CHAPTER 1

INTRODUCTION

The Race Car Driver Management System is designed to efficiently manage and organize data related to race car drivers. The system utilizes file structures and indexing methods to ensure fast and accurate retrieval of information. This report aims to provide an overview of the system's architecture and functionality.

Figure 1.1 RACE CAR

1.1 PROJECT OVERVIEW

Race Car Driver Management System is a software application or platform designed to facilitate the efficient organization, storage, and retrieval of data related to race car drivers. It provides a comprehensive solution for managing and tracking various aspects of race car drivers' information, including personal details, racing statistics, performance records, contracts, schedules, and team affiliations. The system enables race car management teams, organizers, and other stakeholders to effectively handle driver-related tasks such as driver selection, contract management, performance analysis, scheduling, and communication. By centralizing and automating these processes, the Race Car Driver Management System streamlines operations, improves data accuracy, and enhances decision-making in the competitive world of motorsports.

1.2 INTRODUCTION TO PYTHON

Python is a versatile and powerful programming language widely used in various domains, including software development, data analysis, and automation. In the context of the Blood Bank Management System project, Python serves as the foundation for developing a robust and efficient software application to automate and streamline blood bank operations.

Python offers several advantages that make it an ideal choice for this project. Firstly, it has a clear and readable syntax, which makes it easier for developers to write and maintain code. This readability factor is particularly crucial in a project like the Blood Bank Management System, where multiple team members may collaborate and work on the codebase

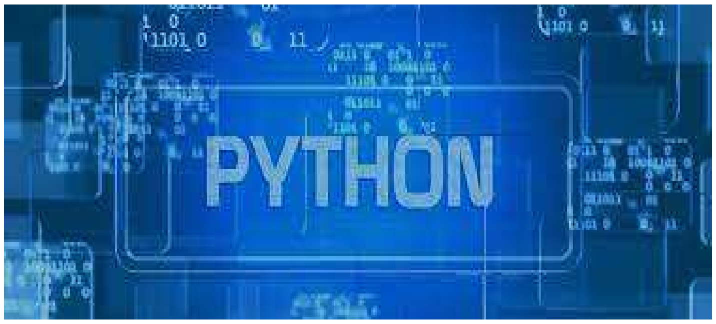


Figure 1.2 A view of Python

Secondly, Python has a vast ecosystem of libraries and frameworks that provide pre-built functionalities for different purposes. These libraries allow developers to leverage existing tools and components, significantly speeding up the development process. For example, libraries such as Flask or Django can be utilized to develop the user interface or web-based components of the Blood Bank Management System.

Python is a versatile programming language that offers a wide range of features, making it a popular choice among developers. Here are some key features of Python:

1.Easy to Learn and Readable Syntax: Python has a simple and readable syntax, making it easy for beginners to grasp the fundamentals of programming. The code is structured using indentation, which enhances code readability and reduces the chances of errors.

2.Cross-platform Compatibility: Python is a cross-platform language, meaning that code written in Python can run on various operating systems such as Windows, macOS, and Linux without requiring any modifications. This allows developers to write code once and deploy it on multiple platforms.

3.Extensive Library Support: Python boasts a vast standard library that provides a wide range of modules and functions for different purposes. These libraries enable developers to perform various tasks without having to build everything from scratch. Additionally, Python has a rich ecosystem of third-party libraries and frameworks that further extend its capabilities.

4.Dynamically Typed: Python is dynamically typed, which means that variable types are determined at runtime. Developers don't need to explicitly declare variable types, making code writing faster and more flexible. However, it's important to note that Python also supports type annotations to enhance code clarity and enable static type checking with tools like mypy.

5.Strong Community and Support: Python has a large and active community of developers, which means there is a wealth of resources, tutorials, and documentation available. The community is known for its helpfulness and collaborative nature, making it easy for developers to assist, share knowledge, contribute to open-source projects.

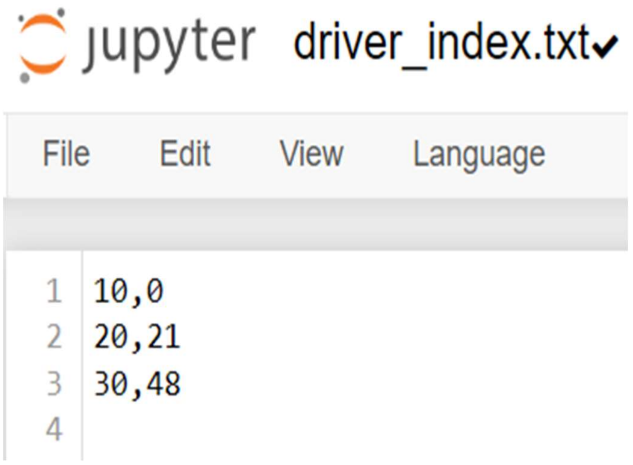
6.Data Analysis and Scientific Computing: Python has gained significant popularity in the field of data analysis and scientific computing. Libraries like NumPy, Pandas, and Matplotlib provide powerful tools for data manipulation, analysis, and visualization. Combined with frameworks like SciPy and scikit-learn, Python becomes a robust platform for scientific computing and machine learning.

7.Web Development: Python offers various frameworks like Flask and Django that simplify web development. These frameworks provide pre-built components, routing mechanisms.

1.3 INTRODUCTION TO TEXTFILE

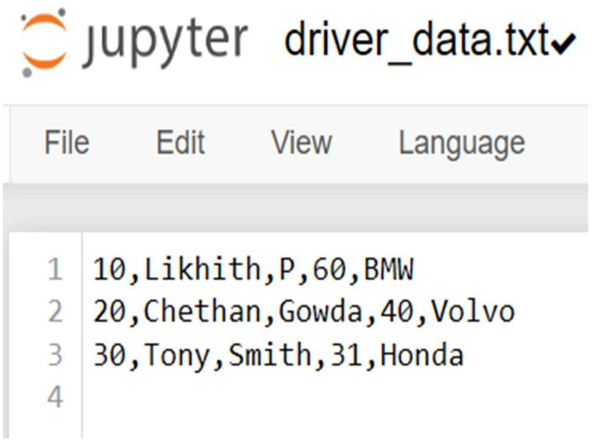
This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi. Name and version of text editor can vary on different operating systems. For example, Notepad will be used on Windows and vim or vi can be used on windows as well as Linux, or UNIX. The files you create with your editor are called source files and for Python they typically are named with the extension.py.

A text file is a kind of computer file that is structured as a sequence of lines of electronic text. A text file exists stored as data within a computer file system. In operating systems such as CP/M and MS- DOS, where the operating system does not keep track of the file size in bytes, the end of a text file is denoted by placing one or more special characters, known as an end-of-file marker, as padding after the last line in a text file. On modern operating systems such as Microsoft Windows and Unix-like systems, text files do not contain any special EOF character, because file systems on those operating systems.



```
10,0
20,21
30,48
```

Figure 1.3 A Sample index File



```
10,Likhith,P,60,BMW
20,Chethan,Gowda,40,Volvo
30,Tony,Smith,31,Honda
```

Figure 1.4 A Sample Data File

1.4 INTRODUCTION TO SIMPLE INDEXING

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely. Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing not only helps in querying, but can also be used in any algorithm being built to reduce the time taken by that algorithm. Simple indexes use simple arrays.

An index lets us impose order on a file impose order on a file without rearranging the file. Indexes provide multiple access paths multiple access paths to a file— multiple indexes multiple indexes (like library catalog providing search for author, book and title) An index can provide keyed access to variable-length record files.

Index is sorted (main memory). Records appear in file in the order they entered. An index is defined on one or more columns, called key columns. The key columns (also referred to as the index key) can be likened to the terms listed in a book index. They are the values that the index will be used to search for. As with the index found at the back of a text book, the index is sorted by the key columns.

Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.

Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values. Many operations can be performed efficiently like search, delete, modify after building the index.

CHAPTER 2

LITERATURE SURVEY

A race car driver management system plays a crucial role in the motorsports industry, providing comprehensive support to race car drivers and their teams. This literature survey aims to explore the various aspects of race car driver management systems, including their functions, benefits, and technological advancements.

Race car driver management systems encompass a range of tools and resources designed to enhance the performance and success of race car drivers. These systems typically include features such as data analysis and performance tracking, race scheduling and logistics management, sponsor and partnership management, and communication platforms. By integrating these functionalities, driver management systems streamline operations and enable drivers to focus on their racing skills.

One of the key benefits of a driver management system is the ability to analyze and interpret data collected during races and training sessions. Advanced telemetry systems capture various parameters such as speed, lap times, braking points, and acceleration, providing drivers and their teams with valuable insights for performance improvement. By leveraging this data, drivers can identify areas of strength and weakness, refine their driving techniques, and optimize their overall performance on the track.

Furthermore, race car driver management systems facilitate efficient race scheduling and logistics management. They allow teams to coordinate race participation, manage travel arrangements, and handle accommodation and catering requirements. This streamlines the process and ensures that drivers and their teams can focus on their racing commitments without being overwhelmed by logistical details.

Another crucial aspect of driver management systems is the management of sponsorships and partnerships. These systems enable drivers to track and manage their sponsorship contracts, communicate with sponsors, and fulfill their contractual obligations. By providing a centralized platform for sponsor management, driver management systems help drivers secure and maintain sponsorships, which are vital for funding their racing careers. Advanced telemetry systems capture various parameters such as speed, lap times, braking points, and acceleration, providing drivers and their teams with valuable insights for performance improvement. By leveraging this data, drivers can identify areas of strength and weakness, refine their driving techniques, and optimize their overall performance on the track.

Communication platforms within driver management systems allow for effective collaboration between drivers, their teams, and other stakeholders. These platforms facilitate seamless communication, enabling quick and efficient sharing of information, race strategies, and updates. By enhancing communication and collaboration, driver management systems contribute to improved team performance and coordination. This literature survey aims to explore the various aspects of race car driver management systems, including their functions, benefits, and technological advancements.

In recent years, technological advancements have further enhanced race car driver management systems. The integration of artificial intelligence and machine learning algorithms enables more sophisticated data analysis and performance predictions. Predictive analytics can assist drivers in making informed decisions on race strategies and car setups, giving them a competitive edge on the track. These systems typically include features such as data analysis and performance tracking, race scheduling and logistics management, sponsor and partnership management, and communication platforms. By integrating these functionalities, driver management systems streamline operations and enable drivers to focus on their racing skills.

In conclusion, race car driver management systems have become essential tools for the motorsports industry. These systems provide drivers and their teams with comprehensive support, ranging from data analysis and performance tracking to race scheduling and sponsor management. With technological advancements, driver management systems continue to evolve, empowering drivers to optimize their performance and achieve success in the highly competitive world of motorsports.

CHAPTER 3

SYSTEM ANALYSIS AND REQUIREMENTS

Analysis can be defined as breaking up of any whole so, as to find out their nature, function etc., it defines design as so make preliminary sketches. System analysis and design can be characterized as a set of techniques and processes, a community of interests, a culture and intellectual orientation.

System analysis is an important phase of any system development process. The system is studied to minutest detail and analyzed. A system analysis places an important role of an integrator and dwells deep into the working the present system. The system is studied and viewed as a whole and the input to the system are identified. The various tasks in the system analysis include the following:

- Understanding application
- Planning
- Scheduling
- Developing candidate solution
- Performing trade studies
- Performing alternative solution
- Performing cost benefit analysis
- Selling of system

System Analysis is a separation of a substance into parts for study and their implementation and detailed examination. Before designing any system, it is important that the nature of the businessand the way it currently operates are clearly understood. The detailed examination provides the specific data required during designing in order to ensure that all the client's requirements are fulfilled. The investigation or the study conducted during the analysis phase is largely based on the feasibility study. Rather it would not be wrong to say that the analysis and feasibility phases overlap. High-level analysis begins during the feasibility study. Though analysis is represented asone

phase of the system development life cycle (SDLC), this is not true.
Analysis begins with system initialization and continues until its maintenance. Even after successful implementation ofthe system, analysis may play its role for periodic maintenance and up gradation of the system.

One of the main causes of project failures is the inadequate understanding, and one of the main causes of inadequate understanding of the requirements is the poor planning of system analysis.
Analysis was done by keeping in mind the two modules of the project. The Analysis part of theproject was the user module. Users of this application may or may not have much computer knowledge, so we mainly focused on our design, which had to be as user friendly as possible. The next important thing was to provide user level security. It was necessary to provide privacyto community members. Another thing was the appearance of the application; it had to be madepleasant and decent enough to attract the user. Last but not the least, was to provide the authorities to the administrators. Proper validations where to be implemented of the registration-form.

3.1 EXISTING SYSTEM

The Race Car Driver Management System consists of the following key components:

a. Data Storage Files: These files store the driver records, including their personal information, racing statistics, and other relevant data.

b. Index Files: These files contain index structures that facilitate efficient searching and retrieval of driver records based on various search criteria.

c. User Interface: This component enables users to interact with the system, perform searches, update records, and generate reports.

d. File Structure Management: This module handles file organization, record insertion, deletion, and modification, ensuring data integrity and performance.

3.2 PROPOSED SYSTEM

The proposed Race Car Driver Management System is to provide a comprehensive and efficient solution for managing race car drivers and their related information.

3.3 OBJECTIVES OF PROPOSED SYSTEM

the objective of the proposed Race Car Driver Management System is to enhance the efficiency, accuracy, and effectiveness of managing race car drivers' information, thereby supporting race car management teams in their decision-making processes and improving overall performance in the motorsports industry.

The system aims to achieve the following objectives:

1. Centralized Data Management: The system will centralize all relevant data pertaining to race car drivers, including personal information, racing statistics, performance records, contracts, schedules, and team affiliations. This centralized database will ensure easy access, accuracy, and consistency of driver information.

2. Efficient Driver Selection: The system will facilitate the process of driver selection by providing a platform for evaluating and comparing driver profiles, racing achievements, and other relevant factors. This objective is to enable race car management teams to make informed decisions when selecting drivers for specific races or teams.

3. Streamlined Contract Management: The system will simplify and streamline the management of driver contracts. It will enable the creation, storage, and tracking of driver contracts, including terms, durations, payment details, and obligations. This objective is to ensure that contract-related tasks such as renewals, amendments, and terminations are efficiently handled.

4. Performance Analysis and Reporting: The system will provide tools and features for analyzing and reporting driver performance. It will allow race car management teams to assess driver statistics, track race results, analyze trends, and generate reports for evaluating driver performance and making data-driven decisions.

5. Scheduling and Communication: The system will support scheduling

functionalities, allowing race car management teams to plan and organize driver schedules for races, practice sessions, and other related events. It will also facilitate communication between drivers, team members, and race organizers, ensuring effective coordination and collaboration.

3.4 BENEFITS OF PROPOSED SYSTEM

Utilizing file structures and indexing methods in the Race Car Driver Management System offers several advantages, including:

a. Improved Performance: Indexing techniques enable faster search and retrieval of driver records, enhancing overall system responsiveness.

b. Efficient Data Organization: File structures ensure data integrity and enable easy insertion, deletion, and modification of records.

c. Enhanced User Experience: The system's quick response times and advanced search capabilities empower users to find and manage driver information efficiently.

d. Scalability: File structures and indexing methods provide scalability, allowing the system to handle a growing number of driver records without significant performance degradation.

3.5 DRAWBACKS OF EXISTING SYSTEM

- Time consuming.
- More man power.
- It leads to error prone results
- It consumes lot of manpower to better results
- It lacks of data security
- Retrieval of data may take some time.

3.6 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software applicationis the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatibility and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

- **HARDWARE REQUIREMENTS:**

Processor	Intel Core i5 or AMD FX 8 core series with clock speed of 2.4 GHz or above
RAM	4GB or above
Hard disk	1.5 GB or above
Input device	Keyboard or mouse or compatible pointing devices
Display	XGA (1024*768 pixels) or higher resolution monitor with 32-bit color settings
Miscellaneous	USB Interface, Power adapter, etc.

Table 3.2 Hardware Requirements

3.7 SOFTWARE REQUIREMENTS

Software Requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. Theserequirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed.

- **SOFTWARE REQUIREMENTS**:

Operating System	Windows
Programming Language – Backend	Notepad
Programming language - Frontend	Python
Database	Notepad

Table 3.3 Software Requirements

CHAPTER 4

DEVELOPMENT AND METHODOLOGY

The implementation of the Race Car Driver Management System involves a thorough process of requirement gathering, system design, development, testing, deployment, and ongoing maintenance. It requires close collaboration with stakeholders and a focus on creating a robust, user-friendly, and efficient software solution that meets the specific needs of race car management teams and the motorsports industry.
Implementation is the realization of application, or execution of plan, idea, model, design, application, standards, algorithm, or policy. In other words, an implementation is a realizationof technical specification or algorithm as a program, software component, or other computer System through programming. Many implementations may exist for the given specification orstandards Implementation is one of the important phases of the Software Development Life Cycle (SDLC). It encompasses all processes involved in getting new software or hardware operating properly in its environment, including installation configuration, running, testing andmaking necessary changes. Specifically, it involves coding the system using a particular programming language and transferring the design into actual working system. This phase of system is conducted with the idea that whatever is designs should be implemented, keeping inmind that it fulfills user requirements, objectives and scope of the system. The implementationphase produces the solution to the user problem.

4.1 MODULES DESCRIPTION

Front End and Back End:

Frontend and Backend are two most popular terms used in web development. These terms are very crucial for web development but are quite different from each other. Each side needs to communicate and operate effectively with the other as a single unit to improve the website’s functionality.

1. Insert Record
2. Display Record
3. Search Record
4. Delete Record
5. Update Record

1. Insert Record: Here records of the donor are inserted and can retrieve the data easily.

2. Display Record: Here we can display all the records.

3. Search Record: Here we can search the records.

4. Delete Record: Here we can delete the record if there is any need.

5. Update Record: Here the records are updated if there are any changes.

- **Python**:
  Python is a versatile and powerful programming language widely used in various domains, including software development, data analysis, and automation. In the context of the Blood Bank Management System project, Python serves as the foundation for developing a robust and efficient software application to automate and streamline blood bank operations.

Back end:

Text file: Because of their simplicity, text files are commonly used for storage of information. They avoid some of the problems encountered with other file formats, such as endianness, padding bytes, or differences in the number of bytes in a machine word. Further, when data corruption occurs in a text file, it is often easier to recover and continue processing the remaining contents.

4.2 BUILDING INDEX FILE

Primary Index File (driver_index.txt) The primary index file comprises of the primary key and it's starting position in the record file. Usually, the first column in the record file will be unique and it will be considered as primary key column. In the Human Resources records data-set used in this project "Driver ID" is the first column which is unique So it is being used as primary key for the purpose of running various operations based on this primary key like searching the record based on primary index, deleting the record based on primary index and also modifying the records based on this primary index file.

4.3 INSERTING THE RECORDS

The user is given with an option to insert the records into the record file. Upon execution of the insertion operation, the new record will be inserted into the record file, primary-index file and, also the secondary index file. This record will be inserted at the end of the file, that is it will be appended to these files. After insertion the index files are rebuilt once again. This insertion operation is carried out by add_driver() method which reads all the required fields, and then add_driver() method packs all these data and then writes it to corresponding files.

Implemented code for add_driver():

```
def add_driver():
    driver_id = input("Enter driver ID: ")
    first_name = input("Enter driver's first name: ")
    last_name = input("Enter driver's last name: ")
    races_won = input("Enter number of races won: ")
    sponsor = input("Enter driver's sponsor: ")
    with open(index_file, "r") as index_f:
        for line in index_f:
            index_driver_id, _ = line.strip().split(",")
            if index_driver_id == driver_id:
                print("Driver ID already exists!")
                return
    with open(index_file, "a") as index_f, open(data_file, "a") as data_f:
        index_f.write(f"{driver_id},{data_f.tell()}\n")
  data_f.write(f"{driver_id},{first_name},{last_name},{races_won},{sponsor}\n")
    print("Driver added successfully!")
```

4.4 SEARCHING AND MODIFYING RECORDS

The user can search a record from the record file using search_driver() method based on primary index or secondary-index. Searching using primary-index file prompts the user to enter the primary key, this key will be used to locate the record in the record file. This search operation makes use of the binary search algorithm in order to efficiently search the primary key and seek its position. Searching using secondary-index prompts the user to enter the secondary key, this key will be used to locate all the records which contains this secondary key. The user can also modify the records present in the record file. This update_driver() method permits the user to modify any field of the record which he has searched.

Implemented code for search_driver():

```
def search_driver():
    driver_id = input("Enter driver ID to search: ")
    with open(index_file, "r") as index_f:
        index_data = index_f.readlines()
    with open(data_file, "r") as data_f:
        data_data = data_f.readlines()
        found = False
        for i in range(len(index_data)):
            index_driver_id, data_offset = index_data[i].strip().split(",")
            if index_driver_id == driver_id:
                found = True
                driver_data = data_data[i].strip().split(",")
                print("----- Driver Details -----")
                print(f"Driver ID: {driver_data[0]}")
                print(f"Name: {driver_data[1]} {driver_data[2]}")
                print(f"Races Won: {driver_data[3]}")
                print(f"Sponsor: {driver_data[4]}")
                print("-------------------------")
                break
        if not found:
            print("Driver ID not found!")
```

Implemented code for update_driver():

```
def update_driver():
    driver_id = input("Enter driver ID to update: ")
    with open(index_file, "r") as index_f:
        index_data = index_f.readlines()
    with open(data_file, "r+") as data_f:
        data_data = data_f.readlines()
        found = False
        for i in range(len(index_data)):
            index_driver_id, data_offset = index_data[i].strip().split(",")
            if index_driver_id == driver_id:
                found = True
                print("Driver Found!")
                driver_data = data_data[i].strip().split(",")
                print("Driver Details to be updated:")
                print(f"Driver ID: {driver_data[0]}")
                print(f"Name: {driver_data[1]} {driver_data[2]}")
                print(f"Races Won: {driver_data[3]}")
                print(f"Sponsor: {driver_data[4]}")
                print("1. Update first name")
                print("2. Update last name")
                print("3. Update number of races won")
                print("4. Update sponsor")
                choice = input("Enter your choice: ")
                if choice == "1":
                    new_first_name = input(f"Enter new first name for driver {driver_id}:
")
                    driver_data[1] = new_first_name
                elif choice == "2":
                    new_last_name = input(f"Enter new last name for driver {driver_id}:
")
                    driver_data[2] = new_last_name
                elif choice == "3":
                    new_races_won = input(f"Enter new number of races won for driver
{driver_id}: ")
                    driver_data[3] = new_races_won
                elif choice == "4":
                    new_sponsor = input(f"Enter new sponsor for driver {driver_id}: ")
                    driver_data[4] = new_sponsor
                else:
                    print("Invalid choice!")
                    return
                data_data[i] = ",".join(driver_data) + "\n"
                data_f.seek(0)
                data_f.writelines(data_data)
                data_f.truncate()
                print("Driver updated successfully!")
                break
        if not found:
            print("Driver ID not found!")
```

4.5 DELETING RECORDS

Similar to search operation, in delete operation the user has the privilege to delete the records. Deletion of records can again be done based on primary-index or secondary-index. The record to be deleted will be prefixed with a asterisk (*) symbol, which indicates that the record is deleted and this record will not be considered while unpacking the records.

Implemented code for delete_driver():

```
def delete_driver():
    driver_id = input("Enter driver ID to delete: ")
    with open(index_file, "r") as index_f:
        index_data = index_f.readlines()
    with open(data_file, "r") as data_f:
        data_data = data_f.readlines()
    found = False
    for i in range(len(index_data)):
        index_driver_id, data_offset = index_data[i].strip().split(",")
        if index_driver_id == driver_id:
            found = True
            driver_data = data_data[i].strip().split(",")
            print("Driver Found!")
            print("Driver Details to be deleted:")
            print(f"Driver ID: {driver_data[0]}")
            print(f"Name: {driver_data[1]} {driver_data[2]}")
            print(f"Races Won: {driver_data[3]}")
            print(f"Sponsor: {driver_data[4]}")
            confirmation = input("Are you sure you want to delete this driver?
(yes/no): ")
            if confirmation.lower() == "yes":
                del index_data[i]
                del data_data[i]
                # Update index file
                with open(index_file, "w") as index_f:
                    index_f.writelines(index_data)
                # Update data file
                with open(data_file, "w") as data_f:
                    data_f.writelines(data_data)
                print("Driver deleted successfully!")
            else:
                print("Deletion canceled.")
            break
    if not found:
        print("Driver ID not found!")
```

CHAPTER 5

TESTING

5.1 INTRODUCTION

The testing phase is an important part of software development. It is the processes of finding errors and missing operations and also complete verifications to determine whether theobjectives are requirements are satisfied. Software testing is carried out in three steps.

The first step includes unit testing where in each module is tested to provide its correctness, todetermine any missing operations and to verify whether the objectives have been met. Errors arenoted down and corrected immediately. Unit testing is the important and major part of the project. So, errors are rectified easily in particular modules and program quality is increased. In this project, entire system is divided into several modules.

Second step include integration testing. If we need not be the case that software whose moduleswhen run individually and showing perfect result with also show perfect result as whole. The individual modules are clipped under this major module and tested again and verified the results. A module can have inadvertent, adverse effect on any other on the global data structurecausing serious problems. Levels in testing:

- Unit testing
- Integration testing
- Validation testing

5.2 LEVELS OF TESTING

5.2.1 UNIT TESTING

Unit testing is a method by which individual units of source code, sets of one or more computerprogram modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In object-oriented programming a unit is often an entire interface, such as a class, but could be an individual method. For unit testing first we adopted thecode testing strategy, which examined the logic of program. During the development process

itself all the syntax errors etc. got rooted out. For this developed test case that result in executingevery instruction in the program or module i.e., every path through program was tested.

User Input

User will be inputting all the data from using a web browser/Terminal.

Error Handling

In this system, we have tried to handle all the errors that occurred while running the application.the common errors we saw were reading a tuple with an attribute set to null and database connection getting lost. For Testing we used Top-Down design a decomposition process which focuses as the flow of control, at latter strategies concern itself with code production. The first step is to study the overall aspects of the tasks at hand and break it into a number of independentmodules. The second step is to break one of these modules further into independent sub modules.

5.2.2 INTEGRATION TESTING

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined may not produce the desired functions. Integrated testing is the systematic testing to uncover the errors with an interface. This testing is done with simple data and developed system has run successfully with this simple data. The need for integrated systemis to find the overall system performance.

Steps to perform integration testing:

Step 1: Create a Test Plan

Step 2: Create Test Cases and Test Data

Step 3: Once the components have been integrated execute the test

cases

Step 4: Fix the bugs if any and re test the code

Step 5: Repeat the test cycle until the components have been successfully integrated.

Name of the Test	Integration testing
Test plan	To check whether the system works properly when all the modules are integrated.
Test Data	Sample text data

Table 5.1 Test cases for Integration Testing

5.2.3 SYSTEM TESTING

Ultimately, software is included with other system components and the set of system validation and integration tests are performed. System testing is a series of different tests whose main aim isto fully exercise the computer-based system. Although each test has a different role all work should verify that all system elements are properly integrated and formed allocated functions.

Name of the Test	**System Testing**
Item being tested	Over all functioning of File structure with all functions properly linked.
Sample Input	Sample data files
Expected Output	All the modules like adding donor details, searching donor details, etc; working as expected
Actual Output	Application reacts to user inputs in expected manner.
Remarks	Successful

Table 5.2 Test cases for Input-Output

5.2.4 VALIDATION TESTING

At the culmination of black box testing, software is completely assembled is as a package. Interfacing errors have been uncovered and the correct and final series of tests, i.e., validationtests begin. Validation test is defined with a simple definition that validation succeeds when thesoftware function in a manner that can be reasonably accepted by the customer.

5.2.5 OUTPUT TESTING

After performing validation testing, the next step is output testing of the proposed system. Sincethe system cannot be useful if it does not produce the required output. Asking the user about the format in which the system is required tests the output displayed or generated by the systemis required tests the output displayed or generated by the system under consideration. The output format on the screen is found to be corrected as the format was designated in the systemhas according to the user needs. As for the hard copy the output comes according to the specification requested by the user. The output testing does not result in any correction in the system.

5.2.6 TEST DATA AND OUTPUT

Taking various kind soft data plays a vital role in system testing. After preparing the test data system under study is tested using the test data. While testing, errors are again uncovered and corrected by using the above steps and corrections are also noted for future use.

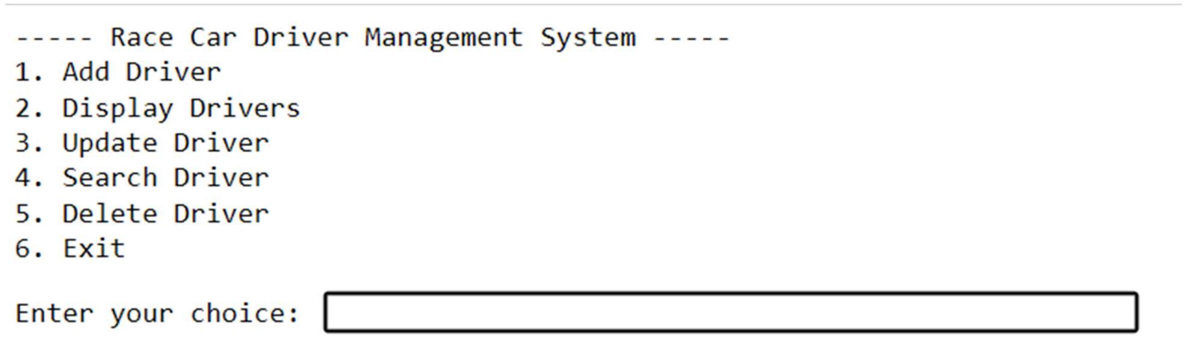
5.2.7 USER ACCEPTANCE TESTING

User acceptance testing of the system is the key factor for the success of the system. A systemunder consideration is tested for user acceptance by constantly keeping in touch with the prospective system at the time of development and making change whenever required. This isdone with regard to the input screen design and output screen design.

CHAPTER 6

RESULTS

The fig 6.1 shows the main menu of the project. The main menu contains the various options displayed to the users. Here user can choose any option to perform various operations.

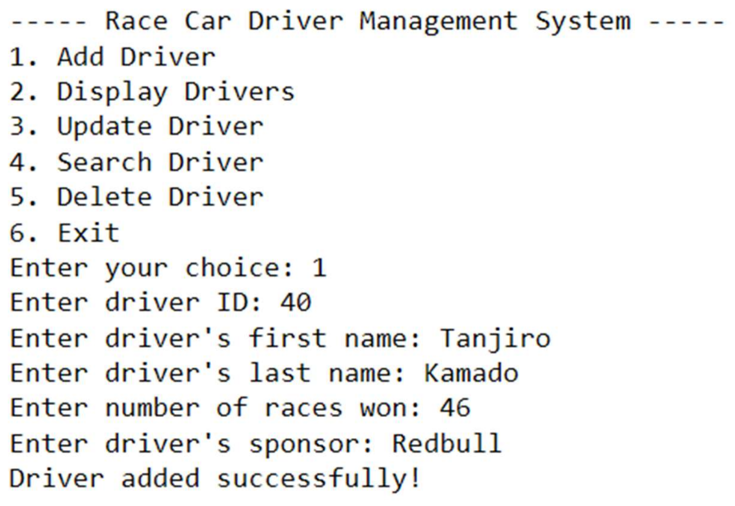


```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit

Enter your choice:
```

Figure 6.1 Main Menu

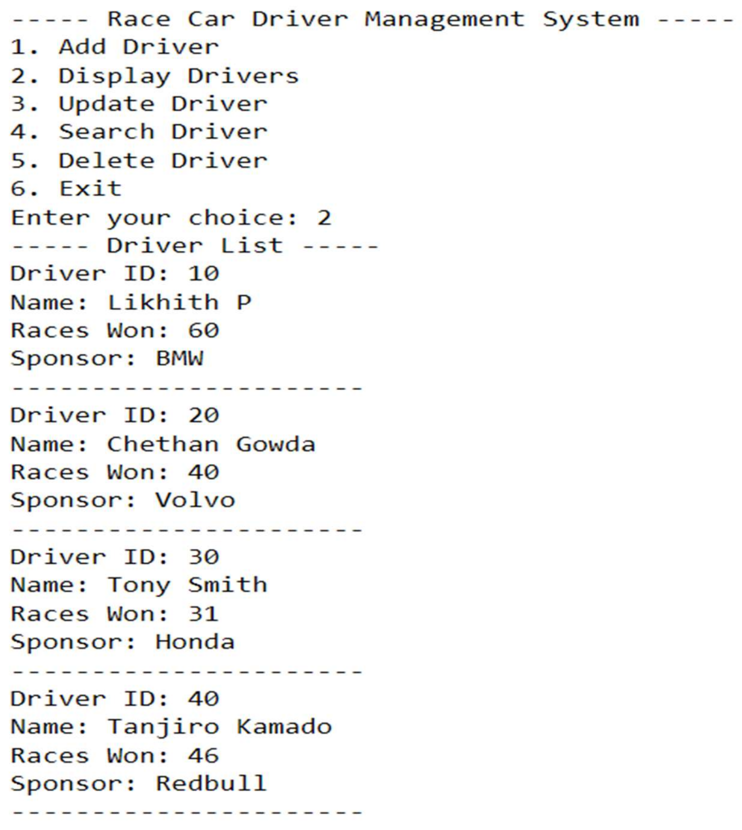
In fig 6.2 the insertion page is shown, the insertion page allows users to add new record. The user should enter details such as id, first name, last name, races won and sponsor. After entering all these details, the record will be inserted.



```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 1
Enter driver ID: 40
Enter driver's first name: Tanjiro
Enter driver's last name: Kamado
Enter number of races won: 46
Enter driver's sponsor: Redbull
Driver added successfully!
```

Figure 6.2 Insertion Page

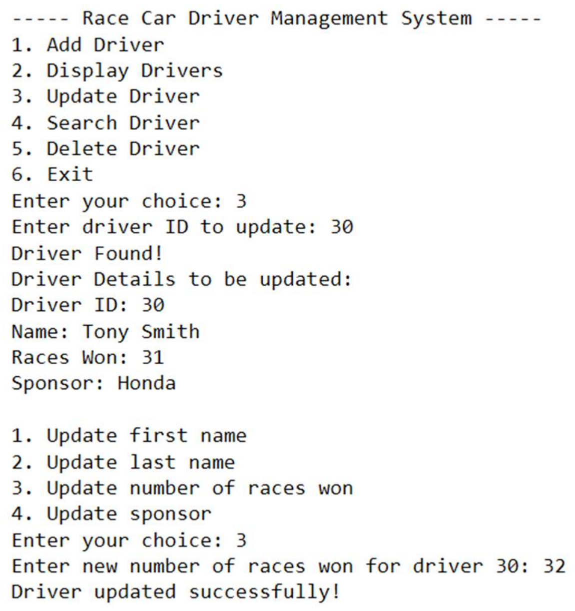
The display page in fig 6.3, it displays all the details of record inserted in the project.



```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 2
----- Driver List -----
Driver ID: 10
Name: Likhith P
Races Won: 60
Sponsor: BMW
----------------------
Driver ID: 20
Name: Chethan Gowda
Races Won: 40
Sponsor: Volvo
----------------------
Driver ID: 30
Name: Tony Smith
Races Won: 31
Sponsor: Honda
----------------------
Driver ID: 40
Name: Tanjiro Kamado
Races Won: 46
Sponsor: Redbull
----------------------
```

Figure 6.3 Display Page

The modification page in fig 6.5, allows user to update the record which is already inserted in the project.. The updated record will be stored in the text file.

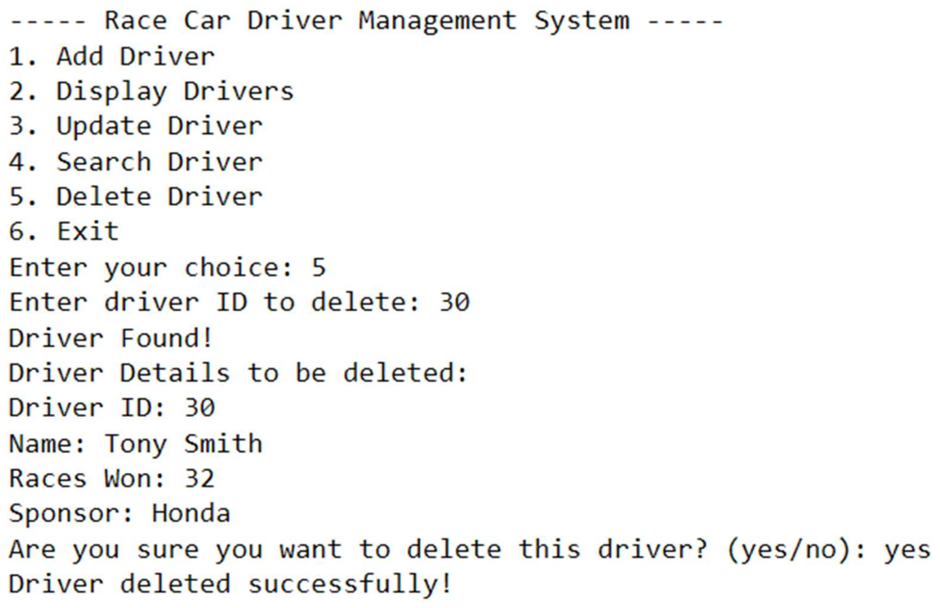


```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 3
Enter driver ID to update: 30
Driver Found!
Driver Details to be updated:
Driver ID: 30
Name: Tony Smith
Races Won: 31
Sponsor: Honda

1. Update first name
2. Update last name
3. Update number of races won
4. Update sponsor
Enter your choice: 3
Enter new number of races won for driver 30: 32
Driver updated successfully!
```

Figure 6.4 Modify Page

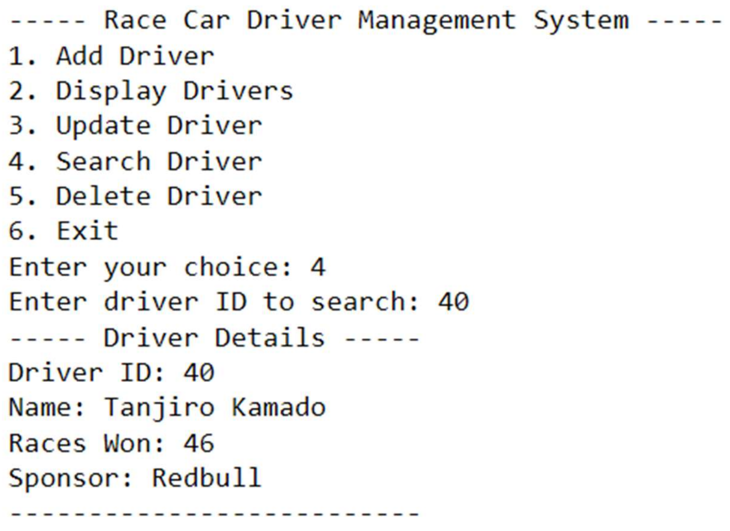
The fig 6.5 shows the deletion page, here users can delete any record which is already inserted in the project. The user needs to enter the Donor Id of the record to be deleted, if the Donor Id exists the particular record will be deleted.



```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 5
Enter driver ID to delete: 30
Driver Found!
Driver Details to be deleted:
Driver ID: 30
Name: Tony Smith
Races Won: 32
Sponsor: Honda
Are you sure you want to delete this driver? (yes/no): yes
Driver deleted successfully!
```

Figure 6.5 Deletion Page

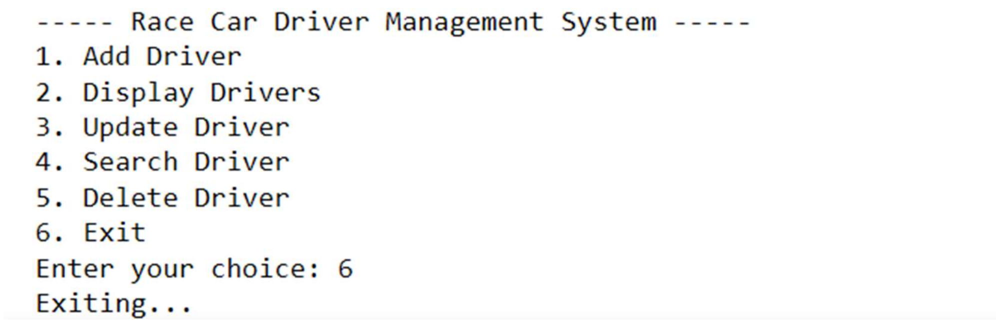
The fig 6.6 shows the search page, here user can search any record which is inserted. The user needs to enter the Donor Id of the record to be searched, if the record is found it displays the details of the record. If the record is not found it displays a message saying record not found.



```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 4
Enter driver ID to search: 40
----- Driver Details -----
Driver ID: 40
Name: Tanjiro Kamado
Races Won: 46
Sponsor: Redbull
--------------------------
```

Figure 6.6 Search Page

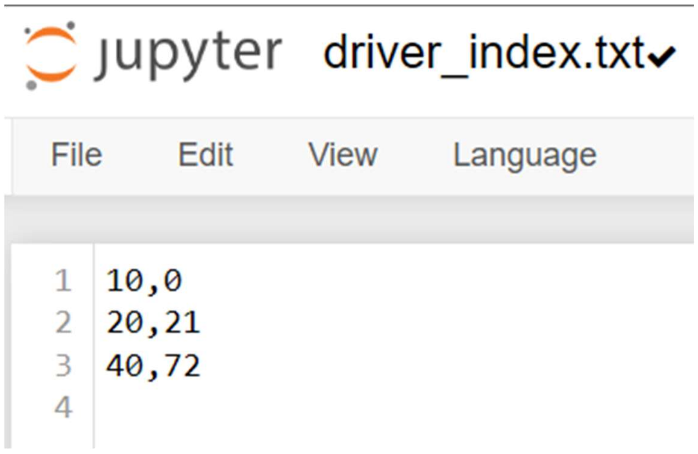
The fig 6.7 shows the exit page, after performing the required operation user can exit from the project by clicking the exit option from the main menu. After user selecting the exit option the project terminates.



```
----- Race Car Driver Management System -----
1. Add Driver
2. Display Drivers
3. Update Driver
4. Search Driver
5. Delete Driver
6. Exit
Enter your choice: 6
Exiting...
```

Figure 6.7 Exit Page

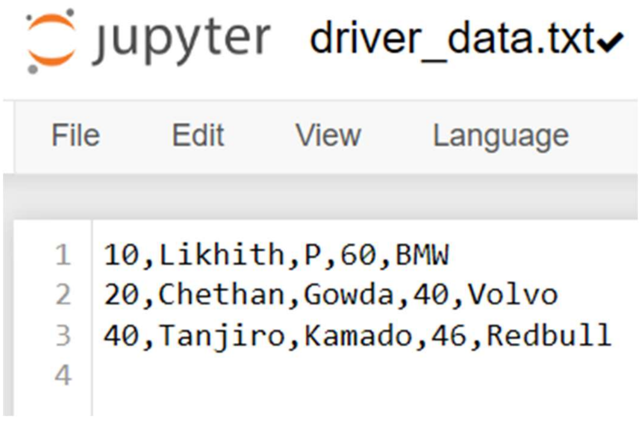
The fig 6.8 shows the backend content of the project. All the records inserted,Its index are stored in Index file. It is not shown to the user as it is the backend part of the project.



```
10,0
20,21
40,72
```

Figure 6.8 Index File

The fig 6.9 shows the backend content of the project. All the records inserted, are stored in Data file. It is not shown to the user as it is the backend part of the project.



```
10,Likhith,P,60,BMW
20,Chethan,Gowda,40,Volvo
40,Tanjiro,Kamado,46,Redbull
```

Figure 6.9 Data File

CONCLUSION

The Race Car Driver Management System is a comprehensive software solution designed to streamline the management of race car drivers and their related information. By centralizing driver data, facilitating efficient selection processes, and providing tools for contract management, performance analysis, scheduling, and communication, the system aims to enhance the overall effectiveness and efficiency of race car management teams. The Race Car Driver Management System serves as a valuable tool for race car management teams, organizers, and other stakeholders involved in the selection, management, and performance analysis of race car drivers. It provides a user-friendly interface, efficient data management, and comprehensive reporting capabilities, empowering users to make informed decisions and optimize their operations. The Race Car Driver Management System presents a robust solution for efficient and effective management of race car drivers. Its features and functionalities cater to the specific needs of race car management teams, providing a user-friendly and secure platform for optimizing driver selection, performance analysis, contract management, scheduling, and communication. The system holds great promise in elevating the capabilities and success of race car management in the dynamic world of motorsports.

REFERENCES

❖ Reference URL's:

- www.w3schools.com
- www.sourcecodesworld.com
- www.pythonprogramming.com
- www.youtube.com

❖ Reference Books:

- Bala guru swamy E , Programming in Python the Tata McGraw-Hill Companies, 8th Edition,2008.
- Kanetkar Yashavant, BPB Publication, 9th Edition, 2009.
- Gottfriend, Baryon S, Schaum's outlines Programming with python, the Tata McGraw- Hill, 2007.