

SYSTEM ANALYSIS AND DESIGN

3.1 Introduction:

System analysis is the application of the system approach to problem solving using computers. Analyst must consider its elements like outputs and input processors, control, feedback and environment when constructing the system.

The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the step-1 moving from the problem domain to solution domain. It is also a bridge between requirements specifications and final solution for satisfying requirements.

The design activity is often divided into two separate process – System design and Detailed Design. System Design is also called as Top-Level design. It aims to identify the modules that should be in the system, specifications of these modules and how they interact with each other to produce the desired result. At the end of System Design all the major data structures, file formats, major modules in the system and their specifications are decided.

3.2 Logical Design:

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modelling, using an over-abstract (and sometimes graphical) representation of the actual system. In the context of system design, modelling can undertake the following parts, including:

- Data Flow Diagram
- Entity Relationship Diagram

3.3 Physical Design:

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified.

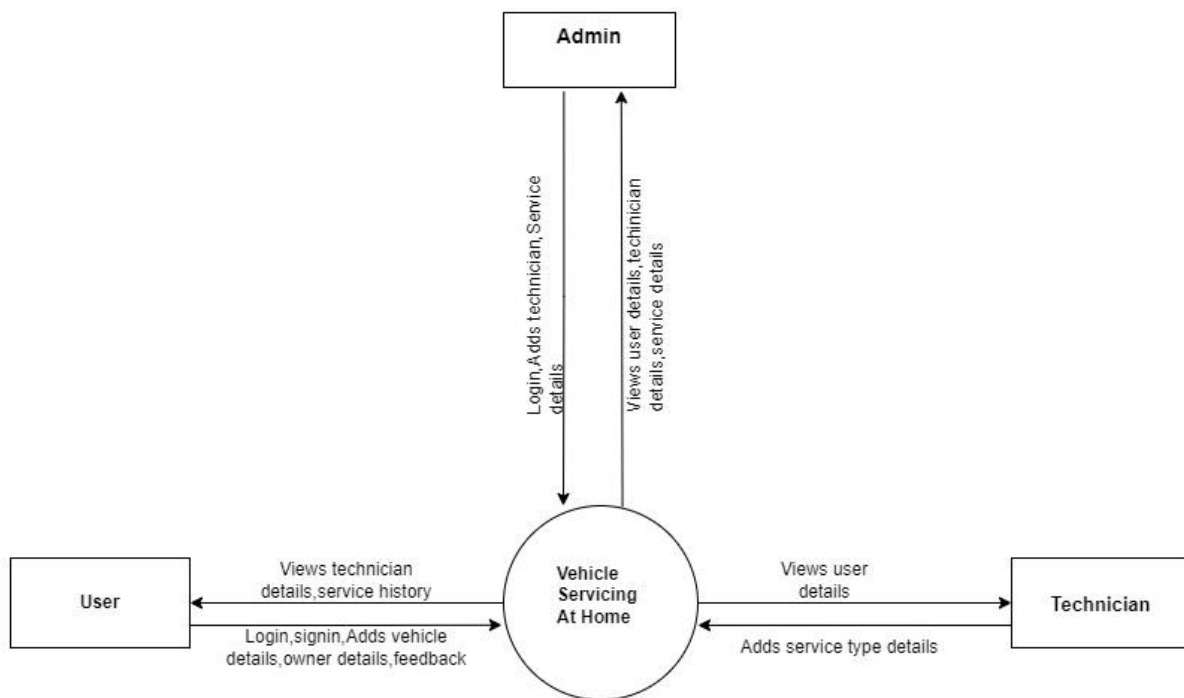
3.4 Applicable Document:

The applicable document to be referred here is Software Requirement Specification.

3.5 Description of Programs

3.5.1 Context Flow Diagram:

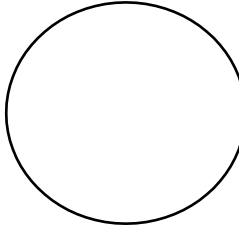

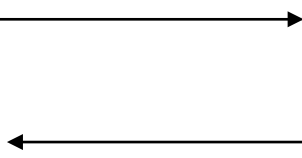
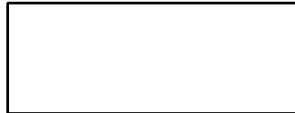
Context flow diagram is a top-level data flow diagram. It only contains one process node that generalizes the function of the entire system in relationship to external entities. In context diagram the entire system is treated as a single process and all its inputs, outputs, sinks and sources are identified and shown.



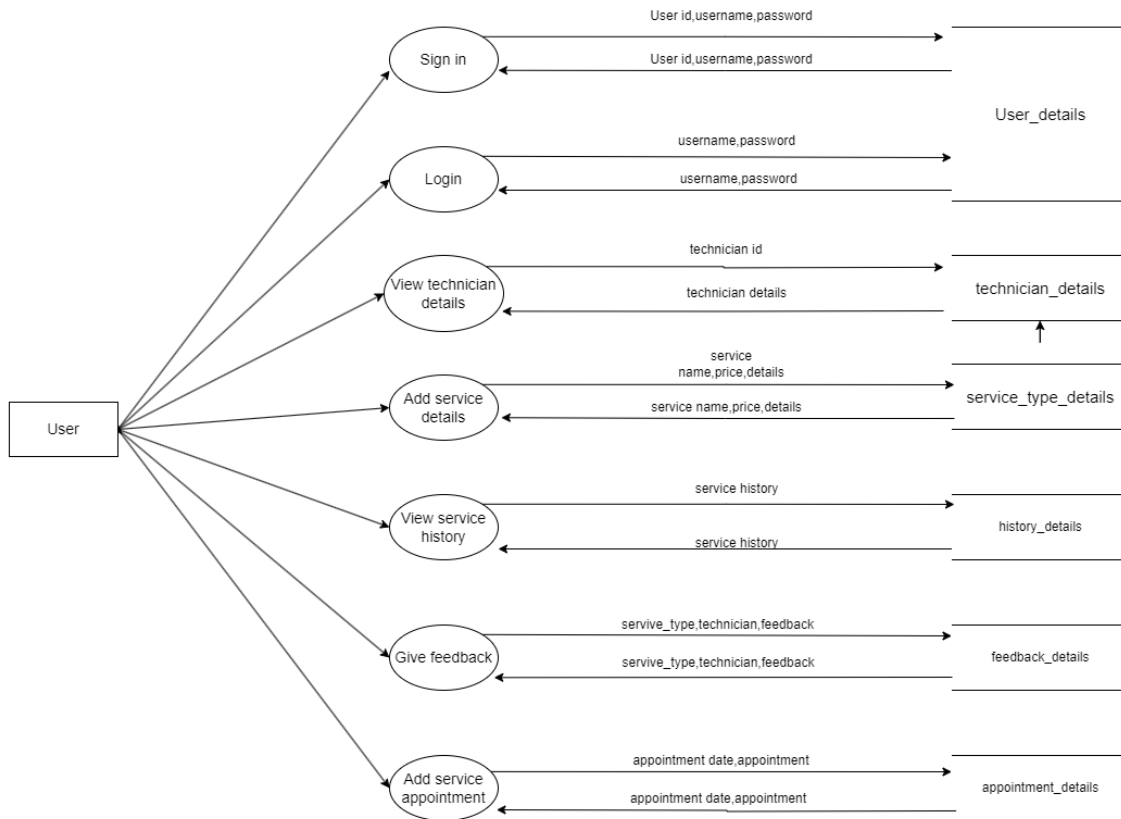
3.5.2 Data Flow Diagrams:

A DFD shows the flow of data through system. It views a system as a function that transforms the input into desired output. Data flow diagram is drawn to pictorially represent the data stored, processed, entities involved in the system and information flow. DFDs are useful in understanding a system and can be efficiently used during problem analysis.

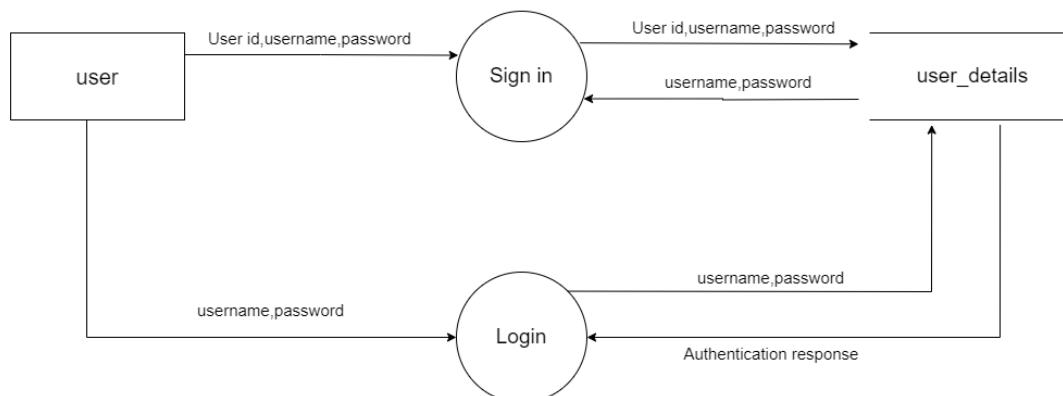
DFD Symbols:

Name	Notation	Description
Process		A process transforms incoming data flow into outgoing data flow. The processes are shown by named circles.
Datastore		Data stores are repositories of data in the system. They are sometimes also referred to as files.
Dataflow		Data flows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.
External Entity		External entities are objects outside the system with which the system communicates. External Entities are sources and destinations of the system's inputs and output.

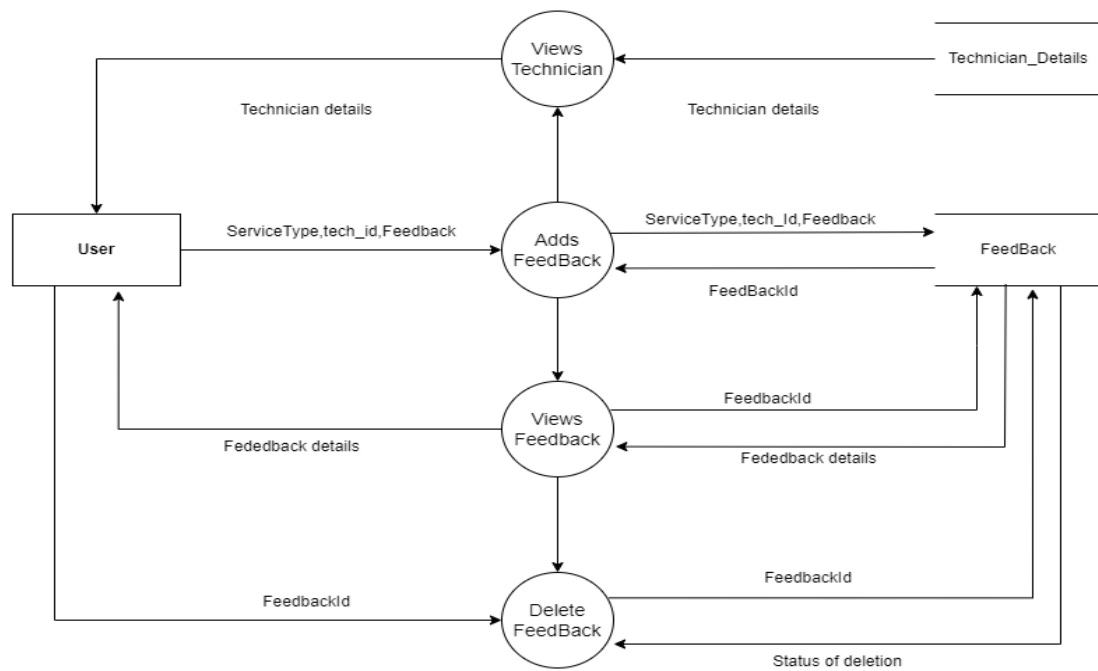
LEVEL 1 DFD USER



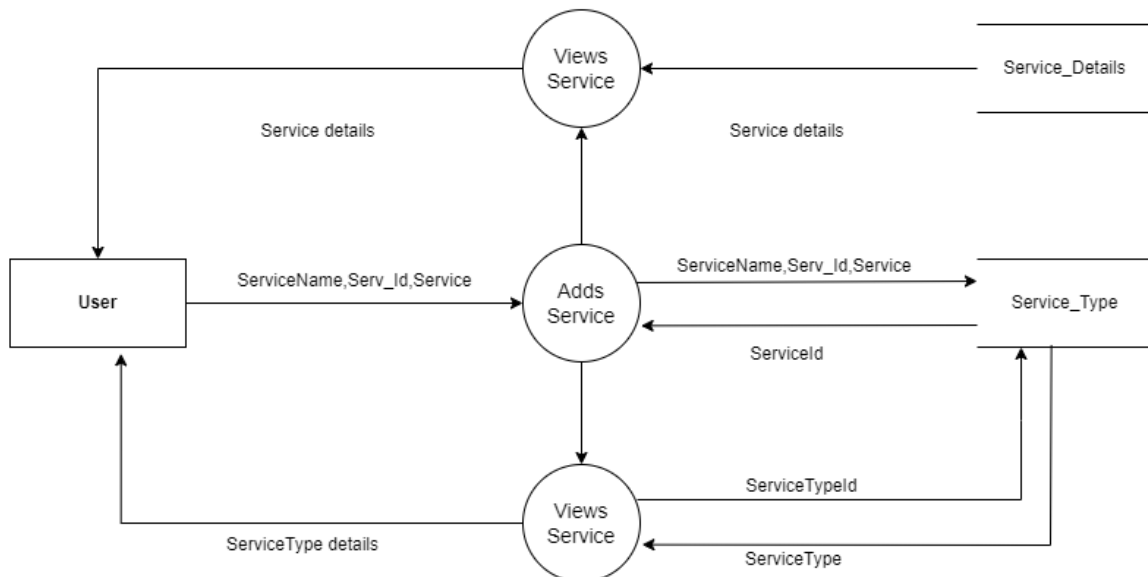
LEVEL 2 DFD USER SIGNIN AND LOGIN



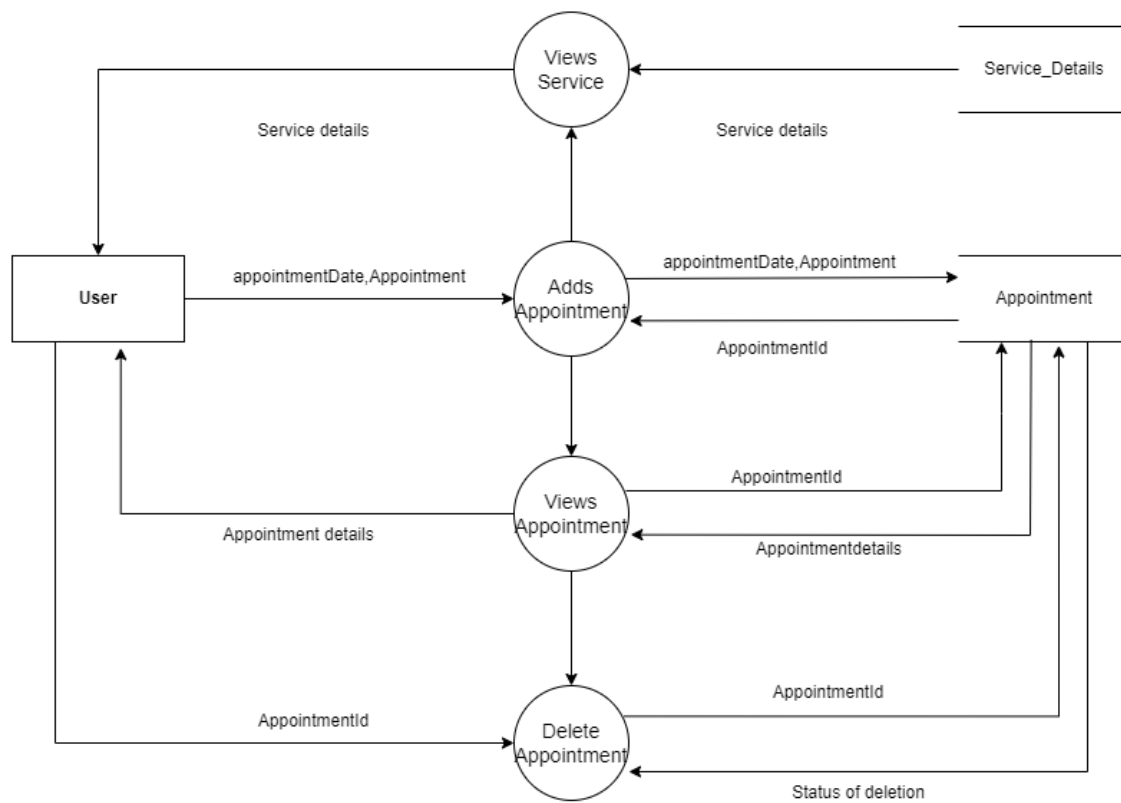
LEVEL 2 DFD USER FEEDBACK



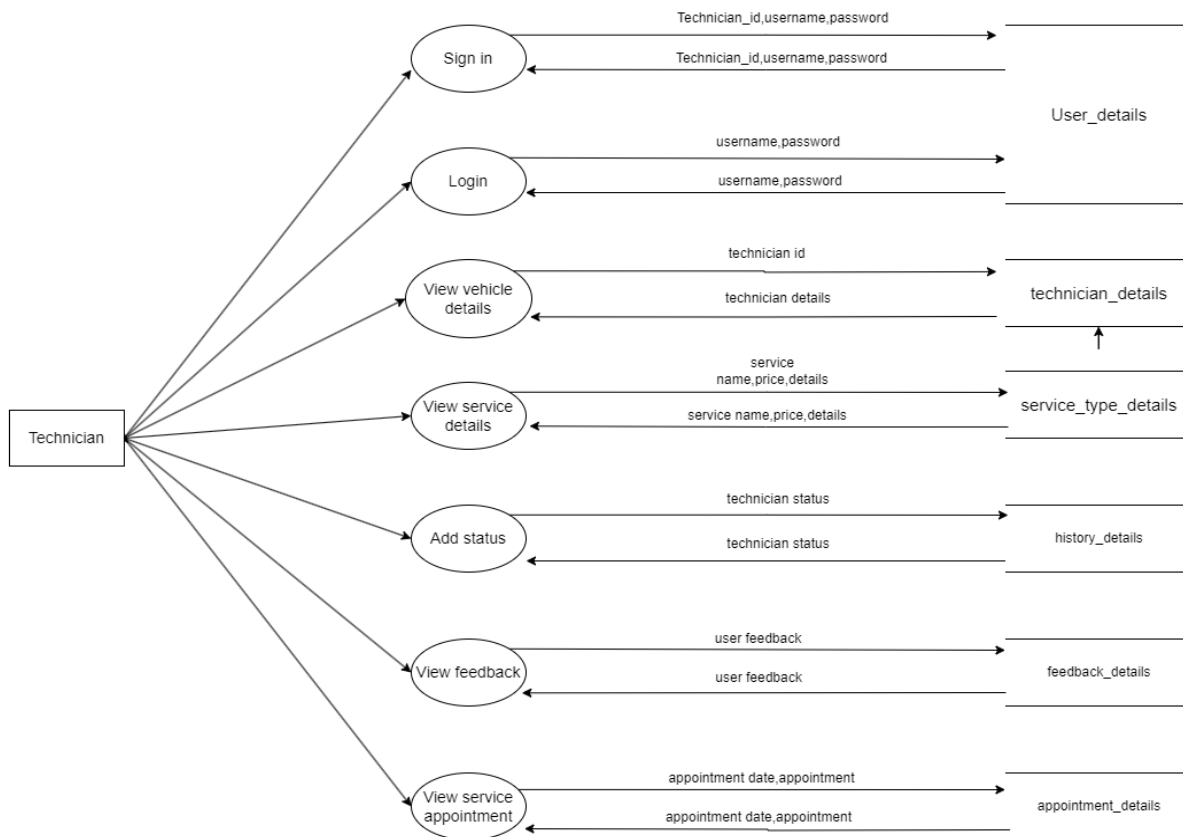
LEVEL 2 DFD USER SERVICE



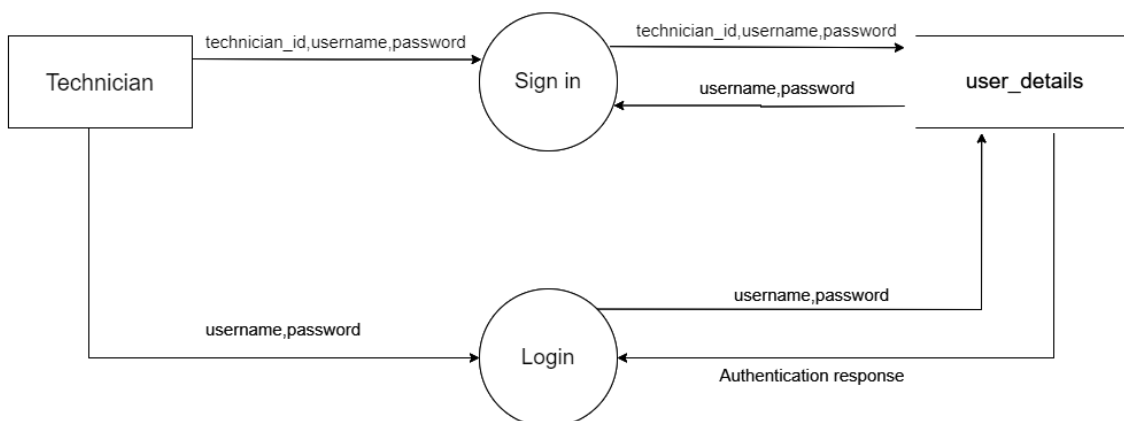
LEVEL 2 DFD USER APPOINTMENT



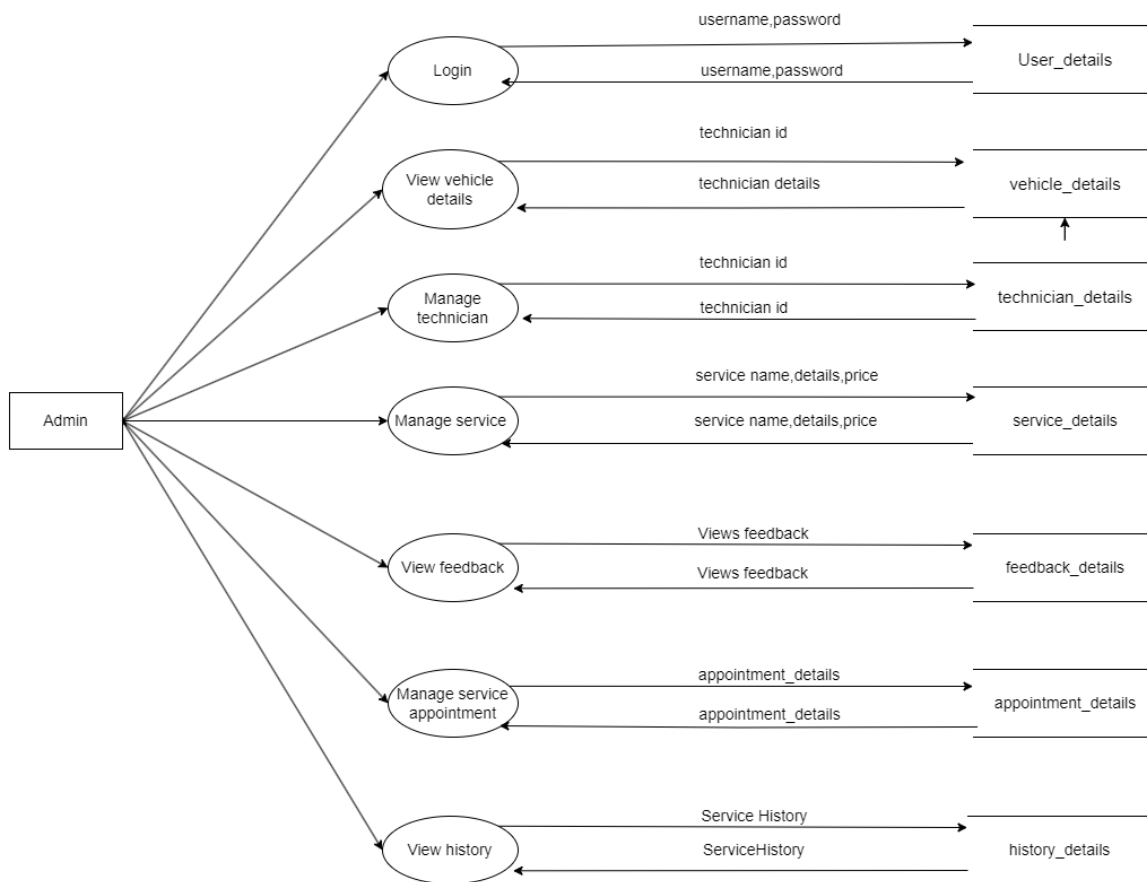
LEVEL 1 DFD TECHNICIAN



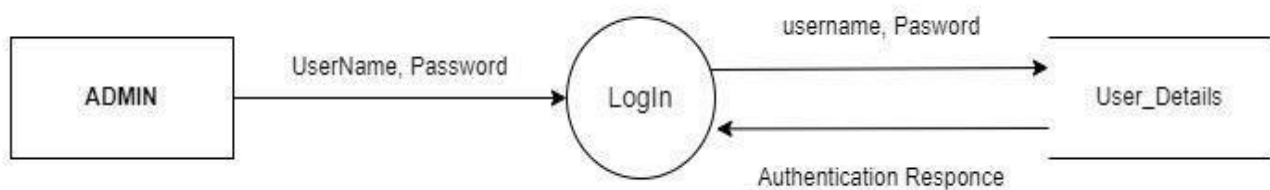
LEVEL 2 DFD TECHNICIAN SIGNIN AND LOGIN



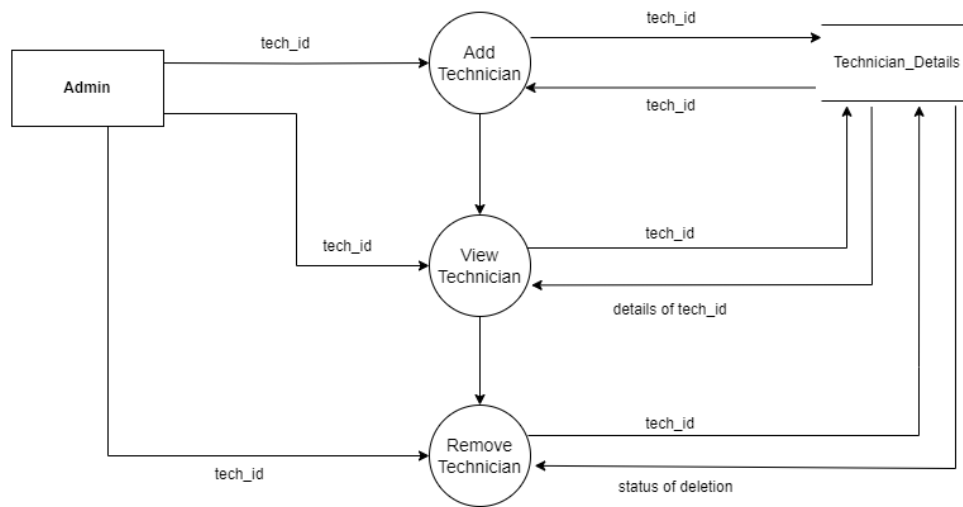
LEVEL 1 DFD ADMIN



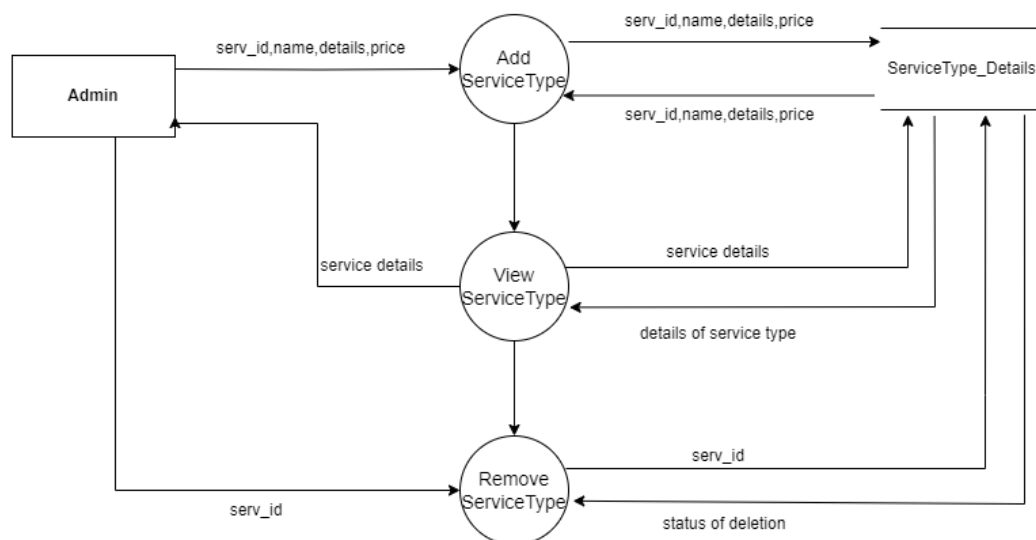
LEVEL 2 DFD ADMIN LOGIN



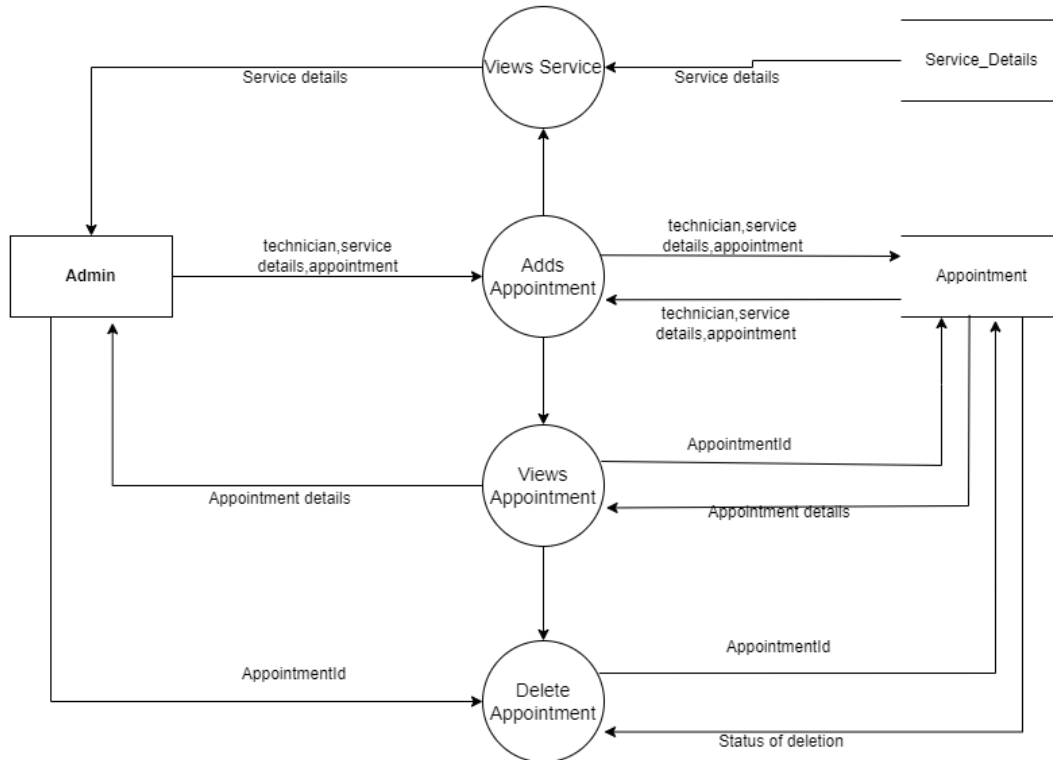
LEVEL 2 DFD ADMIN MANAGE TECHNICIAN



LEVEL 2 DFD ADMIN MANAGE SERVICE



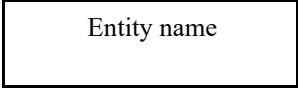
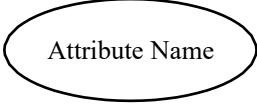
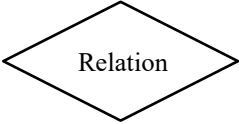


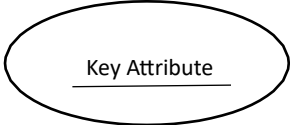
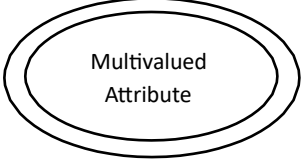
LEVEL 2 DFD ADMIN MANAGE APPOINTMENT



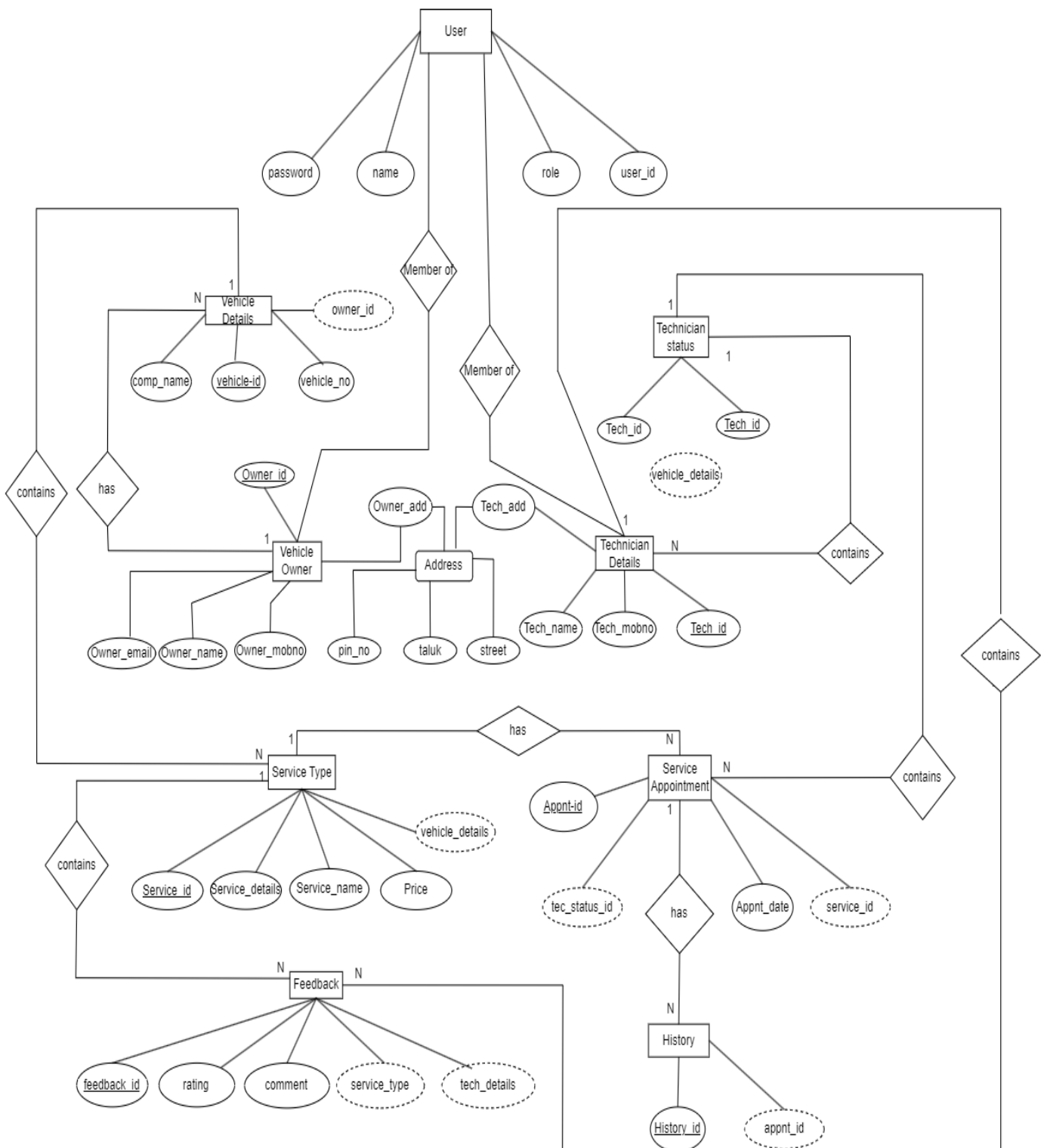
3.4 Entity Relationship Diagram:

The basic objective of the ER model representation is an entity which is a “thing” in a real world with an independent existence. Entities are physical items or aggregations of data items that are important to the business we analyze or to the system; we intend to build. An entity represents an object defined within the information system about which you want to store information. Entities are named as singular nouns and are shown in rectangles in an ER Diagram.

ER-Diagram Symbols and Description:

Name	Notation	Description
Entity		It may be an object with the physical existence or conceptual existence. It is represented by a Rectangle.
Attribute		The properties of the entity can be an attribute. It is represented by an ellipse.
Relationship		Whenever an attribute of one entity refers to another entity, some relationship exists. It is represented by a diamond
Link		Lines links attributes to entity sets and entity sets to relation
Derived Attribute		Dashed ellipse denotes derived attribute
Key Attribute		An entity type usually has an attribute whose values are distinct for each individual entry in the entity set.it is denoted by an underlined word in an ellipse.
Multivalued Attribute		Attributes that have different numbers of values for a particular attribute. It is represented by a double ellipse.
Cardinality Ratio	<ol style="list-style-type: none">1:11:NN:1M:N	It specifies the maximum number of relationships that an entity can participate in. There are four cardinality ratios.

ER Diagram:



3.1 Class Diagram:

A Class diagram in the Unified Modelling Language is a type of static structure that describes the structure of a system by showing the system's classes, their attributes, operations and the relationships among the objects. The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.









A UML class diagram is made up of:

- A set of classes.
- A set of relationships between classes.

‡ **Class Notation** A class notation consists of three parts:

- **Class Name:** The name of the class appears in the first partition.
- **Class Attributes:** Attributes are shown in the second partition. The attribute type is shown after the colon. Attributes map onto member variables (data members) in code.
- **Class Operations (Methods):** Operations are shown in the third partition.
 1. They are services the class provides.
 2. The return type of a method is shown after the colon at the end of the method signature.
 3. The return type of method parameters is shown after the colon following the parameter name.
 4. Operations map onto class methods in code.

‡ **Visibility** To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the member's name:

Icon for field	Icon for method	Visibility
		private
		protected
		package private
		public

Class Relationships: A class may be involved in one or more relationships with other classes.

Inheritance (or Generalization):

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class.

‡ **Aggregation:** A special type of association. It represents a "part of" relationship. • Class2 is part of Class1.

- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite.

‡ **Composition:** A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite.

Dependency: ○ Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).

- Class1 depends on Class2.
- A dashed line with an open arrow.

