**Heavy Vehicle Inspection and Rental Mobile Application**

**Abstract**

The Heavy Vehicle Inspection and Renting Application, developed using Flutter and SQLite, revolutionizes the way users interact with heavy vehicles and their operators. This comprehensive platform seamlessly integrates inspection and rental services for heavy vehicles, providing users with a one-stop solution for their transportation needs. Users can effortlessly find certified and reliable heavy vehicle operators for transportation tasks or inspection services, ensuring the efficiency and safety of their operations.

Through an intuitive user interface built with Flutter, the application allows users to browse and connect with a diverse network of skilled heavy vehicle operators. The integrated inspection feature ensures that users can assess the condition of the vehicles before committing to a rental, enhancing transparency and trust in the process. The application's backend, powered by SQLite, ensures robust data management and retrieval, enabling quick and accurate access to relevant information. Overall, this Heavy Vehicle Inspection and Renting Application streamlines the process of finding, inspecting, and renting heavy vehicles, fostering a more efficient and reliable ecosystem for both operators and users alike.

This application ensures farmers get their desired operators for their fields while cultivating crops, thus application makes those people to connect without hesitation. The application was developed to work with most heavy vehicles and their providers.

**Introduction:**

In an era where efficient transportation is paramount, the Heavy Vehicle Inspection and Renting Application emerges as a groundbreaking solution, meticulously crafted with Flutter and empowered by SQLite. This innovative platform addresses the burgeoning demand for streamlined heavy vehicle services, offering users a unique blend of convenience, reliability, and transparency. Whether businesses are seeking qualified heavy vehicle operators for transportation or individuals require a thorough inspection before renting, this application serves as the definitive hub, redefining the landscape of heavy vehicle management.

The user-friendly interface, developed using Flutter, ensures a seamless experience for individuals and businesses alike, enabling them to effortlessly navigate through a network of certified heavy vehicle operators. Beyond its user-centric design, the application's core functionality lies in its robust integration of SQLite, providing a secure and efficient database for storing and retrieving crucial information. As a result, users can confidently engage in the inspection and rental process, assured of the platform's commitment to data accuracy and real-time accessibility. This Heavy Vehicle Inspection and Renting Application heralds a new era in the logistics and transportation industry, ushering in a paradigm shift towards a more connected, efficient, and accountable heavy vehicle ecosystem.

**Existing System**

Before the advent of the Heavy Vehicle Inspection and Renting Application, the heavy vehicle management landscape grappled with inefficiencies and challenges in both inspection and rental processes. The conventional system often relied on fragmented approaches, with operators and users struggling to find a centralized platform for their heavy vehicle needs. Inspection services were disjointed and lacked a standardized framework, making it challenging for users to assess the condition of vehicles consistently. Additionally, the rental process was hindered by a lack of transparency and accessibility, as users had to rely on disparate channels to connect with operators, leading to delays and uncertainties

Fragmented Approach:

* The conventional system relied on fragmented processes for heavy vehicle management, lacking a centralized platform for both inspection and rental services.

Inspection Services:

* Inspection services were disjointed and lacked a standardized framework, making it difficult for users to consistently assess the condition of heavy vehicles.

Transparency Issues in Rental Process:

* The rental process suffered from a lack of transparency and accessibility. Users had to navigate through various channels to connect with operators, leading to delays and uncertainties.

Data Management:

* Data management was a significant bottleneck in the existing system, with manual record-keeping methods prone to errors and delays.

**Proposed Solution**

Centralized Platform:

* Our proposed Heavy Vehicle Inspection and Renting Application provides a unified and centralized platform, powered by Flutter, enabling users to seamlessly connect with certified heavy vehicle operators and access standardized inspection services in one location.

Enhanced Transparency:

* The application prioritizes transparency in the rental process by offering a user-friendly interface that facilitates direct communication between users and operators. This ensures a clear and efficient rental experience and eliminating the uncertainties and delays present in the existing system.

Efficient Data Management:

* Leveraging the capabilities of SQLite, our solution ensures robust data management. This eliminates the manual record-keeping bottlenecks present in the current system and providing a secure and efficient database for storing and retrieving critical information.

User-Centric Design:

* The proposed application is designed with the end-user in mind, offering a seamless experience through Flutter's user-friendly interface. This user-centric design enhances accessibility and usability, promoting a more efficient and satisfying heavy vehicle inspection and rental process for all stakeholders.

**System Requirements**

**Hardware Requirements:**

Processor : Intel(R) Core(TM) i5-6500T CPU @ 2.50GHz

Installed memory (RAM) : 8 GB

Hard Disk : 160 GB

Graphics Card : Intel(R) UHD Graphics

**Software Requirements:**

Operating System : Windows 10

Front-End : Flutter

Back-End : Dart, SQFlite, Path, Path provider, GetX

Tools : Android Studio, Visual Studio Code

Database : SQLite

**Software Description**

**Front End**

Flutter is an open-source UI (User Interface) software development framework created by Google. It is designed for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter was first introduced in 2017 and has gained significant popularity due to its unique approach to creating user interfaces and its ability to provide a consistent experience across different platforms.

Cross-Platform Development Flutter is a powerful framework for building cross-platform applications, meaning a single codebase can be used to create apps for both iOS and Android platforms. This eliminates the need for separate development teams or codebases for each platform, resulting in significant time and cost savings. Widgets and UI Components Flutter&#39;s core building blocks are widgets, which are used to create the user interface (UI) of the app. Flutter provides an extensive library of pre-designed widgets for various UI elements such as buttons, text inputs, images, lists, and more. These widgets are highly customizable, enabling developers to create consistent and visually appealing interface.

Responsive Design Flutter encourages the use of a responsive design approach. Widgets automatically adjust to different screen sizes and orientations, ensuring a consistent and pleasant user experience across various devices, including smartphones and tablets.

**Features**

**1. Handset Layouts:**

The platform is adaptable to larger, VGA, 2D graphics library, 3D graphic library based on OpenGL ES 2.0 specification, and traditional smartphone layouts.

**2. Storage:**

SQLite, a lightweight relational database, is used for data storage purposes.

**3. Connectivity:**

Android supports connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, and Bluetooth.

**4. Messaging:**

SMS and MMS are available forms of messaging, including threaded text messaging and now Android Cloud To Device Messaging (C2DM) is also a part of Android push messaging service.

**5. Web Browser:**

The web browser available in Android is based on the open-source Web kit layout engine, coupled with Chrome’s V8 JavaScript engine. The browser scores 100/100 on the Acid3 test on Android 4.0.

**6. Additional Hardware:**

Android can use video/still cameras, touch screens, GPS, accelerometers, gyroscopes, barometers, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers, accelerator 2D bit blots (with hardware orientation, scaling, pixel format conversion) and accelerated 3d graphics.

**7. Multi-Touch:**

Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. The features were originally disabled at the kernel level(possibly to avoid infringing Apple’s patents on touchscreen technology at the time). Google has since released an update for the Nexus One and the Motorola Droid which enables multi-touch natively.

**8. Applications:**

Applications are usually developed in the Java language using the Android Software Development Kit, but other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers and various Cross-platform mobile web applications frameworks.

**9. Google Play (Android Market):**

Android devices. An application program (“app”) called “Market” is preinstalled on most Android devices and allows users to browse and download apps published by third-party developers, hosted on Android Market. As of October 2011 there were more than 300,000 apps available for Android, and the estimated number of applications downloaded from the Android Market as of December 2011 exceeded 10 billion. The operating system itself is installed on 130 million total devices.

**10. Security:**

Android applications run in a sandbox, an isolated area of the operating system that does not have access to the rest of the system’s resources, unless access permissions are granted by the user when the application is installed. Before installing an application, the Play Store displays all required permissions. A gain may need to enable vibration, for example, but should not need to read messages or access the phone book. After reviewing these permissions, the user can decide whether to install the application.

**Different Dart Files Used In Android Studio:**

Flutter Widgets Files (.dart): These Dart files define the various widgets that make up your app’s user interface. Widgets are reusable building blocks, and each Dart file can define multiple widgets. Common widgets include StatefulWidget, StatelessWidget, and custom widgets you create. Main Dart File (main. dart): This is the entry point of your Flutter app. It’s where the void main() function is located, and it’s responsible for running your app. Widget Tests Files (widget\_test.dart): Flutter encourages testing, and widget tests allow you to test the behavior and rendering of your widgets. These test files are usually named with a \_test. dart suffix and are located in the test folder.

Asset Files (e.g., images, fonts): Your app might include asset files like images, fonts, or configuration files. These files are placed in the assets directory and are referenced in your pubspec.yaml file.

Pubspec File (pubspec. yaml): This YAML file defines your app&#39;s metadata, dependencies, assets, and other settings. You specify external packages and their versions here. It&#39;s also where you list assets that your app uses.

iOS and Android Configuration Files: While not specific to Flutter, your Flutter app will include iOS (.plist) and Android (AndroidManifest.xml) configuration files. These files contain platform-specific settings and configurations for your app.

Localization Files (.arb): If you&#39; re implementing internationalization (i18n) and localization for your app, you&#39; ll use .arb files to define translations for different languages. State Management Files: Depending on the state management solution you choose (e.g., Provider, Bloc, MobX), you might have additional files for managing app state and logic. Generated Files: Flutter generates certain files automatically. For example, when you add a new Flutter widget using Android Studio, it generates the corresponding Dart file and updates the entry in the lib/main. dart file.

Configuration Files: Your app might use various configuration files for integrating with services like Firebase, API keys, and more. These files are usually named according to the service they&#39; re related to. Android and iOS Project Files: While not typically interacted with directly in Flutter development, your project will contain Android (android) and iOS (ios) folders, each containing platform-specific code, resources, and configurations.

**Back End**

**Android SQLite**

SQLite is an open-source relational database i.e. used to perform database operations on Android devices such as storing, manipulating, or retrieving persistent data from the database. It is embedded in Android by default. So, there is no need to perform any database setup or administration task. SQLite is not directly comparable to client/server SQL database engines such as MySQL, Oracle, PostgreSQL, or SQL Server since SQLite is trying to solve a different problem.Client/server SQL database engines strive to implement a shared repository of enterprise data. They emphasize scalability, concurrency, centralization, and control. SQLite strives to provide local data storage for individual applications and devices. SQLite emphasizes economy, efficiency, reliability, independence, and simplicity. SQLite does not compete with client/server databases. SQLite competes with open ().

If many client programs are sending SQL to the same database over a network, then use a client/server database engine instead of SQLite. SQLite will work over a network file system, but because of the latency associated with most network file systems, performance will not be great. Also, file-locking logic is buggy in many network file system implementations (on both UNIX and Windows). If file locking does not work correctly, two or more clients might try to modify the same part of the same database at the same time, resulting in corruption.

**Android –Php/Mysql**

PHP started as a small open-source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994. PHP is a recursive acronym for &quot; PHP: Hypertext Preprocessor. PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, and even build entire e-commerce sites. It is integrated with several popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the UNIX side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time and supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time. PHP is forgiving and PHP language tries to be as forgiving as possible Syntax is the same as C language.

MYSQL is used as a database at the web server and PHP is used to fetch data from the database. The application will communicate with the PHP page with the necessary parameters and PHP will contact the MYSQL database and will fetch the result and return the results to us.

**GetX**

GetX is a powerful state management, dependency injection, and route management library for Flutter applications, offering a lightweight yet highly efficient solution for developers. With GetX, Flutter developers can streamline their app development process by simplifying common tasks such as managing state, navigating between screens, and handling dependencies. One of the standout features of GetX is its simplicity and ease of use, allowing developers to write clean, concise code without sacrificing functionality or performance.

At the core of GetX is its reactive state management system, which leverages the power of reactive programming to automatically update UI components in response to changes in application state. This allows developers to build highly responsive and interactive user interfaces with minimal effort. GetX also provides a comprehensive set of tools for managing app routes, making it easy to define and navigate between different screens within the application. Additionally, GetX includes a powerful dependency injection system, which simplifies the process of managing and injecting dependencies throughout the app, resulting in cleaner and more maintainable code.

Overall, GetX is a versatile and feature-rich package that empowers Flutter developers to build high-quality applications with ease. Whether you're a beginner looking to streamline your development workflow or an experienced developer seeking a lightweight and efficient state management solution, GetX is definitely worth considering for your next Flutter project.

**Path**

The `path` package in Flutter is an essential tool for handling file system paths in a platform-independent manner. It provides a set of utilities for manipulating file paths, making it easier to work with directories and files across different platforms, including iOS, Android, and web. With the `path` package, Flutter developers can avoid hardcoding file paths and instead rely on a consistent and reliable API for constructing, joining, and manipulating paths.

One of the key features of the `path` package is its ability to handle file system intricacies specific to different platforms seamlessly. This means that developers can write code that works consistently across various operating systems without worrying about platform-specific quirks or inconsistencies. Whether it's constructing paths to assets bundled with the application, accessing files stored locally on the device, or working with directories in the application's sandboxed environment, the `path` package provides the necessary tools to accomplish these tasks efficiently and reliably.

In addition to basic path manipulation functions, the `path` package also offers utilities for resolving relative paths, normalizing paths, and determining the extension of a file. These features further enhance the package's versatility and make it a valuable asset for Flutter developers building file-intensive applications. Overall, the `path` package simplifies file system operations in Flutter, allowing developers to focus on building robust and feature-rich applications without getting bogged down by platform-specific file path handling.

**Path Provider**

The `path\_provider` package in Flutter serves as a crucial bridge between your application and the underlying file system of the device it's running on. It provides a unified API for accessing directories such as the documents directory, temporary directory, and the application directory, ensuring consistency across different platforms like iOS, Android, and web. By abstracting away the platform-specific details, `path\_provider` simplifies file management tasks and enhances the portability of Flutter applications.

One of the most common use cases for the `path\_provider` package is accessing the documents directory, where user-generated content like files, images, or databases are often stored. Through simple method calls, developers can obtain the path to this directory and perform read and write operations on files within it. Similarly, the package offers access to the temporary directory, which is useful for storing data that is only needed temporarily and can be safely deleted after use, such as cached files or temporary downloads.

Another noteworthy feature of the `path\_provider` package is its support for platform-specific directories, such as the application directory, where assets bundled with the application are stored. This functionality enables developers to access resources packaged with their app, like images or configuration files, and use them within their Flutter application seamlessly. Overall, `path\_provider` simplifies file system interactions in Flutter, making it easier for developers to manage files and directories across different platforms while ensuring compatibility and reliability.

**Project Description**

**System Modules**

The system comprises one major module with its sub-modules as follows:

**User:**

**Home Page:** It consists of suggestions where people can review their type of need in each suggestion

**Posting Page:** In this page users can input specific data for the need of technician according to their work

**Booking Page:** After going through various number of profile they have posted in the application, users can select the correct candidate for their required need

**Account Page:** The user login data can be shown over here

**Technician:**

**Home Page:** The posted data from user can be shown over here and technician can bid the required or correct position in the section

**Booking Page:** The booking page shows the selected post.

**BidsPage:** Technician can bid the work accordingly.

**Account Page:** The user login data can be shown over here

**Data Flow Diagram**

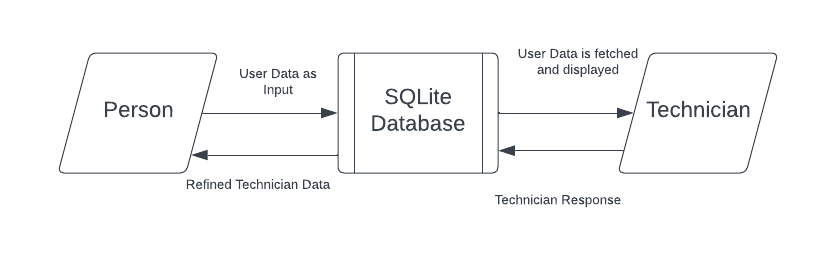
A data flow diagram (DFD) is a graphical representation of the “Flow” of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will stored.

DFD shows how the information moves through the system and how it’s modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

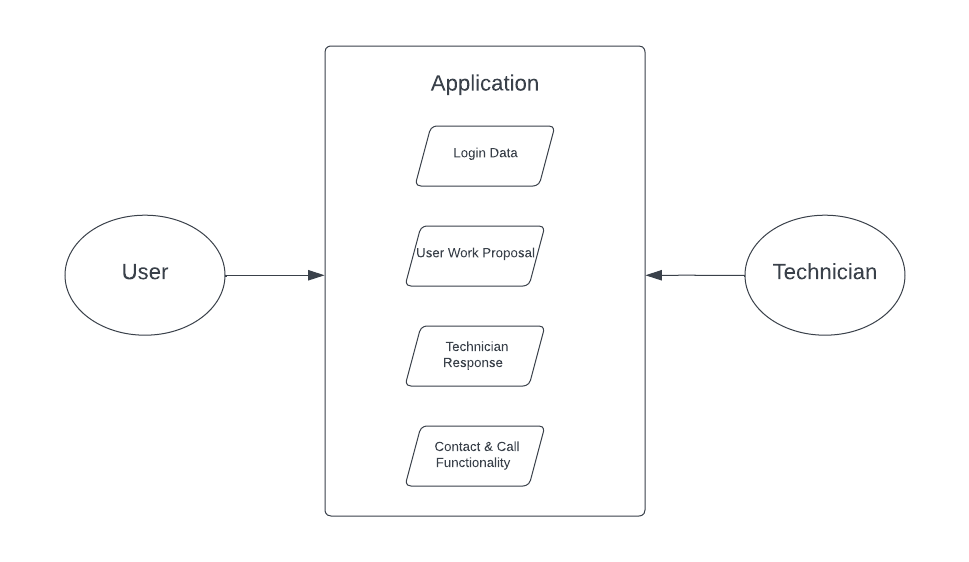
The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

**DFD diagrams**

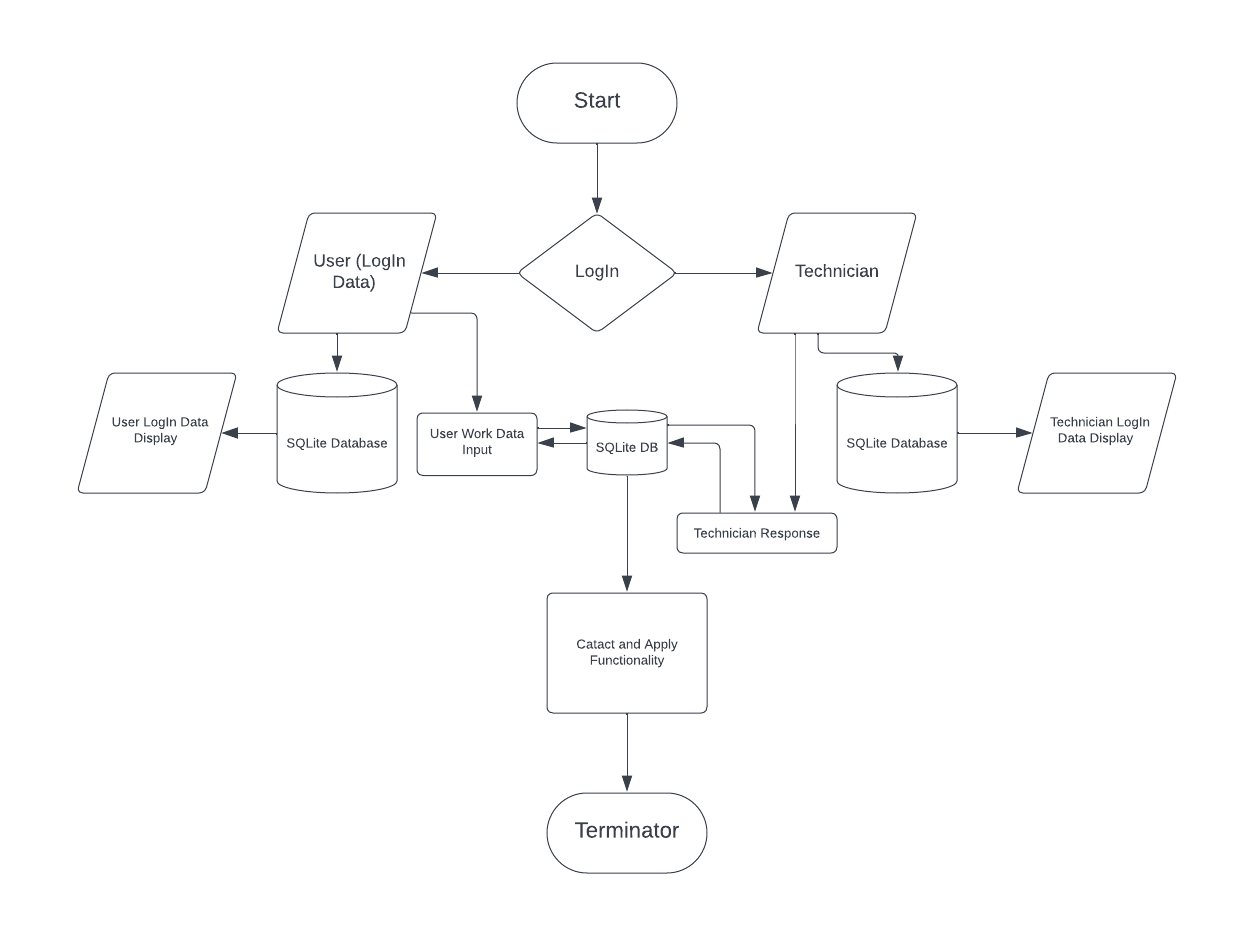
**Level 0:**



**Level 1**



**System Flow Diagram**



**Design Phase**

* 1. **Introduction**

The design phase is the period in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements.

Details on computer programming language and environments, application packages, layering, memory size, platform, algorithms, data structure, interfaces, and many others. The design may include the usage of existing components. The components are as follows

* Input design
* Output design
* Code design
* Database design
  + 1. **Input Design**

Input design is the process of converting an external user-oriented description of the input system into a machine-oriented format. The source document is prepared for the input of data to make the entry accurate and impact the source document was prepared. The data elements were sent out in a system that the data entry operator would easily follow.

* + 1. **Objectives Of Input Design**

In accuracy, input data is the most common cause of errors in data processing. Errors found at the data entry can be controlled by the proper input design. The input was designed with the following objectives

* To produce a cost-effective method.
* To get the highest level of accuracy.
* To ensure that the input is acceptable and understandable by the people who are using it. With the above objectives, the major activities that were done during the input design are
* The data are collected from its source.
* Data are converted to the mobile acceptable form.
* The converted data was verified.
  1. **Code Design**

The process of code is to facilitate the identification and retrieval of items of information. The code should be written such that it is simple and easy to understand. The following characteristics were also considered while designing the code.

* Ambiguity
* Uniqueness
* Simplicity
* Stability
* Easy to update

The code should be adequate for present and anticipated data processing for machine and human use.

* 1. **Database Design**

Data is pivotal to any organization's activities. The objectives of the database design are to store the captured data efficiently and to facilitate timely retrieval. Proper attention has to be taken in designing the database.

The database is the heart of any information system any information must be collected, accumulated, edited, and retrieved. Therefore it should directly need files should be properly designed and planned, Files are a crucial component for the successful performance of the system. File design includes master files and transaction files.

* 1. **Output Design**

Output design is a process that involves designing necessary output that has to be given to various users according to their requirements, Efficient, intelligible output design will improve the system's relationship with the user and help indecision-making. Since the reports are directly required by the management to make decisions and to draw conclusions, they must be designed with care for the user. The options for the output and report are given in the system menu. When designing output, system analyst must accomplish the following

* Determine the information to present.
* Arrange the presentation of information acceptable format.
* Determine how to distribute the output.

**Table Design**

Screen design begins with the recognition that the screen is composed of different areas. Layout tools assist the analyst in specifying the content of the signal and multiple design formats. All screens have been provided with menus, push button facilities, icons, and control buttons. The main screen consists of two buttons which we can move to another screen. In designing output screens we need area for

* Heading and titles.
* The content of the display.
* Message and instructions.
* Sometimes explanations for information in the report.

A database is a collection of data about a specific topic.

**Table Name:** post\_table

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **DATA TYPE** | **DESCRIPTION** |
| name | TEXT | Name of the user |
| fromdate | TEXT | From date to start work |
| todate | TEXT | To date to stop work |
| location | TEXT | Working location |
| zipcode | TEXT | Zip code of the posting user |
| address | TEXT | Address of the posting user |
| questions | TEXT | Questions for particular problems |
| technician | TEXT | Which kind of technician needed |

* 1. **Interface Description**
  2. **Internal Interface**

A key principle of design is to prohibit access to all resources by default, allowing access only through well-defined entry points, i.e. interfaces. Software interfaces provide access to computer resources (such as memory, CPU, storage, etc.) of the underlying computer system; direct access (i.e. not through well designed interfaces) to such resources by software can have major ramifications sometimes disastrous ones for functionality and stability. Interfaces between software components can provide: constants, data types , types of procedures, exception specification and method signatures. Sometimes, public variables are also defined as part of an interface.

* + 1. **External Interface**

Hardware interfaces exist in many of the components such as the various buses, storage devices, other I/O devices, etc. A hardware interface is described by the mechanical, electrical and logical signals at the interface and the protocol for sequencing them (sometimes called signaling). A standard interface, such as SCSI, decouples the design and introduction of computing hardware, such as I/O devices, from the design and introduction of other components of a computing system, thereby allowing users and manufacturer’s great flexibility in the implementation of computing systems. Hardware interfaces can be parallel with several electrical connections carrying parts of the data simultaneously, or serial where data is sent one bit at a time.

* + 1. **User Interface**

The **User Interface** (**UI**), in the industrial design field of human–machine interaction, is the space where interactions between humans and machines occur. The goal of user interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process. Examples of the broad concept of user interfaces include the interactive aspects of computer operating systems, hand tools, heavy machinery operator controls, and process controls.

* 1. **Functional Description**
     1. **Processing Narrative**

Processing narrative deals with the function, which are used to solve the problem in this project. Description of each function is required to solve is presented to the functional description .Processing narrative is provided for each function, design constraints are stated and justified. Performance characteristics will be stated also one more diagrams are includes to represent the overall system structure of software.

**Restrictions And Limitations**

Restrictions and limitations are nothing but level of work where a user cannot continue further. The login screen is made with the authorization username and password the overall application is only handled by the students and they can login after registration.

* Unauthorized users login are restricted
* Registered users are only eligible to login the system.
  + 1. **Performance Requirements**

**Security**

The application is secured it doesn’t provide any way to enter the users other than the authorized user.

**Availability**

Availability is the probability that a program is operating according to requirements at a given point of time. The availability is an indirect measure of the maintenance of the software. The functionality of the program is immediately available once it has been logged in.

**Response Time**

Response time is the time with in which an application identifies the instruction of the customer and response to it. The response time of the application is made efficient and quick so that the requests are accepted according to their validity and the result can be delivered to the customer with an immediate effect.

**Behavioural Description System Event Anactions**

A state is any observable mode of behaviour member state, element state is two important states represent a mode of behavior of the system. Member state has two states, active and deactivate, normally all members are in de-active state, there corresponding action is member activation. As well as to change active to deactivate state same event of member detail modification should be fired respective action is member de-activation.

The system uses various kinds of events and states. System state represents the action that starts when an event occurred. The system contains the events such as Load, Click, and Selected Index Changed etc.

**Load Event**

Load event is the first event which is focused at the time of execution. Mainly the connection and initialization activities are done under this event.

**Click Event**

The controls like button and hyperlinks has click event as a default event. It helps to accelerator the process.

**Selected Index Changed**

This event is mainly used in the drop down list boxes, list boxes, radio buttons and checkboxes. In these types of events the process are carried out according to the choice selection.

**Testing And Implementation**

* 1. **Testing**

System testing is a stage of implementation which is aimed at ensuring that the system works accurately and efficiently . The primary goal of the test process is to make sure that the application performs as discussed in the requirement and specifications. Testing begins at the module level and works towards the integration of the entire computer based system.

Testing methodologies

* Unit Testing
* Widget Testing
* Integration Testing

**Unit Testing**

Unit testing in Flutter involves testing individual units of code, such as functions, methods, or classes, in isolation to ensure they behave as expected. Flutter provides the test package for writing and executing unit tests. The goal is to ensure that each unit of code behaves correctly in isolation. By regularly running unit tests as part of your development workflow, you can catch issues early and maintain a high level of code quality in your Flutter application.

* 1. **Widget Testing**
  2. Widget testing in Flutter involves testing individual widgets and their interactions within a Flutter app. It's a type of testing that focuses on the visual and interactive aspects of your app's user interface. Widget tests are particularly useful for ensuring that your widgets render correctly, respond to user interactions as expected, and maintain their behavior across different app states. By writing and running widget tests regularly, you can catch visual and interactive issues early in the development process and maintain a stable user interface.

**Integration Testing**

* 1. Integration testing in Flutter involves testing the interaction between different components, screens, or widgets of your app to ensure they work together as expected. Integration tests help you ensure that different parts of your app interact correctly and that your app's overall functionality is not compromised as you make changes or add new features. By regularly running integration tests, you can catch issues related to navigation, state management, and widget interactions early in the development process.
  2. **Test Cases**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case id** | **Test objective** | **Test data** | **Expected results** | **Actual result** | **Test result** |
| Tc 001 | name | Valid name | Show error | Accept | Pass |
| Tc 002 | name | Invalid name | Accept | Accept | Pass |
| Tc 003 | email | Valid email | Show error | Accept | Pass |
| Tc 004 | email | Invalid email | Accept | Accept | Pass |
| Tc 005 | phonenumber | Valid number | Show error | Accept | Pass |
| Tc 006 | phinenumber | Invalid number | Accept | Accept | Pass |
| Tc 007 | spcialist | Valid data | Show error | Accept | Pass |
| Tc 008 | specialist | Invalid date | Accept | Accept | Pass |
| Tc 009 | location | Valid location | Show error | Accept | Pass |
| Tc 010 | location | Invalid location | Accept | Accept | Pass |

* 1. **System Implementation**

System Implementation is the stage of the project when the theoretical design is turned into practical system. After proper testing and validation, system implementation should be done. System implementation includes all those activities that take a place to convert an old system to the new system.

**Conclusion**

The creation of a heavy vehicle inspection application using Flutter and SQLite, complemented by call functionality, signifies a pivotal shift towards modernizing and streamlining the inspection processes for heavy vehicles. Flutter's robust cross-platform capabilities ensure a seamless and consistent user experience on both iOS and Android devices, making the application accessible to a wide range of users, from vehicle inspectors to fleet managers. The choice of SQLite for data management provides a lightweight yet powerful solution for storing and retrieving critical inspection data efficiently, supporting offline capabilities that are essential for fieldwork in remote locations where internet connectivity may be unreliable.

Incorporating call functionality within the application facilitates direct communication between inspectors, vehicle owners, and service providers, enhancing the clarity and speed of addressing identified issues. This level of integration not only speeds up the inspection process but also contributes to maintaining high safety standards by ensuring that vehicle issues are promptly and effectively communicated and resolved. As the application continues to evolve, future enhancements could focus on integrating advanced diagnostics tools, AI-driven analytics for predictive maintenance, and blockchain technology for secure and transparent record-keeping. Such advancements will further empower stakeholders in the heavy vehicle industry with tools that promote efficiency, safety, and compliance, ultimately elevating the standard of heavy vehicle operations and management.

**Future Enhancement**

Looking ahead, the heavy vehicle inspection application built with Flutter and SQLite, and featuring call functionality, is poised for significant future enhancements that could revolutionize the way heavy vehicle inspections are conducted. A major area for development is the integration of advanced diagnostic tools and Internet of Things (IoT) technology. By connecting the application directly with vehicle diagnostic systems, inspectors could automatically gather critical data on a vehicle's condition in real-time, streamlining the inspection process and enhancing the accuracy of assessments. IoT integration would also enable continuous monitoring of vehicle health, facilitating preventative maintenance and reducing the likelihood of critical failures on the road.

Another promising direction for future enhancement involves the application of artificial intelligence (AI) and machine learning (ML) algorithms. These technologies could be used to analyze historical inspection data, identifying patterns and predicting potential issues before they become serious problems. This predictive maintenance capability would not only improve vehicle safety and reliability but also help fleet managers optimize maintenance schedules, reducing downtime and operational costs. Additionally, implementing blockchain technology could enhance the security, transparency, and efficiency of inspection records management. By creating an immutable ledger of inspection data, stakeholders can ensure the authenticity and accuracy of vehicle histories, boosting trust among inspectors, vehicle owners, and regulatory bodies.

These future enhancements would position the heavy vehicle inspection application at the forefront of digital transformation in the transportation sector. By leveraging cutting-edge technologies to improve the efficiency, accuracy, and predictive capabilities of vehicle inspections, the application will contribute to safer roads and more sustainable heavy vehicle operations, aligning with broader industry trends towards automation and data-driven decision-making.