**Chapter 4-Implementation**

**4.1 Models**

SDLC stands for Software Development Life Cycle. SDLC is a process that consists of a series of planned activities to develop or alter the Software Products.

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software’s. The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

* SDLC is the acronym of Software Development Life Cycle.
* SDLC is a framework defining tasks performed at each step in the software development process.
* ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.[10]

**What is SDLC?**

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



Figure4.1.1SoftwareDevelopmentLifeCycle

A typical Software Development Life Cycle consists of the following stages −

**Stage 1: Planning and Requirement Analysis**

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

**Stage 2: Defining Requirements**

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

**Stage 3: Designing the Product Architecture**

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

**Stage 4: Building or Developing the Product**

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

**Stage 5: Testing the Product**

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

**Stage 6: Deployment in the Market and Maintenance**

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

**SDLC Models**

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry &miuns;

* Waterfall Model
* Iterative Model
* Spiral Model
* V-Model
* Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

**Agile Model**

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

* Planning
* Requirements Analysis
* Design
* Coding
* Unit Testing and
* Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

**What is Agile?**

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model −



Figure 4.1.3.1: Agile model

The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles −

* **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
* **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
* **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
* **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

**Agile Vs Traditional SDLC Models**

Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

**Customer Interaction** is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.[10]

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

**The advantages of the Agile Model are as follows −**

* Is a very realistic approach to software development
* Promotes teamwork and cross training.
* Functionality can be developed rapidly and demonstrated.
* Resource requirements are minimum.
* Suitable for fixed or changing requirements
* Delivers early partial working solutions.
* Good model for environments that change steadily.
* Minimal rules, documentation easily employed.
* Enables concurrent development and delivery within an overall planned context.
* Little or no planning required.
* Easy to manage.
* Gives flexibility to developers.

**The disadvantages of the Agile Model are as follows −**

* Not suitable for handling complex dependencies.
* More risk of sustainability, maintainability and extensibility.
* An overall plan, an agile leader and agile PM practice is a must without which it will not work.
* Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
* Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
* There is a very high individual dependency, since there is minimum documentation generated.
* Transfer of technology to new team members may be quite challenging due to lack of documentation.

**USECASE DIAGRAM**

**Use case diagrams** are usually referred to as [behaviour diagrams](http://www.uml-diagrams.org/uml-25-diagrams.html#behavior-diagram) used to describe a set of actions ([use cases](http://www.uml-diagrams.org/use-case.html)) that some system or systems ([subject](http://www.uml-diagrams.org/use-case-subject.html)) should or can perform in collaboration with one or more **external users** of the system ([actors](http://www.uml-diagrams.org/use-case-actor.html)). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Note, that UML 2.0 to 2.4 specifications also described **use case diagram** as a specialization of a [class diagram](http://www.uml-diagrams.org/class-diagrams-overview.html), and class diagram is a [structure diagram](http://www.uml-diagrams.org/uml-25-diagrams.html#structure-diagram).

Use case diagrams are in fact twofold - they are both [behaviour diagrams](http://www.uml-diagrams.org/uml-25-diagrams.html#behavior-diagram), because they describe behaviour of the system, and they are also [structure diagrams](http://www.uml-diagrams.org/uml-25-diagrams.html#structure-diagram) - as a special case of class diagrams where classifiers are restricted to be either [actors](http://www.uml-diagrams.org/use-case-actor.html) or [use cases](http://www.uml-diagrams.org/use-case.html) related to each other with [associations](http://www.uml-diagrams.org/association.html).

 Moved use cases out of behaviourmodelling to UML **supplementary concepts**. So, it is an unfortunate quandary what kind of UML diagrams use case diagrams are.

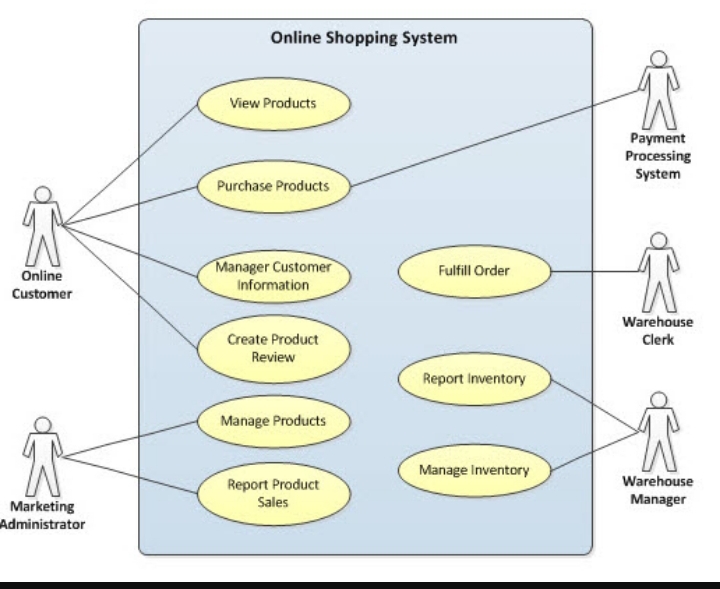


Figure 4.1.3.2 Use Case Diagram

**4.2Data Flow Diagram**

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an [information system](https://en.wikipedia.org/wiki/Information_system), modelling 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](https://en.wikipedia.org/wiki/Data_visualization) of [data *processing*](https://en.wikipedia.org/wiki/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 be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured [flowchart](https://en.wikipedia.org/wiki/Flowchart) which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

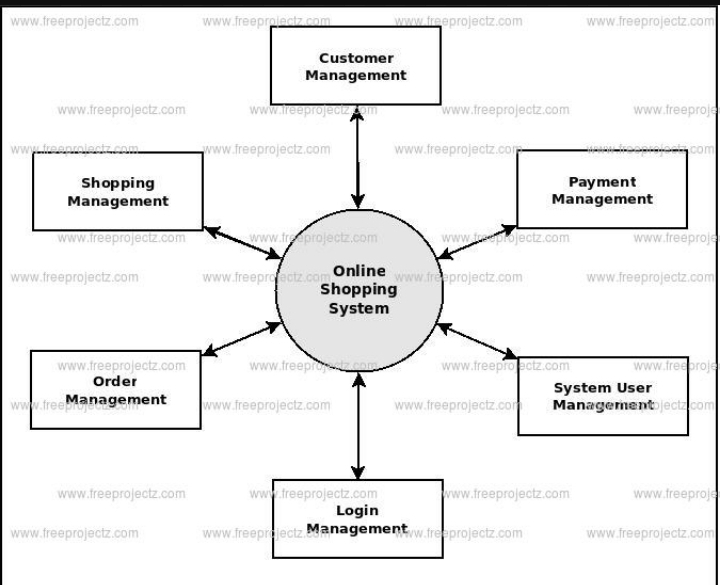


Figure 4.2.1.1 Data Flow Diagram

**4.3 E-R Diagram**

An entity-relationship diagram (ERD) is a graphical representation of an information system that shows the relationship between people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and can be used as the foundation for a relational database.

For example, an ER diagram representing the information system for a company's sales department might start with graphical representations of entities such as the sales representative, the customer, the customer's address, the customer's order, the product and the warehouse. (See diagram) Then lines or other symbols can be used to represent the relationship between entities, and text can be used to label the relationships.

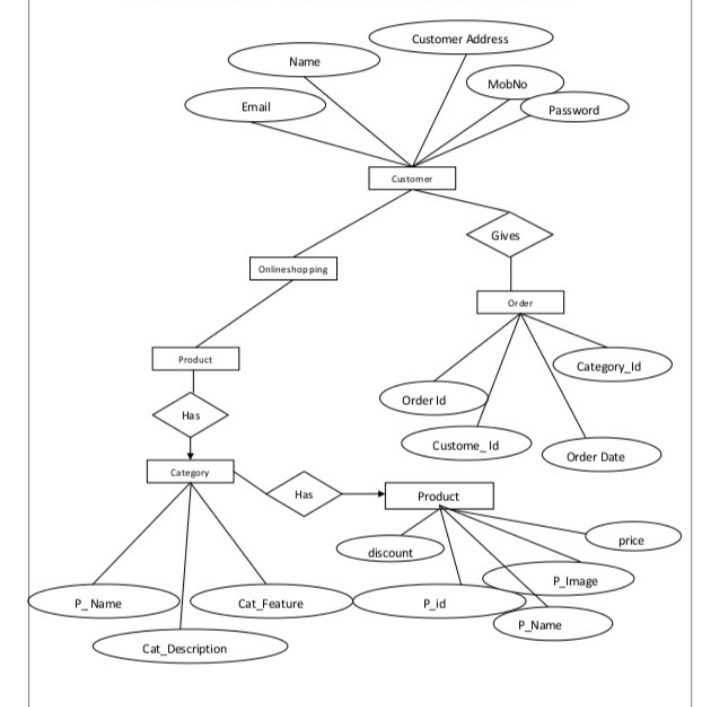
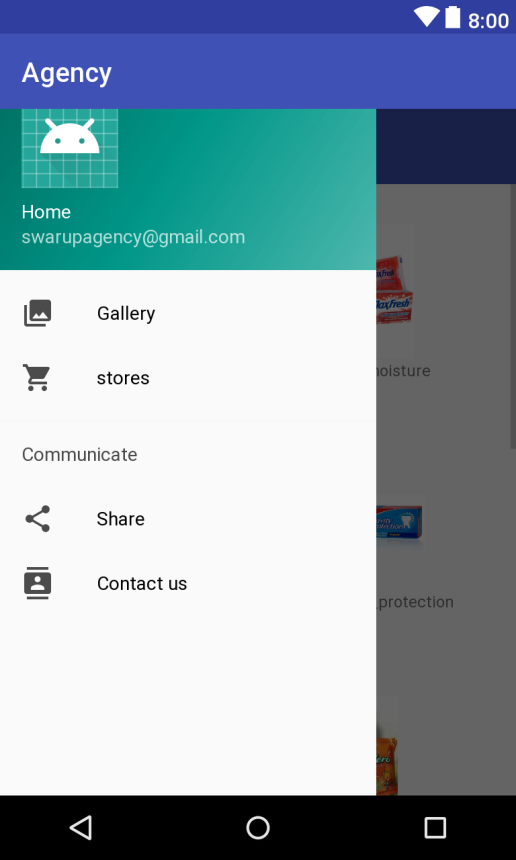
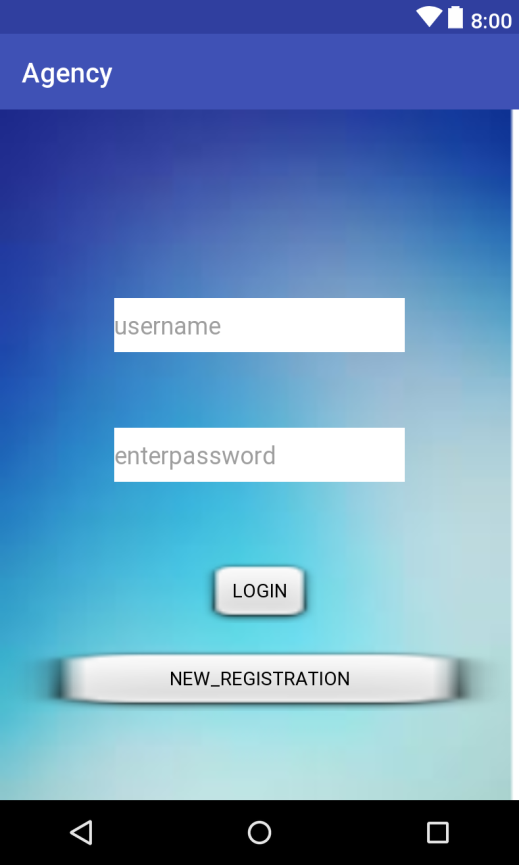


Figure 4.3.1 E-R Diagram

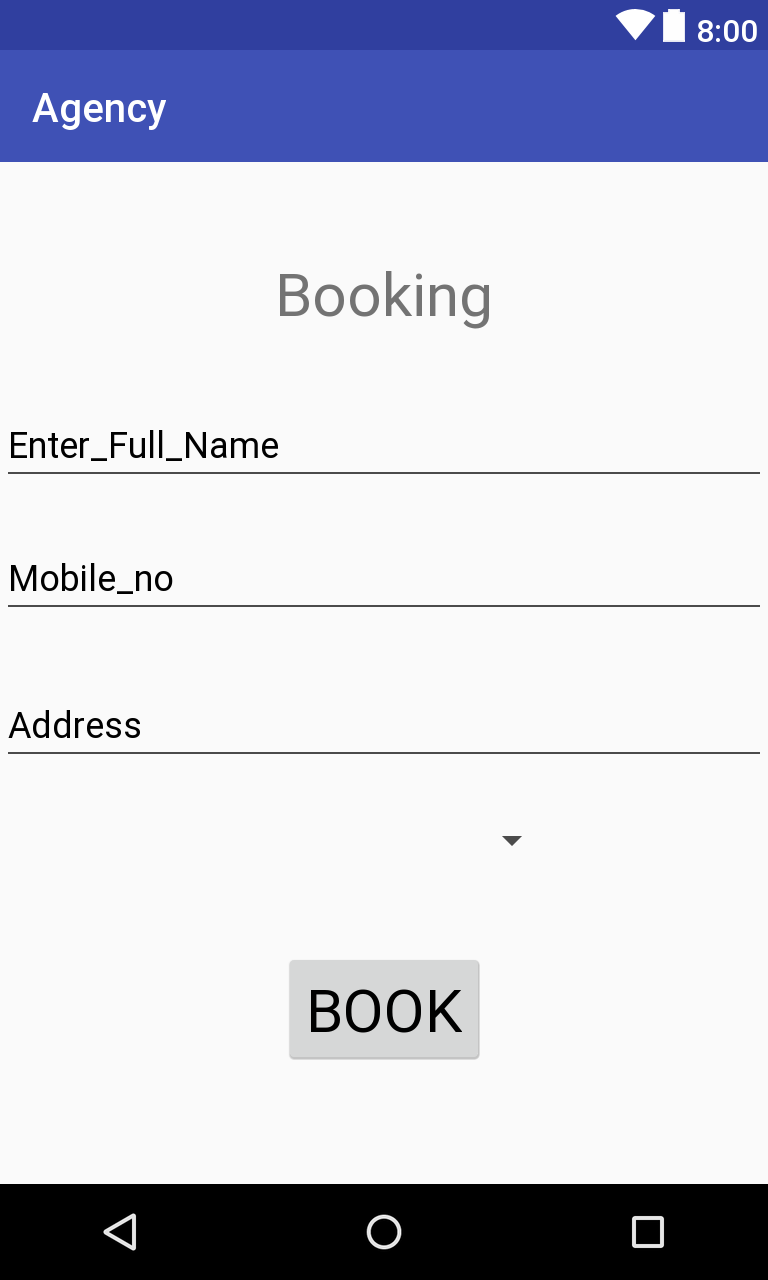
**SCREENSHOT**

****

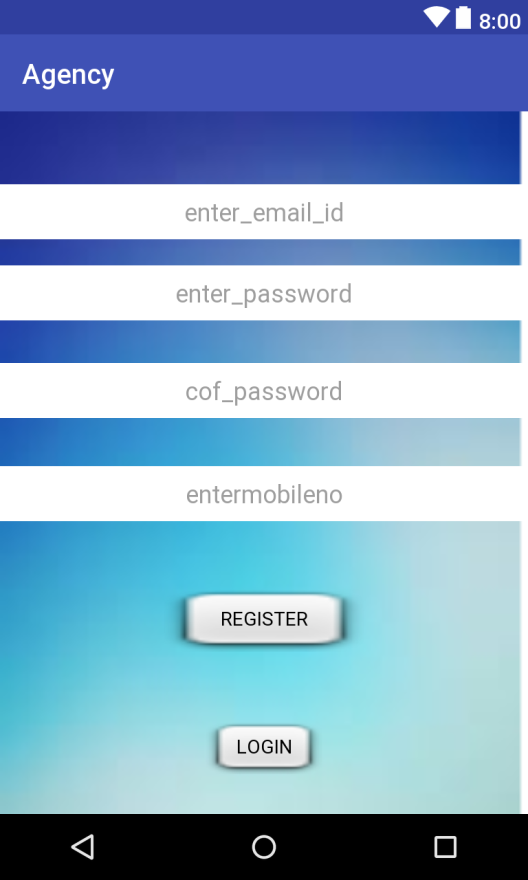
**Screenshot 1: Home page**

****

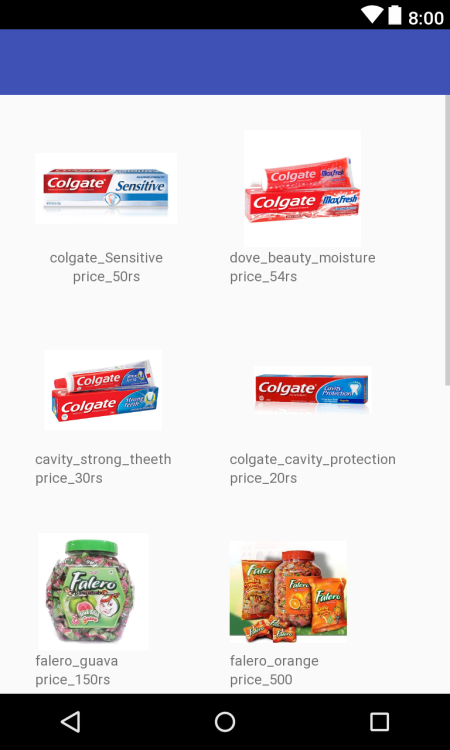
**Screenshot 2: Login page**

****

**Screenshot 3: Booking page**

****

**Screenshot 4: Register Page**

****

**Screenshot 5: Product Gallary**