OUTSYSTEMS APP DEVELOPMENT BEST PRACTICES USING THE 3 LAYER ARCHITECTURE CANVAS

WHAT IS THE ARCHITECTURE CANVAS?

The Architecture Canvas in OutSystems is a visual tool that helps software architects and developers to define and communicate the architecture of their applications effectively. It provides a structured way to capture key architectural decisions, components, and relationships in a single canvas or diagram.

The Architecture Canvas, an OutSystems architecture tool, simplifies the design of Service-Oriented Architectures (SOA), advocating for proper abstraction of reusable (micro)services and isolation of distinct functional modules. It's particularly beneficial when managing multiple applications sharing common modules. In a typical medium to large OutSystems setup, it supports over 20 mission-critical applications and 200+ interdependent modules.

The Architecture Canvas typically consists of several sections or layers, each representing a different aspect of the system's architecture. These layers often include:

- **Purpose**: This section describes the system's purpose, goals, and objectives. It defines the problem domain that the system aims to address and outlines the desired outcomes.

- **Stakeholders**: Here, the primary stakeholders involved in the system, such as users, customers, developers, and administrators, are identified. Their concerns, needs, and expectations regarding the system are documented.

- **Functional Requirements**: This section outlines the functional requirements of the system, describing the features, capabilities, and behaviors it should exhibit to satisfy stakeholders' needs.

- **Non-functional Requirements**: Non-functional requirements, such as performance, scalability, reliability, security, and usability, are documented in this section. These requirements define the quality attributes and constraints that the system must meet.

- **Architecture Decisions**: The key architectural decisions made during the design and development of the system are captured here. This includes decisions related to architectural styles, patterns, components, interfaces, and technologies.

- **Architecture Views**: Different views or perspectives of the system's architecture, such as logical, physical, and deployment views, are depicted in this section. Each view focuses on specific aspects of the architecture, providing different insights into the system's structure and behavior.

- **Dependencies and Relationships**: The dependencies and relationships between architectural components, such as modules, layers, and subsystems, are represented here. This helps visualize how different parts of the system interact and collaborate to fulfill the system's functionality.

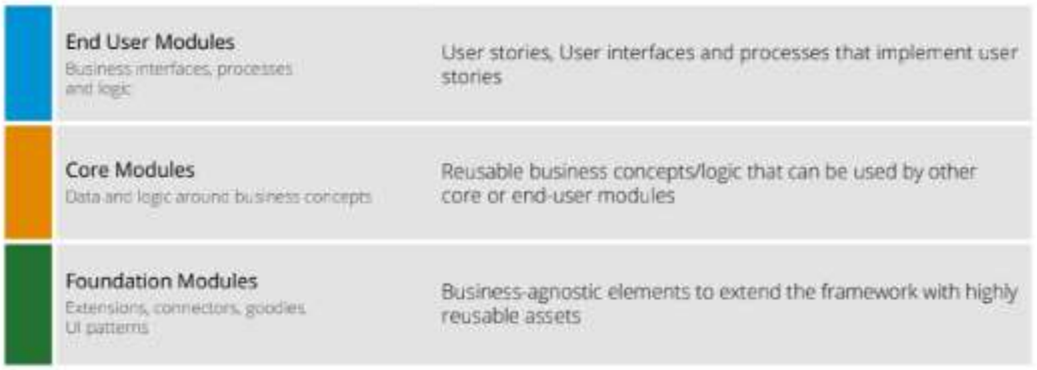
- **Roadmap and Evolution**: The future roadmap for the system's architecture and its evolution over time are outlined in this section. This includes planned enhancements, upgrades, migrations, and adaptations to accommodate changing requirements and technologies.

What are Architecture Solution Goals?

- Aligned with Business concepts
- Reusability promotion
- Maintainability
- Scalability (but still Flexible)
- Performant

What is 3 Layer Architecture Canvas and How It's used?

The 3 layer architecture is as follows.



🟦	**End User Modules** Business interfaces, processes and logic	User stories, User interfaces and processes that implement user stories
🟧	**Core Modules** Data and logic around business concepts	Reusable business concepts/logic that can be used by other core or end-user modules
🟩	**Foundation Modules** Extensions, connectors, goodies, UI patterns	Business-agnostic elements to extend the framework with highly reusable assets

- **EndUser Module:** Its store only interface Related Files and Also its visible to end user.

- **Core Modules:** Its Store Core Logic that reusable services around business concepts, Business roles and web blocks , Provide APIs to expose External API Services, Core Widgets, Composite Logic that help us for logic to synchronize data, Core Services

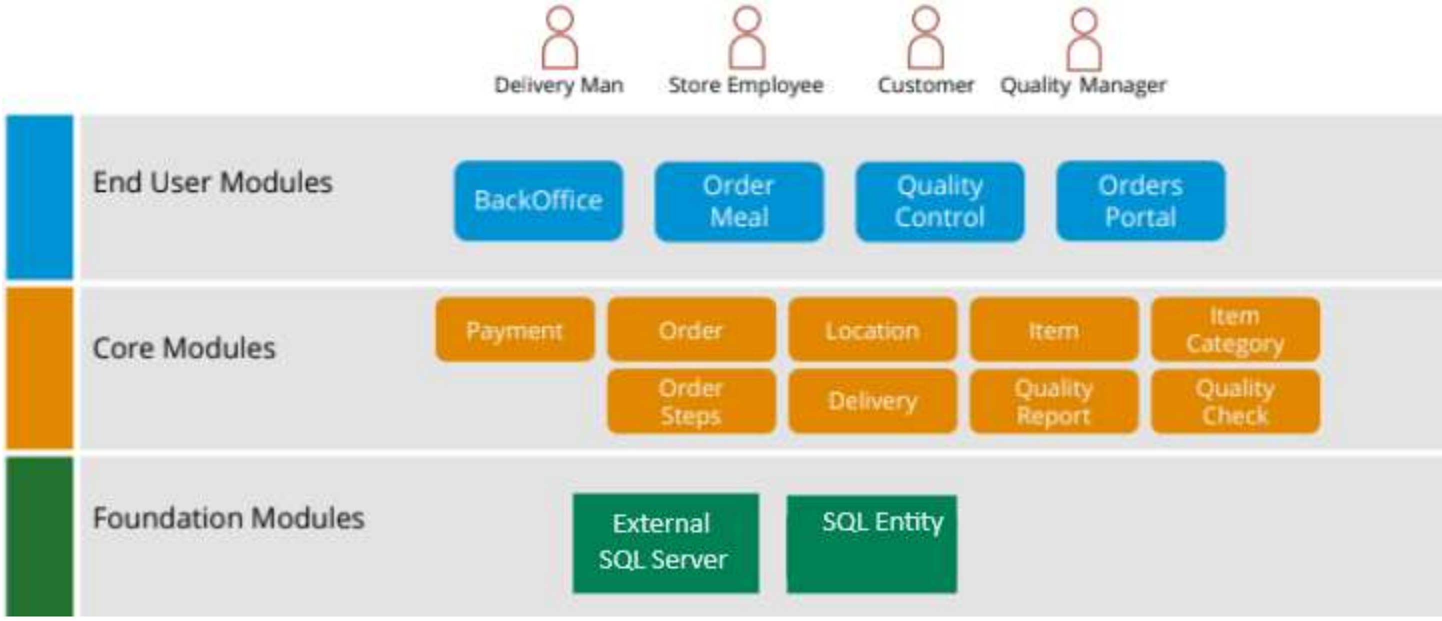
- **Foundation Modules:** Its store Non Functional Requirements or integration module reusable in any business context. Libraries for external plug-ins

Example
Requirement
Meal Ordering: Requested, prepared and delivered a meal order
- Order Management External ERP
- Delivery person Selected on the Delivery location.

Module used
Integration Needs: External SqlDataBase
Business Concepts: Payment, Delivery, Location, Order Steps, Order,Items
User Interface: Order Meal

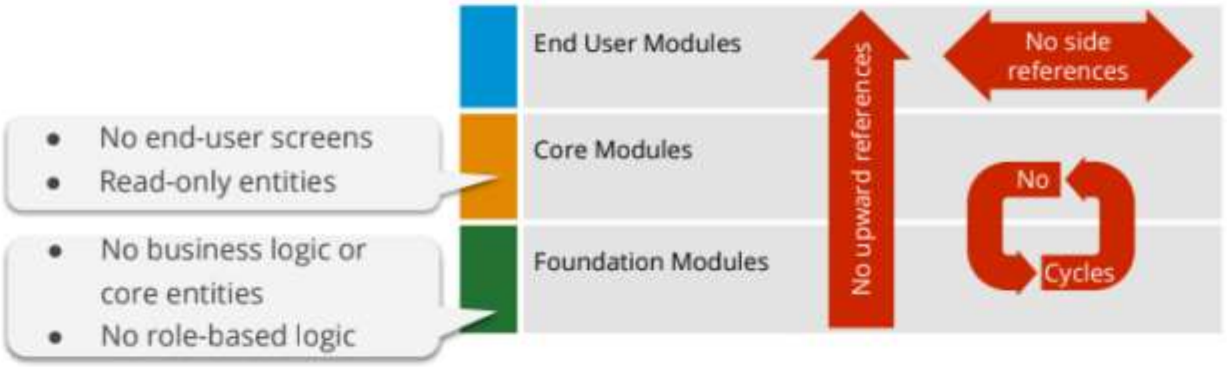
Users
Delivery Man, Store , Customer, Quality Manager, Super User

SOLUTION



Validating the Architecture

1. **No Upward references**
2. **No side references in end-user modules**
3. **No cyclic references**



End User Modules
Core Modules
Foundation Modules
No upward references
No side references
No Cycles
No end-user screens
Read-only entities
No business logic or core entities
No role-based logic