Functional and Non-Functional Requirements

Functional Requirements

Template Method

FR1: The system shall allow the definition of a general plant care routine, while supporting variations for specific plant types.

Why?
To avoid code duplication and enforce a standard care sequence with customizable steps. The base class defines the general case procedure and subclasses override specific care steps such as watering amount or sunlight duration.

State

FR2: The system shall track each plant's life cycle and adjust its behaviour based on its current growth state.

Why?
The State pattern is used to represent the various stages of a plant's life. Each state encapsulates behaviour relevant to that phase. When a plant's state changes, its behaviour dynamically adapts without multiple conditionals.

Mediator
FR3: The system shall maintain an up – to – date inventory of all plants and coordinate updates between the plants, stock and sales subsystem. Coordinate actions between greenhouse and sales floor staff.

Why?
To prevent tight coupling between multiple patterns that need to inventory, the Mediator pattern is used. Without coordination, each subsystem would directly communicate with each other.

Abstract Factory
FR4: The system shall allow new plant stock to be added efficiently by either generating new plant types.

Why?
The Abstract Factory pattern is used to create families of related plant objects without specifying their concrete classes.

Prototype
FR5: The system shall allow new plant stock to be added efficiently by either cloning existing plants.

Why?
And the Prototype pattern may be used to duplicate existing plant objects with slight modifications to reduce the cost and the time of stock creation.

Observer
FR5: The system shall monitor the plant health and life cycles and automatically take action when plant conditions change and stock levels change.

Why?

The Observer pattern allows automatic detection and notification of state changes.

Factory Method

FR6: The system should support adding new staff roles dynamically as the nursery grows.

Why?
The nursery currently employs several staff roles such as Sales Staff, Gardeners, and Managers each with unique responsibilities.
As the nursery grows, new roles like Delivery Staff, Landscapers, or Greenhouse Managers will be introduced.
Using the Factory Method allows new staff types to be added without altering existing code.

Strategy

FR7: The system shall allow new customer interaction behaviours to be added without modifying existing staff code. One-to-one **staff ↔ customer** interactions.

Why?
Each type is encapsulated in a separate strategy class (e.g., AdviceStrategy, AvailabilityStrategy and AccessoriesStrategy), supporting system extensibility and maintainability. Staff can dynamically choose the appropriate strategy for each customer interaction.

why Strategy is the right choice for customer interaction, and why Mediator doesn't fit this scenario.

Customer ↔ Staff is mostly one-to-one and behaviour-driven:

- A customer asks a question or requests advice.
- A staff member responds based on the type of request.
- The response varies dynamically (care advice, stock info, upsell, etc.).

This is not a coordination problem between multiple components — it's about choosing the correct behaviour for a given request.

Customer ↔ Staff is mostly one-to-one and behaviour-driven:

- A customer asks a question or requests advice.
- A staff member responds based on the type of request.
- The response varies dynamically (care advice, stock info, etc.).

This is not a coordination problem between multiple components. It's about choosing the correct behaviour for a given request.

Another use of Strategy: Staff Task Handling Strategy using CoR

FR8: The system shall allow each staff member to execute delegated tasks using dynamically selected behaviour, once the Chain of Responsibility decides who handles the request.

Handles internal tasks delegated through the CoR staff chain like:

Why?
Each staff member can handle tasks differently based on their role and the task type. For example:

- Gardener → PlantCareStrategy (Template Method for watering, pruning, fertilizing)
- SalesStaff → InventoryUpdateStrategy (update inventory or notify Mediator)
- Manager → ApprovalStrategy (approve discounts or special requests)

Iterator

FR9: The system shall allow customers to view all available plants without exposing the internal structure of the nursery's plant collection**.** The system shall allow customers to select a plant from the collection and retrieve detailed information (care instructions, price, stock availability). The system shall allow customers to request plant recommendations based on categories or preferences.

Why?
Iterator provides a standard way to traverse the plant collection sequentially, abstracting the underlying vector. Customers can browse plants in any order supported by the iterator.

Decorator

FR10: The system shall allow customers to add optional decorations or enhancements to plants, such as decorative pots, gift wrapping, or special arrangements.

Why?
Decorator pattern allows extending the functionality of plant objects dynamically, without modifying the base Plant class. Each decoration or enhancement can be wrapped around the plant object.

Composite

FR11: The system shall allow customers to purchase individual plants or groups of plants in a single transaction. The system shall automatically update inventory levels when plants are purchased, whether individually or as part of a composite purchase. The system shall optionally record details of the transaction.

Why?
The Composite pattern allows both individual items (single plants) and composite items (bundled plants, arrangements) to be treated uniformly, simplifying purchase processing.

Chain of Responsibility

FR12: The system shall allow staff to forward tasks or requests they cannot handle to the next staff member in the chain. The system shall allow tasks requiring higher authority. The system shall allow new staff roles to be added dynamically to the chain without modifying existing staff or request handling logic.

Why?
Enables dynamic delegation of tasks without requiring the sender to know which staff member is responsible. Supports one-to-many staff workflows while reducing tight coupling.

Bridge (working with Abstract Factory)
FR13: The system shall allow flowers to have different colours without creating a separate class for each flower–colour combination. The system shall allow flowers to be dynamically assigned multiple colours or changed at runtime.

Why?
Separates flower abstraction (type) from colour/appearance (implementation), enabling flexible combinations.

Non-Functional Requirement

NFR1: Extensibility
The system shall be easily extendable to support new plant types, staff roles and customization options without modifying existing code.

Patterns that ensure this:

- Abstract Factory
- Strategy
- Template Method

NFR2: Maintainability
The system shall minimize code duplication and simplify maintenance.

Patterns that ensure this:

- Template Method
- Strategy
- Chain of Responsibility

NFR3: Scalability

The system shall handle growth in the number of plants, staff members, and customer interactions. Composite handles scalable plant structures and Mediator and Observer decouple components for scalable communication.

Patterns that ensure this:

- Composite
- Mediator

- Observer

NFR4: Performance
The system shall provide staff and customers with intuitive, predictable behaviours for common actions (watering, buying, decorating, etc.).

Patterns that ensure this:

- Mediator
- Observer
- Composite