**ABSTRACT**

This project provides an overview of an innovative online nursery plant platform that revolutionizes the way customers can order plants conveniently from the comfort of their homes. The platform harnesses the power of e-commerce to provide a user-friendly and accessible interface for individuals seeking to enhance their living spaces with a wide array of plants. By leveraging advanced technology and a comprehensive inventory, the online nursery plant platform offers customers a seamless ordering experience, enabling them to browse through an extensive selection of plants, access detailed information, and make well-informed decisions. The platform prioritizes customer satisfaction by ensuring high-quality plants sourced from reputable nurseries and employing efficient packaging and delivery processes to preserve the health and vitality of the plants during transit. Additionally, the platform offers a range of flexible payment options and secure transactions, providing customers with peace of mind while making their purchases. To enhance the customer experience, the platform may include interactive features such as personalized recommendations, plant care guides, and customer reviews, fostering a sense of community and knowledge sharing among plant enthusiasts. Overall, the online nursery plant platform redefines convenience and accessibility by enabling customers to effortlessly order and receive their desired plants, bringing the beauty and benefits of nature directly to their doorstep.Top of Form

**Keywords:** Plants, customers, orders, payments.

**1.INTRODUCTION**

**1.1 About The Project:**

The world of plants has long been revered for its beauty, serenity, and ability to transform living spaces into vibrant sanctuaries. However, the process of acquiring and nurturing plants has often been seen as cumbersome, requiring extensive research, visits to physical nurseries, and the logistical challenge of transporting delicate greenery. Fortunately, the advent of technology has revolutionized the way we connect with nature, giving rise to the emergence of online nursery plant platforms.

The objective of these platforms is clear: to provide a convenient and accessible online space for individuals to order a diverse range of plants and bring the beauty of nature into their homes with ease. These platforms leverage the power of e-commerce, advanced technology, and expert plant knowledge to streamline the entire plant purchasing process.In the past, the lack of a comprehensive inventory and detailed information about plants made it challenging for customers to confidently select the right plants for their specific needs and preferences. However, the online nursery plant platform seeks to address this problem by offering a vast selection of plants from reputable nurseries worldwide. Customers can browse through an extensive catalog that encompasses various species, sizes, and colors, ensuring that they find the perfect fit for their living spaces.

To empower customers in their decision-making process, the platform provides detailed plant descriptions, including care instructions, growth habits, and ideal environmental conditions. This wealth of information allows even those with limited gardening experience to make informed choices and successfully nurture their chosen plants. More over, the online nursery plant platform prioritizes customer satisfaction by ensuring the delivery of high-quality plants. The platform collaborates with responsible nurseries that prioritize ethical and environmentally conscious cultivation practices. Plants are carefully packaged using eco-friendly materials to maintain their health and vitality during transit. Additionally, efficient delivery processes and secure transactions ensure a seamless experience for customers.

Beyond the convenience of ordering plants online, these platforms foster a sense of community among plant enthusiasts. Interactive features such as forums, live chats, and virtual workshops allow individuals to connect, share experiences, and learn from one another. This not only enhances the customer experience but also provides a platform for knowledge sharing and growth within the plant-loving community.

**1.2. Project Description:**

To further enhance convenience, the platform offers various payment options, ensuring that customers can choose the method that best suits their needs. Secure transactions are a priority, providing customers with peace of mind when making purchases.

The platform goes beyond just facilitating transactions by providing additional interactive features. These may include personalized recommendations based on customer preferences, plant care guides to assist with plant maintenance, and customer reviews that foster a sense of community and knowledge sharing among plant enthusiasts.

In summary, this online nursery plant platform offers customers a convenient and accessible way to order a wide variety of plants, ensuring customer satisfaction through high-quality products and efficient delivery processes. It prioritizes security and offers interactive features that enhance the overall customer experience, ultimately bringing the beauty and benefits of nature directly to customers' doorsteps.

**1.3. Modules:**

In this application involved two characters to implement this process they are admin and Customer.

**Admin:** Admin is person to maintain the whole application.

**Operation-Login:** Admin will login into the application by entering the valid details like (email and password).

**Operation-Add/Edit/Delete Customer:** Add, Update & Delete

**Add Customer:** Admin will add the customer with their details like(Name, Email, Password, Mobile and address). After submitting the details system will send login details to customer through mail.

**Update Customer Details:** If any modification occurs, Admin have the right to update the customer profile. That time system will send automatic mail to customer regarding the changes and login details.

**Operation - Add Plants:** Admin will add plants with details like(name, rice, address, image and description).

**Operation - View Plants:** After adding plants admin can view list of added plants.

**Operation – View Ordered Plants:** Admin can view the plants which is ordered by the user.

**Operation – Add Shipping details:** Admin will add the ordered delivery expected date and shipping date. So that user can track their order.

**Operation - Sign out:** After completion of work admin can logout from the application.

**Customer:** Here customer role is to find and order their wished plants.

**Operation - Register:** Customer will register with their details like (name, email,

password, mobile number and address).

**Login:** Customer must login with valid registered email and password.

**View Plants:** Customer can view all the available plants those are added by admin.

**Operation – Add to cart:** Customer can add their selected plant to cart once after gone through all available plants.

**Operation – Payment & Place order:** Customer can place their order by making payment if they want to purchase.

**Operation** ­**-Track Order:** After admin adds shipping details customer can track their order with the help of ship details.

**Operation** - **Logout:** He/she can log out of the application once the work is completed.

**2.SYSTEM ANALYSIS**

**2.1.DOMAIN ANALYSIS:**

The purpose of the domain analysis is to gain a comprehensive understanding of the key elements and functionalities required for developing an online nursery plant platform using Django, a popular web framework in Python. This analysis will identify the main entities, relationships, and features specific to the online nursery plant domain and guide the design and implementation of the Django project.

**2.2.REQUIREMENT ANALYSIS:**

**Functional and non-functional requirements:**

Requirement’s analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

**2.2.1. Functional Requirements**:

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples of functional requirements:

1. Authentication of user whenever he/she logs into the system
2. System shutdown in case of a cyber-attack
3. A verification email is sent to user whenever he/she register for the first time on some software system.

**2.2.2. Non-functional requirements**: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are

implemented varies from one project to other. They are also called non-behavioral requirements.  
They basically deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

Examples of non-functional requirements:

1. Emails should be sent with a latency of no greater than 12 hours from such an activity.
2. The processing of each request should be done within 10 seconds
3. The site should load in 3 seconds whenever of simultaneous users are > 10000

**2.2.3. User Interface:**

The User Interface is a GUI developed using HTML, CSS, JS.

**2.2.4. Software Interface**

The main processing is done in Python and console applications.

**2.2.5. Manpower Requirements**

Members can complete the project in 2-3 months if they work fulltime on it.

**2.3. EXISTING SYSTEM**

The existing method for purchasing plants involves visiting physical nurseries or garden centers, which may have a limited selection of plants. Customers need to allocate time for transportation, rely on the expertise of nursery staff, and conduct independent research for plant information. Additionally, the traditional method may not offer convenient payment options or reliable delivery services, requiring customers to arrange transportation themselves. These limitations call for a more convenient and comprehensive approach, leading to the rise of online nursery plant platforms.

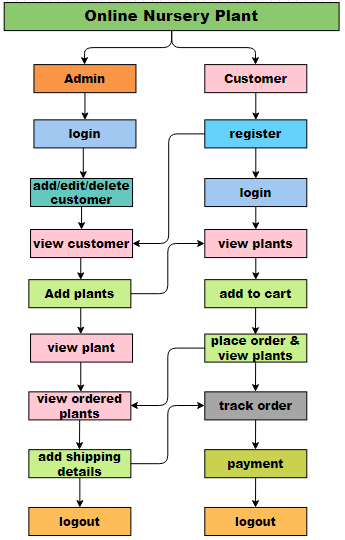
**2.3.1.** **Disadvantages**

* **Not User Friendly:** The existing system is not user friendly because the retrieval of data is very slow and data is not maintained efficiently.
* **Limited Selection:** Physical nurseries often have a limited selection of plants, depending on the region and season. This can restrict customers' choices and make it challenging to find specific plant species or cultivars they desire for their gardens.
* **Inconvenience and Time-Consuming:** Visiting physical nurseries requires customers to invest time and effort in transportation, navigating through rows of plants, and seeking assistance from nursery staff. This process can be time-consuming, especially for individuals with busy schedules or limited access to transportation.
* **Lack of Comprehensive Information:** Customers may encounter challenges in obtaining comprehensive plant information and guidance from nursery staff. The expertise and availability of staff members can vary, leaving customers to rely on their own knowledge or conduct independent research to gather essential details about plant care, compatibility, and other relevant factors.
* **Transportation and Delivery Issues:** The traditional method of purchasing plants may not offer convenient payment options or reliable delivery services. Customers may need to arrange transportation for their chosen plants themselves, risking potential damage during transit. This can be particularly challenging for individuals without access to suitable vehicles or for those purchasing plants from distant nurseries.

**2.4. PROPOSED SYSTEM**

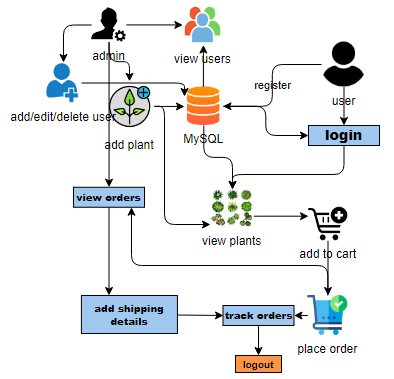
The proposed method involves the development of an online nursery plant platform that offers a comprehensive and convenient plant purchasing experience. The platform will feature an extensive catalog of plants from reputable nurseries, providing customers with a wide selection to choose from. Detailed plant information, including care instructions, will be provided to assist customers in making informed decisions. The platform will offer secure and convenient ordering processes. Reliable delivery services will be employed to ensure the safe and efficient transportation of plants to customers' doorsteps.

**2.4.1. Work Flow of Proposed system**

****

**Fig 2.4.1 :-Work Flow of Proposed System**

**2.4.2. ARCHITECTURE**

****

**Fig 2.4.2 :-Architecture**

**2.4.3. Advantages**

**1.** **User Friendly:** - The proposed system is user friendly because the retrieval and storing of data is fast and data is maintained efficiently. More over the graphical user interface is provided in the proposed system, which provides user to deal with the system very easily.

**2. Wide Variety of Plants:** The proposed online nursery plant platform offers a diverse and extensive selection of plants from reputable nurseries worldwide. Customers have access to a wide variety of plant species, cultivars, and sizes, allowing them to find the perfect plants to suit their preferences, environmental conditions, and aesthetic vision for their living spaces.

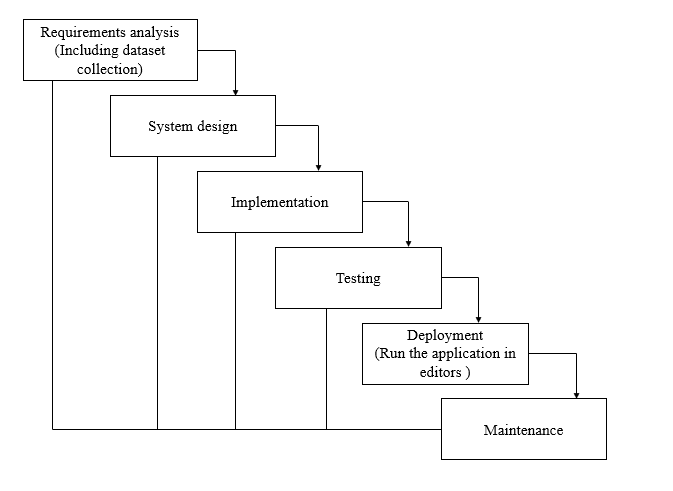
**3. Convenient and Time-Saving:** With the online platform, customers can browse and order plants at their convenience, eliminating the need for physical visits to nurseries. This saves time and effort, particularly for busy individuals who may have limited availability to explore physical nurseries.

**4. Detailed Plant Information:** The platform provides detailed plant descriptions, care instructions, and growth habits for each plant. Customers can access comprehensive information about plant care, ensuring they have the necessary knowledge to nurture and maintain their plants successfully.

**5. Reliable Delivery:** The proposed method ensures efficient delivery services. The platform also partners with reliable delivery services, ensuring that plants are carefully packaged and delivered in a timely manner, preserving their health and vitality during transit.

**2.4.4. SOFTWARE DEVELOPMENT LIFE CYCLE(SDLC):**

In our project we use waterfall model as our software development cycle because of its step-by-step procedure while implementing.



**Fig 2.4.4:- Waterfall Model**

**Requirement Gathering and analysis** − all possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

* **System Design** − the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
* **Implementation** − with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
* **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

**3.FEASIBILITY STUDY**

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* Economical feasibility
* Technical feasibility
* Social feasibility

**3.1. Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### **3.2. Technical Feasibility**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**3.3. Social Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**4. SYSTEM REQUIREMENT**

**4.1. Hardware Requirements**

# Processor - I3/Intel Processor

Hard Disk - 160GB

RAM - 8GB

* 1. **Software Requirements:**

Operating System : Windows 7/8/10

Server side Script : Java Script

Programming Language : Python

Libraries : Node.JS

IDE/Workbench : Visual Studio Code

Technology : Python 3.6+

Server Deployment : SQL Server

**5. SYSTEM DESIGN**

**5.1.UML DIAGRAMS INTRODUCTION**

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**5.2. SYSTEM DESIGN ASPECTS**

**5.2.1 Design of Inputs:**

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties −

* It should serve specific purpose effectively such as storing, recording, and retrieving the information.
* It ensures proper completion with accuracy.
* It should be easy to fill and straightforward.
* It should focus on user’s attention, consistency, and simplicity.
* All these objectives are obtained using the knowledge of basic design principles regarding −
  + What are the inputs needed for the system?
  + How end users respond to different elements of forms and screens.

### **5.2.2. Objectives for Design of Input:**

The objectives of input design are −

* To design data entry and input procedures
* To reduce input volume
* To design source documents for data capture or devise other data capture methods
* To design input data records, data entry screens, user interface screens, etc.
* To use validation checks and develop effective input controls.

**5.2.3. Design of Output:**

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

### Objectives of Output Design:

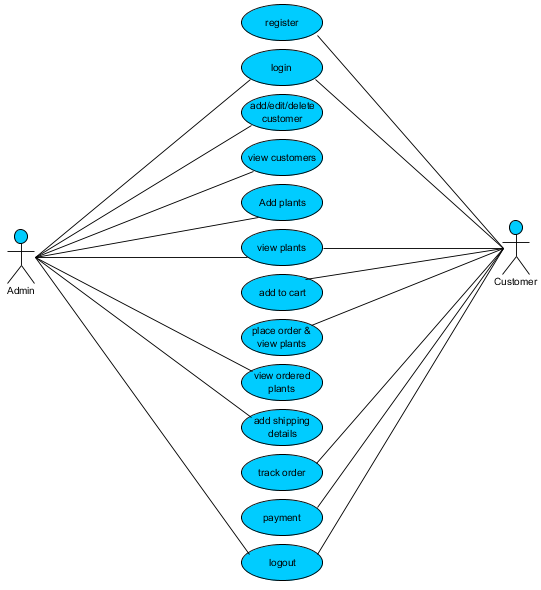
The objectives of input design are:

* To develop output design that serves the intended purpose and eliminates the production of unwanted output.
* To develop the output design that meets the end user’s requirements.
* To deliver the appropriate quantity of output.
* To form the output in appropriate format and direct it to the right person.
* To make the output available on time for making good decisions.

**5.3. UML DIAGRAMS:**

**5.3.1. Use Case Diagram:**

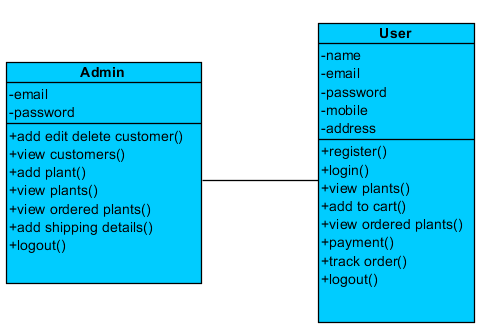
* A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis.
* Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.
* The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

****

**Fig 5.3.1 :-** Use-Case Diagram

**5.3.2 Class Diagram:**

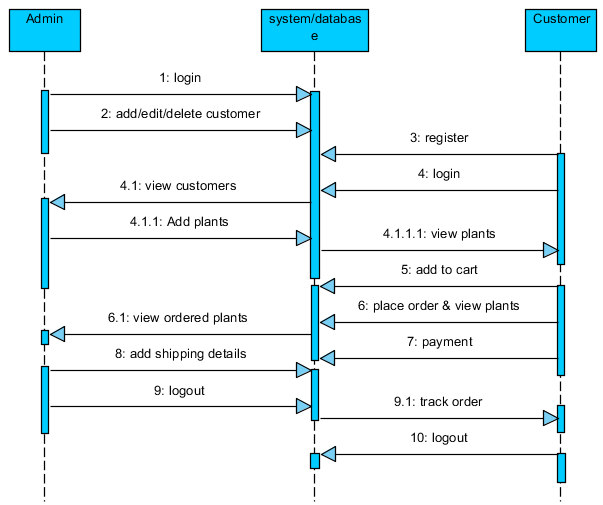
In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes,their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



**Fig 5.3.2 :-** Class Diagram

**5.3.3. Sequence Diagram:**

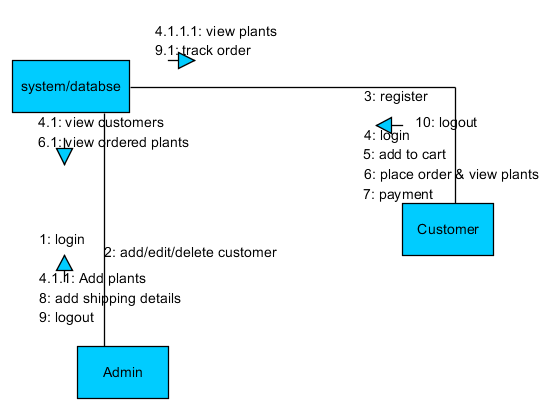
A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



**Fig 5.3.3 :-** Sequence Diagram

**5.3.4. Collaboration Diagram:**

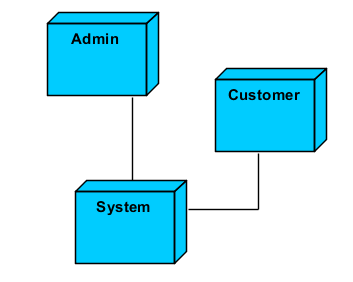
In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.



**Fig 5.3.4 :-** Collaboration Diagram

**5.3.5. Deployment Diagram**

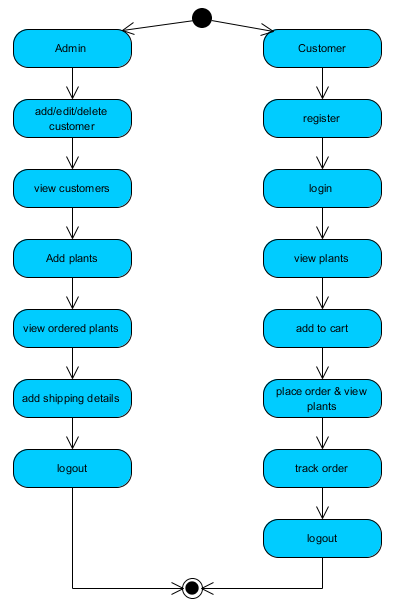
Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware’s used to deploy the application.

****

**Fig 5.3.5 :-** Deployment Diagram

**5.3.6. Activity Diagram:**

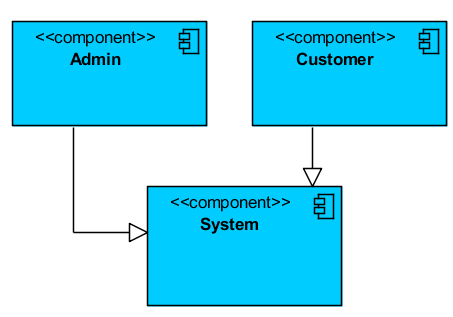
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



**Fig 5.3.6 :-** Activity Diagram

**5.3.7. Component Diagram**:

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

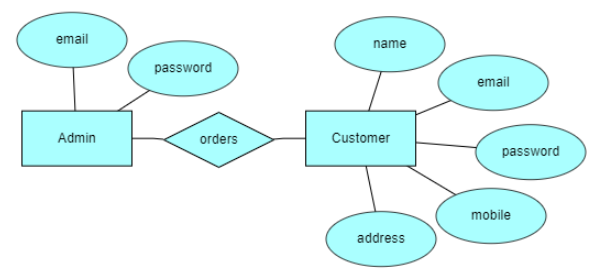


**Fig 5.3.7 :-** Component Diagram

**5.3.8. ER Diagram:**

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let’s have a look at a simple ER diagram to understand this concept.

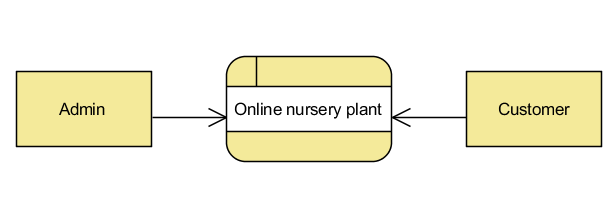
****

**Fig 5.3.8 :-**ER Diagram

**5.3.9. DFD Diagram:**

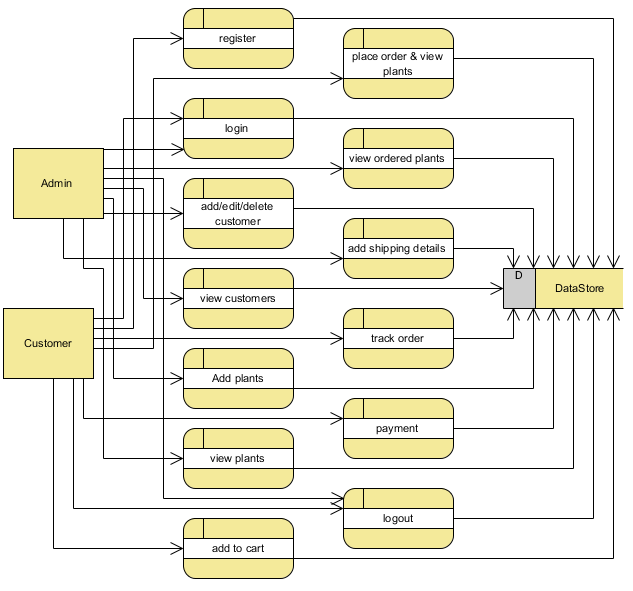
A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

**Context Level Diagram:**

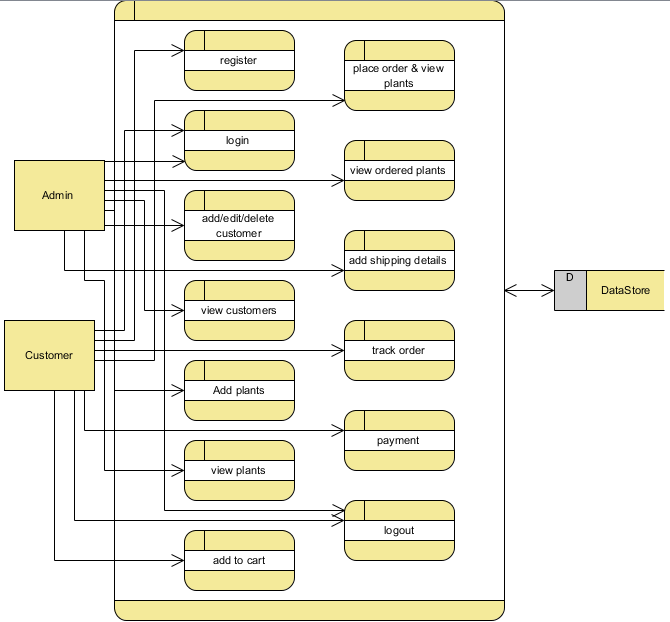
****

**Fig 5.3.9 :-** DFD Diagram

**Level 1 Diagram**

****

**Level 2 Diagram:**

****

**6.OVERVIEW OF TECHNOLOGIES**

**6.1. PYTHON TECHNOLOGY**

Python

### What Is a Script?

Up to this point, I have concentrated on the interactive programming capability of Python.  This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode

**Scripts are reusable**

Basically, a script is a text file containing the statements that comprise a Python program.  Once you have created the script, you can execute it over and over without having to retype it each time.

**Scripts are editable**

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor.  Then you can execute each of the individual versions.  In this way, it is easy to create different programs with a minimum amount of typing.

**You will need a text editor**

Just about any text editor will suffice for creating Python script files.

You can use Microsoft Notepad, Microsoft WordPad, Microsoft Word,or just about any word processor if you want to.

**Difference between a script and a program**

**Script:**

Scripts are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or byte code, whereas the applications they control are traditionally compiled to native machine code.

**Program:**

The program has an executable form that the computer can use directly to execute the instructions.

The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled)

**Python**

What is Python? Chances you are asking yourself this. You may have found this book because you want to learn to program but don’t know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to “big name” languages. Hopefully I can explain it for you.

**Python concepts**

If you’re not interested in the how’s and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it’s a great one to start programming with.

• Open-source general-purpose language.

• Object Oriented, Procedural, Functional

• Easy to interface with C/ObjC/Java/Fortran

• Easy-is to interface with C++ (via SWIG)

• Great interactive environment

• Great interactive environment

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
* Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

**History of Python**

Python was developed by Guido van Possum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Possum still holds a vital role in directing its progress.

**Python Features**

Python's features include –

* Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* Easy-to-read − Python code is more clearly defined and visible to the eyes.
* Easy-to-maintain − Python's source code is fairly easy-to-maintained.
* A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* Extendable − you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
* Databases − Python provides interfaces to all major commercial databases.
* GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
* Scalable − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte-code for building large applications.
* It provides very high-level dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

**Dynamic vs Static**

Types Python is a dynamic-typed language. Many other languages are static typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of “thing” each data value is.

For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a “float” type.

This tells the compiler that the only data that can be used for that variable must be a floating-point number, i.e., a number with a decimal point.

If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program.

Python, however, doesn’t require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating-point number) you need in your program.

With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating-point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment).

If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double.

With Python, it doesn’t matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values. For example, say you are dividing two numbers. One is a floating-point number and one is an integer. Python realizes that it’s more accurate to keep track of decimals so it automatically calculates the result as a floating-point number

**Variables**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

**Standard Data Types**

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

* Numbers
* String
* List
* Tuple
* Dictionary

## **Python Numbers**

Number data types store numeric values. Number objects are created when you assign a value to them

## **Python Strings**

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

## **Python Lists**

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.

## **Python Tuples**

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

## **Python Dictionary**

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

**Different modes in python**

Python has two basic modes: normal and interactive.

The normal mode is the mode where the scripted and finished .pie files are run in the Python interpreter.

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole

**20 Python libraries**

1. Requests. The most famous http library written by Kenneth remits. It’s a must have for every python developer.

2. Scrappy. If you are involved in web scraping then this is a must have library for you. After using this library, you won’t use any other.

3. Python. A guy toolkit for python. I have primarily used it in place of tinder. You will really love it.

4. Pillow. A friendly fork of PIL (Python Imaging Library). It is more user friendly than PIL and is a must have for anyone who works with images.

5. SQL Alchemy. A database library. Many love it and many hate it. The choice is yours.

6. Beautiful Soup. I know it’s slow but this xml and html parsing library is very useful for beginners.

7. Twisted. The most important tool for any network application developer. It has a very beautiful ape and is used by a lot of famous python developers.

8. Numbly. How can we leave this very important library? It provides some advance math functionalities to python.

9. Skippy. When we talk about numbly then we have to talk about spicy. It is a library of algorithms and mathematical tools for python and has caused many scientists to switch from ruby to python.

10. Matplotlib. A numerical plotting library. It is very useful for any data scientist or any data analyzer.

11. Pygmy. Which developer does not like to play games and develop them? This library will help you achieve your goal of 2d game development.

12. Piglet. A 3d animation and game creation engine. This is the engine in which the famous [python port](https://github.com/fogleman/Minecraft) of mine craft was made

13. Pit. A GUI toolkit for python. It is my second choice after python for developing GUIs for my python scripts.

14. Pit. Another python GUI library. It is the same library in which the famous Bit torrent client is created.

15. Scaly. A packet sniffer and analyzer for python made in python.

16. Pywin32. A python library which provides some useful methods and classes for interacting with windows.

17. Notch. Natural Language Toolkit – I realize most people won’t be using this one, but it’s generic enough. It is a very useful library if you want to manipulate strings. But its capacity is beyond that. Do check it out.

18. Nose. A testing framework for python. It is used by millions of python developers. It is a must have if you do test driven development.

19. Simply. Simply can-do algebraic evaluation, differentiation, expansion, complex numbers, etc. It is contained in a pure Python distribution.

20. I Python. I just can’t stress enough how useful this tool is. It is a python prompt on steroids. It has completion, history, shell capabilities, and a lot more. Make sure that you take a look at it.

**NumPy**

Humpy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In numbly dimensions are called axes. The number of axes is rank.

• Offers Matlab-ish capabilities within Python

• Fast array operations

• 2D arrays, multi-D arrays, linear algebra etc.

**Matplotlib**

• High quality plotting library.

**Python class and objects**

These are the building blocks of OOP. Class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g., a chair. Each class has individual characteristics unique to that class, including variables and methods. Classes are very powerful and currently “the big thing” in most programming languages. Hence, there are several chapters dedicated to OOP later in the book.

The class is the most basic component of object-oriented programming. Previously, you learned how to use functions to make your program do something.

Now will move into the big, scary world of Object-Oriented Programming (OOP). To be honest, it took me several months to get a handle on objects.

When I first learned C and C++, I did great; functions just made sense for me.

Having messed around with BASIC in the early ’90s, I realized functions were just like subroutines so there wasn’t much new to learn.

However, when my C++ course started talking about objects, classes, and all the new features of OOP, my grades definitely suffered.

Once you learn OOP, you’ll realize that it’s actually a pretty powerful tool. Plus many Python libraries and APIs use classes, so you should at least be able to understand what the code is doing.

One thing to note about Python and OOP: it’s not mandatory to use objects in your code in a way that works best; maybe you don’t need to have a full-blown class with initialization code and methods to just return a calculation. With Python, you can get as technical as you want.

As you’ve already seen, Python can do just fine with functions. Unlike languages such as Java, you aren’t tied down to a single way of doing things; you can mix functions and classes as necessary in the same program. This lets you build the code

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Here’s a brief list of Python OOP ideas:

• The class statement creates a class object and gives it a name. This creates a new namespace.

• Assignments within the class create class attributes. These attributes are accessed by qualifying the name using dot syntax: **ClassName.Attribute.**

• Class attributes export the state of an object and its associated behavior. These attributes are shared by all instances of a class.

• Calling a class (just like a function) creates a new instance of the class.

This is where the multiple copy’s part comes in.

• Each instance gets ("inherits") the default class attributes and gets its own namespace. This prevents instance objects from overlapping and confusing the program.

• Using the term self identifies a particular instance, allowing for per-instance attributes. This allows items such as variables to be associated with a particular instance.

**Inheritance**

First off, classes allow you to modify a program without really making changes to it.

To elaborate, by sub classing a class, you can change the behaviour of the program by simply adding new components to it rather than rewriting the existing components.

As we’ve seen, an instance of a class inherits the attributes of that class.

However, classes can also inherit attributes from other classes. Hence, a subclass inherits from a superclass allowing you to make a generic superclass that is specialized via subclasses.

The subclasses can override the logic in a superclass, allowing you to change the behavior of your classes without changing the superclass at all.

Operator Overloads

Operator overloading simply means that objects that you create from classes can respond to actions (operations) that are already defined within Python, such as addition, slicing, printing, etc.

Even though these actions can be implemented via class methods, using overloading ties the behavior closer to Python’s object model and the object interfaces are more consistent to Python’s built-in objects, hence overloading is easier to learn and use.

User-made classes can override nearly all of Python’s built-in operation methods

**Exceptions**

I’ve talked about exceptions before but now I will talk about them in depth. Essentially, exceptions are events that modify program’s flow, either intentionally or due to errors.

They are special events that can occur due to an error, e.g., trying to open a file that doesn’t exist, or when the program reaches a marker, such as the completion of a loop.

Exceptions, by definition, don’t occur very often; hence, they are the "exception to the rule" and a special class has been created for them. Exceptions are everywhere in Python.

Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

Here are just a few examples:

• Accessing a non−existent dictionary key will raise a Key Error exception.

• Searching a list for a non−existent value will raise a Value Error exception

. • Calling a non−existent method will raise an Attribute Error exception.

• Referencing a non−existent variable will raise a Name Error exception.

• Mixing data types without coercion will raise a Type Error exception.

One use of exceptions is to catch a fault and allow the program to continue working; we have seen this before when we talked about files.

This is the most common way to use exceptions. When programming with the Python command line interpreter, you don’t need to worry about catching exceptions.

Your program is usually short enough to not be hurt too much if an exception occurs.

Plus, having the exception occur at the command line is a quick and easy way to tell if your code logic has a problem.

However, if the same error occurred in your real program, it will fail and stop working. Exceptions can be created manually in the code by raising an exception.

It operates exactly as a system-caused exceptions, except that the programmer is doing it on purpose. This can be for a number of reasons. One of the benefits of using exceptions is that, by their nature, they don’t put any overhead on the code processing.

Because exceptions aren’t supposed to happen very often, they aren’t processed until they occur.

Exceptions can be thought of as a special form of the if/elf statements. You can realistically do the same thing with if blocks as you can with exceptions.

However, as already mentioned, exceptions aren’t processed until they occur; if blocks are processed all the time.

Proper use of exceptions can help the performance of your program.

The more infrequent the error might occur, the better off you are to use exceptions; using if blocks require Python to always test extra conditions before continuing.

Exceptions also make code management easier: if your programming logic is mixed in with error-handling if statements, it can be difficult to read, modify, and debug your program.

User-Defined Exceptions

I won’t spend too much time talking about this, but Python does allow for a programmer to create his own exceptions.

You probably won’t have to do this very often but it’s nice to have the option when necessary.

However, before making your own exceptions, make sure there isn’t one of the built-in exceptions that will work for you.

They have been "tested by fire" over the years and not only work effectively, they have been optimized for performance and are bug-free.

Making your own exceptions involves object-oriented programming, which will be covered in the next chapter.

To make a custom exception, the programmer determines which base exception to use as the class to inherit from, e.g., making an exception for negative numbers or one for imaginary numbers would probably fall under the Arithmetic Error exception class.

To make a custom exception, simply inherit the base exception and define what it will do.

**Python modules**

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

**Testing code**

As indicated above, code is usually developed in a file using an editor.

To test the code, import it into a Python session and try to run it.

Usually there is an error, so you go back to the file, make a correction, and test again.

This process is repeated until you are satisfied that the code works. T

His entire process is known as the development cycle.

There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

**Functions in Python**

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carved out, the function can or cannot return one or more values.

There are three types of functions in python:

Help (), min (), print ().

Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

* global names of a module
* local names in a function or method invocation
* built-in names: this namespace contains built-in functions (e.g., abs (), camp (), ...) and built-in exception names

**Garbage Collection:** Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

**Python XML Parser**

XML is a portable, open-source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

What is XML? The Extensible Markup Language XML is a markup language much like HTML or SGML.

This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

XML Parser Architectures and APIs the Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

Simple API for XML SAX: Here, you register call-backs for events of interest and then let the parser proceed through the document.

This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.

Document Object Model DOM API: This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical tree − based form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.

**Python Web Frameworks**

A web framework is a code library that makes a developer's life easier when building reliable, scalable and maintainable web applications.

Why are web frameworks useful?

Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application.

Common web framework functionality

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

URL routing

HTML, XML, JSON, and other output format tinplating

Database manipulation

Security against Cross-site request forgery (CSRF) and other attacks

Session storage and retrieval

Not all web frameworks include code for all of the above functionality. Frameworks fall on the spectrum from executing a single use case to providing every known web framework feature to every developer. Some frameworks take the "batteries-included" approach where everything possibly comes bundled with the framework while others have a minimal core package that is amenable to extensions provided by other packages.

## Comparing web frameworks

There is also a repository called [compare-python-web-frameworks](https://github.com/mattmakai/compare-python-web-frameworks) where the same web application is being coded with varying Python web frameworks, tinplating engines and object.

## Web framework resources

1. When you are learning how to use one or more web frameworks it's helpful to have an idea of what the code under the covers is doing.
2. Frameworks is a really well-done short video that explains how to choose between web frameworks. The author has some particular opinions about what should be in a framework. For the most part I agree although I've found sessions and database ORMs to be a helpful part of a framework when done well.
3. What is a web framework? Is an in-depth explanation of what web frameworks are and their relation to web servers?
4. Jingo vs. Flash vs. Pyramid: Choosing a Python web framework contains background information and code comparisons for similar web applications built in these three big Python frameworks.
5. This fascinating blog post takes a look at the code complexity of several Python web frameworks by providing visualizations based on their code bases.
6. Python’s web frameworks benchmarks are a test of the responsiveness of a framework with encoding an object to JSON and returning it as a response as well as retrieving data from the database and rendering it in a template. There were no conclusive results but the output is fun to read about nonetheless.
7. What web frameworks do you use and why are they awesome? Is a language agnostic Reedit discussion on web frameworks? It's interesting to see what programmers in other languages like and dislike about their suite of web frameworks compared to the main Python frameworks.
8. This user-voted question & answer site asked "What are the best general purpose Python web frameworks usable in production?” The votes aren't as important as the list of the many frameworks that are available to Python developers.
9. Web frameworks learning checklist
10. Choose a major Python web framework (Jingo or Flask are recommended) and stick with it. When you're just starting it's best to learn one framework first instead of bouncing around trying to understand every framework.

Work through a detailed tutorial found within the resource’s links on the framework's page.

Study open-source examples built with your framework of choice so you can take parts of those projects and reuse the code in your application.

Build the first simple iteration of your web application then go to the [deployment](https://www.fullstackpython.com/deployment.html) section to make it accessible on the web.

**6.2. ABOUT PYTHON PLATFORM**

**CPython:**

CPython is the reference implementation of Python and the most widely used version.t is written in C and provides the official Python interpreter.

CPython offers excellent compatibility with the Python language specification and a vast ecosystem of third-party libraries.

It has good integration with C/C++ code and allows developers to write Python extension modules using the C API.

**Jython:**

Jython is an implementation of Python that runs on the Java Virtual Machine (JVM).

It allows seamless integration between Python and Java code, enabling Python programs to utilize Java libraries.

Jython supports the Python language syntax and standard library, but not all Python extensions are compatible due to differences in implementation.

It is commonly used for scripting in Java applications or for leveraging existing Java codebases.

**IronPython:**

IronPython is a Python implementation targeting the .NET Framework, developed by Microsoft.

It provides interoperability with the .NET ecosystem, allowing Python code to interact with .NET libraries and frameworks.

IronPython can be used for building .NET applications or scripting within the .NET environment.

It supports both Python 2 and Python 3 syntax, although Python 3 support is still a work in progress.

**PyPy:**

PyPy is an alternative implementation of Python that focuses on speed and efficiency.

It utilizes a Just-in-Time (JIT) compiler, which can provide significant performance improvements over CPython in certain scenarios.

PyPy aims to be compatible with the Python language and supports most Python libraries, although some CPython extensions may not work directly.

It is particularly useful for performance-critical applications or when running Python code on resource-constrained environments.

**MicroPython:**

MicroPython is a minimalistic implementation of Python designed for microcontrollers and embedded systems.

It provides a subset of the Python language and standard library optimized for small memory footprints and low-power devices.

MicroPython offers a compact runtime and can run on micro-controllers with limited resources, making it suitable for Internet of Things (IoT) projects.

**6.3. ABOUT HYPER TEXT MARKUP LANGUAGE (HTML)**

HTML (Hypertext Markup Language) is the standard markup language for creating web pages and applications. It provides the structure and content of a web page, defining the elements and their arrangement on the page.

Key aspects of HTML:

Structure: HTML uses tags to define the structure and content of a web page. Tags are enclosed in angle brackets (<>) and usually come in pairs: an opening tag and a closing tag. For example, <p> denotes a paragraph element, and </p> marks the end of the paragraph.

Elements: HTML elements represent different components of a web page, such as headings, paragraphs, images, links, forms, tables, and more. Each element has a specific purpose and semantic meaning. Elements can be nested inside each other to create a hierarchical structure.

Attributes: HTML elements can have attributes, which provide additional information about the element. Attributes are specified within the opening tag and consist of a name and a value. For example, the <img> element can have attributes like src (source) and alt (alternative text).

Text and Media: HTML supports various content types, including text, images, audio, video, and embedded content. Text can be formatted using headings (<h1> to <h6>), paragraphs (<p>), lists (<ul>, <ol>, <li>), and other formatting elements.

Hyperlinks: HTML enables the creation of hyperlinks using the <a> (anchor) element. Links are specified using the href attribute, which defines the destination URL. Clicking on a link navigates the user to the linked page or resource.

Forms: HTML provides form elements (<form>, <input>, <select>, <textarea>, etc.) for capturing user input and submitting data to a server. Forms can include text fields, checkboxes, radio buttons, dropdown menus, and more.

Semantics: HTML includes semantic elements (<header>, <nav>, <section>, <footer>, etc.) that convey the meaning and structure of the content. Semantic elements enhance accessibility, search engine optimization, and overall understanding of the page's structure.

Cascading Style Sheets (CSS): Although HTML defines the structure and content of a web page, CSS is used for styling and presentation. CSS allows developers to control the appearance of HTML elements, including fonts, colors, layouts, and more.

**6.4. ABOUT CSS**

CSS (Cascading Style Sheets) is a style sheet language used to describe the presentation and visual styling of HTML (or XML) documents. It provides a way to control the layout, colors, fonts, and other visual aspects of a web page.

Key aspects of CSS:

Selectors: CSS selectors are used to target specific HTML elements or groups of elements to apply styles. Selectors can be based on element types (<h1>, <p>, etc.), classes (.class-name), IDs (#element-id), attributes, and more.

Properties and Values: CSS properties define specific aspects of an element's appearance, such as color, font, margin, padding, width, height, and background. Each property can have one or more values, determining how the property should be applied.

Style Rules: CSS rules consist of one or more selectors and a set of property-value pairs enclosed in curly braces. Style rules define how elements matching the selectors should be styled.

Cascading and Specificity: CSS follows a cascading model, where multiple styles can be applied to an element, and conflicts are resolved based on specificity and order of appearance. Specificity determines which style rules take precedence over others.

Inheritance: CSS properties can be inherited by child elements from their parent elements. Inheritance allows for the propagation of styles throughout the document, reducing the need for redundant styling.

Box Model: CSS treats each HTML element as a rectangular box with properties like width, height, margin, padding, and border. The box model determines how these properties affect the element's size and spacing.

Media Queries: CSS supports media queries, which allow developers to apply different styles based on the characteristics of the device or viewport, such as screen size, resolution, or orientation. Media queries enable responsive design and adaptability across various devices.

External Style Sheets: CSS can be defined in an external CSS file, separate from the HTML document. This promotes separation of concerns and reusability, as multiple HTML pages can reference the same CSS file.

**6.5. ABOUT JAVASCRIPT**

JavaScript is a popular programming language primarily used for front-end web development. It allows developers to add interactivity, behavior, and dynamic features to web pages. JavaScript is supported by all major web browsers, making it a versatile language for client-side scripting.

**Key aspects of JavaScript:**

Client-side Scripting: JavaScript is primarily used as a client-side scripting language, meaning it runs on the user's browser. It can manipulate and modify the HTML and CSS of a web page in real-time, enabling dynamic and interactive functionality.

Object-Oriented and Event-Driven: JavaScript is an object-oriented language, supporting the creation and manipulation of objects and their properties and methods. It also follows an event-driven programming model, allowing developers to respond to user actions and trigger appropriate actions.

DOM Manipulation: The Document Object Model (DOM) represents the structure of an HTML or XML document, and JavaScript provides powerful APIs to interact with the DOM. Developers can modify, add, or remove HTML elements, change styles, handle events, and update content dynamically.

Web APIs and Libraries: JavaScript has a vast ecosystem of built-in Web APIs (Application Programming Interfaces) that allow developers to access browser features and capabilities. These APIs include functionalities for manipulating browser history, making AJAX requests, performing animations, and more. Additionally, JavaScript supports various libraries and frameworks like React, Angular, and Vue.js, which provide additional abstractions and tools for building complex web applications.

Asynchronous Programming: JavaScript has excellent support for asynchronous programming through mechanisms like callbacks, promises, and async/await. Asynchronous programming allows executing tasks concurrently without blocking the execution of other code, enabling smooth user experiences and efficient handling of data requests.

Cross-Browser Compatibility: JavaScript is supported by all major web browsers, including Chrome, Firefox, Safari, and Edge. While some browser-specific differences exist, JavaScript code is generally portable across different platforms and browsers.

Server-side Development: Although primarily used for client-side scripting, JavaScript can also be used for server-side development using platforms like Node.js. Node.js enables running JavaScript on the server, allowing developers to build scalable and efficient back-end systems using a consistent language and toolset.

Libraries and Frameworks: JavaScript has a vast ecosystem of libraries and frameworks that simplify and accelerate web development. Popular libraries include jQuery, lodash, and moment.js, while frameworks like React, Angular, and Vue.js provide more comprehensive tools for building interactive web applications.

**7. SYSTEM IMPLEMENTATION**

**7.1 Software Installation**

**Installing Python:**

1. To download and install Python visit the official website of Python <https://www.python.org/downloads/> and choose your version.



**Screen 1 :-** Installating Python

1. Once the download is complete, run the exe for install Python. Now click on Install Now.
2. You can see Python installing at this point.
3. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

**Installing PyCharm:**

1. To download PyCharm visit website <https://www.jetbrains.com/pycharm/download/> and Click the “DOWNLOAD” link under the Commity Section.



**Screen 2 :-** Installing PyCharm

1. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.
2. On the next screen, Change the installation path if required. Click “Next”.
3. On the next screen, you can create a desktop shortcut if you want and click on “Next”.
4. Choose the start menu folder. Keep selected Jet Brains and click on “Install”.
5. Wait for the installation to finish.
6. Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.
7. After you click on "Finish," the Following screen will appear.



**Screen 3 :-** Creating Project in pyCharm

8. You need to install some packages to execute your project in a proper way.

9. Open the command prompt/ anaconda prompt or terminal as administrator.

10. The prompt will get open, with specified path, type “pip install package name” which you want to install (like numpy, pandas, seaborn, scikit-learn, matplotlib.pyplot)

Ex: pip install numpy

# 

**Screen 4 :-** Installing Libraries

**8.SOURCE CODE**

import React, { useState } from "react";

import { useNavigate } from "react-router";

import { getGroupById } from "../api/GroupApi";

import { loginUser } from "../api/UserApi";

import Validate from "../validation/ValidateLogin";

const user = JSON.parse(localStorage.getItem("token"));

const Login = () => {

const [data, setData] = useState({

email: "",

password: "",

});

const [errors, setErrors] = useState([]);

const navigate = useNavigate();

const handleChange = (e) => {

setData({

...data,

[e.target.name]: e.target.value,

});

};

const handleSubmit = async (e) => {

e.preventDefault();

const newErrors = Validate(data);

setErrors(newErrors);

if (!Object.keys(newErrors).length) {

try {

await loginUser(data, (user) => {

localStorage.setItem("token", JSON.stringify(user));

if (user.is\_superuser === false) {

navigate("/dashboard");

window.location.assign("/dashboard");

} else if (user.is\_superuser === true) {

navigate("/");

window.location.assign("/");

}

});

} catch (error) {

console.log(error);

}

}

};

return (

<>

<div

style={{

// width: "100%",

// minHeight: "100vh",

// backgroundImage: `url('https://storage.googleapis.com/twg-content/original\_images/sustainability-thumbnail.jpg')`,

// backgroundPosition: "center",

// backgroundRepeat: "revert",

// backgroundSize: "cover",

backgroundColor: "aliceblue",

paddingBottom: "4rem",

}}

>

<div>

<img

src="/src/assets/nursery.jpg"

className="px-5"

alt="logo"

style={{

height: "5rem",

marginTop: "2rem",

width: "16rem",

marginLeft: "2rem",

}}

/>

</div>

<div className="container w-75 py-4">

<div className="row">

<div className="col-7">

<img

src="https://thumbs.dreamstime.com/b/tropical-plant-nursery-4074172.jpg"

className="img-fluid"

alt="..."

style={{ height: "400px", borderRadius: "100px" }}

/>

</div>

<div className="col-4">

<div className="card mx-auto">

<div className="card-header d-flex">

<img

src="/login.png"

className="img-thumbnail w-50 me-4"

alt="logo"

/>

<h1 className="pt-3">Login</h1>

</div>

<div className="card-body">

<form onSubmit={handleSubmit}>

{/\* {errors && (errors.email ? toast.error(errors.email) : toast.error(errors.password))} \*/}

<div className="mb-3">

<label

htmlFor="formGroupExampleInput"

className="form-label"

>

Email

</label>

<input

type="email"

className="form-control"

id="formGroupExampleInput"

placeholder="Enter Email"

name="email"

value={data.email}

onChange={handleChange}

/>

{errors && errors.email && (

<span className="text-danger">{errors.email}</span>

)}

</div>

<div className="mb-3">

<label

htmlFor="formGroupExampleInput2"

className="form-label"

>

Password

</label>

<input

type="password"

className="form-control"

id="formGroupExampleInput2"

placeholder="Enter Password"

name="password"

value={data.password}

onChange={handleChange}

/>

{errors && errors.password && (

<span className="text-danger">{errors.password}</span>

)}

</div>

<div className="d-grid gap-2">

<button className="btn btn-success" type="submit">

Login

</button>

</div>

</form>

<div className="d-grid gap-2 w-25 mx-3 mt-2 float-end">

<button

className="btn btn-info text-white"

type="submit"

onClick={() => {

navigate("/registerUsers");

}}

>

Register

</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</>

);

};

export default Login;

import React, { useEffect, useState } from "react";

import { getShipment } from "../../api/ShipmentApi";

import BarChart from "../../components/BarChart";

import Navbar from "../../components/Navbar";

import PieChart from "../../components/PieChart";

import Table from "../../components/Table";

import AllReturnedProducts from "./AllReturnedProducts";

import OrderedProducts from "./OrderedProducts";

import ProductTracking from "./ProductTracking";

const CustomerDashboard = () => {

const [details, setDetails] = useState([]);

const [display, setDisplay] = useState(false);

const getShipping = async () => {

try {

const res = await getShipment();

setDetails(res.data);

} catch (error) {

console.log(error);

}

};

useEffect(() => {

getShipping();

}, []);

return (

<>

<div style={{ backgroundColor: "rgb(230 238 245)", height:"100vh"}}>

<Navbar />

<div className="container pt-5">

<div className="container mt-5">

<div className="row">

<div className="col-12">

<button

className="btn btn-success mx-5"

type="button"

onClick={() => setDisplay(false)}

>

Ordered Nursery Plants

</button>

</div>

</div>

</div>

<div>{!display && <OrderedProducts />}</div>

</div>

</div>

</>

);

};

export default CustomerDashboard;

import React, { useContext, useEffect, useState } from "react";

import Navbar from "../../components/Navbar";

import CartContext from "../../utils/CartContext";

import { FaRupeeSign } from "react-icons/fa";

import { addPayment } from "../../api/PaymentApi";

import { useNavigate } from "react-router-dom";

import { toast } from "react-hot-toast";

import Validate from "../../validation/ValidatePayment";

import axios from "axios";

import moment from "moment/moment";

const user = JSON.parse(localStorage.getItem("token"));

const Payment = () => {

const { total, cartItems, clearCart } = useContext(CartContext);

const [payment, setPayment] = useState({

id: "",

payment\_done: "",

payment\_date: "",

card\_No: "",

card\_Name: "",

expiry\_date: "",

cvv: "",

});

const [data, setData] = useState({

quantity: "",

total\_price: "",

user: user.id,

shipment: "",

products: "",

expected\_date: moment().add(5, "days").format("YYYY-MM-DD"),

});

const [errors, setErrors] = useState([]);

const [display, setDisplay] = useState(false);

const navigate = useNavigate();

const handleChange = (e) => {

setData({

...data,

[e.target.name]: e.target.value,

});

};

const handlePayChange = (e) => {

setPayment({

...payment,

[e.target.name]: e.target.value,

});

};

const sendMailToUser = async () => {

try {

const body = {

Recipients: [

{

Email: user.email,

},

],

Content: {

Body: [

{

ContentType: "HTML",

Content: `Order ID has been send on Your Email...}`,

Charset: "string",

},

],

From: "cse.takeoff@gmail.com",

Subject: "Order ID Send On Mail",

},

};

const headers = {

"X-ElasticEmail-ApiKey":

" FCE09D54A8F56F09716DC8FB3550636773E71D1219444F7EF87613C6E8B29E58621E031BBCB92D5D51ECBC63E0DA7BE9",

};

await axios.post(`https://api.elasticemail.com/v4/emails`, body, {

headers: headers,

});

// toast.success(`OTP has been sent to your mail.`, {});

// toast.success(`Order Id has been sent`, {});

} catch (error) {

console.log(error);

toast.error(error.response.data.msg, {});

}

};

const handlePaymentSubmit = async (e) => {

e.preventDefault();

const newErrors = Validate(payment);

setErrors(newErrors);

if (!Object.keys(newErrors).length) {

try {

const item = [];

for (let i of cartItems) {

item.push(i.id);

}

data.products = item;

sendMailToUser();

const res = await addPayment(data);

navigate("/viewProducts");

clearCart();

setDisplay(false);

// toast.success("Order Id has been sent");

toast(

(t) => (

<span>

<div className="ml-3 flex-1">

<p className="text-sm font-medium text-gray-900">Order ID</p>

<p className="mt-1 text-sm text-gray-500">

Your Order Id is :

<span className="text-info">{res.data.id}</span>

</p>

</div>

<button

className="btn btn-success"

onClick={() => toast.dismiss(t.id)}

>

Dismiss

</button>

</span>

),

{ duration: 7000 }

);

} catch (error) {

console.log(error);

}

}

};

const handleClick = () => {

setDisplay(true);

};

return (

<>

<div style={{ backgroundColor: "rgb(237 238 241)", height: "100vh" }}>

<Navbar />

<div className="container mt-5 pt-3">

<div className="row pt-5">

<div className="col-6">

<div className=" card py-3 w-75">

<div className="h4 mx-5 pb-3">

Hello,

<span className="px-3 h3 text-info">

{user.first\_name.toUpperCase()}

</span>

</div>

<div className="mx-3">

<div className="h4">

Total Items :{" "}

<span style={{ color: "#0d6efd" }}>{cartItems.length}</span>

</div>

<div className="h4">

Total Items Price :{" "}

<span style={{ color: "#0d6efd" }}>

<FaRupeeSign fontSize="20px" />

{total} /-

</span>

</div>

</div>

<div className="mx-auto">

<button

className="btn btn-success mt-3 "

// data-bs-toggle="modal"

// data-bs-target="#exampleModal"

onClick={handleClick}

>

Payment

</button>

</div>

</div>

</div>

<div className="col-6">

{display && (

<div className="card px-3 pb-3 w-75">

<form onSubmit={handlePaymentSubmit} className="row g-3">

<div className="col-8">

<input

type="text"

className="form-control"

id="inputUsername4"

name="payment\_done"

value={(payment.payment\_done = true)}

onChange={handlePayChange}

hidden

/>

<input

type="text"

className="form-control"

id="inputUsername4"

name="payment\_date"

value={(payment.payment\_date = new Date())}

onChange={handlePayChange}

hidden

/>

<input

type="text"

className="form-control"

id="inputUsername4"

name="quantity"

value={(data.quantity = cartItems.length)}

onChange={handleChange}

hidden

/>

<input

type="text"

className="form-control"

id="inputUsername4"

name="total\_price"

value={(data.total\_price = parseInt(total))}

onChange={handleChange}

hidden

/>

<input

type="text"

className="form-control"

id="inputUsername4"

name="user"

value={(data.user = user.id)}

onChange={handleChange}

hidden

/>

</div>

<div className="col-8">

<label htmlFor="inputUsername4" className="form-label">

Card Name

</label>

<input

type="text"

className="form-control"

id="inputUsername4"

name="card\_Name"

value={payment.card\_Name}

onChange={handlePayChange}

/>

{errors && errors.card\_Name && (

<span className="text-danger">{errors.card\_Name}</span>

)}

</div>

<div className="col-12">

<label htmlFor="inputUsername4" className="form-label">

Card Number

</label>

<input

type="text"

className="form-control"

id="inputUsername4"

name="card\_No"

value={payment.card\_No}

onChange={handlePayChange}

/>

{errors && errors.card\_No && (

<span className="text-danger">{errors.card\_No}</span>

)}

</div>

<div className="col-6">

<label htmlFor="inputUsername4" className="form-label">

Expiry Date

</label>

<input

type="date"

className="form-control"

id="inputUsername4"

name="expiry\_date"

min={moment(new Date()).utc().format("YYYY-MM-DD")}

value={payment.expiry\_date}

onChange={handlePayChange}

/>

{errors && errors.expiry\_date && (

<span className="text-danger">

{errors.expiry\_date}

</span>

)}

</div>

<div className="col-1"></div>

<div className="col-3">

<label htmlFor="inputUsername4" className="form-label">

CVV

</label>

<input

type="text"

className="form-control"

id="inputUsername4"

name="cvv"

value={payment.cvv}

onChange={handlePayChange}

/>

{errors && errors.cvv && (

<span className="text-danger">{errors.cvv}</span>

)}

</div>

<div className="d-grid gap-2">

<button

className="btn btn-success w-50 mx-auto"

type="submit"

>

PAYMENT

</button>

</div>

</form>

</div>

)}

</div>

</div>

</div>

</div>

</>

);

};

export default Payment;

import axios from "axios";

import React, { useEffect, useState } from "react";

import { toast } from "react-hot-toast";

import { useNavigate } from "react-router";

import { getGroup } from "../../api/GroupApi";

import { addUser } from "../../api/UserApi";

import Navbar, { UserNavbar } from "../../components/Navbar";

import Validate from "../../validation/ValidateUser";

const RegisterUsers = () => {

const [data, setData] = useState({

username: "",

first\_name: "",

email: "",

password: "",

groups: "",

});

const [errors, setErrors] = useState([]);

const navigate = useNavigate();

const handleChange = (e) => {

setData({

...data,

[e.target.name]: e.target.value,

});

};

const sendMailToUser = async () => {

try {

const body = {

Recipients: [

{

Email: data.email,

},

],

Content: {

Body: [

{

ContentType: "HTML",

Content: ` You are added for Ship Yard...`,

Charset: "string",

},

],

From: "cse.takeoff@gmail.com",

Subject: "You have been added in Ship Yard!",

},

};

const headers = {

"X-ElasticEmail-ApiKey":

" FCE09D54A8F56F09716DC8FB3550636773E71D1219444F7EF87613C6E8B29E58621E031BBCB92D5D51ECBC63E0DA7BE9",

};

await axios.post(`https://api.elasticemail.com/v4/emails`, body, {

headers: headers,

});

toast.success(`Email Sended to Customers Successfully!`, {});

} catch (error) {

console.log(error);

toast.error(error.response.data.msg, {});

}

};

const handleSubmit = async (e) => {

e.preventDefault();

const newErrors = Validate(data);

setErrors(newErrors);

if (!Object.keys(newErrors).length) {

try {

const role = [];

role.push(data.groups);

data.groups = role;

// sendMailToUser();

await addUser(data);

navigate("/");

} catch (error) {

toast.error(error.response.data.username);

}

}

};

return (

<>

<div

style={{

width: "100%",

minHeight: "100vh",

backgroundImage: `url('https://c0.wallpaperflare.com/preview/795/1024/833/greenhouse-hanging-baskets-seedlings.jpg')`,

backgroundPosition: "center",

backgroundRepeat: "revert",

backgroundSize: "cover",

}}

className="pb-2"

>

{/\* <UserNavbar /> \*/}

<div className="container py-5">

<div className="card mx-auto w-50 mt-4">

<div className="card-header d-flex">

<h1>Register</h1>

<img

src="/login.png"

className="img-thumbnail mx-auto w-25"

alt="logo"

style={{ height: "3.5rem" }}

/>

</div>

<div className="card-body">

<form onSubmit={handleSubmit} className="row g-3">

<div className="col-md-6">

<label htmlFor="inputUsername4" className="form-label">

Username

</label>

<input

type="text"

className="form-control"

id="inputUsername4"

name="username"

value={data.username}

onChange={handleChange}

/>

{errors && errors.username && (

<span className="text-danger">{errors.username}</span>

)}

</div>

<div className="col-md-6">

<label htmlFor="inputName4" className="form-label">

Name

</label>

<input

type="text"

className="form-control"

id="inputName4"

name="first\_name"

value={data.first\_name}

onChange={handleChange}

/>

{errors && errors.first\_name && (

<span className="text-danger">{errors.first\_name}</span>

)}

</div>

<div className="col-md-6">

<label htmlFor="inputEmail4" className="form-label">

Email

</label>

<input

type="email"

className="form-control"

id="inputEmail4"

name="email"

value={data.email}

onChange={handleChange}

/>

{errors && errors.email && (

<span className="text-danger">{errors.email}</span>

)}

</div>

<div className="col-md-6">

<label htmlFor="inputPassword4" className="form-label">

Password

</label>

<input

type="password"

className="form-control"

id="inputPassword4"

name="password"

value={data.password}

onChange={handleChange}

/>

{errors && errors.password && (

<span className="text-danger">{errors.password}</span>

)}

</div>

<div className="col-6">

{/\* <select

className="form-select"

aria-label="Default select example"

type="text"

name="groups"

value={data.groups = "delivery"}

onChange={handleChange}

>

<option defaultValue>Role</option>

{groups &&

groups.map((role, index) => {

return (

<option key={index} value={(role.id = 2)}>

{role.name }

</option>

);

})}

</select> \*/}

<input

type="text"

className="form-control"

id="inputgroups4"

name="groups"

value={(data.groups = 1)}

onChange={handleChange}

hidden

/>

{errors && errors.groups && (

<span className="text-danger">{errors.groups}</span>

)}

</div>

<div className="d-grid gap-2">

<button

className="btn btn-success w-50 mx-auto"

type="submit"

>

Create

</button>

</div>

</form>

<div className="d-grid gap-2 w-25 mx-3 mt-3 float-end">

<button

className="btn btn-info text-white"

type="submit"

onClick={() => {

navigate("/login");

}}

>

Login

</button>

</div>

</div>

</div>

</div>

</div>

</>

);

};

export default RegisterUsers;

**9. SYSTEM STUDY AND TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that theSoftware system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**9.1. Types of Tests**

**9.1.1. Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**9.1.2. Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered. **Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**9.1.3. Functional testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

* Valid Input : identified classes of valid input must be accepted.
* Invalid Input : identified classes of invalid input must be rejected.
* Functions : identified functions must be exercised.
* Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**9.2. SYSTEM TEST**

**9.2.1. White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**9.2.2. Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**9.2.3. Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**9.2.4 Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

* Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.
* The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**9.2.4. Test strategy and approach**

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

**9.2.5. Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

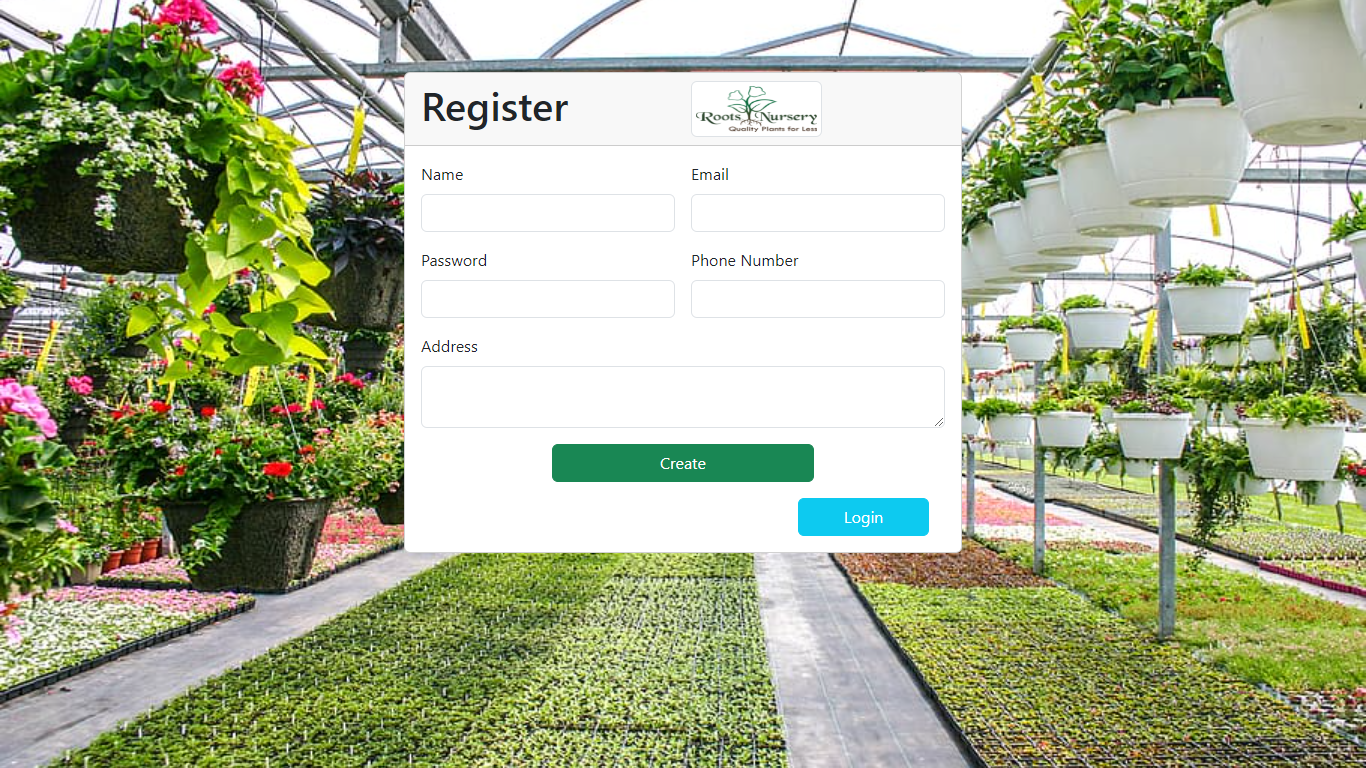
**9.2.6. Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

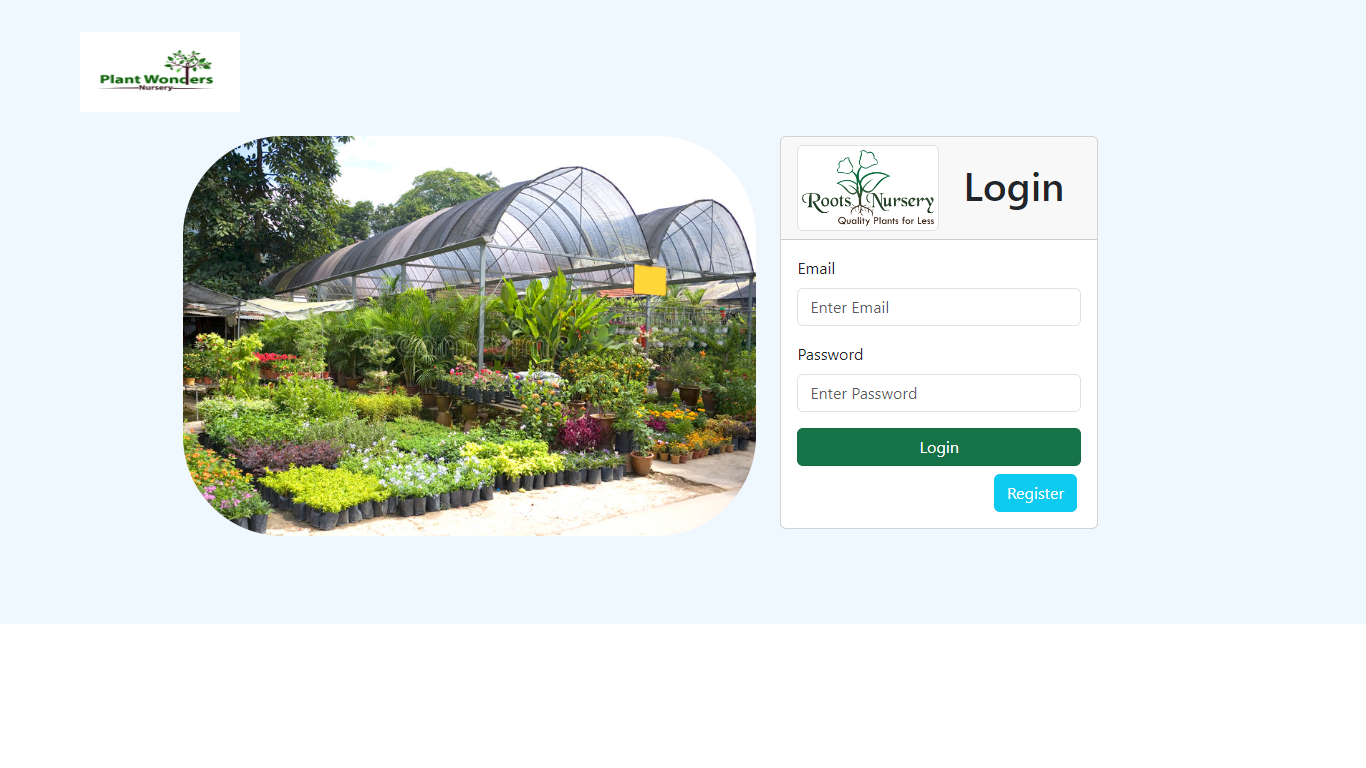
**10. OUTPUT SCREENS**

**Register:** Customer can register with their personal details such as name, email, password, phone number and address.



**Screen 5:-** Register Page for Users

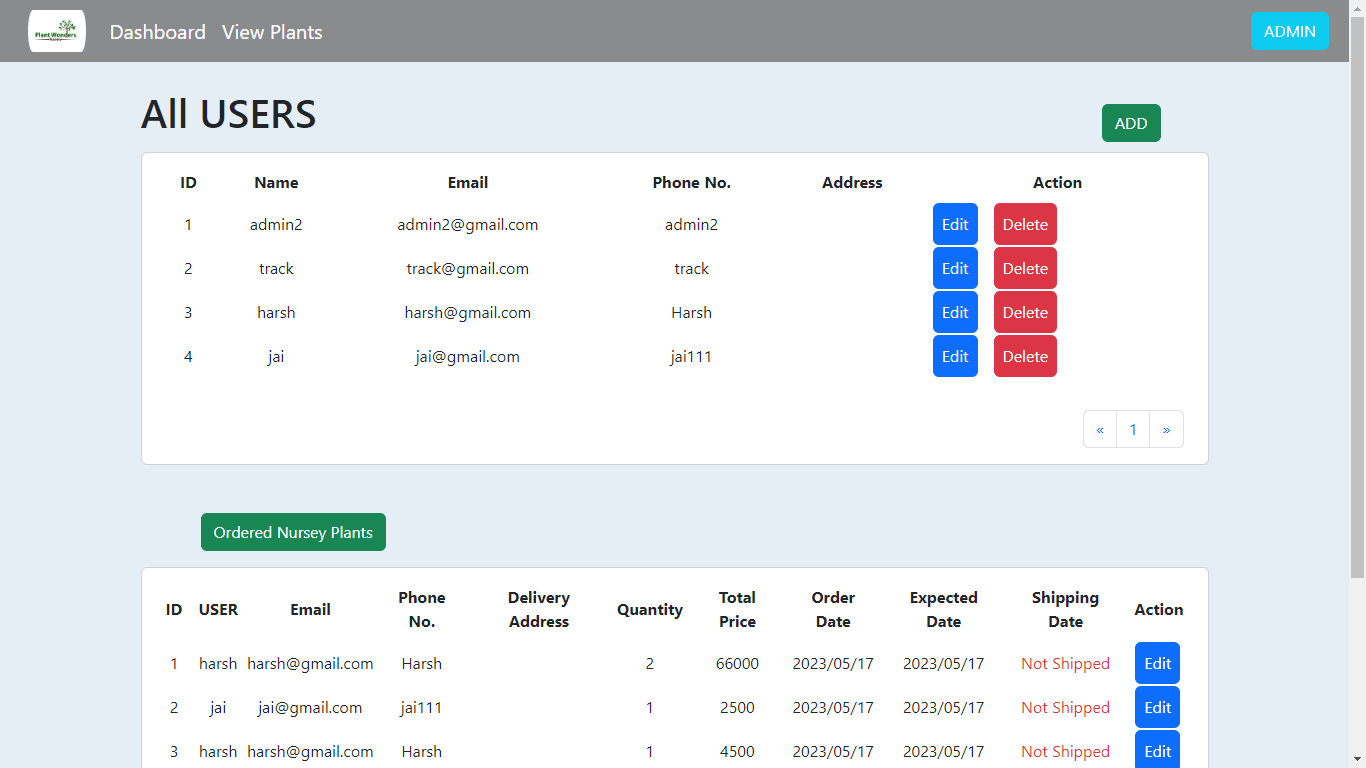
**Login:** Customer must login with valid email and password.



**Screen 6:-** Login Page for both User and Admin

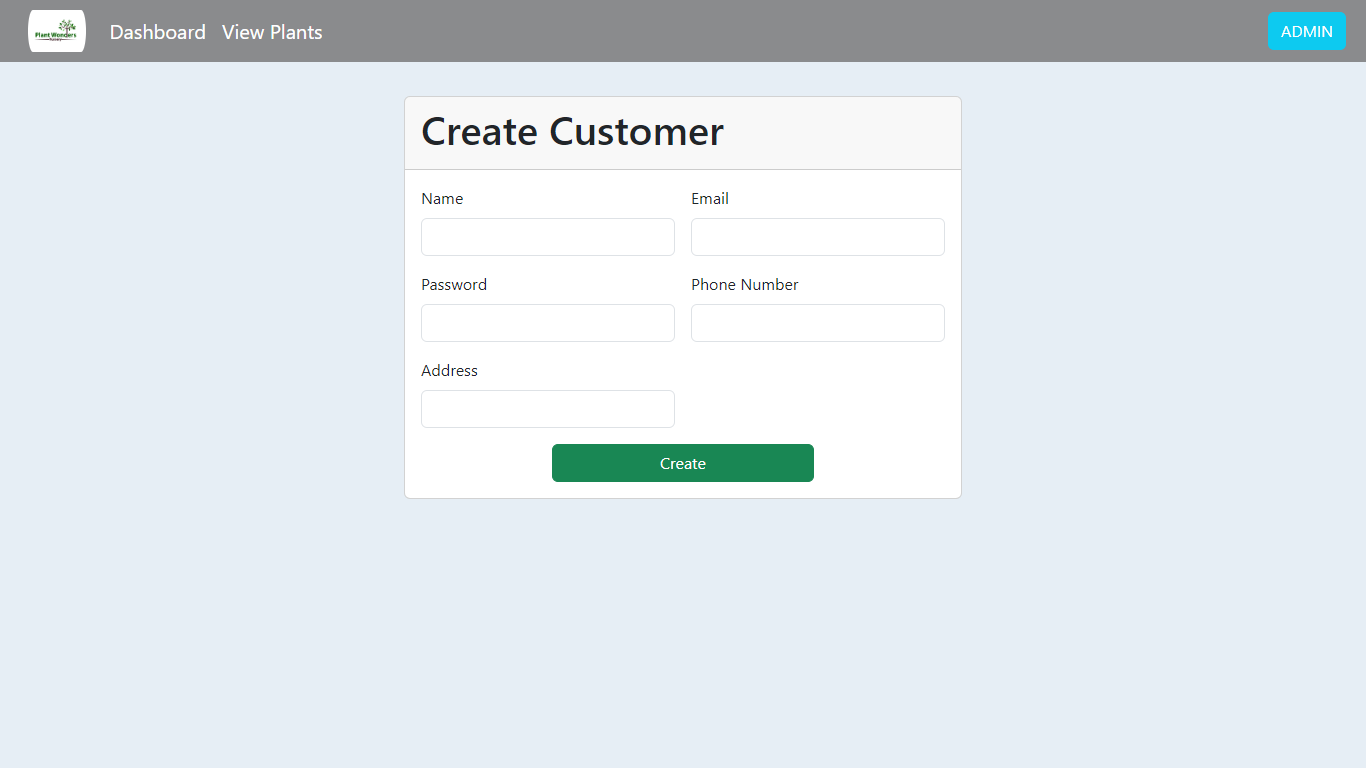
**All Users:** Admin can view all registered customers.

**View ordered plants:** Admin can view all plants which is ordered by customers.



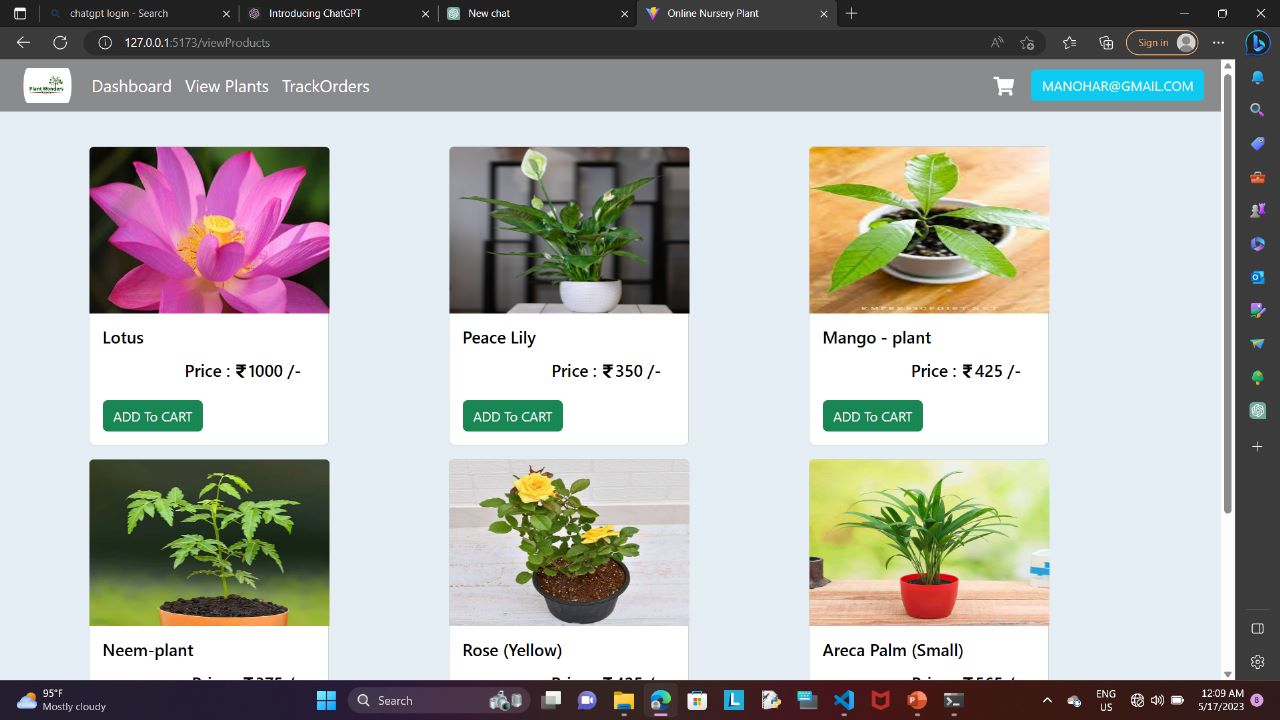
**Screen 7:-** Dashboard for Admin

**Add Customer:** Admin can add the customer with their personal details such as name, email, password, phone number and address.



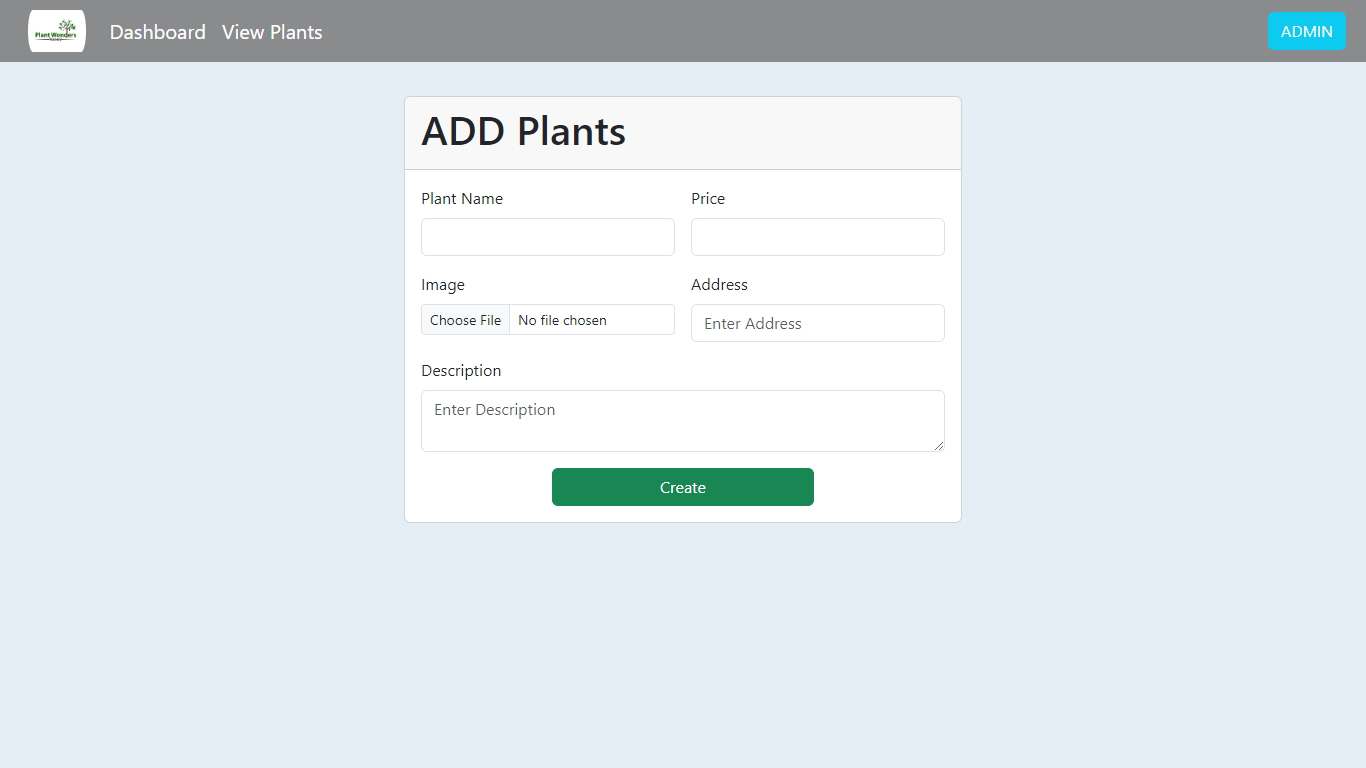
**Screen 8:-** Admin Add Customer page

**View Plants:** Admin can view all added plants along with plant name, price and image.



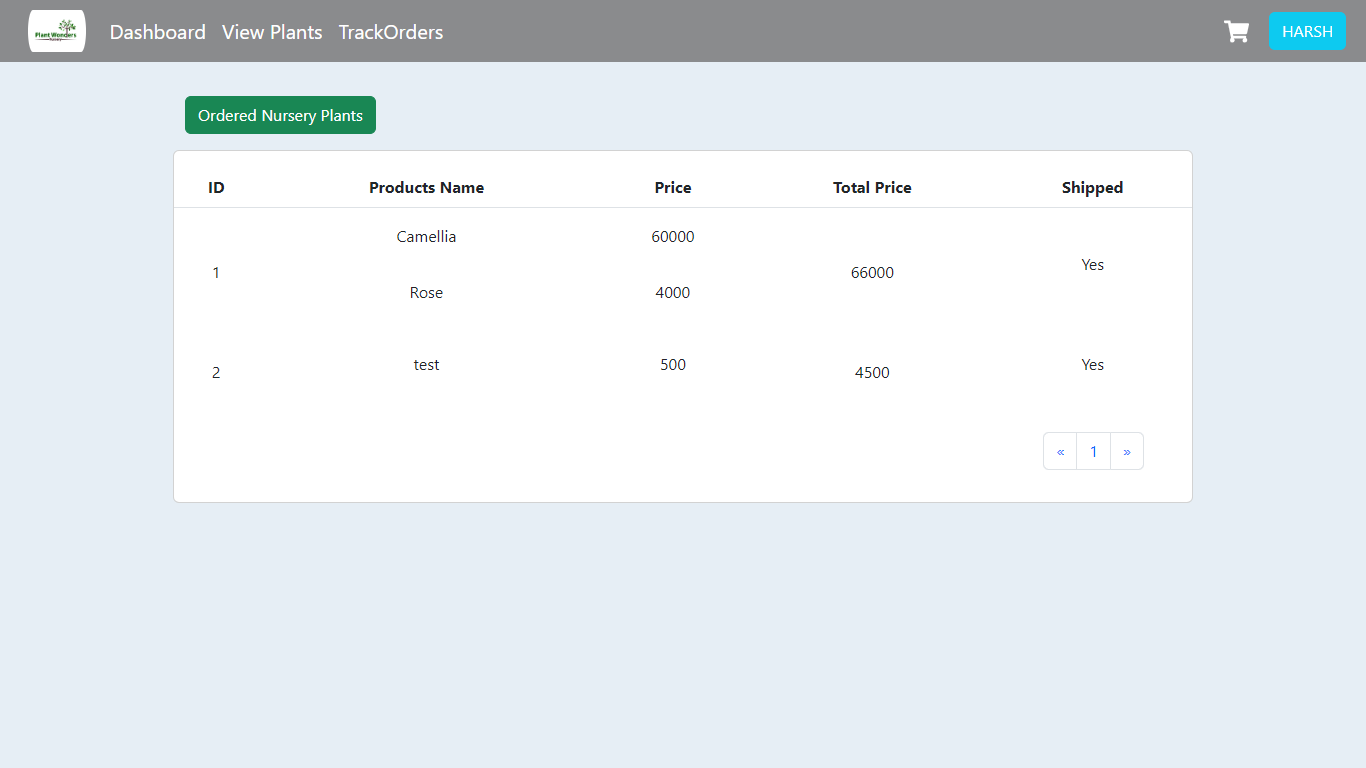
**Screen 9**:- List of Plants Page

**Add Plants:** Admin can add plants with details such as plant name, price, image, address and description.



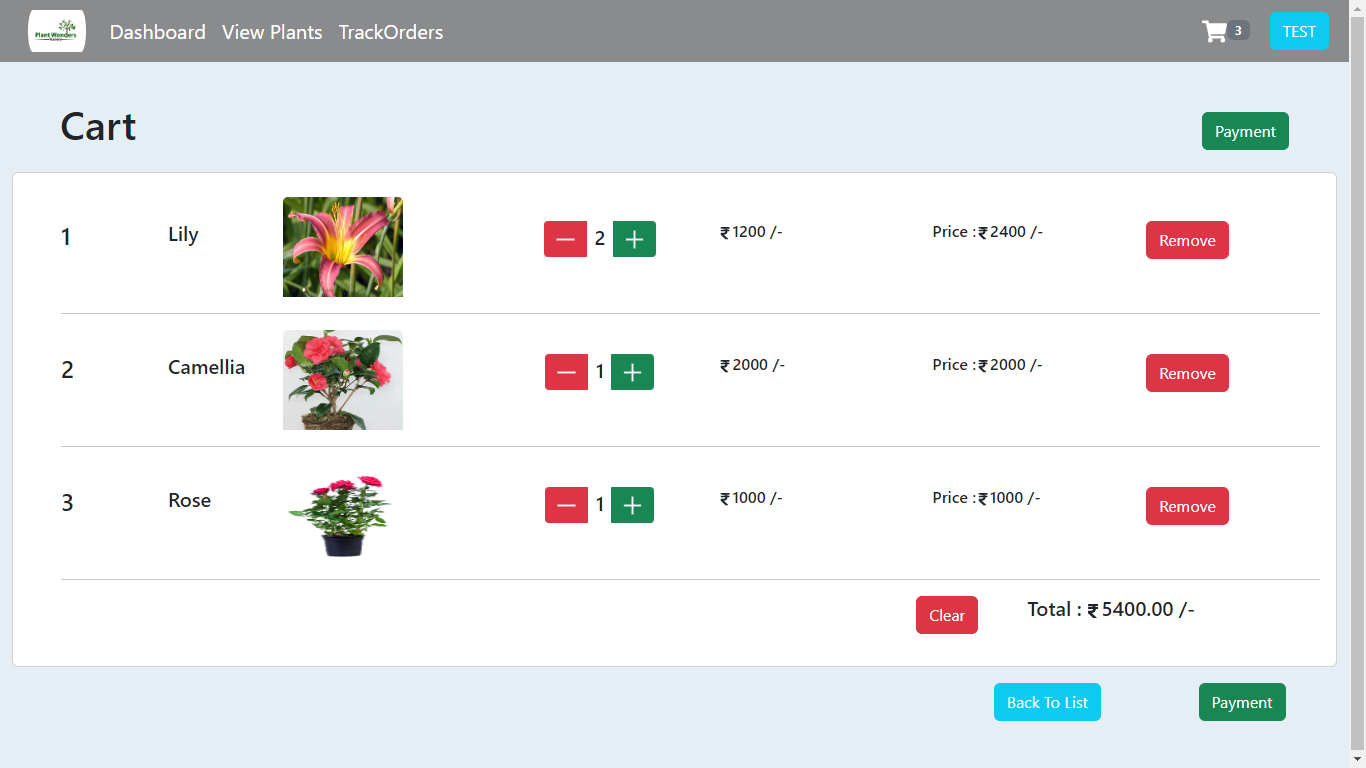
**Screen 10:-** Admin Add the plants page

**View Ordered Plants:** Admin can view all ordered plants.



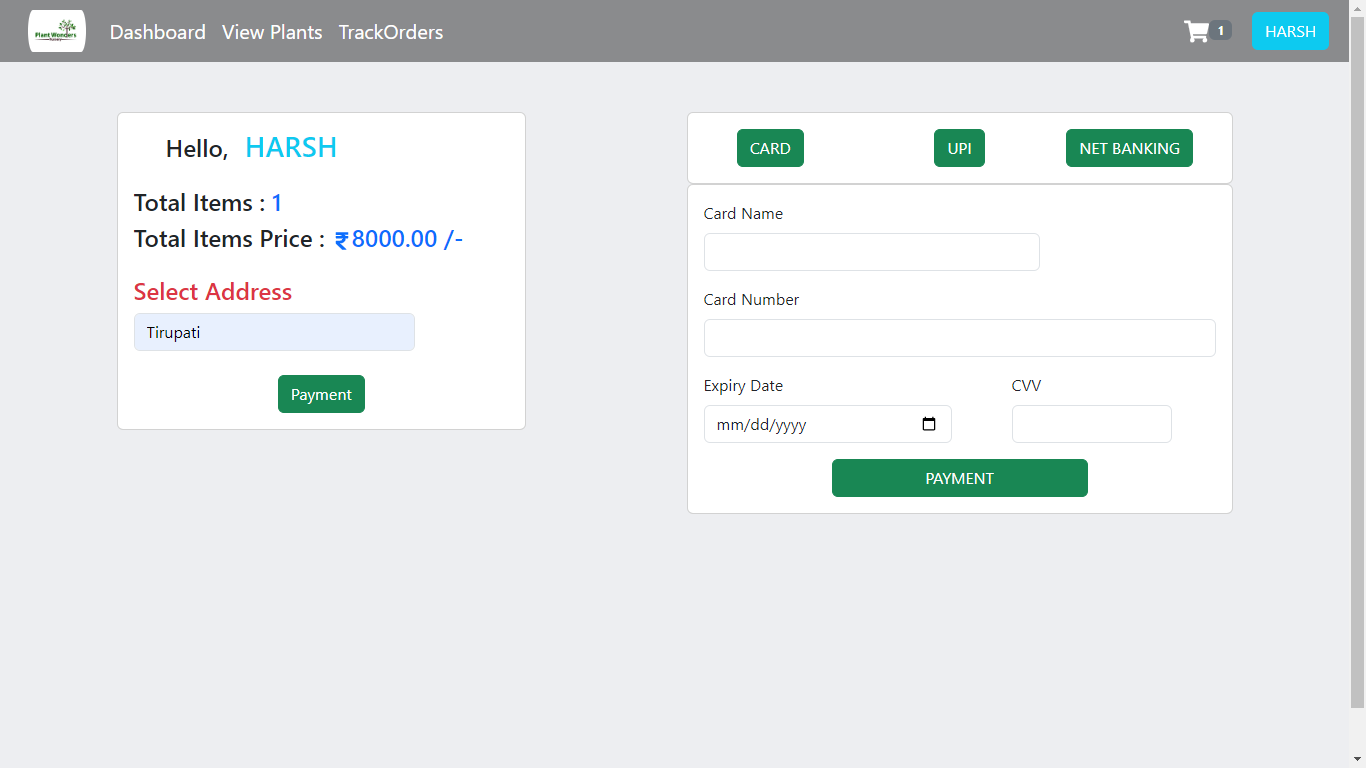
**Screen 11:-**View Ordered Plants Page

**Add to cart:** Customer can add plants to cart along with quantity and customer can remove any plant from cart.



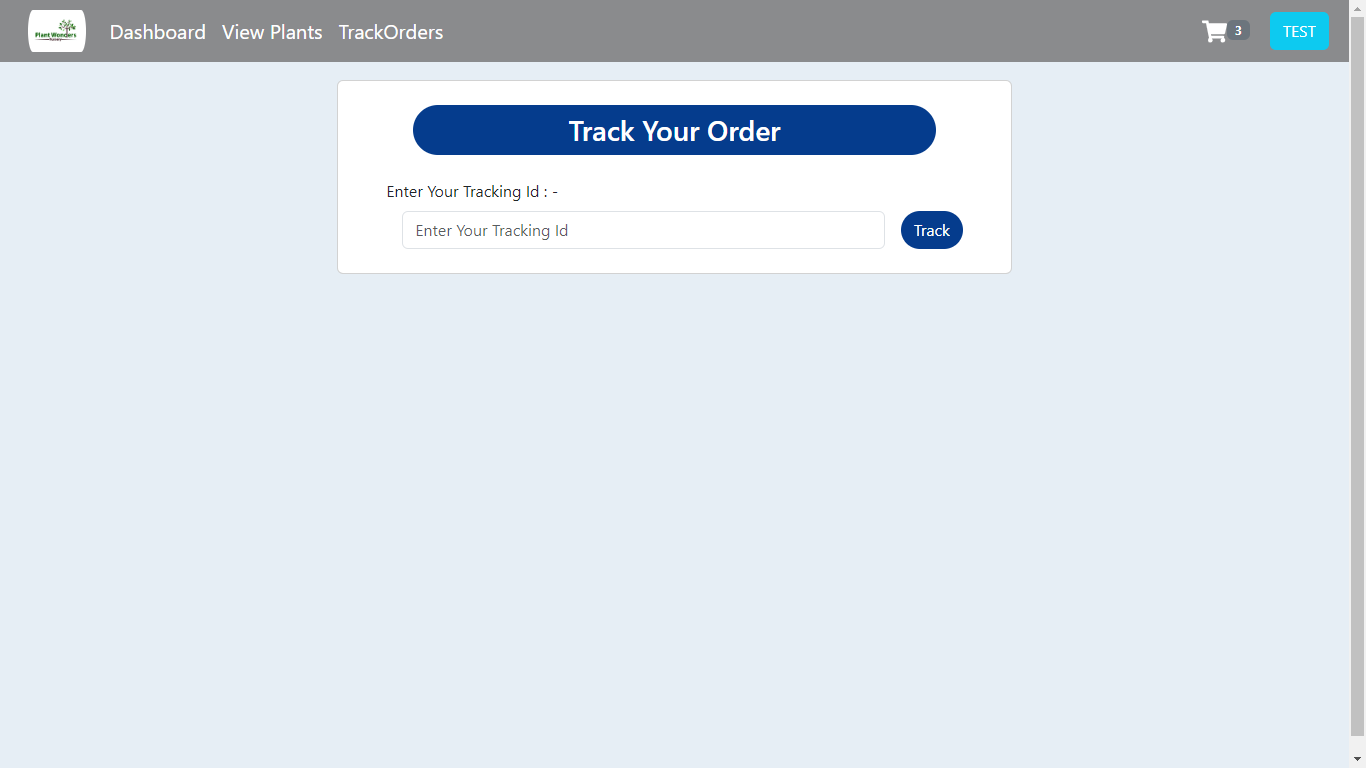
**Screen 12:-** Add to Cart Page

**Payment:** Customer can make payment (by providing card name, card number and expiry date and CVV) form their purchasing plant to place their order.



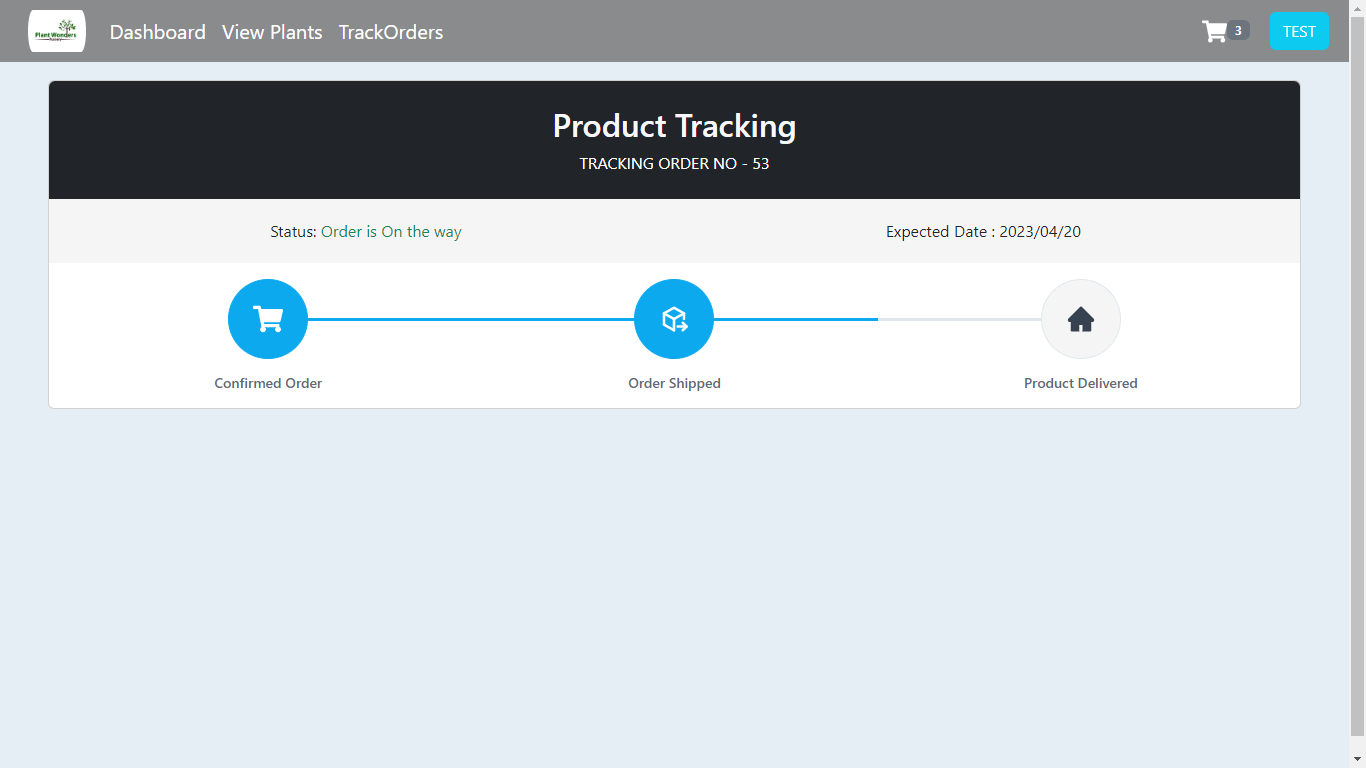
**Screen 13:-**Payment Page

**Track order:** Customer can track their order with the help of tracking ID.



**Screen 14:-** Track the order Page

**Tracking Status:** Customer can view the status of their order delivery/ship tracking for their order plant.



**Screen 15:-** Tracking status

**11.CONCLUSION**

The rise of online nursery plant platforms has brought about a transformative shift in the world of plant purchasing. By providing a convenient and accessible space for individuals to browse, select, and order plants, these platforms have addressed the limitations of traditional methods. With a wide variety of plants, detailed information, secure transactions, and reliable delivery services, customers can effortlessly enhance their living spaces with greenery. Moreover, the sense of community cultivated through interactive features fosters knowledge sharing and connects plant enthusiasts worldwide. Overall, online nursery plant platforms have revolutionized the way we bring nature into our homes, making it easier and more enjoyable for everyone to embrace the benefits of indoor gardening.

**12. FUTURE SCOPE**

In the future, the online nursery plant platform can consider incorporating the following enhancements to further improve the customer experience:

Augmented Reality (AR) Technology: Integrate AR technology that allows customers to visualize how different plants will look in their specific living spaces. This immersive experience can help customers make more informed decisions and visualize the aesthetics of their gardens or indoor settings.

Plant Care App: Develop a dedicated mobile app that offers personalized plant care reminders, watering schedules, and pest control tips. This app can provide real-time notifications and guidance to help customers effectively care for their plants and promote their healthy growth.

Sustainability Initiatives: Implement eco-friendly packaging materials and partner with carbon-neutral delivery services to reduce the environmental impact of plant transportation. Additionally, consider promoting sustainable gardening practices, such as offering organic plant food options and promoting the use of compost and renewable resources.

Plant Swapping Community: Introduce a feature that allows users to connect with local plant enthusiasts and engage in plant swapping. This promotes sustainability, expands plant variety, and fosters community interaction among plant lovers.

**REFERENCES**

[1] Mohammad Ausaf Anwar, Durgaprasad Gangodkar, “Design and Implementation of Mobile Phones based Attendance Marking System”, Department of Computer Science Engineering, Graphic Era University, Dehradun, Uttarakhand, India, 2015.

[2] Jun Lio, “Attendance Management System using a Mobile Device and a Web Application”, Department of Socio-informatics, Faculty of Letters Chuo University 742-1 Higashinakano, Hachioji-shi, Tokyo 192-0393, Japan, 2016.

[3] Mahesh G, Jayahari KR, Kamal Bijlani, “A Smart Phone Integrated Smart Classroom”, Amrita e-Learning Research Lab (AERL) Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, Amrita University, India, 2016.

[4] Ekta Chhatar, Heeral Chauhan, Shubham Gokhale, Sompurna Mukherjee, Prof. Nikhil Jha, “Survey on Student Attendance Management System”, S.B. Jain Institute of Technology, Management and Research, Nagpur, 2016.

[5] Md. Milon Islam, Md. Kamrul Hasan, Md Masum Billah, Md. Manik Uddin, “Development of Smartphone-based Student Attendance System”, Department of Computer Science and Engineering Khulna University of Engineering & Technology, Khulna-9203, Bangladesh, 2017.

[6] Karwan Jacksi, Falah Ibrahim, Shahab Ali, “Student Attendance Management System”, University of Zakho, Iraq, 2018.