfront access to all the functionalities of the application as compared to the previous design. It makes the home page look more consistent and easy to understand. The new options on the home page include "My Events", "Joined Events", "Event Songs", "View Report".

Creating an event

Image HFD-2 shows the Create Event page. The previous design had a number of fields for creating an event, making it a cumbersome activity to perform. The new design makes the event name and event location as required fields for creating an event. Features to add image

and people to the event are optional and all the other fields have been removed. The previous design was also missing a cancel button to cancel the event creation process. A snack-bar message gets displayed on successful creation of an event, and the new event can be accessed under the My Events option (as shown in HDF-4) .

Enabling and Configuring Game Mode

This option to enable and configure the game mode is removed in the new high-fidelity design. A new "BEGIN GAME-MODE" button is added to the Event Page below the audio player that manually controls the game mode. Following this design save the application from unnecessary complexity and extra coding and testing efforts. Providing this functionality gives more control to the restaurant to enable or disable the game mode manually as and when required. Providing an option to control the game mode (i.e., frequency to repeat the game mode after a certain number of songs) implies a need for another field to keep track of completed game mode rounds.

Updating Event

Users can navigate to the My Events page and click on the edit button to update an event, as shown in the image HFD-4 and HFD-5. Previously this option will appear on the home page after at least one event is created. For a first-time user, this information is not available upfront. In the new design, users can navigate to the "My Events" page to view/ update all the created events.

Searching Event

There are no major changes to this functionality. Users can search for an event by entering the full name of the event in the search box, as shown in image HFD-6.

Event Page

The Event Page is also updated to include a "BEGIN GAME MODE" button below the audio player, as shown in the image HFD-8. The game mode counter is no longer needed as the restaurant staff can control the game mode using the "BEGIN GAME MODE" button. As the playlist sorts in descending order based on the number of votes, there is no need to display the next song. Users can vote and view the vote count in real-time. The search songs functionality is also not part of this design, as the complete playlist is available on the Event Page, and the user can scroll through the playlist.

Game Mode - Revised

Restaurant staff initiates game mode by clicking the "BEGIN GAME MODE" button. Instead of replacing the components of the event page, a new modal window appears for the game mode, as shown in image HFD-9. The game mode starts with a 5 seconds drum-roll, followed by the 10 seconds preview of the artist's new song. The game mode now has only one option to display, as listening to 4 song previews of 10 seconds each may discourage the users from participating in game mode. Also, the voting option now includes a like and a dislike button represented by happy and sad icons. After the preview is complete, a message appears on the screen telling the users that the voting duration is 10 seconds. On completion of the voting duration, the winner and the count of votes gets displayed.

Game Mode Algorithm - Revised

The design goals for the game mode remain the same as before. The main change is the number of songs displayed as option is now reduced to one song. This design decision makes the implementation of game mode easy. The updated game mode algorithm is as follows:

- Randomly select one song from the artists repository.
- The users will vote for the songs in terms of like or dislike.
- On completion of voting duration, if the number of votes for like is greater than or equal to the number of votes for dislike, then the song gets played, else the song is not played.
- A winner is selected in both the cases whether the number of likes is more or less than the dislikes. This decision is taken because the user has shared his opinion and is eligible for rewards in return.
- The winner is a randomly selected user from either the like or dislike group, whichever has more votes.
- In case of equal number of votes for like and dislike, the winner will be selected from the users who voted for like icon.

The winner is decided randomly and not based on the speed of voting because if the users feel that they are late in casting their vote then they may get demotivated to share their vote as the chances of winning will be less.

My Events Page

My Events page is the new addition in the high-fidelity design, as shown in the image HFD-4. This page lists all the events created by the restaurant staff. This is the place to update the event details.

Joined Events Page

Joined Events page is another new addition, as shown in the image HFD-7. This page lists all the joined events by the users. This is the place from where the user can click on the cross icon to un-join/ leave the event. On clicking the list item, the user gets navigated to that event.

Event Songs Page

Event songs page is for the use of restaurant staff to upload the songs to be used in the Normal Mode of the application, as shown in the image HFD-3. These songs are the already released songs and the users place their request on these songs by voting in the normal mode.

Upload Artist Songs

Upload artist songs page is for the artists to upload their original songs into the application (screenshot in appendix). These songs are used in the Game Mode. The users vote on these songs to compete for the rewards.

View Report Page

View Report page is for the artists to view the real-time vote data on their songs, as shown in the image HFD-10. The artist is displayed with a page consisting of a list view displaying a card for each of the artist's song. This report captures the number of times a song appeared in option, the number of times a song got played, the number of likes, and the number of dislikes of a song.

7.4.4 Application Pages - Final List

Based on the requirements specification, below is the list of pages required to achieve the various functionalities of this application:

1. **Splash:** this is the decision page to navigate the logged-in user to the home page, otherwise to the login page.
2. **Login:** this is the page to login the application using the email id or social media account. This page has two text fields to enter username and password and a button to login with the email id. There is another button to login with google credentials. Priority is to implement the google login first and if time permits, the email id login will also be implemented.
3. **Home:** this is the landing page after login. This page has buttons or tiles corresponding to all the features provided by the application. The buttons/tiles are Create Event, Search Event, My Events, Joined Events, Event Songs, View Report, Upload Artist Songs.
4. **Create Event:** this page will have the form to create the event. User can provide the event related details such as enter name, enter location, select event photo, date, time, add people and create the event. There is a cancel button to navigate back to the home page.
5. **My Events:** This page shows a list view of all the events created by the user. This is the page from where the event can be updated. The individual list items will display a clickable edit icon to open the "Update Event" modal window. All the event detail fields can be updated on this page. There are additional buttons at the bottom of the modal page to update, delete or cancel the update for the selected event.
6. **Event Songs:** This page is used by the restaurant/host to upload songs to the application. These are the songs which are associated with the event created by the host. Customers will be voting on these songs in the Normal Mode of this application. There is a button on this page to select the song to upload and then click the upload button to start the upload process. The third button is to go back to the home page.
7. **Search Events:** This page is used to search for a particular event and the join the event. This page has a text field to enter the search text, a button to perform the search and a list view to display the results of the search. The list items have a clickable join icon. Clicking this icon will make the customers a member of that event. On clicking the join button, user directly navigates to the Event Page.
8. **Joined Events:** This page shows the list of events that the customer has joined. The list item has a clickable cross button to un-join/ leave the event.

9. **Event Page:** This is the main event page of the application. This page is used for the Normal Mode of this application and helps the users with placing their song requests. It has an audio player widget, a songs playlist with an icon button to vote for that song and the "Begin Game Mode" button. The list items are prefixed with a read-only vote counter. Clicking the begin game mode button will open another modal window and the game mode will start. The game mode will preview the song displayed as option. Voting window starts after the song preview is over. After the voting is over, and the winner gets displayed, the host can click on the "End Game Mode" button to return to the main event page.
10. **Upload Artist Songs:** This page has the similar functionality to that of the event songs page. Artists use this page to upload their songs into the application. These are the songs used as an option in the game ode on which the customers vote.
11. **View Report:** This is the report only visible to the artist. This is a read-only page and displays the statistics for each of the songs uploaded by the artist.

Chapter 8

Implementation

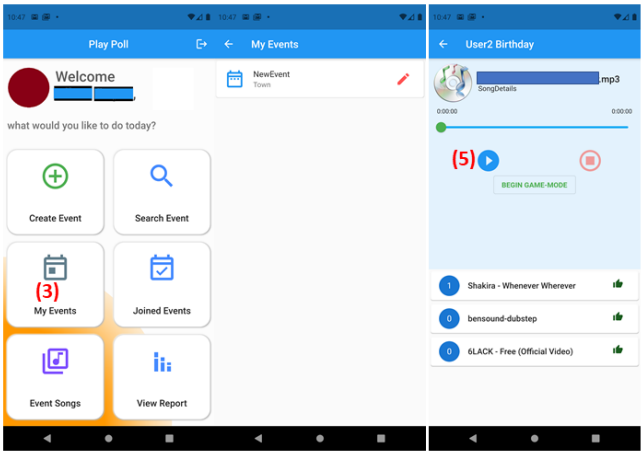
This section will discuss the implementation details of all the prominent features of this application. The approach taken to develop this application is to build the user interface first and then latch the API calls to the interface using the BLoC. The tools and software are discussed in detail in the appendix section B.1. Creating a user interface (UI) first provides a better picture of the events that a user can trigger to interact with the UI components. It also helps in identifying the corresponding states to be returned. After identifying the events and states, a BLoC is created to define the mapping between events and states. The BLoC is then used to make the API calls, perform the required back-end tasks, and return the new state. The complete coding implementation and database design is explained in the Maintenance Guide appendix chapter D. The source code is available in the GitHub repository.

8.1 Event Page Implementation

"Event Page" represents the "Normal Mode" of this application and helps users/ customers to place their song request (Image HFD-8, Figure 7.11).

Below Figure 8.1 explains how the Normal Mode is started by restaurant, how the customers join the event in Normal Mode, and the Game Mode details.

Event Page – (by Restaurant Staff)

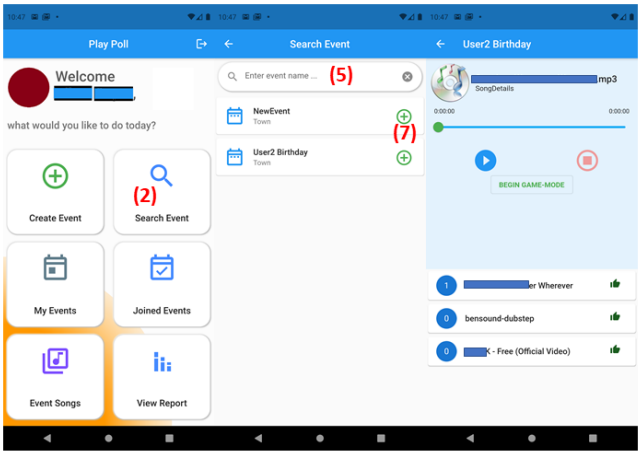


1. The created events are listed under My Events page
2. Restaurant Staff Logs into the application and lands on the home page
3. Clicks on the My Events button on the home page
4. Restaurant Staff is navigated to My Events page
5. Clicks on the Play button to start the Normal Mode

Normal Mode Started

Home Page My Events Page Event Page

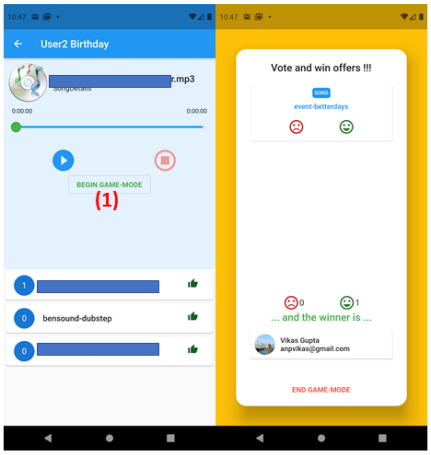
Search and Join Events – (by User/ Customer)



1. User/ Customer Logs into the application and lands on the home page
2. Clicks on the Search Event button on the home page
3. User/ Customer is navigated to Search Event page
4. By default, all events are listed on the search page
5. User/ Customer enters full name of an event and clicks enter
6. The search results shows that event as list item with a green Join button
7. Click on the Join button to join that event
8. User gets navigated to the Event Page

Home Page Search Event Page Event Page

Event Page – (by Restaurant Staff)



1. Restaurant Staff clicks on the BEGIN GAME MODE button to start the game mode
2. Game mode open in a new window with a drum roll for 5 seconds, and a song appearing in option to vote for.
3. Once the drum roll is over, the 10 seconds song preview is played.
4. After the song preview, a message appears to start voting.
5. On Voting the message disappears
6. After the voting duration is over, the winner is displayed on the screen
7. Restaurant Staff can announce the rewards and offers for the winner
8. Restaurant Staff can now click on the END GAME MODE button to return to the Event Page
9. Restaurant Staff can again start the Normal Mode

Game Mode Started

Event Page Game Mode Page

Figure 8.1: Event Page User Interface Interaction Explained

Presentation Layer

The Event Page is divided into two parts, the audio player and the songs playlist with voting button. The audio player part shows the current playing song with the play and stop buttons (to be used by the restaurant staff). The seek-bar and count-down times shows the song progress and duration. It also has the "BEGIN GAME MODE" button.

The second part of the page has the playlist of songs uploaded by the restaurant staff. These are the songs on which the users/ customers vote and place their song request.

Application Layer

The events for the "event_bloc.dart" BLoC are also divided into two parts to cater the requirements of the Normal and Game Mode. The Normal Mode events are: the default "started" event, "incrementVoteCount", "onPlayerCompletionEvent". The event state mapping for Normal Mode is shown below (Listing D.16):

Listing 8.1: Normal Mode Event and State mapping for BLoC : event_bloc

```
// To display the songs playlist
started: (e) async* {
    dynamic received = await
        _eventRepository.fetchCreatorSongs(e.data);
    yield EventState.showFetchedSongs(received);
  },

// To vote on songs in the playlist
incrementVoteCount: (e) async* {
    dynamic received = await
        _eventRepository.registerVote(e.songId, e.uid);
    yield EventState.incrementedVoteCount('$received');
  },

  // To reset vote after the song finishes
  onPlayerCompletionEvent: (e) async* {
    dynamic received =
        await _eventRepository.resetVoteToZero(e.songId, e.uid);
    yield SongsPlayerState.onPlayerCompletionState();
  },
```

The Game Mode events are: "fetchArtistSongsEvent", "createGameModeEntryEvent", "votingStartedEvent", "gameModeVoteEvent", "showWinnerEvent". Below is code (Listing D.17) for event state mapping:

Listing 8.2: Event and State mapping for BLoCs : event_bloc and song_player_bloc

```
  // To fetch the artist songs
  fetchArtistSongsEvent: (e) async* {
  dynamic received = await
      _iStorageRepository.fetchArtistSongs();
  yield SongsPlayerState.fetchArtistSongsState(received);
},
```

```
// To initiate the game mode in database
createGameModeEntryEvent: (e) async* {
        await _eventRepository.createGameModeEntry(
            e.eventId, e.songId, e.artistUid);
        yield EventState.createGameModeEntryState();
      },

// To display game mode started message
votingStartedEvent: (e) async* {
        int value = 10;
        int output = 0;
        Timer votingTimer = Timer.periodic(Duration(seconds:
           1), (votingTimer) {
          value = value - 1;
          output = value;
          if (value == 0) {
            votingTimer.cancel();
          }
        });
        yield
            EventState.votingStartedState(votingTimer.tick.toString());
      },

// To vote in Game Mode
gameModeVoteEvent: (e) async* {
        await _eventRepository.registerGameModeVote(
            e.songId, e.artistUid, e.voteSmiley);
        yield EventState.gameModeVoteState();
      },

// To display winner of game mode
showWinnerEvent: (e) async* {
        dynamic selectedWinner =
            await
                _eventRepository.decideGameModeWinner(e.eventId,
                e.songId);
        yield EventState.showWinnerState(selectedWinner);
      },
```

Domain Layer

New methods are added to the existing interface "i_event_repository.dart" file.

Methods added for Normal Mode are: fetchCreatorSongs(), registerVote(), resetVoteToZero().

Methods added for Game Mode are: fetchArtistSongs(), createGameModeEntry(), registerGameModeVote(), decideGameModeWinner().

Infrastructure Layer

This section will show the main lines for implementing the features of Normal Mode and Game Mode.

Normal Mode:

- fetchCreatorSongs() method:
  The application uses this method to display all the event songs on the "Event Page". These are the songs uploaded by the restaurant staff. The implementation is a simple fetch query from the restaurant staff's "songs" collection, sorted by the number of votes in descending order.

  Below Figure 8.2 shows the architecture layer functioning in the back-end for loading the event page:

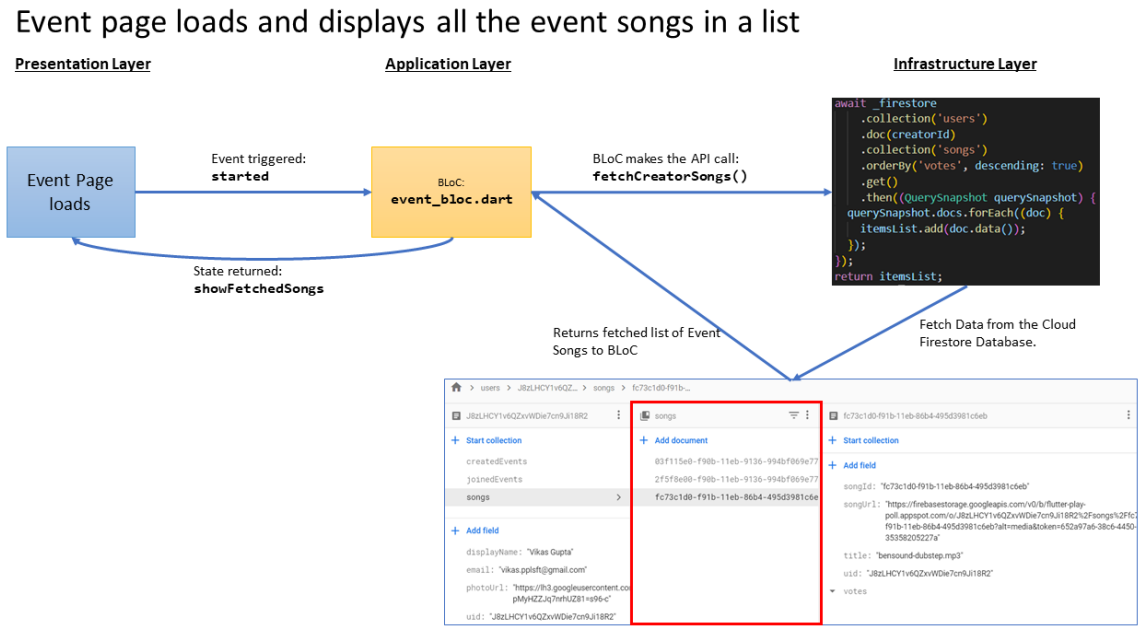


Figure 8.2: Event Page Loads

- registerVote() method:
  The application calls this method on clicking the "thumbs up" icon to register the users/customers votes for the songs.

  Below Figure 8.3 shows the architecture layer functioning in the back-end for capturing votes on the event page:

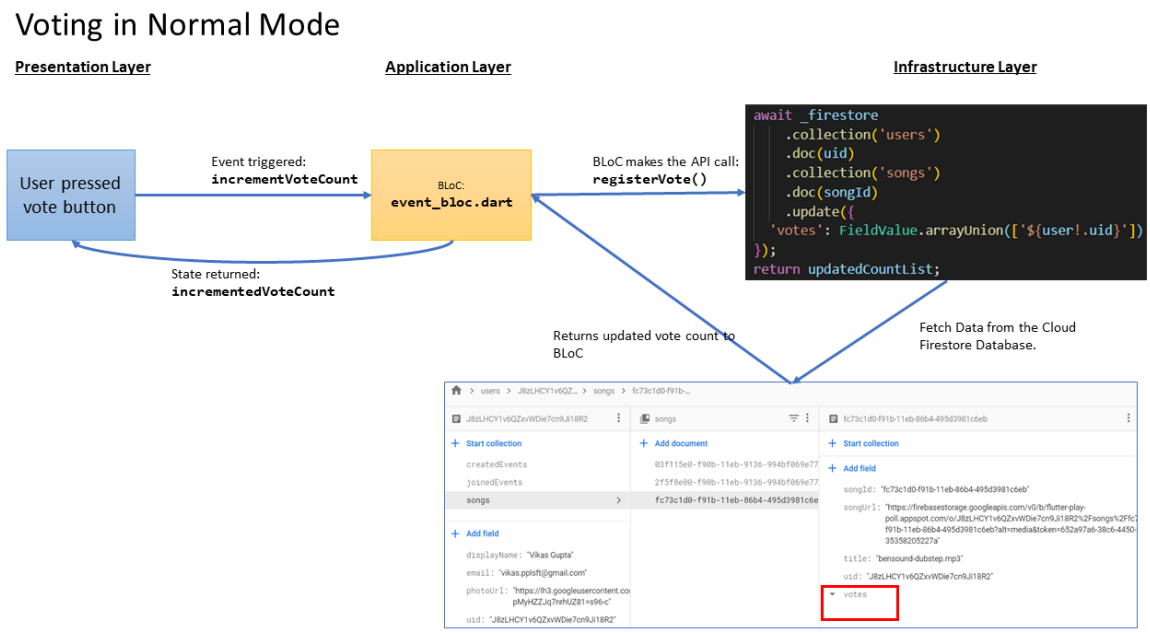


Figure 8.3: Voting in Normal Mode

- resetVoteToZero() method:
  This method resets the vote count to zero after the song has finished playing.

Game Mode:

- fetchArtistSongs() method:
  This method gets called on click of the "BEGIN GAME MODE" button, and a 5 seconds drum-roll indicates the start of game mode. This method selects a random song from the artist's song repository. The selected song is displayed to the users/customers for voting after listening to the song preview.
- createGameModeEntry() method:
  This method creates a placeholder in the database for the game mode round.
- registerGameModeVote() method:

  Below Figure 8.4 shows the architecture layer functioning in the back-end for capturing votes on the game mode page:

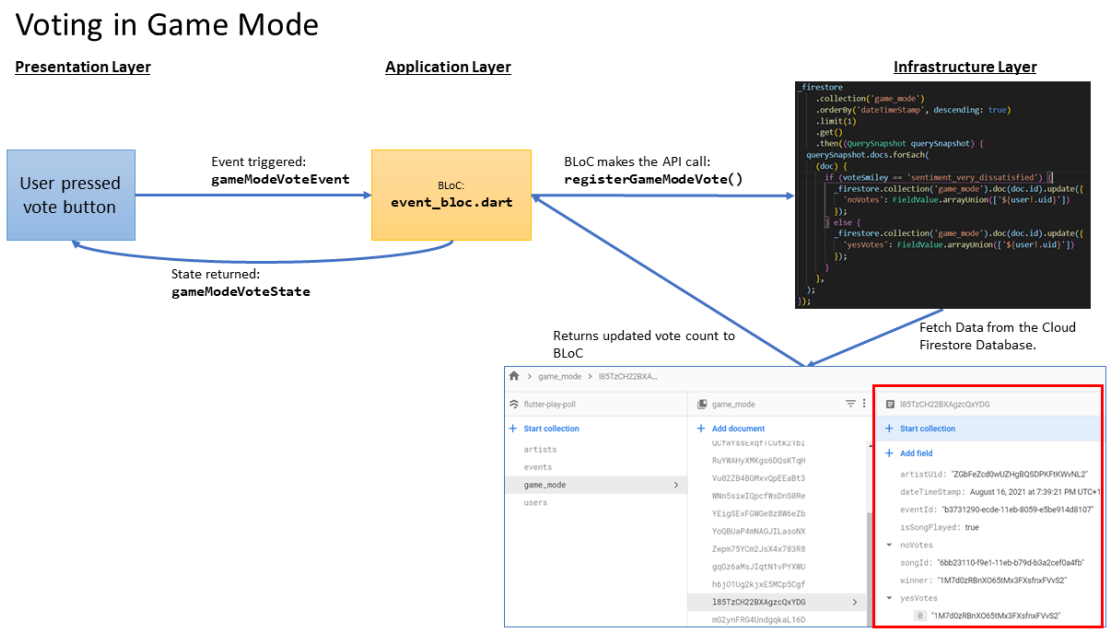


Figure 8.4: Voting in Game Mode

This method registers the votes of the users/customers in the game mode.

- decideGameModeWinner() method:

Below Figure 8.5 shows the architecture layer functioning in the back-end for deciding the winner in the game mode page:

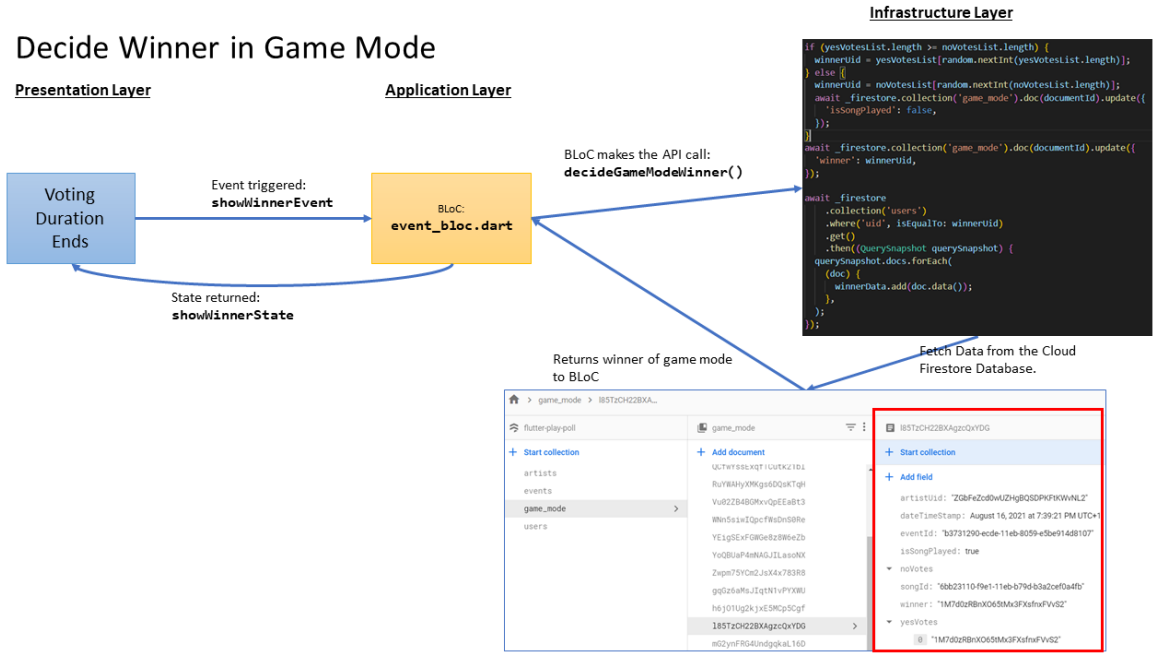


Figure 8.5: Deciding Winner in Game Mode

The application uses this method to select the winner of the game mode. This method gets called after the voting duration is over. The votes are categorised as "Liked" and "Disliked". Whichever category has more votes, the application selects a winner from that category. However, if the winning category is the "Liked" category, the previewed song gets played. Whether the song is liked or disliked, the application always announces a winner.

8.2 Unimplemented Project Requirements

Below is the list of requirements from chapter 5 - Project Requirements, which are future requirements.

For User

- The song search functionality implementation is future work. As an alternative solution, the event page lists all the songs in the song repository on the event page with a descending order of votes.

For Artist

- Sharing the artist report functionality is not implemented because of shortage of time, but from the artist's perspective, it is an essential feature to showcase the song's popularity to record companies. This feature will be implemented on priority to support the artist's work.

For Restaurant

- The functionality of sharing the event will make it easier for the users to join a particular event. As an alternate solution, the users can directly search for the event name on the search event page and join the event. This functionality has less priority as compared to the other restaurant-related functionalities, and it is future work.
- Instead of Setting up the song repository, there is an option to upload the songs to the application directly, and the application will pick the songs from the same list. Every event has its song folder in the database to which the restaurant can add the songs.
- Add/Remove users from the event will be implemented in the future, as this will not affect the functionality of the application from a restaurant point of view.
- Instead of giving a setting for the game mode, a separate button is provided on the event page to begin the game mode. The manual button will decrease the need to monitor the application by restaurant staff, as now the restaurant staff has a direct option to begin and end the game mode as and when required and reveal the offer to the winner.

Chapter 9

System Testing

9.1 Testing Strategy

This section will look into the system testing approach followed for the development of this application. The testing strategy includes:

- Module identification and scope
- Test Cases for the module
- Bug reporting

9.1.1 Module Identification and Scope

This application consists of the below mentioned main modules:

- Create Event:
  This module includes the create event feature. The applications lets the restaurant staff to create event by providing name and location of the event. The scope is limited to event creation.
- My Events - Update/ Delete Events:
  This module includes the "my events" feature, which displays the events created by the restaurant user. This module also includes the update and delete event features.
- Event Songs and Upload Artist Songs:
  These are basically two different modules, but have similar functionalities and can be grouped together. The only difference is how the uploaded file is stored. The scope of this module is to very the file gets uploaded.
- Search and Join Events:
  This modules includes the search functionality for searching events and joining events. The scope of this module is to search for an event and join that event.
- Joined Events:
  This module includes the list of joined events. The scope is to verify the list of joined events is populated and user is able to navigate to the event page on clicking the list items.

- Event Page:
  This is the main event module which has the normal mode and game mode of the application. The scope of this module is to check the functionalities of the normal mode and game mode, including song playlist loading, voting, playing song.
- View Report:
  This module includes the final report meant for the artist. The scope is to verify the report is getting generated and accessible to the artist.

9.1.2 Unit Test Cases for modules

Create Event

Title: Create Event
ID: UTC-CE
Precondition: User is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on Create Event button	User is navigated to Create Event page	same as expected	Pass
2	Enter name and location and click on create button	Success message appears and user is navigated back to home page	same as expected	Pass

Table 9.1: Test Cases for Create Event Module

My Events - Update / Delete Events

Title: My Events - Update / Delete Events
ID: UTC-ME
Precondition: User is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on My Events button	User is navigated to My Events page	same as expected	Pass
2	Click on edit icon button	Modal window appears	same as expected	Pass
3	Enter new event name and location and click update button	Modal window closes and user is on My Events page	same as expected	Pass
4	Verify the updated event name and location is displayed	Event name and location is updated	same as expected	Pass
5	Click on edit icon button	Modal window appears	same as expected	Pass
6	Click on delete button	Modal window closes	same as expected	Pass
7	Verify the event is deleted	Event is deleted	same as expected	Pass

Table 9.2: Test Cases for My Events - Update/ Delete Module

Event Songs and Upload Artist Songs

Title: Event Songs and Upload Artist Songs
ID: UTC-ES
Precondition: User is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on Event Songs button	User is navigated to Upload Songs page	same as expected	Pass
2	Click on Select Song File button	Permission window appears	same as expected	Pass
3	Click allow	Device file browser appears	same as expected	Pass
4	Select a mp3 song file	File gets selected with file name and size getting displayed	same as expected	Pass
5	Click on Upload Song File button	Upload progress starts and file gets uploaded	same as expected	Pass
6	Click on Back button	User navigates to home page	same as expected	Pass

Table 9.3: Test Cases for Event Songs and Upload Artist Songs Module

Search and Join Events

Title: Search and Join Events
ID: UTC-SJ
Precondition: User is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on Search Event button	User is navigated to Search Event page and list of all events gets displayed	same as expected	Pass
2	Enter full name of an event	Search result shows the typed event	same as expected	Pass
3	Click on add button to join event	User navigates to the event page	same as expected	Pass

Table 9.4: Test Cases for Search and Join Events Module

Joined Events

Title: Joined Events
ID: UTC-JE
Precondition: User is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on Joined Events button	User is navigated to Joined Events page and list of joined events gets displayed	same as expected	Pass
2	Click on any list item	User is navigated to event page	same as expected	Pass
3	Click back button to return to Joined Events page	User is navigated to Joined Events page	same as expected	Pass
4	Click on red cross icon button to un-join/ leave event	List item gets deleted and user has un-joined/ left the event	same as expected	Pass

Table 9.5: Test Cases for Joined Events Module

Event Page

Title: Event Page
ID: UTC-EP-RST
Precondition: Restaurant Staff is on the home page.

For restaurant staff

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on My Events button	User is navigated to the selected Event Page and an audio player and songs playlist is visible with a thumbs up icon to vote	same as expected	Pass
2	Click on the audio player Play button to start the event	Normal mode gets started and songs start playing	same as expected	Pass
3	Click on the BEGIN GAME MODE button	Game mode gets started with a drum-roll and the game mode window appears	same as expected	Pass
4	After the winner of game mode is displayed, Click on the END GAME MODE button	The game mode window pops and the Event Page appears to start the Normal mode	same as expected	Pass

Table 9.6: Test Cases for Event Page Module for Restaurant Staff

Title: Event Page
ID: UTC-EP-USR
Precondition: Customer is on the Joined Events page.

For restaurant customers

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on list item on the Joined Events page	User is navigated to the selected Event Page and an audio player and songs playlist is visible with a thumbs up icon to vote	same as expected	Pass
2	Click on the thumbs up icon to vote for a song	vote count increases by one and the song moves up in the list	same as expected	Pass
3	Game mode song option gets previewed for 10 seconds	user is able to see the game mode window and listen to the song preview	same as expected	Pass
4	After the preview is over, message appears on the screen to vote in 10 seconds	User is able to see the voting message	same as expected	Pass
5	User clicks the like or dislike button to vote	The voting message disappears	same as expected	Pass
6	Voting duration is over and winner gets displayed on the screen	User is able to see the winner of the game mode round	same as expected	Pass

Table 9.7: Test Cases for Event Page Module for Customers

View Report

Title: View Report
ID: UTC-VR
Precondition: Artist is logged in and on the home page.

#	Steps	Expected Result	Actual Result	Pass/ Fail
1	Click on View Report button	User is navigated to View Report page	same as expected	Pass
2	Artist is able to see the report for all the songs	Verified	same as expected	Pass

Table 9.8: Test Cases for View Report Module

9.1.3 Bug Report

BUG-1 : In the normal mode, the voting button is not disabled after voting.

BUG-2 : In the game mode, the voting button is not disabled after voting and the user can press both the voting options for a song.

BUG-3 : Join event functionality is broken.

BUG-4 : On create event page, clicking on the create button without entering any values for name or location throws error.

BUG-5 : Both the name and location values must be provided to update an event's name and location, even if the requirement is to only update the name.

BUG-6 : If no songs are added in an event and a user joins that event, a circular progress indicator shows indefinitely.

Chapter 10

Survey

10.1 Survey Setup

This section shows the various questions asked in the survey and defines the survey's components, such as the rating scale and the intent of asking the questions.

10.1.1 Rating Scale and Application Screenshots

The survey scale is from 1 to 5, with one being minor and 5 being the maximum value of the user response. Apart from the rating questions, there is one paragraph question to get comments and suggestions from the users.

Relevant screenshots of the application are to conduct the survey. The screenshot captures the application home page, creating, updating, searching and joining the event. Another screenshot shows the 'Event page' and the Game-Mode page. The event page takes the customers' song requests to play the highest voted song next. The Game-Mode page displays the songs from new artists, and the customers are required to vote to compete for rewards and offers from the restaurant.

10.1.2 Survey Components

The survey design covers all the application features and gets the opinion from the users. Below are the main categories of the survey:

Application User Interface

The survey includes a general question about the user interface of the application, asking how easy it is to follow the instructions on the various pages of the application on a scale of 1 to 5. A related paragraph question asks the survey takers to provide any suggestions related to the user interface. The feedback from this section will help improve the user interface of the application and know which part of the user interface was difficult or easy to understand for the end-users.

Feature Related Questions

There are, in total, four questions related to the features of the application. Two questions rate on a scale of 1-5, and the remaining two are multiple-choice questions. The rated questions explicitly rate the two most prominent features of this application, i.e. the ability to poll for the next song to be played at the restaurant and the game-mode feature to rate the songs from new artists.

The multiple-choice questions ask about the likes and dislikes of the users on the various aspects/features of this application, such as getting rewards in return for sharing the taste of music, helping new artists with feedback and the ability to manage offers/rewards by the restaurants. The multiple-choice questions have a text field to provide comments about their likes and dislikes of any feature of this application apart from the listed options.

The intent of asking these questions is to gauge the popularity and likes/dislikes of the features offered by this application. The feedback will help decide which feature needs improvement, which feature is most popular, and which was not appreciated by the users.

Customer Related Questions

There are four questions to know how likely the users are to install the application, use its features, their interest in getting rewards, and how this application motivates the users to visit a restaurant frequently. The response to these questions will help analyse the success of this application and know if this application will be of any benefit for the restaurants in terms of increasing their customer base and getting promotions and sales.

Artist Related Questions

There are two general questions about artists as not every user of this application will be an artist. The first question asks a general opinion about how this application will be helpful for the new artists to get feedback on their work. The second question is how likely an artist is willing to use this application to get feedback. The reason for asking these questions is to know how well artists will receive this concept.

Restaurant Related Questions

There are three questions related to restaurants, and these questions ask about a general opinion from the survey takers. As the survey takers may or may not have in-depth knowledge about the restaurants, these questions ask them about their perception of the usage and benefits of this application for a restaurant. The first question mainly focuses on assessing that how this application will be beneficial for restaurant's promotions. The second question talks about the feature of this application that lets the restaurant regulate their rewards, offers and discount and control their running cost. The final question asks the user's perception of how likely a restaurant will be willing to run such an application to help and support the local or new artists. These questions will help know the survey takers general perception about how willing a restaurant will be to run such an application on their premise.

10.2 Survey Result Analysis

This section looks deep into the survey results and provides an analysis of the survey.

Below Figure 10.1, shows the response for the user interface and artist related questions.

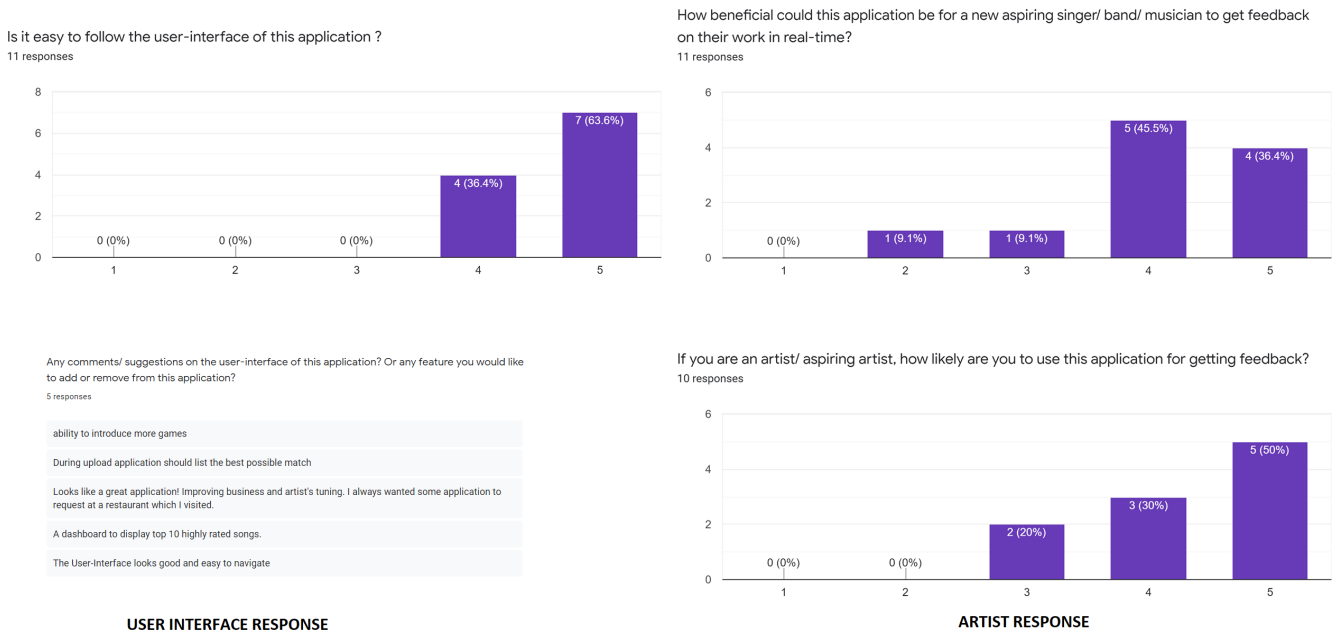


Figure 10.1: User Interface and Artist Response

User Interface Response:
All the responses for the user interface were above the average point of the scale (1-5). 63.6% of the responses find the user interface very easy to follow, and the remaining 36.4% responses find the user interface to be somewhat easy to follow.

There were many great responses related to improvement in the user interface and features of this application. There were suggestions to add more games in the game mode. Some responses appreciated the overall design of the application. One interesting response was to include a dashboard to display the top ten songs. Another response was related to the upload functionality of the application to suggest a possible match of the song already existing in the application to avoid duplicate songs.

Overall, the user interface was relatively easy to understand for most of the survey respondents, and it is a good sign that they suggest adding more games and features to the application.

Artist Response:
When asked about how beneficial this application will be for artists, 36.4% of the responses find the application to be very beneficial, whereas 45.5% of the responses think that the application is somewhat beneficial. On the other hand, 9.1% of respondents think that this application will not make any difference or benefit artists.

The next question asked if the artists were likely to use this application for getting feedback. 50% respondents think that the application is very likely to be used by artists, whereas 30% of responses think that this application is somewhat likely to be used. The remaining responses have a neutral opinion about the application used by artists.

Overall, the majority thinks that this application will somewhat benefit the artists, and majority of the respondents who are artists, are willing to use this application.

Below Figure 10.2, shows the response for the feature related questions.

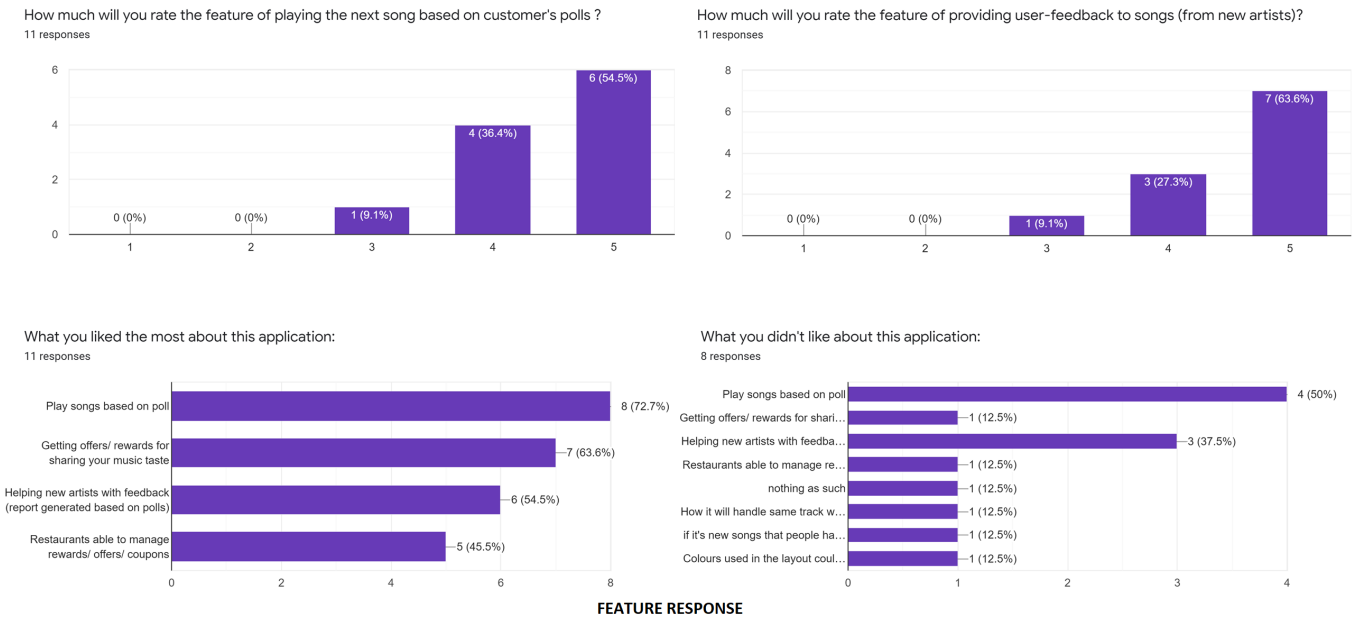


Figure 10.2: Feature Response

Feature Response:
When asked about the feature of playing the next song based on the poll, a majority (54.5%) of the respondents find it an up-and-coming feature and some (36.4%) find it somewhat promising 9.1% have a neutral opinion. When asked about providing user feedback to artists, 63.6% find this feature very promising, and 27.3% find it somewhat promising, whereas 9.1% have a neutral opinion. From the responses, it seems that providing artist feedback is more liked by the end-users when asked directly.

When asked to select what they liked the most about this application, the overall result (72.7%) favours the feature of playing songs based on the poll, followed by the feature of getting offers from the restaurant with 63.6% responses in favour. About 54.5% of responses liked the idea of helping the artists, whereas the least liked feature is of the restaurants able to manage their offers with 45.5% responses in favour. The feature of restaurant able to manage offers is least popular, maybe because not every respondent can think from a restaurant point of view.

When asked what they did not like about the application, about 50% of responses said the feature of playing songs based on the poll, followed by providing artist feedback with 37.5% responses. However, this question also had the option to provide other answers that might have distributed the responses, and it is not easy to deduce anything conclusive from the responses in this case. One response did not dislike anything about this application, whereas one response did not like the application's colour scheme. One response asked a question about the upload functionality to handle the same song getting uploaded multiple times, and one response was not aware of the song preview in game mode.

Below Figure 10.3, shows the response for the Customer and Restaurant related questions.

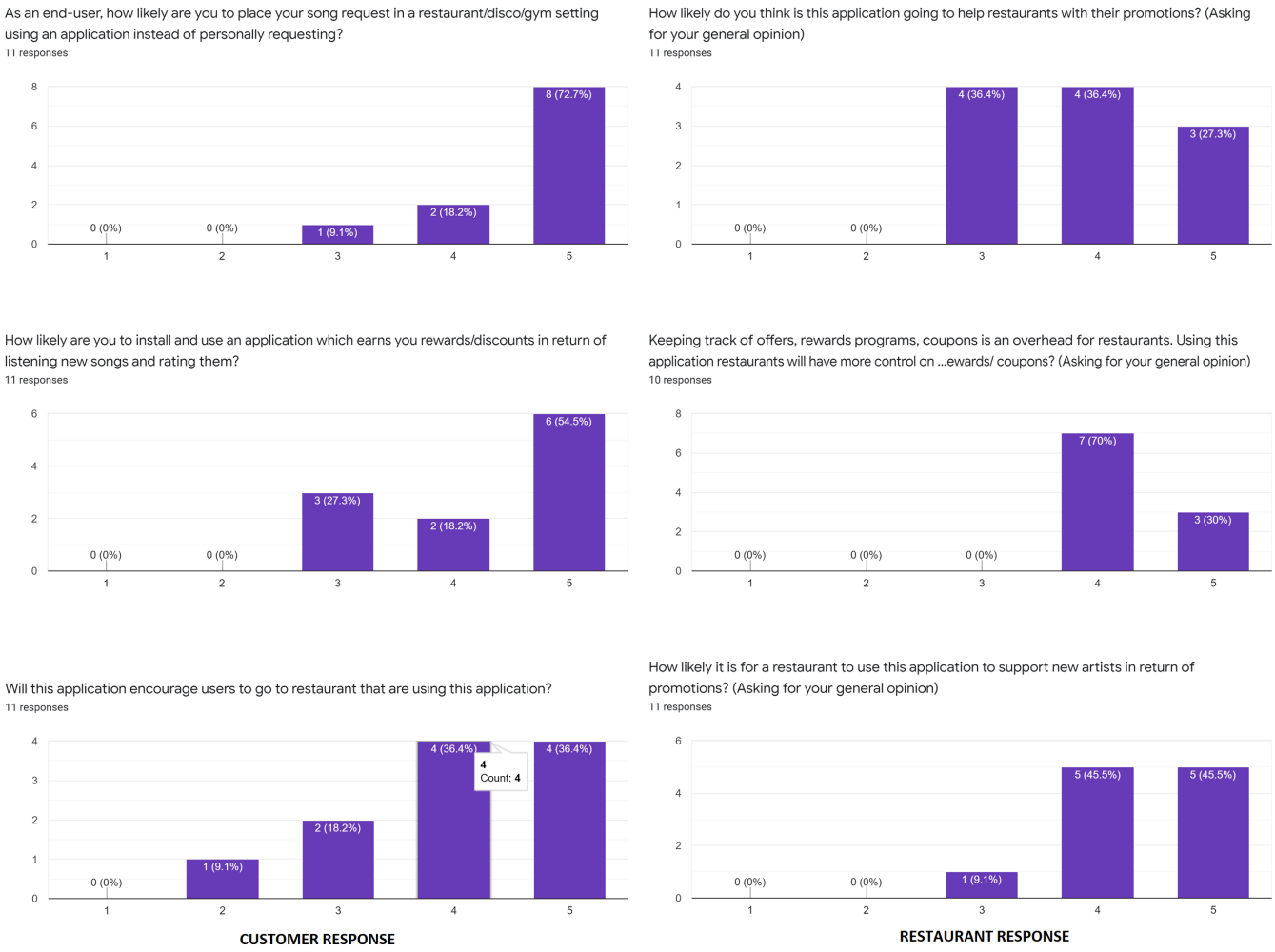


Figure 10.3: Customer and Restaurant Response

Customer Response:

The majority (72.7%) of survey respondents prefer using an application to place song requests rather than making personal requests. The only 18.2% of respondents feel somewhat using an application for placing a request, whereas 9.1% have neutral preferences.

When asked about installing and using an application that earns them rewards in return for providing feedback, the majority (54.5%) were willing to install the application, whereas 18.2% are somewhat willing to install it. About 27.3% of respondents have a neutral opinion about installing the application.

When asked about how encouraging it is to go to a restaurant using this application, about 36.4% of respondents are very much encouraged, whereas an equal number of respondents are somewhat encouraged to go to such restaurants. About 18.2% of respondents are not affected by the restaurant using this application, and 9.1% thinks this application is not encouraging enough to go to the restaurant.

Overall, it seems that most respondents prefer using an application to place their request and are willing to install an application that earns them rewards for rating songs. Also, it is overall

encouraging for the respondents to go to such restaurants which are using this application.

Restaurant Response:
The restaurant-related questions provide a general opinion as not everyone knows restaurant operations. However, 36.4% of respondents think that this application will be somewhat helpful for restaurants with their promotions. An equal number of respondents have neutral opinions, but 27.3% of respondents think this application will be handy for promotions.

A majority (70%) of respondents think that this application will be somewhat helpful for the restaurants in managing the offers and the remaining respondents think this application will help the restaurants.

About 45.5% of respondents think that restaurants will support the new artists by using this application and gain promotions in return, and an equal number of respondents think it is somewhat likely for the restaurant to help new artists and benefit from this application. About 9.1% of respondents have neutral opinions.

Overall, respondents find this application promising for a restaurant, and the majority of respondents think that the application will be helpful to regulate the rewards and offers of the restaurant.

Chapter 11

Limitations and Future Work

This chapter discusses the limitations of the application and the future work that will be incorporated in later releases to address the limitations. The points mentioned in this chapter includes the suggestions revealed by the survey analysis. The limitations and the future work are divided into categories corresponding to the various features of this application, as shown below.

11.1 Limitations

- ***Application Login:***

  The application works only for Android devices. As this application only supports google login, the users are required to have a google account. Although for the proof of concept even google login is sufficient to show the login functionality.

- ***Music Provider Integration:***

  For the purpose of this proof-of-concept, the songs are currently being pulled from Google Firestore Storage which serves as the song repository for the host and the artists.

- ***First-Time User:***

  There is not enough visual information available for the first time user, which could make it difficult for the user to understand the concept of this application.

- ***Playback and Connectivity:***

  The application is installed on the host's mobile device which is then connected to a sound system for amplifying the playing songs. The connected device is therefore required to be active at all times.

- ***User History:***

  There is no feature to keep history of the rewards won by the user.

- ***Song Preview Duration:***

  The songs preview is currently chosen by the host who creates the event and the same preview duration is applicable to all the songs uploaded by the artists. There is no facility

in the application which lets the artist define their best bit of the song to be used for preview.

11.2 Future Work

- ***Application Login:***

  The later releases of this application will include an iOS version and a web application. To reach a broader range of users, email login and other social media logins such as Facebook and Twitter will be supported in the future. The guest login feature will make more sense once the web application is released. The web application will attract users who do not want to install the application but want to participate in the event.

- ***Music Provider Integration:***

  Uploading songs manually to the application could discourage some of the users from using this application. It is required to have integration with music providers such as Spotify to tackle this apparent issue. After this integration, the hosts can point their Spotify playlist as the song repository instead of manually uploading them.

- ***First-Time User:***

  Any new application needs to make the users informed about the core concept of the application. There are several ways to address this requirement. The login page could have a carousel of few slides depicting the core idea of this application. An onboarding screen can display all the features and make the user aware of the available options. A video splash page to demonstrate the purpose of the application will be helpful. Also, a video could be played on the first-time login of the user.

- ***Playback and Connectivity:***

  A new feature will cast the application on other devices such as a television or projector screen and use their sound system. Also, background playing support is an essential feature for hosting a live event so that there is no discontinuity in playing songs.

- ***User History:***

  There is a good chance that the user who won the reward may not want to use it immediately and want to redeem it later. For example, if the user won a free dessert reward but did not feel like having dessert, the reward could be carried forward for their next visit. This suggests implementing a user wallet feature to store the incentives and uses the incentives later on some other day.

- ***Intellectual Property Agreement:***

  From an artist's perspective, this is the most crucial feature that can be offered in this application. The application should help protect the artist's work by assuring them that their content will not go out of this application.

- ***Song Preview:***

  This feature favours the artist in providing the best bit of their song for preview during the event instead of just picking the beginning few seconds of the song by default. The feature should allow the artist to select multiple song preview duration such as 10, 20 or

30 seconds, and the application should be able to pick the song preview provided by the artist.

- ***Restaurant Feedback, Promotions and Customer Rating:***

  The application must provide a feature to collect customer feedback to help the restaurant get feedback and increase their promotions. This feature will allow the customers to give ratings and comments about the restaurant, and also, the customers will be able to recommend the restaurant by creating social media posts and tagging their friends and family.

- ***Other Features (including survey results):***
  - For artists, if this application can connect to some of the record companies, then the A&R of the record companies can periodically check the artists' performance and may give a chance to the budding artists.
  - Providing ability to share group photos and collage and share it to other social media platforms such as Facebook.
  - Setup Advertisements for the restaurant's food menu or other offers or to promote some event on the application screen, which can be used to generate revenue from this application.
  - Adding more song related games such as - ordering some songs in the order of their likes and matching with the current song rating from popular websites such as billboard or Spotify.

Chapter 12

Conclusion

This project develops a software solution to integrate the restaurant, its customer's and budding artists and address some of their core problems. This section will discuss the changes the requirements went through, the impact of the design decisions taken, the assessment of the application based on the survey result and the plan for the second iteration of the application.

As the project progressed, there were many small and significant changes incorporated in the design phase. The design user interface section under the design phase shows how the design of this application evolved. The initial low fidelity design (Figure 7.9) only captured the broader picture of the application and helped decide the minimum required features. The medium-fidelity design phase (Figure 7.10) adds more details to the page designs, such as the event creation and the game mode settings. After carefully considering the complexities involved in implementing the solution and the project's timelines, the design was modified to keep the essential feature of the application intact and either removing or moving around the other features. The final high-fidelity design aims to make user interaction easier by eliminating the need to enter details manually. The significant changes were removing the game mode settings and also changing the game mode process and algorithm. The game mode is modified to display only one song as an option. This decision was taken so that the user may not feel overwhelmed with too many song options and lose interest in the game mode. Another reason was the complexity that arises because of too many options, as it will require keeping track of the number of game mode rounds and number of rewards distributed. The restaurant announces the rewards instead of tracking them in the application, as entering and tracking them will require constant monitoring by the restaurant staff. The home page in the high fidelity design looks much more straightforward, and the same reflects in the survey result analysis related to the user interface.

The timely design decisions made it possible to successfully achieve all the core functionalities of the application and complete one full iteration. The high fidelity design phase significantly helped in keeping the project on track. Choosing the exemplary architecture helped with the development process. The files were organised and easy to find, which saved a good amount of development time. Also, having a clear project structure makes it easy to maintain the codebase. Implementing the Domain-Driven Design made it possible to remove or replace any part of the application at any time. This architecture provides much flexibility in including new functionalities and scaling the overall project.

This project completed all the primary requirements, except for the ones mentioned in the

section 8.2. The search song functionality is crucial when there are many songs in the playlist, and the user has to perform the song search to find the desired song. As this implementation does not deal with many songs, this requirement becomes secondary to implementing the core functionalities. As the number of songs is relatively few, currently, the system displays all the event songs on the Events Page. For the artists, the requirement to share the report is not implemented, mainly considering the project's timelines; also, this feature requires a little more analysis to decide over the proper channels to share the reports to benefit the artists. For the restaurant, the functionality to share the event has less priority as the users can search the event using the Search Event page. Setting song repository is no more required as the uploaded songs are stored in the Cloud Storage under the restaurant's folder, and it is the only source of songs for the normal mode in restaurants. Add/ Remove users from the event by the restaurant was not implemented as this does not affect the core functionalities for a restaurant, although it will be implemented in the future. The game mode setting is replaced with the "BEGIN GAME MODE" button, which gives the restaurant more control over the number of rounds to conduct and saves the restaurant staff from entering more details and constant monitoring.

The project aims, as discussed in the section 4.1, are successfully met. The restaurant can control the rewards and offers by having control over the game mode. The restaurant can decide to provide rewards and offers as and when required by initiating the game mode. The customers get the rewards and offer in return for sharing their opinion about the artist's songs. The application selects the winner and displays it on the game mode screen, enriching the overall participation experience. The artists are receiving the song feedback, which will benefit them to polish their work further and released an improved version of the song.

The survey results analysis strengthens the belief that this application will prove to be beneficial. The survey respondents find the user interface very easy to follow and are looking forward to seeing more games and features in the future. For the artists, the majority think that the application is somewhat beneficial, and they are also willing to use it to get feedback. The survey results show that respondents favour both the normal and game mode of this application. The respondents also find it encouraging to visit a restaurant using this application. The majority of respondents think that this application will be pretty helpful for restaurants to regulate their rewards and offers. Overall, the survey results show a positive response to the application.

If given a chance to implement the application all over again, the choice for architecture will remain the same (Figure 7.3). The second iteration of the application will provide a chance to include the unimplemented requirements, followed by including the suggestions and features mentioned in the survey results and finally the future work mentioned in the section 11.2 will be incorporated into the application. Writing test cases using Mockito (dart.dev, 2021) framework will be considered for this iteration on priority. One of the features in the survey results asked to display the top ten highest rated songs, which will help the customers vote for these songs upfront to play in the normal mode. Another response in the survey asked to introduce more games. There were also suggestions to improve the colour choice used in the user interface. Incorporating all these changes will make the Play Poll application even more likely to be used

Bibliography

Alibaba, 1999. *Alibaba* [Online]. Available from: https://www.alibaba.com/ [Accessed 2021-08-25].

Angelov, F., 2018. *Flutter bloc* [Online]. Available from: https://bloclibrary.dev/#/ [Accessed 2021-08-25].

Cechanowicz, J., Gutwin, C., Brownell, B. and Goodfellow, L., 2013. Effects of gamification on participation and data quality in a real-world market research domain [Online]. *Proceedings of the first international conference on gameful design, research, and applications*. New York, NY, USA: Association for Computing Machinery, Gamification '13, p.58–65. Available from: https://doi.org/10.1145/2583008.2583016.

Chen, Z., Yavuz, E.A. and Karlsson, G., 2012. What a juke! a collaborative music sharing system [Online]. *2012 IEEE international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)*. IEEE. Available from: https://doi.org/10.1109/wowmom.2012.6263751.

Crowston, K., 2012. Amazon mechanical turk: A research tool for organizations and information systems scholars [Online]. *Shaping the future of ICT research. methods and approaches*. Springer Berlin Heidelberg, pp.210–221. Available from: https://doi.org/10.1007/978-3-642-35142-6_14.

dart.dev, 2021. *mockito: 5.0.15* [Online]. Available from: https://pub.dev/packages/mockito/versions [Accessed 2021-03-24].

eBay, 2006. *ebay motors* [Online]. Available from: https://www.ebay.co.uk/b/Motors/bn_7001210476 [Accessed 2021-08-25].

Firestore, C., 2017. *Cloud firestore* [Online]. Available from: https://firebase.google.com/docs/firestore [Accessed 2021-08-25].

github, 2021. *Where the world builds software* [Online]. Available from: https://github.com [Accessed 2021-03-24].

Google, 2021a. *Dart* [Online]. Available from: https://dart.dev [Accessed 2021-03-24].

Google, 2021b. *Firebase helps you build and run successful apps* [Online]. Available from: https://firebase.google.com/ [Accessed 2021-03-24].

Google, 2021c. *Flutter* [Online]. Available from: https://flutter.dev [Accessed 2021-03-24].

GROUPON, 2008. *Groupon* [Online]. Available from: https://www.groupon.com/ [Accessed 2021-08-25].

Hamari, J., Koivisto, J. and Sarsa, H., 2014. Does gamification work? – a literature review of empirical studies on gamification [Online]. *2014 47th hawaii international conference on system sciences*. IEEE. Available from: https://doi.org/10.1109/hicss.2014.377.

Jackson, F.H., Titz, K. and Defranco, A.L., 2004. Frequency of restaurant advertising and promotion strategies. *Journal of food products marketing* [Online], 10(2), pp.17–31. Available from: https://doi.org/10.1300/j038v10n02_02.

Jang, D. and Mattila, A.S., 2005. An examination of restaurant loyalty programs: what kinds of rewards do customers prefer? *International journal of contemporary hospitality management* [Online], 17(5), pp.402–408. Available from: https://doi.org/10.1108/09596110510604823.

Jukestar, 2020. *Fire the dj and start the social jukebox at your next party* [Online]. Available from: https://jukestar.mobi/ [Accessed 2021-03-24].

Koskela, T., Julkunen, J., Kassinen, O. and Ylianttila, M., 2010. User evaluation of a community-centric music voting service. *International journal of digital content technology and its applications* [Online], 4(1), pp.69–78. Available from: https://doi.org/10.4156/jdcta.vol4.issue1.7.

Koskela, T., Julkunen, J., Keränen, V., Kostamo, N. and Ylianttila, M., 2009. User experiences on a community-based music voting service [Online]. *2009 congress on services - i*. IEEE. Available from: https://doi.org/10.1109/services-i.2009.33.

Kukka, H., Patino, R. and Ojala, T., 2009. Ubirockmachine: A multimodal music voting service for shared urban spaces [Online]. *Proceedings of the 8th international conference on mobile and ubiquitous multimedia*. New York, NY, USA: Association for Computing Machinery, MUM '09. Available from: https://doi.org/10.1145/1658550.1658559.

Leo, M..M..., 2014-2018. *Get your party started with festify* [Online]. Available from: https://festify.rocks/ [Accessed 2021-03-24].

Microsoft, 2021. *Code editing redefined* [Online]. Available from: https://code.visualstudio.com [Accessed 2021-03-24].

Microsoft Corporation, n.d. *Microsoft powerpoint* (v.2019 (16.0)). Available from: https://office.microsoft.com/powerpoint.

Müller, M., Otero, N. and Milrad, M., 2016. Shared interactive music experiences in public spaces: User engagement and motivations [Online]. *Proceedings of the 2016 acm international conference on interactive surfaces and spaces*. New York, NY, USA: Association for Computing Machinery, ISS '16, p.287–296. Available from: https://doi.org/10.1145/2992154.2992183.

O'Hara, K., Lipson, M., Jansen, M., Unger, A., Jeffries, H. and Macer, P., 2004. Jukola: Democratic music choice in a public space [Online]. *Proceedings of the 5th conference on designing interactive systems: Processes, practices, methods, and techniques*. New York, NY, USA: Association for Computing Machinery, DIS '04, p.145–154. Available from: https://doi.org/10.1145/1013115.1013136.

OutLoud, 2020. *Your social jukebox* [Online]. Available from: https://outloud.dj/ [Accessed 2021-03-24].

Resetar, M., 2019. *Flutter firebase and domain driven design principles* [Online]. Available from: `https://resocoder.com/` [Accessed 2021-08-25].

Sprague, D., Wu, F. and Tory, M., 2008. Music selection using the partyvote democratic jukebox [Online]. *Proceedings of the working conference on advanced visual interfaces*. New York, NY, USA: Association for Computing Machinery, AVI '08, p.433–436. Available from: `https://doi.org/10.1145/1385569.1385652`.

stackoverflow, 2019. *Developer survey results 2019* [Online]. Available from: `https://insights.stackoverflow.com/survey/2019` [Accessed 2021-08-25].

stackoverflow, 2020. *Developer survey results 2020* [Online]. Available from: `https://bit.ly/3Dtii4J` [Accessed 2021-08-25].

Stavness, I., Gluck, J., Vilhan, L. and Fels, S., 2005. The MUSICtable: A map-based ubiquitous system for social interaction with a digital music collection [Online]. *Entertainment computing - ICEC 2005*. Springer Berlin Heidelberg, pp.291–302. Available from: `https://doi.org/10.1007/11558651_29`.

Steininger, D.M. and Gatzemeier, S., 2019. Digitally forecasting new music product success via active crowdsourcing. *Technological forecasting and social change* [Online], 146, pp.167–180. Available from: `https://doi.org/10.1016/j.techfore.2019.04.016`.

Storage, C., 2017. *Cloud storage* [Online]. Available from: `https://firebase.google.com/docs/storage` [Accessed 2021-08-25].

VSCode, M., 2017. *Visual studio code at connect(); 2017* [Online]. Available from: `https://code.visualstudio.com/blogs/2017/11/16/connect` [Accessed 2021-08-25].

Wu, L., Mattila, A.S. and Hanks, L., 2015. Investigating the impact of surprise rewards on consumer responses. *International journal of hospitality management* [Online], 50, pp.27–35. Available from: `https://doi.org/10.1016/j.ijhm.2015.07.004`.

Zwaan, K. and Bogt, T.F. ter, 2009. Research note: Breaking into the popular record industry. *European journal of communication* [Online], 24(1), pp.89–101. Available from: `https://doi.org/10.1177/0267323108098948`.

Appendix A

Project Plan

Below figure shows the detailed project plan:

Tasks	Sub-Tasks	Start	End
Project Proposal	Decide Topic	03-Mar-21	24-Mar-21
	Problem Description		
	Related Work		
	Project Requirements/ Objectives		
	Resources		
Literature Review	Find Papers/ Articles	25-Mar-21	29-Mar-21
	Evaluate	30-Mar-21	07-Apr-21
	Draft Literature Review	08-Apr-21	15-Apr-21
	Complete Literature Review	15-Apr-21	25-Apr-21
Design and Documentation	Explore Music Provider API	20-May-2021	25-May-2021
	UI Design - Home, Game Mode, Song Player	26-May-21	31-May-21
	Login Page Design and Login Code	01-Jun-21	02-Jun-21
	Logic for creating and sharing party	03-Jun-21	04-Jun-21
	Dissertation Documentation Update	04-Jun-21	05-Jun-21
	Logic for Editing and Voting on Playlist	06-Jun-21	09-Jun-21
	UI Design for Song Player Host and User views	10-Jun-21	12-Jun-21
	Logic for Game Mode	13-Jun-21	15-Jun-21
	Dissertation Documentation Update	15-Jun-21	16-Jun-21
	Capture Data for Game Mode	17-Jun-21	18-Jun-21
	Generate Report for Game mode	19-Jun-21	20-Jun-21
Development	Sprint-1	21-Jun-21	27-Jun-21
	Sprint-2	28-Jun-21	04-Jul-21
	Sprint-3	05-Jul-21	11-Jul-21
	Sprint-4	12-Jul-21	18-Jul-21
	Sprint-5	19-Jul-21	25-Jul-21
	Sprint-6	26-Jul-21	01-Aug-21
Deployment and Testing	Manual Testing and Group Testing	02-Aug-21	04-Aug-21
Final Documentation	*Complete Dissertation and Review*	01-Aug-21	29-Aug-21

Figure A.1: Project Plan

Appendix B

Tools and Software

B.1 Tools and Software

This section explains all the technical tools and resources used to build this application. Also having prior experience in the chosen technical stack gave more confidence to continue with the same stack for the development of this application. The main consideration while choosing the software is the idea of adapting mobile-first approach, where the main objective is to build a mobile application with lesser delivery time.

B.1.1 Language and Framework - Dart and Flutter

Dart (Google, 2021a) is an object oriented language from Google, mostly used for client side development purpose, although it can also be used for server side development. Dart is relatively new language with its first official release on 10th October, 2011. Dart has syntax similar to C-language that makes it much easier to understand and code. The simplicity of Dart makes it as the first choice. Flutter (Google, 2021c) is a google's UI software development kit and it is among the top frameworks which are currently popular for creating user interface for mobile application. Flutter has gained a lot of popularity within a very small span of time. It is built for designing attractive user interface with minimum efforts and without spending much time. In flutter all the user interface objects are called widgets and it come with a wide range of out-of-the-box widgets to help support the application development.

When working on a mobile application in Dart and Flutter, the only language that a developer is required to learn is Dart. Flutter uses dart as its coding language, so there is no separate need of learning HTML or CSS for building the user interface. This the main reason for choosing Dart and Flutter.

One of the most prominent reason for using Flutter and Dart is their support for ***cross-platform development***. Code binaries can be generated for Android, iOS and web application using the same code base. This eliminates the need of writing and maintaining different code bases for iOS, Android, and web applications. Few examples of applications created using flutter are Alibaba (Alibaba, 1999), Groupon (GROUPON, 2008), eBay Motors (eBay, 2006), and many more.

Both Flutter and Dart are open-source projects with a very active community support that contributes towards its development and continual improvement. Being free to use and having

a large and growing community makes it more likely to get development support. Below figure B.1 shows the results of the developer's survey 2020 conducted by (stackoverflow, 2020). The survey results shows the popularity of flutter and dart, and explains the reason of their selection for this project.

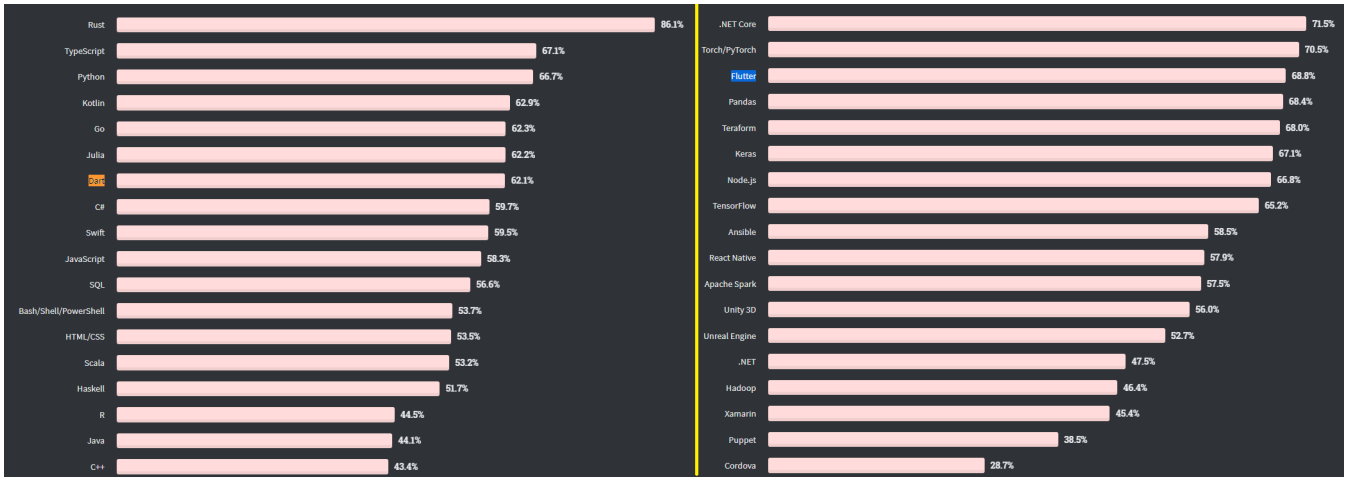


Figure B.1: stackoverflow - Developer's Survey Results, 2020 - Dart and Flutter

B.1.2 Database - Cloud Firestore and Storage

After selecting Flutter and Dart, it becomes an easy decision to select Cloud Firestore (Firestore, 2017) as database and Google Storage (Storage, 2017) for storing application files. Although there are many databases (considering both SQL and No-SQL databases) such as Oracle, MySQL, MongoDB, but it makes more sense to use firestore and storage because of the following reasons:

- Both Cloud Firestore and Google Storage belongs to the same family as Flutter and Dart.
- Firestore is a NoSQL cloud based database, thus it is accessible from anywhere.
- It is a highly scalable and flexible database because is uses collections and documents as compared to the SQL databases with rigid structures consisting of tables, rows and columns.
- The queries written to retrieve data from firestore are very easy to comprehend.
- It also supports indexing over multiple fields, filtering the fetched data, sorting, and ordering of data.
- The updates happen in real-time and the changes reflect as soon as the data is updated.
- Cloud Firestore also support offline updates by caching the frequently used application data.
- Cloud Storage for firebase provides a secure way of storing files using the Firebase authentication.
- Cloud storage also provides the ability to resume the uploads and downloads.

- It is also highly scalable and can easily support a fast growing application.

B.1.3 Code Editor - VS Code

Code editor selection is one of the most crucial decisions when starting with a software development project. Below are the criteria followed for selecting the code editor. The selected code editor must be/have:

- easy and intuitive to use
- easy to organise project files efficiently
- options to configure the look and feel (including fonts and themes)
- options to split or organise the coding windows
- error highlighting and error messages
- easy and efficient debugging mechanism
- mobile application development support

The names of some of the top code editors include Notepad++, IntelliJ, Sublime Text, Android Studio, Atom, Eclipse, etc. These are some of the go-to names when talking about code editors. Microsoft Visual Studio Code (Microsoft, 2021) being relatively new in the list of code editors is one of the most popular editors in recent times. A developer's survey conducted by Stack Overflow (stackoverflow, 2019) from 2019 shows the popularity of VS Code compared with 22 other code editors. More than 87000 responses were collected for this survey. The result shows that VS code is the winner by a good amount of votes, as shown in the figure B.2.

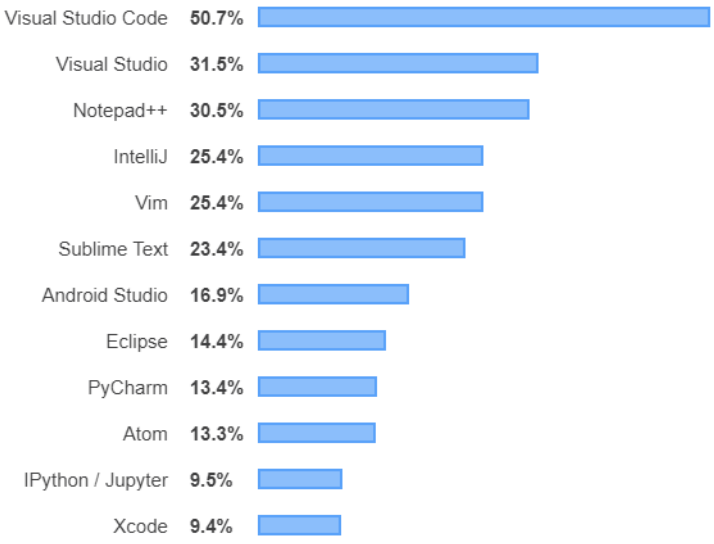


Figure B.2: stackoverflow - Developer's Survey Results, 2019

Within a year of its release in October 2016, Visual Studio Code became very popular. The number of active users for Visual Studio Code increased to 2.6 Million (VSCode, 2017). The first reason for choosing Visual Studio Code is that it is an open-source application. The other reasons are the regular updates from their team and the consistent addition of new features.

It also supports a large number of plugins for various languages. VS Code is easy to use and also supports themes. The support for Flutter and Dart makes this editor the best choice for application development.

Appendix C

Installation Guide

The project code is available in the GitHub repository.

Alternatively, the project installation file can be downloaded from google drive:

Project Installation File

Open the link in the mobile browser to download and install the application.

Appendix D

Maintenance Guide

D.1 Coding Implementation Steps

Below list describes the general implementation steps used to develop the features of this application during the development phase:

- Identify the features of this application and create the project folder structure as shown in the figure 7.5.
- Create the page (UI component) in the presentation folder, under the appropriate feature folder. Add this page to the router file to generate the navigation.
- Create the BLoC files (consists of three dart files corresponding to the event, state and the BLoC) under the application layer, using the BLoC plugin in VS Code.
- Identify the events and states and add them to the event and state dart files.
- Define the mapping of events and states in the BLoC file.
- Register the BLoC using the @injectable annotation (Dependency Injection implemented using the dart package injectable: version 1.4.1).
- Add the BLoC as a provider to the root level of the widget tree i.e. app_widget.dart file to make it available across the application.
- Create a new interface in the domain folder under the appropriate feature folder, if this functionality requires data to be fetched from the repositories. Create the value object and failures, if this functionality requires validation and returning error messages.
- Create a new dart file in the infrastructure folder under the appropriate feature folder, to code the concrete implementation of the interface. This is where the code is written to perform the CRUD operations on the database.

D.2 Database Design

- Below figure D.1 shows the main collections created:

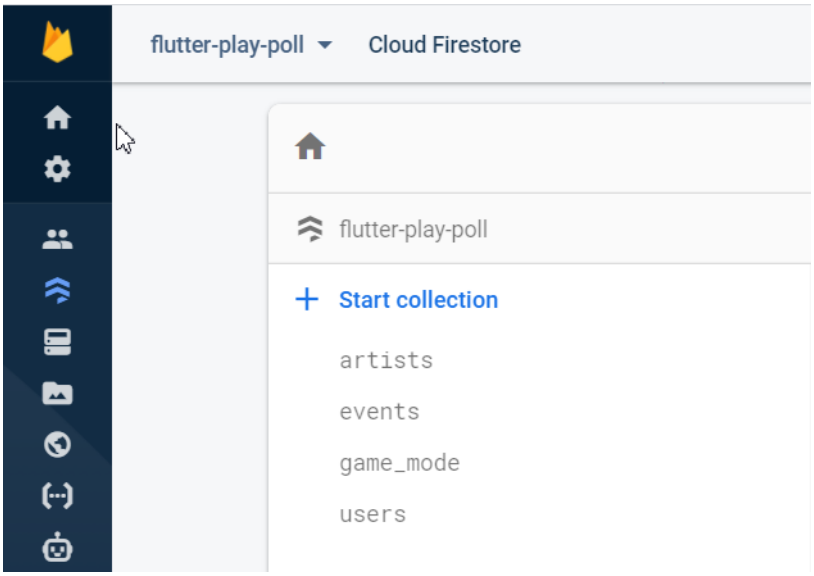


Figure D.1: Main Collection Created in Cloud Firestore Database

- Below figure D.2 shows the restaurant owner data under the users collection:

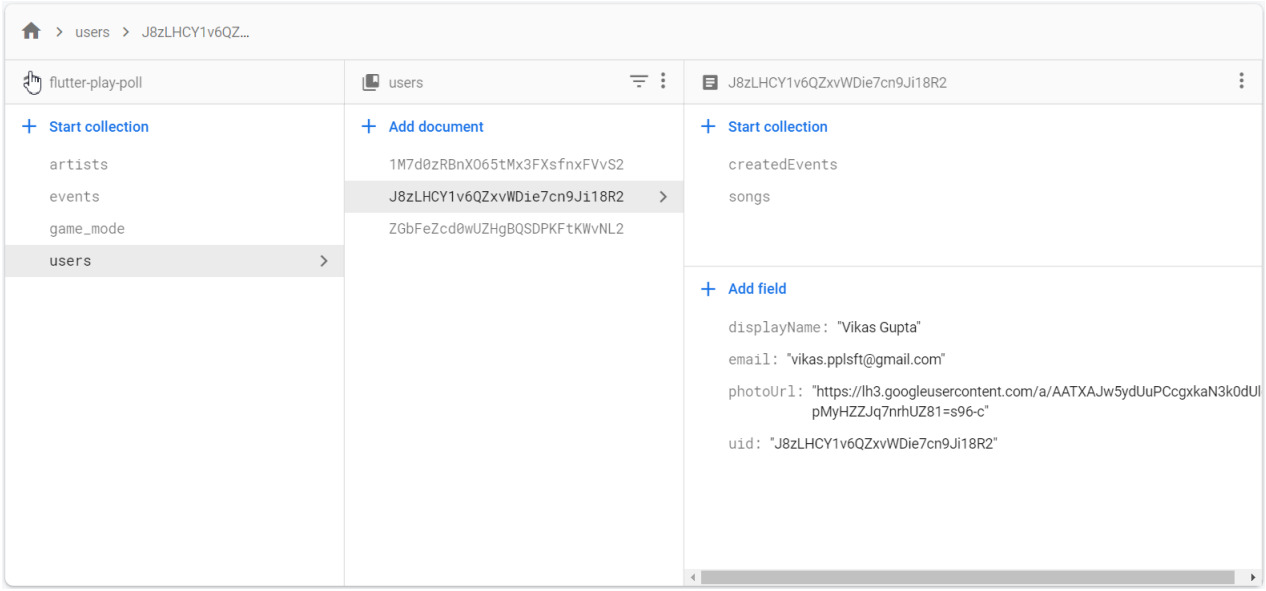


Figure D.2: Restaurant owner document in Cloud Firestore Database

- Below figure D.3 shows the created events under the users > createdEvents collection:

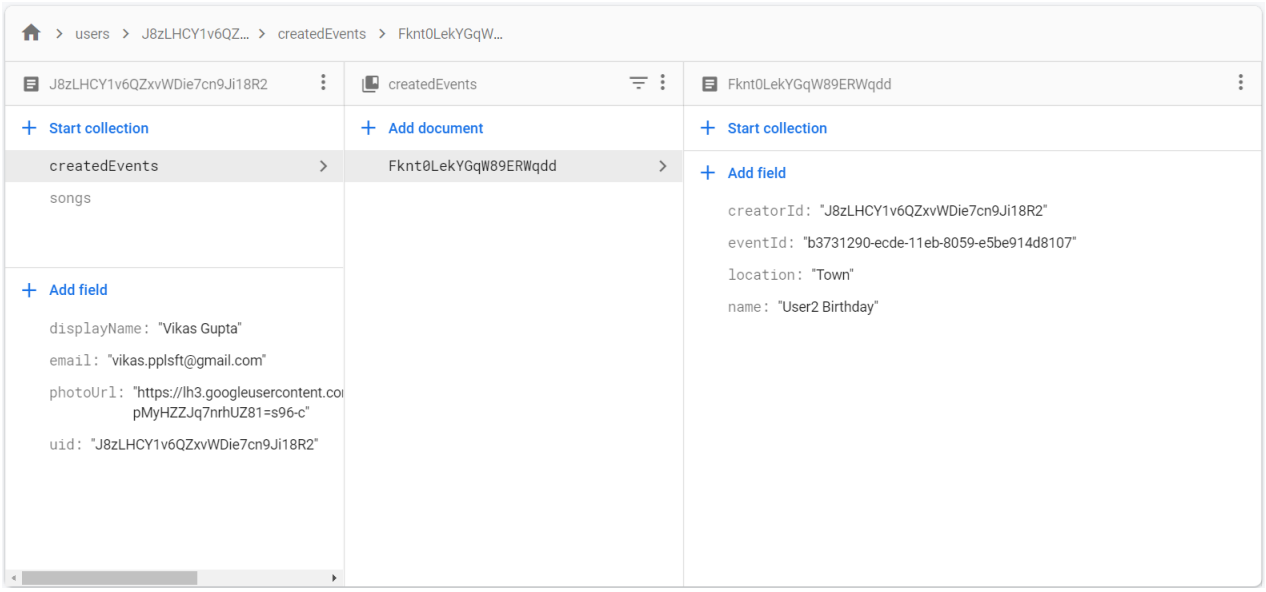


Figure D.3: Created Events Collection Created in Cloud Firestore Database

- Below figure D.4 shows the songs collection:

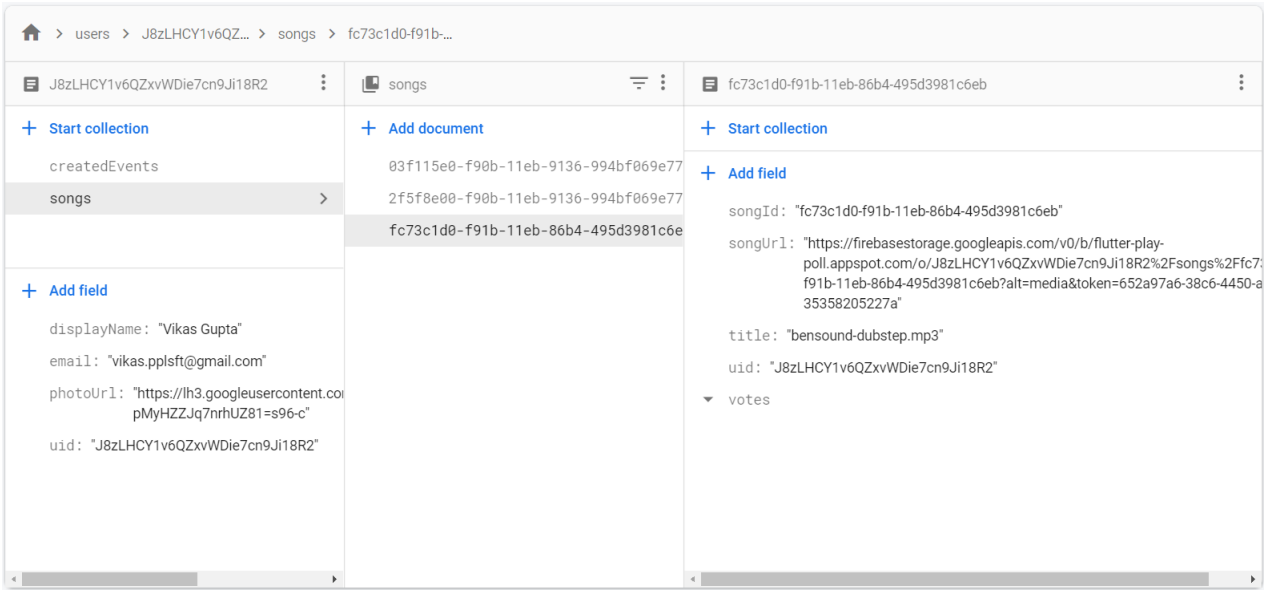


Figure D.4: Event Songs Collection Created in Cloud Firestore Database

- Below figure D.5 shows the joined events collection:

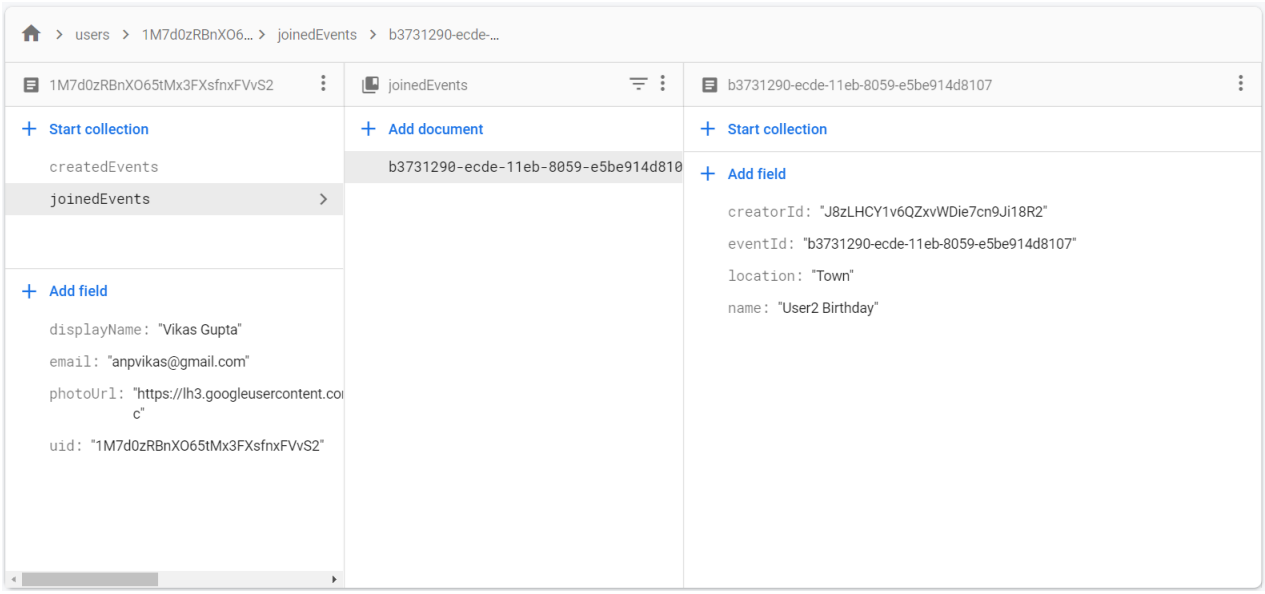


Figure D.5: Joined Events Collection Created in Cloud Firestore Database

- Below figure D.6 shows the events collection:

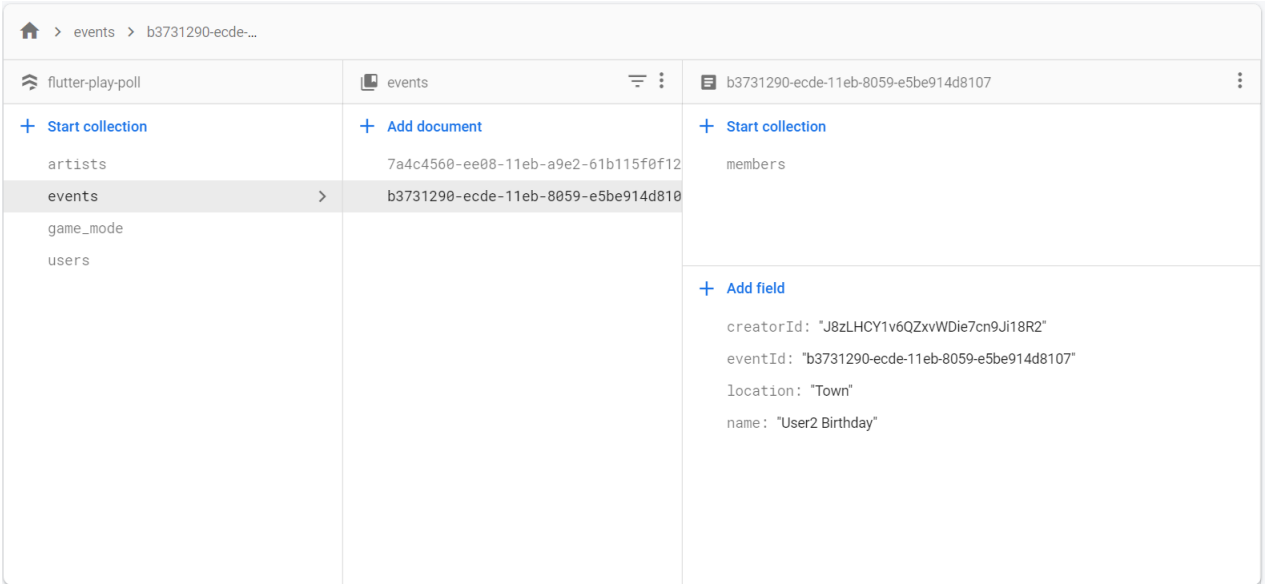


Figure D.6: Events Collection Created in Cloud Firestore Database

- Below figure D.7 shows the event members collection:

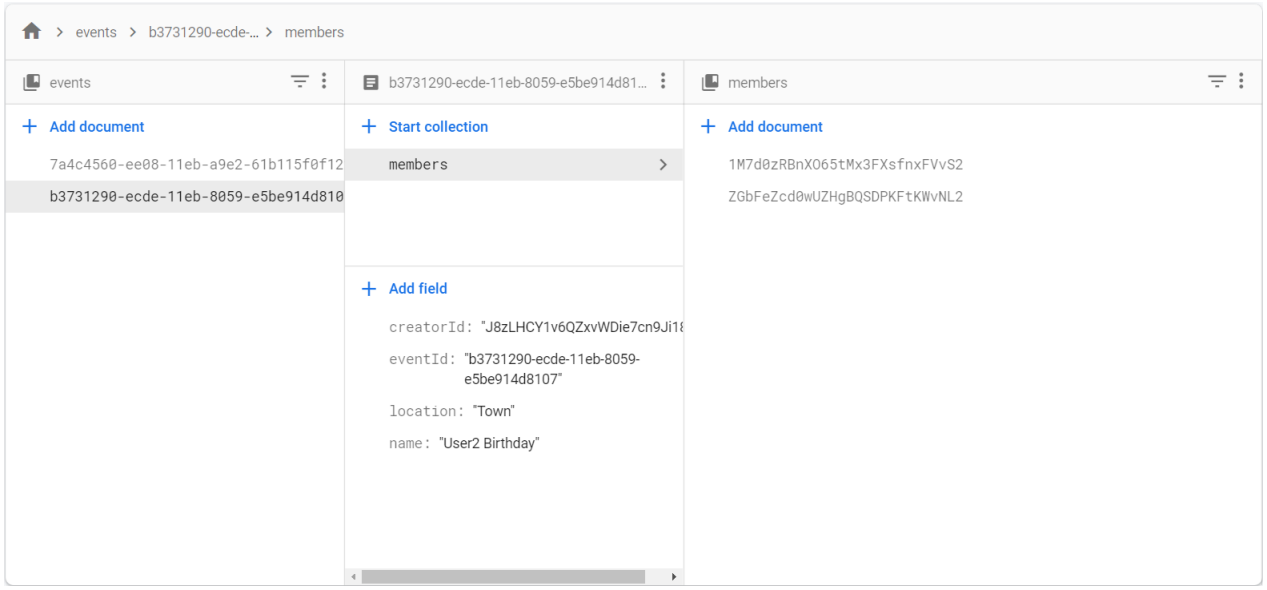


Figure D.7: Event Members Collection Created in Cloud Firestore Database

- Below figure D.8 shows the Artist Collection collection:

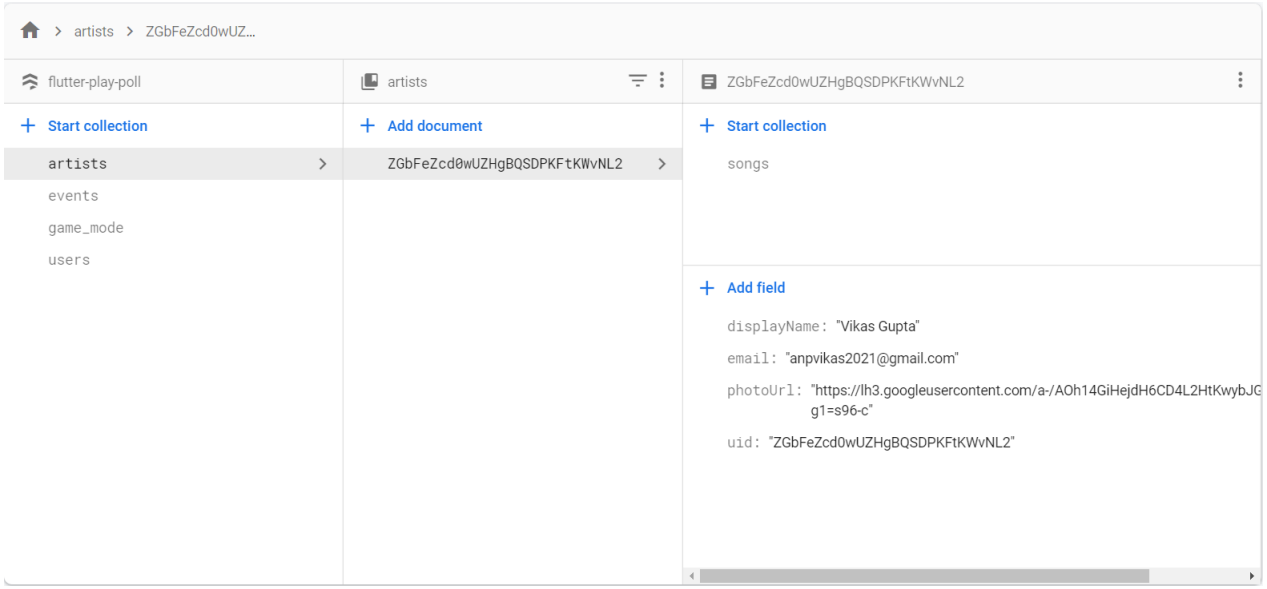


Figure D.8: Artist Collection Created in Cloud Firestore Database

- Below figure D.9 shows the Artist Songs collection:

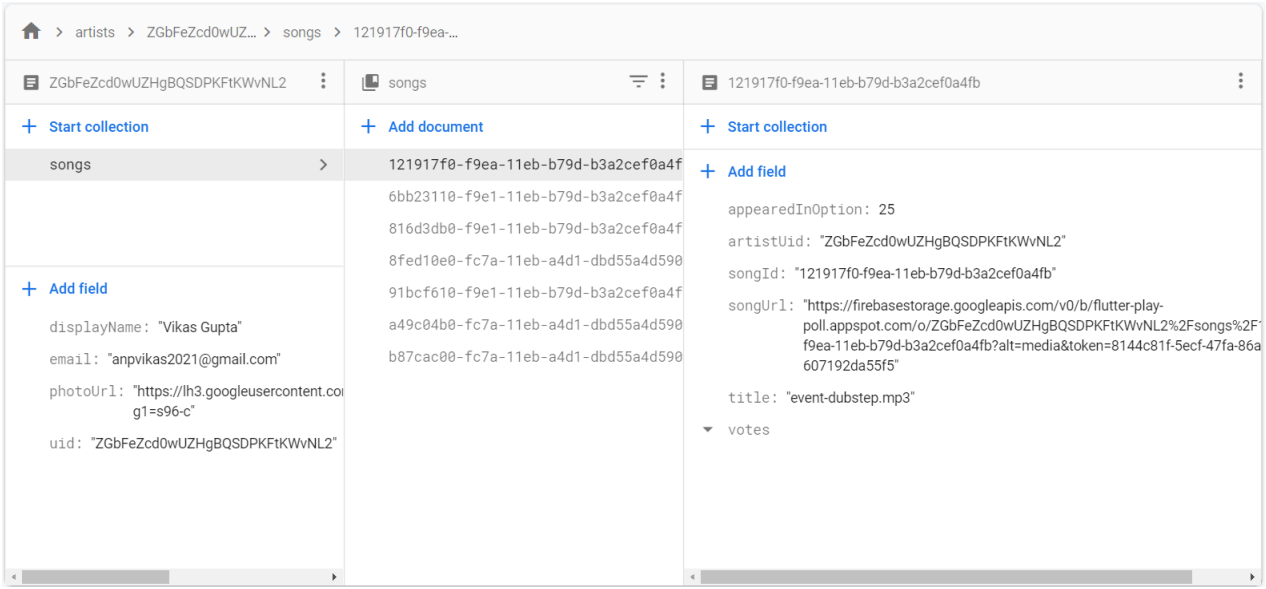


Figure D.9: Artist Songs Collection Created in Cloud Firestore Database

- Below figure D.10 shows the Game Mode collection:

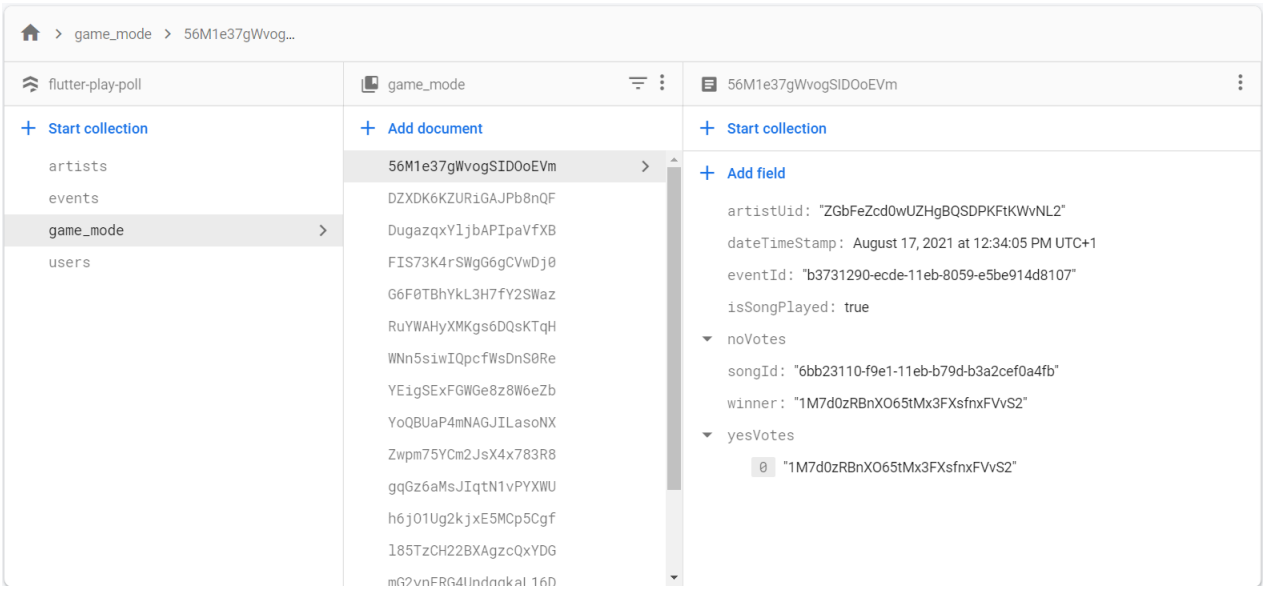


Figure D.10: Game Mode Collection Created in Cloud Firestore Database

- Below figure D.11 shows the songs uploaded by artists in Cloud Storage:

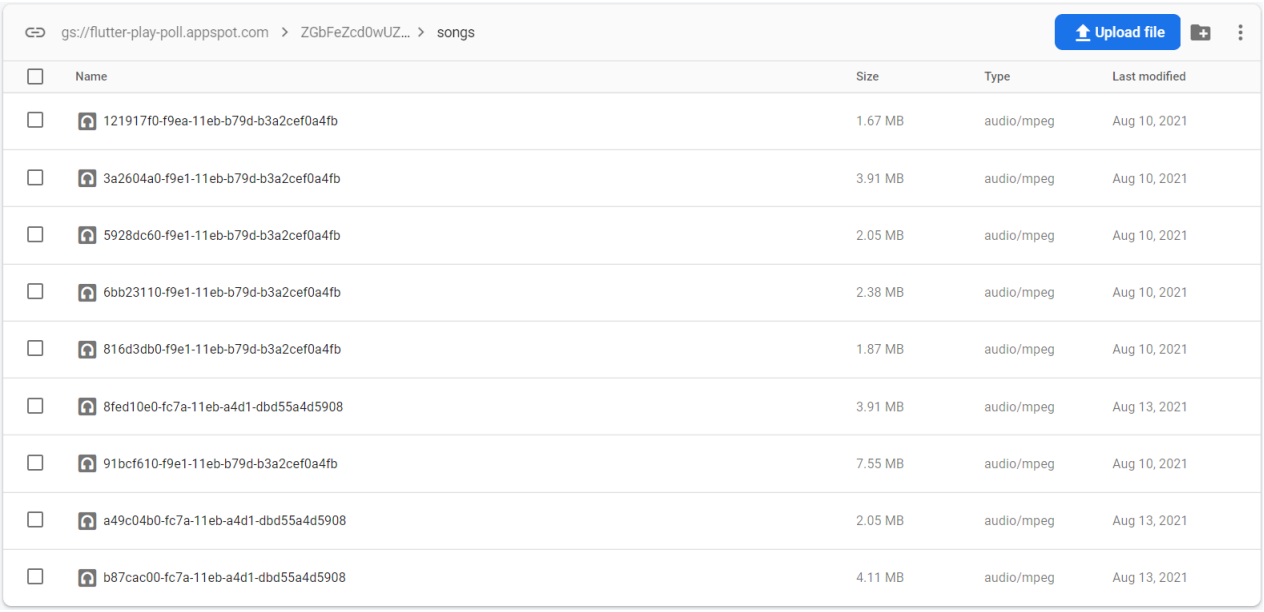


Figure D.11: Artist's Songs in Cloud Storage

- Below figure D.12 shows the songs uploaded by restaurant in Cloud Storage:

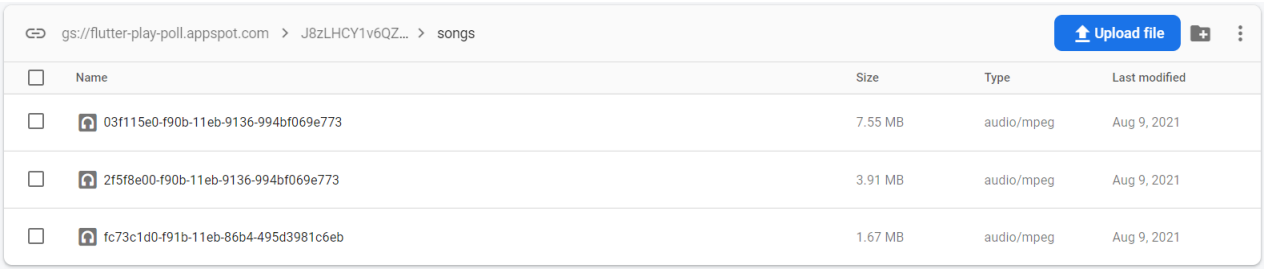


Figure D.12: Restaurant's Songs in Cloud Storage

D.3 Implementation Details

The remaining part of this section will describe the implementation of each of the prominent features of this application and it will follow the same development steps as listed above. The explanation includes the architecture layer-wise details as well. This section will also highlight the implementation decisions taken during the development phase. The feature-wise implementation details are as follows:

D.3.1 Login and Logout Feature

Login and logout feature is the starting point of any application. The login screen is shown in the Figure D.13. The user clicks on the button "Sign in With Google" and a modal window appears to select the google account to login the application. This feature was implemented following the tutorial from (Resetar, 2019). The application currently supports Google sign-in

and the email login will be implemented in future. The login and logout functionality is implemented using the "sign_in_form" and the "auth" BLoCs.

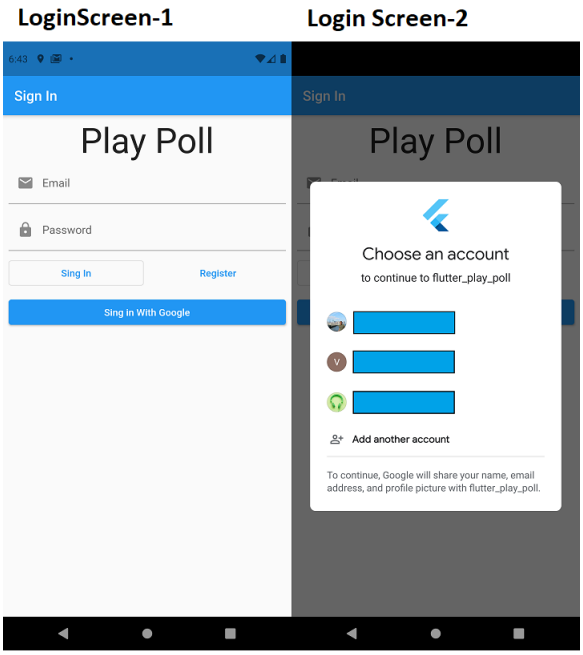


Figure D.13: Login page and Account selection modal window

Presentation Layer

The feature starts with implementing the UI in the presentation folder, under the splash folder and the "sign_in" folder, which includes the "splash_page.dart" and the "sign_in_page.dart" files. These are the pages that will be displayed to the user on opening the application for the first time. Splash page acts as the decision maker page which shows on the screen till the user is authenticated. On authentication, the user is navigated to the home page (Image HFD-1, Figure 7.11).

Application Layer

An "auth" folder is created inside the application folder to represent the authentication/ Login feature of this application and it contains the "auth" and "sing_in_form" BLoCs. The auth BLoC consists of the "authCheckRequested" and "signedOut" events and "authenticated" and "unacuthenticated" states. The "sign_in_form" bloc consists of "signInWithGooglePressed" event and "signInFormState". Below code snippet D.1 shows the mapping for sign_in_form BLoC:

Listing D.1: Event and State mapping for BLoC : sign_in_form

```
Stream<SignInFormState> mapEventToState(
    SignInFormEvent event,
  ) async* {
    yield* event.map(signInWithGooglePressed: (e) async* {
      yield state.copyWith(
        authFailureOrSuccessOption: none(),
```

```
        );
        final failureOrsuccess = await
            _authFacade.signInWithGoogle();
        yield state.copyWith(
          authFailureOrSuccessOption: some(failureOrsuccess),
        );
      });
    }
```

Domain Layer

The user entity, auth failure, and an interface "i_auth_facade.dart" are the part of domain layer. The interface has definitions of methods: getSignedInUser(), signInWithGoogle() and signOut(). The user entity comprises of a unique id, name, email address, and a photo URL.

Infrastructure Layer

This layer has the concrete implementation of the interface methods defined in the domain layer. This is where the actual sign-in code is implemented. Sign-in with google is a generic code which uses the methods from the package google_sign_in: ^5.0.4 and firebase_auth: ^1.4.1. The method signInWithGoogle() is implemented using the signIn() method from package google_sign_in, and then getting the user's authentication details to generate the authentication credentials. This authentication credential is then provided to the firebaseAuth's signInWithCredential() method, and the user gets signed in to the application.

Explanation: The event is triggered by the user on clicking the "Sign in With Google" button, which goes into the "sign_in_form" BLoC. The BLoC calls the signInWithGoogle() method to login the user into the application. User can click on the logout button on the home page (as shown in image HFD-1 of Figure 7.11). This will trigger the "signedOut" event in the auth BLoC, which in return calls the singOut() method on google sign-in object and the firebaseAuth object and the user is signed out of the application.

D.3.2 Create Event Feature

The restaurant staff logs into the application and lands on the home page (Image HFD-1, Figure 7.11). User clicks on the "Create Event" button and navigates to the "Create Event Page" (Image HFD-2, Figure 7.11).

Presentation Layer

The "Create Event" page is in the presentation folder, under the create_event folder, with file name "create_event_page.dart". This page has an event "Name" and an event "Location" text fields, followed by unimplemented "Add Image" and "Add People" button. And finally there are "Create" and "Cancel" buttons. The user fills the event name and event location field and clicks on the create button to create an event.

Application Layer

The prominent events defined for this page are: validateEventName, validateEventLocation, create, cancelButtonClicked.

The corresponding states are: eventNameValidated, eventLocationValidated, eventCreated, eventCreationFailed.
The event name, the event location from the user input and a unique id for event along with a blank creator id are passed to the create() method. Below code (D.2) shows the mapping for create event from the "create_bloc.dart" file:

Listing D.2: Create event mapping in the BLoC : create_bloc

```
create: (e) async* {
    Either<EventFailure, Unit> failureOrSuccess;

    failureOrSuccess = await _eventRepository.create(Event(
        id: UniqueId(),
        name: eventName!,
        location: eventLocation!,
        eventId: UniqueId(),
        creatorId: ''));

    eventLocation = Location('');
    eventName = Name('');
    yield CreateState.eventCreated();
  },
```

Domain Layer

The event entity, event failure, value object and an interface "i_event_repository.dart" are the part of domain layer. This interface has all the methods related to event. This section deals with the create() method specifically. The event entity comprises of a unique id, name, location, event id and creator id. The value objects hold the validated values of event name and event location entered into the text fields in the UI. The event failure class returns error message.

Infrastructure Layer

This layer has the concrete implementation of the create() method defined in the interface "i_event_repository.dart". The "create()" method is responsible for creating an event using the values passed in the create_bloc from the presentation layer. The "create()" method takes the input event and creates an event Data Transfer Object (DTO). For storing data in the database, the data needs to be a primitive data type. The ".toJson()" method converts event DTO to primitive data type. The create() method performs three duties. First, it creates the user in the Firestore database under the "users collection". Second, it creates a new document for the event, under the "users collection" for the logged-in user (that later helps to fetch the list of events created by a particular user.) Third, it also creates an entry for the event under the events collection, where all the events reside.

Below code snippet (Listing D.3), shows the duties of create() method:

Listing D.3: The create() method for creating event in the Repository : event_repository

```
// Create the user
```

```
    await _firestore.collection('users').doc(user!.uid).set({
    'displayName': user.displayName,
    'email': user.email,
    'uid': user.uid,
    'photoUrl': user.photoURL
  });

    // Create the event entry under the logged-in user
    final updatedUserDoc = await _firestore
      .collection('users')
      .doc(user.uid)
      .collection('createdEvents')
      .add(eventDto.toJson());

    // Create the event entry in the events collection
    await _firestore
      .collection('events')
      .doc(eventDto.id)
      .set(eventDto.toJson());
```

Explanation: The validation event is triggered by the user on entering the values for the event name and location followed by the create event, that is triggered by the user on clicking the "Create" button. This event is sent to the "create_bloc.dart" BLoC. The BLoC makes a call to the "create() API" defined in the "event_repository.dart". The "create() API" then performs the three tasks of creating the user entry in the "users collection", creating the event entry under the user entry and creating the event entry under the "events collection".

D.3.3 My Events Feature

The restaurant staff logs into the application and lands on the home page (Image HFD-1, Figure 7.11). User clicks on the "My Events" button and navigates to the "My Events Page" (Image HFD-4, Figure 7.11).

Presentation Layer

The "My Events" page is in the presentation folder, under the "my_events" folder, with file name "my_events_page.dart". This page shows the list of events created by the logged in restaurant staff. The list item consists of the event icon, followed by the event name and event location and a trailing edit button. This is the place to update an event as well, that is discussed in the next feature.

Application Layer

This section will only discuss the "started" event and the "initial" state of the "my_events_bloc.dart" that are related to displaying the list of created events. Below code (Listing D.4) shows the mapping for the "started" event from the "my_events_bloc.dart" file:

Listing D.4: "started" Event and State mapping for BLoC : my_events_bloc

```
    started: (e) async* {
    dynamic received = await _eventRepository.myEvents();
```

```
    yield MyEventsState.showEventsCreated(received);
  },
```

Domain Layer

There is no new entity required in the domain layer for this feature. This feature uses the same interface defined during the create_event feature development. For implementing this feature, a new method definition "myEvents()" is added to the "i_event_repository.dart" interface.

Infrastructure Layer

This layer has the concrete implementation of the "myEvents()" method defined in the interface"i_event_repository.dart". The "myEvents()" method is responsible for fetching all the events created by the logged-in user by using the creator_id field that was introduced at the time of creating event. The code for this feature is a simple fetch query where the creator_id is equal to the logged-in user id. The code for the same is shown below in listing D.5:

Listing D.5: "myEvents()" method for fetching user created events in the Repository : event_repository.dart

```
    await _firestore
      .collection('users')
      .doc('${user!.uid}')
      .collection('createdEvents')
      .where('creatorId', isEqualTo: user.uid)
      .get()
      .then((QuerySnapshot querySnapshot) {
    querySnapshot.docs.forEach((doc) {
      itemsList.add(doc.data());
    });
  });
```

Explanation: The user selects the "My Events" button on the home page and navigates to the "My Events Page". The default "started" event gets triggered on navigating to the "My Events Page". The "started" event is sent to the "my_events" BLoC. The BLoC makes an API call to the "myEvents()" method defined in the "event_repository.dart". This API fetches the list of events created by the logged in user. This list is then displayed on the "My Events Page".

D.3.4 Update and Delete Events Feature

The user is on the "My Events Page", that has an edit button to update the created event, as shown in Image HFD-4, Figure 7.11. On clicking the edit button, a modal window appears to update the event, as shown in Image HFD-5, Figure 7.11. This section will discuss the feature of updating and deleting an event.

Presentation Layer

The modal window is a part of the "My Events Page" and it has options to "Update" the event name and event location. There is "Delete" option to delete the event, and a "Cancel" button to close the modal window and return back to the "My Events Page".

Application Layer

This section will discuss the "updateEvent" and "deleteEvent" events and the corresponding "updatedEventState" and "deletedEventState" states of the "my_events_bloc.dart" BLoC. The "deleteEvent" event state mapping is same as the updateEvent, except for delete() method is called instead of update(). The details of the selected event on the "My Events Page" is passed in the update() and delete() methods. The details in the update method contains the updated value for the event name and event location fields. Below code (Listing D.6) shows the mapping for the "updateEvent" events from the "my_events_bloc.dart" BLoC.

Listing D.6: "updateEvent" and "deleteEvent" events and State mapping for BLoC : my_events_bloc

```
updateEvent: (e) async* {
    print('Updating Event');
    final possibleFailure = await _eventRepository.update(
      Event(
          id: e.event.id,
          name: e.event.name,
          location: e.event.location,
          eventId: e.event.eventId,
          creatorId: e.event.creatorId),
    );
    yield possibleFailure.fold(
      (f) => MyEventsState.updatedFailedState(),
      (_) => const MyEventsState.updatedEventState(),
    );
  },
```

Domain Layer

There is no new entity required in the domain layer for this feature. This feature uses the same interface defined during the create_event feature development. For implementing this feature, two new method definitions "update()" and "delete()" are added to the "i_event_repository.dart" interface.

Infrastructure Layer

This layer has the concrete implementation of the "update()" and "delete()" methods added to the "i_event_repository.dart" interface. The update event is responsible for updating the new values of event name and event location in the database. This is done by passing the updated event object to the update() method call. In case of the delete action by user, the details of the event selected on the "My Events Page" is passed to the delete() method. When creating the events, the entries were made to both the users collection as well as the events

collection. Therefore, the events must be updated or deleted from both the locations. Below code (Listing D.7) shows the update method:

Listing D.7: Update Event API from the Repository : event_repository.dart

```
await userDoc.createdEventCollection
  .doc(eventDto.id)
  .update(eventDto.toJson());

await _firestore
      .collection('events')
      .doc(eventId)
      .update(eventDto.toJson());
```

Explanation: The user clicks on the edit button for an event listed on the "My Events Page" to open the modal window which shows the options to update event, delete event or cancel the modal window. The user triggers the events by clicking on the update, delete or cancel action buttons. The cancel button pops the current page on button click. The update and delete button triggers respective events which is sent to the BLoC and the corresponding API call is made to update or delete the event in the database.

D.3.5 Upload Event Songs Feature

After creating an event, the restaurant staff needs to upload the songs that will be played in the Normal Mode of the application. These uploaded songs are only accessible to the users who have joined the event. The users vote on these songs to place their song request. This section discusses the songs upload feature (Image HFD-3, Figure 7.11).

Presentation Layer

The "Upload Songs" page is in the presentation folder, under the "upload_event" folder, with the file name "upload_event_page.dart". The "Upload Songs" page has three actionable buttons. The first button is to open the location to select songs to upload. The second button is to upload the selected song and the third button takes the user back to the home page.

Application Layer

The "selectSongFileClicked" and "uplodFileClicked" events are mapped to states "selectSongFileState" and "uploadFileState". Below is the mapping code (Listing D.8):

Listing D.8: Event and State mapping for file upload in BLoC : upload_event_bloc

```
selectSongFileClicked: (e) async* {
    dynamic filePath = await _iStorageRepository.selectSong();
    yield UploadEventState.selectSongFileState(filePath);
  },

uplodFileClicked: (e) async* {
    dynamic uploadFileTask = await
        _iStorageRepository.uploadSong();
    yield UploadEventState.uploadFileState(uploadFileTask);
```

```
    },
```

Domain Layer

A new interface is defined for file storage in the domain folder under storage folder with file name "i_storage_repository.dart". This file has the definition for selectSong() and uploadSong() methods.

Infrastructure Layer

Below are the code lines (Listing D.9) for file upload from uploadSong() method, to upload the file in Cloud Storage under the "songs" folder:

Listing D.9: Event and State mapping for BLoC : sign_in_form

```
final snapshot = await (FirebaseStorage.instance
        .ref('$uid/songs/$songId')
        .putFile(fileObj!))
    .whenComplete(() {});
```

Explanation: The user navigates to the "Upload Songs Page", clicks on "Select Song File" button to browse the files and selects a file. User clicks on the "Upload Song File" to upload the file. The upload song utility for artists is also similar to the events song utility but it also makes an entry of the song in the "artists" collection with the details of the artist song.

D.3.6 Search and Join Event Feature

The search functionality is for the users to search for an event and join the event, as shown in Image HFD-6, Figure 7.11.

Presentation Layer

The search page has a search text field and a pre-populated list of all the events. The user can enter the full event name and enter to search the event. The search result will show the event as list item and the user can click on the "add" button to join the event.

Application Layer

The "started" event pre-populates the page with all the events when the user is on search page. The "searchButtonClicked" event performs the search and the "joinEvent" joins the logged-in user to the event. Below is the event to state mapping code from the BLoC "search_event_bloc.dart":

Listing D.10

Listing D.10: Event and State mapping for BLoC : sign_in_form

```
// Initial "started" Event Mapping
    started: (e) async* {
    dynamic received = await _eventRepository.allEventsFetched();
```

```
        yield SearchEventState.showAllEvents(received);
      },

    // Search Event Mapping
      searchButtonClicked: (e) async* {
        dynamic received = await
            _eventRepository.search(e.queryString);
        if (received.length > 0) {
          yield SearchEventState.searchResultsDisplayed(received);
        } else {
          yield SearchEventState.noDataFetchedState();
        }
      },

    // Join Event Mapping
      joinEvent: (e) async* {
        Either<EventFailure, Unit> failureOrSuccess;
        failureOrSuccess = await _eventRepository.join(
          Event(
              id: UniqueId.fromUniqueString(e.data['eventId']),
              name: Name((e.data['name']).toString()),
              location: Location((e.data['location']).toString()),
              eventId: UniqueId.fromUniqueString(e.data['eventId']),
              creatorId: (e.data['creatorId']).toString()),
        );
        eventLocation = Location('');
        eventName = Name('');
        yield SearchEventState.joinState();
      },
```

Domain Layer

New methods allEventsFetched(), search() and join() definitons are added to the existing interface "i_event_repository.dart" file.

Infrastructure Layer

The allEventsFetched() method performs a simple fetch query to get all the application events, as shown below (Listing D.11):

Listing D.11: allEventsFetched() method implementation in Repository : event_repository

```
        await _firestore
          .collection('events')
          .get()
          .then((QuerySnapshot querySnapshot) {
        querySnapshot.docs.forEach((doc) {
          itemsList.add(doc.data());
        });
      });
```

The search() methods performs a conditional fetch query by finding the match for the text entered by the user in all the events, using the "where" clause, as shown below (Listing D.12):

Listing D.12: search() method implementation in Repository : event_repository

```
await _firestore
  .collection('events')
  .where('name', isEqualTo: inputText)
  .get()
  .then((QuerySnapshot querySnapshot) {
querySnapshot.docs.forEach((doc) {
  itemsList.add(doc.data());
});
});
```

The join() method creates an entry for the user in "users" collection along with the event details under the "joinedEvents" collection and adds the user as member under the "events" collection. Below code lines shows the implementation:

Listing D.13

Listing D.13: join() method implementation in Repository : event_repository

```
// Create User Entry
await _firestore.collection('users').doc(user.uid).set({
'displayName': user.displayName,
'email': user.email,
'uid': user.uid,
'photoUrl': user.photoURL
});

// Add Event Details
userDoc.joinedEventCollection.doc(eventDto.id).set(EventDto(
      eventId: eventDto.eventId,
      creatorId: event.creatorId,
      name: eventDto.name,
      location: eventDto.location,
    ).toJson());

// Create Member Entry
userDoc.joinedEventCollection.doc(eventDto.id).set(EventDto(
      eventId: eventDto.eventId,
      creatorId: event.creatorId,
      name: eventDto.name,
      location: eventDto.location,
    ).toJson());
```

Explanation: The users entries are added to the Firebase database only after the user joins an event. The user performs search for an event using the full name of the event and then clicks on the "add" button to join the event. On joining the event, the user navigates to the "Event Page".

D.3.7 Joined Events Feature

The implementation is similar to that of "My Events" but in this case these are the events joined by the users/customers. The important feature here is the un-join or leaving the event, as shown in Image HFD-7, Figure 7.11.

Presentation Layer

The "Joined Events Page" displays the list of events joined by the user/ customer. Clicking on the list item navigates to the "Event Page". The trailing "cross" icon un-joins the user/ customer from the event.

Application Layer

The default "started" event displays the list of all the joined events. The "unjoinEvent" deletes the event from the list. The "viewSelectedEvent" navigates to the selected "Event Page". Below code (Listing D.14) shows the mapping for "unjoinedEvent":

Listing D.14: "unjoinedEvent" Event and State mapping for BLoC : joined_events_bloc

```
unjoinEvent: (e) async* {
    dynamic received = await _eventRepository.unjoin(
      Event(
          id: UniqueId.fromUniqueString(e.data['eventId']),
          name: Name((e.data['name']).toString()),
          location: Location((e.data['location']).toString()),
          eventId: UniqueId.fromUniqueString(e.data['eventId']),
          creatorId: (e.data['creatorId']).toString()),
    );
    yield JoinedEventsState.unjoinState(received);
  },
```

Domain Layer

New methods joinedEvents() and unjoin() definitons are added to the existing interface "i_event_repository.dart" file.

Infrastructure Layer

The joinedEvents() method performs a simple conditional query to fetch all the events from the "joinedEvents" collection where the userid is equal to the logged-in user.

The unjoin() method deletes the event entry from the "joinedEvents" collection, and also deletes the member from the "events" collection.

Below code (Listing D.15) shows the implementation of the unjoin() method:

Listing D.15: unjoin() method implementation in Repository : event_repository

```
// Delete event from joined events
await _firestore
  .collection('users')
  .doc(user!.uid)
```

```
        .collection('joinedEvents')
        .where('eventId', isEqualTo: eventId)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((element) {
        _firestore
            .collection('users')
            .doc(user.uid)
            .collection('joinedEvents')
            .doc(element.id)
            .delete();
      });
    });

    // Delete the members of the deleted event
    await _firestore
        .collection('events')
        .doc(eventId)
        .collection('members')
        .doc(user.uid)
        .delete();
```

Explanation: This feature is opposite of the join feature. On clicking the un-join button the event gets deleted from the collection "joinedEvents" for the logged-in user, and the user gets deleted from the "members" collection inside the "events" collection.

D.3.8 Event Page Feature

"Event Page" represents the "Normal Mode" of this application and helps users/ customers to place their song request (Image HFD-8, Figure 7.11).

Presentation Layer

The Event Page is divided into two parts, the audio player and the songs playlist with voting button. The audio player part shows the current playing song with the play and stop buttons (to be used by the restaurant staff). The seek-bar and count-down times shows the song progress and duration. It also has the "BEGIN GAME MODE" button.

The second part of the page has the playlist of songs uploaded by the restaurant staff. These are the songs on which the users/ customers vote and place their song request.

Application Layer

The events for the "event_bloc.dart" BLoC are also divided into two parts to cater the requirements of the Normal and Game Mode. The Normal Mode events are: the default "started" event, "incrementVoteCount", "onPlayerCompletionEvent". The event state mapping for Normal Mode is shown below (Listing D.16):

Listing D.16: Normal Mode Event and State mapping for BLoC : event_bloc

```
// To display the songs playlist
```

```
    started: (e) async* {
        dynamic received = await
            _eventRepository.fetchCreatorSongs(e.data);
        yield EventState.showFetchedSongs(received);
      },

    // To vote on songs in the playlist
    incrementVoteCount: (e) async* {
        dynamic received = await
            _eventRepository.registerVote(e.songId, e.uid);
        yield EventState.incrementedVoteCount('$received');
      },

      // To reset vote after the song finishes
      onPlayerCompletionEvent: (e) async* {
        dynamic received =
            await _eventRepository.resetVoteToZero(e.songId, e.uid);
        yield SongsPlayerState.onPlayerCompletionState();
      },
```

The Game Mode events are: "fetchArtistSongsEvent", "createGameModeEntryEvent", "votingStartedEvent", "gameModeVoteEvent", "showWinnerEvent". Below is code (Listing D.17) for event state mapping:

Listing D.17: Event and State mapping for BLoCs : event_bloc and song_player_bloc

```
      // To fetch the artist songs
      fetchArtistSongsEvent: (e) async* {
      dynamic received = await
          _iStorageRepository.fetchArtistSongs();
      yield SongsPlayerState.fetchArtistSongsState(received);
    },

      // To initiate the game mode in database
      createGameModeEntryEvent: (e) async* {
              await _eventRepository.createGameModeEntry(
                  e.eventId, e.songId, e.artistUid);
              yield EventState.createGameModeEntryState();
            },

      // TO display game mode started message
      votingStartedEvent: (e) async* {
              int value = 10;
              int output = 0;
              Timer votingTimer = Timer.periodic(Duration(seconds:
                 1), (votingTimer) {
                value = value - 1;
                output = value;
                if (value == 0) {
                  votingTimer.cancel();
                }
```

```
            });
            yield
                EventState.votingStartedState(votingTimer.tick.toString());
          },

      // To vote in Game Mode
      gameModeVoteEvent: (e) async* {
            await _eventRepository.registerGameModeVote(
                e.songId, e.artistUid, e.voteSmiley);
            yield EventState.gameModeVoteState();
          },

      // To display winner of game mode
      showWinnerEvent: (e) async* {
            dynamic selectedWinner =
                await
                    _eventRepository.decideGameModeWinner(e.eventId,
                    e.songId);
            yield EventState.showWinnerState(selectedWinner);
          },
```

Domain Layer

New methods are added to the existing interface "i_event_repository.dart" file.

Methods added for Normal Mode are: fetchCreatorSongs(), registerVote(), resetVoteToZero().

Methods added for Game Mode are: fetchArtistSongs(), createGameModeEntry(), registerGameModeVote(), decideGameModeWinner().

Infrastructure Layer

This section will show the main lines for implementing the features of Normal Mode and Game Mode.

Normal Mode:

- fetchCreatorSongs() method:
  The application uses this method to display all the event songs on the "Event Page". These are the songs uploaded by the restaurant staff. The implementation is a simple fetch query from the restaurant staff's "songs" collection, sorted by the number of votes in descending order.
- registerVote() method:
  The application calls this method on clicking the "thumbs up" icon to register the users/customers votes for the songs.
- resetVoteToZero() method:
  This method resets the vote count to zero after the song has finished playing.

Game Mode:

- fetchArtistSongs() method:
  This method gets called on click of the "BEGIN GAME MODE" button, and a 5 seconds drum-roll indicates the start of game mode. This method selects a random song from the artist's song repository. The selected song is displayed to the users/customers for voting after listening to the song preview.
- createGameModeEntry() method:
  This method creates a placeholder in the database for the game mode round.
- registerGameModeVote() method:
  This method registers the votes of the users/customers in the game mode.
- decideGameModeWinner() method:
  The application uses this method to select the winner of the game mode. This method gets called after the voting duration is over. The votes are categorised as "Liked" and "Disliked". Whichever category has more votes, the application selects a winner from that category. However, if the winning category is the "Liked" category, the previewed song gets played. Whether the song is liked or disliked, the application always announces a winner.

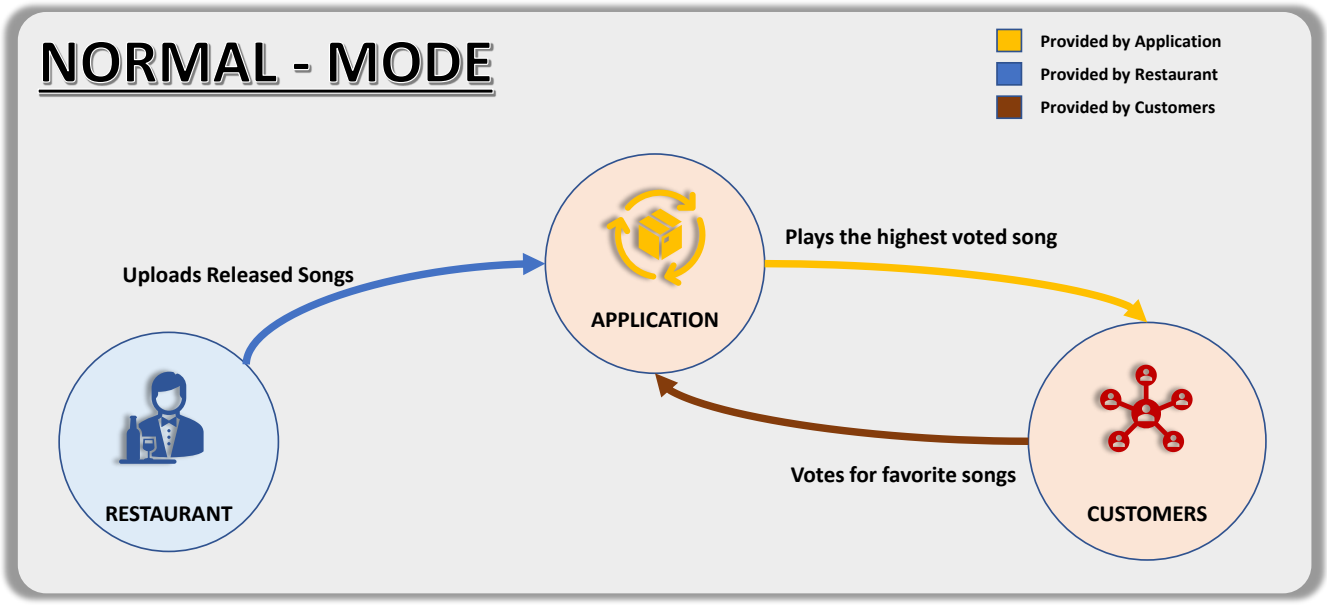
Appendix E

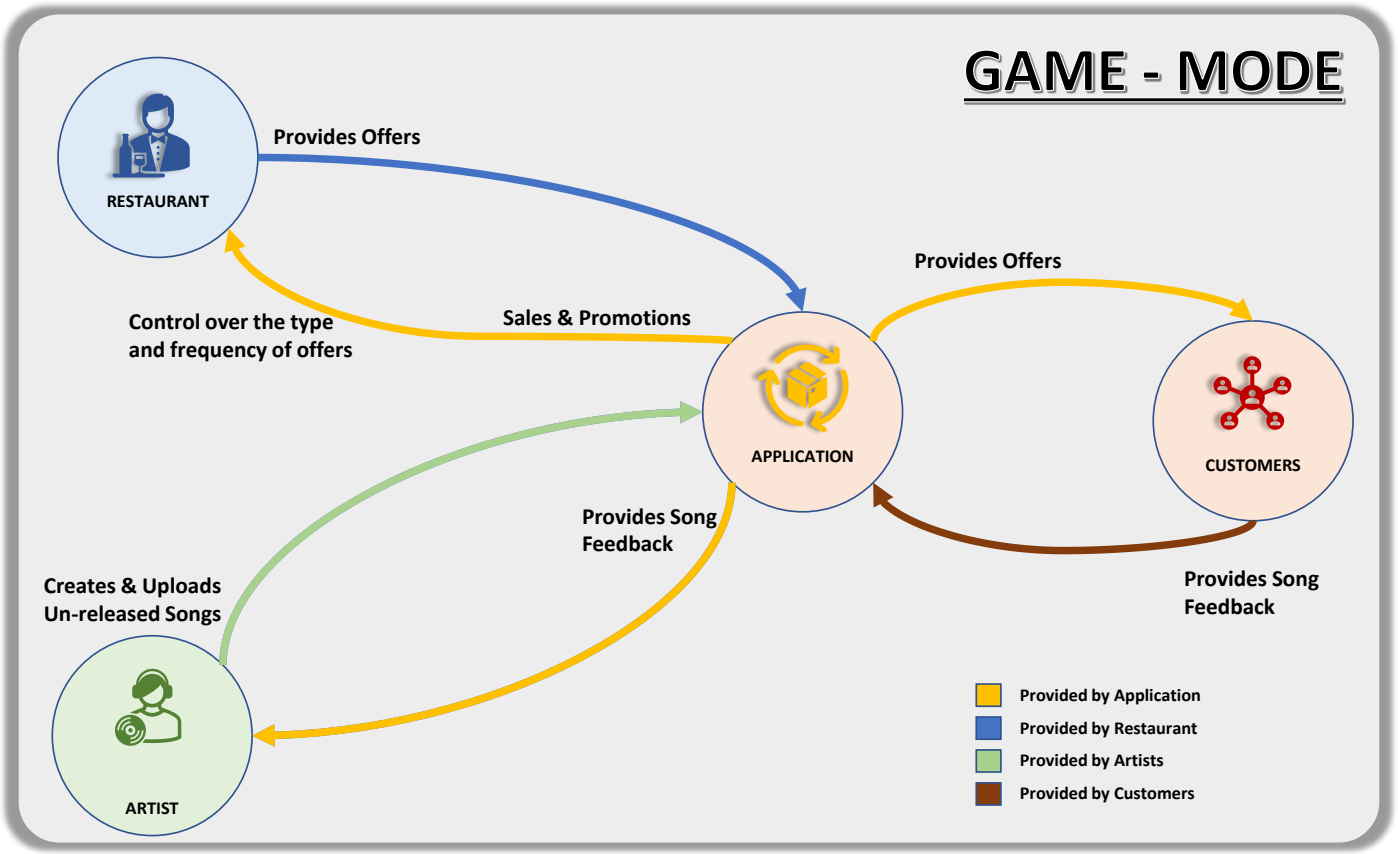
User Manual

Once the application is installed on the device, the next step is to get information about the usage and operations of the various features of this application. This section covers the instructions for efficiently using the features of this application by the end-users.

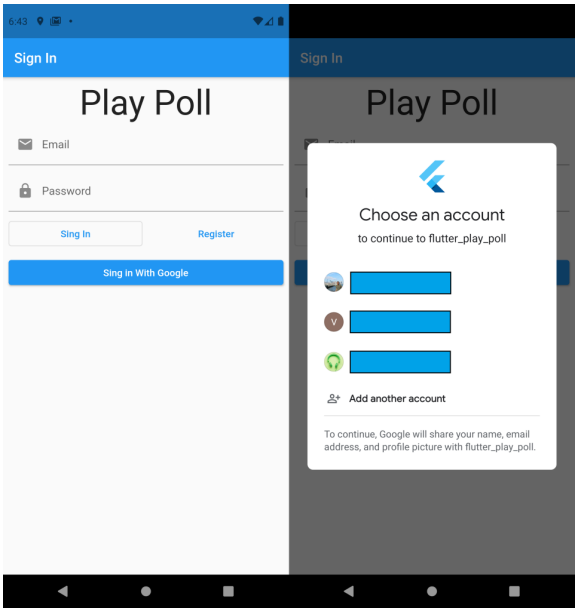
Play Poll

User Guide – version 0.1


NORMAL - MODE
Provided by Application
Provided by Restaurant
Provided by Customers
Uploads Released Songs
APPLICATION
Plays the highest voted song
RESTAURANT
Votes for favorite songs
CUSTOMERS

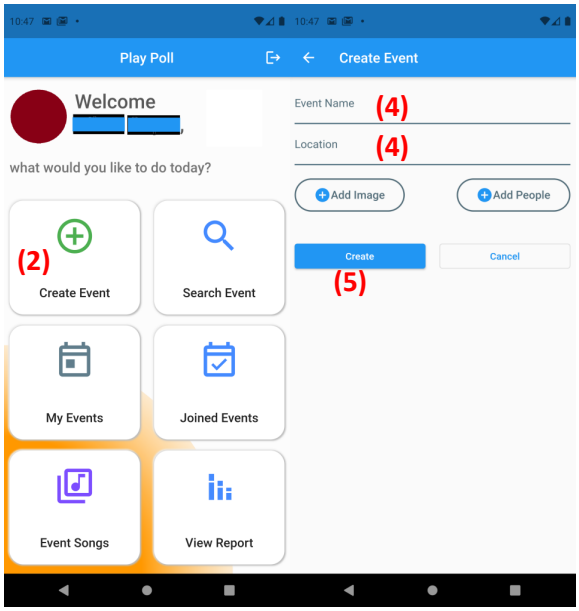

GAME - MODE
RESTAURANT
Provides Offers
Control over the type
and frequency of offers
Sales & Promotions
APPLICATION
Provides Offers
CUSTOMERS
Provides Song
Feedback
Provides Song
Feedback
Creates & Uploads
Un-released Songs
ARTIST
Provided by Application
Provided by Restaurant
Provided by Artists
Provided by Customers

Login



1. This is the login page of the application Play Poll.
2. This is the first page which appears when the application is opened.
3. User can click on the button “Sing in with Google” and login by selecting an account.

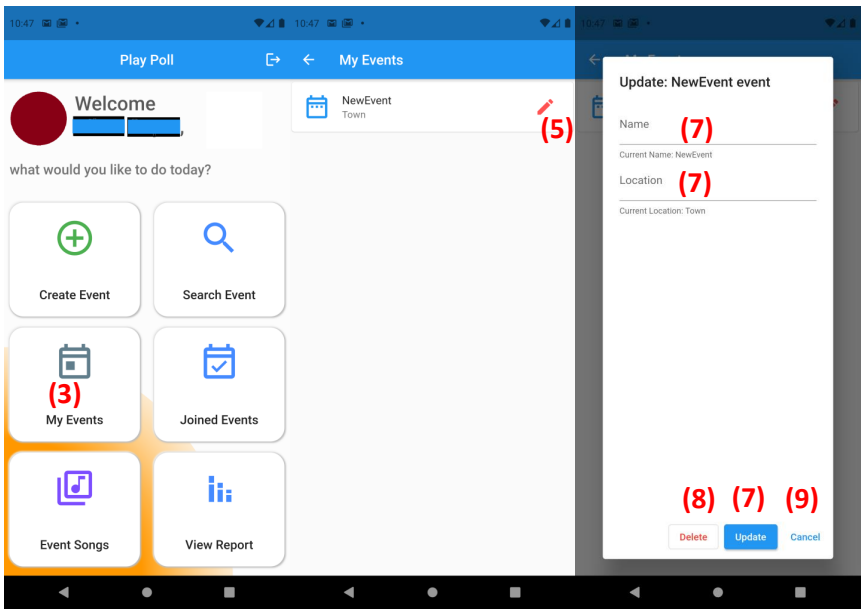
Create Event – (by Restaurant Staff)



Home Page Create Event Page

1. Restaurant Staff Logs into the application and lands on the home page.
2. Clicks on the Create Event Button on the home page
3. Restaurant Staff is navigated to Create Event page
4. Enters the event name and location
5. Clicks on “Create” button
6. Event gets created

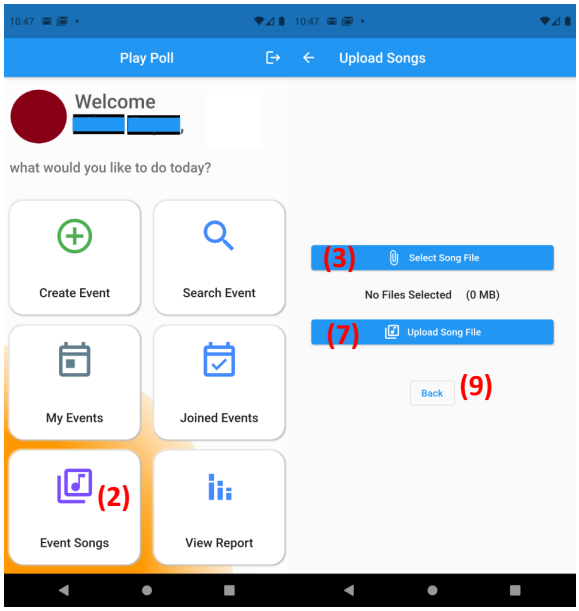
My Events – Update/ Delete Event (by Restaurant Staff)



1. The created events are listed under My Events page
2. Restaurant Staff Logs into the application and lands on the home page
3. Clicks on the My Events button on the home page
4. Restaurant Staff is navigated to My Events page
5. Clicks on the edit icon button
6. Modal window appears
7. Enter new event name and location and click update button and the event detail gets updated
8. To delete the Event, click on the Delete button
9. To go back to previous page, click on Cancel button

Home Page My Events Page Update/ Delete Event

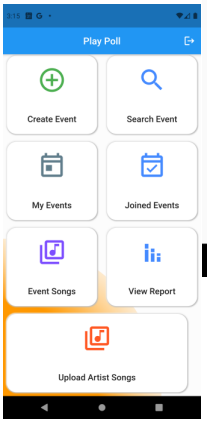
Event Songs and Upload Artist Songs – (by Restaurant Staff and Artists)

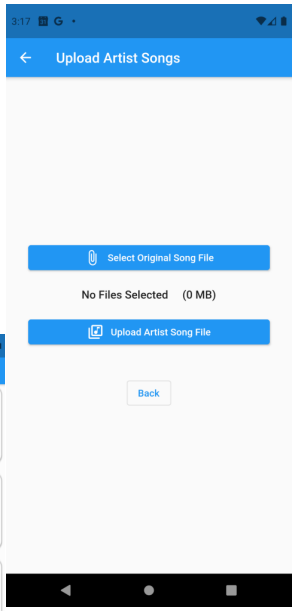


Home Page

Upload Songs

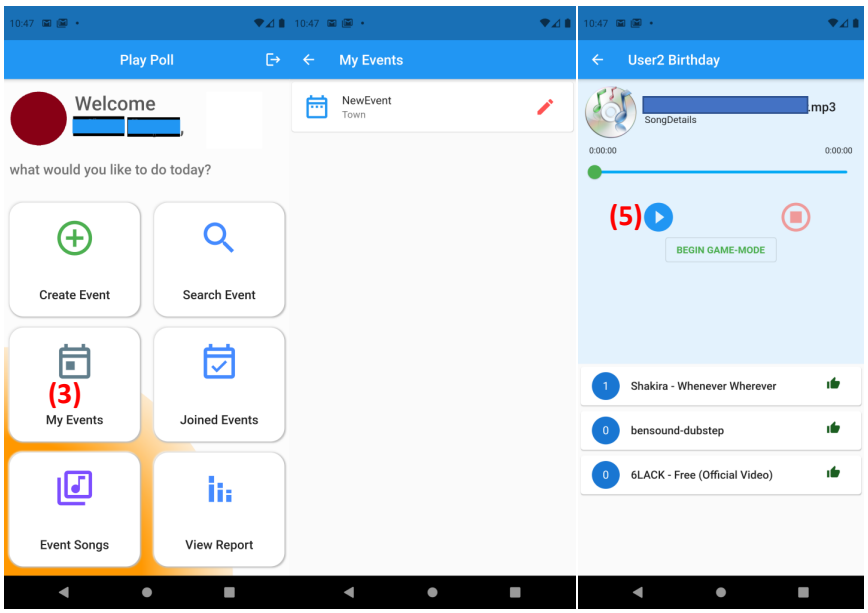
1. Restaurant Staff Logs into the application and lands on the home page
2. Clicks on the Event Songs button on the home page
3. Restaurant Staff is navigated to Upload Songs page
4. Clicks on the Select Song File button
5. Modal window appears to select song file from the device. Select a file.
6. File name and fie size gets displayed on the screen
7. Click on the Upload Song File button
8. File gets uploaded
9. Click on Back button to return to home page





Upload utility for Artist Songs

Event Page – (by Restaurant Staff)

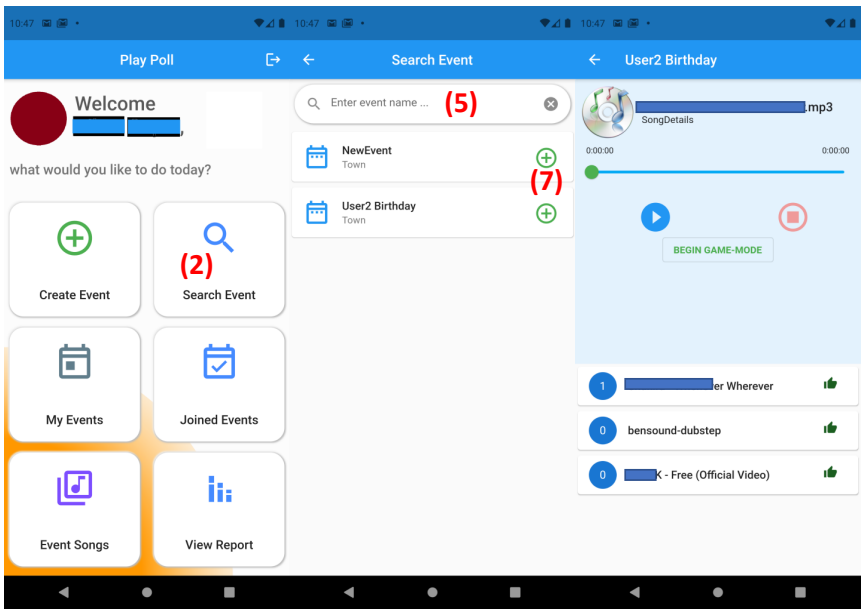


1. The created events are listed under My Events page
2. Restaurant Staff Logs into the application and lands on the home page
3. Clicks on the My Events button on the home page
4. Restaurant Staff is navigated to My Events page
5. Clicks on the Play button to start the Normal Mode

Normal Mode Started

Home Page My Events Page Event Page

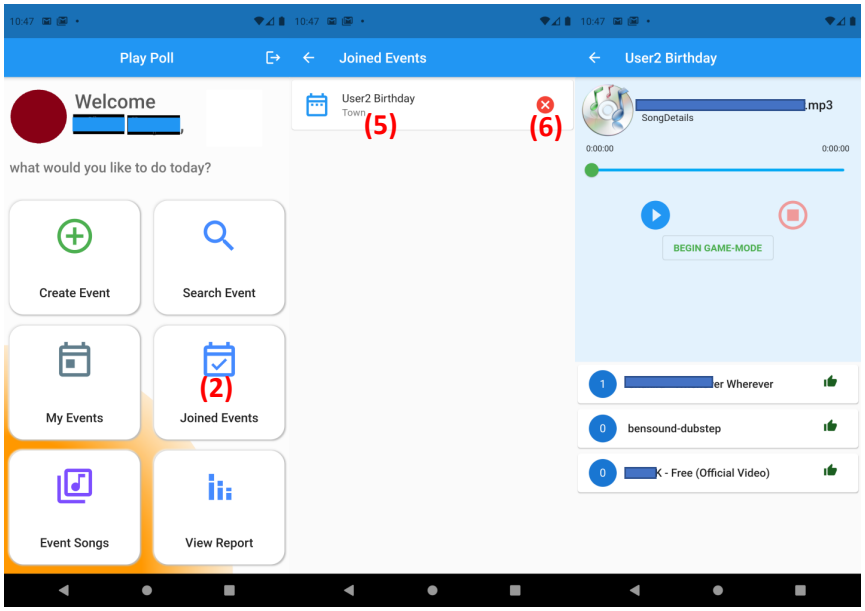
Search and Join Events – (by User/ Customer)



Home Page Search Event Page Event Page

1. User/ Customer Logs into the application and lands on the home page
2. Clicks on the Search Event button on the home page
3. User/ Customer is navigated to Search Event page
4. By default, all events are listed on the search page
5. User/ Customer enters full name of an event and clicks enter
6. The search results shows that event as list item with a green Join button
7. Click on the Join button to join that event
8. User gets navigated to the Event Page

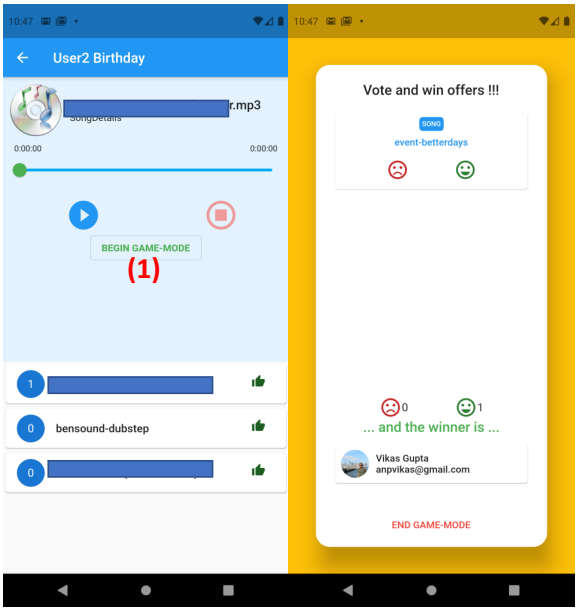
Joined Events – (by User/ Customer)



Home Page Joined Events Page Event Page

1. User/ Customer Logs into the application and lands on the home page
2. Clicks on the Joined Events button on the home page
3. User/ Customer is navigated to Joined Events page
4. All the joined events are listed on the Joined Events page
5. User/ Customer can click on the list item to navigate to the Event Page
6. User can click on the red cancel icon button to un-join the event and remove from the Joined Events list

Event Page – (by Restaurant Staff)



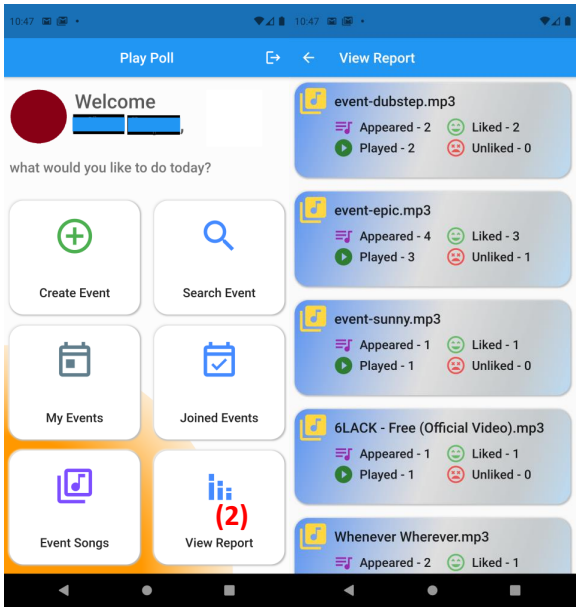
1. Restaurant Staff clicks on the BEGIN GAME MODE button to start the game mode
2. Game mode open in a new window with a drum roll for 5 seconds, and a song appearing in option to vote for.
3. Once the drum roll is over, the 10 seconds song preview is played.
4. After the song preview, a message appears to start voting.
5. On Voting the message disappears
6. After the voting duration is over, the winner is displayed on the screen
7. Restaurant Staff can announce the rewards and offers for the winner
8. Restaurant Staff can now click on the END GAME MODE button to return to the Event Page
9. Restaurant Staff can again start the Normal Mode

Game Mode Started

Event Page

Game Mode Page

View Report



1. Artist Logs into the application and lands on the home page
2. Clicks on the View Report button on the home page
3. Artist is navigated to View Report page
4. Artist can see the report of each of the songs

Home Page

View Report Page

Appendix F

Survey Questions

This section shows the survey questionnaire that was circulated to get the feedback. The survey starts with an initial description of the application, followed by the high-level diagram of interaction between the restaurant, customers, and the artist. Finally the survey shows the user-interface of the application that will be used by the end-users.

Reward-based song feedback in restaurant setting

Hello, please provide your valuable feedback/ suggestions.

This application is designed to provide an ecosystem between a restaurant, its customers and artists, using which all the stakeholders are benefitted.

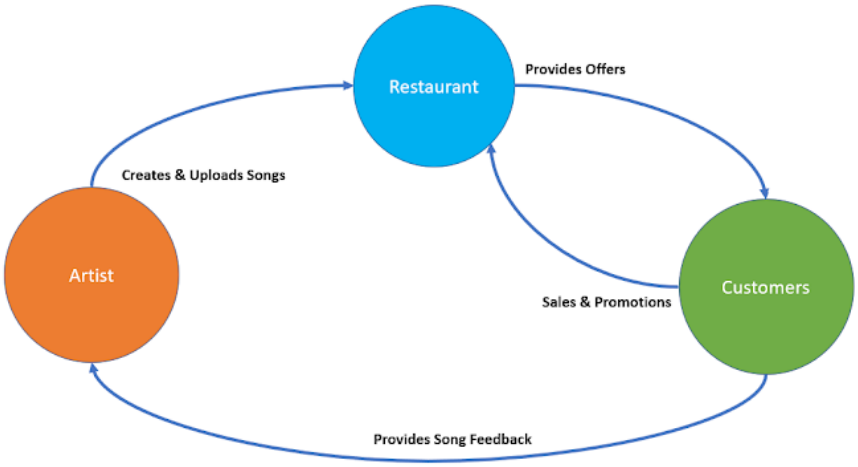
- Artists may benefit by receiving feedback of their new songs.
- Restaurant owners may benefit by regulating the offers/ rewards/ coupons and controlling operating expenses.
- Customers receives offers and rewards by participating and voting for a particular song (by liking or disliking a song).

One aspect of this application lets the customers control the music that they want to listen to by polling for the songs that they like. Highest voted song gets played.
Another aspect is to leverage the customer polls and use them to provide feedback for songs from new artists after listening a 10 seconds preview. This activity is a gamified version of polling as it involves surprise rewards and offers by the restaurant for the winner of the poll, which helps the restaurant with its promotions and sales.

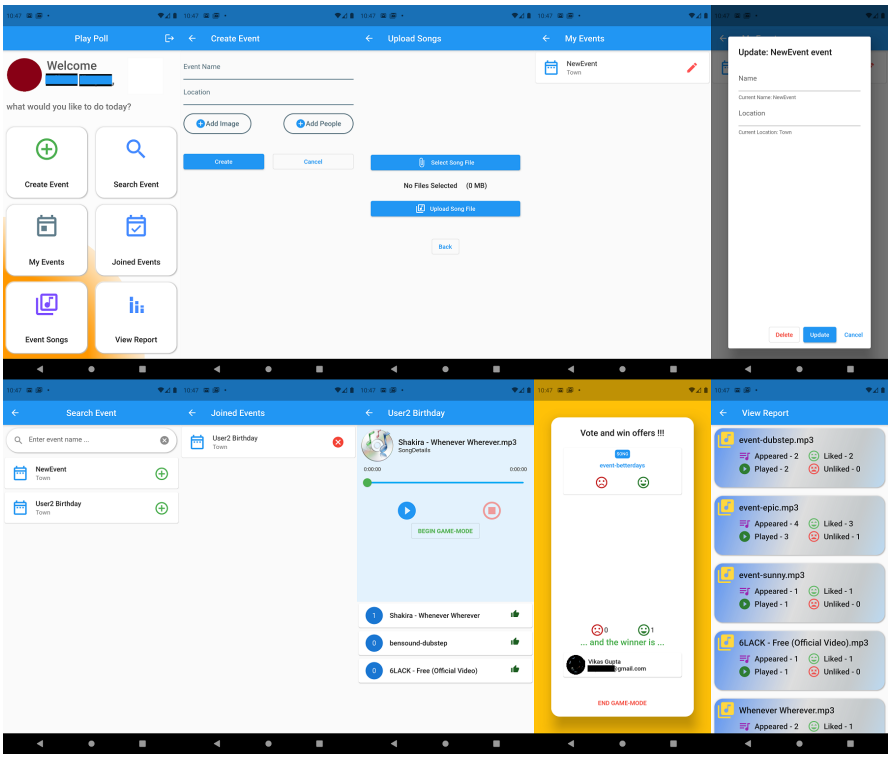
Thank you :)

* Required

Application Overview



Restaurant creates an event. Customers can join events and vote for songs to play. Artist can upload songs and view feedback. Below are the basic UI screens from the application. Please zoom-in to view the image.



1. Is it easy to follow the user-interface of this application ? *

Mark only one oval.

	1	2	3	4	5	
No, its difficult	◯	◯	◯	◯	◯	Yes, its easy

2. How much will you rate the feature of playing the next song based on customer's polls ? *

Mark only one oval.

	1	2	3	4	5	
Lowest	◯	◯	◯	◯	◯	Highest

3. How much will you rate the feature of providing user-feedback to songs (from new artists)? *

Mark only one oval.

	1	2	3	4	5	
Lowest	◯	◯	◯	◯	◯	Highest

4. What you liked the most about this application: *

Check all that apply.

- [ ] Play songs based on poll
- [ ] Getting offers/ rewards for sharing your music taste
- [ ] Helping new artists with feedback (report generated based on polls)
- [ ] Restaurants able to manage rewards/ offers/ coupons
- [ ] Other: _____

5. What you didn't like about this application:

Check all that apply.

- [ ] Play songs based on poll
- [ ] Getting offers/ rewards for sharing your music taste
- [ ] Helping new artists with feedback (report generated based on polls)
- [ ] Restaurants able to manage rewards/ offers/ coupons
- [ ] Other: _____

6. Any comments/ suggestions on the user-interface of this application? Or any feature you would like to add or remove from this application?

7. As an end-user, how likely are you to place your song request in a restaurant/disco/gym setting using an application instead of personally requesting? *

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

8. How likely are you to install and use an application which earns you rewards/discounts in return of listening new songs and rating them? *

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

9. Will this application encourage users to go to restaurant that are using this application? *

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

10. How beneficial could this application be for a new aspiring singer/ band/ musician to get feedback on their work in real-time? *

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

11. If you are an artist/ aspiring artist, how likely are you to use this application for getting feedback?

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

12. How likely do you think is this application going to help restaurants with their promotions? (Asking for your general opinion)

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

13. Keeping track of offers, rewards programs, coupons is an overhead for restaurants. Using this application restaurants will have more control on discounts and rewards by limiting (or increasing) the number of polling rounds. How likely are the restaurants going to use such an application which helps them to regulate the offers/ rewards/ coupons? (Asking for your general opinion)

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

14. How likely it is for a restaurant to use this application to support new artists in return of promotions? (Asking for your general opinion)

Mark only one oval.

	1	2	3	4	5	
Least	◯	◯	◯	◯	◯	Most

Appendix G

Raw Survey Results Output

This section consists of the survey results, as shown below:

Reward-based song feedback in restaurant setting

Questions Responses 11

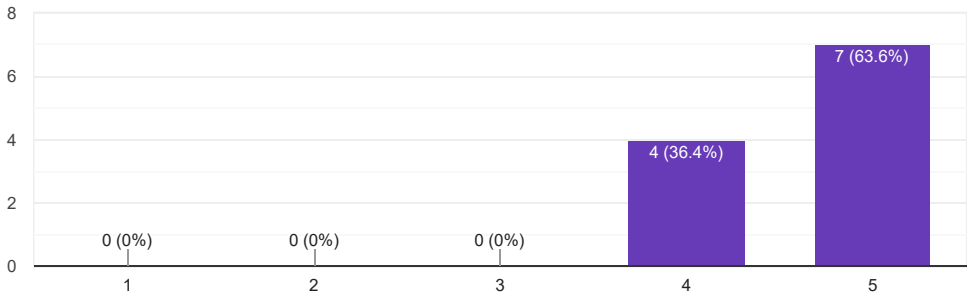
11 responses

Accepting responses

Summary Question Individual

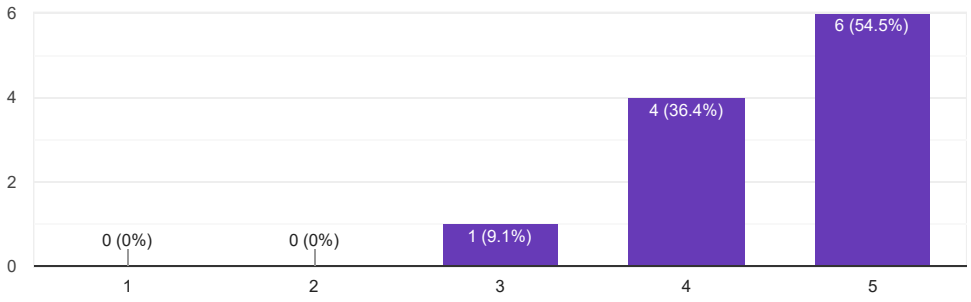
Is it easy to follow the user-interface of this application ?

11 responses



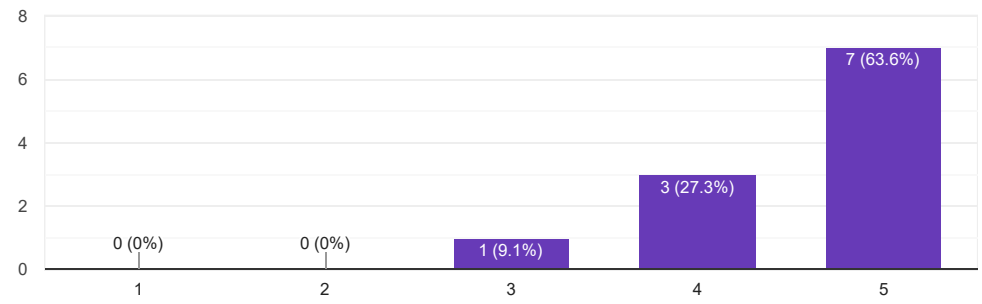
How much will you rate the feature of playing the next song based on customer's polls ?

11 responses



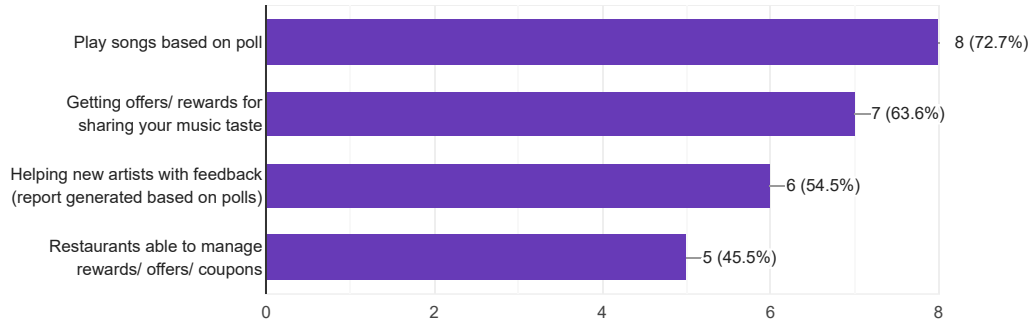
How much will you rate the feature of providing user-feedback to songs (from new artists)?

11 responses



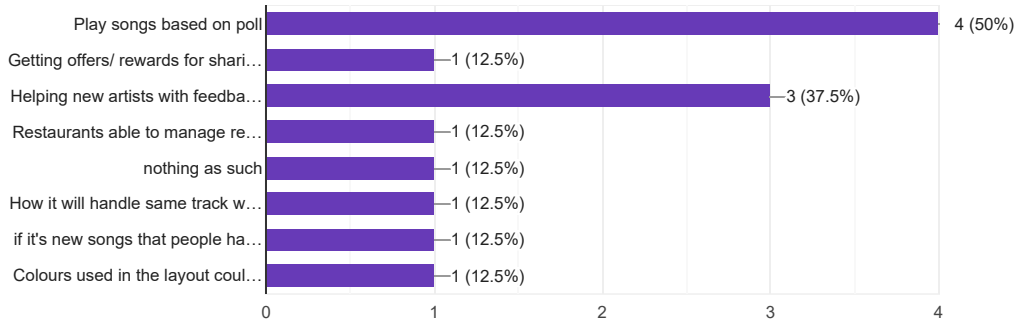
What you liked the most about this application:

11 responses



What you didn't like about this application:

8 responses



Any comments/ suggestions on the user-interface of this application? Or any feature you would like to add or remove from this application?

5 responses

ability to introduce more games

During upload application should list the best possible match

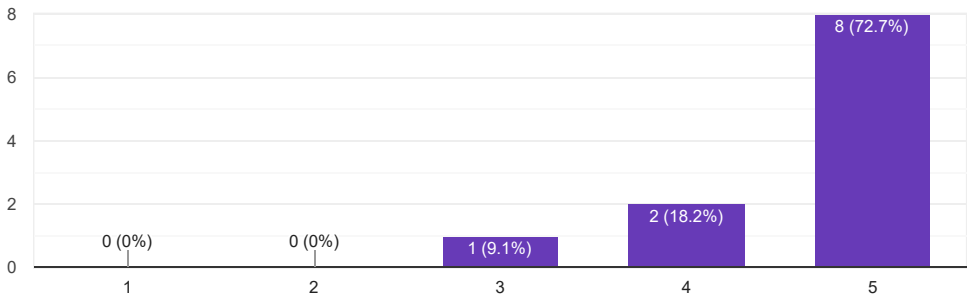
Looks like a great application! Improving business and artist's tuning. I always wanted some application to request at a restaurant which I visited.

A dashboard to display top 10 highly rated songs.

The User-Interface looks good and easy to navigate

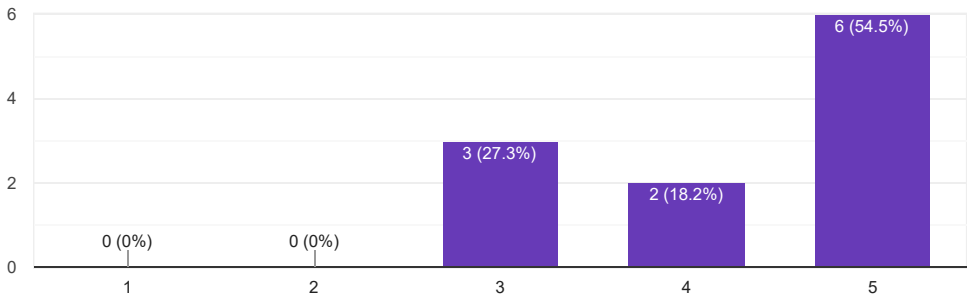
As an end-user, how likely are you to place your song request in a restaurant/disco/gym setting using an application instead of personally requesting?

11 responses



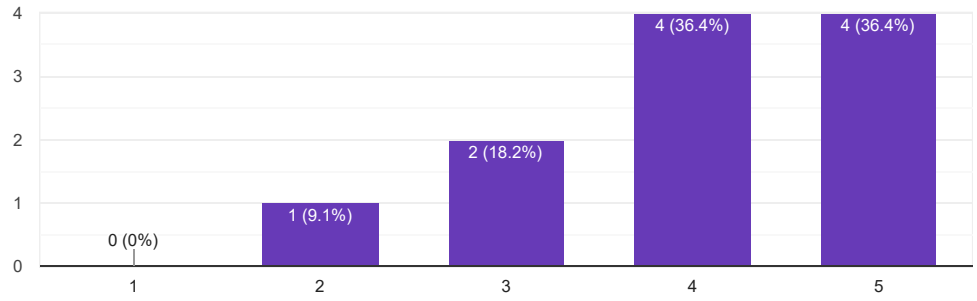
How likely are you to install and use an application which earns you rewards/discounts in return of listening new songs and rating them?

11 responses



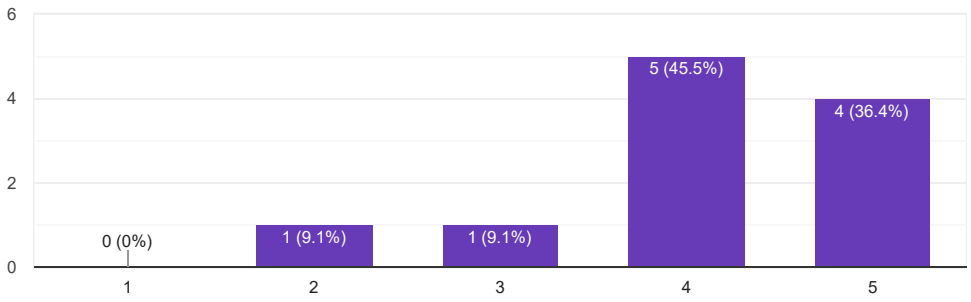
Will this application encourage users to go to restaurant that are using this application?

11 responses



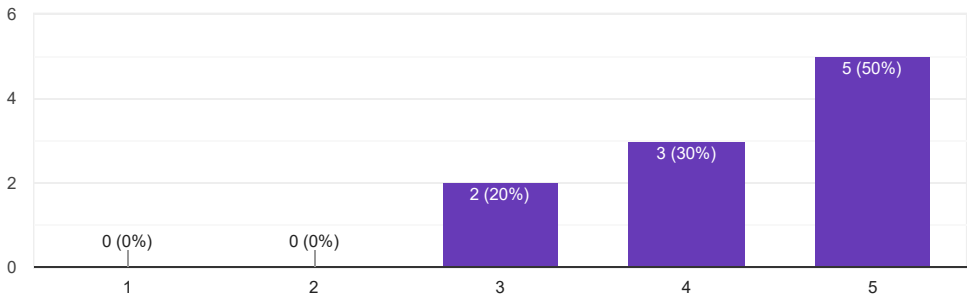
How beneficial could this application be for a new aspiring singer/ band/ musician to get feedback on their work in real-time?

11 responses



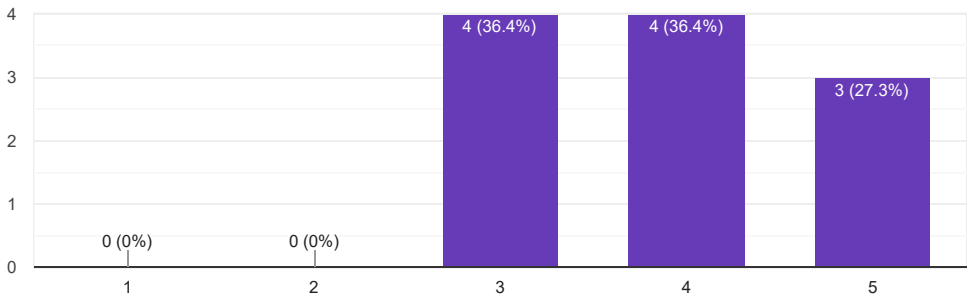
If you are an artist/ aspiring artist, how likely are you to use this application for getting feedback?

10 responses



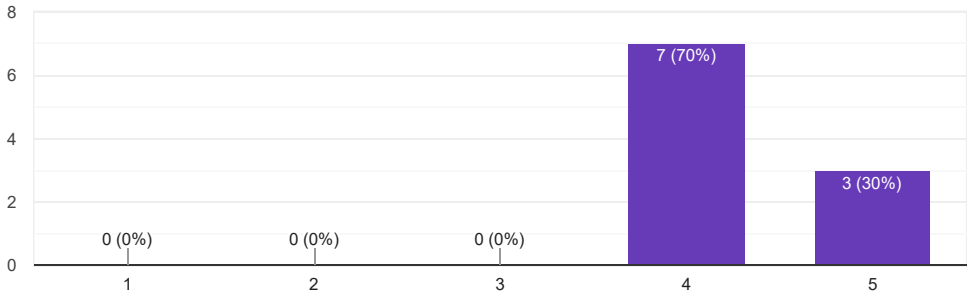
How likely do you think is this application going to help restaurants with their promotions? (Asking for your general opinion)

11 responses



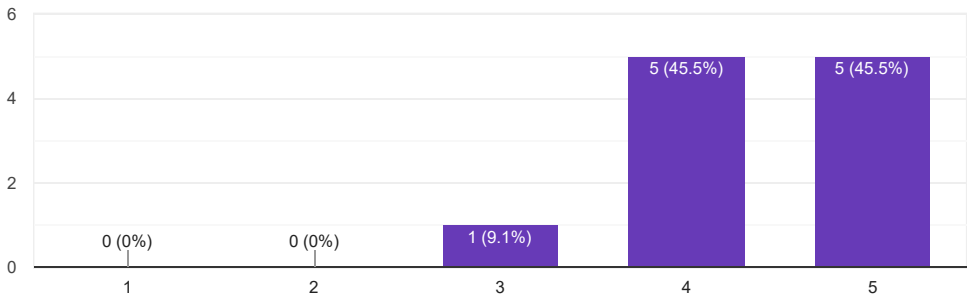
Keeping track of offers, rewards programs, coupons is an overhead for restaurants. Using this application restaurants will have more control on discounts and rewards by limiting (or increasing) the number of polling rounds. How likely are the restaurants going to use such an application which helps them to regulate the offers/ rewards/ coupons? (Asking for your general opinion)

10 responses



How likely it is for a restaurant to use this application to support new artists in return of promotions? (Asking for your general opinion)

11 responses



Appendix H

Ethics Checklist

This section shows the ethics checklist:

This form must be attached to the dissertation as an appendix.

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student	VIKAS KUMAR GUPTA
Academic Year or Project Title	Rewards-based gamified polling system to provide song feedback to artists in a restaurant setting – Play Poll
Supervisor	Dr. Alan Hayes

Does your project involve people for the collection of data other than you and your supervisor(s)? NO

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?* YES / NO

   This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO

   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?* YES / NO

   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO

   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature

of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO

Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Will you be offering any incentive to the participants?* YES / NO

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Will you be in a position of authority or influence over any of your participants?* YES / NO

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Will any of your participants be under the age of 16?* YES / NO

Parental consent is required for participants under the age of 16.

10. *Will any of your participants have an impairment that will limit Their understanding or communication?* YES / NO

Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO

All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Will you have a data management plan for all recorded data?* YES / NO

Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).

Appendix I

Source Code

The section includes the main project files, complete code is uploaded on moodle, and is also available in GitHub repository.

I.1 File: event_repository.dart

```dart
import 'dart:io';
import 'dart:math';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:file_picker/file_picker.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';

import 'package:flutter/services.dart';

import
    'package:flutter_play_poll/domain/events/event_failure.dart';
import 'package:flutter_play_poll/domain/events/event.dart';
import 'package:dartz/dartz.dart';
import
    'package:flutter_play_poll/domain/events/i_event_repository.dart';
import
    'package:flutter_play_poll/infrastructure/auth/firebase_user_mapper.dart';
import
    'package:flutter_play_poll/infrastructure/events/event_dtos.dart';
import
    'package:flutter_play_poll/infrastructure/core/firestore_helpers.dart';
import 'package:injectable/injectable.dart';

@prod
@LazySingleton(as: IEventRepository)
class EventRepository implements IEventRepository {
  final FirebaseFirestore _firestore;
  final FirebaseAuth _firebaseAuth;
  final FirebaseUserMapper _firebaseUserMapper;
  // final FirebaseStorage _firebaseStorage;

  var id;

  var name;

  var emailAddress;

  var photoUrl;

  EventRepository(
    this._firestore,
    this._firebaseAuth,
    this._firebaseUserMapper,
  );

  @override
  Future<Either<EventFailure, Unit>> create(Event event) async {
    try {
      final userDoc = await _firestore.userDocument();
      final eventDto = EventDto.fromDomain(event);
      print('${_firebaseAuth.currentUser} CURRENT USER<------');

      final user = _firebaseAuth.currentUser;
      print('${eventDto.creatorId} CREATOR <------');

      /// Create the user
      await _firestore.collection('users').doc(user!.uid).set({
        'displayName': user.displayName,
        'email': user.email,
        'uid': user.uid,
        'photoUrl': user.photoURL
      });

      /// Create the evnet entry under the logged-in user
      final updatedUserDoc = await _firestore
          .collection('users')
          .doc(user.uid)
          .collection('createdEvents')
          .add(eventDto.toJson());

      /// Create the event entry in the events collection
      await _firestore
          .collection('events')
          .doc(eventDto.id)
          .set(eventDto.toJson());

      return right(unit);
    } on PlatformException catch (e) {
      if (e.message!.contains('PERMISSION_DENIED')) {
        return left(const EventFailure.insufficientPermissions());
      } else {
        return left(const EventFailure.unexpected());
      }
    }
  }

  @override
  Future myEvents() async {
    List itemsList = [];

    try {
```

```
    User? user = FirebaseAuth.instance.currentUser;

    await _firestore
        .collection('users')
        .doc('${user!.uid}')
        .collection('createdEvents')
        .where('creatorId', isEqualTo: user.uid)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        print('${doc.data()} <------REPO------');
        itemsList.add(doc.data());
      });
    });

    return itemsList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future<Either<EventFailure, Unit>> join(Event event) async {
  try {
    final userDoc = await _firestore.userDocument();
    final eventDto = EventDto.fromDomain(event);
    User? user = FirebaseAuth.instance.currentUser;
    print('${eventDto.creatorId} CREATOR <----');

    if (event.creatorId == user!.uid) {
      print('The creator dont need to join their events <----');
      return left(const EventFailure.insufficientPermissions());
    }

    await _firestore.collection('users').doc(user.uid).set({
      'displayName': user.displayName,
      'email': user.email,
      'uid': user.uid,
      'photoUrl': user.photoURL
    });

    _firestore
        .collection('events')
        .doc(eventDto.id)
        .collection('members')
        .doc(user.uid)
        .set(
          EventDto(
            eventId: eventDto.eventId,
            creatorId: event.creatorId,
            name: eventDto.name,
            location: eventDto.location,
          ).toJson(),
        );

    // .set(eventDto.toJson());

    userDoc.joinedEventCollection.doc(eventDto.id).set(EventDto(
          eventId: eventDto.eventId,
          creatorId: event.creatorId,
          name: eventDto.name,
          location: eventDto.location,
        ).toJson());

    return right(unit);
  } on PlatformException catch (e) {
    if (e.message!.contains('PERMISSION_DENIED')) {
      return left(const EventFailure.insufficientPermissions());
    } else {
      return left(const EventFailure.unexpected());
    }
  }
}

@override
Future joinedEvents() async {
  List itemsList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;
    await _firestore
        .collection('users')
        .doc('${user!.uid}')
        .collection('joinedEvents')
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        // print('${doc.data()} <------REPO------');
        itemsList.add(doc.data());
      });
    });
    return itemsList;
  } catch (e) {
```

```
    print(e.toString());
    return null;
  }
}

@override
Future fetchCreatorSongs(String creatorId) async {
  List itemsList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;
    await _firestore
        .collection('users')
        .doc(creatorId)
        .collection('songs')
        .orderBy('votes', descending: true)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        itemsList.add(doc.data());
      });
    });
    return itemsList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future incrementSongVote(
    String currentVoteCount, String songId, String uid) async {
  List updatedCountList = [];
  try {
    String updatedCount = '0';
    updatedCount = (int.parse(currentVoteCount) + 1).toString();
    updatedCountList.add(updatedCount);

    await _firestore
        .collection('users')
        .doc(uid)
        .collection('songs')
        .doc(songId)
        .update({'votes': updatedCount});
    return updatedCountList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future registerVote(String songId, String uid) async {
  List updatedCountList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;
    String updatedCount = '0';

    updatedCountList.add(updatedCount);

    await _firestore
        .collection('users')
        .doc(uid)
        .collection('songs')
        .doc(songId)
        .update({
      'votes': FieldValue.arrayUnion(['${user!.uid}'])
    });
    return updatedCountList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

///
/// ------------------------- RESET SONG VOTE TO ZERO
/// -------------------------
///
@override
Future resetVoteToZero(String songId, String uid) async {
  try {
    User? user = FirebaseAuth.instance.currentUser;
    String updatedCount = '0';

    print('INSIDE resetVoteToZero method <------');
    await _firestore
        .collection('users')
        .doc(uid)
        .collection('songs')
        .doc(songId)
```

```
        .update({'votes': []});
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future updateAppearedInOptionCount(String songId, String
    artistUid) async {
  print('DATABASE <----');

  try {
    _firestore
        .collection('artists')
        .doc(artistUid)
        .collection('songs')
        .doc(songId)
        .update(
      {'appearedInOption': FieldValue.increment(1)},
    );
  } catch (e) {
    print(e.toString());
    return null;
  }
}

///
/// ------------------------------------------------------------------///
/// ---------------------------- CREATE GAME MODE ENTRY
    ----------------------------///
///
    ------------------------------------------------------------------///

@override
Future createGameModeEntry(
    String eventId, String songId, String artistUid) async {
  try {
    User? user = FirebaseAuth.instance.currentUser;

    print('INSIDE createGameModeEntry method <----');
    await _firestore.collection('game_mode').doc().set(
      {
        'eventId': eventId,
        'songId': songId,
        'artistUid': artistUid,
        'yesVotes': [],
        'noVotes': [],
        'winner': '',
        'isSongPlayed': true,
        'dateTimeStamp': DateTime.now()
      },
    );
  } catch (e) {
    print(e.toString());
    return null;
  }
}

///
/// ------------------------------------------------------------------///
/// ---------------------------- REGISTER GAME MODE VOTE
    ----------------------------///
///
    ------------------------------------------------------------------///

@override
Future registerGameModeVote(
    String songId, String artistUid, String voteSmiley) async {
  try {
    User? user = FirebaseAuth.instance.currentUser;

    print('INSIDE registerGameModeVote method <----');

    _firestore
        ///.collection('game_mode')
        .orderBy('dateTimeStamp', descending: true)
        .limit(1)
        .get()
        ///.then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach(
        (doc) {
          if (voteSmiley == 'sentiment_very_dissatisfied') {
            _firestore.collection('game_mode').doc(doc.id).update({
              'noVotes': FieldValue.arrayUnion(['${user!.uid}'])
            });
          } else {
            _firestore.collection('game_mode').doc(doc.id).update({
              'yesVotes': FieldValue.arrayUnion(['${user!.uid}'])
            });
          }
        },
      );
    });
```

```
  } catch (e) {
    print(e.toString());
    return null;
  }
}

///
/// ------------------------------ GAME-MODE WINNER
    ------------------------------///
///

@override
Future decideGameModeWinner(String eventId, String songId) async
    {
  try {
    User? user = FirebaseAuth.instance.currentUser;
    Random random = Random();
    List gameModeFullData = [];
    List yesVotesList = [];
    List noVotesList = [];
    String winnerUid = '';
    List winnerData = [];
    String documentId = '';

    print('INSIDE decideGameModeWinner method <------');

    await _firestore
        .collection('game_mode')
        .where('eventId', isEqualTo: eventId)
        .where('songId', isEqualTo: songId)
        .orderBy('dateTimeStamp', descending: true)
        .limit(1)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach(
        (doc) {
          gameModeFullData.add(doc.data());
          documentId = doc.id;
        },
      );
    });

    print('~~~~> gameModeFullData - ${gameModeFullData}');
    yesVotesList = gameModeFullData[0]['yesVotes'];
    print('~~~~> gameModeFullData YES - '
        '${gameModeFullData[0]['yesVotes']}');
    print('yesVotesList - $yesVotesList');

    noVotesList = gameModeFullData[0]['noVotes'];
    print('~~~~> gameModeFullData NO - '
        '${gameModeFullData[0]['noVotes']}');
    print('noVotesList - $noVotesList');

    if (yesVotesList.length >= noVotesList.length) {
      winnerUid =
          yesVotesList[random.nextInt(yesVotesList.length)];
    } else {
      winnerUid =
          noVotesList[random.nextInt(noVotesList.length)];
      await
          _firestore.collection('game_mode').doc(documentId).update({
        'isSongPlayed': false,
      });
    }
    await
        _firestore.collection('game_mode').doc(documentId).update({
      'winner': winnerUid,
    });

    await _firestore
        .collection('users')
        .where('uid', isEqualTo: winnerUid)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach(
        (doc) {
          winnerData.add(doc.data());
        },
      );
    });

    /// Below code is to add the vote counts to the winnerData
        list
    /// to show it on the game mode page

    /// Zero length check
    if (yesVotesList.length == 0) {
      winnerData.add(0);
    } else {
      winnerData.add(yesVotesList.length);
    }
```

```
    /// Zero length check
    if (noVotesList.length == 0) {
      winnerData.add(0);
    } else {
      winnerData.add(noVotesList.length);
    }

    // return yesVotesList[random.nextInt(yesVotesList.length)];
    return winnerData;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

///
/// ------------------------------ ARTIST REPORT
    ------------------------------///
///
    ------------------------------------------------------------------------
@override
Future artistReport() async {
  List artistData = [];
  List items = [];
  List gameModeDocIdsForLoggedInArtist = [];

  ///------------------------
  List songIdList = [];
  List songNameList = [];
  List timesSongAppearedAsOption = [];
  List yesCountList = [];
  List noCountList = [];
  List isSongPlayedCountList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;

    /// Prepairing the list of Song Ids for the current logged
        in Artist
    await _firestore
        .collection('artists')
        .doc(user!.uid)
        .collection('songs')
        .where('artistUid', isEqualTo: user.uid)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        songIdList.add(doc.id);
        songNameList.add(doc.get('title'.split('.')[0]));
      });
    });

    // print('songIdList --> $songIdList');
    // print('songNameList --> $songNameList');

    /// Iterating for each song in the songIdList and performing
        analysis

    for (var songId in songIdList) {
      await _firestore
          .collection('game_mode')
          .where('songId', isEqualTo: songId)
          /// .get()
          .then((QuerySnapshot querySnapshot) {
        // print(songId + '-' + querySnapshot.size.toString());
        timesSongAppearedAsOption.add(querySnapshot.size.toString());
        ///
        int yesCount = 0;
        int noCount = 0;
        int isSongPlayedCount = 0;
        querySnapshot.docs.forEach((doc) {
          /// FETCHING YES COUNT - HAPPY SMILEY
          yesCount =
              yesCount +
                  int.parse(((doc.get('yesVotes')).length.toString()));

          /// FETCHING NO COUNT - SAD SMILEY
          noCount =
              noCount +
                  int.parse(((doc.get('noVotes')).length.toString()));

          /// FETCHING NUMBER OF TIMES THE SONG GOT PLAYED
          if (doc.get('isSongPlayed') == true) {
            isSongPlayedCount = isSongPlayedCount + 1;
          }
        });
        yesCountList.add(yesCount.toString());
        noCountList.add(noCount.toString());
        isSongPlayedCountList.add(isSongPlayedCount.toString());
      });
    }
```

```
    // print('timesSongAppearedAsOption ---->
        $timesSongAppearedAsOption');
    // print('isSongPlayedCountList ---> $isSongPlayedCountList');
    // print('yesCountList ---> $yesCountList');
    // print('noCountList ---> $noCountList');

    artistData.add(songIdList);
    artistData.add(songNameList);
    artistData.add(timesSongAppearedAsOption);
    artistData.add(isSongPlayedCountList);
    artistData.add(yesCountList);
    artistData.add(noCountList);

    return artistData;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
String getCurrentUserId() {
  User? user = FirebaseAuth.instance.currentUser;
  print('${user!.uid} <---- INSIDE METHOD - getCurrentUserId');
  return user.uid.toString();
}

@override
Future<Either<EventFailure, Unit>> update(Event event) async {
  try {
    final userDoc = await _firestore.userDocument();
    final eventDto = EventDto.fromDomain(event);
    final eventId = event.id.getOrCrash();

    await userDoc.createdEventCollection
        .doc(eventDto.id)
        .update(eventDto.toJson());

    await _firestore
        .collection('events')
        .doc(eventId)
        .update(eventDto.toJson());

    return right(unit);
  } on PlatformException catch (e) {
    if (e.message!.contains('PERMISSION_DENIED')) {
      return left(const EventFailure.insufficientPermissions());
    } else {
      return left(const EventFailure.unexpected());
    }
  }
}

@override
Future allEventsFetched() async {
  List itemsList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;

    await _firestore
        .collection('events')
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        print('${doc.data()} <---------- REPO ---------');
        itemsList.add(doc.data());
      });
    });
    return itemsList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future search(String inputText) async {
  List itemsList = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;

    await _firestore
        .collection('events')
        .where('name', isEqualTo: inputText)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) {
        print('${doc.data()} <---------- REPO ---------');
        itemsList.add(doc.data());
      });
    });
```

```
    return itemsList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future<Either<EventFailure, Unit>> view(Event event) async {
  // TODO: implement view
  throw UnimplementedError();
}

@override
Future<Either<EventFailure, Unit>> unjoin(Event event) async {
  try {
    final userDoc = await _firestore.userDocument();
    final eventId = event.id.getOrCrash();
    final user = _firebaseAuth.currentUser;

    await _firestore
        .collection('users')
        .doc(user!.uid)
        .collection('joinedEvents')
        .where('eventId', isEqualTo: eventId)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((element) {
        print('${element.id} <<<<-UNJOIN_AFTERDELETE----');
        _firestore
            .collection('users')
            .doc(user.uid)
            .collection('joinedEvents')
            .doc(element.id)
            .delete();
      });
    });

    ///
    ///  ------------------------------------------------------------
    /// Delete the members of the deleted event
    ///
    ///  ------------------------------------------------------------
    await _firestore
        .collection('events')
        .doc(eventId)
        .collection('members')
        .doc(user.uid)
        .delete();

    return right(unit);
  } on PlatformException catch (e) {
    if (e.message!.contains('PERMISSION_DENIED')) {
      return left(const EventFailure.insufficientPermissions());
    } else {
      return left(const EventFailure.unexpected());
    }
  }
}

@override
Future<Either<EventFailure, Unit>> delete(Event event) async {
  try {
    final userDoc = await _firestore.userDocument();
    final eventId = event.id.getOrCrash();
    final user = _firebaseAuth.currentUser;
// If a user deleted an event then the event should get deleted
    from:
//                1. events collection
    // events > doc > members
//                2. createdEvents collection for the creator
//                3. from users collection joined events for other
    users
    ///
    ///  ------------------------------------------------------------
    /// 2-Delete the event from createdEvent collection
    ///
    ///  ------------------------------------------------------------
    // await
        userDoc.createdEventCollection.doc(eventId).delete();
    await _firestore
        .collection('users')
        .doc(user!.uid)
        .collection('createdEvents')
        .where('eventId', isEqualTo: eventId)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((element) {
        print('${element.id} <<<<-AFTERDELETE----');
        _firestore
            .collection('users')
            .doc(user.uid)
            .collection('createdEvents')
            .doc(element.id)
```

```
        .delete();
  });
});

///
/// 3-Delete the event from joinedEvent collection for other
    users/members
///

await _firestore
    .collection('users')
    .where('uid', isNotEqualTo: user.uid)
    .get()
    .then((QuerySnapshot querySnapshot) {
  querySnapshot.docs.forEach((element) {
    print('${element.id} <<<<-AFTERJOINED_DELETE---');

    _firestore
        .collection('users')
        .doc(element.id)
        .collection('joinedEvents')
        // .where('eventId', isEqualTo: eventId)
        .doc(eventId)
        .delete();
  });
});

///
/// Delete the members of the deleted event
///

await _firestore
    .collection('events')
    .doc(eventId)
    .collection('members')
    .where('eventId', isEqualTo: eventId)
    .get()
    .then((QuerySnapshot querySnapshot) {
  querySnapshot.docs.forEach((element) {
    print('${element.id} <<<<-AFTERDELETE---');
    _firestore
        .collection('events')
        .doc(eventId)
        .collection('members')
        .doc(element.id)
        .delete();
  });
});

/// Delete the event from the events collection
await _firestore.collection('events').doc(eventId).delete();

return right(unit);
} on PlatformException catch (e) {
if (e.message!.contains('PERMISSION_DENIED')) {
  return left(const EventFailure.insufficientPermissions());
} else {
  return left(const EventFailure.unexpected());
}
}
}
}
```

I.2 File: firebase_storage_repository.dart

```
import 'dart:io';
import 'dart:math';

import 'package:cloud_firestore/cloud_firestore.dart';

import 'package:file_picker/file_picker.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter_play_poll/domain/core/value_objects.dart';
import
    'package:flutter_play_poll/domain/storage/i_storage_repository.dart';
import
    'package:flutter_play_poll/infrastructure/auth/firebase_user_mapper.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:injectable/injectable.dart';

@prod
@LazySingleton(as: IStorageRepository)
class FirebaseStorageRepository implements IStorageRepository {
  final FirebaseAuth _firebaseAuth;
  final GoogleSignIn _googleSignIn;
  final FirebaseUserMapper _firebaseUserMapper;
  final FirebaseFirestore _firestore;
```

```
FirebaseStorageRepository(
  this._firebaseAuth,
  this._googleSignIn,
  this._firebaseUserMapper,
  this._firestore,
);
File? fileObj;
FilePickerResult? result;
UploadTask? uploadFileTask;

@override
Future selectSong() async {
  result = await FilePicker.platform.pickFiles(
    allowMultiple: false,
  );

  if (result == null) return;
  this.fileObj = File(result!.files.single.path.toString());
  return result;
}

@override
Future uploadSong() async {
  // Future<UploadTask?> uploadSong() async {
  // Future<Stream<TaskSnapshot>?> uploadSong() async {
  if (result != null) {
    String fileName = result!.files.first.name;
    String uid = _firebaseAuth.currentUser!.uid;

    String fileType = result!.files.first.extension.toString();

    print('$fileType <--- FILE TYPE');
    // Upload file under signed-in users folder

    // uploadFileTask = FirebaseStorage.instance
    //     .ref('$uid/songs/$fileName')
    //     .putFile(fileObj!);

    // fileObj = File('');
    // fileObj!.delete();
    // fileUploadStatus(uploadFileTask!);

    // uploadFileTask = FirebaseStorage.instance
    //     .ref('$uid/songs/$fileName')
    //     .putFile(fileObj!);

    UniqueId uniqueSongId = UniqueId();

    String songId = uniqueSongId.value.fold((l) =>
        (l.failedValue), (r) => r);

    final snapshot = await (FirebaseStorage.instance
            // .ref('$uid/songs/$fileName')
            .ref('$uid/songs/$songId')
            .putFile(fileObj!))
        .whenComplete(() {});

    final widgetOutput = await
        fileUploadStatus(FirebaseStorage.instance
        // .ref('$uid/songs/$fileName')
        .ref('$uid/songs/$songId')
        .putFile(fileObj!));

    // final snapshot = await (uploadFileTask)!.whenComplete(()
        {});

    final downloadUrl = await snapshot.ref.getDownloadURL();
    print('$downloadUrl <--- DOWNLOAD URL');

    await _firestore
        .collection('users')
        .doc(uid)
        .collection('songs')
        .doc(songId)
        .set({
      'songId': songId,
      'songUrl': downloadUrl,
      'title': fileName,
      'uid': uid,
      'votes': []
    });
    // return uploadFileTask!;
    return widgetOutput;
  } else
    return null;
}

Future<Widget> fileUploadStatus(UploadTask? uploadFileTask)
    async {
  return StreamBuilder<TaskSnapshot>(
    stream: uploadFileTask!.snapshotEvents,
    builder: (context, snapshot) {
      if (snapshot.hasData) {
```

```
        print('inside <------');
        print('${snapshot.data} <------ HAS DATA');
        final fileData = snapshot.data;
        final status =
            ((fileData!.bytesTransferred / fileData.totalBytes)
                * 100)
                .toStringAsFixed(2);
        print('$status <------ STATUS PERCENTAGE');
        return Text(
          '$status % Uploaded',
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.w500,
          ),
        );
      } else {
        print('outside <------');
        return Text(
          'Uploading ... ',
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.w500,
          ),
        );
      }
    },
  );
}

@override
Future uploadArtistSong() async {
  if (result != null) {
    String fileName = result!.files.first.name;
    String uid = _firebaseAuth.currentUser!.uid;
    final user = _firebaseAuth.currentUser!;

    String fileType = result!.files.first.extension.toString();

    print('$fileType <------ FILE TYPE FOR ARTIST');

    UniqueId uniqueSongId = UniqueId();

    String songId = uniqueSongId.value.fold((l) =>
        (l.failedValue), (r) => r);

    final snapshot = await (FirebaseStorage.instance
            .ref('$uid/songs/$songId')
            .putFile(fileObj!))
        .whenComplete(() {});

    final widgetOutput = await fileUploadStatus(
        FirebaseStorage.instance.ref('$uid/songs/$songId').putFile(fileObj!));

    final downloadUrl = await snapshot.ref.getDownloadURL();
    print('$downloadUrl <------ DOWNLOAD URL');

    await _firestore.collection('artists').doc(uid).set({
      'displayName': user.displayName,
      'email': user.email,
      'uid': user.uid,
      'photoUrl': user.photoURL
    });

    await _firestore
        .collection('artists')
        .doc(uid)
        .collection('songs')
        .doc(songId)
        .set({
      'songId': songId,
      'songUrl': downloadUrl,
      'title': fileName,
      'artistUid': uid,
      'votes': [],
      'appearedInOption': 0
    });

    return widgetOutput;
  } else
    return null;
}

@override
Future fetchArtistSongs() async {
  List itemsList = [];
  List items = [];
  List artistUid = [];

  try {
    User? user = FirebaseAuth.instance.currentUser;
    await _firestore
        .collection('artists')
        .get()
        .then((QuerySnapshot querySnapshot) {
```

```
      querySnapshot.docs.forEach((doc) async {
        print('${doc.data()} <------ REPO-ARTISTS-SONGS ------');
        items.add(doc.data());
      });
    });

    for (var item in items) {
      print('$item <------ PRINTING ITEM');
      artistUid.add(item['uid']);
    }

    print('$artistUid <------ ARTIST UID ');

    for (var artist in artistUid) {
      await _firestore
          .collection('artists')
          .doc(artist)
          .collection('songs')
          .get()
          .then(
        (QuerySnapshot _querySnapshot) {
          _querySnapshot.docs.forEach(
            (doc) {
              print('${doc.data()} <------ REPO ------');
              itemsList.add(doc.data());
            },
          );
        },
      );
    }

    print('$itemsList <------ ITEMS RETURNED ------');
    return itemsList;
  } catch (e) {
    print(e.toString());
    return null;
  }
}

@override
Future generateSongOptions(gameModeFullSongList) async {
  List itemsList = [];
  Random random = Random();

  try {
    // print('----------> INSIDE STORAGE REPO
        $gameModeFullSongList ');

      itemsList = List.generate(
          2,
          (_) => gameModeFullSongList[
              random.nextInt(gameModeFullSongList.length)]);

      // for (var song in gameModeFullSongList) {
      //   print('${song['votes'].length} <------ PRINTING ITEM
          Length');
      // }
      print('------> FINAL SONG OPTIONS $itemsList ');

      return itemsList;
    } catch (e) {
      print(e.toString());
      return null;
    }
  }
}
```

I.3 File: event_bloc.dart

```
import 'dart:async';

import 'package:bloc/bloc.dart';
import 'package:flutter_play_poll/domain/auth/i_auth_facade.dart';
import
    'package:flutter_play_poll/domain/events/i_event_repository.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';

part 'event_event.dart';
part 'event_state.dart';
part 'event_bloc.freezed.dart';

@injectable
class EventBloc extends Bloc<EventEvent, EventState> {
  final IEventRepository _eventRepository;

  EventBloc(this._eventRepository) : super(_Initial());

  // static get fetchEventSongs => fetchEventSongs;

  @override
  Stream<EventState> mapEventToState(
```

```
  EventEvent event,
) async* {
  yield* event.map(
    started: (e) async* {
      print('On Main Event page ${e.data}');
      dynamic received = await
          _eventRepository.fetchCreatorSongs(e.data);
      print('$received <---- INSIDE Bloc');
      // yield EventState.initial();
      yield EventState.showFetchedSongs(received);
    },
    displaySongs: (e) async* {
      yield EventState.displayFetchedSongs(e.data);
    },
    incrementVoteCount: (e) async* {
      // print('${e.currentVoteCount} INSIDE Increment <----');

      // dynamic received = await
          _eventRepository.incrementSongVote(
      //     e.currentVoteCount, e.songId, e.uid);

      dynamic received = await
          _eventRepository.registerVote(e.songId, e.uid);
      yield EventState.incrementedVoteCount('$received');
    },
    getSignedInUserEvent: (e) async* {
      dynamic userId = _eventRepository.getCurrentUserId();
      yield EventState.getSignedInUserState(userId.toString());
    },
    gameModeVoteEvent: (e) async* {
      await _eventRepository.registerGameModeVote(
          e.songId, e.artistUid, e.voteSmiley);
      yield EventState.gameModeVoteState();
    },
    updateAppearedInOptionCountEvent: (e) async* {
      dynamic updatedCount =
          _eventRepository.updateAppearedInOptionCount(e.songId,
              e.artistUid);
      yield
          EventState.updateAppearedInOptionCountState(updatedCount);
    },
    showWinnerEvent: (e) async* {
      dynamic selectedWinner =
          await _eventRepository.decideGameModeWinner(e.eventId,
              e.songId);
      yield EventState.showWinnerState(selectedWinner);
    },
    createGameModeEntryEvent: (e) async* {
      await _eventRepository.createGameModeEntry(
          e.eventId, e.songId, e.artistUid);
      yield EventState.createGameModeEntryState();
    },
    votingStartedEvent: (e) async* {
      int value = 10;
      int output = 0;
      Timer votingTimer = Timer.periodic(Duration(seconds: 1),
          (votingTimer) {
        value = value - 1;
        print('FROM TIMER ----> $value');
        print(votingTimer.tick.toString());
        output = value;

        if (value == 0) {
          votingTimer.cancel();
        }
      });

      yield
          EventState.votingStartedState(votingTimer.tick.toString());
    },
  );
 }
}
```

I.4 File: songs_player_bloc.dart

```
import 'dart:async';

import 'package:audioplayers/audioplayers.dart';
import 'package:bloc/bloc.dart';
import
    'package:flutter_play_poll/domain/events/i_event_repository.dart';
import
    'package:flutter_play_poll/domain/storage/i_storage_repository.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';

part 'songs_player_event.dart';
part 'songs_player_state.dart';
part 'songs_player_bloc.freezed.dart';

@injectable
```

```
class SongsPlayerBloc extends Bloc<SongsPlayerEvent,
    SongsPlayerState> {
  final IEventRepository _eventRepository;
  final IStorageRepository _iStorageRepository;
  SongsPlayerBloc(this._eventRepository, this._iStorageRepository)
      : super(_Initial());

  @override
  Stream<SongsPlayerState> mapEventToState(
    SongsPlayerEvent event,
  ) async* {
    yield* event.map(
      started: (e) async* {
        yield SongsPlayerState.initial();
      },
      onAudioPositionChangedEvent: (e) async* {
        yield
            SongsPlayerState.onAudioPositionChangedState(e.position);
      },
      onDurationChangedEvent: (e) async* {
        yield SongsPlayerState.onDurationChangedState(e.duration);
      },
      onPlayerCompletionEvent: (e) async* {
        dynamic received =
            await _eventRepository.resetVoteToZero(e.songId,
                e.uid);
        yield SongsPlayerState.onPlayerCompletionState();
      },
      fetchArtistSongsEvent: (e) async* {
        dynamic received = await
            _iStorageRepository.fetchArtistSongs();

        print('\n HERE \n $received <---- ');
        yield SongsPlayerState.fetchArtistSongsState(received);
      },
      generateOptionsEvent: (e) async* {
        print('\n INSIDE generateOptionsEvent Bloc Event <---- \
            \n');
        dynamic received = await _iStorageRepository
            .generateSongOptions(e.gameModeFullSongList);
        yield SongsPlayerState.generateOptionsState(received);
      },
    );
  }
}
```

I.5 File: constants.dart

```
import 'package:flutter/material.dart';

BoxShadow boxTopShadow = BoxShadow(
  color: Colors.grey.shade200,
  blurRadius: 1.0, // soften the shadow
  spreadRadius: 1.0, //extend the shadow
  offset: Offset(
    -1.0, // Move to right 10  horizontally
    -1.0, // Move to bottom 10 Vertically
  ),
);

BoxShadow boxBotomShadow = BoxShadow(
  color: Colors.grey.shade400,
  blurRadius: 1.0, // soften the shadow
  spreadRadius: 1.0, //extend the shadow
  offset: Offset(
    1.0, // Move to right 10  horizontally
    1.0, // Move to bottom 10 Vertically
  ),
);

Gradient forContainerLinear_backup = LinearGradient(
  begin: Alignment.topLeft,
  end: Alignment.bottomRight,
  // center: Alignment.bottomLeft,
  stops: [0.3, 0.85, 0.95, 1],
  colors: [
    Colors.greenAccent.withOpacity(0.8),
    Colors.orange[100]!.withOpacity(0.6),
    Colors.purple[100]!.withOpacity(0.5),
    Colors.blueAccent[100]!.withOpacity(0.9),
  ],
);

Gradient forContainerLinear = LinearGradient(
  begin: Alignment.topLeft,
  end: Alignment.bottomRight,
  // center: Alignment.bottomLeft,
  stops: [0.01, 0.5, 0.8, 1],
  colors: [
    Colors.blueAccent.withOpacity(0.8),
    Colors.grey[100]!.withOpacity(0.6),
    Colors.blueGrey[100]!.withOpacity(0.5),
    Colors.blue[100]!.withOpacity(0.9),
```

```
  ],
);

Gradient forPageDesignRadial = RadialGradient(
  radius: 0.8,
  center: Alignment.bottomLeft,
  stops: [0.6, 1, 3, 6],
  colors: [
    Colors.orange.withOpacity(1),
    Colors.orange[100]!.withOpacity(0.6),
    Colors.pink[100]!.withOpacity(0.5),
    Colors.white.withOpacity(0.3),
  ],
);
```

I.6 File: constants.dart

```
import 'package:flutter/material.dart';

BoxShadow boxTopShadow = BoxShadow(
  color: Colors.grey.shade200,
  blurRadius: 1.0, // soften the shadow
  spreadRadius: 1.0, //extend the shadow
  offset: Offset(
    -1.0, // Move to right 10  horizontally
    -1.0, // Move to bottom 10 Vertically
  ),
);

BoxShadow boxBotomShadow = BoxShadow(
  color: Colors.grey.shade400,
  blurRadius: 1.0, // soften the shadow
  spreadRadius: 1.0, //extend the shadow
  offset: Offset(
    1.0, // Move to right 10  horizontally
    1.0, // Move to bottom 10 Vertically
  ),
);

Gradient forContainerLinear_backup = LinearGradient(
  begin: Alignment.topLeft,
  end: Alignment.bottomRight,
  // center: Alignment.bottomLeft,
  stops: [0.3, 0.85, 0.95, 1],
  colors: [
    Colors.greenAccent.withOpacity(0.8),
    Colors.orange[100]!.withOpacity(0.6),
    Colors.purple[100]!.withOpacity(0.5),
    Colors.blueAccent[100]!.withOpacity(0.9),
  ],
);

Gradient forContainerLinear = LinearGradient(
  begin: Alignment.topLeft,
  end: Alignment.bottomRight,
  // center: Alignment.bottomLeft,
  stops: [0.01, 0.5, 0.8, 1],
  colors: [
    Colors.blueAccent.withOpacity(0.8),
    Colors.grey[100]!.withOpacity(0.6),
    Colors.blueGrey[100]!.withOpacity(0.5),
    Colors.blue[100]!.withOpacity(0.9),
  ],
);

Gradient forPageDesignRadial = RadialGradient(
  radius: 0.8,
  center: Alignment.bottomLeft,
  stops: [0.6, 1, 3, 6],
  colors: [
    Colors.orange.withOpacity(1),
    Colors.orange[100]!.withOpacity(0.6),
    Colors.pink[100]!.withOpacity(0.5),
    Colors.white.withOpacity(0.3),
  ],
);
```

I.7 File: user$_{h}ome_{p}age.dart$

```
import 'package:auto_route/auto_route.dart';

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_play_poll/application/auth/auth_bloc.dart';
import
    'package:flutter_play_poll/application/create_event/create_bloc.dart';

import 'package:flutter_play_poll/application/home/home_bloc.dart';
import
    'package:flutter_play_poll/application/my_events/my_events_bloc.dart
```

```
import
    'package:flutter_play_poll/application/search_event/search_event_bloc.dart';
import
    'package:flutter_play_poll/application/view_report/view_report_bloc.dart';
import 'package:flutter_play_poll/constants.dart';
import 'package:flutter_play_poll/injection.dart';
import
    'package:flutter_play_poll/presentation/my_events/my_events_page.dart';

import
    'package:flutter_play_poll/presentation/routes/router.gr.dart';
import
    'package:flutter_play_poll/presentation/upload_event/upload_event_page.dart';

class UserHomePage extends StatelessWidget {
  const UserHomePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    double height = MediaQuery.of(context).size.height;
    double width = MediaQuery.of(context).size.width;

    return MultiBlocListener(
      listeners: [
        // BlocListener<CreateBloc, CreateState>(
        //   listener: (context, state) {
        //     state.map(
        //       initial: (_) {
        //         print('Started Event fired');
        //       },
        //       returnToHomePage: (_) {},
        //       onHomePageFromCreatePage: (_) {},
        //       eventNameValidated: (_) {},
        //       eventCreated: (_) {},
        //       eventLocationValidated: (_) {},
        //       eventCreationFailed: (_) {},
        //     );
        //   },
        // ),
        BlocListener<HomeBloc, HomeState>(
          listener: (context, state) {
            state.map(
              onHomePageState: (_) {},
              navigatedToCreateEventPage: (_) {
                print(
                    '3-HomeBloc Listener responds to the added
                        event and yields state =
                        navigatedToCreateEventPage ');
                print(
                    '4-Another event = onCreateEventPage added to
                        the HomeBloc');
                print(' 5-Navigating HomePage --> '
                    'CreateEventPage');
                context.read<HomeBloc>().add(HomeEvent.onCreateEventPage());
                AutoRouter.of(context).navigate(CreateEventRoute());
              },
              navigatedToSearchEventPage: (_) {
                context
                    .read<HomeBloc>()
                    .add(HomeEvent.onSearchEventsPageEvent());
                context.read<SearchEventBloc>().add(SearchEventEvent.started());
                AutoRouter.of(context).navigate(SearchEventRoute());
              },
              navigatedToMyEventPage: (_) {
                context.read<HomeBloc>().add(HomeEvent.onMyEventsPageEvent());
                //
                    context.read<MyEventsBloc>().add(MyEventsEvent.onMyEventPage()
                AutoRouter.of(context).navigate(MyEventsRoute());
              },
              navigatedToJoinedEventPage: (_) {
                context
                    .read<HomeBloc>()
                    .add(HomeEvent.onJoinedEventsPageEvent());
                AutoRouter.of(context).navigate(JoinedEventsRoute());
              },
              onCreateEventPage: (_) {},
              onMyEventsPage: (_) {},
              onSearchEventsPage: (_) {},
              onJoinedEventPage: (_) {},
              navigatedToUploadEventPage: (_) {
                context.read<HomeBloc>().add(HomeEvent.onUploadPageEvent());
                AutoRouter.of(context).navigate(UploadEventRoute());
              },
              onUploadEventPage: (_) {},
              navigatedToUploadArtistEventPage: (_) {
                context
                    .read<HomeBloc>()
                    .add(HomeEvent.onUploadArtistPageEvent());
                AutoRouter.of(context).navigate(UploadArtistRoute());
              },
              onUploadArtistEventPage: (_) {
                context
                    .read<HomeBloc>()
                    .add(HomeEvent.onUploadArtistPageEvent());
```

```
          AutoRouter.of(context).navigate(UploadArtistRoute());
        },
        navigatedToViewReportEventPage: (_) {
          //
              context.read<HomeBloc>().add(HomeEvent.onViewReportPageEvent());
          context.read<ViewReportBloc>().add(ViewReportEvent.started());
          context
              .read<ViewReportBloc>()
              .add(ViewReportEvent.showArtistReportEvent());
          AutoRouter.of(context).navigate(ViewReportRoute());
        },
        onViewReportEventPage: (_) {
          context.read<HomeBloc>().add(HomeEvent.onViewReportPageEvent());
          //
              AutoRouter.of(context).navigate(ViewReportRoute());
        },
      );
    },
  ),
  BlocListener<AuthBloc, AuthState>(
    listener: (context, state) {
      state.map(
        initial: (_) {},
        authenticated: (_) {},
        unauthenticated: (_) {
          print('HERE-2 <------');
          context.replaceRoute(SignInRoute());
        },
      );
    },
  ),
],
child: Scaffold(
  appBar: AppBar(
    automaticallyImplyLeading: false,
    title: Text('Play Poll'),
    centerTitle: true,
    actions: [
      IconButton(
          onPressed: () {
            context.read<AuthBloc>().add(const
                AuthEvent.signedOut());
          },
          icon: Icon(Icons.logout))
    ],
  ),
  body: Container(
    decoration: BoxDecoration(gradient: forPageDesignRadial),
    child: ListView(
      children: [
        SizedBox(height: height / 200),Padding(
          padding: const EdgeInsets.all(8.0),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
              context.read<AuthBloc>().state.maybeMap(
                authenticated: (_) {
                  return CircleAvatar(
                    radius: 40,
                    backgroundImage: NetworkImage(
                      _.user.photoUrl.getOrElse(''),
                    ),
                  );
                },
                orElse: () {
                  return Text('User Not Found');
                },
              ),
              SizedBox(width: width / 30),
              context.read<AuthBloc>().state.maybeMap(
                authenticated: (_) {
                  return Align(
                    alignment: Alignment.centerLeft,
                    child: Text(
                      'Welcome
                          \n${_.user.name.getOrElse('')},',
                      style: TextStyle(
                          fontSize: 28,
                          fontWeight: FontWeight.w500,
                          color:
                              Colors.black87.withOpacity(0.65)),
                    ),
                  );
                },
                orElse: () {
                  return Text('User Not Found');
                },
              ),
            ],
          ),
        ),
        SizedBox(height: height / 80),
        Padding(
```

```
        padding: const EdgeInsets.all(8.0),
        child: Align(
          alignment: Alignment.centerLeft,
          child: Text(
            'what would you like to do today?',
            style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.w500,
                color: Colors.black54),
          ),
        ),
      ),
      SizedBox(height: height / 40),
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          children: [
            Expanded(
              child: BlocBuilder<HomeBloc, HomeState>(
                builder: (context, state) {
                  return GestureDetector(
                    onTap: () {
                      print(
                          '1-Create Event Pressed on
                              HomePage<--- $context');
                      print(
                          '2-Event added to the HomeBloc =
                              createEventClicked <---');
                      context
                          .read<HomeBloc>()
                          .add(HomeEvent.createEventClicked());
                    },
                    child: Container(
                      height: height / 5,
                      width: width / 2,
                      decoration: BoxDecoration(
                        color: Colors.white,
                        // color:
                            Colors.blueGrey.withOpacity(0.3),
                        // border: Border.all(color:
                            Colors.green, width: 2),
                        borderRadius:
                            BorderRadius.circular(20),
                        // gradient: forContainerLinear,
                        boxShadow: [boxTopShadow,
                            boxBotomShadow],
                      ),
                      child: Column(
                        mainAxisAlignment:
                            MainAxisAlignment.center,
                        children: [
                          Icon(
                            Icons.add_circle_outline_outlined,
                            size: 60,
                            color: Colors.green,
                          ),
                          SizedBox(height: height / 20),
                          Text(
                            'Create Event',
                            style: TextStyle(
                                fontSize: 18,
                                fontWeight:
                                    FontWeight.w500),
                          ),
                        ],
                      ),
                    );
                  },
                ),
              ),
              SizedBox(width: width / 20),
              Expanded(
                child: GestureDetector(
                  onTap: () {
                    print('1-Search Event Pressed on
                        HomePage<--- ');
                    print(
                        '2-Event added to the HomeBloc =
                            searchEventClicked <---');
                    context
                        .read<HomeBloc>()
                        .add(HomeEvent.searchEventClicked());
                  },
                  child: Container(
                    height: height / 5,
                    width: width / 2,
                    decoration: BoxDecoration(
                      color: Colors.white,
                      // border:
                      //     Border.all(color:
                          Colors.blueAccent, width: 2),
                      borderRadius:
                          BorderRadius.circular(20),
```

```
                boxShadow: [boxTopShadow,
                    boxBotomShadow],
              ),
              child: Column(
                mainAxisAlignment:
                    MainAxisAlignment.center,
                children: [
                  Icon(
                    Icons.search_outlined,
                    size: 60,
                    color: Colors.blueAccent,
                  ),
                  SizedBox(height: height / 20),
                  Text(
                    'Search␣Event',
                    style: TextStyle(
                        fontSize: 18, fontWeight:
                            FontWeight.w500),
                  )
                ],
              ),
            ),
          ),
        ),
      ],
    ),
  ),
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: Row(
      children: [
        Expanded(
          child: BlocBuilder<HomeBloc, HomeState>(
            builder: (context, state) {
              return GestureDetector(
                onTap: () {
                  print('1-My␣Event␣Pressed␣on␣
                      HomePage<----␣');
                  print(
                      '2-Event␣added␣to␣the␣HomeBloc␣=␣
                          myEventClicked␣<----');
                  context
                      .read<HomeBloc>()
                      .add(HomeEvent.myEventClicked());

                  // context
                  //     .read<MyEventsBloc>()
                  //     .add(MyEventsEvent.started());
                },
                child: Container(
                  height: height / 5,
                  width: width / 2,
                  decoration: BoxDecoration(
                    color: Colors.white,
                    // color: Colors.lightGreen,
                    // border:
                    //     Border.all(color:
                        Colors.green, width: 2),
                    borderRadius:
                        BorderRadius.circular(20),
                    boxShadow: [boxTopShadow,
                        boxBotomShadow],
                  ),
                  child: Column(
                    mainAxisAlignment:
                        MainAxisAlignment.center,
                    children: [
                      Icon(
                        Icons.today_outlined,
                        size: 60,
                        color: Colors.blueGrey,
                      ),
                      SizedBox(height: height / 20),
                      Text(
                        'My␣Events',
                        style: TextStyle(
                            fontSize: 18,
                            fontWeight:
                                FontWeight.w500),
                      ),
                    ],
                  ),
                ),
              );
            },
          ),
        ),
        SizedBox(width: width / 20),
        Expanded(
          child: GestureDetector(
            onTap: () {
              context
                  .read<HomeBloc>()
                  .add(HomeEvent.joinedEventClicked());
```

```
            },
            child: Container(
              height: height / 5,
              width: width / 2,
              decoration: BoxDecoration(
                color: Colors.white,
                // border:
                // Border.all(color:
                    Colors.blueAccent, width: 2),
                borderRadius:
                    BorderRadius.circular(20),
                boxShadow: [boxTopShadow,
                    boxBotomShadow],
              ),
              child: Column(
                mainAxisAlignment:
                    MainAxisAlignment.center,
                children: [
                  Icon(
                    Icons.event_available_outlined,
                    size: 60,
                    color: Colors.blueAccent,
                  ),
                  SizedBox(height: height / 20),
                  Text(
                    'Joined Events',
                    style: TextStyle(
                        fontSize: 18, fontWeight:
                            FontWeight.w500),
                  )
                ],
              ),
            ),
          ),
        ),
      ],
    ),
  ),
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: Row(
      children: [
        Expanded(
          child: BlocBuilder<HomeBloc, HomeState>(
            builder: (context, state) {
              return GestureDetector(
                onTap: () {
                  print('1-My Event Pressed on
                      HomePage<---- ');
                  print(
                      '2-Event added to the HomeBloc =
                          uploadEventClicked <----');
                  context
                      .read<HomeBloc>()
                      .add(HomeEvent.uploadEventClicked());

                  // context
                  //     .read<MyEventsBloc>()
                  //     .add(MyEventsEvent.started());
                },
                child: Container(
                  height: height / 5,
                  width: width / 2,
                  decoration: BoxDecoration(
                    color: Colors.white,
                    // color: Colors.lightGreen,
                    // border:
                    //     Border.all(color:
                        Colors.green, width: 2),
                    borderRadius:
                        BorderRadius.circular(20),
                    boxShadow: [boxTopShadow,
                        boxBotomShadow],
                  ),
                  child: Column(
                    mainAxisAlignment:
                        MainAxisAlignment.center,
                    children: [
                      Icon(
                        Icons.my_library_music_outlined,
                        size: 60,
                        color: Colors.deepPurpleAccent,
                      ),
                      SizedBox(height: height / 20),
                      Text(
                        'Event Songs',
                        style: TextStyle(
                            fontSize: 18,
                            fontWeight:
                                FontWeight.w500),
                      ),
                    ],
                  ),
                ),
              ),
```

```
        );
      },
    ),
  ),
  SizedBox(width: width / 20),
  Expanded(
    child: GestureDetector(
      onTap: () {
        context
            .read<HomeBloc>()
            .add(HomeEvent.viewReportEventClicked());
      },
      child: Container(
        height: height / 5,
        width: width / 2,
        decoration: BoxDecoration(
          color: Colors.white,
          // border:
          // Border.all(color:
              Colors.blueAccent, width: 2),
          borderRadius:
              BorderRadius.circular(20),
          boxShadow: [boxTopShadow,
              boxBotomShadow],
        ),
        child: Column(
          mainAxisAlignment:
              MainAxisAlignment.center,
          children: [
            Icon(
              Icons.stacked_bar_chart_outlined,
              size: 60,
              color: Colors.blueAccent,
            ),
            SizedBox(height: height / 20),
            Text(
              'View Report',
              style: TextStyle(
                  fontSize: 18, fontWeight:
                      FontWeight.w500),
            )
          ],
        ),
      ),
    ),
  ),
],
  ),
),
Padding(
  padding: const EdgeInsets.all(8.0),
  child: Row(
    children: [
      Expanded(
        child: BlocBuilder<HomeBloc, HomeState>(
          builder: (context, state) {
            return GestureDetector(
              onTap: () {
                print(
                    '1-uploadArtistEventClicked
                        Pressed on HomePage<---- ');
                print(
                    '2-Event added to the HomeBloc =
                        uploadArtistEventClicked
                        <---- ');
                context
                    .read<HomeBloc>()
                    .add(HomeEvent.uploadArtistEventClicked());

                // context
                //     .read<MyEventsBloc>()
                //     .add(MyEventsEvent.started());
              },
              child: Container(
                height: height / 5,
                width: width / 2,
                decoration: BoxDecoration(
                  color: Colors.white,
                  // color: Colors.lightGreen,
                  // border:
                  //     Border.all(color:
                      Colors.green, width: 2),
                  borderRadius:
                      BorderRadius.circular(20),
                  boxShadow: [boxTopShadow,
                      boxBotomShadow],
                ),
                child: Column(
                  mainAxisAlignment:
                      MainAxisAlignment.center,
                  children: [
                    Icon(
                      Icons.my_library_music_outlined,
                      size: 60,
```

```
                                color: Colors.deepOrange,
                              ),
                              SizedBox(height: height / 20),
                              Text(
                                'Upload Artist Songs',
                                style: TextStyle(
                                    fontSize: 18,
                                    fontWeight:
                                        FontWeight.w500),
                              ),
                            ],
                          ),
                        ),
                      );
                    },
                  ),
                ),
                SizedBox(width: width / 20),
              ],
            ),
          ),
        ),
      ],
    ),
  ),
),
);
}
}
```

I.8 File: create$_{e}$vent$_{p}$age.dart

```
import 'package:auto_route/auto_route.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
    'package:flutter_play_poll/application/create_event/create_bloc.dart';

import 'package:flutter_play_poll/application/home/home_bloc.dart';
import
    'package:flutter_play_poll/domain/events/value_objects.dart';

class CreateEventPage extends StatelessWidget {
  const CreateEventPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiBlocListener(
      listeners: [
        BlocListener<CreateBloc, CreateState>(
          listener: (context, state) {
            state.map(
              initial: (_) {},
              returnToHomePage: (_) {
                print(
                    '8-CreateBloc Listener responds to the added
                        event cancelButtonClicked and yields state
                        = returnToHomePage ');
                print('inside listener returntohomepage <----
                    $context');
                print(
                    '9-Another event = onHomePageEvent added to
                        the HomeBloc');
                context.read<HomeBloc>()..add(HomeEvent.onHomePageEvent());
                print('10-Another event = onHomePage added to the
                    CreateBloc');
                context.read<CreateBloc>().add(CreateEvent.onHomePage());
              },
              onHomePageFromCreatePage: (_) {},
              eventNameValidated: (_) {},
              eventCreated: (_) {
                print('Event Successfully Created');
                final snackBar = SnackBar(
                  content: Text(
                    'Event Successfully Created Under \n My
                        Events',
                    textAlign: TextAlign.center,
                    style: TextStyle(fontSize: 20),
                  ),
                  backgroundColor: Colors.green,
                );
                ScaffoldMessenger.of(context).showSnackBar(snackBar);
                context
                    .read<CreateBloc>()
                    .add(CreateEvent.cancelButtonClicked());
              },
              eventLocationValidated: (_) {},
              eventCreationFailed: (_) {
                print('FAILED <----');
              },
            );
          },
        ),
```

```
  BlocListener<HomeBloc, HomeState>(listener: (context,
      state) {
    state.map(
      onHomePageState: (_) {
        print('After␣9—Popping␣page␣to␣go␣back␣to␣HomePage');
        AutoRouter.of(context).pop();
      },
      navigatedToCreateEventPage: (_) {},
      onCreateEventPage: (_) {},
      navigatedToSearchEventPage: (_) {},
      navigatedToMyEventPage: (_) {},
      navigatedToJoinedEventPage: (_) {},
      onMyEventsPage: (_) {},
      onSearchEventsPage: (_) {},
      onJoinedEventPage: (_) {},
      navigatedToUploadEventPage: (_) {},
      onUploadEventPage: (_) {},
      navigatedToUploadArtistEventPage: (_) {},
      onUploadArtistEventPage: (_) {},
      navigatedToViewReportEventPage: (_) {},
      onViewReportEventPage: (_) {},
    );
  })
],
child: Scaffold(
  appBar: AppBar(
    title: Text('Create␣Event'),
  ),
  body: BlocBuilder<HomeBloc, HomeState>(
    builder: (context, state) {
      return Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
          child: ListView(
            shrinkWrap: true,
            children: [
              EventNameFieldWidget(),
              LocationFieldWidget(),
              SizedBox(height: 20),
              Row(
                mainAxisAlignment:
                    MainAxisAlignment.spaceBetween,
                children: [
                  AddImageButtonWidget(),
                  AddPeopleButtonWidget(),
                ],
              ),
              SizedBox(height: 20),
              // SongsRepoButtonWidget(),
              SizedBox(height: 20),
              Row(
                children: [
                  Expanded(
                    child: ElevatedButton(
                      onPressed: () {
                        context
                            .read<CreateBloc>()
                            .add(CreateEvent.create());
                      },
                      child: Text('Create'),
                    ),
                  ),
                  SizedBox(width: 20),
                  Expanded(
                    child: OutlinedButton(
                      onPressed: () {
                        print(
                            '6—Cancel␣Button␣Pressed␣on␣
                                CreateEventPage');
                        print(
                            '7—Event␣added␣to␣the␣CreateBloc␣
                                =␣cancelButtonClicked␣
                                <———');
                        context
                            .read<CreateBloc>()
                            .add(CreateEvent.cancelButtonClicked());
                      },
                      child: Text('Cancel'),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      );
    },
  ),
);
  }
}

class EventNameFieldWidget extends StatelessWidget {
```

```
  Name name = Name('');
  EventNameFieldWidget({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return TextFormField(
      onChanged: (value) {
        name = Name(value);
        context.read<CreateBloc>().add(CreateEvent.validateEventName(name));
      },
      validator: (_) {
        context.read<CreateBloc>().add(CreateEvent.validateEventName(name));
        if (name.value.toString().length <= 2) {
          return 'Too Short';
        }
      },
      cursorColor: Colors.blueGrey,
      decoration: InputDecoration(
        labelText: "Event Name",
        labelStyle: TextStyle(color: Colors.blueGrey[600]),
        border: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
        focusedBorder: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
        enabledBorder: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
      ),
    );
  }
}

class LocationFieldWidget extends StatelessWidget {
  Location location = Location('');
  LocationFieldWidget({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return TextFormField(
      onChanged: (value) {
        location = Location(value);
        context
            .read<CreateBloc>()
            .add(CreateEvent.validateEventLocation(location));
        // location = Location('');
      },
      cursorColor: Colors.blueGrey,
      decoration: InputDecoration(
        labelText: "Location",
        labelStyle: TextStyle(color: Colors.blueGrey[600]),
        border: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
        focusedBorder: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
        enabledBorder: UnderlineInputBorder(
          borderSide: BorderSide(color: Colors.blueGrey, width: 2),
        ),
      ),
    );
  }
}

class AddImageButtonWidget extends StatelessWidget {
  const AddImageButtonWidget({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      child: OutlinedButton(
        style: OutlinedButton.styleFrom(
          shape: StadiumBorder(),
          side: BorderSide(width: 2, color: Colors.blueGrey[600]!),
        ),
        onPressed: () {},
        child: Padding(
          padding: const EdgeInsets.all(12.0),
          child: Row(
            children: [
              Icon(Icons.add_circle_outlined),
              Text(
                'Add Image',
                style: TextStyle(color: Colors.blueGrey[600],
                    fontSize: 16),
              ),
            ],
          ),
        ),
```

```
      ),
    );
  }
}

class AddPeopleButtonWidget extends StatelessWidget {
  const AddPeopleButtonWidget({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      child: OutlinedButton(
        style: OutlinedButton.styleFrom(
          shape: StadiumBorder(),
          side: BorderSide(width: 2, color: Colors.blueGrey[600]!),
        ),
        onPressed: () {},
        child: Padding(
          padding: const EdgeInsets.all(12.0),
          child: Row(
            children: [
              Icon(Icons.add_circle_outlined),
              Text(
                'Add People',
                style: TextStyle(color: Colors.blueGrey[600],
                    fontSize: 16),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

class SongsRepoButtonWidget extends StatelessWidget {
  const SongsRepoButtonWidget({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      child: OutlinedButton(
        style: OutlinedButton.styleFrom(
          shape: StadiumBorder(),
          side: BorderSide(width: 2, color: Colors.blueGrey[600]!),
        ),
        onPressed: () {},
        child: Padding(
          padding: const EdgeInsets.all(12.0),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(Icons.music_note),
              Text(
                'Select Default Songs Repository',
                style: TextStyle(color: Colors.blueGrey[600],
                    fontSize: 16),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

I.9 File: my_events_page.dart

```
import 'package:auto_route/auto_route.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_play_poll/application/home/home_bloc.dart';
import
    'package:flutter_play_poll/application/my_events/my_events_bloc.dart';
import 'package:flutter_play_poll/domain/core/value_objects.dart';
import 'package:flutter_play_poll/domain/events/event.dart';
import
    'package:flutter_play_poll/domain/events/value_objects.dart';

class MyEventsPage extends StatelessWidget {
  const MyEventsPage({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    double height = MediaQuery.of(context).size.height;
    double width = MediaQuery.of(context).size.width;
```

```
TextEditingController _eventNameController =
    TextEditingController();
TextEditingController _eventLocationController =
    TextEditingController();
return MultiBlocListener(
  listeners: [
    BlocListener<MyEventsBloc, MyEventsState>(listener:
        (context, state) {
      print('HERE1 <---- ');
      state.map(
          initial: (_) {},
          eventViewed: (_) {},
          noEventsCreated: (_) {},
          showEventsCreated: (allEvents) {},
          onMyEventsPageState: (_) {},
          deletedEventState: (_) {},
          deletedFailedState: (_) {},
          updatedEventState: (_) {},
          updatedFailedState: (_) {});
    }),
    BlocListener<HomeBloc, HomeState>(
      listener: (context, state) {
        print('HERE2 <---- ');
        state.map(
          navigatedToCreateEventPage: (_) {},
          navigatedToSearchEventPage: (_) {},
          navigatedToMyEventPage: (_) {},
          navigatedToJoinedEventPage: (_) {},
          onCreateEventPage: (_) {},
          onHomePageState: (_) {},
          onMyEventsPage: (_) {},
          onSearchEventsPage: (_) {},
          onJoinedEventPage: (_) {},
          navigatedToUploadEventPage: (_) {},
          onUploadEventPage: (_) {},
          navigatedToUploadArtistEventPage: (_) {},
          onUploadArtistEventPage: (_) {},
          navigatedToViewReportEventPage: (_) {},
          onViewReportEventPage: (_) {},
        );
      },
    ),
  ],
  child: Scaffold(
    appBar: AppBar(
      title: Text('My Events'),
    ),
    body: BlocBuilder<MyEventsBloc, MyEventsState>(
      builder: (context, state) {
        context.read<MyEventsBloc>().add(MyEventsEvent.started());
        print('Printing CONTEXT ');
        return context.read<MyEventsBloc>().state.maybeMap(
          showEventsCreated: (received) {
            if (received.data.length > 0) {
              print(received);
              return ListView.builder(
                itemCount: received.data.length,
                itemBuilder: (context, index) {
                  return Card(
                    child: ListTile(
                      title:
                          Text('${received.data[index]['name']}'),
                      subtitle:
                          Text('${received.data[index]['location']}'),
                      leading: Icon(
                        Icons.date_range_outlined,
                        color: Colors.blue,
                        size: 40,
                      ),
                      trailing: IconButton(
                          onPressed: () {
                            showDialog(
                                context: context,
                                builder: (context) {
                                  return AlertDialog(
                                    scrollable: true,
                                    title: Text(
                                        'Update: 
                                            ${received.data[index]['name']} 
                                            event'),
                                    content:
                                        SingleChildScrollView(
                                      child: Container(
                                        height: height / 0.5,
                                        width: width,
                                        child: ListView(
                                          shrinkWrap: true,
                                          children: [
                                            TextFormField(
                                              controller:
                                                  _eventNameController,
                                              decoration:
                                                  InputDecoration(
                                                helperText:
```

```
                'Current␣
                    Name:␣
                    ${received.data[index]['name']}',
              labelText:
                  'Name',
            ),
          ),
          TextFormField(
            controller:
                _eventLocationController,
            decoration:
                InputDecoration(
            helperText:
                'Current␣
                    Location:␣
                    ${received.data[index]['location']}',
            labelText:
                'Location',
            ),
          ),
        ],
      ),
    ),
  ),
  actions: [
    OutlinedButton(
      onPressed: () {
        print(
            '${received.data[index]}␣
               <———RECEIVED———');
        context.read<MyEventsBloc>().add(
              MyEventsEvent.deleteEvent(
              Event(
                name:
                    Name(received
                    .data[index]['name']),
                eventId:
                    UniqueId
                    .fromUniqueString(
                        received.data[
                                index]
                            ['eventId']),
                creatorId:
                    received.data[index]
                        ['creatorId'],
                id: UniqueId
                    .fromUniqueString(
                        received.data[
                                index]
                            ['eventId']),
                location:
                    Location(
                    received.data[index]
                        ['location']),
              ),
            ));

        AutoRouter.of(context).pop();
      },
      child: Text(
        'Delete',
        style:
            TextStyle(color:
                Colors.red),
      ),
    ),
    ElevatedButton(
      onPressed: () {
        print('UPDATE');
        context.read<MyEventsBloc>().add(
              MyEventsEvent.updateEvent(
             Event(
               name: Name(
                    _eventNameController
                        .text),
               eventId:
                    UniqueId
                    .fromUniqueString(
                        received.data[
                                index]
                            ['eventId']),
               creatorId:
                    received.data[index]
                        ['creatorId'],
               id: UniqueId
                    .fromUniqueString(
                        received.data[
                                index]
                            ['eventId']),
               location:
                    Location(
                    _eventLocationController
                        .text),
             ),
```

```
                                    ));
                                AutoRouter.of(context).pop();
                              },
                              child: Text('Update'),
                            ),
                            TextButton(
                              onPressed: () {
                                AutoRouter.of(context).pop();
                              },
                              child: Text('Cancel'),
                            ),
                          ],
                        );
                      });
                },
                icon: Icon(
                  Icons.edit,
                  color: Colors.redAccent,
                  size: 30,
                )),
            onTap: () {
              print(received.data[index]['eventId']);
            },
          ),
        );
      },
    );
  } else {
    return Center(
        child: Text(
      'No Events Created',
      style: TextStyle(fontSize: 20),
    ));
  }
},
orElse: () {
  return Center(child: CircularProgressIndicator());
},
);
},
),
),
);
}
}
```

I.10 File: upload_event_page.dart

```
import 'dart:io';

import 'package:auto_route/annotations.dart';
import 'package:auto_route/auto_route.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
    'package:flutter_play_poll/application/upload_event/upload_event_bloc.dart';

class UploadEventPage extends StatelessWidget {
  const UploadEventPage({Key? key}) : super(key: key);
  // File? fileObj;
  // String? fileName = 'No Files Selected';
  // String? fileSize = '0';
  // UploadTask? uploadFileTask;

  @override
  Widget build(BuildContext context) {
    File? fileObj;
    String? fileName = 'No Files Selected';
    String? fileSize = '0';
    UploadTask? uploadFileTask;
    context.read<UploadEventBloc>().add(UploadEventEvent.started());

    return MultiBlocListener(
      listeners: [
        BlocListener<UploadEventBloc, UploadEventState>(
          listener: (context, state) {
            state.map(
              initial: (_) {},
              selectSongFileState: (recieved) {
                print(recieved.filePath.files.single.extension);

                fileName = recieved.filePath.files.single.name;
                fileObj =
                    File(recieved.filePath.files.single.path.toString());
                fileSize = ((recieved.filePath.files.single.size)
                    / 1000000)
                    .toStringAsPrecision(3);
              },
              fileSelectedForUploadState: (_) {
                print('$_ <---- UPLOAD STATUS');
              },
              uploadFileState: (uploadFileTaskRec) {
```

```
          print(
              '${uploadFileTaskRec.uploadStatusCurrent.toString()} 
                 <——— RECIEVED UploadTaskFile');
        },
      );
    },
  ),
],
child: Scaffold(
  appBar: AppBar(
    title: Text('Upload Songs'),
  ),
  body: BlocBuilder<UploadEventBloc, UploadEventState>(
    bloc: BlocProvider.of<UploadEventBloc>(context),
    builder: (context, state) {
      return Container(
        child: Padding(
          padding: const EdgeInsets.all(32.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  print('HERE');
                  context
                      .read<UploadEventBloc>()
                      .add(UploadEventEvent.selectSongFileClicked());
                },
                child: Row(
                  mainAxisAlignment:
                      MainAxisAlignment.center,
                  children: [
                    Icon(Icons.attach_file),
                    SizedBox(width: 10),
                    Text('Select Song File'),
                  ],
                ),
              ),
              SizedBox(height: 20),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text(
                    '$fileName',
                    style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                  SizedBox(width: 20),
                  Text(
                    '($fileSize MB)',
                    style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                ],
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  print(' <——— FILENAME UPLOAD BUTTOn');

                  context
                      .read<UploadEventBloc>()
                      .add(UploadEventEvent.uplodFileClicked());
                },
                child: Row(
                  mainAxisAlignment:
                      MainAxisAlignment.center,
                  children: [
                    Icon(Icons.my_library_music_outlined),
                    SizedBox(width: 10),
                    Text('Upload Song File'),
                  ],
                ),
              ),
              SizedBox(height: 20),
              context.read<UploadEventBloc>().state.maybeMap(
                uploadFileState: (rec) {
                  return rec.uploadStatusCurrent;
                },
                orElse: () {
                  return Container();
                },
              ),
              SizedBox(height: 20),
              OutlinedButton(
                onPressed: () {
                  AutoRouter.of(context).pop();
                },
                child: Text('Back'),
              ),
```

```
                ],
              ),
            ),
          );
        },
      ),
    ),
  );
 }
}
```

I.11 File: serach$_e$vent$_p$age.dart

```
import 'package:auto_route/auto_route.dart';
import 'package:dartz/dartz_unsafe.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_play_poll/application/home/home_bloc.dart';
import
    'package:flutter_play_poll/application/my_events/my_events_bloc.dart';
import
    'package:flutter_play_poll/application/search_event/search_event_bloc.dart';
import 'package:flutter_play_poll/constants.dart';
import
    'package:flutter_play_poll/domain/events/i_event_repository.dart';
import 'package:flutter_play_poll/injection.dart';
import
    'package:flutter_play_poll/presentation/routes/router.gr.dart';

class SearchEventPage extends StatelessWidget {
  SearchEventPage({Key? key}) : super(key: key);

  dynamic allEventsData;
  @override
  Widget build(BuildContext context) {
    TextEditingController _eventNameController =
        TextEditingController();
    String queryString = '';

    return MultiBlocListener(
      listeners: [
        BlocListener<SearchEventBloc, SearchEventState>(
          listener: (context, state) {
            state.map(
              initial: (_) {
                context.read<SearchEventBloc>().add(SearchEventEvent.started());
              },
              searchResultsDisplayed: (searchResults) {},
              showAllEvents: (searchResults) {
                allEventsData = searchResults;
                print('WORKING2?? $allEventsData <----');
              },
              emptySearchState: (_) {},
              noDataFetchedState: (_) {
                print('NO DATA FETCHED??');
                return Center(
                    child: Text('No data fetched for the input
                        query string2'));
              },
              searchQueryStringExistsState: (_) {
                print('${_.toString()} <---- PRINTING UNDERSCORE');

                context
                    .read<SearchEventBloc>()
                    .add(SearchEventEvent.searchButtonClicked(_.queryString));
              },
              joinState: (eventData) {
                AutoRouter.of(context).navigate(EventRoute(data:
                    eventData));
              },
            );
          },
        ),
        BlocListener<HomeBloc, HomeState>(
          listener: (context, state) {
            state.map(
              navigatedToCreateEventPage: (_) {},
              navigatedToSearchEventPage: (_) {},
              navigatedToMyEventPage: (_) {},
              navigatedToJoinedEventPage: (_) {},
              onCreateEventPage: (_) {},
              onHomePageState: (_) {},
              onMyEventsPage: (_) {},
              onSearchEventsPage: (_) {},
              onJoinedEventPage: (_) {},
              navigatedToUploadEventPage: (_) {},
              onUploadEventPage: (_) {},
              navigatedToUploadArtistEventPage: (_) {},
              onUploadArtistEventPage: (_) {},
              navigatedToViewReportEventPage: (_) {},
              onViewReportEventPage: (_) {},
            );
```

```
    },
  ),
],
child: Scaffold(
  appBar: AppBar(
    title: Text('Search Event'),
    centerTitle: true,
    // actions: [],
  ),
  body: BlocBuilder<SearchEventBloc, SearchEventState>(
    builder: (context, state) {
      return Column(
        children: [
          buildSearchBox(queryString, _eventNameController,
              context),
          context.read<SearchEventBloc>().state.maybeMap(
            showAllEvents: (searchResults) {
              return ListView.builder(
                shrinkWrap: true,
                itemCount: searchResults.data.length,
                itemBuilder: (context, index) {
                  return Card(
                    child: ListTile(
                      title: Text(
                        '${searchResults.data[index]['name']}',
                        style: TextStyle(fontWeight:
                            FontWeight.w500),
                      ),
                      subtitle: Text(
                          '${searchResults.data[index]['location']}'),
                      leading: Icon(
                        Icons.date_range_outlined,
                        color: Colors.blue,
                        size: 40,
                      ),
                      trailing: IconButton(
                        onPressed: () {
                          print(
                              '${searchResults.data[index]}
                                  Button to join event');
                          context.read<SearchEventBloc>().add(
                              SearchEventEvent.joinEvent(
                                  searchResults.data[index]));
                        },
                        icon: Icon(
                          Icons.add_circle_outline_outlined,
                          color: Colors.green,
                          size: 35,
                        ),
                      ),
                      onTap: () {
                        print(searchResults.data[index]['eventId']);
                      },
                    ),
                  );
                },
              );
            },
            emptySearchState: (_) {
              return Center(
                  child: Text(
                'Please enter an event name',
                style: TextStyle(color: Colors.red.shade500),
              ));
            },
            noDataFetchedState: (_) {
              return Center(
                  child: Text(
                'No data fetched for the input event name.
                    \n Try with different event name.',
                // style: TextStyle(color:
                    Colors.red.shade500),
              ));
            },
            searchResultsDisplayed: (searchResults) {
              print(
                  '${searchResults.data.length} <------SEARCH
                      DATA LENGTH');
              // if (searchResults.data.length > 0) {
              return ListView.builder(
                shrinkWrap: true,
                itemCount: searchResults.data.length,
                itemBuilder: (context, index) {
                  return Card(
                    child: ListTile(
                      title: Text(
                        '${searchResults.data[index]['name']}',
                        style: TextStyle(fontWeight:
                            FontWeight.w500),
                      ),
                      subtitle: Text(
                          '${searchResults.data[index]['location']}'),
                      leading: Icon(
                        Icons.date_range_outlined,
```

```
            color: Colors.blue,
            size: 40,
          ),
          trailing: IconButton(
            onPressed: () {
              print(
                  '${searchResults.data[index]}␣
                      Button␣to␣join␣event␣in␣
                      searchResultsDisplayed␣
                      state.map');
              context.read<SearchEventBloc>().add(
                  SearchEventEvent.joinEvent(
                      searchResults.data[index]));
            },
            icon: Icon(
              Icons.add_circle_outline_outlined,
              color: Colors.green,
              size: 35,
            ),
          ),
          onTap: () {
            print(searchResults.data[index]['eventId']);
          },
        ),
      );
    },
  );
  // } else {
  //   print('NO DATA LENGTH');
  //   // context.read<SearchEventBloc>().add(
  //   //
      SearchEventEvent.searchButtonClicked(queryString));
  //   context
  //       .read<SearchEventBloc>()
  //
      .add(SearchEventEvent.noDataFetchedEvent());
  //   return Center(
  //       child: Text('No Data fetched for the
      query string'));
  // }
},
orElse: () {
  context
      .read<SearchEventBloc>()
      .add(SearchEventEvent.started());
  return Text('No␣Events␣Created␣by␣any␣user');
},
            ),
          ],
        );
      },
    ),
  );
}

Container buildSearchBox(String queryString,
    TextEditingController _eventNameController, BuildContext
        context) {
  return Container(
    margin: EdgeInsets.all(8),
    padding: EdgeInsets.symmetric(vertical: 4, horizontal: 4),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(30),
      boxShadow: [boxTopShadow, boxBotomShadow],
    ),
    child: TextField(
      onChanged: (value) {
        print(value);
        queryString = value;
        checkInputQueryString(queryString, context);
      },
      onSubmitted: (queryString) {
        // context
        //     .read<SearchEventBloc>()
        //
            .add(SearchEventEvent.searchButtonClicked(queryString));
        print('Search␣Button␣SUBMITTED␣by␣keyboard␣
            $allEventsData<———');
        checkInputQueryString(queryString, context);
      },
      controller: _eventNameController,
      decoration: InputDecoration(
        prefixIcon: IconButton(
          icon: Icon(Icons.search),
          onPressed: () {
            // context
            //     .read<SearchEventBloc>()
            //
                .add(SearchEventEvent.searchButtonClicked(queryString));
            print(
                '$allEventsData␣␣Button␣Clicked␣ZZZZ␣
                    ${allEventsData.data}<———');
```

```
            checkInputQueryString(queryString, context);
          },
        ),
        suffixIcon: IconButton(
          icon: Icon(Icons.cancel),
          onPressed: () {
            _eventNameController.clear();
          },
        ),
        hintText: 'Enter␣event␣name␣...',
        enabledBorder: InputBorder.none,
        focusedBorder: InputBorder.none,
      ),
    ),
  );
}

void checkInputQueryString(String queryString, BuildContext
    context) {
  if (queryString.length > 0) {
    List<dynamic> newEventsList =
        allEventsData.data.map((m) => (m['name'])).toList();
    print('$newEventsList␣<~~~~');
    int i;
    for (i = 0; i < newEventsList.length; i++) {
      if (newEventsList[i].toString().toLowerCase() ==
          queryString.toLowerCase()) {
        print(newEventsList[i].toString());
        context.read<SearchEventBloc>().add(
            SearchEventEvent.searchQueryStringExistsEvent(
                newEventsList[i].toString()));
        // context.read<SearchEventBloc>().add(
        //     SearchEventEvent.searchButtonClicked(
        //         newEventsList[i].toString()));
      } else {
        print('TESTING-123');
        context
            .read<SearchEventBloc>()
            .add(SearchEventEvent.noDataFetchedEvent());
      }
    }
  } else {
    //
        context.read<SearchEventBloc>().add(SearchEventEvent.emptySearchEvent());
    context.read<SearchEventBloc>().add(SearchEventEvent.started());
  }
  }
}
```

I.12 File: joined_events_page.dart

```
import 'package:auto_route/auto_route.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
    'package:flutter_play_poll/application/joined_events/joined_events_bloc.dart';
import
    'package:flutter_play_poll/presentation/event/event_page.dart';
import
    'package:flutter_play_poll/presentation/routes/router.gr.dart';

class JoinedEventsPage extends StatelessWidget {
  const JoinedEventsPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiBlocListener(
      listeners: [
        BlocListener<JoinedEventsBloc, JoinedEventsState>(
            listener: (context, state) {
          state.maybeMap(
              viewSelectedEventState: (eventData) {
                AutoRouter.of(context).navigate(EventRoute(data:
                    eventData));
              },
              orElse: () {});
        })
      ],
      child: Scaffold(
        appBar: AppBar(
          title: Text('Joined␣Events'),
        ),
        body: BlocBuilder<JoinedEventsBloc, JoinedEventsState>(
          builder: (context, state) {
            context.read<JoinedEventsBloc>().add(JoinedEventsEvent.started());
            return context.read<JoinedEventsBloc>().state.maybeMap(
              // viewSelectedEventState: (selectedEvent) {

              // },
              showJoinedEvents: (received) {
                if (received.data.length > 0) {
```

```
    return ListView.builder(
      itemCount: received.data.length,
      itemBuilder: (context, index) {
        return Card(
          child: ListTile(
            title:
                Text('${received.data[index]['name']}'),
            subtitle:
                Text('${received.data[index]['location']}'),
            leading: Icon(
              Icons.date_range_outlined,
              color: Colors.blue,
              size: 40,
            ),
            trailing: IconButton(
                onPressed: () {
                  print(
                      '${received.data[index]}␣
                          Button␣to␣join␣event');
                  context.read<JoinedEventsBloc>().add(
                      JoinedEventsEvent.unjoinEvent(
                          received.data[index]));
                },
                icon: Icon(
                  Icons.cancel,
                  color: Colors.red,
                  size: 30,
                )),
            onTap: () {
              print(received.data[index]['eventId']);
              context.read<JoinedEventsBloc>().add(
                  JoinedEventsEvent.viewSelectedEvent(
                      received.data[index]));
            },
          ),
        );
      },
    );
  } else {
    return Center(
        child: Text(
      'No␣Events␣Joined',
      style: TextStyle(fontSize: 20),
    ));
  }
},
orElse: () {
          return Center(child: CircularProgressIndicator());
        },
      );
    },
  ),
),
);
}
```

I.13 File: view$_{r}eport_{p}age.dart$

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_play_poll/application/home/home_bloc.dart';
import
    'package:flutter_play_poll/application/view_report/view_report_bloc.dart';
import 'package:flutter_play_poll/constants.dart';

class ViewReportPage extends StatelessWidget {
  const ViewReportPage({Key? key}) : super(key: key);

/**
 *
 *     artistData.add(songIdList);
       artistData.add(songNameList);
       artistData.add(timesSongAppearedAsOption);
       artistData.add(isSongPlayedCountList);
       artistData.add(yesCountList);
       artistData.add(noCountList);
 */

  @override
  Widget build(BuildContext context) {
    double height = MediaQuery.of(context).size.height;
    double width = MediaQuery.of(context).size.width;

    List artistSongData = [];
    return MultiBlocListener(
      listeners: [
        BlocListener<ViewReportBloc, ViewReportState>(
            listener: (context, state) {
          state.map(initial: (_) {
            print('MultiBlocListenenr␣-␣INITIAL-1');
            // context
```

```
      //      .read<ViewReportBloc>()
      //      .add(ViewReportEvent.showArtistReportEvent());
    }, showArtistReportState: (received) {
      print('---> MultiBlocListenenr - 
          showArtistReportState-2');
      // if (received.artistData[0].length > 0) {

      // }
      artistSongData = received.artistData;

      print(received.artistData);
      print(artistSongData);
    }, onViewReportPageState: (_) {
      // print('----> PRINTING UNDERSCORE -
          ${_.artistData}');
      artistSongData = _.artistData;
    });
  })
],
child: Scaffold(
  appBar: AppBar(
    title: Text('View Report'),
  ),
  body: BlocBuilder<ViewReportBloc, ViewReportState>(
      builder: (context, state) {
    //
      context.read<ViewReportBloc>().add(ViewReportEvent.started());
    return context.read<ViewReportBloc>().state.maybeMap(
      initial: (_) {
        print('---> state.maybeMap - INITIAL-3');
        // context
        //      .read<ViewReportBloc>()
        //      .add(ViewReportEvent.showArtistReportEvent());
        // return Text('Initial');
        return Center(child: CircularProgressIndicator());
      },
      showArtistReportState: (received) {
        print(
            '---> state.maybeMap - showArtistReportState-4 
                $artistSongData');
        // return Text(artistSongData.toList().toString());
        // if ((artistSongData[0]).length == 0) {
        //   return Center(child: Text('Upload songs to view
            report.'));
        // } else {

        // }
        return ArtistReportWidget(
            artistSongData: artistSongData, height: height);
      },
      orElse: () {
        // return Text('No songs uploaded as Artist');
        return Center(child: CircularProgressIndicator());
      },
    );
  }),
),
);
}
}

class ArtistReportWidget extends StatelessWidget {
  const ArtistReportWidget({
    Key? key,
    required this.artistSongData,
    required this.height,
  }) : super(key: key);

  final List artistSongData;
  final double height;

  @override
  Widget build(BuildContext context) {
    return ListView.separated(
        separatorBuilder: (context, index) {
          return SizedBox(
            height: 4,
          );
        },
        itemCount: artistSongData.length,
        itemBuilder: (context, index) {
          return Padding(
            padding: const EdgeInsets.all(8.0),
            child: Container(
              decoration: BoxDecoration(
                  shape: BoxShape.rectangle,
                  borderRadius: BorderRadius.circular(20),
                  boxShadow: [boxTopShadow, boxBotomShadow],
                  gradient: forContainerLinear),
              height: height / 6,
              child: Padding(
                padding: const EdgeInsets.all(4.0),
                child: Column(
                  children: [
```

```
//Song Name
Row(
  children: [
    Icon(Icons.library_music,
        size: 45, color: Colors.amberAccent),
    SizedBox(width: 8),
    Text(
      '${artistSongData[1][index]}',
      style: TextStyle(
          fontSize: 20, fontWeight:
              FontWeight.w500),
      overflow: TextOverflow.ellipsis,
    ),
  ],
),

Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    // Times Appeared as Option
    Column(
      crossAxisAlignment:
          CrossAxisAlignment.start,
      children: [
        Row(
          children: [
            Icon(Icons.queue_music,
                size: 30, color: Colors.purple),
            SizedBox(width: 8),
            Text('Appeared␣-␣
                ${artistSongData[2][index]}',
                style: TextStyle(
                    fontSize: 18,
                    fontWeight:
                        FontWeight.w500)),
          ],
        ),
        // Times the Song got Played
        Row(
          children: [
            Icon(Icons.play_circle_fill_outlined,
                size: 30, color:
                    Colors.green.shade800),
            SizedBox(width: 8),
            Text('Played␣-␣
                ${artistSongData[3][index]}',
                style: TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.w500))
          ],
        ),
      ],
    ),
    SizedBox(width: 20),
    // Total Yes Votes Count
    Column(
      crossAxisAlignment:
          CrossAxisAlignment.start,
      children: [
        Row(
          children: [
            Icon(Icons.sentiment_very_satisfied_outlined,
                size: 30, color:
                    Colors.green.shade400),
            SizedBox(width: 8),
            Text('Liked␣-␣
                ${artistSongData[4][index]}',
                style: TextStyle(
                    fontSize: 18,
                    fontWeight:
                        FontWeight.w500)),
          ],
        ),
        // Total No Votes Count
        Row(
          children: [
            Icon(Icons.sentiment_very_dissatisfied_outlined,
                size: 30, color:
                    Colors.red.shade400),
            SizedBox(width: 8),
            Text('Unliked␣-␣
                ${artistSongData[5][index]}',
                style: TextStyle(
                    fontSize: 18,
                    fontWeight:
                        FontWeight.w500)),
          ],
        ),
      ],
    ),
  ],
),
```

```
    );
        }
    }
```

```
    ),
        ),
            );
        },
```