Jonathan Gilmartin

Full Stack .NET Developer & Architect

Highly skilled **.NET Full Stack Developer & Architect** with **15 years** industry experience. Very capable Backend, Frontend and DevOps Engineer. Expert skills in **C#, SOLID Principles, .NET core, MVC, WebAPI, Microservices, Docker, Kubernetes, Azure Search, Cosmos DB, Modern JavaScript/ES6, React, CI/CD, DevOps, Automated Deployments, PowerShell** and **Azure Cloud Infrastructure**.

Contact

Website : www.jgilmartin.co.uk **Phone** : 07908 859 818
Location : Derby, UK
LinkedIn : linkedin.com/in/jgilmartin-dev
Stackoverflow : jgilmartin 6,847 10 55 78

Development Skills

Last Contract

.NET Core /MVC/ WebAPI
Microservices
Docker
Kubernetes
Azure Search
ES6
React
Cosmos DB
Azure DevOps Build and Deploy Pipelines

C# Programming

SOLID Principles
Async/Await
IoC/DI
LINQ
Generics
TDD & Mocking with Moq
Decoupled Unit tests
SRP Class Design
Repository & Adapter Pattern

.NET Full/Core

ASP.NET MVC
ASP.NET Core MVC
Hosted Services
Entity Framework
Web API
Command Line Apps
WCF Web services
Visual Studio & VS Code

SQL Server

DB Design & Implementation
SQL, T-SQL
Stored Procs
Views
SQL Agent
Maintenance Plans

Azure

Azure Search
Azure Containers
Cosmos DB
Redis Cache
Azure Service Bus
PaaS/IaaS infrastructure
Infrastructure migration
ARM Templates
SQL Azure
App Services
Virtual Machines
Auto Scaling
Azure Functions
Application Insights
Load Balancing/Gateways
Geo-Replication

Web Development

React + ES6
MVC + HTML5 + Razor
Fetch, AJAX, JSON
Node + Gulp
Legacy JavaScript & JQuery
REST Web API + Swagger API
Hand Coded HTML/CSS/SASS
Bootstrap
XML, XSD, XSLT, XSL, X-path

DevOps

Azure DevOps
CI/CD Pipelines
PowerShell
Automated Deployments
Custom Deployment routines (PowerShell & Gulp)
Database Deployments
TFS & GIT Branch Conventions

CMS

Sitecore
Umbraco Headless

Recent Work Experience

Sytner IT

Contract Software Engineer

Start: May 2019
End: April 2020

Sytner IT – Tech Stack	
Technology	C#, DI/IoC, SOLID Principles, .NET Core, .NET Core MVC, Microservices using Docker Images, Kubernetes Orchestration, Azure Container Registry, Razor, View Components, Azure Search Cosmos DB, React, Es6, Legacy JavaScript, JQuery, HTML, SASS, PowerShell, Node, Gulp, GIT
Architecture	Containerized Microservices Orchestrated with Kubernetes in Azure. Geo replicated between North and West Europe Azure Regions
Environments	Local, CI, QC UAT, Prod, Live
Deployment	Azure DevOps Pipelines, PowerShell, Docker, Helm, Kubernetes
Cloud Infrastructure	Azure Container Registry, Cosmos DB, Azure Search, Azure Service Bus, Redis Cache, Application Insights
CMS	Umbraco Headless
Tools	Visual Studio, VS Code, Postman, Swagger
Third Party Relationships	Evolution Finance, Black Horse, Expnoea

Project –**Sytneraffinity.com**
An application developed with a decoupled .Net Core MVC frontend with React components for the home and search pages. Powered by a number of microservices for business domains such as Search, Finance, Email, Content and many more. Each service was developed using .Net Core WebAPI and had its own build and build and release pipelines in Azure DevOps. All Services are compiled into a Docker image and deployed to Azure Kubernetes Service manged using kubectl and the Kubernetes Dashboard. We use a dedicated Kubernetes instance for all test environments which are split by namespaces (CI, QC, UAT). The Live environment has its own Kubernetes Service in a dedicated Azure subscription.

Each Microservice features the same solution structure. The entry point is a .Net Core Web API which depends on two .Net Standard Class Libraries. The first is for Domain Models and the second for Business Logic, a Unit test project that uses Moq for dependency mocking and a docker compose project for local development orchestration.
The overall architecture ensures that the Microservices do not directly communicate with each other, instead a C# REST client per service is generated and referenced in the UI project as a dependency. The only dependency each Microservice has is to the configuration Service, this acts as a distributed key value pair provider which is connected to each Microservice via Azure Service bus. This allows the services to be notified when a configuration change has occurred.

All Microservices implement the new .Net Core built in IoC container for Dependency Injection, referencing interface abstractions with object instances. Each class is abstracted with an interface and dependencies injected into the constructor.

For the Database we use Azure Cosmos DB, a NoSQL data store which saves JSON documents into collections. This works well for the application as we aren't restricted by strict schema definitions and we invoke CRUD operations via the official Microsoft SDK. While this works great for an object data store, it does pose a problem for reporting. To overcome this, we import the relevant data into a SQL Azure instance and then render a visualization with PowerBI.

Project –**Azure Search**
Sytner Affinity features a comprehensive search facility that utilises Azure Search, it featured facet filters and supported search via free text user input. Senior stakeholders felt that the search needed improvement and I was tasked to improve the quality, UX and accuracy of the search results. There was a significant number of mismatched results and some keywords returned results completely unrelated to the search criteria.
To achieve this task, I took advantage of the advanced scripting features available in Azure Search and refactored the search microservice to accept a Lucene syntax search string. This allowed me to restrict the search based on a number of key factors such relevance, partially matched keywords and keyword synonyms.
For bespoke search requirements I implemented a custom Edge Case system where we could specify automotive edge case exceptions. An example is the manufacturer "Seat" and "2 Seat" the algorithm would check to see if the string that contained "Seat" was proceed by a numeric value. If so, it would filter on the facet "seats" rather than the manufacturer "Seat "and therefore returned accurate results. Some other examples were "Mazda 6" and "6 Series", "Lotus" and "Lotus" as the partial name of some vehicle colours. The edge case system was config driven, so new edge cases could easily be added without the need for a deployment.
The end result was a far superior search facility which returned results far more accurate than the initial version.

Project –**Online Finance Proposals**
A major project in which I played a significant role, Online Finance Proposals was a new feature of SytnerAffinity.com which allows customers to purchase a vehicle and apply for finance in a 100% digital online journey. The main goal for this solution was to have a customer successfully transacted by the end 2019.

To deliver this solution I worked closely with Black Horse and Evolution Finance. I assisted Evolution with their API which provided endpoints to submit a finance application, download customer documents, check the application status, ect. It also offered a Webhook that provided status updates on applications.

The project involved a lot of Operational Decisions and Compliance Reviews. I was required to work with Black Horse to hone the journey for their legal team to approve. Once it was signed off we could move forward with the development. The most notable section of the journey which consisted of 50% of the development work was the Finance Proposal form. This was a very complex form that featured dynamic sections for address history, employment history and affordability. We integrated with Loqate, a third party API for address lookup and used JQuery Validate for the client-side validation. The validation was very extensive to ensure the user could only enter valid data. The UX was designed to be as simple as possible to keep the user engaged and to minimizing any risk of confusion.

I developed a .NET core hosted service which monitored a web hook. When a response was received, the hosted service would trigger updates in Cosmos DB which in turn allowed for a real time finance decision. The Hosted Service also would trigger emails to the customer when new status updates were received from lenders. The project was completed and released in late November. The First finance proposal was completed early December resulting in the project being deemed a success.

Highly skilled **.NET Full Stack Developer & Architect** with **15 years** industry experience. Very capable Backend, Frontend and DevOps Engineer. Expert skills in **C#, SOLID Principles, .NET core, MVC, WebAPI, Microservices, Docker, Kubernetes, Azure Search, Cosmos DB, Modern JavaScript/ES6, React, CI/CD, DevOps, Automated Deployments, PowerShell** and **Azure Cloud Infrastructure**.

Education

Sitecore Certified Professional Developer
Awarded – April 2018

Developing cross-platform apps with C# using Xamarin
Awarded – Oct 2015

Advanced MSC in Computer Science
Institute – Leicester University
Result - Merit

BSC Multimedia Computing
Institute – De'Montfort University
Result - 1st Class Honours

Contact

Website : www.jgilmartin.co.uk **Phone** : 07908 859 818
Location : Derby, UK
LinkedIn : linkedin.com/in/jgilmartin-dev
Stackoverflow : jgilmartin  6,847 10 55 78

Recent Work Experience

Pinewood/Pendragon PLC
Technical Architect

Start: May 2017
End: April 2019

Pendragon PLC – Tech Stack	
Technology	MVC Core, .NET WebAPI, Azure, Microservices, Razor, SQL Server, Entity Framework, C#, IoC, SOLID Principles, VueJS, JavaScript, HTML, SASS, SOLR, PowerShell, Gulp, TFS, Moq
Architecture	SQL Azure with Azure App Services, 100% PaaS Infrastructure
Environments	Local, DEV (OnPrem VM), MAIN (OnPrem VM), PROD (OnPrem VM), Live (Azure PaaS)
Deployment	Azure DevOps Pipelines, PowerShell, Gulp
Cloud Infrastructure	SQL Azure, Azure App Services, Redis Cache, Application Insights, Application Gateway, SOLR from SearchStax
CMS	Sitecore 9.0
Tools	Visual Studio, Postman, Swagger
Third Party Relationships	Codeweavers, 3Chillies, Sitecore, SearchStax

Project –**carstore.com**
An ASP.NET MVC Core Application powered by a suite of .NET WebAPI Microservices. The MVC App takes full advantage of new .NET core features such as Tag Helpers, start-up config and the built in IoC container. The front end uses VueJS and SASS. I use Gulp to compile the JS files down to single uglified & minified JS file for deployment. The SASS is Transpiled to a single CSS file. The service client interfaces are injected into the controllers and I use model binding to serialize the JSON request payloads. The controller action methods simply call the injected service clients to invoke the microservices. For error and exception management I created a generic logger that could switch between log4Net (Local, OnPrem) and AppInsights (Live Azure) based on the injected interface. All C# code follows SOLID principles. This consists of Dependency Injection, Interface Segregation and classes that have very specific responsibilities. I use the Repository and Adapter patterns for accessing data and Asynchronous code throughout the stack using Async/Await and Task<T>.

Each microservice was designed at the domain level, e.g. (finance, Vehicle Search, ect) and adhered to a strict solution architecture. The result is that all microservices are structured the same with only the business logic being different so developers can easily switch between working on other services.
Swagger has been integrated for endpoint API testing and I use postman which is configured to easily test each service endpoints per environment.
The microservices have a single SQL database that is private. Its DB connection abstracted via the service endpoints. I use Entity Framework and Reverse POCO to get the benefits of Code First but with an existing SQL Server database design. The search service uses SOLR. This is an open source enterprise search system.
The Microservice Request/Response Models and HTTP Clients are separated into libraries and then published as Nuget packages to a private repository. Any microservice that depended on another, references these packages and consumes them by calling the client methods.
Each microservice client inherits a base http client so they all correctly persist .NET's HttpClient object.

The MVC app is completely de-coupled and relies 100% on the microservices. It therefore references the models and client nuget packages to consume the services.
Each service is secured by using token auth over HTTPS. Each environment has a CI/CD pipeline. We have a dedicated pipeline per environment. so, for example, when a developer checks code into DEV from local, it will build that changeset and on success deploy to the OnPrem Dev environment. The same is also true for merges. Once code in DEV has been tested, we merge the DEV branch to MAIN and that will trigger the same process which deploys to the MAIN environment. The CI/CD Pipelines are setup using Azure DevOps. The deploy tasks consist of custom PowerShell and Gulp scripts combined with default AzureDevOps build tasks. The test environments (DEV, MAIN, PROD) are OnPrem using dedicated Virtual Machines to host the MVC app and each service in IIS. The live environment is 100% Azure PaaS using AppServices and SQL Azure.

Motorpoint PLC

Start: Mar 2011
End: May 2017

Pendragon PLC – Tech Stack	
Technology	.NET C#, MVC 3, Razor, C#, ASP.NET WebAPI, SQL Server 2012, jQuery, JQueryUI, JQueryMobile, C# Console Apps, TFS, Unit & Integration Tests
Architecture	Classic N-Tier Architecture with 5 load balanced VM's for the web server and 2 Database Server VM's, Hosted onPrem
Environments	Local, Staging, Live
Deployment	Manual Deploy
Cloud Infrastructure	Azure CDN
CMS	N2CMS
Tools	Visual Studio, Postman
Third Party Relationships	Autotrader, Motors

Project – **Motorpoint.co.uk**
The site was developed using ASP.NET MVC using C# for the server-side and Razor for the view engine. The data layer consists of several SQL Server Databases in which I used Entity framework as the ORM. For the client side I used JQuery and JQueryUI for AJAX interactions and UI elements. The application featured over 200 unit tests to ensure all areas of functionality could be tested. The project was managed using TFS for both the source control and Agile Sprint work items. For the CMS I integrated N2CMS, an open source .NET CMS system with an elegant C# API. N2CMS provided powerful CMS functionality together with a simple authoring experience.

Project – **Auction4Cars.com**
The Application was developed using ASP.NET MVC with Razor for the Views and C# for the Controllers and Models. It used a SQL Server Database and Entity Framework got the ORM. For the real time auction countdown and bid notifications I used JQuery AJAX and WebAPI services. The Front-end UI was developed using Bootstrap.
To get the auction logic 100% accurate in line with the documented scenarios I created over 250 integration tests. These proved critical to ensure the bidding logic was 100% actuate against the documented bidding scenarios. This enabled me to accurately recreate the existing bidding logic and provided a safety net for future code changes.