telliant
Intelligent Software. Delivered.

Microsoft Partner
Gold Application Development
Gold Collaboration and Content

SYSTEM ARCHITECTURE AND DESIGN STRIVE INTERNAL WEB APPLICATION

Abstract

This document is to provide Architecture and the building block details of the Mammoth internal web application at High Level.

Author	Reviewed By	Approved By
Naveen Kumar Dhasarathan	Amos	
8/16/2019	8/29/2019	
Naveen Kumar Dhasarathan	Seth Narayanan	Seth Narayanan
09/10/2019	09/10/2019	09/12/2019

Table of Contents

Introduction

Mammoth Detail Salons specializes in hand car washing and detailing. At Mammoth hand car wash and detail there are many services available like Mini Wash, Mammoth Wash, Ultra – Mammoth Wash, Mega-Mammoth Wash and membership services. On the technical side, it has Web Applications and Mobile applications to maintain the daily business seamlessly. Mammoth is building a point of sale software product called Strive with the following web applications.

Web Applications:

1. Mammoth Internal Web Application

The purpose of this document is to provide High-level design details to create Strive web application from the scratch using latest technology.

Project Purpose

The Strive internal web application should maintain details about wash services, detail services, sales related information, reports, admin functionalities like register setup, employee information, vehicles, gift cards etc.

The project architecture will be built to support car wash businesses with a future expansion into multiple lines of business such as nail salons, hair salons, and restaurants.

The scope of this document is to describe the solution and the high-level design of the Strive internal web application. This document is meant for software architects, developers, designers and for the reference of other stakeholders.

Executive Summary

The existing applications of Mammoth Detail Salon which maintain the day to day operations of the car wash business are written in outdated technologies.

This document proposes a new software architecture and a new technology stack for the mobile application platform. This new platform will provide the following benefits:

1. improved maintainability
2. enhanced scalability
3. extensive security
4. robustness
5. extensibility

Goals and Objectives

The goal is to design an easily scalable and maintainable web application as referenced in the FRS documents for Strive with the design considerations as specified in this document. <See Design Considerations section>

Current system overview

Currently the Mammoth internal web application uses two web applications:

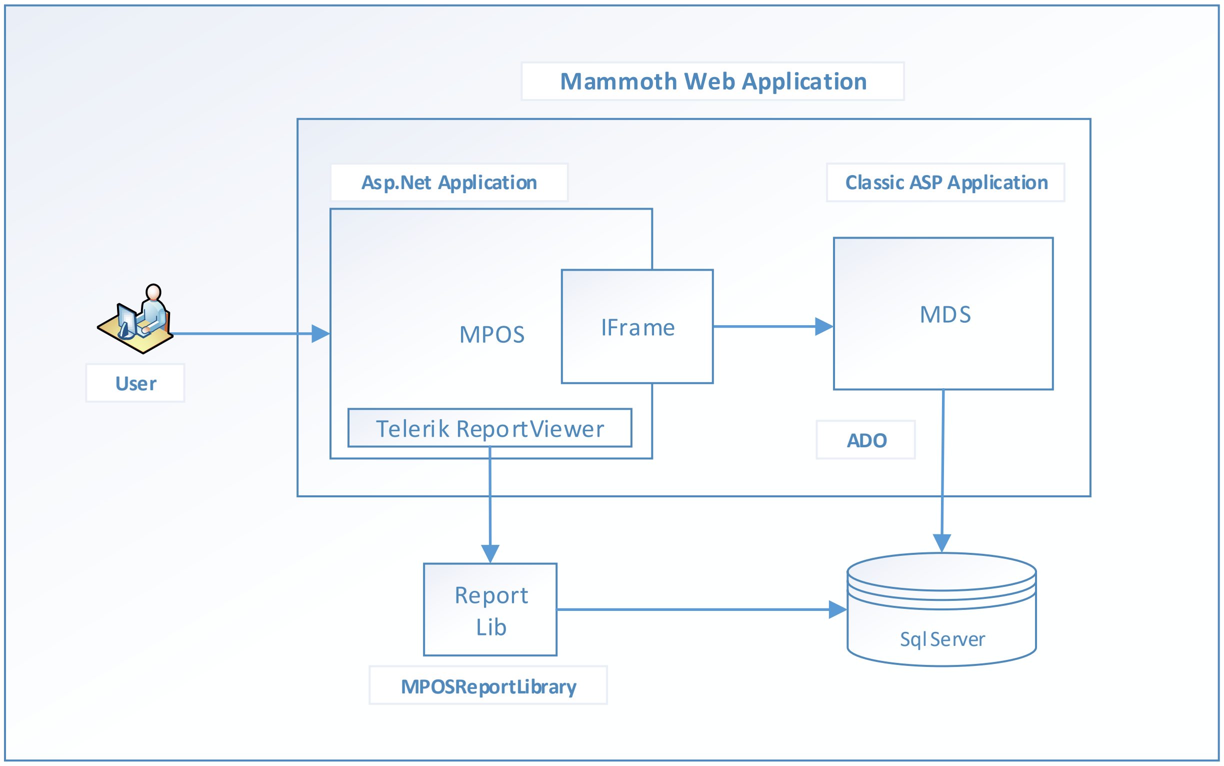
1. MPOS (Asp.Net 4.0)
2. MDS (Classic ASP)

All the user requests will be received by MPOS and the relevant ASP page will be rendered via iframe from the MDS application.

The current application employs a MSSQL database. ADO and ADO.Net have been used to connect the database from MDS and MPOS respectively. Telerik report viewer controls are used to display reports.

Current workflow

The below diagram represents the current flow of the current Mammoth internal web application.



Proposed Architecture

In order to create a robust web application with loosely coupled tiers, the following technical stack and workflow is being proposed.

The application will consist of two projects.

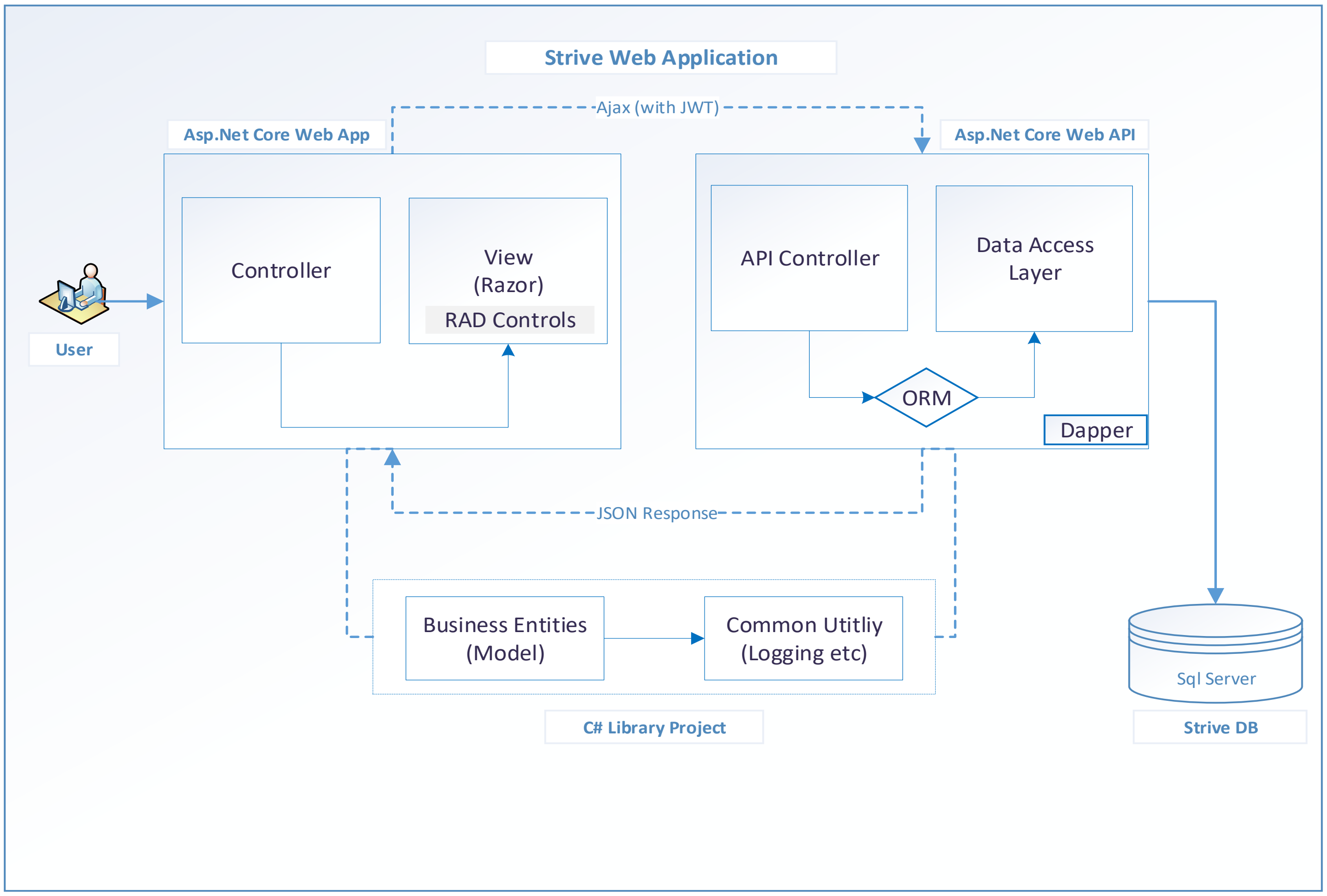
1. Strive Web App
2. Strive Web API

Strive Web App will be the presentation layer of the application. Strive Web API will contain the business logic and data access layer. Every call from the presentation layer to API will be an asynchronous call.

Dapper will be used for object relational mapping (ORM) and AutoMapper will be used to map entities. All the common utility functions and common entities will be maintained in separate projects to avoid code repetition. JWT (JSON Web Token) and OAuth will be used for authentication and authorization.

Proposed workflow

The following diagram represents the proposed workflow of the application



Assumptions

1. The proposed application architecture is high level, minor changes during the design phase are likely, though the overall concept will remain the same.

2. A technical design document will be derived from this document. It will include all the components required with specific details of the actual implementation.

3. The third-party commercial tools are listed in the section "3rd party Plugins". Additional third-party commercial tools may be included during the design phase, but the recommendation would be to utilize open source technologies.

4. All the plugins mentioned in section “3rd Party Plugins” need to be validated by doing Proof of Concept (POC) to make sure that it can handle all technical/functional requirements of the applications.

5. All the plugins mentioned in the section “3rd Party Plugins” require validation by creating a Proof of Concept (POC) to ensure that the application can function as expected per the functional requirements specification document.

6. The database will be designed as needed.

7. OAuth2.0 and JWT (Json Web Token) will be used for authentication and authorization.

8. Dapper and AutoMapper plugins will be used as required.

Risks

1. Existing Telerik controls feasibility with new Asp.Net core technology must be verified. If it is not feasible, other commercial tools will be considered.
2. As Genbook doesn’t expose any API’s, retrieving scheduled appointments asynchronously from it is not possible. However, Genbook synchronizes its calendar with Google calendar (if Gmail is the primary email used to connect to Genbook). Proof of Concept is required to check the feasibility to retrieve scheduled appointments from google calendar API.

Prerequisites

The following prerequisites are essential for the deployment server

1. ASP.Net Core 2.2 runtime (Windows / Linux based on the server)
2. Telerik license
3. Internet Information Services (IIS)
4. Kestrel web server

Environments

The current design of the application has three major tiers:

1. User Interface Tier
2. Web API Tier
3. Database

The following sections explains where these tiers will be deployed during different stages of the application.

Dev

All three tiers can be deployed in a single server.

QA

QA environment should be a replica of the production environment.
Each tier should be deployed in its own dedicated server.

UAT

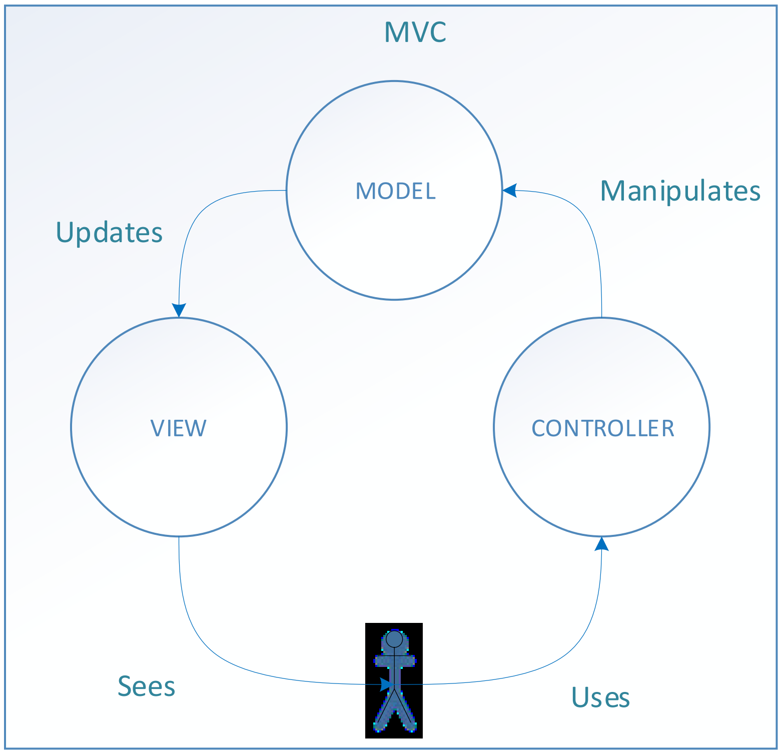
UAT environment should follow the production environment setup.

Prod

Production environment should have dedicated servers for user interface tier, web API tier, and the database. Web API servers can be expanded horizontally based on the user requests and the load balancing must be configured.

Design Pattern

The **Model View Controller** (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these are separated into different objects.



The Model contains only the pure application data, it contains no logic describing how to present the data to a user.

The View presents the model's data to the user. The view understands how to access the Model's data, but it does not know what this data means or what the user can do to manipulate it.

The Controller exists between the View and the Model. It monitors events triggered by the View (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the Model. Since the View and the Model are connected through a notification mechanism, the result of this action is then automatically reflected in the View.

Design Considerations

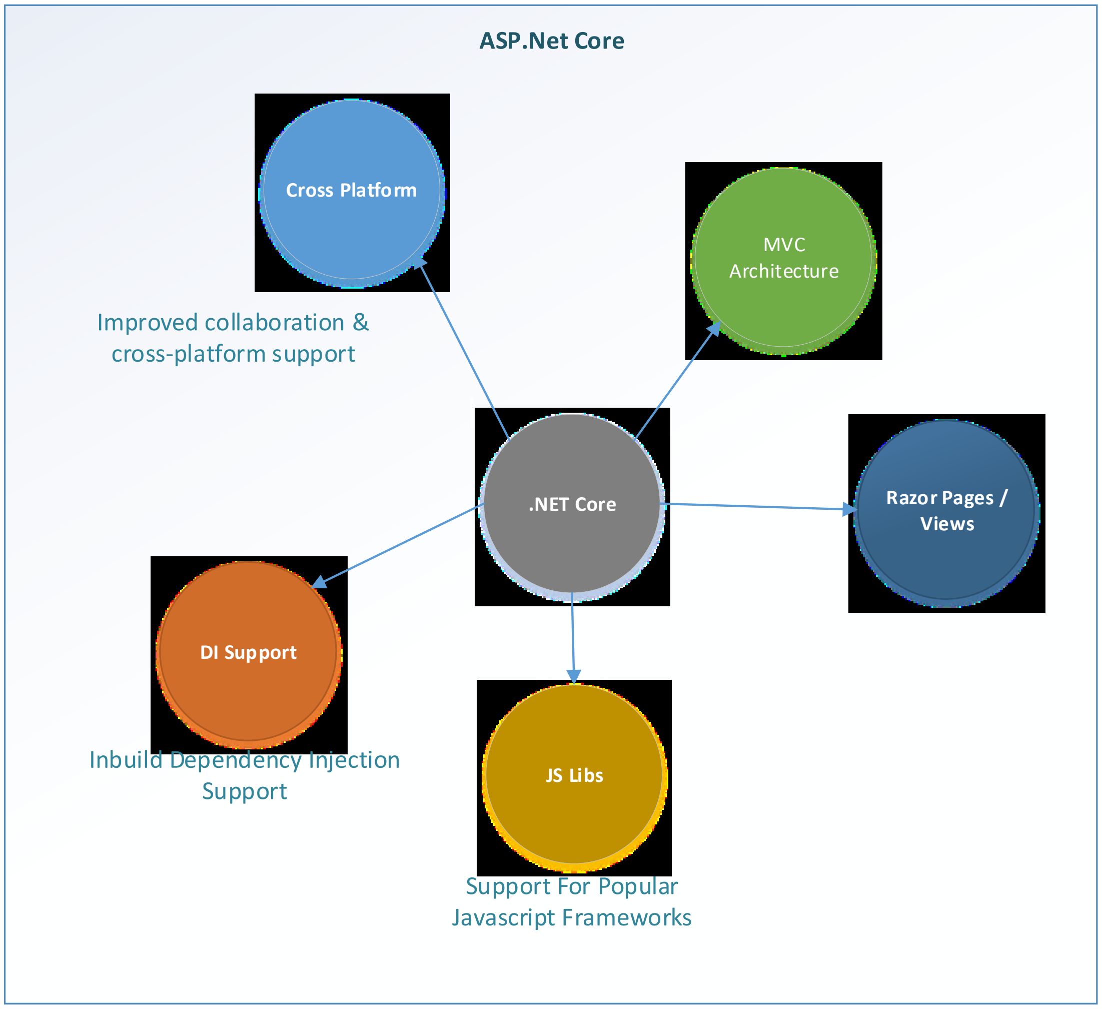
Characteristic	Solution
Correctness	Usage of Object-Oriented Programming concept, design patterns will make the application highly flexible for modifications. Inversion of control is one of the concepts that allows change which will increase extensibility.
Usability/ Knowledge	In addition to using the W3C guide for usability and knowledge, our team follows universal design standards. Telliant uses its UX/UI practice to build responsive web design to build web applications supporting multiple delivery channels/devices. It is important to design the application to support multiple devices and make them equally usable.
Reliability	In any software system or architecture, reliability is one of the most important non functional requirement. There are many factors that affect the system's reliability. We address reliability concerns by planning for minimized server downtime, failure free operations, ease of maintenance, planning for testing and staging environments, simplified & automated release process, and backups and redundancy.
Efficiency	Telliant is keenly aware of the needs of the software to perform efficiently in a cloud environment, dealing with the constraint on the speed of executing various parts of the system, storage size, data flow, ad response times. Further, Telliant ensures the following wherever applicable: 1. Compliance with OOP and SQL best practices 2. Compliance with garbage collection best practices

Security & Safety	Logical Security: pertaining to passwords, encryption, virus protection, software, e-mail, and security on the Internet. Physical Security: including facility access control, laptop and desktop security, modems, and paperwork. External Security: pertaining to security needs to be addressed outside the office to ensure that privileged client information is not shared.
Scalability and Testability	The MVC framework is designed to promote testability. Scalability ensures enhanced application performance despite load increases. Telliant assures highly functional software applications by following scalability and testing best practices.
Maintainability	Telliant ensures high code maintainability by utilizing coding standards. To ensure smooth coding with version control, we maintain the source code in advance source control applications such as Git/Bitbuckets etc. enabling the developers to perform code branching and efficient maintainenace.
Interoperability, Reusability and Portability	Telliant ensures interoperability by utilizing interfaces and architectures that are fully defined. The coding standards used by our team are always clear and unambiguous with specifications that are well maintained. Components are frequently reused wherever applicable to ensure smaller development timelines for reusability and portability.

Technology Stack

Category	Technologies/Tools	Remarks
Platform	1. ASP.NET Core 2.2x Web App (MVC), 2. ASP.NET Core 2.2x Web Api, 3. Dapper, AutoMapper 4. HTML5, 5. CSS3, 6. Bootstrap 4.3+, 7. Telerik Controls / Other UI tools (TBD),	Telerik controls will be used to display reports if it is feasible with the proposed design and architecture. Additional controls will be explored based on the proof of concept if Telerik controls are unfeasible.

	8. jQuery	A decision on license and version of the Telerik controls will be made after the feasibility evaluation.
Database	SQL Server 2016	Database will be designed as needed.
Browser	Edge 43+, Chrome 75+ , Safari 11+ on MAC	
Source Control	Azure DevOps	Azure DevOps helps teams continuously deliver software at a faster pace and with lower risk, while improving efficiency and collaboration between all teams that participate in the release processes. Azure DevOps helps teams set up continuous integration builds for the application that run with every code check in.
Editor	Visual Studio 2017, SQL Management Studio	


ASP.Net Core
Cross Platform
Improved collaboration & cross-platform support
MVC Architecture
.NET Core
Razor Pages / Views
DI Support
Inbuild Dependency Injection Support
JS Libs
Support For Popular Javascript Frameworks

Solution Overview

The following will explain the presentation, business and data layers details:

1. Strive Web App - This project will be developed using Asp.Net Core 2.2x with MVC pattern using Razor views.
2. Strive Web API - This project will be developed using Asp.Net Core 2.2x API (RESTful).
3. OAuth2.0 and JSON Web Tokens will be used for authentication and authorization.

To maintain entities and common utility functionalities, new layers can be created as required as a multi-page application.

The components within the layered architecture are organized into horizontal layers with each layer performing a specific role within the application.

The architecture promotes modularity, reusability, maintainability and extensibility while keeping the overall architecture simple.

Presentation Layer

This represents the user interface components. Presentation layer will be developed using modern tools such as bootstrap and related plugins to provide better UX. The Application will support all modern browsers.

Presentation layer will interact with the API layer to perform business actions related to user events and data access, resulting in a clean separation of the layers. As a good design principle, presentation layer will not access the database directly.

Presentation layer will be implemented using HTML with Razor syntaxes.

View

A Razor View generates an output to the user based on changes in the model. The View would be written using HTML with Razor.

Report

Telerik report or other report tools would be used across project after R&D. The version and license details of the report tool to be used will be explored and decided.

API Layer

This layer will be implemented as RESTful API. The APIs expose business functions to the rest of the world. For every request it validates data and applies the business logic before the data is saved to or retrieved from the database.

These components will define the transaction boundaries as required by the application.This layer accesses data through the service layer.

OAuth2.0 and JWT will be used to authenticate & authorized the user.

HTTP Methods

To define a set of request methods and indicate the preferred action to be performed, HTTP Methods are required. Although a group of them share some common features, each of them implements a different semantic.

The following methods will be used in the application

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

POST

The POST method is used to submit an entity to the specified resource, often causing a change in state.

API Controller

Controller implements contracts/interfaces which help external applications to easily integrate and consume services. All basic http validations are done here. All communications with other layers are done through this layer.

Json

JSON (JavaScript Object Notation) is a lightweight data-interchange format. API layer would expose data in JSON format to third party applications.

Business Layer

The Business Layer contains the business objects with their own properties and methods. Each business object has the ability to handle validation business rules.

Business object

Business object represents the domain model of the application.

Business rule

A business rule is a rule that defines or constrains some aspect of business and always resolves to either true or false. Business rules are intended to assert business structure or to control or influence the behavior of the business.

Data Access Layer

The Data Access Layer (DAL) is the layer of a system that exists between the service logic layer and the persistence / storage layer. Data access layer uses Repository patterns to read and write data into storage.

Repository

Repository pattern to separate the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model

Object Relational Mapping (ORM)

Dapper is a simple object mapper for .NET and own the title of King of Micro ORM in terms of speed and is virtually as fast as using a raw ADO.NET data reader.

Async Operations

The dotted lines between web and presentation layer represents Async operations (Non-blocking calls). The methods that take more processing time would be marked as Async.

AutoMapper

AutoMapper is used to map one object to another instead manual mapping. More can be found at http://automapper.org/.

Dependency Injection

Dependency Injection (DI) is a software design pattern that allows us to develop loosely coupled code. DI is a great way to reduce tight coupling between software components. DI also enables us to better manage future changes and other complexity in our software.

Exception

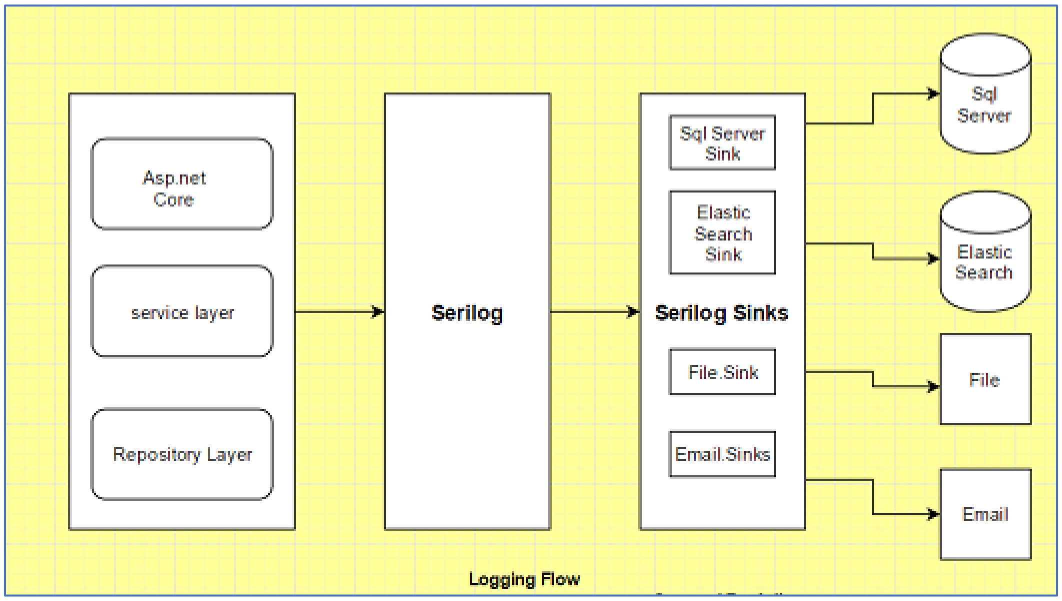
Exceptions are handled across application and logged into database table/file. The system would display a generic configurable more business friendly message to end user while logging actual exception for trouble shooting purpose.

Exceptions are handled at Service layer level, API level and Application level. All service layer exception are logged. All unhandled exceptions are logged at global (API) level. However there are few errors still cannot be handled at API level so it is handled at application level.

Logging

<u>SeriLog</u>

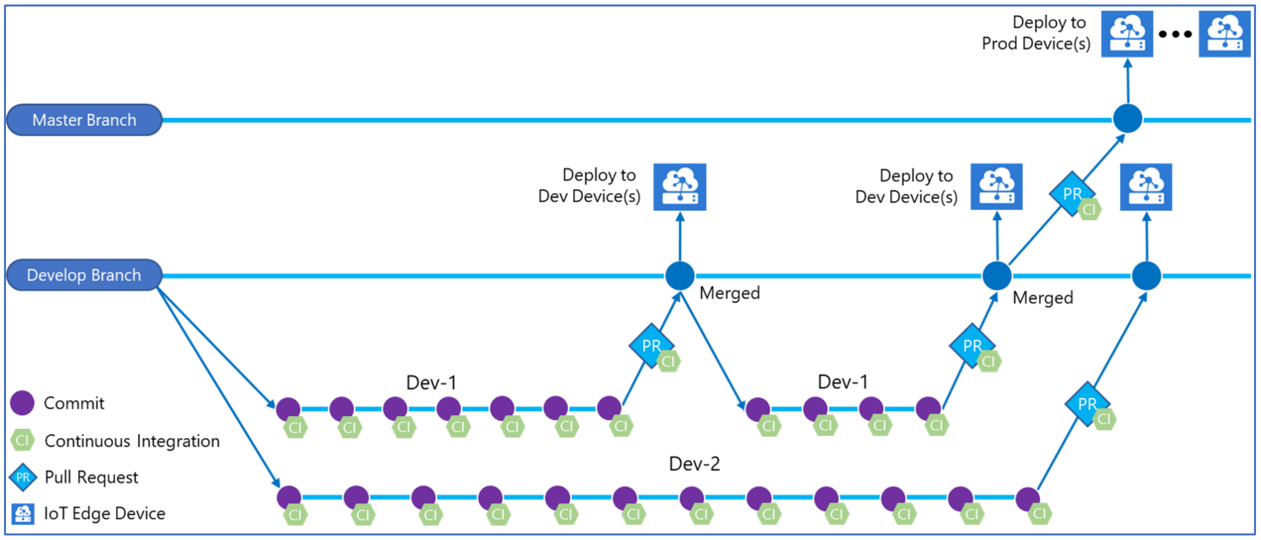
Serilog is a well-known logging framework for .NET core. It was built with structured logging in mind. It makes it easy to record custom object properties and even output logs to JSON.
Serilog uses what are called sinks to send logs to a text file, database, or log management solutions, or potentially dozens of other places, all without changing code.



Configuration Management (CI/CD)

Source Control

Azure DevOps



Azure DevOps is a cloud-based service for CI/CD pipeline provided by Microsoft. It helps you to create pipelines for continuous deployment to the server.

Azure DevOps Projects presents a simplified experience where user can bring their application code to create a continuous integration (CI) and continuous delivery (CD) pipeline to Azure.

Azure DevOps refers to the process of continuous integration and continuous delivery of the product to end user. Whenever a new change detected in the repository the build process is started and if a successful build is created the process of deployment started and the newly committed change reflected on the server automatically. So one can automate the process of build and deployment by using DevOps pipeline.

Benefits of Azure DevOps

Azure Boards

In a DevOps organization, we can track who is responsible for which task by using Azure Board. We can also check the health of the project using Azure board.

Azure Repos

Azure Repos provide you build Git Repositories hosted in the cloud. Hosting code in Azure Repos, you ensure that everyone on your team can access, track task and contribute to it if they have access for this.

Azure Pipelines

DevOps must have a continuous delivery pipeline. In order to be effective, this pipeline automates the most of task to deliver software, from developing software to deployment and deployment to production. This pipeline should provide the facility to clear communication between the various teams that manage these tasks.

Test Plans

Azure DevOps Pipeline provides a central location where all team members can coordinate all your manual test activities, track project progress, and get critical insights.

Azure Artifacts

Create, host, and share packages with the team, and add artifacts to your CI/CD pipelines or Azure DevOps Pipeline with a single click.

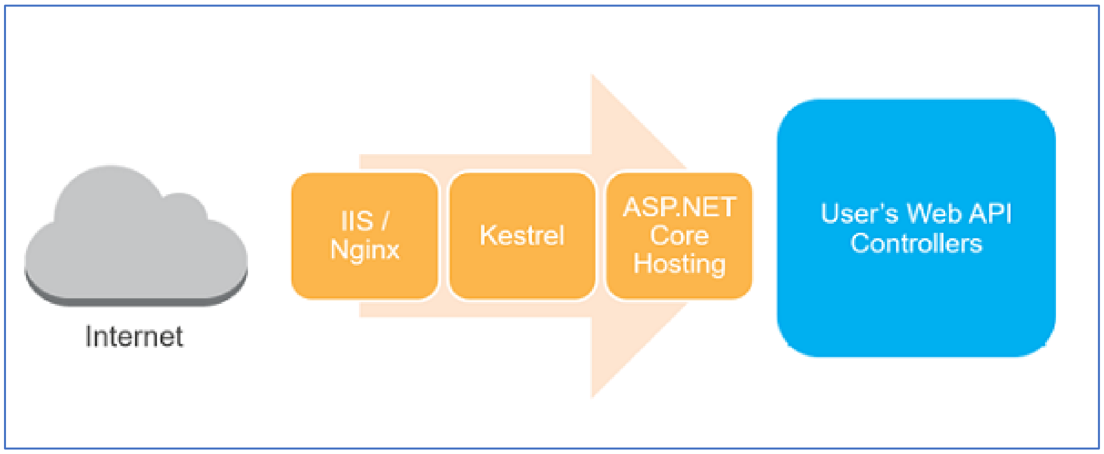
Deployment

- Published app can be deployed on the hosting server.
- process manager can be set so that it starts the app when requests arrive and restarts the app after it crashes or the server reboots.
- For configuration of a reverse proxy, reverse proxy can be set to forward requests to the app.

The dotnet publish command compiles app code and copies the files required to run the app in to a publish folder. The publish folder contains one or more app assembly files, dependencies, and optionally the .NET runtime.

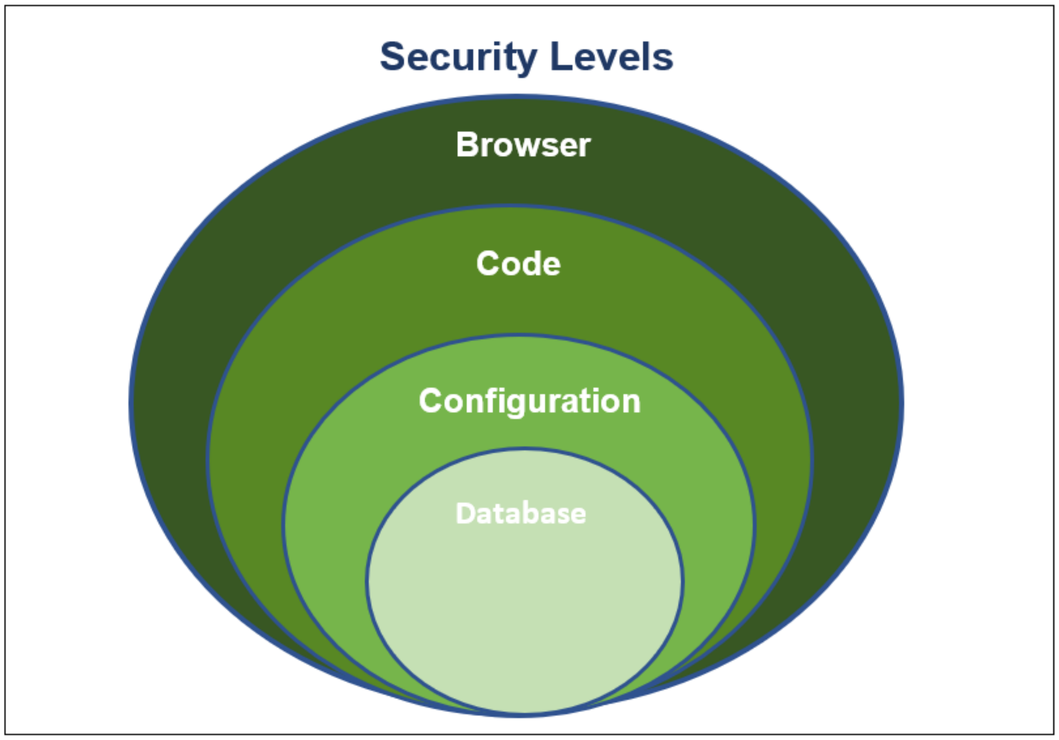
A .NET Core app can be published as self-contained deployment or framework-dependent deployment. If the app is self-contained, the assembly files that contain the .NET runtime are included in the publish folder.

For more information on .Net Core deployment, please refer:
https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-2.2



Security

There are four levels of security implemented to protect application from malicious users. The following diagrams shows how application is protected in different layers.



To protect the application, the following sections detail on how different security mechanisms are implemented.

Form Authentication

Providing Username and Password will generate JWT based on the user’s role and JWT will be used to access protected resources for the subsequent calls to API.

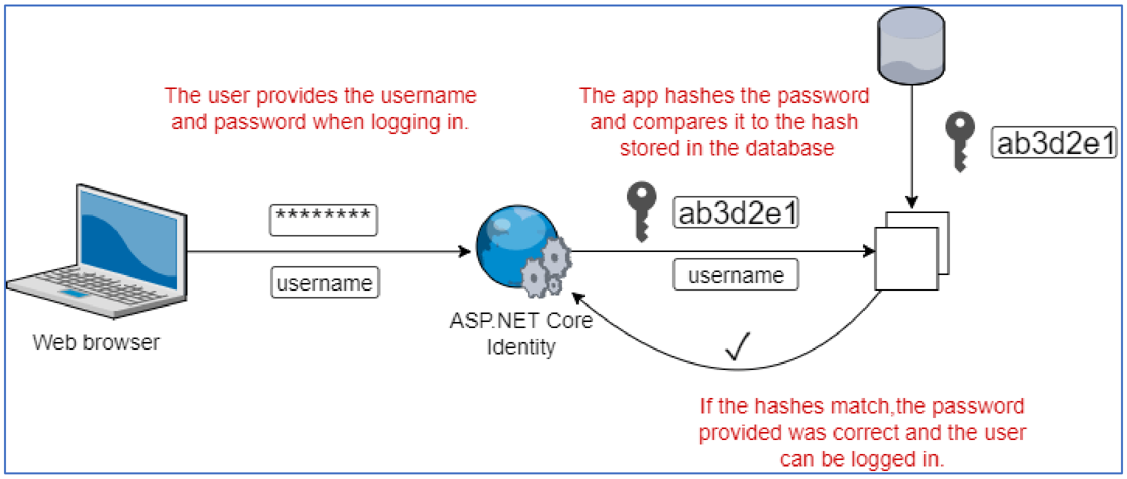
Password Hashing

When a user registers with the app, they provide a username and password (and any other required information). The app will create a *hash* of the password, and store it in the database along with the user's details.

A hash is a one way function, once password is hashed there is no way to get the original password back. For security reasons, the characteristics of the hash function are important; in particular, the hash function should be relatively costly to compute, so that if your database of password hashes were to be compromised, it would take a long time to crack them.

Asp.Net Core Identity and Password Hashing Technique

Asp.Net Core Identity Version 2:
PBKDF2 with HMAC-SHA1, 128-bit salt, 256-bit subkey, 1000 iterations


The user provides the username and password when logging in.
The app hashes the password and compares it to the hash stored in the database
ab3d2e1
****
username
ab3d2e1
username
Web browser
ASP.NET Core Identity
✓
If the hashes match,the password provided was correct and the user can be logged in.

OWASP (Open Web Application Security Project) Top Rules for Web Application

The Open Web Application Security Project (OWASP) is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.

The following sections describe top rules of OWASP.

Injection

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

The following section describes how to prevent SQL injection.

Approaches to prevent

- Applying the principle of least privilege to the database account
- Always validating untrusted data against a whitelist
  1. Type conversion
     Integer, date, GUID, etc.
  2. Using a regular expression
     Email address, phone number, name (but be careful)
  3. Listing of known good values
     Countries, products, colors etc.
- Using ORMs and their native ability to parameterize

Cross Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

The following section describes how to prevent XSS attacks.

Approaches to prevent
• Output encoding 1. The output encoding can be done in following context: CSS, HTML, HTML attribute, HTML form URL and JavaScript etc. • Validating request such input sanitization.

3rd party Plugins

The following table contains commercial third-party plugins will be considered in the application to handle all required features.

Plugin name	Purpose	Url
Telerik Reports	These RAD controls which makes development faster.	http://www.telerik.com/purchase/aspnet-mvc
Resharper	ReSharper enhances the Visual Studio development environment to several features that are especially useful during development.	https://www.jetbrains.com/resharper/

3rd party API

API name	Purpose	Url
Converge API	To make Secured Payment online.	https://developer.elavon.com/
Weather API	To get current weather or date wise weather.	https://openweathermap.org/api
Google Calendar API	To get scheduled appointments which synced from Genbook Calendar	https://developers.google.com/calendar/