Department of Mathematics and Information Sciences
University of North Texas at Dallas
Dallas - Texas

Parks and Recreation Mobile Application

IT Capstone I

Eduardo Duarte

Christopher Wilson

Stanley Dappa

Daniel Uzueta

Jorge Mendoza

Gerardo Gaspar

Supervisor:

Dr. Saif Al-Sultan

April 23, 2020

Abstract

The Parks and Recreation Mobile application will provide the citizens of a city with information about local parks and recreation centers through their mobile devices. The application will allow the user to view information such as opening hours, program information, membership pricing, and contact information. The project will use a combination of the waterfall and incremental software process models. The tools used for this project will be Unified Modeling Language, the Kotlin programming language, and MySQL for the database management system.

Contents

1 Introduction **7**

1.1 Introduction 8

1.2 Problem Definition 8

1.3 Motivation 8

1.4 Project Outline 9

1.4.1 Chapters organization 9

2 Literature Review **10**

2.1 Introduction 11

2.2 Related Work 11

2.2.1 NYC 311 11

2.2.2 Florida State Parks Guide-Pocket Ranger 12

2.2.3 Parks Path - Local Park Finder 13

2.2.4 Charlottesville Parks and Recreation Application 14

2.2.5 Golden Gate Park 15

2.2.6 Recreation Gov 15

2.2.7 WMU Student Recreation Center 16

2.2.8 Philadelphia Parks and Recreation 17

2.3 Software Development 18

2.3.1 Rapid Application Development 18

2.3.2 Waterfall Model 19

2.3.3 Chief Programmer Team Approach 20

2.3.4 Incremental Model 20

2.3.5 Spiral Model 21
2.3.6 Rational Unified Process 22
2.3.7 Agile Processes 23
2.3.8 Kanban Method 24
2.3.9 Crystal Methodologies 24
2.3.10 Open Source Development 25
2.4 The Process Model that will be used in this Project 26
2.5 Software Development Tools 26
2.5.1 Programming Languages 27
2.5.2 Database Management System 29
2.5.3 Unified Modeling Language 30
2.6 Tool Used in this Project 33
2.7 Summary 33

3 Requirement Specification and System Modeling 34
3.1 Introduction 35
3.2 Requirement Engineering 35
3.3 Requirement Documentation 38
3.3.1 User Requirements 38
3.3.2 System Requirements 39
3.3.3 Requirement Modeling 42
3.4 Summary 45

4 Design 46
4.1 Introduction 47
4.2 Model View Controller and Mobile Development 47
4.3 Graphic User Interface Layout 48
4.3.1 Loading Screen 48
4.3.2 Application Home Page 48
4.3.3 Recreation Center Home Page 49
4.3.4 Program/Event/Fitness Room Page 50
4.3.5 Drawer Menu 51

4.3.6 Setting Page . . . 52
4.3.7 Tutorial Page . . . 52
4.4 Application Navigation . . . 53
4.5 Database Design . . . 54
4.5.1 Conceptual Model . . . 54
4.5.2 Logical Model . . . 55
4.5.3 Physical Model . . . 56
4.6 Summary . . . 57

5 Implementation, Testing, and Evaluation 58
5.1 Introduction . . . 59
5.2 User Interface . . . 59
5.3 Implementation . . . 63
5.3.1 Loading Screen . . . 63
5.3.2 Home Page . . . 67
5.3.3 Recreation Center . . . 73
5.3.4 Fitness Room . . . 81
5.3.5 Gymnasium . . . 86
5.3.6 Park Information . . . 90
5.3.7 Programs . . . 93
5.3.8 Rooms . . . 102
5.3.9 Special Events . . . 107
5.3.10 Database . . . 110
5.3.11 Miscellaneous Code . . . 114
5.4 Testing . . . 128
5.5 Summary . . . 129

6 Conclusion and Future Work 130
6.1 Conclusion . . . 131
6.2 Future Work . . . 131

List of Figures

2.1 NYC311 12
2.2 Park Path Menu 13
2.3 Charlottesville Park & Rec Menu 14
2.4 Golden Gate Park Map 15
2.5 RecreationGov Home Menu 16
2.6 WMU Menu 17
2.7 Philadelphia Parks and Rec. Website 18
2.8 WaterFall Method 19
2.9 Chief Programmer Approach [1] 20
2.10 Spiral Method [1] 22
2.11 Rational Unified Process [1] 22
2.12 Use Case Diagram [1] 30
2.13 Class Diagram [1] 31
2.14 UML Information Flow Diagram [2] 32

3.1 Input/Output Diagram [1] 35
3.2 Data Flow Diagram [1] 36
3.3 Entity-Relationship [1] 36
3.4 Staff Information Edit 42
3.5 User Program View 43
3.6 User Register 43
3.7 User Enroll 44
3.8 User View Prices 45

4.1 Model-View-ViewModel Class Structure [3] 47
4.2 Loading Screen 48
4.3 Home Page 49
4.4 Recreation Center Home Page 49
4.5 Fitness Room 50
4.6 Program Tae-kwon-do 51
4.7 Drawer Menu 51
4.8 Setting Page 52
4.9 Tutorial Page 53
4.10 Applicaiton Navigation 54
4.11 Conceptual Model 55
4.12 Logical Model 56
4.13 Physical Model 57

5.1 Home Page 60
5.2 Recreation Centers 61
5.3 Programs at Kiest 62
5.4 Programs at Martin Weiss 63
5.5 Home Screens 128

Acronyms

3D Three Dimensional

DBMS Database Management System

GPS Global Positioning System

UML Unified Modeling Language

UI User Interface

Chapter 1

Introduction

Objectives:

- Provides an introduction to the proposed application.
- Outlines the chapters in this report.
- Define the motivations for this project.

1.1 Introduction

This chapter will introduce the project by defining the problem definitions and explain the motivations behind the development of the Parks and Recreation mobile application. The chapter will also summarize what the report contains and give an updated time-line that will display what needs to be completed in the future for this project to succeed.

1.2 Problem Definition

When trying to get information about a recreation center's services and information about the surrounding park, a person would have to either to either call the recreation center, drive to the recreation center, or view the website for that location. Information tends to change constantly and what is on website or told to a patron on one day might not be the same the next day. The team behind the Parks and Recreation mobile application believe that a mobile application will streamline the process of getting information from a recreation center and make it easier for the community to connect with the services that their city's Park and Recreation Department may provide.

1.3 Motivation

The motivation for the development of the system is to provide residents of any given city with all information regarding local parks and recreation centers. Since people these days tend to use their phones to look for information online, having a dedicated mobile application will make it easier for resident's to find what they are looking for. The application will be able to provide hours of operation, program pricing, membership pricing, and information regarding special events. Through personal experience, the team came to understand that sometimes there may be cases where a center may be closed due to unforeseen circumstances. If this was to happen not everybody who is part of the center's activities may not be notified of this. The application will also be able to provide emergency updates to let user know about any changes to programs, closures, and any other emergency updates. The team behind this project firmly believe that the Parks and Recreation mobile application will come of great use both the resident's of any given city and to a city that provides parks and recreation services.

1.4 Project Outline

The following describes the proposed outline of the project.

1.4.1 Chapters organization

- **Chapter 1: (Introduction)** This chapter presents the proposed system and the motivations behind the development.
- **Chapter 2: (Literature Review)**
  This chapter discussed similar applications and the tools that could be used to develop the system.
- **Chapter 3: (Requirement Specification and System Modeling)**
  This chapter explains requirement engineering and defined the user and system requirements.
- **Chapter 4: (Design)** This chapter will discuss the design of the system, databases, and user interfaces.
- **Chapter 5: (Implementation, Testing, and Evaluation)** This chapter presents that underlying code of the system, and define the test cases used gauge the quality of the system.
- **Chapter 6: (Conclusion and Future Work)** A conclusion of the proposed system will be given in this chapter. In addition, this chapter will present the possible directions of the future work.

Chapter 2

Literature Review

Objectives:

- Provide information on work related to the proposed system.
- Outline the Software development processes.
- Define software tools used for developing software.

2.1 Introduction

Developing a large piece of software can be difficult, a team must do their research to make sure the best piece of software can be created. That research can include looking into similar projects and what type of processes can be used to develop a piece of software. This chapter of the report will go over work related to this project, software development and their processes, and the tools used to develop a software system.

2.2 Related Work

There are already multiple applications that are similar to what the development team will be developing. This section will go over work that is related to our project.

2.2.1 NYC 311

This mobile application offers users the ability to get information on daily parking availability, schools, and garbage collection. As shown in Figure 2.1, one can also view a month calendar to see future events that may impact those three categories. The mobile application also features a very easy way to submit service requests and check the status on pending ones that a user may have already submitted. The mobile application looks very nice. It has a nice color scheme that really pops and is not too busy. It is also very smooth and responsive and has some nice graphics that describe exactly what you would think. For instance, a rat icon used in conjunction with the words "Rat or mouse condition". This is important because the user wants to be able to do things as fast and efficiently as possible. This mobile application is available on both iOS and Android devices. While it is understood that this is not a parks and recreation center mobile application, it is believed that the UI and the effectiveness of the mobile application make it something that the team should strive and aim for. It is offered on both iOS and Android with a 4-star rating on iOS and 4.5 with over one thousand ratings on Google Play Store [4].

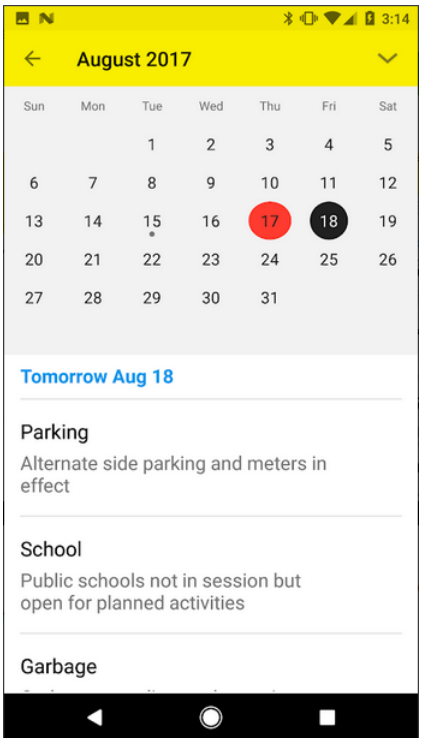


Figure 2.1: NYC311

2.2.2 Florida State Parks Guide-Pocket Ranger

This application is a bit busy, but it offers a plethora of information to the user. The application is designed for state parks but not for city parks. The application utilizes the phone's GPS location, to help identify surrounding attractions in relation to the user's location. The software does not seem to be optimized for larger phones, which is a bit of a turn off. The application contains the following: a map, beach/pools information, camping information, contact information, FindFriends, news, explore, Events calendar, and a more section which contains anything from weather to volunteer information. All the preceding items are also offered in-app, meaning that the software does not forward the user to a web page on their phone.. As mentioned before, the application offers an abundance of services, this section only mentions a fraction of them. There are some things that are out of date though, and not all of the items the user selects will work. It is offered on iOS with a 4-star rating with 200 ratings [5].

2.2.3 Parks Path - Local Park Finder

The "Park Path – Local Park Finder" mobile application is a piece of software that allows users to find information about local parks in the country by providing pictures, contact information, and hours of operation. The application provides a small description of the park, a virtual tour of the park, and a list of activities. As shown in Figure The application has a sleek and clean UI, with the menu on the left-hand side providing options. This making it easier to find the information one may be looking for. The main problem with the application is the severe lack of information. For instance, if one wants to see recreation services, then they would be directed to website which defeats the purpose of using the application. The "Park Path – Local Park Finder" mobile application was created by the National Recreation & Park Association and can be downloaded from the Google Play Store and IOS Store [6].

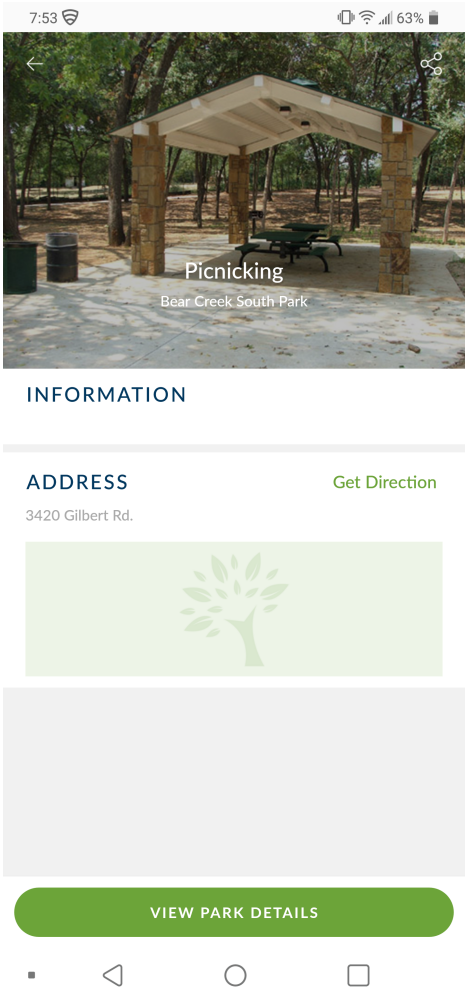


Figure 2.2: Park Path Menu

2.2.4 Charlottesville Parks and Recreation Application

The "Charlottesville Parks and Rec App" mobile application is a piece of software that allows users to find information about local recreation centers in the city of Charlottesville. The application provides information about upcoming programs with descriptions, provides contact information, and includes the option to create an account and sign up for programs from the application. The UI design for the software feels ancient, when looking through the list view for programs it can be hard to select a program since the text is too close together. The lower section of the screen provides information about trails and pools but takes up too much space. The mobile application will also crash when attempting to get information about recreation news. The "Charlottesville Parks and Rec App" was created by the City of Charlottesville and is available on the Google Play Store and the Apple IOS Store [7].

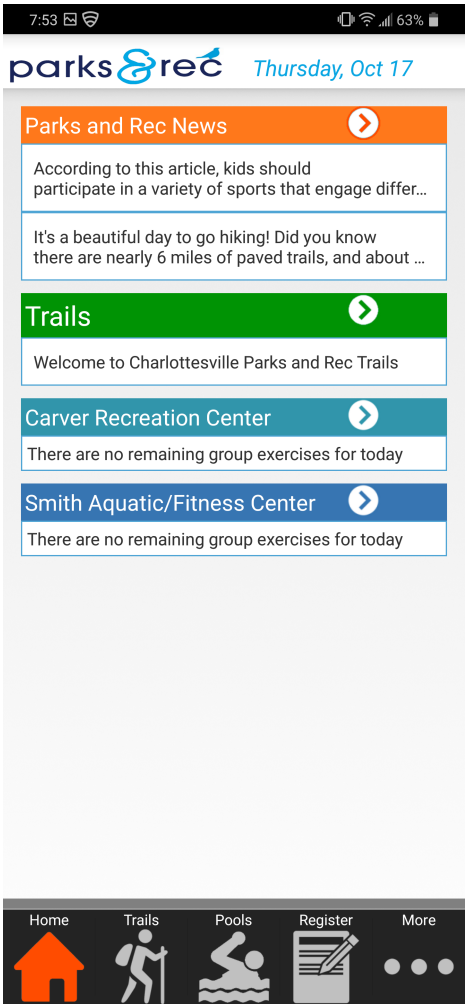


Figure 2.3: Charlottesville Park & Rec Menu

2.2.5 Golden Gate Park

The "Golden Gate Park" mobile application is designed for the city of San Francisco, California. The application includes a 3-D Map of the park, a list of different aspects throughout the park, a social media information page, contact information, admission prices, and park hours. As show in Figure 2.4, the menu is placed at the bottom, which makes it easy for the user to navigate. One issue with the application is that at times, it would be unresponsive. The loading screen can take a while to load. Once loaded, the map can lag and some of the text within the application may not properly display on a user's phone. The application is too simple and lacks features that a large city like San Francisco would need. The application was developed by Guidekick, Inc. and is available on both Google Play and iOS's app store [8].

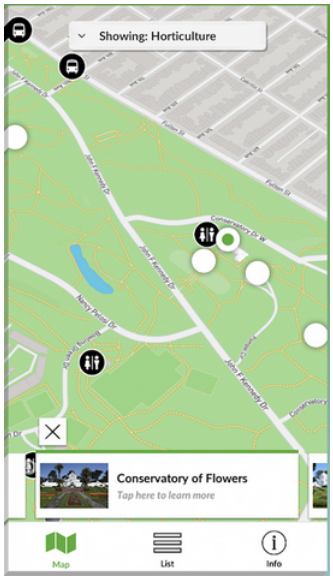


Figure 2.4: Golden Gate Park Map

2.2.6 Recreation Gov

Created by Booz Allen Hamilton Inc., the "Recreation Gov" mobile application is designed to aid a user in finding different campgrounds and campsites across the United States. The menu is placed at the bottom of the application and is divided into four different categories: Explore, Reservations, Account and Cart. Explore serves as the home page of the application and is the only page available to view if a user does not have an account. The explore section of the application gives the user everything they would want to see if they are interested in finding a nice place to camp. As show in Figure 2.5, the top of the screen includes a search bar that allows a user to filter camping sites through different criteria and shows the user

nearest campsites and the distance from the user. Upon selecting a campsite, the user is shown pictures, prices, details, general information, and any extra activities at campsite. The application also offers a three dimensional view of the campsite. Overall, the "Recreation.Gov" application really serves its purpose at an extraordinary level and goes above and beyond to help its users have the best camping experience. The only complaint this application has had is with the global positioning system feature. One user explains in a Google Play Store review "when I open again (the app) it says I'm not inside the park". The GPS seems to have an issue with possibly not recognizing the locations of users while they are within the park. The "Recreation.Gov" mobile application is available on both Google Play and iOS [9].

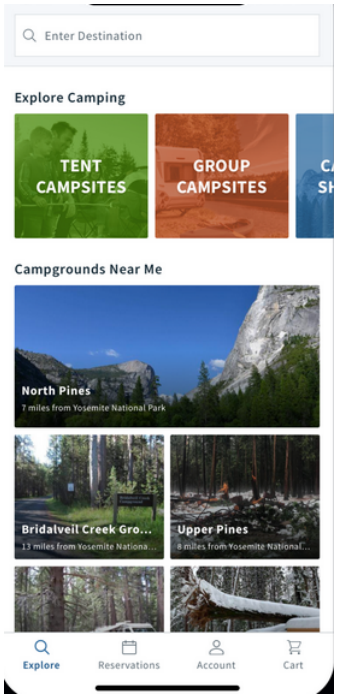


Figure 2.5: RecreationGov Home Menu

2.2.7 WMU Student Recreation Center

The Western Michigan University Student Recreation Center (WMU SRC) mobile application is designed primarily for WMU students, faculty, staffs and other affiliates. The application has an easy to navigate interface and has a well-structured menu. The menu has buttons that include home, schedule, favorite, announcement, contact and hours, notification, and settings. As shown in Figure 2.6, the home screen is divided into sections that displays announcement, links to social media, schedules, contact information, hours of operation, and a barcode function that allows a user to sign in and out of the center. The schedule is built

alongside a calendar. The schedule button also has sub options that allow easy access to activities offered in the center, instructor details, and other useful resources. The application is generally very responsive and has features that the team can incorporate into the Parks and Recreation project. Users are mostly complaining that the application occasionally crashes and is unresponsive, some suggested reinstalling the application, but this did not fix their issues. WMU SRC is available in the Google Play and iOS App Store [10].

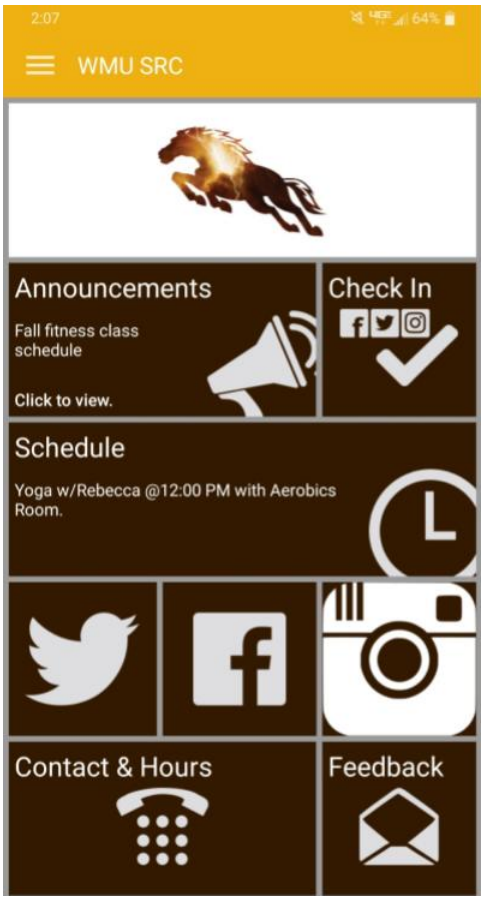


Figure 2.6: WMU Menu

2.2.8 Philadelphia Parks and Recreation

The "Philadelphia Parks and Recreation" is a very well-designed website. The website has an abundance of information. There is a section of the website that is dedicated to parks and recreational centers. There is a feature in the website that allow users to find different centers by entering their name, zip code, or address. As show in Figure 2.7,the parks and recreation centers listed on the website has very useful information

including a brief description of the center or park, age limits, hours of operation, the coordinator, link to Google Maps, and social media information. The website works seamlessly and has a many features that are very helpful to a user. The only complaint is that there is no dedicated standalone website for this [11].

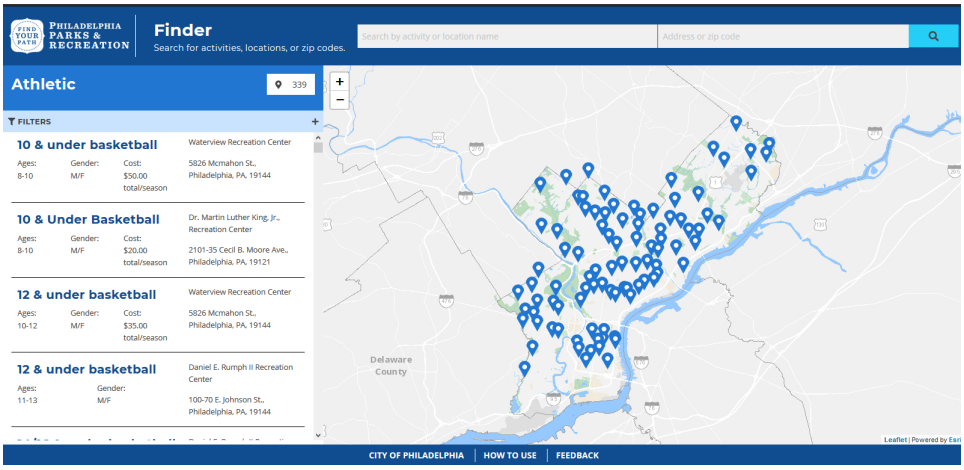


Figure 2.7: Philadelphia Parks and Rec. Website

2.3 Software Development

Developing the Parks and Recreation mobile application will require the team to follow a software engineering process. The term "software engineering" does not have one specific definition since it has been defined by other engineers. It is a broad field that will include the technical business process, methodologies and techniques used, products and how they are measured, the people in team part of the development process, and the management strategies used to achieve project success [1].

Software development is process is used to develop software in an efficient manner that includes requirement gathering, design, testing, and implementation. This section will go over the potential software development models that may be used to develop the Parks and Recreation mobile application.

2.3.1 Rapid Application Development

Rapid Application Development (RAD) is a software development model that uses an adaptive approach instead of the traditional plan driven approach that most engineering model mimics. RAD provide a wide range of flexibility that allows programmers and developers to adjust and make changes to key aspects of the process. In RAD, just about any part or phase of the development process is subjected to change as the project progress. New ideas and knowledge gained while developing the software is added to the process to

help improve the overall quality of the end product. Unlike plan-driven models that requirements and design specifications are set and stone, RAD adaptive nature makes changes and incorporation of new requirements and design into the process relatively easy and natural [1].

2.3.2 Waterfall Model

Software production and development process have always involved deploying different development techniques or methodologies. The complexities involved in software development have make software development methodology a necessity. Over the past few decades, several software development methodologies have been proposed, developed, and adopted by several companies. Spiral, incremental, rapid application development (RAD), rational unified process (RUP), rapid prototyping, Agile software development, and the Waterfall are among some of the well-known and most successful models. All these methodologies use different approach to plan, manage, and control software development process, they also share similar characteristics, they all rely heavily on performing tasks in phases or steps.

The Waterfall model was developed partly by Winston W. Royce in 1970. As shown in Figure 2.8, the Waterfall model progressively increases downward like a waterfall. The output of preceding phase is an input to the subsequent phase. The phases are structured to work consecutively, hence the preceding phase must be completed before moving to the next phase. The waterfall model has five phases: requirement or analysis, design, implementation, testing or verification, and maintenance [1] [12].

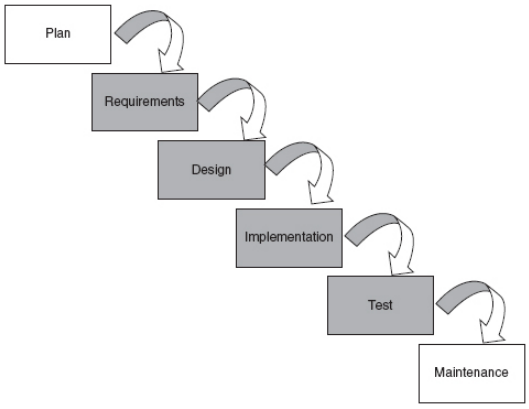


Figure 2.8: WaterFall Method

2.3.3 Chief Programmer Team Approach

The Chief Programmer Team approach is a coordination and management methodology that aids in coordinating the activities of software development. It is not a software process. This concept originated with Harlan Mills of IBM in the mid '70's. This approach is much like a surgical team in which there is a chief surgeon and a team of specialists that support him/her. In this approach, the chief programmer is the one working on the problem and the supporting team is providing the support. As shown in Figure 2.9, In this example, the chief, the surgeon, is the one that directs all the work activities. With this process, instead of having multiple small teams working on different pieces of a problem, the chief programmer coordinates, divides, and assigns the work to different members of the team. Team sizes are generally seven to ten people and is composed of designers, programmers, testers, document editor, and the chief programmer [1].

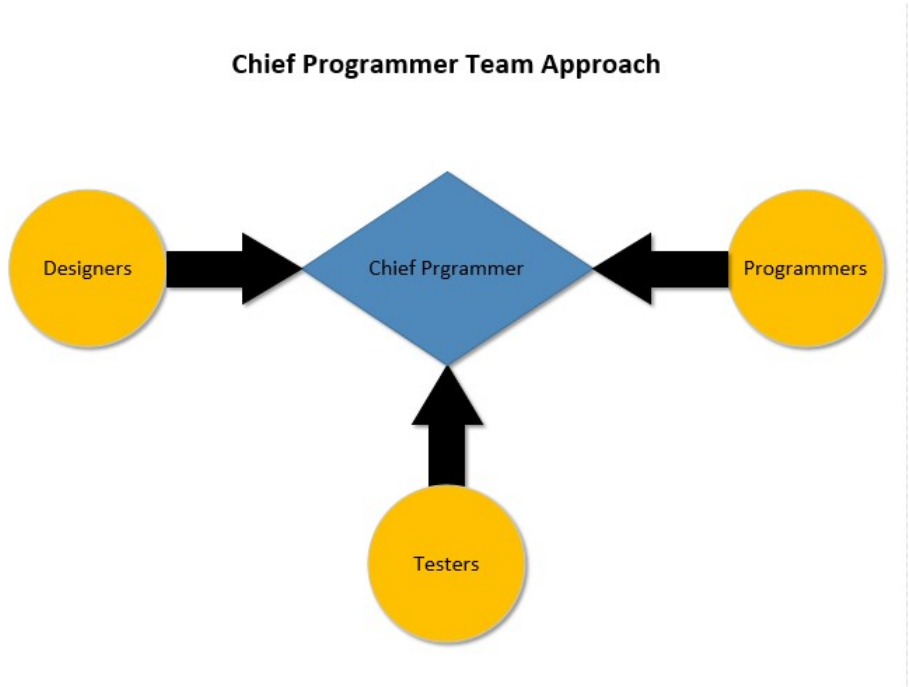


Figure 2.9: Chief Programmer Approach [1]

2.3.4 Incremental Model

As software projects grew, developers soon realized that it was easier to develop software if the project was divided into smaller components that were progressively developed incrementally and iteratively. The incremental model is essentially a modification of the waterfall model in that each small component in the waterfall process model continues to pass through each step iteratively. There are several advantages to the

incremental model. This model provides some level of risk containment, meaning that if one component in this model were to encounter an error, that component will not prevent the rest of the steps from execution. This process also utilizes what is known as the "divide and conquer" methodology. The incremental model utilizes a few different approaches, the first being the Continuous integration approach. In this approach, the requirements is developed separately using a recursive process that consists of the following: requirements, design, code, and testing. Then as the processes are completed, they are continuously merged and integrated into a central location for testing.

The Incremental model also utilizes another approach where only the core software is developed at first which contains most of the required core functionalities. This is the initial release and subsequent releases are then released as separate releases. Each new release will contain additional functionalities and supplemental features. Aside from new functions, each release also contains bug fixes. Although these releases are built independently form each other, there still exists a link between them. The existing design and code of the release before is the basis upon which the following releases are built on. This is known as the Multiple Release Incremental Model [1].

2.3.5 Spiral Model

The spiral model is one of the most useful Software Development models, which provides support for risk handling. A risk is any adverse situation that will hinder the progress of the software project. Handling these unknown risks after the project starts is where the spiral model comes to fruition. Absolving these risks can be taken care of by developing a prototype. The spiral model can cope up the risks by providing the scope to make the prototype at every phase of the development process. As show in Figure 2.10, by looking at the spiral model, each loop of the spiral is called a "Phase" of the software development process, and it may vary from project to project. The number of risks that may pop up depends of the number of phases needed to develop the project. The radius of the spiral determines the expenses, or costs, of the project as development time went on [1].

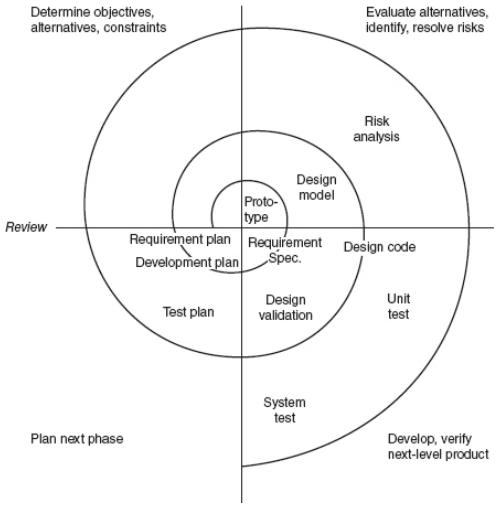


Figure 2.10: Spiral Method [1]

2.3.6 Rational Unified Process

The Rational Unified Process is defined as a prescriptive, well-defined system development process that is mainly used to help develop systems based on object and/or component-based technologies. This process was originally owned and created by the Rational Software Corporation but was later bought out by IBM in 2003. The Rational Unified Process's goal is to produce high-quality software that meets the needs of its end users all while maintaining itself within a predictable schedule and budget. As show in Figure 2.11the RUP consists of 4 phases in its life cycle and it can be depicted using the "RUP hump chart". The following are the 4 phases of the Rational Unified Process and a brief description of each one: [1].

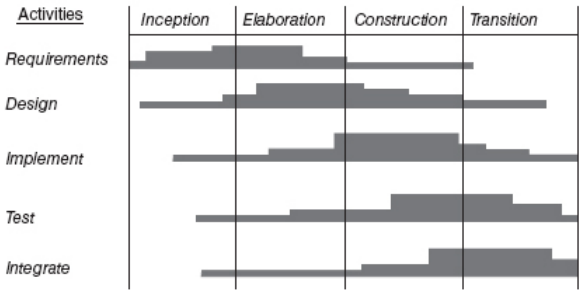


Figure 2.11: Rational Unified Process [1]

- Inception: Where the idea of the project is brainstormed. Development teams determine whether the project is worth it or not and what resources will be needed.

- Elaboration: Team further evaluates architecture and required resources of the project. Developers overlook and weigh out their options of applications for the software and the costs that will come with whatever approach they choose to take.
- Construction: The project is done. Team or developers finally develop and complete the project. All software is finished being designed, written and tested
- Transition: This is where the software is published and released to the public. Feedback and comments from end users determine if any adjustments or updates need to be made.

One of the advantages that RUP has is that it helps companies develop and envision ideas that could turn into software programs. With the RUP providing a specific plan for each step of the development process, mistakes and resources being wasted are at a low risk of happening and the risk of development costs go down as well [13].

2.3.7 Agile Processes

Agile processes are a type of software engineering model that focuses more on getting software out to the client as soon as possible and fix any issues after and during development. Agile processes focus on having a potential user there during the development to give the team feedback and see if any changed need to be made. Some characteristics of the agile process include short development times, heavy user involvement, and the ability to cope with changes during development. The agile process includes the following methods:

- **Extreme Programming**

Extreme Programming (XP) is a methodology based on agile processes that focuses on having minimal documentation. The five fundamentals of XP programming include rapid feedback, simplicity, incremental change, embracing change, and quality work.

Extreme programming, like agile methods, focuses on getting the software to the consumer as quickly as possible, but does like to ensure that quality of the program is up to standards. The only type of documentation that is kept for XP is the test code for the program. Extreme programming's best benefit for using this method would be its focus on the quality of code. Extreme programming would be a very useful methodology and would ensure that our group would release our software as soon as possible and error free as well [1].

- **Scrum**

Scrum is also based on the agile method and shares some similarities with Extreme Programming. Scrum is a very iterative heavy process that has most of its development take place in "sprint". Sprint is where development takes place small units in short periods of time. Scrum has three major rules which include the product owner, development team, and scrum master.

Each role responsible for their part, with the scrum master making sure that everybody else is following the scrum process. This method ensures that every part of the process is visible to those part of the project and if any errors are made, they can be corrected easily. At the end of each project team members will go over how committed they were to the scrum methods and how they can improve themselves for the next project [1].

2.3.8 Kanban Method

A recent addition to the family of agile methods, the Kanban method, focuses on making sure all parts of a unit have been tested together before fully implementing that part of the system. By notifying the development team that a part of the system is ready to go, the team can focus on getting the related part of that unit ready to go so that it can be tested and implemented when all testing is complete.

One major factor of the Kanban method is teamwork, since all parts of the unit must be completed before it can be tested, the team must be on the same page so that way it can be completed as soon as possible. Each process must be as clear as possible to ensure that a unit of the system is completed as efficiently as possible [1].

2.3.9 Crystal Methodologies

Crystal Methodologies were first introduced in 1991 by Alistair Cockburn and described the software development process as primarily focusing on people and the interaction among them while they work on a project. People can show that they can add value to whatever project they are working on by taking advantage of close communication, morale, access to users and others. These three aspects are just a few of the many aspects that go into developing a successful software system and it is important that the end-users and developers themselves realize how practicing these aspects can lead into developing a successful project.

Crystal Methodologies are primarily based on two assumptions which are: Teams can streamline their processes as their work and become a more optimized team, and Projects are unique and dynamic and require specific methods. Cockburn had 3 specific factors that he used to classify projects. The first one was (1) their size, which is measured in maximum number of developers; (2) their criticality, which is gauged by

the losses that a malfunction would cause; and (3) their priority, which is measured by the time pressure on the project. These factors are then followed by the Crystal Family Methodologies that Cockburn uses to explain how heavy the methodologies would be. The three crystal methodologies are: Crystal Clear which usually requires teams of 6-8 people, Crystal Orange and Crystal Orange Web who can contain up to 40 or even more people. Along with the three types of crystal methodologies, this development process also has seven properties that are beneficial and can help guide teams on creating a successful project.

1. Frequent delivery: Deliver running and tested code to real users as frequently as possible, at least every few months.
2. Reflective improvement: Stop and think if code can be improved.
3. Close communication: Encourage close communication among team members.
4. Personal Safety: Encourage team members to speak up without fear.
5. Focus: Addresses any issues that may affect the project and helps concentrate on the task at hand.
6. Easy access to expert users: Make it possible to receive feedback from those who are expertise's in the field.
7. Good technical environment: Establish an environment that includes automated tests, configuration management, and frequent integration.

The main purpose of Crystal Methodologies is to remind people how to stay centered and focused on their work during the project development so it can lead into deeper and more extensive ideas that could help the project all while using communication to its full advantage to help the team grow more like a family [1].

2.3.10 Open Source Development

The open software development model has many unique characteristics that sets it apart from the traditional proprietary software development model. It starts with the idea for a new project, as a new functionality to an existing open source software component. The next step is to conjure a design for implementing the function as well as a prototype of it and translate it from the idea to running software. When the software runs, it can be released as developmental release, despite the bugs it may contain. The software will be tested by the community to provide feedback on bug reports in email and internet message boards. That

feedback will be documented to further consideration by the developers and maintainers to improve the implementation and re-release it when it is available. This cycle can occur as many times as it needs to until the implementation Is deemed stable. Some advantages of open source development include [1]:

- Bottom Up Development: The members who got the most work done have the final say on the decisions of the project's design.
- "Release early, release often": The project will be released early to be used by the user community and update further releases when software is modified. This leads to higher quality software because of peer review and the userbase can test the software and address bugs and fixes.
- Peer review: Members of the project will review the code, comment on upon adding quality of life improvements, and test it early in the development cycle to catch bugs.
- Small, incremental changes: Often, additional features are small and non-intrusive for good reason. It is easier to understand small patches than making a large change. A small change is beneficial to the testing phase which is cyclical and ongoing in every increment of the software. Plus, it is less likely to have unintended consequences.
- Features that ignore security concerns are flagged: Any part of development that can potentially adhere the security of the software will be flagged until further notice.

2.4 The Process Model that will be used in this Project

This project will be making use of the traditional waterfall method while taking parts of the iterative and incremental method and integrating them into the waterfall method. Since the development process will be straightforward the team believes that the waterfall method will be an efficient model to follow, and adding in parts of the iterative will make up for the negative aspects of the waterfall method.

2.5 Software Development Tools

In the development market there are various tools that can be used to develop a mobile application. The emergence and continuous growth of the computing field has given our team a large range of tools to choose from. This section of the report will go over the potential programming languages, database management systems, and modeling language that may be used to develop the mobile application.

2.5.1 Programming Languages

There are various programming languages that can be used to develop the Parks and Recreation mobile application. Some of the best of languages that may be used include the following:

- **C#:** The C# language, pronounced as C-Sharp, is a programming language created primarily by Anders Hejlsberg who first had its roots planted back in 1998. Developed by Microsoft as part of the .NET initiative, C# was first released to the public in 2002. C# is defined as a simple, modern, object-oriented and type-safe programming language. The C# programming language is part of the C family and it was originally supported within Visual Studio .NET 2002. C# has received its critics of it resembling Java and how it has never really been a revolutionary language to begin with and in 2000 Anders Hejlsberg defended his programming language by stating that C# isnt a clone of Java but is much closer to C++ in terms of design.C# also supports component-oriented programming that introduces a lot of concepts that C# is more than capable of supporting in order to become a very natural language to create and use software components. With C# now being in the stage of previewing its 8th version, C# has always had the priority of making sure of keeping up with aspects such as the separate virtual and override modifiers, rules for method overload resolution and support for explicit interface member declarations just to name a few with the end-goal in mind of avoiding malfunctions and crashes of the programming language. [14] [15].

- **Kotlin:** The Kotlin programming language is very well-known as being a programming language used to build mobile applications for Android devices. The programming language was first introduced in 2010 as it originated at JetBrains which is a Czech company. At the time when Android cellular devices were first being introduced to the world, printing something like "Hello World" with Java is said to have taken up 3 lines of code but with a modern language like Kotlin printing a simple phrase like that could now be printed out with 3 words. Kotlin is based off Java and mainly focuses on interoperability safety, clarity, and tooling support and combines object-oriented and functional programming features. Examples of applications that use the Kotlin programming language are Postmates, Uber, and Square [16].

- **Java:** Java programming language was originally created by Canadian computer scientist James Gosling at Sun Microsystems in 1995. Sun Microsystems was later acquired by Oracle. Java is an object oriented and class-based computer programming language that has very similar syntax as C,

C++, and other similar high-level language. Java is one of the most widely used programming language partly because of its portability. Java can run on virtually any kind of computer architecture that can host Java Virtual Machine (JVM). Java can be used to develop application across several platforms and environment. Gmail, Android Operating System, and Wear OS are all developed using Java [17].

- **Javascript:** Is a high-level scripting language that was created by Brendan Eich in 1995. JavaScript is one of the most widely used computer language. It is one of the core languages used in the World Wide Web. JavaScript is based on C, Java, Lua and few other object-oriented languages. It supports manipulation of texts, arrays, and other event-driven features [18]

- **Python:** Python is a programming language that was created by Guido van Rossum in the early 1990's at Stichting Mathematisch Centrum (CWI) in the Netherlands. Python is a successor to the programming language, ABC and is defined as a high-level general-purpose programming language. Some interesting features about Python is that it can be an extended in C and C++, leading Python to be able to provide the speed needed for even compute intensive tasks. Another characteristic of Python is that it is an object-oriented language and it has an interesting rule to it: everything is an object in Python. The first version of Python was released in 1991 and now it is on its 3.7.4 version according to the Python website. Python can is a great programming language that is used to develop applications such as Instagram, Spotify and Pinterest. [19]

- **Lua:** Lua (LOO-ah) was designed, implemented, and maintained in 1993 by a team at Pontifical Catholic University of Rio de Janeiro in Brazil (PUC-Rio). It supports various types of programming such as procedural, object-oriented, functional, data-driven, as well as data description. Lua has proven to be a robust language as it is in many industrial applications with an emphasis on embedded systems and is a leading scripting language for video games. It also has a reputation on being the fastest leading scripting language out there. Lua can be ported a variety of platforms with a standard C compiler, and it can run on all flavors of Unix, Windows, mobile phones, embedded microprocessors, and IBM mainframes. It has also received a simple, yet well documented API that can extend its code to be integrated on other languages. Lua can provide meta-mechanisms for implementing features rather than providing a host of features directly in it. It is a lightweight program, and it is free open-source software. [20]

2.5.2 Database Management System

A Database Management System (DBMS) is a type of software that allows the creation of a database using Data Definition Language (DDL) and the manipulation of a database using Data Manipulation Language (DML). A database is a centralized location for the storage of logically related data and are a vast improvement on the old file-based system. The file-based system is inferior because it allowed for multiple applications to share the same information independently of each other. Which means that if there are changes made in one application, those changes will not appear on the other applications unless they are updated manually; which is tedious and time consuming. A DBMS acts as the middleman between the user and the database itself; it allows a user to modify and interact with a database without having to meddle with it directly. One of the biggest advantages of utilizing a DBMS is that they let both users, and developers access the same information simultaneously. They also allow for a reduction in redundant data and improved data integrity. Since a database is centralized, all changes made to it will be applied to everyone who uses it. Therefore, the data across all systems will be accurate and up to date. Data security is also improved because a DBMS can implement access control policies that grant only authorized personnel access to the database. and a way to balance the needs of multiple applications that require the same data.

A **Relational DMS** is a subset of DBMS's that structures its data in the form of tables, utilizing rows and columns to organize different types of information. Relational DBMS means that the different tables within are somehow "related" to each other; hence "relational". These kinds of DBMS utilize Structured Query Language to access and manipulate the DB: [21]

- **Oracle**

Founded in 1977, Oracle was founded by Larry Ellison, Bob miner, and Ed Oates. Oracle is not open sourced and is ideal for large applications and is popular amongst banks due to its extensive integrated business tools. [22]

- **MySQL**

It was founded in 1994 by Michael Widenius, David Axmark, and Allan Larsson. It is currently owned by Oracle (2010). It is hailed as the largest and most popular of the RDMS, MySQL is open source and claims to be easy to use. MySQL is used in by Facebook, Twitter, YouTube and more. MySQL also offers built in security and is ideal for web-based applications. Drawbacks of MySQL are that it may not include some advanced features and it may suffer in performance when scaling [23] [24]

- **SQLite**

Founded in 2000, by D. Richard Hipp, SQLite is a server-less SQL database engine. It is written in C and reads and writes onto disk files. This means you can store an entire server, its tables and indices, on one single file. Another key feature of SQLite is that it is open source and free to use.

Matisse and Jade, on the other hand, are Object oriented DBMS's. I would recommend using a RDBMS only because the team will be working with MySQL in a college course. This means that the development team will learn about DBMS's while planning to create one for the Parks and Recreation application. There is no "ideal" DBMS, it depends on the application in which one plans to use it for [25] [26].

2.5.3 Unified Modeling Language

The following section will go over the Unified Modeling Language (UML).UML is an object-orientated modeling language that gives developers the tools to define the elements and relationships that will be used to model the software requirements and design. UML is used to model the design of a software system that can understood by both those on the software development team and their clients. It gives everybody part of the project a clear view of what is going to be built and how it will be built. UML is currently the most popular modeling language with wide industry and academic support. There are different diagrams of the UML, the following will explain each that may be used in this project [1].

- **Use Case Diagram:** This type of diagram is created with all the user requirements defined entirely, with the main cases for the program already defined as well. As shown in Figure 2.12 Potential users of the system would go through different situations of a system function to help the designer understand what is needed for the final product. Extensive documentation of the results will be required. [1]

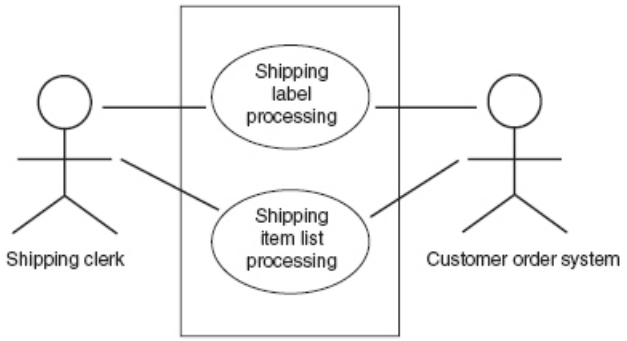


Figure 2.12: Use Case Diagram [1]

- **Class Diagrams:** An object orientated design method. Entities are defined as classes and each of those classes can have their attributes (instance variables) and their own behaviors (methods). Relationships can be defined as lines and each class can have its own subclasses that be used to better define new entities. As show in Figure 2.13 a class diagram is very useful and easy to understand. [1]

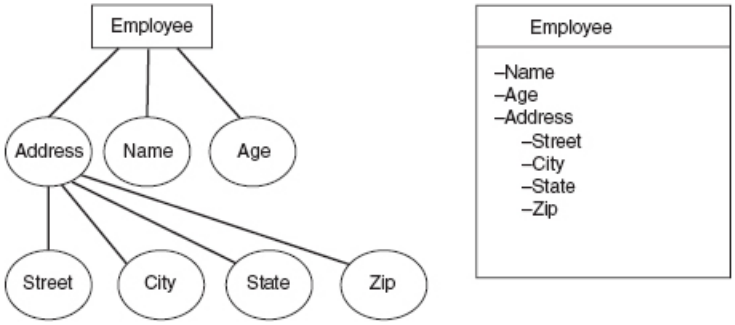


Figure 2.13: Class Diagram [1]

- **State Modeling:** An object orientated design method that is used to describe the current state that an object is in. It is useful for showing how one object can affect another object through various functions.[27]

- **Sequence Diagrams:** Shows what order some entities will execute a behavior and how long it may take them to do it. [27]

- **Collaboration Diagrams:** Like sequence diagram but does not include a lifeline in the diagram and will have more messages given to behaviors. [27]

- **Package Diagram:** This UML diagram will show the packages of classes and other related subclasses that are among them. To be more specific, it shows all related classes and their dependencies. This will help a designer get a better understanding on dependent systems and what changes will affect those systems. [27]

- **Activity Diagram:** The activity diagram combines different techniques from various other models (event models, state models etc...). Any type of task done in the real world be written down as an activity in the diagram. These activities can be completed as the time as others and follow a sort of sequence that allows designers to see what activities are connected and which one is next. [27]

- **Profile Diagram:** Describes the lightweight extension mechanism that will define the possible values

and constraints that may be used to extend current UML diagrams to potentially other platforms that may be used. [28]

- **Deployment Diagrams:** Shows the relationships between hardware and software in the final product. This is a good way to show how different components in the system are connected and how they work in the system. [27]

- **Component Diagram:** This diagram shows the components in a system and their various dependencies on other systems. Very similar to the package diagram as it will show how a change in components will affect dependencies [27].

- **Timing Diagram:** Shows the interactions of a module in relation to a set condition of time. The main goal is to see how a function of the system interacts with other modules and what changes are made in a certain amount of time. [29]

- **Information Flow Diagram:** This diagram shows how information is exchanged between multiple models in an abstract way. The diagram does not focus on the exact information that is exchanged, but rather a general idea of what is. As show in Figure 2.14 the diagram will show what general information will be transferred between modules. [2]

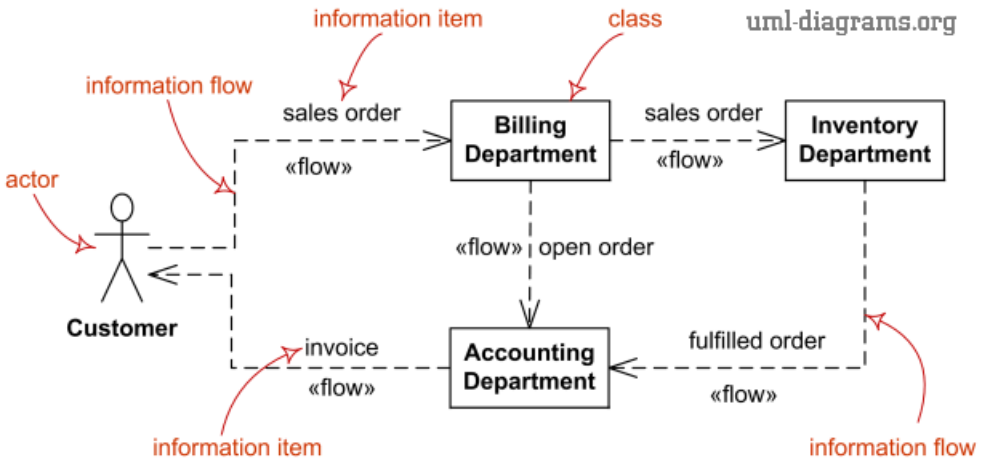


Figure 2.14: UML Information Flow Diagram [2]

- **Communication Diagram:** This diagram displays the interactions between two objects using sequenced messages. Objects are considered lifelines and are linked to other objects with messages to show the interactions with them. The communication diagram shares many similarities with sequence diagram and can even be considered an unstructured form of the sequence diagram. [30]

2.6 Tool Used in this Project

Though there are many tools to choose from, the development team has already decided on what tools to use. For the programming language the team will be using Kotlin since it has excellent documentation, can run on any android device, and is built upon Java which is a language the team has experience with. For the DMBS the team will be using MySQL since it is free and will be taught to the team during development. Finally, the unified modeling language will be used since it has excellent documentation and is an industry standard.

2.7 Summary

This chapter of the report outlined the possible software development tools and processes that may be used to develop the Parks and Recreation mobile application. This chapter also evaluated already existing applications and how they compare to this project's development. This chapter discussed all tools and processes that may be used and eventually came down the specific tools that will be used to develop the mobile application. The next chapter will introduce the requirements of the proposed system.

Chapter 3

Requirement Specification and System Modeling

Objectives:

- Define The Requirement Engineering process
- List The Requirements of the Parks and Recreation Mobile Application
- Illustrate The System Requirements with the Unified Modeling Language

---

3.1 Introduction

This chapter will define the requirements for the Parks and Recreation mobile application. As mentioned in chapter 2, all software engineering processes have a requirement phase where the software's main function (what it will do) will be defined. The chapter will go over the requirement engineering processes, the user and system requirements, and the requirement models that will illustrate the system requirements.

3.2 Requirement Engineering

A requirement is a statement that describe what a software system should be, but not how it is to be constructed. Requirement engineering is a set of activities that correlates to the development and agreement of the final set of requirement specifications. The completeness and clarity of the requirements are a very important factor in determining whether a project will be successful or not. The following are the requirement engineering activities involved in a project [1]:

- **Elicitation** Requirements elicitation is the process of gathering and collecting requirements, pertaining to the system being designed, from the user and other sources of information. A requirements analyst is usually tasked with communicating with the business to collect these requirements. The requirements analyst must possess good communication skills and be able to interpret what the user wants because the users may not know how to state their needs. Sources of information can include documentation, prior systems, and conversations with other stakeholders to identify how they plan on interacting with the system [1].

- **Documentation and Definition** Involve documenting, in detail, the requirements. There are several forms to write the requirements definition which can include the following:

  Figure 3.1 displays an input/process/output method that depicts the requirements in English text using numbers to identify which requirement it is in the list [1].

Requirement Number	Input	Process	Output
12: Customer order	• Items by type and quantity • Submit request	• Accept the items and respective quantities	• Display acceptance message • Ask for confirmation message

Figure 3.1: Input/Output Diagram [1]

Figure 3.2 displays a data-flow diagram which graphically displays a system's data flow and where it will go within the system [1].

Figure 3.2: Data Flow Diagram [1]

Figure 3.3 displays an entity-relationship Diagram which is used to show relationship between two entities and their attributes as well [1].

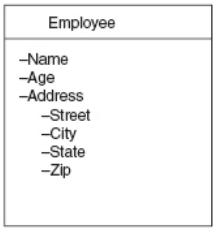


Figure 3.3: Entity-Relationship [1]

- **Specifications** This step involves finalizing the Software Requirement Specification (SRS) document, which serves as an internal software engineering document for the software team. Once all the requirements are defined, analyzed, prototyped, and reviewed, it must be typed on the SRS. As the requirements become more numerous and complex, they must be more specified formally and completely. This document can be used by a variety of users, such as system customers, managers, system engineers, test engineers, maintenance engineers. The SRS must include:
  - Introduction: : Gives an overview of the purpose, scope, references, and definitions of terms.
  - High Level Description: Tells a general description of the product, its functions, user characteristics, constraints, and dependencies.
  - Detailed Descriptions: describes details of each functionality, interfaces, performance, design constraints etc. [1]

- **Prototyping** During the prototyping phase is when the design team allows the users to get the first actual feel for the software that the team is attempting to create. The user interface is an important aspect of prototyping phase. During prototyping the team is not interested in the system internals or the algorithms but how the user will interact with the system. The visual looks, display, and interaction with people and flow are the two main aspects that we will analyze during the prototyping phase. The team must make sure to review all the requirements with users and customers as this an essential part of prototyping. From here the feedback or comments that the team receives will make it easier to go back and correct any issues that were found with the system [1].

- **Analysis** Once the team is done with the elicitation process, the next step is the analysis of the gathered requirements where the team will have to organize the set of requirements. During the analysis requirement process the team design must perform two activities: Categorizing or clustering the requirements and prioritizing them as well. One of the main things that designers should look for in analysis is that all our requirements maintain consistency and completeness. Requirements that could be incomplete or duplicated could deeply affect a project down the line. Another critical step in analyzing requirements is prioritizing them based on different factors that will benefit the user. The development team must go through each requirement and decide of how much each requirement will be a priority. Once completed, the team will have to trace back and make sure requirements were prioritized in the order the team want them to be in and make sure again that there are not any duplicates again [1].

- **Review**: is the process of analyzing and checking the gathered requirements for correctness. Errors and inconsistencies found in the requirements are corrected and the changes and corrections made are added to the requirement definition. Considering that incorrect and inconsistent requirement definition is one of the chief causes of software project failure, it is very crucial to ensure that requirements are error free. Errors and inconsistencies not detected in the review phase can lead to multiple costly design errors. The review is usually done by a team of users, costumers, and developers [1].

- **Agreement and Acceptance** The requirement agreement and acceptance are the final step of requirement engineering processes. In this step, all stakeholders including users, owners, and technical team involved in the software project will agree and accept the various gathered requirements that are stated in the requirement specification document. The specification document will then serve as the official contract binding all stakeholders involve in the project. The conclusion of the agreement and acceptance step also concludes the requirement engineering phase of software engineering [1].

3.3 Requirement Documentation

The following section will go over the user requirements which define what the application will do and the system requirements which define how the user requirements will work.

3.3.1 User Requirements

1. User must be able to select the recreation centers they only wish to see (a local profile of the recreation center).
2. User must be able to view hours of operation.
3. User must be able to view contact information and call immediately if they wish too.
4. User must be able to view pictures of the inside/outside of the recreation center, possibly a three-dimensional tour of the building.
5. User must be able to view map of the recreation center and the surrounding park.
6. User must be able to view pictures of the surrounding park.
7. User must be able to view programs available at the recreation which includes the following: information, pictures, instructor information, prices, date, and program hours.
8. User must be able to view prices for recreation memberships such as standard pass, daily pass, and gym memberships.
9. User must be able to register for programs.
10. User must be able to view rental rooms with their respective prices and possibly pictures of the room set up like a party. All room details such as space, equipment, rental conditions.

11. User must be able to view upcoming events at the specified recreation center.
12. User must be able to get push notifications for emergency updates such as building closures.
13. User must be able to sign up for reminders such as practice/game cancellations, payments, and class times.
14. User must be able to use GPS to find the nearest recreation center.
15. User must be able to view calendar that shows all the events, programs, and their times.
16. User must be able to see recommended programs and events based on their browsing history.
17. User must be able to view information on recreation center staff.
18. Staff must be able to update any changes to prices, program information, staff information, etc...
19. User must decide whether or not to create an account.
20. User must be able to have an accessibility section if they have disabilities or troubles navigating the application.
21. User must be able to have access to other social media websites (YouTube, Twitter, Instagram etc.) through quick links.

3.3.2 System Requirements

1. System should allow user to create an account. This enables a user to save events, classes, favorite recreation centers, and have a personal calendar. This will be the first option a user will view when opening the application for the first time.
2. System must allow a user to select the recreation center they wish to see (a local profile of the recreation center). This will make the selected recreation center's homepage appear first when the application is launched.
3. System will allow the user to view hours of operation, this will be displayed on home page of each recreation center.
4. System must be able to show contact information and allow user to call immediately if they wish too. The application will be integrated with the phone's call application. This makes it easy for the user to quickly contact their selected recreation center.

5. System must show pictures of the inside and outside of the recreation center, possibly a three dimensional tour of the building. Various pictures of the inside and outside of the building and park and through a "tour" option the user will be able to view all these pictures.

6. System must show pictures of the surrounding park. Users will be able to select "view park" and see various pictures of the surrounding area.

7. System will allow user to view map of the recreation center and the surrounding park. This can be done via a link to Google street view.

8. System must show programs available at the recreation center. This includes information, pictures, date and times, and instructor information. This allows users to decide on what programs they want to participate in. If the provided times do not suit the user, then the system will display similar programs or other locations that offer that program during their preferred time

9. System should allow users to register for programs. Their registered programs should then be added to their calendar.

10. System must display prices for recreation memberships such as recreation card prices, and gym prices. The system should have a fee schedule page that has a list of all programs and their associated prices.

11. System must show rental rooms with their respective prices and all room details such as space, equipment, rental conditions. There should be a link between each room and the fee schedule page. The price tag on each room should be hyper linked so by clicking the price tag users are transferred to the fee schedule page for full price details of the room. The name of the programs in the fee schedule page should also be hyper linked, so users can go back and forth between the room and the fee schedule. There should be a list of amenities that comes standard with each room.

12. System must show upcoming events at the specified recreation center. There should an event page that list past, current, and upcoming events. The system should allow users to view the events in a calendar. By clicking an event, users are directed to the actual event. There they can get full details of the event. The system should allow staff members to be able to create, modify, and remove events.

13. System must send push notifications for emergency updates such as building closures, in-climate weather, etc. This type of notification will be sent out to all users and will also appear again when they open the app if they don't have push notifications activated. The notification will be a pop up

placed in the middle of the screen and will be sent to users according to the certain rec center they have set as a favorite.

14. System must allow users to sign up for reminders such as practice/game cancellations, and payment dates. When a user registers for a program it will be saved under their profile. Two hours before the program starts a push notification will remind the user that they have a program to attend. Due to time constraints, we were not able to implement this section, but is saved for future work.

15. User will be able to use phone's Global Positioning System (GPS) to find the nearest recreation center. By setting up predetermined addresses of the different parks and recreation centers around, the system will be able to find the nearest park or rec center based on the user's location with the use of a GPS. There will be a button located somewhere near the top that will say "Find nearest parks/recs" and once clicked on, the system where locate the user and display a list of parks and recreation centers closest to the user based on their location. Due to time constraints, we were not able to implement this section, but is saved for future work.

16. System must show calendar that shows all the events, programs, and their times. There will be a section at the bottom of the screen that will split the sections of the app with "Calendar" being one of the sections. The system will have color-coded pre-filled events and programs (with different dates and times) offered by each respected park. System will allow user to manually enter any extra information they would like to add and automatically color-code it for them.

17. System must show users recommended programs and events based on their browsing history. A count ticker will keep track with the amount times a program/event is selected. Whatever top 3 programs have the most amount of ticks will be displayed to the user first when they open the application.

18. System must show staff who work there with small description. This will be displayed in a staff section in the home page of a recreation center. This will include small description of duties and personal information such as hobbies and likes.

19. System should allow staff to update any changes to prices, closures, events, etc. Staff will be able to make an administrative account on the mobile application. Once registered as an administrator they will be able to make the necessary changes and additions to information on the Parks and Recreation application database.

20. System should have an accessibility section for those who have disabilities or trouble navigating through application. Some of these features may include colorblind options, and text-to-speech functionality.

21. Application will allow users to directly view any social media pages such as YouTube, Twitter, and Instragram. Links will be places at the bottom of each recreation center home page that leads the user to the social media pages either through the respected application or through a web browser.

3.3.3 Requirement Modeling

This part of the section will use UML diagrams to demonstrate how some of the system requirements will work within the context of the system.

Figure 3.4 illustrates a use case diagram to demonstrate how a supervisor at a recreation center will be able to easily edit information about their center. A supervisor account will be created for each recreation center only allowing access to the management.

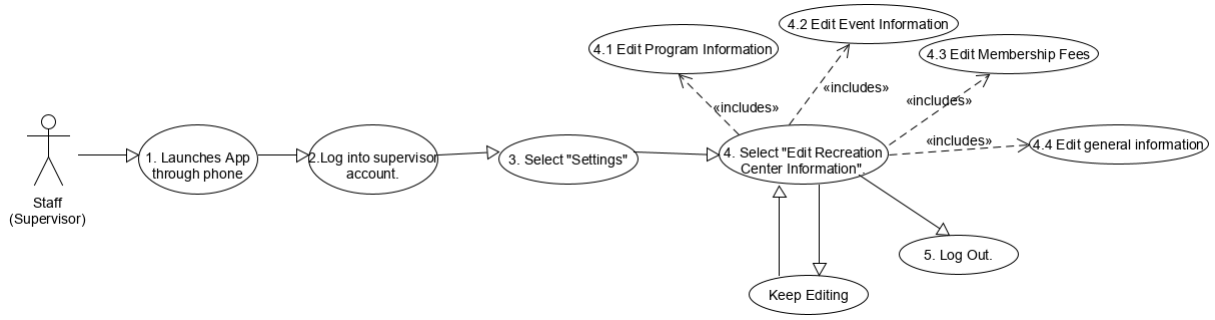


Figure 3.4: Staff Information Edit

Figure 3.5 uses a use case diagram to demonstrate how a user will view information on programs and events at a recreation center and how these programs/events will be saved for each account.

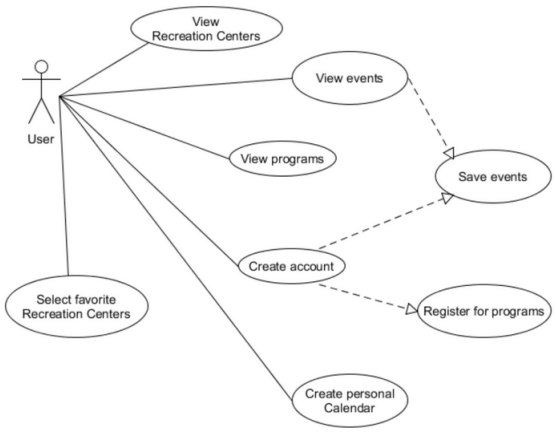


Figure 3.5: User Program View

Figure 3.6 uses a use case diagram to demonstrate how a user will be able to register for a program through the application.

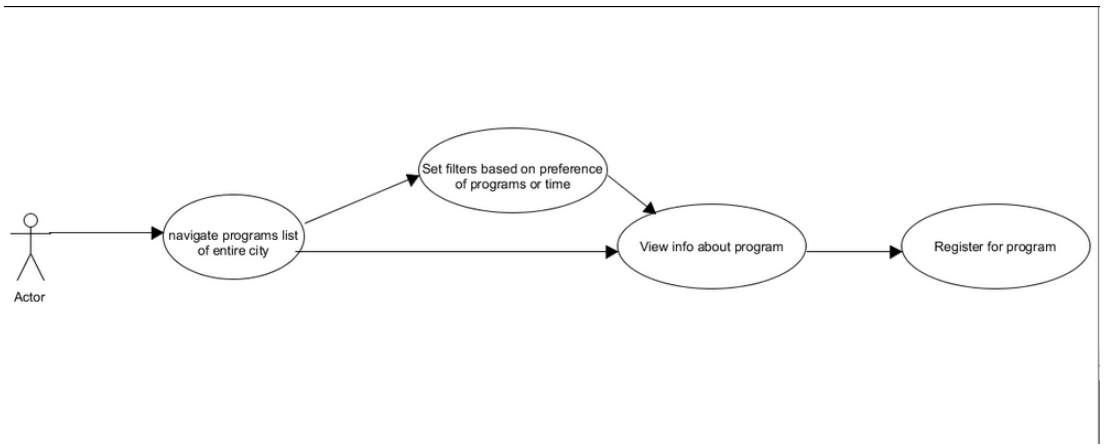


Figure 3.6: User Register

Figure 3.7 uses a use case diagram to demonstrate how a user will enroll in a program through the application.

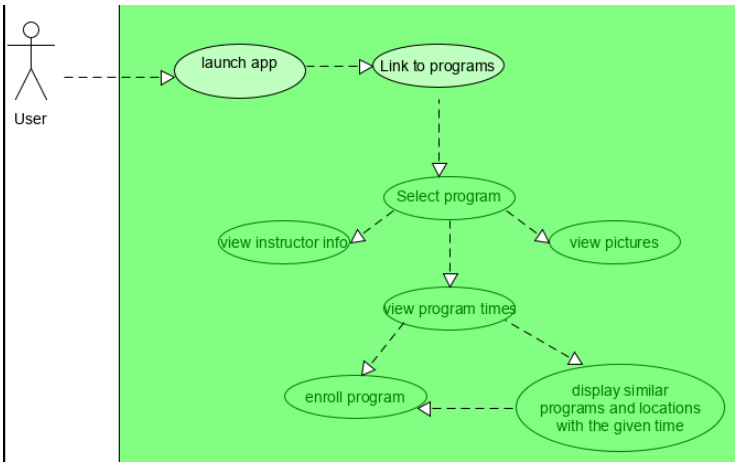


Figure 3.7: User Enroll

Figure 3.8 uses a use case diagram to demonstrate how a user will be able to enter the application and quickly see the membership and program fees of the recreation center they select.

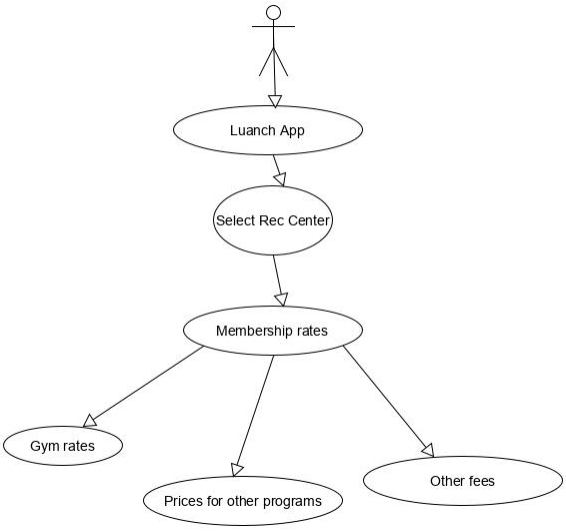


Figure 3.8: User View Prices

3.4 Summary

The requirement phase of software engineering is a very complex step in software development. This section went over the requirement engineering process and its steps to gather and analyze requirements. The user requirements were gathered to show what the Parks and Recreation application, and the system requirements were defined to explain how each user requirement will be implemented into the application. Finally, the use of the Unified Modeling Language helped illustrate the requirements that will be implemented into the system. The next chapter will explain the design of the proposed system.

Chapter 4

Design

Objectives:

---

1. Demonstrate the flow of the system.
2. Explain the database design of the system.
3. Illustrate the proposed GUI of the system.

4.1 Introduction

This chapter will demonstrate the use of the UML diagrams to illustrate how the system will work within the context of the user. This chapter will discuss the user-interface (UI) views and describe their context within the application, how the user navigates through application, and the database that will hold the applications information.

4.2 Model View Controller and Mobile Development

The Model View controller (MVC) is a design architecture that focuses on separating a web or mobile application into three components.

- Model: The base logic of the application and its properties..
- View: The UI components of the application that the user will see to navigate through the application.
- Controller: Between the model and the view that manages how the information in the model is presented in the view.

Though this design philosophy has been followed by many developers and is still very popular, the MVC architecture can create large pieces of code through the view-controller component that makes it harder to organize and modify. Due to this and the simplicity of this project, the team will be using a modified version of the MVC called the Model-View-ViewModel architecture (MVVM). Figure 4.1 shows the class structure of the MVVM architecture. This design architecture combines both the view and the controller into one view model. This model creates a direct link to the data model without having to have a third component to get information. The team has decided that following the MVVM design architecture is best for develop the Parks and Recreation Mobile Application. [3] [31]

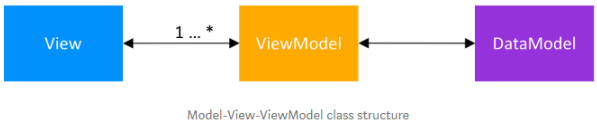


Figure 4.1: Model-View-ViewModel Class Structure [3]

4.3 Graphic User Interface Layout

This section of the report will display all of the UI screens a user may see on the device and give a brief description of their use within the application. The following screens represent the view controller segment of the Model-View-View-Model

The loading screen will be the first thing the user sees when they open the application. As shown in figure 4.2, this screen will have the city's logo in the center of the screen with a loading circle under it to indicate to the user that the application is launching. Behind these two views is an image of the city that is using the application's services. This image will change depending on the time of day. The loading screen is expected to only be on screen for 5 seconds to let the main components of the application load up. The logo will be moved up to bring up the next screen of the application.

4.3.1 Loading Screen

Figure 4.2: Loading Screen

4.3.2 Application Home Page

Once the loading screen is finished the logo will move up to the top and the page shown in figure 4.3 will come in from the bottom of the screen. The user will be presented with the application homepage that will consist of a search bar, several click-able tiles that represent different recreation centers that are divided into two groups, favorites and all other recreation centers. There will also be a menu button at the top left-hand

corner. Each Tile will have the name of the recreation center with a transparent image behind the text.

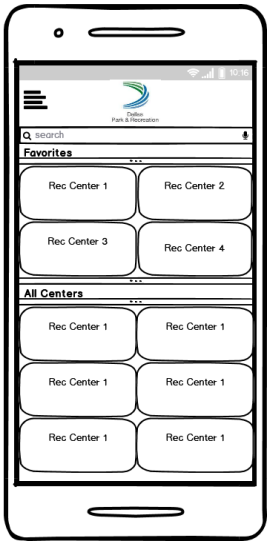


Figure 4.3: Home Page

4.3.3 Recreation Center Home Page

Figure shows the main page for each recreation center. From this screen users can navigate to the "Programs", "Events", "Rooms", and "Fitness Room" screens. Hours of operation, contact information, and address of the center can be found on this screen. A "Back" button located on the top left-hand corner can be used to return to the app Homepage. On the top right of the screen there will be a star option that allows the user to set the recreation center as a favorite center.

Figure 4.4: Recreation Center Home Page

4.3.4 Program/Event/Fitness Room Page

Figure 4.5 shows a template page that will be used to show information such as programs, special events coming up, room reservations, and fitness room information. All pages will share the same structure but differ in content. Each page will include a description of their content, pictures of the content, and any other information such as prices and time. Using this structure throughout the application allows the team to create design consistency and makes it easier for the user to navigate throughout the application. Figure 4.6 displays another use of the template that is displaying.

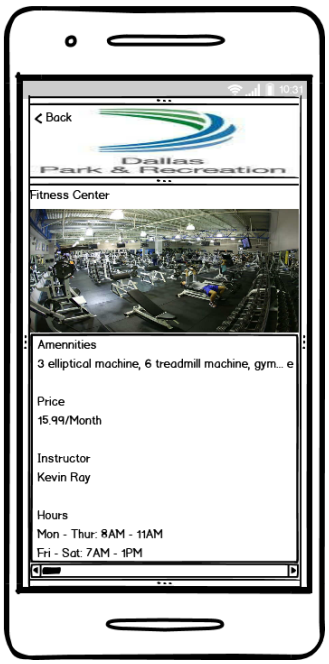


Figure 4.5: Fitness Room

Figure 4.6: Program Tae-kwon-do

4.3.5 Drawer Menu

As shown in figure 4.7 upon clicking the menu button, a drawer style menu will open and present the "Settings", "About", "Run Tutorial" buttons. The settings option will take the user to the settings page, the about setting will give users general information about the application, and the run tutorial option will show users how to use the application again.

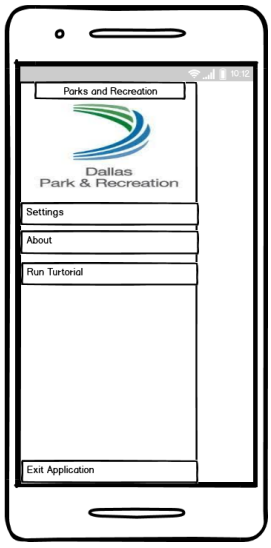


Figure 4.7: Drawer Menu

4.3.6 Setting Page

Figure 4.8 displays the setting screen which is accessed through the drawer menu. In this screen users can edit their favorite recreation centers, restore the application to its default settings, turn notifications on or off, and control location access. Due to time constraints, we were not able to implement this section.

Figure 4.8: Setting Page

4.3.7 Tutorial Page

Figure 4.9 displays the tutorial page which will be shown when the user first launches the application and will be available for future use. The screen will show users how to navigate through the application and how to set up favorite recreation centers. An image will be shown along with a description on the lower half of the screen. If they wish to view the screen again they can access it through the drawer menu. Due to time constraints, we were not able to implement this section.

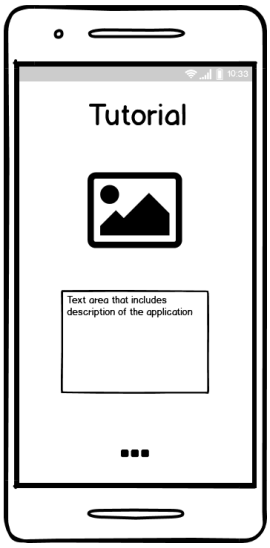


Figure 4.9: Tutorial Page

4.4 Application Navigation

The work-flow explained in Figure 4.10 displays how a user will navigate through the application. Though not every user requirement is shown in the figure, many of the main functions of application are shown which include the following:

1. Users can select the homepage of a recreation center and see the general information of the center such as hours of operation and contact information.

2. Users can search for a recreation center by using a valid zip code, the name of the recreation center, or using GPS functionality. Due to time constraints, we were not able to implement this section.

3. Users can select events, programs, or rooms to see their information and easily return to the recreation center page.

4. Users can easily navigate to the tutorial page to get help to use the application.

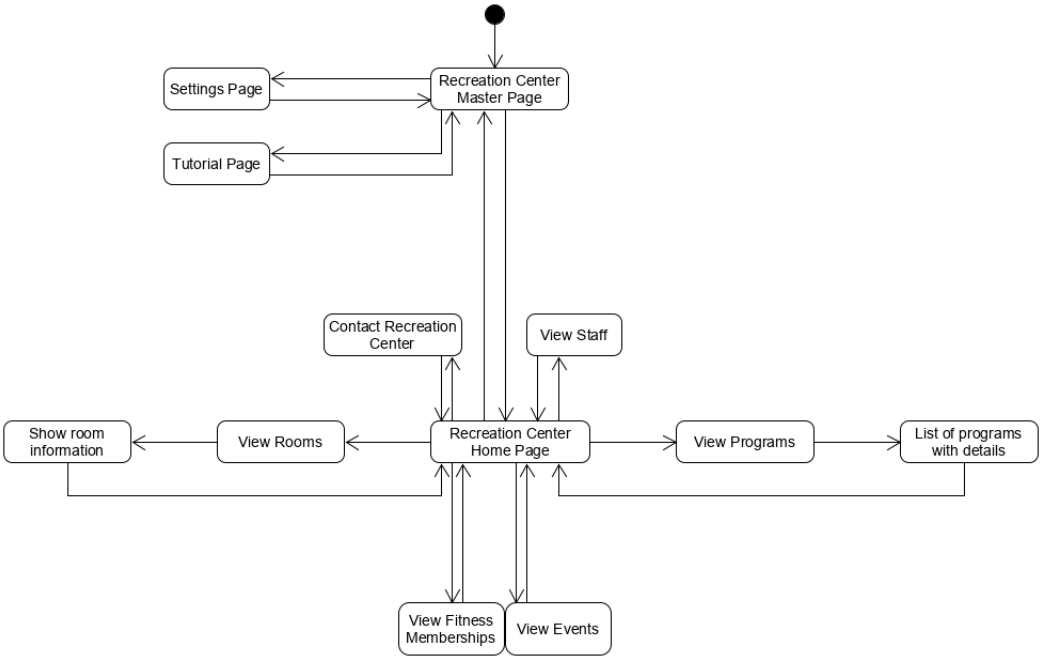


Figure 4.10: Applicaiton Navigation

4.5 Database Design

This section of the chapter will discuss the overall database design of the Parks and Recreation mobile application. The section covers all the steps taken to create the database along with descriptions of the team's work. This component of the overall application design represents the model portion of the Model-View-View-Model architecture.

4.5.1 Conceptual Model

As shown in figure 4.11 below, the purpose of the conceptual model is to show the the relationships and entities in the Parks and Recreation center mobile application database. Some of the relationships the team found was that each recreation center will have rooms for reservations, events being held, a gym membership, its programs, as well as its hours of operations. That way, users can find the ideal recreation center that fits their optimal priorities, all in one space.

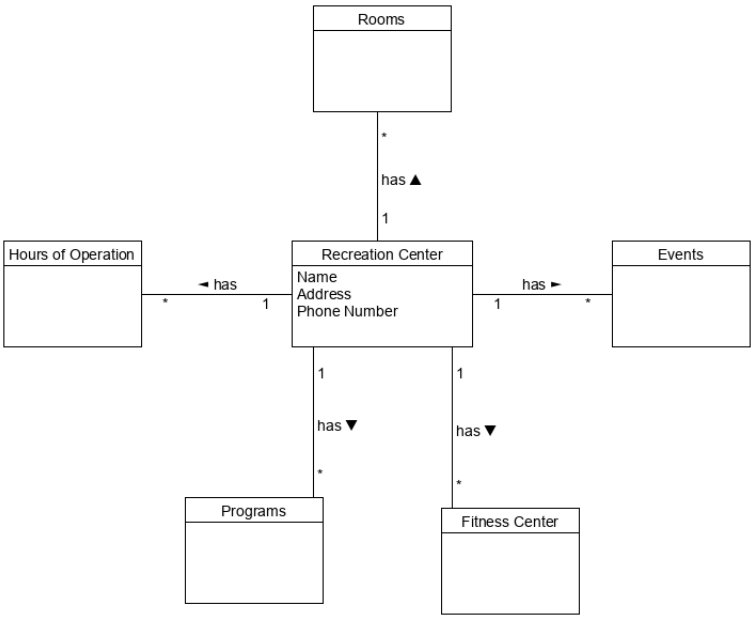


Figure 4.11: Conceptual Model

4.5.2 Logical Model

This model is based on the conceptual model. As shown in figure 4.12 in the logical model the foreign keys are implemented. A second primary key, date, was added to the Events entity. The many to many relationship between Hours of Operation and Recreation Center was resolved by a join table. This is due to the recreation centers having varied opening and closing hours on different days. An example is that Center A can have the same hours every day while Center B can have the same hours as Center A, but only on the weekends.

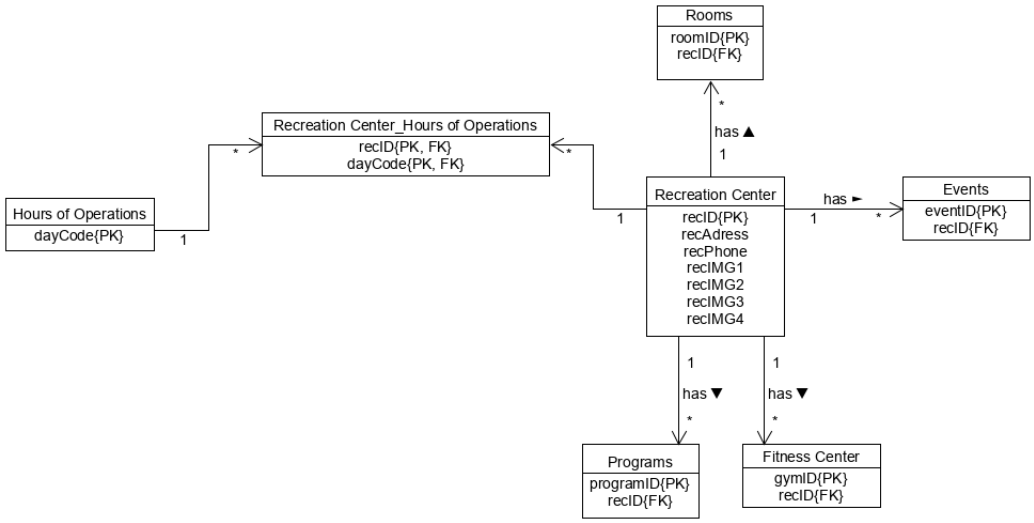


Figure 4.12: Logical Model

4.5.3 Physical Model

This model builds off the logical model. As shown in figure 4.13, in the physical model the attributes are added along with their respective data types. Attributes are the different characteristics for each of the different entities. For example, the table "Recreation Center" has the address attribute which is given the data type of varchar (60). Varchar is a data type that holds a certain number of characters which is indicated by the number within the parenthesis. Therefore, a database administrator can store up to sixty different characters in this field. The table "Staff" was also added during this step which also has a data type of varchar. This table will consist of the different staff members for a given recreation center. There is also a recursive relationship that can be noted. This is shown as an arrow that goes from one table and its destination is within itself. This means that one member of staff is also the manager that will manage the recreation center. Other data types that are used are int, blob, and double. The data type int is used when regular positive and negative numbers are being used. Blob is the data type of choice for our images and will be used to specify the file path to our images. Finally, double is the datatype that we will use for prices. This is because prices contain decimals, and the double data type will allow us to show that.

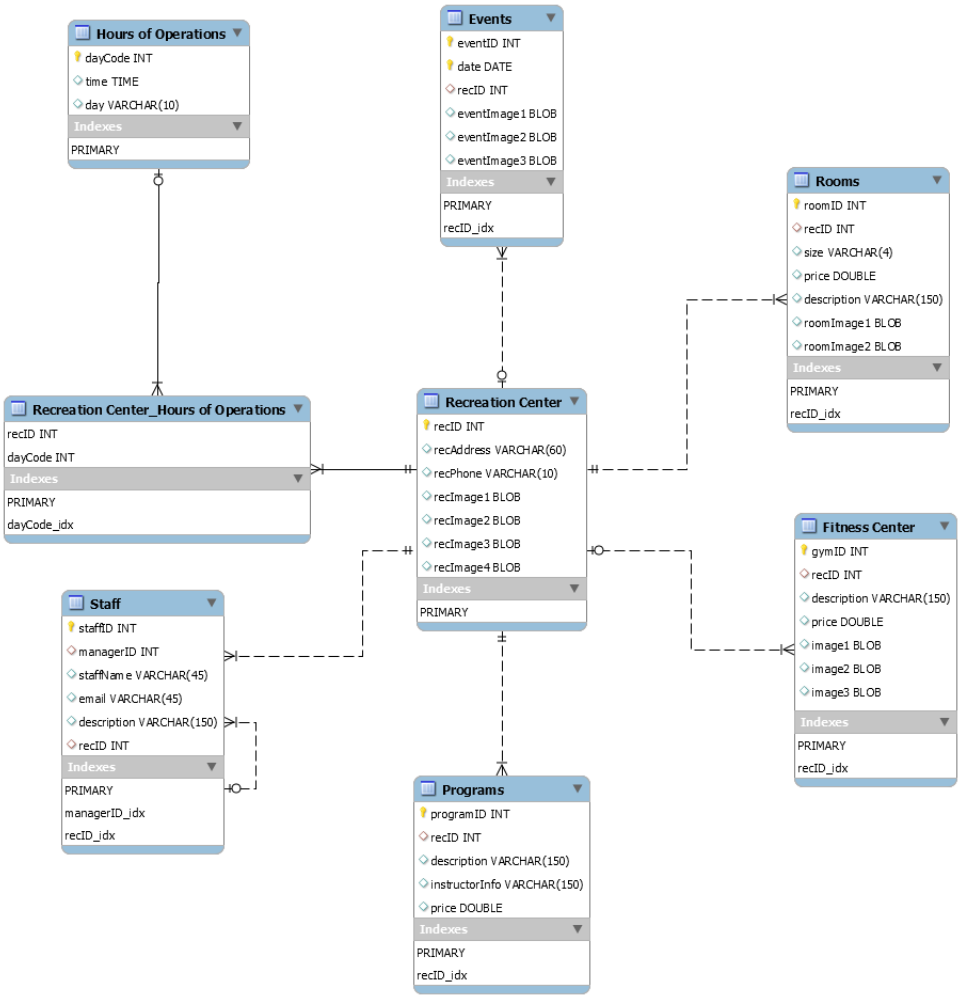


Figure 4.13: Physical Model

4.6 Summary

This chapter of the report discussed the architecture this application will be built upon (MVVM), the GUI's users will see while using the application, how users will navigate through the application, and the design of the database. The next chapter of the report will contain the source code for all parts of the Parks and Recreation mobile application with explanations.

Chapter 5

Implementation, Testing, and Evaluation

Objectives:

- Showcase the implementation of the proposed system.
- Provide the implemented graphical user interfaces for each feature.
- Provide the code behind each feature.
- Present testing scenarios to evaluate the system's performance.

5.1 Introduction

In this chapter, we will discuss the implementation of the solution for the problems that are identified in the application. In this project, the team has been conjuring solutions to multiple problems, which in turn has developed better methods, techniques, and tools in the process. A lot of earlier ideas were not implemented to the application, and of those that are, were changed considerably during the duration. Thus, the team have progressively come to realize that their task should not only include solving specific problems but also designing problem-solving and implementation systems that predict and prevent future problems, identify and solve current ones, and implement and maintain these solutions under changing conditions.

The following three sections will demonstrate how far the application has come in its current state. These sections will provide information on the User Interface, Database Design, and the System Testing. Visual aid is provided to make the team's vision clearer to the observer.

5.2 User Interface

This section will discuss the various User Interface (UI) in the Parks and Recreation Application. Images of the UIs will be accompanied with a brief description and details. This section will also include a brief guide on how users can easily and seamlessly interact with key aspects of the application.

The image below, figure 5.1, displays the homepage of the Park and Recreation app. The page comprises of the various Recreation Centers. For the meantime, the page has five centers. As we progress, we plan to add more centers. Each button in the page represents an individual center. Information about the Arcadia Recreation Center can be accessed by clicking the Arcadia button. Users can navigate to their desired center by clicking or tapping the button above the name of the center. To navigate to the previous page, users can use the triangle looking button at the bottom far left of the screen to navigate to the previous page at anytime. The centers all have identical UI structure and layout.

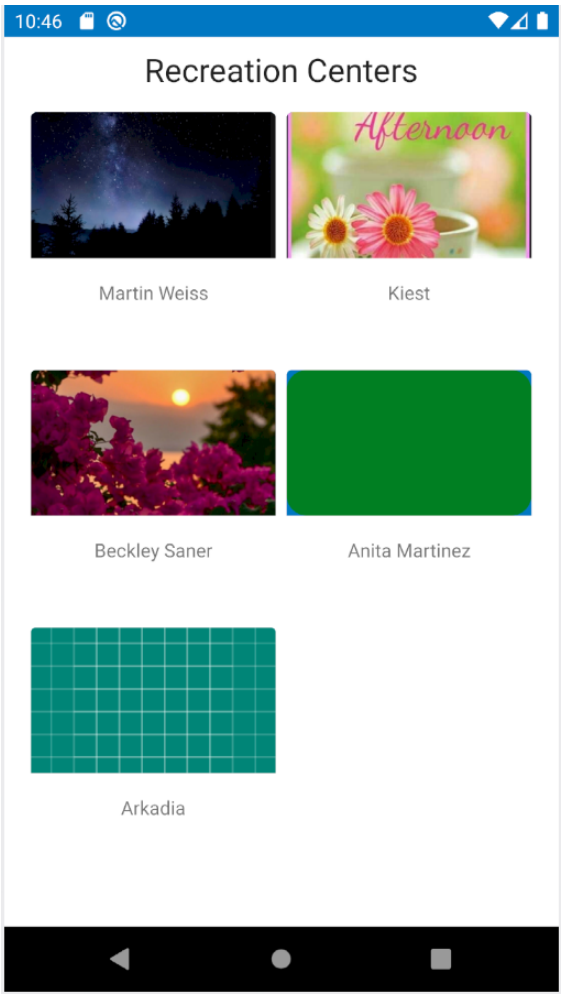
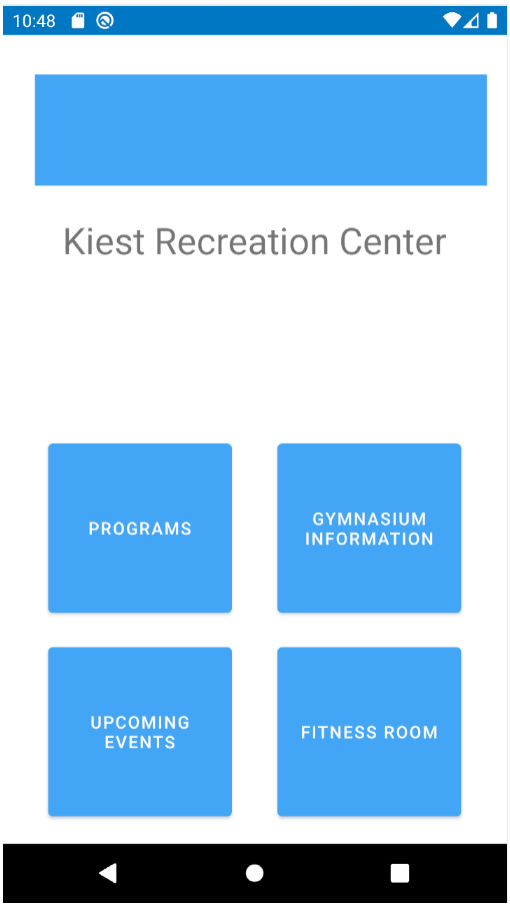


Figure 5.1: Home Page

Clicking or tapping a center from the homepage will direct users to the homepage of that center. The homepage will consist of various information about different facilities housed inside the Recreation Center. While the UIs of the various Recreation Centers might be the same, the facilities and programs available in each of them might defer slightly. Availability of programs and facilities in Recreation Centers are usually based on the needs of the surrounding communities.

The images below show the homepage of Kiest, figure 5.2a and Martin Weiss Recreation Centers, figure5.2b. The pages have similar layout. Both centers have Programs, Room Reservation, Upcoming Events, Park Information, Gymnasium Information, and Fitness Room buttons. Users looking for Program information can click on the Program button to view lists of Programs offered by the center. Similarly, Park or Event information can be accessed by clicking the corresponding button. All the centers will have the

similar UI as shown below. And as always, user can navigate to previous pages or screens by clicking the triangle looking button on the bottom far left.



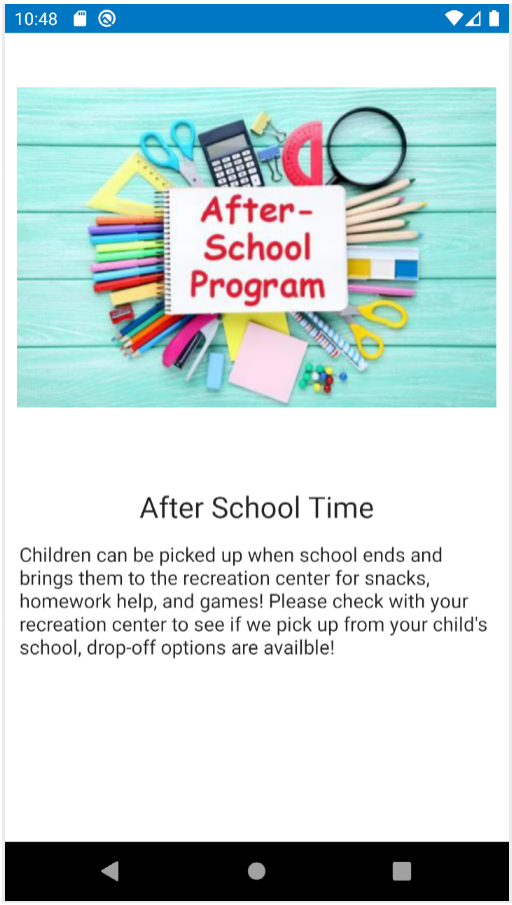
(a) Kiest

(b) Martin Weiss

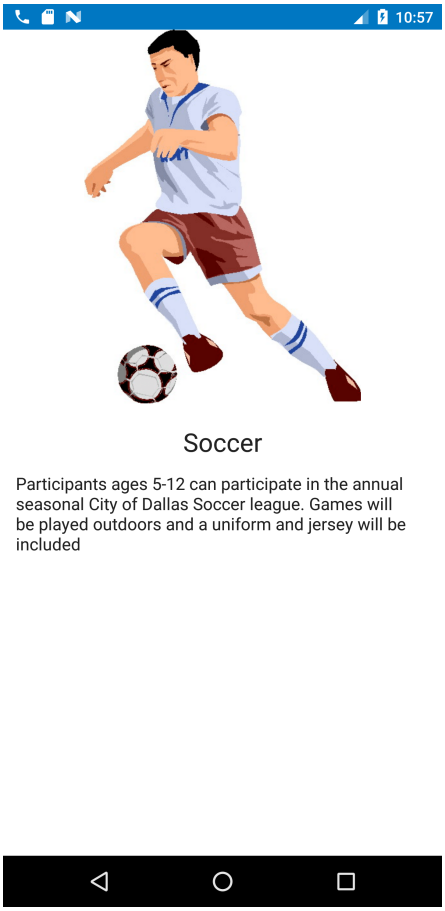
Figure 5.2: Recreation Centers

The images below show two programs offered in the Kiest and Martin Weiss Recreation centers. The pages are still under construction. When completed, users will be able to view pricing, hours of operation, instructor details as well as other useful information about the programs. To view details of programs and facilities offered in a particular Recreation Center, users can click on the corresponding button and swipe either right or left.

Programs offered at Kiest as shown in figures 5.3a and 5.3b.



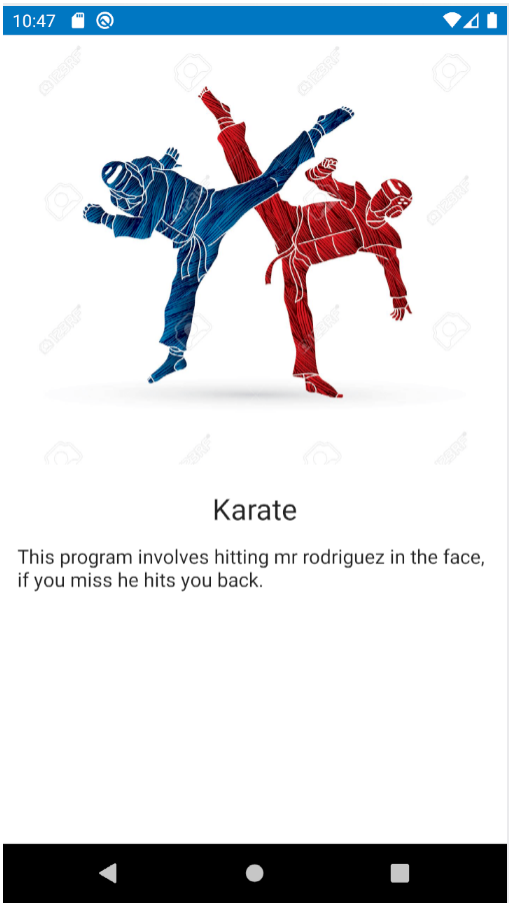
(a) After School



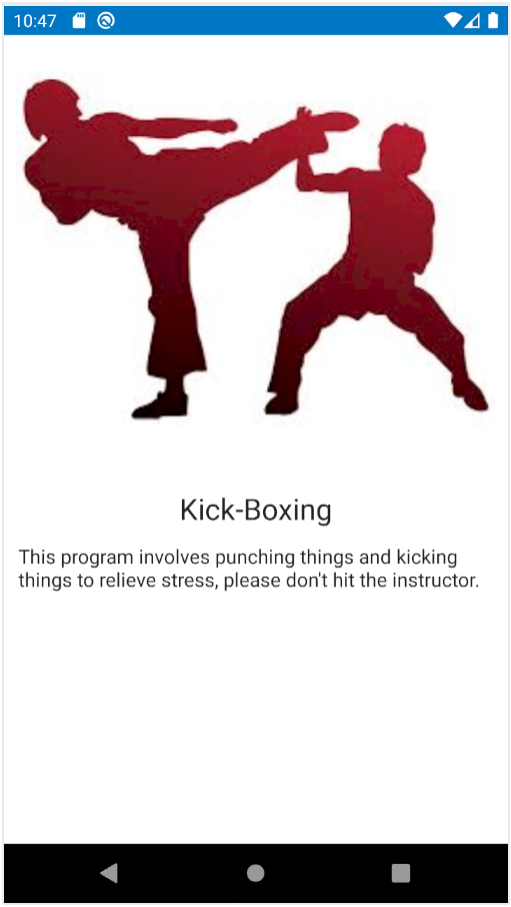
(b) Soccer

Figure 5.3: Programs at Kiest

Programs offered at Martin Weiss as shown in figures 5.4a and 5.4b.



(a) Karate



(b) Kickboxing

Figure 5.4: Programs at Martin Weiss

5.3 Implementation

Here is all the code we used to implement our system.

5.3.1 Loading Screen

This is the Kotlin code for implementing the loading screen.

```
package android.example.parkandrecreation.LoadScreen


import android.example.parkandrecreation.R
```

```
import android.example.parkandrecreation.databinding.FragmentLoadBinding
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.Navigation

/**
 * First fragment user will see when booting up the application.
 * Will ensure all components boot up correctly before moving on.
 */
class LoadFragment : Fragment() {

    lateinit var viewModel: LoadViewModel
    private var _binding: FragmentLoadBinding? = null
    private val binding get() = _binding!!


    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        //inflate the layout for this fragment
        _binding = FragmentLoadBinding.inflate(inflater, container, false)
        val view = binding.root

        //intitlize viewModel to separate UI logic from fragment interface
        viewModel = ViewModelProvider(this).get(LoadViewModel::class.java)

        //set picture of city depending on time.
        binding.cityImage.setImageResource(viewModel.drawableResource)

```

```
        binding.navButton.setOnClickListener(
            Navigation.createNavigateOnClickListener(R.id.action_loadFragment_to_homepageFragment)
        )
        return view
    }

    //clean up system resources
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}

package android.example.parkandrecreation.LoadScreen

import android.example.parkandrecreation.R
import androidx.lifecycle.ViewModel
import java.util.Calendar

/**
 * ViewModel to set image on Load screen depending on the time of day.
 */

class LoadViewModel : ViewModel() {

    private var currentTime = Calendar.HOUR_OF_DAY

    var drawableResource = when (currentTime){
        in 1..10 -> R.drawable.morning_image
        in 11..17 -> R.drawable.afternoon_image
        else -> R.drawable.night_image
    }
}
```

This is the XML code for implementing the loading screen.

```xml
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".LoadScreen.LoadFragment">

    <!-- Load screen fragment layout -->

    <ImageView
        android:id="@+id/city_image"
        android:layout_width="match_parent"
        android:layout_height="350dp"
        android:layout_marginBottom="10dp"/>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp">
    </RelativeLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <Button
            android:id="@+id/nav_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="Next"
            android:layout_centerInParent="true"/>
    </RelativeLayout>

</LinearLayout>
```

5.3.2 Home Page

This is the Kotlin code for implementing the home page.

```
package android.example.parkandrecreation.RecScreens.HomePage


import android.example.parkandrecreation.R
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView
import kotlinx.android.synthetic.main.fragment_homepage.view.*

/**
 * Home page to display card views for each recreation center, implements Recycler View for re-use
 * of cardviews for efficient code.
 */
class HomepageFragment : Fragment(){

    //lateinit var centers: ParkRecDatabase

    val MartinWeiss =
        RecCenters(
            "MW",
```

```
25            "Martin Weiss",
26            R.drawable.night_image
27        )
28    val Kiest = RecCenters(
29        "KP",
30        "Kiest",
31        R.drawable.afternoon_image)
32
33    val recreationList = listOf<RecCenters>(MartinWeiss, Kiest)
34
35    override fun onCreateView(
36        inflater: LayoutInflater, container: ViewGroup?,
37        savedInstanceState: Bundle?
38    ): View? {
39        //inflate the binding layout
40        val view = inflater.inflate(R.layout.fragment_homepage, container, false)
41
42        //Set up recycler view
43        view.recycler.setHasFixedSize(true)
44        view.recycler.layoutManager = GridLayoutManager(
45            context, 2, RecyclerView.VERTICAL, false)
46
47        val adapter =
48            HomePageRecyclerViewAdapter(
49                recreationList
50            )
51
52        //connect recyclerview component in layout file to adapter.
53        view.recycler.adapter = adapter
54        val largePadding = resources.getDimensionPixelSize(R.dimen.rec_grid_spacing)
55        val smallPadding = resources.getDimensionPixelSize(R.dimen.rec_grid_spacing_small)
56        view.recycler.addItemDecoration(
57            RecGridItemDecoration(
58                largePadding,
```

```
                smallPadding
            )
        )
        // Inflate the layout for this fragment
        return view
    }
}

package android.example.parkandrecreation.RecScreens.HomePage


import android.example.parkandrecreation.R
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.navigation.findNavController
import androidx.recyclerview.widget.RecyclerView

/**
 *
 */
class HomePageRecyclerViewAdapter internal constructor(private val recList: List<RecCenters>):
    RecyclerView.Adapter<RecCardViewHolder>(){

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecCardViewHolder {
        val layoutView = LayoutInflater.from(parent.context).inflate(R.layout.rec_cardview, parent,
            false)
        return RecCardViewHolder(
            layoutView
        )
    }

    override fun onBindViewHolder(holder: RecCardViewHolder, position: Int) {
        if(position < recList.size){
            val rec = recList[position]
```

```
            holder.recName.text = rec.recName
            holder.recImage.setImageResource(rec.recImage)
            holder.itemView.setOnClickListener{
                it.findNavController().navigate(
                    HomepageFragmentDirections.actionHomepageFragmentToRecreationCenterFragment(
                        rec.recID
                    )
                )
            }
        }
    }

    override fun getItemCount(): Int {
        return recList.size
    }
}

package android.example.parkandrecreation.RecScreens.HomePage

import android.example.parkandrecreation.R
import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

class RecCardViewHolder(itemView: View): RecyclerView.ViewHolder(itemView){
    var recName: TextView = itemView.findViewById(R.id.rec_name)
    var recImage: ImageView = itemView.findViewById(R.id.rec_image)
}

package android.example.parkandrecreation.RecScreens.HomePage

data class RecCenters(
    val recID: String,
```

```
    val recName: String,
    val recImage: Int
)

package android.example.parkandrecreation.RecScreens.HomePage

import android.graphics.Rect
import android.view.View
import androidx.recyclerview.widget.RecyclerView

/**
 * Sets the padding for each card view in the recycler view.
 */

class RecGridItemDecoration(private val largePadding: Int, private val smallPadding: Int) :
    RecyclerView.ItemDecoration() {

    override fun getItemOffsets(outRect: Rect, view: View,
                                parent: RecyclerView, state: RecyclerView.State) {
        outRect.left = smallPadding
        outRect.right = smallPadding
        outRect.top = largePadding
        outRect.bottom = largePadding
    }
}
```

This is the XML code for implementing the home screen.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    tools:context=".RecScreens.HomePage.HomepageFragment">

    <!-- Layout used to display various recreation centers on the screen, using recycler view
    we can display multiple instances of a recreation center using the card view layout -->

    <TextView
        android:layout_width="match_parent"
        style="@style/item_name"
        android:layout_height="wrap_content"
        android:text="@string/rec_title_homepage"
        android:layout_marginTop="8dp"
        android:textAlignment="center"/>

    <androidx.core.widget.NestedScrollView
        android:id="@+id/nestedScrollView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="40dp"
        android:layout_marginBottom="8dp"
        android:paddingStart="@dimen/shr_product_grid_spacing"
        android:paddingEnd="@dimen/shr_product_grid_spacing">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recycler"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

    </androidx.core.widget.NestedScrollView>

</FrameLayout>
```

5.3.3 Recreation Center

This is the Kotlin code for implementing the Recreation Center.

```
package android.example.parkandrecreation.RecScreens

/**
 * Data class used to hold one instance of a recreation center from the database
 */

data class RecreationCenter(
    val centerName: String,
    val centerNumber: String,
    val centerOpening: String,
    val centerClosing: String,
    val centerImageOne: Int,
    val centerImageTwo: Int,
    val centerImageThree: Int,
    val centerImageFour: Int
)

package android.example.parkandrecreation.RecScreens


import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentRecreationCenterBinding
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.ViewModelProvider
import androidx.navigation.Navigation
import androidx.navigation.findNavController
import androidx.navigation.fragment.navArgs
```

```
32
33 /**
34  * Recreation center page that is used constantly for different recreation centers.
35  */
36 class RecreationCenterFragment : Fragment() {
37
38     private lateinit var viewModel: RecreationCenterViewModel
39     private lateinit var viewModelFactory: RecreationCenterViewModelFactory
40
41
42     override fun onCreateView(
43         inflater: LayoutInflater, container: ViewGroup?,
44         savedInstanceState: Bundle?
45     ): View? {
46         // Inflate the layout for this fragment
47         val binding = FragmentRecreationCenterBinding.inflate(layoutInflater)
48
49         //get recKey from previous fragment depending on what was clicked.
50         val args = RecreationCenterFragmentArgs.fromBundle(arguments!!)
51         val recKey = args.recID
52
53         //create ViewModel to display correct page name
54         viewModelFactory = RecreationCenterViewModelFactory(recKey)
55         viewModel = ViewModelProvider(this, viewModelFactory)
56             .get(RecreationCenterViewModel::class.java)
57
58         //set recTitle depending on ViewModel.
59         binding.recTitle.text = viewModel.recText
60
61
62         binding.programButton.setOnClickListener{
63             it.findNavController().navigate(RecreationCenterFragmentDirections.
64                   actionRecreationCenterFragmentToProgramFragment(recKey))
65         }
```

```
        binding.gymButton.setOnClickListener{
            it.findNavController().navigate(RecreationCenterFragmentDirections.
                actionRecreationCenterFragmentToGymFragment(recKey))
        }

        binding.eventButton.setOnClickListener{
            it.findNavController().navigate(RecreationCenterFragmentDirections.
                actionRecreationCenterFragmentToEventsFragment(recKey))
        }

        binding.fitnessButton.setOnClickListener{
            it.findNavController().navigate(RecreationCenterFragmentDirections.
                actionRecreationCenterFragmentToFitnessFragment(recKey))
        }

        binding.roomButton.setOnClickListener{
            it.findNavController().navigate(RecreationCenterFragmentDirections.
                actionRecreationCenterFragmentToRoomFragment(recKey))
        }

        binding.martinParkButton.setOnClickListener{
            it.findNavController().navigate(RecreationCenterFragmentDirections.
                actionRecreationCenterFragmentToParkFragment(recKey))
        }

        return binding.root
    }
}

package android.example.parkandrecreation.RecScreens

import android.example.parkandrecreation.RecScreens.RecreationCenterFragmentArgs
import androidx.lifecycle.ViewModel
```

```
/**
 * ViewModel used to set recreation title on RecreationCenterFragment
 */

class RecreationCenterViewModel(recKey: String) : ViewModel(){

     var recText: String = ""

    //initialize recreation center title text
    init {
        when (recKey){
            "MW" -> recText = "Martin Weiss Recreation Center"
            "KP" -> recText = "Kiest Recreation Center"
            "BK" -> recText = "Beckley Saner Recreation Center"
            "AM" -> recText = "Anita Martinez Recreation Center"
            "AK" -> recText = "Arcadia Recreation Center"
            else -> recText = "Invalid Recreation Center"
        }
    }
}

package android.example.parkandrecreation.RecScreens

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import java.lang.IllegalArgumentException

/**
 * ViewModel Factory that creates viewModels of type RecreationCenterViewModel, takes in recKey
 * argument.
 */

class RecreationCenterViewModelFactory(private val recKey: String): ViewModelProvider.Factory{
```

```
    override fun <T: ViewModel?> create(modelClass: Class<T>): T{
        if(modelClass.isAssignableFrom(RecreationCenterViewModel::class.java)){
            return RecreationCenterViewModel(recKey) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

This is the XML code for implementing the recreation center.

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.RecreationCenterFragment">

    <ImageView
        android:id="@+id/rec_image"
        android:layout_width="369dp"
        android:layout_height="91dp"
        android:layout_marginTop="32dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toTopOf="@+id/nestedScrollView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.619"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@color/colorPrimary" />

    <androidx.core.widget.NestedScrollView
```

```
24          android:id="@+id/nestedScrollView"
25          android:layout_width="0dp"
26          android:layout_height="0dp"
27          android:fillViewport="true"
28          app:layout_constraintBottom_toBottomOf="parent"
29          app:layout_constraintEnd_toEndOf="parent"
30          app:layout_constraintHorizontal_bias="0.0"
31          app:layout_constraintStart_toStartOf="parent"
32          app:layout_constraintTop_toBottomOf="@+id/rec_image"
33          app:layout_constraintVertical_bias="0.94">
34
35          <androidx.constraintlayout.widget.ConstraintLayout
36              android:layout_width="match_parent"
37              android:layout_height="wrap_content">
38
39
40              <TextView
41                  android:id="@+id/rec_title"
42                  android:layout_width="0dp"
43                  android:layout_height="wrap_content"
44                  android:layout_marginTop="16dp"
45                  android:text="@string/martin_weiss_title"
46                  android:textAlignment="center"
47                  android:textSize="30sp"
48                  app:layout_constraintEnd_toEndOf="parent"
49                  app:layout_constraintStart_toStartOf="parent"
50                  app:layout_constraintTop_toTopOf="parent" />
51
52              <com.google.android.material.button.MaterialButton
53                  android:id="@+id/program_button"
54                  android:layout_width="150dp"
55                  android:layout_height="150dp"
56                  android:layout_marginStart="16dp"
57                  android:layout_marginTop="140dp"
```

```
58                 android:layout_marginEnd="8dp"
59                 android:text="@string/program_button_text"
60                 app:layout_constraintEnd_toStartOf="@+id/gym_button"
61                 app:layout_constraintStart_toStartOf="parent"
62                 app:layout_constraintTop_toBottomOf="@+id/rec_title" />
63
64             <Button
65                 android:id="@+id/gym_button"
66                 android:text="@string/gym_button_text"
67                 android:layout_width="150dp"
68                 android:layout_height="150dp"
69                 android:layout_marginStart="8dp"
70                 android:layout_marginTop="140dp"
71                 android:layout_marginEnd="16dp"
72                 app:layout_constraintEnd_toEndOf="parent"
73                 app:layout_constraintHorizontal_bias="0.558"
74                 app:layout_constraintStart_toEndOf="@+id/program_button"
75                 app:layout_constraintTop_toBottomOf="@+id/rec_title" />
76
77             <Button
78                 android:id="@+id/event_button"
79                 android:layout_width="150dp"
80                 android:layout_height="150dp"
81                 android:layout_marginStart="16dp"
82                 android:layout_marginTop="16dp"
83                 android:layout_marginEnd="8dp"
84                 android:text="@string/events_button_text"
85                 app:layout_constraintEnd_toStartOf="@+id/fitness_button"
86                 app:layout_constraintStart_toStartOf="parent"
87                 app:layout_constraintTop_toBottomOf="@+id/program_button" />
88
89             <Button
90                 android:id="@+id/fitness_button"
91                 android:layout_width="150dp"
```

```
92              android:layout_height="150dp"
93              android:layout_marginStart="8dp"
94              android:layout_marginTop="16dp"
95              android:layout_marginEnd="16dp"
96              android:text="@string/fitness_room_button_text"
97              app:layout_constraintEnd_toEndOf="parent"
98              app:layout_constraintStart_toEndOf="@+id/event_button"
99              app:layout_constraintTop_toBottomOf="@+id/gym_button" />
100
101          <Button
102              android:id="@+id/room_button"
103              android:layout_width="150dp"
104              android:layout_height="150dp"
105              android:layout_marginStart="16dp"
106              android:layout_marginTop="16dp"
107              android:layout_marginEnd="8dp"
108              android:layout_marginBottom="8dp"
109              android:text="@string/reservation_room_button_text"
110              app:layout_constraintBottom_toBottomOf="parent"
111              app:layout_constraintEnd_toStartOf="@+id/martin_park_button"
112              app:layout_constraintStart_toStartOf="parent"
113              app:layout_constraintTop_toBottomOf="@+id/event_button" />
114
115          <Button
116              android:id="@+id/martin_park_button"
117              android:layout_width="150dp"
118              android:layout_height="150dp"
119              android:layout_marginStart="8dp"
120              android:layout_marginTop="16dp"
121              android:layout_marginEnd="16dp"
122              android:layout_marginBottom="8dp"
123              android:text="@string/park_button_text"
124              app:layout_constraintBottom_toBottomOf="parent"
125              app:layout_constraintEnd_toEndOf="parent"
```

```
            app:layout_constraintStart_toEndOf="@+id/room_button"
            app:layout_constraintTop_toBottomOf="@+id/fitness_button" />

        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.core.widget.NestedScrollView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

5.3.4 Fitness Room

This is the Kotlin code for implementing the fitness room.

```
package android.example.parkandrecreation.RecScreens.FitnessRoom

import androidx.fragment.app.Fragment
import androidx.viewpager2.adapter.FragmentStateAdapter

/**
 * Adapter that initilzes multiple fragments of "FitnessDisplayFragment depending on amount of
 * informaiton in the database.
 */

class FitnessAdapter(fragment: Fragment): FragmentStateAdapter(fragment){

    val fitnessList = listOf<FitnessRoom>()

    override fun getItemCount(): Int = fitnessList.size

    override fun createFragment(position: Int): Fragment {
        val fragment = FitnessDisplayFragment()

        return fragment
    }
}

```

```
ackage android.example.parkandrecreation.RecScreens.FitnessRoom

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentFitnessDisplayBinding

/**
 * Fragment for FitnesFragment that will be used to display the information on a page. This
 * fragment in instantiated from the FragmentAdapter.
 */

class FitnessDisplayFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentFitnessDisplayBinding.inflate(layoutInflater)

        return binding.root
    }

}

package android.example.parkandrecreation.RecScreens.FitnessRoom

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentFitnessBinding
```

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.viewpager2.widget.ViewPager2

/**
 * Fragment for hte FitnessDisplay Fragment. This page is empty, the Fragment Adapter will insert
 * information on the page during runntime.
 */
class FitnessFragment : Fragment() {

    lateinit var fitnessAdapter: FitnessAdapter
    lateinit var viewPager: ViewPager2

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentFitnessBinding.inflate(layoutInflater)

        return binding.root
    }

    //initialize the viewPager to the fragment adapter.
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        fitnessAdapter = FitnessAdapter(this)
        viewPager = view.findViewById(R.id.fitness_pager)
        viewPager.adapter = fitnessAdapter
    }

}
```

```

package android.example.parkandrecreation.RecScreens.FitnessRoom

/**
 * Data class for one instance of a fitness room from the database
 */
data class FitnessRoom(
    val fitnessName: String,
    val fitnessInfo: String,
    val fitnessPrice: String,
    val fitnessImageOne: Int,
    val fitnessImageTwo: Int,
    val fitnessImageThree: Int
)
```

This is the XML code for implementing the fitness room.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.FitnessRoom.FitnessFragment">

    <!-- View Pager for fitness Fragment -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/fitness_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
18      xmlns:tools="http://schemas.android.com/tools"
19      android:layout_width="match_parent"
20      android:layout_height="match_parent"
21      tools:context=".RecScreens.FitnessRoom.FitnessDisplayFragment">
22
23      <!--Layout to display information about an instance of a fitness room -->
24
25      <LinearLayout
26          android:layout_width="match_parent"
27          android:layout_height="match_parent"
28          android:orientation="vertical">
29
30          <ImageView
31              android:id="@+id/fitness_image"
32              android:layout_width="match_parent"
33              android:layout_height="350dp"
34              android:layout_marginStart="10dp"
35              android:layout_marginEnd="10dp"
36              tools:srcCompat="@tools:sample/backgrounds/scenic" />
37
38          <TextView
39              android:id="@+id/fitness_name"
40              style="@style/item_name"
41              android:layout_width="match_parent"
42              android:layout_height="wrap_content"
43              android:layout_marginTop="20dp"
44              android:textAlignment="center"
45              tools:text="Program Name" />
46
47          <RelativeLayout
48              android:layout_width="match_parent"
49              android:layout_height="wrap_content"
50              android:layout_marginStart="5dp"
51              android:layout_marginEnd="5dp"
```

```
            android:layout_margin="8dp"
            android:layout_marginBottom="8dp">

            <androidx.core.widget.NestedScrollView
                android:layout_width="match_parent"
                android:layout_height="match_parent">

                <TextView
                    android:id="@+id/fitness_description"
                    style="@style/item_description"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    tools:text="@string/item_description"
                    android:padding="4dp"/>

            </androidx.core.widget.NestedScrollView>
        </RelativeLayout>
    </LinearLayout>
</FrameLayout>
```

5.3.5 Gymnasium

This is the Kotlin code for implementing the gymnasium.

```
package android.example.parkandrecreation.RecScreens.Gymnasiums

/**
 * Data class for one instance of a Gym from the database.
 */

data class Gym(
    val gymName: String,
    val gymInfo: String,
    val gymImageOne: Int,
```

```
    val gymImageTwo: Int,
    val gymImageThree:Int
)

package android.example.parkandrecreation.RecScreens.Gymnasiums

import androidx.fragment.app.Fragment
import androidx.viewpager2.adapter.FragmentStateAdapter

/**
 * Fragment State Adapter used to create fragments based on number of instances of a gym
 * in the database
 */

class GymAdapter(fragment: Fragment): FragmentStateAdapter(fragment){

    override fun getItemCount(): Int {
        TODO("not implemented") //To change body of created functions use File | Settings | File
            Templates.
    }

    override fun createFragment(position: Int): Fragment {
        TODO("not implemented") //To change body of created functions use File | Settings | File
            Templates.
    }
}

package android.example.parkandrecreation.RecScreens.Gymnasiums

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
```

```
43
44 import android.example.parkandrecreation.R
45 import android.example.parkandrecreation.databinding.FragmentGymDisplayBinding
46
47 /**
48  * Fragment created by GymAdapter that is displayed in the viewPager inside GymFragment
49  */
50 class GymDisplayFragment : Fragment() {
51
52     override fun onCreateView(
53         inflater: LayoutInflater, container: ViewGroup?,
54         savedInstanceState: Bundle?
55     ): View? {
56         // Inflate the layout for this fragment
57         val binding = FragmentGymDisplayBinding.inflate(layoutInflater)
58
59         return binding.root
60     }
61
62 }
63
64 package android.example.parkandrecreation.RecScreens.Gymnasiums
65
66 import android.example.parkandrecreation.databinding.FragmentGymBinding
67 import android.os.Bundle
68 import androidx.fragment.app.Fragment
69 import android.view.LayoutInflater
70 import android.view.View
71 import android.view.ViewGroup
72
73 /**
74  * Simple fragment that holds the view pager that will display multiple fragments
75  */
76 class GymFragment : Fragment() {
```

```

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentGymBinding.inflate(layoutInflater)

        return binding.root
    }

}
```

This is the XML code for implementing the gymnasium.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.Gymnasiums.GymFragment">

    <!-- View Pager for Gym Fragment -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/gym_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".RecScreens.Gymnasiums.GymDisplayFragment">

    <!-- Layout to display instance of a gym-->


</FrameLayout>
```

5.3.6 Park Information

This is the Kotlin code for implementing the park information.

```
package android.example.parkandrecreation.RecScreens.ParkInformation

data class Park(
    val parkName: String,
    val parkInfo: String,
    val parkImageOne: Int,
    val parkImageTwo: Int,
    val parkImageThree: Int
)

package android.example.parkandrecreation.RecScreens.ParkInformation

import androidx.fragment.app.Fragment
import androidx.viewpager2.adapter.FragmentStateAdapter

class ParkAdapter(fragment: Fragment): FragmentStateAdapter(fragment){

    override fun getItemCount(): Int = 1

    override fun createFragment(position: Int): Fragment {
        return ParkDisplayFragment()
    }
```

```
}

package android.example.parkandrecreation.RecScreens.ParkInformation

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentParkDisplayBinding

/**
 * A simple [Fragment] subclass.
 */
class ParkDisplayFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentParkDisplayBinding.inflate(layoutInflater)

        return binding.root
    }

}

package android.example.parkandrecreation.RecScreens.ParkInformation

import android.example.parkandrecreation.R
import android.example.parkandrecreation.RecScreens.Rooms.RoomAdapter
```

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.example.parkandrecreation.databinding.FragmentParkBinding
import androidx.viewpager2.widget.ViewPager2

/**
 * Simple fragment that holds the view pager that will display multiple fragments
 */
class ParkFragment : Fragment() {

    lateinit var roomAdapter: RoomAdapter
    lateinit var viewPager: ViewPager2

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentParkBinding.inflate(layoutInflater)

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        roomAdapter = RoomAdapter(this)
        viewPager = view.findViewById(R.id.park_pager)
        viewPager.adapter = roomAdapter
    }

}
```

This is the XML code for implementing the park information.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.ParkInformation.ParkFragment">

    <!--View Pager for park fragment -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/park_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.ParkInformation.ParkDisplayFragment">

    <!-- Layout used to display information about an instance of a park. -->

</FrameLayout>
```

5.3.7 Programs

This is the Kotlin code for implementing the programs.

```kotlin
package android.example.parkandrecreation.RecScreens.ProgramScreens

/**
```

```
4  * Data class Program, each class will be one instance of "program" from the database.
5  */
6
7  data class Program(
8      val programName: String,
9      val programInfo: String,
10     val programPrice: String,
11     val ProgramImageOne: Int,
12     val ProgramImageTwo: Int,
13     val ProgramImageThree: Int,
14     val ProgramImageFour: Int
15 )
16
17 package android.example.parkandrecreation.RecScreens.ProgramScreens
18
19 import android.example.parkandrecreation.R
20 import android.os.Bundle
21 import androidx.fragment.app.Fragment
22 import androidx.viewpager2.adapter.FragmentStateAdapter
23
24 class ProgramAdapter(fragment: Fragment, recKey: String): FragmentStateAdapter(fragment){
25
26
27
28     val programs = getProgram(recKey)
29
30     override fun getItemCount(): Int = programs.size
31
32     //create fragments of type ProgramDisplayFragment based on amount of items in "programs"
33     override fun createFragment(position: Int): Fragment {
34         val fragment = ProgramDisplayFragment()
35
36         if(position < programs.size){
37             val program = programs[position]
```

```
38          fragment.arguments = Bundle().apply{
39              putString("name", program.programName)
40              putString("info", program.programInfo)
41              putInt("image", program.ProgramImageOne)
42          }
43      }
44      return fragment
45  }
46
47  /*
48  This function puts information into the list to allow the adapter to display some
49  information on the screen. Since the database was not working this was the best that could be
50  done. However future versions of the application will remove this.
51   */
52  fun getProgram(recID: String): List<Program>{
53      val programList: MutableList<Program> = mutableListOf()
54
55      when(recID){
56          "MW" ->{
57              val exProgram: Program = Program(
58          "Kick-Boxing",
59          "This program involves punching things and kicking things to relieve " +
60                 "stress, please don't hit the instructor.",
61          "10 Dollars",
62          R.drawable.kickboxing,
63          R.drawable.afternoon_image,
64          R.drawable.morning_image,
65          R.drawable.rec_program_button)
66
67              val exProgram2 = Program(
68              "Karate",
69              "This program involves hitting mr rodriguez in the face, if you miss " +
70                     "he hits you back. ",
71              "20 Dollars",
```

```
            R.drawable.karate_picture1,
            R.drawable.afternoon_image,
            R.drawable.morning_image,
            R.drawable.night_image)

            programList.add(exProgram)
            programList.add(exProgram2)
            return programList
        }

        "KP" ->{
            val exProgram: Program = Program(
                "After School Time",
                "Children can be picked up when school ends and brings them to " +
                        "the recreation center for snacks, homework help, and games! Please " +
                        "check with your recreation center to see if we pick up from your " +
                        "child's school, drop-off options are availble!",
                "25",
                R.drawable.afterschool,
                R.drawable.painting,
                R.drawable.painting,
                R.drawable.painting)
            val exProgram2: Program = Program(
                "Soccer",
                "Participants ages 5-12 can participate in the annual seasonal " +
                        "City of Dallas Soccer league. Games will be played outdoors and a " +
                        "uniform and jersey will be included",
                "40",
                R.drawable.soccer,
                R.drawable.painting,
                R.drawable.painting,
                R.drawable.painting)

            programList.add(exProgram)
```

```
                programList.add(exProgram2)
                return programList
            }

            else ->{
                val invalidProgram: Program =Program(
                    "Empty Program",
                    "Empty Program",
                    "0",
                    R.drawable.morning_image,
                    R.drawable.painting,
                    R.drawable.painting,
                    R.drawable.painting
                )

                programList.add(invalidProgram)
                return programList
            }


        }

    }
}

package android.example.parkandrecreation.RecScreens.ProgramScreens

import android.example.parkandrecreation.databinding.FragmentProgramDisplayBinding
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

```

```
/**
 * Fragment that is initialized by the program adapter.
 */
class ProgramDisplayFragment : Fragment() {

    lateinit var binding: FragmentProgramDisplayBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        binding = FragmentProgramDisplayBinding.inflate(layoutInflater)

        return binding.root
    }

    //set information from ProgramAdapter to display on the screen for this fragment.
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        arguments?.takeIf { it.containsKey("name") }?.apply{
            binding.programName.text = getString("name")
            binding.programDescription.text = getString("info")
            binding.programImage.setImageResource(getInt("image"))
        }
    }

}

package android.example.parkandrecreation.RecScreens.ProgramScreens


import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentProgramBinding
import androidx.viewpager2.widget.ViewPager2

/**
 * Simple fragment that holds the view pager that will display multiple fragments
 */
class ProgramFragment : Fragment() {

    lateinit var programAdapter: ProgramAdapter
    lateinit var viewPager: ViewPager2

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentProgramBinding.inflate(layoutInflater)
        return binding.root
    }

    //initialize the viewPager component to the programAdapter.
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

        val key = ProgramFragmentArgs.fromBundle(arguments!!).recID
        programAdapter = ProgramAdapter(this, key)
        viewPager = view.findViewById(R.id.program_pager)
        viewPager.adapter = programAdapter
    }

}
```

This is the XML code for implementing the programs.

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.ProgramScreens.ProgramFragment">

    <!--View Pager for program Fragment -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/program_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </androidx.viewpager2.widget.ViewPager2>
</androidx.constraintlayout.widget.ConstraintLayout>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.ProgramScreens.ProgramDisplayFragment">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

```

```
31        <ImageView
32            android:id="@+id/program_image"
33            android:layout_width="match_parent"
34            android:layout_height="350dp"
35            android:layout_marginStart="10dp"
36            android:layout_marginEnd="10dp"
37            tools:srcCompat="@tools:sample/backgrounds/scenic" />
38
39        <TextView
40            android:id="@+id/program_name"
41            style="@style/item_name"
42            android:layout_width="match_parent"
43            android:layout_height="wrap_content"
44            android:layout_marginTop="20dp"
45            android:textAlignment="center"
46            tools:text="Program Name" />
47
48        <RelativeLayout
49            android:layout_width="match_parent"
50            android:layout_height="wrap_content"
51            android:layout_marginStart="5dp"
52            android:layout_marginEnd="5dp"
53            android:layout_margin="8dp"
54            android:layout_marginBottom="8dp">
55
56            <androidx.core.widget.NestedScrollView
57                android:layout_width="match_parent"
58                android:layout_height="match_parent">
59
60                <TextView
61                    android:id="@+id/program_description"
62                    style="@style/item_description"
63                    android:layout_width="match_parent"
64                    android:layout_height="wrap_content"
```

```
                    tools:text="@string/item_description"
                    android:padding="4dp"/>

            </androidx.core.widget.NestedScrollView>
        </RelativeLayout>
    </LinearLayout>
</FrameLayout>
```

5.3.8 Rooms

This is the Kotlin code for implementing the rooms.

```
package android.example.parkandrecreation.RecScreens.Rooms

data class Room(
    val roomName: String,
    val roomInfo: String,
    val roomPrice: String,
    val RoomImageOne: Int,
    val RoomImageTwo: Int,
    val RoomImageThree: Int
)

package android.example.parkandrecreation.RecScreens.Rooms

import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentStatePagerAdapter
import androidx.viewpager2.adapter.FragmentStateAdapter

class RoomAdapter(fragment: Fragment): FragmentStateAdapter(fragment){
    override fun getItemCount(): Int = 1

    override fun createFragment(position: Int): Fragment {
        return RoomDisplayFragment()
```

```
23     }
24 }
25
26 package android.example.parkandrecreation.RecScreens.Rooms
27
28 import android.os.Bundle
29 import androidx.fragment.app.Fragment
30 import android.view.LayoutInflater
31 import android.view.View
32 import android.view.ViewGroup
33
34 import android.example.parkandrecreation.R
35 import android.example.parkandrecreation.databinding.FragmentRoomDisplayBinding
36
37 /**
38  *
39  */
40 class RoomDisplayFragment : Fragment() {
41
42     lateinit var binding: FragmentRoomDisplayBinding
43     override fun onCreateView(
44         inflater: LayoutInflater, container: ViewGroup?,
45         savedInstanceState: Bundle?
46     ): View? {
47         // Inflate the layout for this fragment
48         binding = FragmentRoomDisplayBinding.inflate(layoutInflater)
49
50         return binding.root
51     }
52
53 }
54
55 package android.example.parkandrecreation.RecScreens.Rooms
56
```

```
57 import android.example.parkandrecreation.R
58 import android.example.parkandrecreation.databinding.FragmentRoomsBinding
59 import android.os.Bundle
60 import androidx.fragment.app.Fragment
61 import android.view.LayoutInflater
62 import android.view.View
63 import android.view.ViewGroup
64 import androidx.viewpager2.widget.ViewPager2
65
66 /**
67  * Simple fragment that holds the view pager that will display multiple fragments.
68  */
69 class RoomFragment : Fragment() {
70
71     lateinit var roomAdapter: RoomAdapter
72     lateinit var viewPager: ViewPager2
73
74     override fun onCreateView(
75         inflater: LayoutInflater, container: ViewGroup?,
76         savedInstanceState: Bundle?
77     ): View? {
78         // Inflate the layout for this fragment
79         val binding = FragmentRoomsBinding.inflate(layoutInflater)
80
81         return binding.root
82     }
83
84     override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
85         roomAdapter = RoomAdapter(this)
86         viewPager = view.findViewById(R.id.room_pager)
87         viewPager.adapter = roomAdapter
88     }
89
90 }
```

This is the XML code for implementing the rooms.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.Rooms.RoomFragment">

   <!--View Pager 2 for room fragment -->
   <androidx.viewpager2.widget.ViewPager2
       android:id="@+id/room_pager"
       android:layout_width="match_parent"
       android:layout_height="match_parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".RecScreens.Rooms.RoomDisplayFragment">


    <ImageView
        android:id="@+id/room_image"
        android:layout_width="match_parent"
        android:layout_height="350dp"
```

```
31          android:layout_marginStart="10dp"
32          android:layout_marginEnd="10dp"
33          android:contentDescription=""
34          tools:srcCompat="@tools:sample/backgrounds/scenic" />
35
36      <TextView
37          android:id="@+id/room_name"
38          style="@style/item_name"
39          android:layout_width="match_parent"
40          android:layout_height="wrap_content"
41          android:layout_marginTop="20dp"
42          android:textAlignment="center"
43          tools:text="Program Name" />
44
45      <RelativeLayout
46          android:layout_width="match_parent"
47          android:layout_height="wrap_content"
48          android:layout_margin="8dp"
49          android:layout_marginStart="5dp"
50          android:layout_marginEnd="5dp"
51          android:layout_marginBottom="8dp">
52
53          <androidx.core.widget.NestedScrollView
54              android:layout_width="match_parent"
55              android:layout_height="match_parent">
56
57              <TextView
58                  android:id="@+id/room_description"
59                  style="@style/item_description"
60                  android:layout_width="match_parent"
61                  android:layout_height="wrap_content"
62                  android:padding="4dp"
63                  tools:text="@string/item_description" />
64
```

```
        </androidx.core.widget.NestedScrollView>
    </RelativeLayout>
</LinearLayout>
```

5.3.9 Special Events

This is the Kotlin code for implementing the special events.

```
package android.example.parkandrecreation.RecScreens.SpecialEvents

data class Event(
    val eventName: String,
    val eventInfo: String,
    val eventImageOne: Int,
    val eventImageTwo: Int,
    val EventImageThree: Int
)

package android.example.parkandrecreation.RecScreens.SpecialEvents

import androidx.fragment.app.Fragment
import androidx.viewpager2.adapter.FragmentStateAdapter

class EventAdapter (fragment:Fragment): FragmentStateAdapter(fragment){

    override fun getItemCount(): Int = 1

    override fun createFragment(position: Int): Fragment {
        return EventDisplayFragment()
    }
}

package android.example.parkandrecreation.RecScreens.SpecialEvents

```

```
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentEventDisplayBinding

/**
 * A simple [Fragment] subclass.
 */
class EventDisplayFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentEventDisplayBinding.inflate(layoutInflater)

        return binding.root
    }

}

package android.example.parkandrecreation.RecScreens.SpecialEvents

import android.example.parkandrecreation.R
import android.example.parkandrecreation.databinding.FragmentEventsBinding
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
```

```
import android.view.ViewGroup
import androidx.viewpager2.widget.ViewPager2

/**
 * Simple fragment that holds the view pager that will display multiple fragments
 */
class EventsFragment : Fragment() {

    lateinit var viewPager: ViewPager2
    lateinit var eventAdapter: EventAdapter

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val binding = FragmentEventsBinding.inflate(layoutInflater)

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        eventAdapter = EventAdapter(this)
        viewPager = view.findViewById(R.id.events_Pager)
        viewPager.adapter = eventAdapter
    }

}
```

This is the XML code for implementing the special events.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.SpecialEvents.EventsFragment">

    <!-- View pager for Event Fragment -->
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/events_Pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecScreens.SpecialEvents.EventDisplayFragment">

    <!-- Empty Layout-->


</FrameLayout>
```

5.3.10 Database

This is the MySQL code used for implementing the database.

```
create database recreation;
use recreation;
create table recreationCenter (recID varchar(5) NOT NULL, recName varchar(30), recAddress
    varchar(60), City varchar(30), State varchar(30), zip varchar(30), recPhone varchar(10),
    recImage1 longblob, recImage2 longblob, recImage3 longblob, recImage4 longblob, PRIMARY KEY
    (recID));
```

```
create table Rooms(roomID varchar(6) not null, recID varchar(5), size varchar(4), price double,
    description varchar(150), roomImage1 longblob, roomImage2 longblob, primary key(roomID),
    foreign key(recID) references recreationCenter(recID));

create table Events (eventID varchar(6) not null, date date, recID varchar(5), eventImage1
    longblob, eventImage2 longblob, eventImage3 longblob, description varchar(150), primary
    key(eventID, date), foreign key(recID) references recreationCenter(recID));

create table fitnessCenter(gymID varchar(6) not null, recID varchar(5), description varchar(150),
    price double, image1 longblob, image2 longblob, image3 longblob, primary key(gymID), foreign
    key (recID) references recreationCenter(recID));

create table Programs(programID varchar(6) not null, recID varchar(5), description varchar(150),
    instructorInfo varchar(150), price double, primary key(programID), foreign key(recID)
    references recreationCenter(recID));

create table Staff(staffID int unsigned not null, managerID int unsigned, staffName varchar(45),
    email varchar(45), description varchar(150), recID varchar(5), primary key staff(staffID),
    foreign key(managerID) references staff(staffID), foreign key(recID) references
    recreationCenter(recID));

create table hoursOfOperations(dayCode varchar(6) not null, startTime time, endTime time, day
    varchar(10), primary key(dayCode));

create table recreationCenterHours_of_Operations(recID varchar(5) not null, dayCode varchar(6) not
    null, primary key(recID, dayCode), foreign key(recID) references recreationCenter(recID),
    foreign key(dayCode) references hoursOfOperations(dayCode));

insert into recreationCenter values("MW001", "Martin Weiss Recreation Center", "1111 Martindell
    Ave", "Dallas", "Texas", "75211", "2146701919", load_file("C:\Projects\Pictures\Path1.jpeg"),
    load_file("C:\Projects\Pictures\Path2.jpeg"), load_file("C:\Projects\Pictures\Path3.jpeg"),
    null);
insert into recreationCenter values("KP002", "Keist Park Recreation Center", "3081 S. Hampton Rd",
```

```
    "Dallas", "Texas", "75224", "2146701918", load_file("C:\Projects\Pictures\Kiest1.jpeg"),
    load_file("C:\Projects\Pictures\Kiest2.jpg"), null, null);

insert into Rooms values ("KSR01","KP002","Sm",40.00,null,null,null);
insert into Rooms values ("KSR02","KP002","Sm",40.00,null,null,null);
insert into Rooms values ("KSR03","KP002","Lg",60.00,null,null,null);
insert into Rooms values ("MWR01","MW001","Lg",60.00,null,null,null);
insert into Rooms values ("MWR02","MW001","Lg",60.00,null,null,null);
insert into Rooms values ("MWR03","MW001","Sm",40.00,null,null,null);


insert into Events values("KPE01","20-04-12","KP002",null,null,null,"Easter Egg Hunt");
insert into Events values("KSPE02","20-05-21","KP002",null,null,null,"Movie Night");
insert into Events values("KPE02","20-05-30","KP002",null,null,null,"Summer Camp");
insert into Events values("MWE01","20-06-01","KP002",null,null,null,"Swimming Pool Opens");
insert into Events values("MWE02","20-05-25","KP002",null,null,null,"Baseball Starts");
insert into Events values("MWE03","20-04-19","KP002",null,null,null,"Yoga");


insert into fitnessCenter values("KSFC01","KP002",null,20.00,null,null,null);
insert into fitnessCenter values("MWFC01","MW001",null,20.00,null,null,null);


insert into Programs values("KPP01","KP002","Soccer",null,"30.00");
insert into Programs values("KPP02","KP002","Basketball",null,"35.00");
insert into Programs values("KPP03","KP002","Baseball",null,"35.00");
insert into Programs values("KPP04","KP002","Karate",null,"35.00");
insert into Programs values("KPP05","KP002","Tae Kwon Do",null,"40.00");
insert into Programs values("KPP06","KP002","After School",null,"50.00");
insert into Programs values("MWP01","MW001","Soccer",null,"30.00");
insert into Programs values("MWP02","MW001","Basketball",null,"35.00");
insert into Programs values("MWP03","MW001","Baseball",null,"35.00");
insert into Programs values("MWP04","MW001","Karate",null,"35.00");
insert into Programs values("MWP05","MW001","Swimming",null,"35.00");
```

```
insert into Programs values("MWP06","MW001","After School",null,"35.00");



insert into Staff values(1,null,"Victor Garcia","VicG@gmail.com",null,"KP002");
insert into Staff values(2,null,"Manuel Garcia","ManG@gmail.com",null,"KP002");
insert into Staff values(3,1,"Monica Gomez","MonG@gmail.com",null,"KP002");
insert into Staff values(111,1,"Gerardo Perez","GerP@gmail.com",null,"MW001");
insert into Staff values(112,2,"Jorge Mendoza","JorM@gmail.com",null,"MW001");
insert into Staff values(113,null,"Jacky Valdez","JacV@gmail.com",null,"MW001");



insert into hoursOfOperations values(1, 090000, 220000, "MTWR"); /*ID 1-4 represents Kiest Park
    Hours*/
insert into hoursOfOperations values(2, 140000, 190000, "F");
insert into hoursOfOperations values(3, 090000, 170000, "Sa");
insert into hoursOfOperations values(4, null, null, "Su"); ID /*4 also applies to Martin Wiess*/
insert into hoursOfOperations values(5, 090000, 200000, "MW"); /*ID 5-8 represents Martin Weiss*/
insert into hoursOfOperations values(6, 090000, 210000, "TR");
insert into hoursOfOperations values(7, 143000, 183000, "F");
insert into hoursOfOperations values(8, 090000, 140000, "Sa");



insert into recreationCenterHours_of_Operations values("KP002", 1);
insert into recreationCenterHours_of_Operations values("KP002", 2);
insert into recreationCenterHours_of_Operations values("KP002", 3);
insert into recreationCenterHours_of_Operations values("KP002", 4);
insert into recreationCenterHours_of_Operations values("MW001", 4);
insert into recreationCenterHours_of_Operations values("MW001", 5);
insert into recreationCenterHours_of_Operations values("MW001", 6);
```

```
insert into recreationCenterHours_of_Operations values("MW001", 7);
insert into recreationCenterHours_of_Operations values("MW001", 8);
```

5.3.11 Miscellaneous Code

This is all the additional XML code used to implement the system.

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:cardElevation="0dp"
    app:cardPreventCornerOverlap="true">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/rec_image"
            android:layout_width="match_parent"
            android:layout_height="@dimen/rec_grid_image_height"
            android:background="?attr/colorPrimaryDark"
            android:scaleType="centerCrop"
            tools:src="@drawable/martin_weiss"/>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
```

```
28            android:padding="16dp">
29
30            <TextView
31                android:id="@+id/rec_name"
32                android:layout_width="match_parent"
33                android:layout_height="wrap_content"
34                android:textAlignment="center"
35                android:gravity="center_horizontal"
36                tools:text="Text Placeholder"/>
37        </LinearLayout>
38    </LinearLayout>
39
40 </com.google.android.material.card.MaterialCardView>
41
42 <?xml version="1.0" encoding="utf-8"?>
43 <navigation xmlns:android="http://schemas.android.com/apk/res/android"
44     xmlns:app="http://schemas.android.com/apk/res-auto"
45     xmlns:tools="http://schemas.android.com/tools"
46     android:id="@+id/nav_graph"
47     app:startDestination="@id/loadFragment">
48
49     <fragment
50         android:id="@+id/loadFragment"
51         android:name="android.example.parkandrecreation.LoadScreen.LoadFragment"
52         android:label="fragment_load"
53         tools:layout="@layout/fragment_load" >
54         <action
55             android:id="@+id/action_loadFragment_to_homepageFragment"
56             app:destination="@id/homepageFragment"
57             app:enterAnim="@anim/slide_up"
58             app:popEnterAnim="@anim/fade_in" />
59     </fragment>
60     <fragment
61         android:id="@+id/homepageFragment"
```

```
62          android:name="android.example.parkandrecreation.RecScreens.HomePage.HomepageFragment"
63          android:label="fragment_homepage"
64          tools:layout="@layout/fragment_homepage" >
65          <action
66              android:id="@+id/action_homepageFragment_to_recreationCenterFragment"
67              app:destination="@id/recreationCenterFragment"
68              app:enterAnim="@anim/fade_in"
69              app:exitAnim="@anim/fade_out"
70              app:popEnterAnim="@anim/slide_in_left"
71              app:popExitAnim="@anim/slide_out_right" />
72      </fragment>
73      <fragment
74          android:id="@+id/recreationCenterFragment"
75          android:name="android.example.parkandrecreation.RecScreens.RecreationCenterFragment"
76          android:label="fragment_recreation_center"
77          tools:layout="@layout/fragment_recreation_center" >
78          <argument
79              android:name="recID"
80              app:argType="string" />
81          <action
82              android:id="@+id/action_recreationCenterFragment_to_programFragment"
83              app:destination="@id/programFragment"
84              app:enterAnim="@anim/fade_in"
85              app:exitAnim="@anim/fade_out"
86              app:popEnterAnim="@anim/slide_in_left"
87              app:popExitAnim="@anim/slide_out_right" />
88          <action
89              android:id="@+id/action_recreationCenterFragment_to_eventsFragment"
90              app:destination="@id/eventsFragment"
91              app:enterAnim="@anim/fade_in"
92              app:exitAnim="@anim/fade_out"
93              app:popEnterAnim="@anim/slide_in_left"
94              app:popExitAnim="@anim/slide_out_right" />
95          <action
```

```
96            android:id="@+id/action_recreationCenterFragment_to_fitnessFragment"
97            app:destination="@id/fitnessFragment"
98            app:enterAnim="@anim/fade_in"
99            app:exitAnim="@anim/fade_out"
100           app:popEnterAnim="@anim/slide_in_left"
101           app:popExitAnim="@anim/slide_out_right" />
102       <action
103           android:id="@+id/action_recreationCenterFragment_to_gymFragment"
104           app:destination="@id/gymFragment"
105           app:enterAnim="@anim/fade_in"
106           app:exitAnim="@anim/fade_out"
107           app:popEnterAnim="@anim/slide_in_left"
108           app:popExitAnim="@anim/slide_out_right" />
109       <action
110           android:id="@+id/action_recreationCenterFragment_to_parkFragment"
111           app:destination="@id/parkFragment"
112           app:enterAnim="@anim/fade_in"
113           app:exitAnim="@anim/fade_out"
114           app:popEnterAnim="@anim/slide_in_left"
115           app:popExitAnim="@anim/slide_out_right" />
116       <action
117           android:id="@+id/action_recreationCenterFragment_to_roomFragment"
118           app:destination="@id/roomFragment"
119           app:enterAnim="@anim/fade_in"
120           app:exitAnim="@anim/fade_out"
121           app:popEnterAnim="@anim/slide_in_left"
122           app:popExitAnim="@anim/slide_out_right" />
123   </fragment>
124   <fragment
125       android:id="@+id/programFragment"
126       android:name="android.example.parkandrecreation.RecScreens.ProgramScreens.ProgramFragment"
127       android:label="fragment_program"
128       tools:layout="@layout/fragment_program" >
129       <argument
```

```
130             android:name="recID"
131             app:argType="string" />
132     </fragment>
133     <fragment
134         android:id="@+id/eventsFragment"
135         android:name="android.example.parkandrecreation.RecScreens.SpecialEvents.EventsFragment"
136         android:label="fragment_events"
137         tools:layout="@layout/fragment_events" >
138         <argument
139             android:name="recID"
140             app:argType="string" />
141     </fragment>
142     <fragment
143         android:id="@+id/fitnessFragment"
144         android:name="android.example.parkandrecreation.RecScreens.FitnessRoom.FitnessFragment"
145         android:label="fragment_fitness"
146         tools:layout="@layout/fragment_fitness" >
147         <argument
148             android:name="recID"
149             app:argType="string" />
150     </fragment>
151     <fragment
152         android:id="@+id/gymFragment"
153         android:name="android.example.parkandrecreation.RecScreens.Gymnasiums.GymFragment"
154         android:label="fragment_gym"
155         tools:layout="@layout/fragment_gym" >
156         <argument
157             android:name="recID"
158             app:argType="string" />
159     </fragment>
160     <fragment
161         android:id="@+id/parkFragment"
162         android:name="android.example.parkandrecreation.RecScreens.ParkInformation.ParkFragment"
163         android:label="fragment_park"
```

```
164         tools:layout="@layout/fragment_park" >
165         <argument
166             android:name="recID"
167             app:argType="string" />
168     </fragment>
169     <fragment
170         android:id="@+id/roomFragment"
171         android:name="android.example.parkandrecreation.RecScreens.Rooms.RoomFragment"
172         android:label="fragment_rooms"
173         tools:layout="@layout/fragment_rooms" >
174         <argument
175             android:name="recID"
176             app:argType="string" />
177     </fragment>
178 </navigation>
179
180 <?xml version="1.0" encoding="utf-8"?>
181 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
182     package="android.example.parkandrecreation">
183
184     <uses-permission android:name="android.permission.INTERNET" />
185     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
186
187
188     <application
189         android:allowBackup="true"
190         android:icon="@mipmap/ic_launcher"
191         android:screenOrientation="portrait"
192         android:label="@string/app_name"
193         android:roundIcon="@mipmap/ic_launcher_round"
194         android:supportsRtl="true"
195         android:theme="@style/AppTheme">
196         <activity android:name=".MainActivity">
197             <intent-filter>
```

```
198                <action android:name="android.intent.action.MAIN" />
199
200                <category android:name="android.intent.category.LAUNCHER" />
201            </intent-filter>
202        </activity>
203    </application>
204
205 </manifest>
206
207 <?xml version="1.0" encoding="utf-8"?>
208 <set xmlns:android="http://schemas.android.com/apk/res/android">
209     <alpha
210         android:duration="@android:integer/config_mediumAnimTime"
211         android:fromAlpha="0.0"
212         android:toAlpha="1.0"
213         />
214 </set>
215
216 <?xml version="1.0" encoding="utf-8"?>
217 <set xmlns:android="http://schemas.android.com/apk/res/android">
218     <alpha
219         android:duration="@android:integer/config_mediumAnimTime"
220         android:fromAlpha="1.0"
221         android:toAlpha="0.0" />
222 </set>
223
224 <?xml version="1.0" encoding="utf-8"?>
225 <set xmlns:android="http://schemas.android.com/apk/res/android">
226     <translate
227         android:fromXDelta="-100%"
228         android:toXDelta="0%"
229         android:fromYDelta="0%"
230         android:toYDelta="0%"
231         android:duration="@android:integer/config_mediumAnimTime"/>
```

```
232 </set>
233
234 <?xml version="1.0" encoding="utf-8"?>
235 <set xmlns:android="http://schemas.android.com/apk/res/android">
236
237     <translate
238         android:duration="@android:integer/config_mediumAnimTime"
239         android:fromXDelta="0%"
240         android:fromYDelta="0%"
241         android:toXDelta="100%"
242         android:toYDelta="0%" />
243 </set>
244
245 <?xml version="1.0" encoding="utf-8"?>
246 <set xmlns:android="http://schemas.android.com/apk/res/android">
247     <translate
248         android:duration="@android:integer/config_mediumAnimTime"
249         android:fromXDelta="0%"
250         android:fromYDelta="-100%"
251         android:toXDelta="0%"
252         android:toYDelta="0%"
253         />
254 </set>
255
256 <vector xmlns:android="http://schemas.android.com/apk/res/android"
257         android:width="24dp"
258         android:height="24dp"
259         android:viewportWidth="24.0"
260         android:viewportHeight="24.0">
261     <path
262         android:fillColor="#FF000000"
263         android:pathData="M5,16c0,3.87 3.13,7 7,7s7,-3.13 7,-7v-4L5,12v4zM16.12,4.37l2.1,-2.1
            -0.82,-0.83 -2.3,2.31C14.16,3.28 13.12,3 12,3s-2.16,0.28
            -3.09,0.75L6.6,1.44l-0.82,0.83 2.1,2.1C6.14,5.64 5,7.68 5,10v1h14v-1c0,-2.32
```

```
            -1.14,-4.36 -2.88,-5.63zM9,9c-0.55,0 -1,-0.45 -1,-1s0.45,-1 1,-1 1,0.45 1,1 -0.45,1
            -1,1zM15,9c-0.55,0 -1,-0.45 -1,-1s0.45,-1 1,-1 1,0.45 1,1 -0.45,1 -1,1z"/>
</vector>

<vector android:alpha="0.8" android:height="24dp"
    android:viewportHeight="24.0" android:viewportWidth="24.0"
    android:width="24dp" xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="#FF000000" android:pathData="M12,3c-4.97,0 -9,4.03 -9,9s4.03,9
        9,9c0.83,0 1.5,-0.67 1.5,-1.5 0,-0.39 -0.15,-0.74 -0.39,-1.01 -0.23,-0.26 -0.38,-0.61
        -0.38,-0.99 0,-0.83 0.67,-1.5 1.5,-1.5L16,16c2.76,0 5,-2.24 5,-5 0,-4.42 -4.03,-8
        -9,-8zM6.5,12c-0.83,0 -1.5,-0.67 -1.5,-1.5S5.67,9 6.5,9 8,9.67 8,10.5 7.33,12
        6.5,12zM9.5,8C8.67,8 8,7.33 8,6.5S8.67,5 9.5,5s1.5,0.67 1.5,1.5S10.33,8
        9.5,8zM14.5,8c-0.83,0 -1.5,-0.67 -1.5,-1.5S13.67,5 14.5,5s1.5,0.67 1.5,1.5S15.33,8
        14.5,8zM17.5,12c-0.83,0 -1.5,-0.67 -1.5,-1.5S16.67,9 17.5,9s1.5,0.67 1.5,1.5 -0.67,1.5
        -1.5,1.5z"/>
</vector>

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <solid android:color="@color/colorSecondaryDark"/>
    <corners android:radius="15dp"/>

</shape>

<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="108"
    android:viewportHeight="108">
    <path
        android:fillColor="#008577"
```

```
289         android:pathData="M0,0h108v108h-108z" />
290     <path
291         android:fillColor="#00000000"
292         android:pathData="M9,0L9,108"
293         android:strokeWidth="0.8"
294         android:strokeColor="#33FFFFFF" />
295     <path
296         android:fillColor="#00000000"
297         android:pathData="M19,0L19,108"
298         android:strokeWidth="0.8"
299         android:strokeColor="#33FFFFFF" />
300     <path
301         android:fillColor="#00000000"
302         android:pathData="M29,0L29,108"
303         android:strokeWidth="0.8"
304         android:strokeColor="#33FFFFFF" />
305     <path
306         android:fillColor="#00000000"
307         android:pathData="M39,0L39,108"
308         android:strokeWidth="0.8"
309         android:strokeColor="#33FFFFFF" />
310     <path
311         android:fillColor="#00000000"
312         android:pathData="M49,0L49,108"
313         android:strokeWidth="0.8"
314         android:strokeColor="#33FFFFFF" />
315     <path
316         android:fillColor="#00000000"
317         android:pathData="M59,0L59,108"
318         android:strokeWidth="0.8"
319         android:strokeColor="#33FFFFFF" />
320     <path
321         android:fillColor="#00000000"
322         android:pathData="M69,0L69,108"
```

```
323         android:strokeWidth="0.8"
324         android:strokeColor="#33FFFFFF" />
325     <path
326         android:fillColor="#00000000"
327         android:pathData="M79,0L79,108"
328         android:strokeWidth="0.8"
329         android:strokeColor="#33FFFFFF" />
330     <path
331         android:fillColor="#00000000"
332         android:pathData="M89,0L89,108"
333         android:strokeWidth="0.8"
334         android:strokeColor="#33FFFFFF" />
335     <path
336         android:fillColor="#00000000"
337         android:pathData="M99,0L99,108"
338         android:strokeWidth="0.8"
339         android:strokeColor="#33FFFFFF" />
340     <path
341         android:fillColor="#00000000"
342         android:pathData="M0,9L108,9"
343         android:strokeWidth="0.8"
344         android:strokeColor="#33FFFFFF" />
345     <path
346         android:fillColor="#00000000"
347         android:pathData="M0,19L108,19"
348         android:strokeWidth="0.8"
349         android:strokeColor="#33FFFFFF" />
350     <path
351         android:fillColor="#00000000"
352         android:pathData="M0,29L108,29"
353         android:strokeWidth="0.8"
354         android:strokeColor="#33FFFFFF" />
355     <path
356         android:fillColor="#00000000"
```

```
357         android:pathData="M0,39L108,39"
358         android:strokeWidth="0.8"
359         android:strokeColor="#33FFFFFF" />
360     <path
361         android:fillColor="#00000000"
362         android:pathData="M0,49L108,49"
363         android:strokeWidth="0.8"
364         android:strokeColor="#33FFFFFF" />
365     <path
366         android:fillColor="#00000000"
367         android:pathData="M0,59L108,59"
368         android:strokeWidth="0.8"
369         android:strokeColor="#33FFFFFF" />
370     <path
371         android:fillColor="#00000000"
372         android:pathData="M0,69L108,69"
373         android:strokeWidth="0.8"
374         android:strokeColor="#33FFFFFF" />
375     <path
376         android:fillColor="#00000000"
377         android:pathData="M0,79L108,79"
378         android:strokeWidth="0.8"
379         android:strokeColor="#33FFFFFF" />
380     <path
381         android:fillColor="#00000000"
382         android:pathData="M0,89L108,89"
383         android:strokeWidth="0.8"
384         android:strokeColor="#33FFFFFF" />
385     <path
386         android:fillColor="#00000000"
387         android:pathData="M0,99L108,99"
388         android:strokeWidth="0.8"
389         android:strokeColor="#33FFFFFF" />
390     <path
```

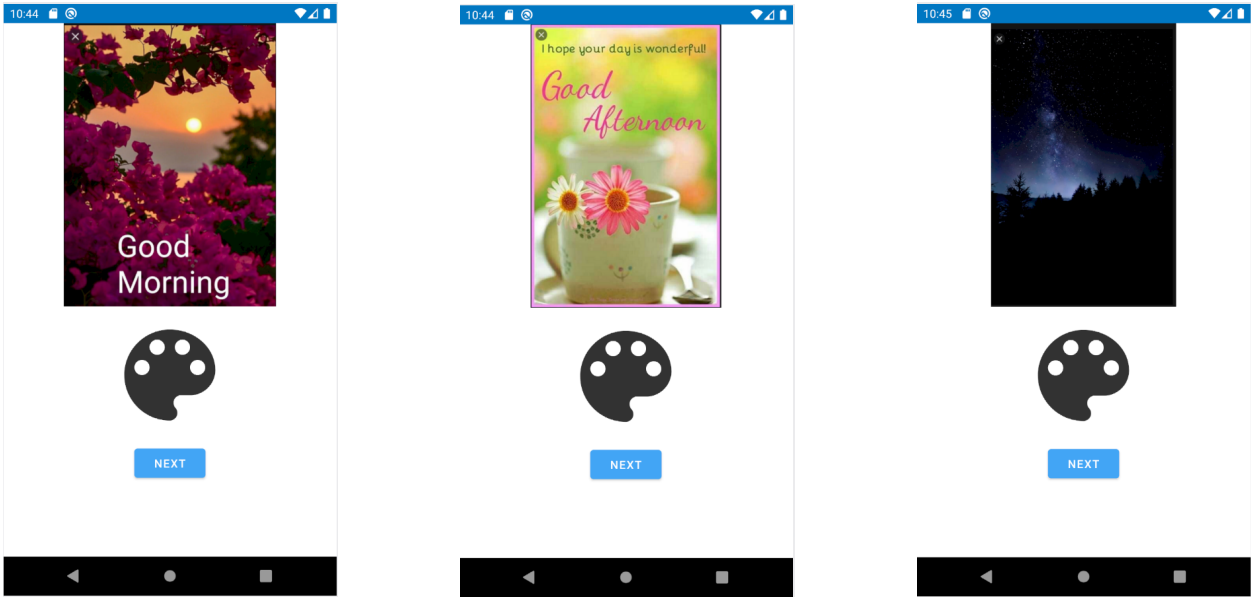
```
391         android:fillColor="#00000000"
392         android:pathData="M19,29L89,29"
393         android:strokeWidth="0.8"
394         android:strokeColor="#33FFFFFF" />
395     <path
396         android:fillColor="#00000000"
397         android:pathData="M19,39L89,39"
398         android:strokeWidth="0.8"
399         android:strokeColor="#33FFFFFF" />
400     <path
401         android:fillColor="#00000000"
402         android:pathData="M19,49L89,49"
403         android:strokeWidth="0.8"
404         android:strokeColor="#33FFFFFF" />
405     <path
406         android:fillColor="#00000000"
407         android:pathData="M19,59L89,59"
408         android:strokeWidth="0.8"
409         android:strokeColor="#33FFFFFF" />
410     <path
411         android:fillColor="#00000000"
412         android:pathData="M19,69L89,69"
413         android:strokeWidth="0.8"
414         android:strokeColor="#33FFFFFF" />
415     <path
416         android:fillColor="#00000000"
417         android:pathData="M19,79L89,79"
418         android:strokeWidth="0.8"
419         android:strokeColor="#33FFFFFF" />
420     <path
421         android:fillColor="#00000000"
422         android:pathData="M29,19L29,89"
423         android:strokeWidth="0.8"
424         android:strokeColor="#33FFFFFF" />
```

```
    <path
        android:fillColor="#00000000"
        android:pathData="M39,19L39,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,19L49,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,19L59,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,19L69,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,19L79,89"
        android:strokeWidth="0.8"
        android:strokeColor="#33FFFFFF" />
</vector>

<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

5.4 Testing

This section will include screen captures of the different pages contained in the Parks and Recreations Application along with a brief description after successful execution during testing. Assume the following scenario: A user downloads and uses the Parks and Recreation Center app and they want to see what recreation center is offering an after-school program. It is worth mentioning that the different images are transitioned through using a sweeping animation. That is, the screen swipes from right to left in a very fluid motion.



(a) Good Morning (b) Good Afternoon (c) Good Evening

Figure 5.5: Home Screens

Upon opening the application, the user is greeted with one of the following images. The time of day determines which image they will see. Morning hours will display the "Good morning" image 5.5a, afternoon hours will display the "Good afternoon image" 5.5b and evening hour will display the evening image 5.5c. Tapping "Next" to advance to the homepage worked every time as shown in figure 5.1.

The user then taps the Martin Weiss recreation center button, the screen transitions to that recreation center's homepage as shown on figure 5.2b. Once on the Weiss page, scrolling worked fine and all the buttons presented the information that they were supposed to. The user is looking for an after-school program, so they tap the program icon.

They soon see that Martin Weiss only offers two programs, karate and kickboxing, figures 5.4a and 5.4b. Therefore, the user needs to back out to the application homepage to check the next recreation center for an after-school program. A tap of the back button takes the user back one page at a time. This worked perfectly fine and made the application very easy to navigate.

Once on the application homepage, the user taps on the next available center, Kiest, figure 5.2a. The user performs the same steps as before and taps on the programs button. After the transition, the user is presented with the after-school program, figure 5.3a. The user can then simply close the application knowing that they have found a recreation center that has what they needed.

5.5 Summary

The application has been made to make navigating and finding specific recreation centers more effective and feasible for the user. People will experience a more streamlined method of signing in to the app, locate their desired center, and be taken to the homepage of said center with little to no hassle at all. The application is far from being considered "done", but the team are more than satisfied with the results.

Chapter 6

Conclusion and Future Work

Objectives:

- Present the conclusion to the system's proposal.
- Describe future work related to the system.

6.1 Conclusion

This report presents a mobile application dedicated to the City of Dallas Parks and Recreation Centers. This mobile application is built for android platforms only and provides information and features to make it easier for residents of the city of Dallas to stay up-to-date with all current and new information about the parks around them. The following section give an overview of all the chapters of this report.

In chapter 1, we defined the problem, presented an outline of our project and explained the motivation behind our decision to develop a mobile application for the City of Dallas Parks and Recs. In chapter 2, we summarized similar mobile applications that we found along the way that resembled what we wanted our mobile application to become. Also, in this chapter, we stated all of the different software development processes at our disposal and also stated which tools we would use to build our app. Chapter 3: In this chapter, listed the user and system requirements that our application should satisfy. In chapter 4 we introduce the design of our application. It presented the activity-flow diagram and illustrated how our application will perform with a user interacting with it. In chapter 5 we displayed the implementation of the GUI's that our application has and give a detailed overview of what each screen does and what does it lead to. In addition, this chapter presented the database design and system testing.

6.2 Future Work

This section summarizes the features that can be added to our application in the future. Due to a lack of time we were unable to implement them. These features are as follows:

- Connecting our application with a MySQL database instead of embedding the data within the application.
- In our app, we are only offering a default theme as of right now that includes a lot of the city of Dallas main colors which are green and blue with a white background but in the future we are planning on adding a dark theme in case any users prefer to have a dark theme instead. Implement a second theme to our application and give users the option to enable a dark mode would be another future work.
- Push Notifications could really help a lot out application with the users that will be using our app as they would be up-to-date quicker with the use of notifications. For example, in the case of an emergency closing or an upcoming event, the user would have a push notification sent to their phone's

notifications and also when they would open our app a notification would appear in the middle of the screen.

Bibliography

[1] F. F. Tsui, O. Karam, and B. Bernal, *Essentials of software engineering*. Jones & Bartlett Learning, 2016.

[2] "UML Information Flow Diagram." https://www.uml-diagrams.org/information-flow-diagrams.html. Accessed: 12/03/2019.

[3] "Android architecture patterns: Part 3: Model-view-viewmodel." https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b. Accessed: 01/21/2020.

[4] "Nyc 311." https://play.google.com/store/apps/details?id=gov.nyc.doitt.ThreeOneOne&hl=en_US. Accessed: 10/08/2019.

[5] "Florida state parks guide - pocket ranger." itmss://apps.apple.com/us/app/florida-state-parks-guide-pocket-ranger/id593195079?l=es&ign-mscache=1. Accessed: 09/09/2019.

[6] "Parks path mobile application." https://play.google.com/store/apps/details?id=com.app.nrpa&hl=en_IE. Accessed: 10/08/2019.

[7] "Charlottesville parks and recreation application." https://play.google.com/store/apps/details?id=com.kbtest.parkandrec&hl=da. Accessed: 10/08/2019.

[8] "Golden gate park application." https://play.google.com/store/apps/details?id=com.guidekickinc.android.goldengatepark&hl=en_US. Accessed: 10/08/2019.

[9] "Recreation gov." https://apps.apple.com/us/app/recreation-gov/id1440487780. Accessed: 10/08/2019.

[10] "Wmu student recreation center." https://play.google.com/store/apps/details?id=com.bottegasol.com.migym.WMUStudentRecreationCenter&hl=en_US. Accessed: 10/08/2019.

[11] "Philadelphia parks and recreation." https://www.phila.gov/parks-rec-finder/#/activities/athletic. Accessed: 10/08/2019.

[12] Y. Bassil, "A simulation model for the waterfall software development life cycle," *International Journal of Engineering And Technology (iJET)*, 2012.

[13] P. Kruchten, *The rational unified process: an introduction.* Addison-Wesley Professional, 2004.

[14] "Whats new in c# 8.0 - c# guide." https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8. Accessed: 2019-09-26.

[15] A. Hejlsberg, S. Wiltamuth, and P. Golde, *The C# programming language.* Adobe Press, 2006.

[16] "The kotlin language: 1.0 beta is here!." TheKotlinLanguage:1.0BetaisHere! Accessed: 09/24/2019.

[17] T. Gaddis, S. Mukherjee, and A. K. Bhattacherjee, *Starting out with Java: From control structures through objects.* Pearson, 2013.

[18] J. Duckett, *JavaScript and JQuery: interactive front-end web development.* Wiley Publishing, 2014.

[19] M. F. Sanner *et al.*, "Python: a programming language for software integration and development," *J Mol Graph Model*, vol. 17, no. 1, pp. 57–61, 1999.

[20] "Lua." https://www.lua.org/about.html. Accessed: 09/24/2019.

[21] C. B. Thomas Connolly, *Database Systems.* Pearson, 2019.

[22] "SQLilte History kernel description." https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT88784. Accessed: 2019-09-20.

[23] "Mysql." https://www.mysql.com/about/. Accessed: 09/20/2019.

[24] "RDMS kernel description." https://www.codecademy.com/articles/what-is-rdbms-sql. Accessed: 2019-09-19.

[25] "SQLite kernel description." https://www.javatpoint.com/sqlite-history. Accessed: 09/25/2019.

[26] "SQLilte History kernel description." (https://www.sqlite.org/about.html). Accessed: 09/25/2019.

[27] M. Fowler, C. Kobryn, and K. Scott, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[28] "UML Profile Diagrams." `https://www.uml-diagrams.org/profile-diagrams.html`. Accessed: 12/03/2019.

[29] "UML Timing Diagrams." `https://www.uml-diagrams.org/timing-diagrams.html`. Accessed: 12/03/2019.

[30] "UML Communication Diagram Overview." `https://www.uml-diagrams.org/communication-diagrams.html`. Accessed: 12/09/2019.

[31] "Android architecture patterns part 1: Model-view-controller." `https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6`. Accessed: 01/21/2020.