# Title: Dependency Injection in Angular- Real World Scenario Lab

Objective: The objective of this lab exercise is to understand and practice Dependency Injection (DI) in Angular by simulating a real-world scenario. Dependency Injection is a fundamental concept in Angular that facilitates loose coupling between different parts of an application.

Scenario: Imagine you are developing a shopping application where users can browse through products, add them to their cart, and proceed to checkout. To accomplish this, you need to implement various services for managing products, cart, and checkout processes. In this lab, you will create and inject these services using Angular's Dependency Injection mechanism.

Lab Steps:

1. Setting Up the Angular Project:
    - Create a new Angular project using Angular CLI.
    - Set up necessary components and modules for the shopping application, such as ProductListComponent, CartComponent, CheckoutComponent, ProductService, CartService, and CheckoutService.
2. Implementing Services:
    - Create a ProductService responsible for fetching and managing products. Mock some sample product data for testing purposes.
    - Create a CartService responsible for managing the user's shopping cart, including adding/removing items.
    - Create a CheckoutService responsible for handling the checkout process, including payment and order confirmation.
3. Implementing Dependency Injection:
    - Inject the ProductService into the ProductListComponent to fetch and display products.
    - Inject the CartService into the CartComponent to manage the shopping cart.
    - Inject the CheckoutService into the CheckoutComponent to handle the checkout process.
4. Testing Dependency Injection:
    - Verify that services are properly injected by logging messages or using Angular's dependency injection debugging tools.
    - Test the functionality of each component by adding products to the cart, proceeding to checkout, and confirming orders.

5. Dependency Injection in Child Components:
    - Create a child component, such as ProductItemComponent, to display individual product details within the ProductListComponent.
    - Inject the ProductService into the ProductItemComponent to fetch additional product details if necessary.
    - Test the dependency injection in child components by ensuring they have access to the required services.
6. Advanced Dependency Injection:
    - Implement a scenario where the CheckoutService depends on external payment gateway APIs.
    - Use Angular's providedIn property to provide services at the application level or feature level.
    - Explore different ways of providing services, such as useClass, useValue, useFactory, etc.
7. Additional Challenges (Optional):
    - Implement lazy loading for feature modules to improve application performance.
    - Utilize Angular's providedIn feature to optimize service injection and reduce bundle size.
    - Experiment with different DI patterns, such as Hierarchical DI vs. Injector Tree.

Conclusion: Reflect on the importance of Dependency Injection in Angular and how it promotes modular, testable, and maintainable code. Discuss any challenges faced during the lab exercise and how they were resolved. Encourage students to explore more advanced topics related to Angular's dependency injection mechanism.