Version 0.1

**Sep 01, 2021**

Online Bookstore

Technical Proposal

**Author:** Dao Minh Hieu

**Security Classification:**

110 Bishopsgate, London EC2N 4AY

Tel: +44 (0)20 7333 0033

Email: [info@nashtechglobal.com](mailto:info@nashtechglobal.com)



Table of contents

[1. Overview 2](#_Toc86528144)

[**1.1** Purpose of the technical proposal 2](#_Toc86528145)

[**1.2** Business Objectives 2](#_Toc86528146)

[**1.3** Assumptions and Scopes 2](#_Toc86528147)

[1.3.1 Assumptions 2](#_Toc86528148)

[1.3.2 In scope 2](#_Toc86528149)

[**1.4** Constraints 2](#_Toc86528150)

[2. Non-functional requirements 3](#_Toc86528151)

[**2.1** Exception Handling 3](#_Toc86528152)

[**2.2** Availability 3](#_Toc86528153)

[**2.3** Security 3](#_Toc86528154)

[3. Architectural Goals 4](#_Toc86528155)

[**3.1** High-Level Architecture (System Overview) 4](#_Toc86528156)

[3.1.1 Client Website 5](#_Toc86528157)

[3.1.2 API Gateway 5](#_Toc86528158)

[3.1.3 Microservices 6](#_Toc86528159)

[3.1.4 Database 6](#_Toc86528160)

[**3.2** Authentication with ASP.NET Identity for Microservices 7](#_Toc86528161)

[**3.3** Relational Data Model 7](#_Toc86528162)

[3.3.1 Product 7](#_Toc86528163)

[3.3.2 Order 9](#_Toc86528164)

[3.3.3 Customer 10](#_Toc86528165)

[**3.4** Continuous Integration and Continuous Delivery 10](#_Toc86528166)

[4. Technology Selection 10](#_Toc86528167)

[**4.1** Advantages of React in This Solution 11](#_Toc86528168)

[**4.2** Advantages of .Net Core in This Solution 11](#_Toc86528169)

[5. Deployment Model 12](#_Toc86528170)

[ Development Environment (on promise and cloud) 12](#_Toc86528171)

[ Staging / Production Environment 12](#_Toc86528172)

# TABLE OF FIGURES

[Figure 1 OAuth2 Protocol Flow 5](#_Toc86528208)

[Figure 2 High-Level Architecture 5](#_Toc86528209)

[Figure 3 Front-end Folders and Files 6](#_Toc86528210)

[Figure 4 Microservices 7](#_Toc86528211)

[Figure 5 Data tables in SQL SERVER 7](#_Toc86528212)

[Figure 6 Product Data Model 10](#_Toc86528213)

[Figure 7 Order Data Model 10](#_Toc86528214)

[Figure 8 Customer Data Model 11](#_Toc86528215)

[Figure 9 CI/CD Process 11](#_Toc86528216)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document Version History | | | | | |
| **Version** | **Effective Date** | **Author** | **Details** | **Reviewer** | **Approvers** |
| 0.1 | Sep 1st 2021 | Dao Minh Hieu | Created | All members | Le Cao |

Confidentiality

This document is distributed on a restricted basis, is commercial in confidence to the recipient, and may not be used for any purpose other than that associated with a NashTech project. The contents of this document may not be disclosed to any third parties without the expressed advance written authorisation of NashTech.

# Overview

## Purpose of the technical proposal

This document describes the proposed architecture (i.e., High-Level Design) for the project “Online Bookstore” and how various modules work in cohesion with each other.

The current solution utilizes all the benefits of React technologies, ASP.NET Core 3.1 and is deployed by cooperating with the CI/CD team to bring the site living on the internet by using Docker and Kubernetes.

## Business Objectives

Our React group aim to develop an online bookstore in which the user interface is built mostly by a common JavaScript library React and the backend service is built by .NET technologies such as ASP.NET Core, C# language, ASP.NET Identity as requirements from our LM. Le Cao. Moreover, the architecture must follow a microservice architecture.

## Assumptions and Scopes

### Assumptions

* There are no members in teams having ideas deeply about React before. However, they do have much knowledge of .NET technologies.
* All members know how to develop a software program following the microservice architecture.
* Isolate knowledge about Docker and Kubernetes for CI/CD and deployment.
* Have a strong background in dealing with SQL Server and SQL language.
* Know how to use Git and other GIT UI to support version control to work with other team members.

### In scope

The scope of the project is:

* Build a client website that has the user interface built in React library.
* Build microservice backend including some basic features like authentication, authorization, other online bookstore features such as CRUD operations, etc.
* Configure variables exposing hosts and ports and other things that are required from CI/CD teams to communicate among services in the systems.
* Set up Azure Storage Account, Blob to store static book images to display in the user interface.

## Constraints

* UI: developed by React.
* Backend:
  + ASP.NET Identity.
  + Based on microservices architecture.
  + ASP.NET Core 3.1.
  + SQL Server.
  + Azure Storage Account – Blob.
  + Deployed using Docker and Kubernetes.

# Non-functional requirements

Describe some main non-function requirements and how the solution has covered each of the items

## Exception Handling

A well-designed application handles exceptions and errors to prevent application crashes. The solution puts forward the following best practices for exception handling following patterns recommended by Microsoft:

* Using Error Boundaries provided by React.
* Use try/catch/finally blocks
* Handle common conditions without throwing exceptions. For conditions that are likely to occur but might trigger an exception, consider handling them in a way that will avoid the exception. Use exception handling if the event doesn't occur very often, that is, if the event is truly exceptional and indicates an error (such as an unexpected end-of-file). When you use exception handling, less code is executed in normal conditions.
* Check for error conditions in code if the event happens routinely and could be considered part of normal execution. When you check for common error conditions, less code is executed because you avoid exceptions.
* Design classes so that exceptions can be avoided.
* Throw exceptions instead of returning an error code. Exceptions ensure that failures do not go unnoticed because the calling code didn't check a return code.

## Availability

Microservice architectures ensure failure isolation. Techniques such as health checking, caching, bulkheads, or circuit breakers would allow a reduction of the blast radius of a failing component and improve the overall availability of a given application.

## Security

* + Token-based authentication

The resources at the data centre are protected by default and following the [OAuth 2.0](http://oauth.net/2/) specification. To access the resources from the data centre, the user must have a token. To acquire a token, the user will input their credential only one time and then the server will create a token.

Every request to the server will include the token in the header of the HTTP request. The server will then validate the token and will respond to the resource accordingly. Below is the basic protocol flow to secure resources following OAuth 2.0:

A screenshot of a social media post

Description automatically generated

Figure 1 OAuth2 Protocol Flow

# Architectural Goals

## High-Level Architecture (System Overview)

The system has a simple client site built by React that will connect to the API Gateway to communicate with the backend services. The high-level architecture is depicted by the following diagram:

Graphical user interface

Description automatically generated with medium confidence

Figure 2 High-Level Architecture

### Client Website

* This client website is structured by HTML, CSS, Typescript and the required JavaScript library, ReactJS. That is provided for end-user to interact with the under system including various features of a bookstore online. They can search for their interesting products, see book details or event do ordering.
* The front end is integrated with Semantic React UI that helps us to build responsive pages that make users feel more comfortable with the UX.
* The following picture shows how we organized folders and files to create beautiful web pages by React:

Graphical user interface, application

Description automatically generated

Figure 3 Front-end Folders and Files

### API Gateway

In the microservice architecture pattern, regarding communication of the clients with the microservices, we usually try to avoid having direct client-to-microservice communication. That has some drawbacks such as the mismatch between the needs of the client and the fine-grained APIs exposed by each of the microservices or web-unfriendly protocols. That is why we have a much better approach is to use an API Gateway that is a server that is the single-entry point into the system. In our system, we use Ocelot API Gateway as a gateway in our design. API Gateway will help us to abstract the under system and make it both developers and end-users easily to develop, use the system.

### Microservices

* There are a bunch of microservices in our system including Admin, product, ordering, comment services that have their tasks and roles in the system. If there is a request coming from the web browser, the API gateway act as a receiver to receive the request first and then it will look for the corresponding service to query the data and send it back to the web to render the result to the end-user.

Text

Description automatically generated with medium confidence

Figure 4 Microservices

### Database

* We use SQL SERVER to manage our database aiming for querying and retrieving data.
* In the database, we also include tables for saving user credentials like username, hashed password, and other information to authenticate them before allowing them to use features in the system.

Table

Description automatically generated

Figure 5 Data tables in SQL SERVER

## Authentication with ASP.NET Identity for Microservices

Authentication is the process of reliably ascertaining a user’s identity. In microservice scenarios, authentication is typically handled centrally and make sure that the individual microservices cannot be reached directly (without the API Gateway)

A picture containing diagram

Description automatically generated

## Relational Data Model

### Product

Graphical user interface, application

Description automatically generated Graphical user interface, application

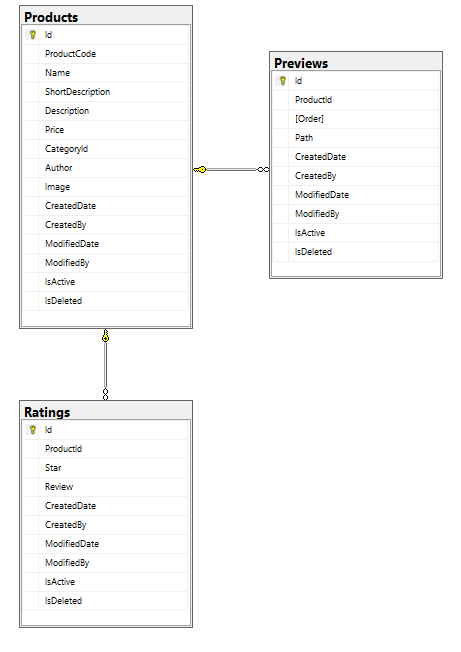
Description automatically generated 

Figure 6 Product Data Model

### Order

Diagram

Description automatically generated

Figure 7 Order Data Model

### Customer

A picture containing graphical user interface

Description automatically generated

Figure 8 Customer Data Model

## Continuous Integration and Continuous Delivery

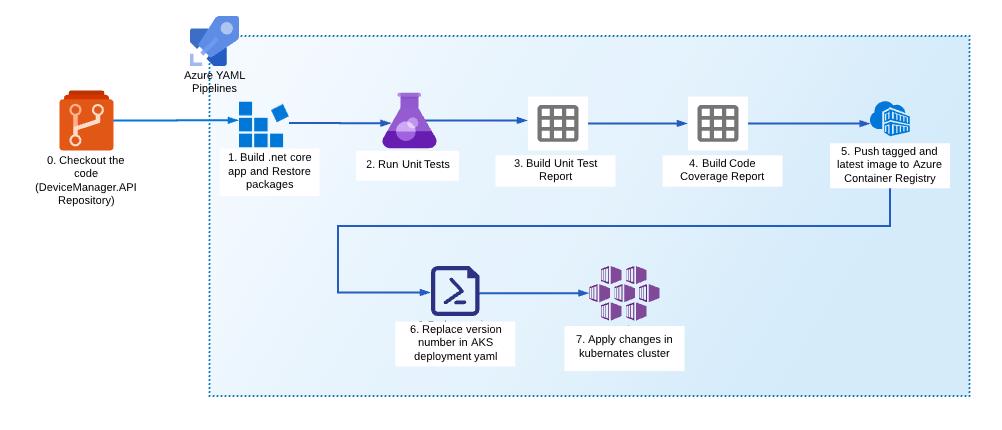


Figure 9 CI/CD Process

In this process, the CI/CD team is responsible for checking out the code hosting in the GITHUB repository and then setting up a pipeline that can build a .NET Core app and restore packages. Then it will run unites and ensure everything in green. After that, it will push tagged and latest image to Azure Container Registry replacing version number in AKS deployment YAML. Lastly, all changes are applied to the Kubernetes cluster.

# Technology Selection

The following section examines the reasons behind the selection of technologies put forward in the proposal.

## Advantages of React in This Solution

React is used in the development of front-end UI functionality, and the advantages of using React are as follows:

* The Single Page Application pattern delivered using React JS delivers a fast, feature-rich user experience and is simple.
* Single-way data flow: in React, we cannot directly modify any properties passed from the outer components so we can easily understand the data flow.
* We easily adapt to learn React.js because it is suggested to use JSX in each react component to render an element. JSX is an extension of JavaScript, so the experience JavaScript developer easily writes it, reads it and maintains it.
* The Virtual Dom architecture used by React delivers responsive applications out of the box.
* React’s component-based architecture allow developers to break down complex UI into simple, reusable components. This also makes it easier to maintain and develop which reduces both the initial cost to build applications in React as well as to maintain them.
* Performance: Angular vs React (http://blog.thinkwik.com/sturdy-faceoff-angular-reactjs/) assessment reveals that ReactJS yields better performance.

## Advantages of .Net Core in This Solution

**Cross-Platform**

.NET Core is a free and open-source framework for developing cross-platform applications targeting Windows, Linux and macOS.

**Microservices architecture**

A microservices architecture allows a mix of technologies across a service boundary. This technology mix enables a gradual embrace of .NET Core for new microservices that work with other microservices or services.

**Performance**

.Net Core is the most performant option for Microservice development. This has been shown in several studies e.g. [Choose between .NET and .NET Framework for server apps | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server)

**Ease of maintenance**

Standardising on .Net Core allows us to minimise the number of languages in the solution which in turn makes it easy to support. It is a widely used technology that also makes it easy for us to research and find out solutions for the problems facing in the coding process.

**Longevity**

Microsoft sees this as it’s the language of choice for the future. The long roadmap and steady investment by Microsoft along with a vibrant open source community around the technology means will result in the possibility of Kaplan taking advantage of these developments over the lifecycle of the product.

# Deployment Model

Describe the deployment model for production, Development environments. Also if CI/CD is applicable then mention it here to bring benefits to the development team and client as well

Diagram, timeline

Description automatically generated with medium confidence

### Development Environment (on promise and cloud)

The team will develop their local environment by interacting with the GitHub repository to commit or pull code and then make modifications to the source code.

### Staging / Production Environment

The staging environment is where users can test the features that are deployed from the development environment. The staging environment is hosted by CI/CD servers.