CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND
COMMUNICATION TECHNOLOGY

PROJECT - SPECIALIZED REPORT
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

SOUVENIR APPLICATION WITH ASP.NET MICROSERVICES AND FLUTTER

Student: Nguyen Thanh Phat
Student ID: B2005853
Cohort: K46

Cantho, 09/2023

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND
COMMUNICATION TECHNOLOGY

PROJECT - SPECIALIZED REPORT
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

SOUVENIR APPLICATION WITH ASP.NET MICROSERVICES AND FLUTTER

Advisor:
Dr. Lam Van Khang

Student:
Nguyen Thanh Phat
Student ID: B2005853
Cohort: K46

Cantho, 09/2023

Contents

List of Figures

List of Tables

Abstract

This project aims to develop an online souvenir shop application using ASP.NET microservices and Flutter. The application will serve as an online store selling various physical products like t-shirts and coffee mugs, providing a familiar experience for those who have shopped online before.

The development process will utilize the microservices architecture with ASP.NET for the backend API, and Flutter for the frontend mobile application. This approach allows for a scalable and maintainable system, while also providing a rich, user-friendly interface.

The scope of the project is focused on creating a small application that satisfies the minimum requirements of an e-shop application. This includes features such as product listing, shopping cart functionality, and order processing.

The expected outcomes of this project include gaining a deeper understanding of microservices architecture, application development using Flutter and ASP.NET, and the business aspects of running an e-shop. The project serves as a practical application of these technologies and concepts, demonstrating their effectiveness in a real-world scenario.

Tóm tắt nội dung

Dự án này nhằm mục đích phát triển một ứng dụng cửa hàng lưu niệm trực tuyến sử dụng ASP.NET microservices và Flutter. Ứng dụng sẽ phục vụ như một cửa hàng trực tuyến bán các sản phẩm vật lý đa dạng như áo phông và cốc cà phê, mang lại trải nghiệm quen thuộc cho những người đã từng mua sắm trực tuyến.

Quá trình phát triển sẽ sử dụng kiến trúc microservices với ASP.NET cho API backend, và Flutter cho ứng dụng di động frontend. Phương pháp này cho phép một hệ thống có thể mở rộng và dễ bảo dưỡng, đồng thời cung cấp giao diện phong phú, thân thiện với người dùng.

Phạm vi của dự án tập trung vào việc tạo ra một ứng dụng nhỏ đáp ứng các yêu cầu tối thiểu của một ứng dụng e-shop. Điều này bao gồm các tính năng như liệt kê sản phẩm, chức năng giỏ hàng, và xử lý đơn hàng.

Kết quả mong đợi của dự án này bao gồm việc hiểu sâu hơn về kiến trúc microservices, phát triển ứng dụng sử dụng Flutter và ASP.NET, và các khía cạnh kinh doanh của việc chạy một e-shop. Dự án phục vụ như một ứng dụng thực tế của các công nghệ và khái niệm này, minh họa hiệu quả của chúng trong một tình huống thực tế.

Chapter 1

Introduction

1.1 Background

The advent of information technology has brought about significant changes in various sectors, including business operations. Businesses, in their quest to enhance management quality and increase revenue, are increasingly integrating software applications into their sales operations. However, the task of selecting a suitable software application and platform can be daunting, especially for small and medium-sized enterprises (SMEs).

One of the traditional models of software design is the monolithic architecture. This model is built as a unified unit, self-contained, and independent from other applications. It can be convenient early on in a project's life due to its ease of code management, cognitive overhead, and deployment. However, as the size of the application increases, so does its start-up and deployment time. For instance, an e-commerce platform like Amazon started with a monolithic architecture but as it grew, the architecture became more complex and harder to manage.

To address these challenges, businesses are turning to microservices architecture. Microservices architecture is a software design approach that breaks down an application into a collection of small, independent services. Each microservice has its own business logic and database, and it can be deployed and scaled independently. This approach offers several advantages such as improved scalability, faster deployment times, and better fault isolation.

For example, Netflix transitioned from a monolithic architecture to a microservices architecture to handle their growing user base and the need for rapid innovation. By breaking down their application into smaller services (like one for recommendations, one for user profiles etc.), they were able to scale their services independently based on demand and deploy updates more frequently without disrupting the entire system.

Souvenir e-shops can use microservices architecture by breaking their e-shop application into smaller services. They could have separate services for managing products, inventory, orders, and payments. This would make it easier to add new features, scale their e-shop as needed, and make their system more reliable.

1.2 Problem statement

E-commerce enterprises, particularly SMEs, face the challenge of selecting scalable software solutions that can accommodate their growth trajectory. While monolithic architectures initially offer simplicity in code management and deployment, they become increasingly complex and unwieldy as applications expand.

To address these challenges, there is a need for a new approach: microservices architecture. This allows for improved scalability, faster deployment times, better fault isolation, and more flexibility in implementing new features.

The objective of this project is not to transition an existing e-shop from a monolithic architecture to a microservices architecture but to implement a new e-shop using microservices architecture from the ground up. The e-shop will have minimal features due to the project's short deadline.

By adopting microservices architecture from the start, the souvenir e-shop aims to develop and deploy new features more easily, scale as needed, and improve the overall reliability of the system.

1.3 Research Objectives

The primary aim of this project is to design and implement a microservices-based e-commerce application for a souvenir e-shop, along with a front-end application using Flutter. The specific objectives are as follows:

- **Literature Review:** Conduct a comprehensive review of existing literature on microservices architecture, its benefits, challenges, and best practices. This will provide a theoretical foundation for the project.
- **Design:** Design a microservices-based e-commerce application model. The design should consider factors such as scalability, fault tolerance, and ease of adding new features. In parallel, design a front-end application using Flutter following Clean Architecture principles.
- **Development:** Develop a minimum viable product (MVP) of an e-commerce application for a souvenir shop using the proposed model. The MVP will include key features such as product management, inventory management, order processing, and payment processing.
- **Documentation:** Document the entire process, including the design decisions made, challenges encountered, solutions implemented, and lessons learned. This will serve as a valuable resource for future projects of similar nature.

1.4 Research Scope

The scope of this research is focused on the design and implementation of a microservices-based e-commerce application for a souvenir e-shop, along with a front-end application using Flutter.

The specific areas covered in this research include:

- **Microservices Architecture:** The research will delve into the principles and practices of microservices architecture. It will explore how to design and implement an e-commerce application using this architecture.
- **Front-end Development with Flutter:** The research will also cover the design and implementation of a front-end application using Flutter. It will follow the principles of Clean Architecture.
- **Key Features:** The research will focus on implementing key features for an e-commerce application, such as product management, inventory management, order processing, and payment processing.
- **Evaluation:** The research will include an evaluation of the implemented system in terms of functionality, performance, scalability, and reliability.

Please note that while the research aims to cover these areas, it is limited by the project's short deadline. Therefore, the e-shop application developed as part of this project will be a minimum viable product (MVP) with basic features.

1.5 Solution approach

The approach to solving the problem at hand involves several steps, each designed to ensure the successful implementation of a microservices-based e-commerce application for a souvenir e-shop, along with a front-end application using Flutter. The steps are as follows:

- **Literature Review:** he first step involves conducting a comprehensive review of existing literature on microservices architecture and front-end development using Flutter. This will provide a theoretical foundation for the project and inform the design and implementation stages.
- **Design:** The next step is to design the microservices-based e-commerce application model and the front-end application using Flutter. The design will take into account factors such as scalability, fault tolerance, and ease of adding new features.
- **Development:** Once the design is complete, the development of the minimum viable product (MVP) begins. This involves coding the back-end services and front-end application, and setting up the necessary databases and interfaces.
- **Testing: :** After the MVP is developed, it will be thoroughly tested to ensure it functions as expected. This includes unit testing, integration testing, and system testing.
- **Evaluation:** The MVP will then be evaluated in terms of its functionality, performance, scalability, and reliability. Feedback from this evaluation will be used to identify areas for improvement.

1.6 Report structure

Chapter 1. Introduction: Introduction to the general overview of the , also the current chapter. In this chapter, it will give a general overview of the topic, its potential and practical applications in the future.

Chapter 2. Literature review This section provides a summary of relevant research on the topic and the foundational knowledge necessary to develop a microservices model using ASP.NET and build an MVP application using the Clean Architecture model in Flutter.

Chapter 3. System design and implementation This section presents the approach to the problem. It discusses the details of the implementation approach, including the tools used.

Chapter 4. Testing and evaluation. This chapter presents the testing plan and management, testing scenarios for the main functions of the system.

Chapter 5. Conclusion.This section presents the results achieved, the remaining limitations, and the system's further development.

Chapter 2

Literature reiview

The aim of this literature review is to provide a comprehensive understanding of the key concepts and technologies that underpin this project.

These include microservices architecture and front-end development using Flutter. This review will serve as the theoretical foundation for the design and implementation of a microservices-based e-commerce application for a souvenir e-shop, along with a front-end application using Flutter. In the next section, we will delve deeper into microservices architecture, starting with its definition and core principles.

2.1 Microservices Architecture

2.1.1 Overview of Microservices Architecture

Microservices architecture is a design approach where an application is built as a collection of small services. Each of these services runs in its own process and communicates with others using lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and are independently deployable by fully automated deployment machinery.

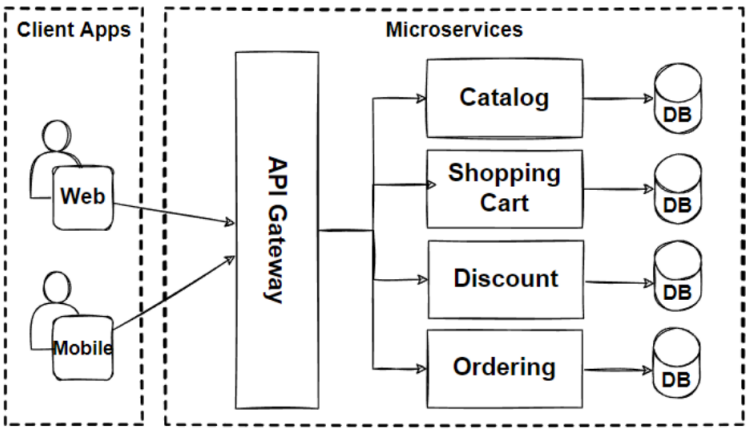

Figure 2.1: Microservices

For instance, consider an e-commerce application. In a monolithic architecture, all functionalities like user management, product catalog, order processing, and payment system would be in a single codebase. In contrast, in a microservices architecture, each of these functionalities would be a separate service with its own database and business logic.

2.1.2 Benefits of Microservices:

The benefits of using microservices architecture in e-commerce applications are numerous. They include improved scalability as each service can be scaled independently based on demand, increased resilience as the failure of a single service does not affect the entire system, and enhanced agility and speed in bringing new features to market.

Amazon is a prime example of the benefits of microservices. Originally, Amazon had a monolithic architecture, but as their services expanded, they moved to a microservices architecture. This allowed them to scale individual services based on demand. For example, during a sale, the order processing service could be scaled up without affecting other services.

2.1.3 Challenges of Microservices:

Despite its benefits, microservices architecture presents several challenges:

- **Complexity:** Managing multiple services can be complex, requiring sophisticated coordination and orchestration.
- **Distributed Data Management:** Handling distributed data across services can be difficult and requires careful consistency management.
- **Communication Overhead:** Implementing communication between services can introduce additional latency and resource usage.

Netflix faced challenges with microservices when they first transitioned from a monolithic architecture. They had to deal with complexities in managing multiple services and ensuring data consistency across services. They also faced challenges in implementing communication between services.

2.2 Microservices in ASP.NET

In this project, a new souvenir e-shop will be implemented using a microservices architecture from the ground up with ASP.NET. Each feature such as product management, inventory management, order processing, and payment processing will be implemented as a separate microservice running in its own Docker container.

2.2.1 ASP.NET and Microservices

ASP.NET is a robust framework for .NET that simplifies the creation of APIs, which form the backbone of microservices. It provides built-in support for Docker containers,

which are lightweight and can be easily managed, making them ideal for deploying microservices.

ASP.NET offers high performance, which is critical in a microservices architecture where each service should be highly responsive. The high throughput of .NET surpasses many popular frameworks, making it an excellent choice for building efficient microservices.

2.2.2 Communication Between Services

In a microservices architecture built with ASP.NET, secure communication between services is crucial. Technologies like HTTPS provide secure communication over a computer network, while RabbitMQ with MassTransit in ASP.NET offers a reliable and efficient message broker system, which is essential for communication between microservices.

2.2.3 Docker and .NET Tools

Docker has become a standard in the container industry due to its ability to encapsulate microservices in containers, enhancing their isolation, portability, and scalability. .NET is built to work seamlessly with Docker, allowing developers to focus on building their microservices without worrying about compatibility issues.

The Visual Studio family of products offers comprehensive tools for developing ASP.NET applications with Docker support on Windows. These tools simplify the process of configuring applications for Docker and allow developers to step through their code line-by-line as it runs in a Docker container.

2.3 Introduction to Flutter and Clean Architecture

2.3.1 Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase. Key features of Flutter include its fast development cycle, expressive and flexible UI, and native performance.

2.3.2 Clean Architecture

Clean Architecture is a software design philosophy that separates software into layers with ‘stable dependencies’. This architecture promotes the separation of concerns, making the system easier to manage and maintain. It is important because it makes the system more flexible, maintainable, and scalable.

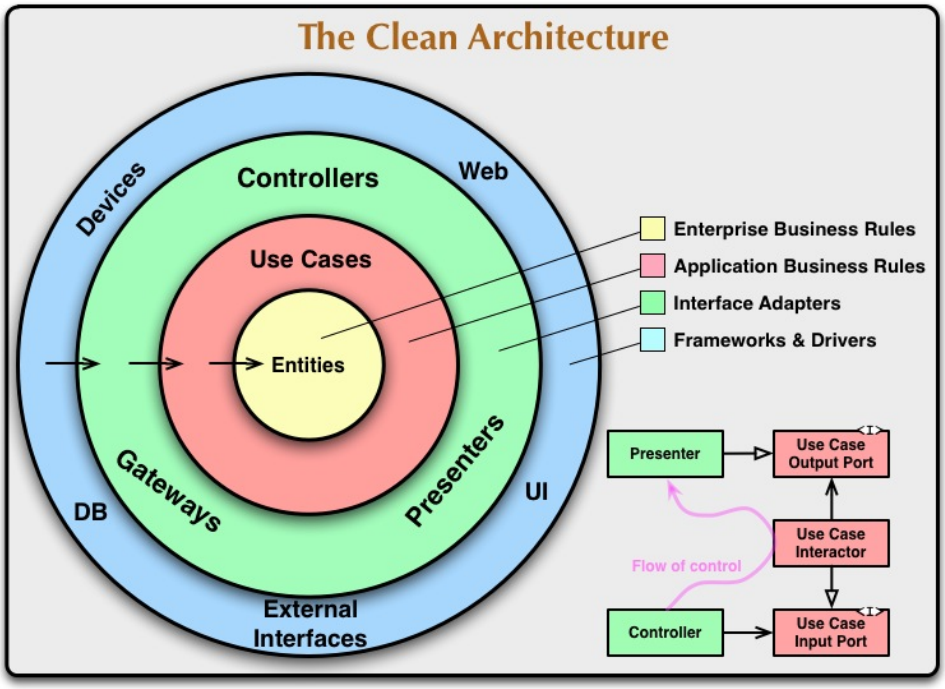

Figure 2.2: Clean architecture by Robert C. Martin [**?**]

2.3.3 Applying Clean Architecture in Flutter

Clean Architecture can be applied in Flutter by separating the application into three layers: Presentation, Domain, and Data. The Presentation layer handles the UI and user interactions. The Domain layer contains business logic and use cases. The Data layer manages data from various sources like network requests and databases.

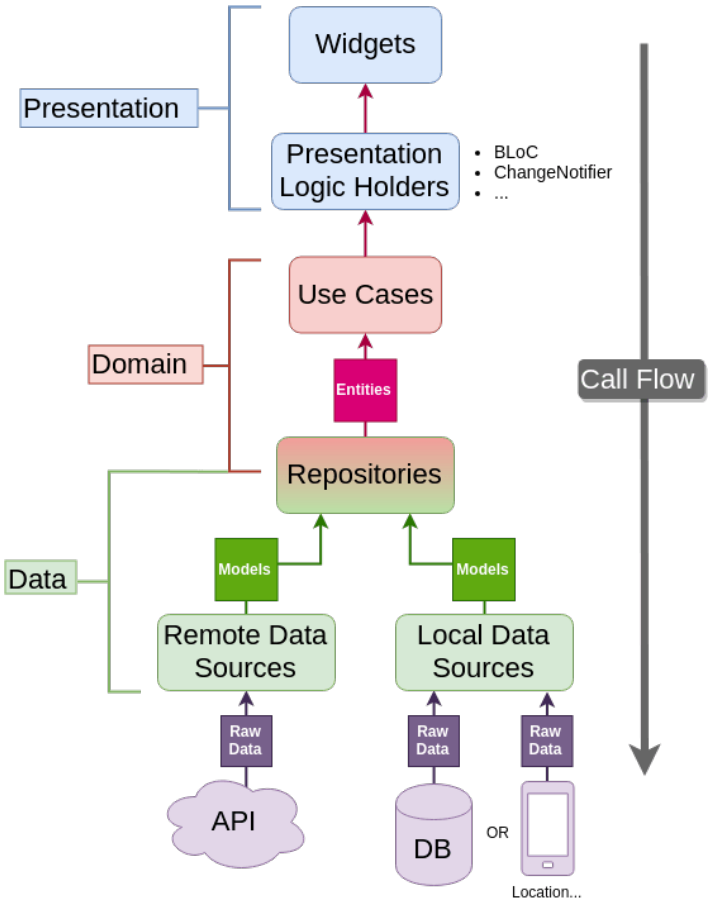

Figure 2.3: Clean Architecture Diagram by Reso Coder[?]

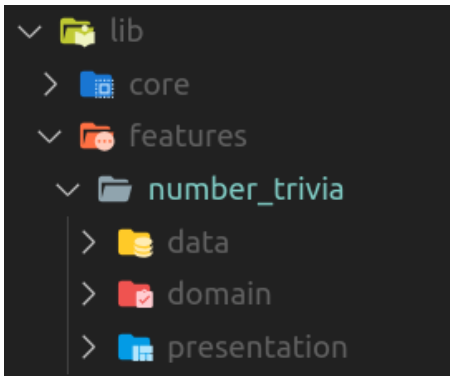

Figure 2.4: Example of structure in Flutter project

2.4 Applications of Flutter and Clean Architecture

2.4.1 Accelerated MVP Development with Flutter

Flutter's ability to deliver high-performance applications on Android and iOS from a single codebase makes it an excellent choice for rapid MVP development. Its rich set of widgets and reactive framework allow for a fast development cycle, enabling developers to quickly implement key features and bring the product to market.

2.4.2 Case Studies of Flutter and Clean Architecture

There are several case studies of applications developed using both Flutter and Clean Architecture. These applications demonstrate how Clean Architecture can enhance the maintainability and scalability of Flutter applications, even when under the constraints of rapid MVP development.

Flutter has been used in various real-world applications due to its ability to deliver high-performance applications on Android and iOS from a single codebase. Some examples include the Alibaba e-commerce app, the Toyota: Improving Infotainment Systems app,

and Google Ads.

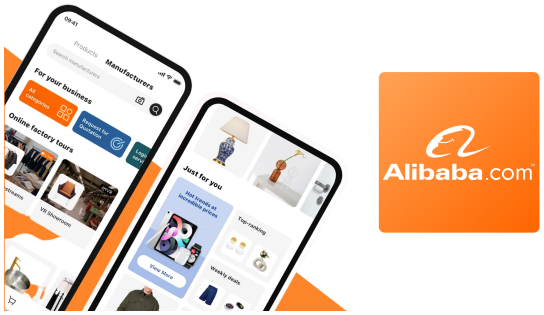

Figure 2.5: Alibaba Group: Ecommerce

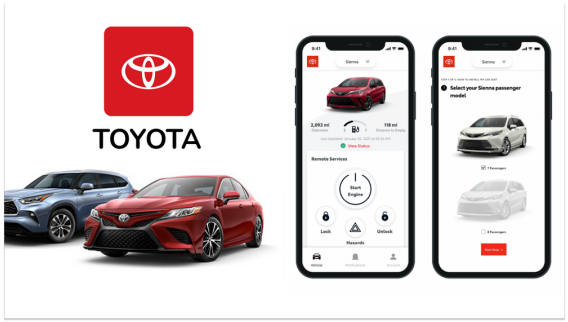

Figure 2.6: Toyota: Improving Infotainment Systems

2.4.3 Impact of Flutter and Clean Architecture

The use of Flutter along with Clean Architecture has been shown to improve the development process by making it easier to manage codebases, implement new features, and fix bugs. It also enhances the performance of applications by ensuring they run smoothly on different platforms.

2.5 Microservices, Flutter, and E-commerce

2.5.1 Application of Microservices and Flutter in E-commerce

Microservices architecture and Flutter can be effectively applied in the context of e-commerce. Microservices allow for the separation of functionalities into independent services such as user authentication, product catalog, order management, and payment processing1. Flutter, with its fast development cycle and expressive UI, can be used to build the frontend of the e-commerce application.

2.5.2 Benefits and Challenges

The benefits of using these technologies in e-commerce include improved scalability, faster deployment times, better fault isolation, and more flexibility in implementing new features. However, challenges may include increased dependency on technology, large costs involved especially for small businesses, risk of job cuts due to automation, security risks related to data and fraud, and dealing with issues like shopping cart abandonment and maintaining customer loyalty.

2.5.3 Case Studies

Several e-commerce giants like eBay, Etsy, Gilt and Zalando have transformed their infrastructures into microservice architectures to create flexible, global systems. The combination of these technologies holds promise for efficient and scalable e-commerce application development.

Chapter 3

System design and implementation

3.1 Section 1

Chapter 4

Testing and evaluation.

4.1 Section 1

Chapter 5

Conclusion

5.1 Section 1

Bibliography

[1] Reso Coder. Flutter tdd clean architecture - explanation and project structure, 2019.

[2] Robert C. Martin. The clean architecture, 2012. Accessed: 2023-10-24.