Justin Spencer

Ashburn, VA • jspencer.jms@gmail.com • jamsupreme.net

Objective

To work at a company that requires skilled engineers to help realize its vision; ideally providing me with learning opportunities that I can also pass on to my fellow teammates.

Education

Neumont University - https://neumont.edu
Bachelor of Science in Computer Science, 2009

Skills

Web
ASP.Net (MVC5, WebAPI, Razor) - Ruby on Rails - JSON Schema - React - GraphQL - HTML & CSS & JS & AJAX - Knockout - Angular (v1) - OData - RESTful APIs - Microservices - XML / JSON / Content negotiation - Telerik/Kendo - JQuery - NodeJS/Express - WebForms

Middle Tier
.NET Framework 4.5.1 - ADO.Net - Entity Framework & IQueryable - NHibernate - LINQ - IOC containers (Autofac, Unity, MEF) - MSFT Enterprise Library - Logging frameworks (Cloud-based, Windows EventLog, custom) - FluentValidation - AutoMapper - TPL Dataflow / Task-based programming (async/await) - Ruby Gems - ActiveRecord - Sidekiq - Kafka - Bouncy Castle (PKI FIPS encryption)

Databases
MS SQL - DbUp - Redis - PostgreSQL - SQLite - Limited use of mySQL, MariaDb, DynamoDb, MongoDB

Infrastructure
Bamboo - Jenkins - Octopus Deploy - Cruise Control - NAnt - Continuous Integration (CI) - Continuous Deployment (CD) - Various AWS tools (S3, SQS, EC2, ECS, ELB, ALB, etc…) - Powershell - Chef (mostly via a custom in-house DSL) - Limited experience with Terraform, OpenShift.

Theory
Metaprogramming (C# and Ruby) - Design patterns - Agile methodologies (Scrum, Kanban) - Localization / Internationalization - Async code - Concurrent code - Domain Specific Languages (DSLs) - Functional Programming (Elixir/Erlang/JS Ramda) - Contract Testing (Pact) - "Fakes" best practices for APIs

Tools / Other
Subversion - Git - JIRA - Confluence - Crucible - TFS - ReSharper - Visual Studio - limited Java / Python skills - Section 508 & ARIA accessibility - ISO / CMMI experience - Slack - NUnit / xUnit / MS Unit / Moq - Nuget - Redgate tools - Facebook / Twitter / Foursquare APIs - LeanSentry - Rollbar - Raygun - NewRelic - Splunk - Nexus Repository Manager

References

Available upon request.

Employment Summary

• Excella Consulting	March 2016 - Present	https://www.excella.com/
• Fishbowl Marketing	August 2014 - March 2016	http://www.fishbowl.com/
• REI Systems	July 2009 - August 2014	https://www.reisystems.com/

Work Experience

Excella Consulting

Lead Consultant (Tech Lead / Supervisor)
Project: https://my.uscis.gov/
Team Size: 4 per "pod", 8 per team, 24 on the web project

My role at Excella began as an individual contributor which later evolved into being a tech lead and supervisor. Since my inception to the project I strived to ensure the architecture remained evolvable as the team and products grew in size. The role of the myUSCIS team also expanded as we were in a pivotal position to ensure several other systems (such as identity providers, user profile storage, adjudication systems, secure file storage, and data governance) were

Justin Spencer
Ashburn, VA • jspencer.jms@gmail.com • jamsupreme.net

all able to effectively work in concert. Given that each of the disparate systems had their own development stacks, skill levels, and competing priorities, it involved a great deal of effort to ensure everyone moved in concert while our team continued to deliver our own products.

Our team was roughly structured as 3 teams, with each team having roughly 8 developers, and eventually we also invented a subdivision of a "pod" if we need to have a single team focus on multiple products simultaneously. Generally speaking a supervisor is responsible for approximately 3 direct reports (the pod) while still being responsible for the health of the team as a whole unit. In addition to keeping the team operating smoothly, the tech leads often have to spearhead various cross-cutting initiatives (best practices, skill gaps, subject matter expertise, etc.) that all serve to help the team grow together. In addition to this, it's common for the tech leads to also be pulled into many client-facing or external stakeholder meetings for expertise or high level estimates. In my role as supervisor I not only gathered feedback for my direct reports but also for that of several other team members on our project and built healthy working relationships with team members of all skill/experience levels. The guidance of growth for each individual varies wildly, but I can speak to several cases from junior to senior level employees in which I helped them grow in their careers.

Since we were an agile team constantly delivering products and iterating on them, I'll attempt to highlight substantial milestones:

- Complete Ruby Gem (and pipeline) overhaul - Prior to my joining the team, all of our internal ruby gems were "file-based" (meaning they were either pulled in by directory or a remote git branch). In my effort to build them as gems and deploy them to a remote repository, we discovered hidden dependencies, inaccurate gemspecs, circular dependencies, and substantial behavior differences in how a published gem behave from local refs. On the surface it sounds like a trivial pursuit but constituted several months of reworking several gems, their dependencies, and the underlying infrastructure (builds in Jenkins and deploying packages to Nexus) to ensure they were correctly deployed.
- myUSCIS contract recompete tech challenge - When the myUSCIS contract was up for a recompete, a tech challenge was one significant factor for determining the winner of the contract. The challenge required that the team be able to build an entire application in the cloud along with its infrastructure and at the end of the day deliver a desirable minimum viable product (MVP) to the requirements specified by the "client". The day itself was an intense affair that involved coordinating a small scrum team throughout a highly-accelerated simulation of sprints. Prior to the actual day, I was involved in setting up much precursor work for tools and libraries that we could quickly leverage for spinning up our infrastructure pipeline (Terraform), front end (React, JSON Schema), and back end (Rails). I also assisted in ironing out coding conventions to maximize how quickly everyone could produce code under the time crunch of the competition.
- Rolling out several "eProcessing" forms - You can now file online for several different benefits. When I initially joined the project the existing framework for building against forms was "too generic" and made assumptions about potential future forms such that it was untenable. I spearheaded several initiatives in which we focused on a simpler strategy for implementing forms and we were later able to refactor out common code and we later built a few Ruby mixins, metaprogramming classes, and DSLs on top of these common behaviors to greatly expedite the speed at which we could add new forms. The ecosystem behind efiling also expanded greatly over time, eventually requiring us to integrate with several different systems at once and often presenting interesting difficulties.
- Expanding "eProcessing" forms into new systems - One of the more interesting problems was expanding eprocessing forms from the initial implementation in which it was a single data store. This was made "more interesting" by the fact that we implemented it incrementally while we were adding the new "G-28" (a form filed by a representative) such that we had many interesting combinations of data in both the new system and the old. Adding another layer of complexity was the fact that we ended up restructuring the "new" database another time such that a few more permutations arose. All of this had to be carefully coordinated to avoid data corruption or loss as we constantly rolled out features.
- JSON Schema adoption - It is difficult to underscore the tremendous value in adopting JSON Schema and being able to convince several other systems to adopt it. Together with contract testing, it adds a compelling layer of safety and predictability when integration with a huge number of external systems that may all disagree on how they want to store data. Even more so because we are in an environment where we are building a substantial number of electronic forms. I played an important role in our initial in-house prototype which later evolved into a full-fledged internal product that we used internally and opened up to everyone else in the USCIS organization. From both a technical and business standpoint, it provided a huge boon in being able to communicate with external systems. As a minor aside, the prior solution was a mishmash of tribal knowledge and copy/pasted excel sheets that were subject to change depending on the email or other remote data source.
- Various Best Practices - There are a number of best practices, patterns, and conventions I attempted to establish while on the team. Some worthy of note:
    - Feature Toggles - Figuring out strategies for managing "feature toggles" in a continuous delivery environment and how to avoid stale toggles or blunders regarding a toggle turning on/off when undesired. There are also potential performance (and complexity) implications if several toggles are nested within one another. We also had an interesting cascade of data stores in which we checked for toggles that was initially a bit convoluted but was later simplified into a neat pipeline.
    - Fault Tolerant Jobs - Outlining our recommended strategy for running jobs in production. The guide illustrated best practices for logging and included at least 8 common failure scenarios and how we can easily write tolerant (and ideally idempotent) code to account for them.
    - Pull Request (PR) Best Practices - How we approach PRs and strategies that can help us all identify what a "Good" PR looks like compared to a "Bad" PR (both from the perspective of an editor and reviewer)
    - Microservices - There are a lot of considerations that can change when moving to microservices from a

Justin Spencer

Ashburn, VA • jspencer.jms@gmail.com • jamsupreme.net

monolith. To name a few: data redundancy, logging best practices, effective domain separation, strategies for decomposition and aggregation, etc. From a less technical standpoint, there are also concerns about developers being siloed in their one microservice, and without a preexisting baseline like contract tests or comprehensive end-to-end tests, there's always the risk of unexpected implications in seemingly innocuous enhancements. I helped guide the team in establishing several conventions to keep us getting the most out of our microservice architecture.

- "Making significant decisions" - To avoid the inevitable "shiny new thing" problem that developers run into, we worked together to devise best practices for adopting new technology in a way that won't result in crippling tech debt within a year. Following these guidelines has helped the team safely keep up with the tech curve without making a mess or exposing ourselves to huge risk when "The one expert" is suddenly out of office.
- Evolvable Architecture ("How to backpedal significant decisions") - Sometimes even the best laid plans go awry. We had to devise strategies for delivering value while working around big pieces of tech debt and ideally chipping away at that debt until it disappears. Optimistically this would be something like a "strangler pattern" but for certain architectural choices (e.g. GraphQL) there can be many implications that bleed into several applications across your full stack. I helped devise rules of thumb for deciding when a technical decision isn't working out for us and how we can illustrate it to our peers as well as practices we can establish for dealing with that debt.

Fishbowl, Inc

Software Engineer

My work at Fishbowl was often variegated, changing depending on the needs of clients and the availability of resources that could develop the software necessary to meet those needs. Given my versatility as a full stack developer, my specific responsibilities were generally in flux from one project to the next. I'll attempt to illustrate some significant accomplishments among those projects.

Continuous Integration / Continuous Deployment overhaul
For many years Fishbowl had been using CruiseControl and NAnt in order to deploy its products. There were a few problems with how it was set up: It was slow, it had lots of opaque behaviors, it did not allow concurrent builds or deploys, so on and so forth. After making a small prototype, I gradually overhauled most of our products into the new setup. Deployments went from an error-prone multi-hour affair to a painless process that could be done in minutes. We also have some Java projects, so I am comfy with Java & Maven in Bamboo (though Octopus is specifically for Windows). In tandem with this, I also turned many of our dependencies that were once filepath-based into internally-hosted nuget packages (with respective docs in our Sharepoint).
Relevant tech details: Bamboo, Octopus Deploy, Powershell, shell, Maven, Unit test runners / parsers, Nuget

Single Sign On (SSO) consolidation
Fishbowl has had a sign on mechanism since its inception. However, it eventually acquired another company (particularly for its mailing engine), and thus ended up with two disparate login systems. This portion of technical debt has been around for several years but eventually needed to be tackled in order to introduce some usability enhancements (a fancy sidebar for navigation across products).
Relevant tech details: Active Directory (AD), Security Assertion Markup Language (SAML), Data migration & facade work, unified API for navigation menu.

API-Based UI re-design
When one of our products sorely required a UI re-design, we had to migrate it from ASP.Net WebForms to MVC5 (as the new UI framework was all in Razor). We thought an API-first approach would help us test the code better, and also give us an easy path for exposing the relevant data to other products that may later require it. In the process the code was made more modular, more testable, and more extensible than it was before. I also built a few interesting IQueryable helpers to enable full text search.
Relevant tech details: MVC5, WebApi 2, Autofac, xUnit, Kendo MVC, Moq

Segmentation UI widget & API
For several of our products, we needed an HTML widget that could access our new "Segmentation" API which can segment (i.e. facet) user data in several ways. The widget itself was interesting because it was pure HTML/JS/CSS (with a small dependency on JQuery and a few other libraries), which we could then encapsulate in an easy MVC helper, which was then encapsulated in a nuget package. Since the API could potentially induce expensive operations, the full stack (JS to API to DB) was all async with cancellation support. While the JS provides "real-time" data, the underlying model was mostly intended for "easy" segments that could be calculated ahead of time.
Relevant tech details: Angular (v1), Jasmine, Protractor, Gulp, Bower, NodeJS / npm, MVC, WebApi

Social Media Publishing
Many clients wanted to be able to publish the content of their mailings to their social media accounts when the mailings go out. We created a UI with which they can create social posts from their mailings and modify the content as desired. The feature itself sounds trivial, but since social posts could be edited independently of mailings and we actually have two mailing systems (one built on top of the other with added workflow and business rules), it became slightly more complex as there were many workflows and mixed states in which the mailings and posts could exist.

Assorted other work

Justin Spencer

Ashburn, VA • jspencer.jms@gmail.com • jamsupreme.net

There's some other work that doesn't quite stand on its own but is worth mentioning: Understanding our mission-critical software (mailing system), database performance tuning, mixed-stack work: Java & Python, Code reviews, architectural planning, improving our documentation (by an order of magnitude).

REI Systems

Sr. Software Engineer

Much of my time at REI was spent on projects within the Health Resources and Services Administration (HRSA) program. Chief among the HRSA products is the Enterprise Handbooks (EHB) which is the nickname given to the web site in which much of their work is done. Most of my work involved transforming complex business rules into well-designed systems.

Audit Tracking Component

For this project, there was a business need to see snapshots of data that had to be visualized in a way that organized it into sections and fields. This was a large crosscutting concern needed to be easily applicable with any data model and solution.

Relevant tech details: Implemented auditing as a crosscutting concern via attribute decoration, interceptors (using Unity), and Reflection to accommodate varying data models. Since we were limited to 3.5, use of dynamic wasn't an option.

HIV/AIDS Bureau (HAB) Formula, Initial Release & Center for Disease Control (CDC) Integration

For HAB, they calculate funding for grantees based on several data points. The purpose of this module was to read information from all those data points, display relevant information to the user, and appropriately make calculations automatically. The result saves several weeks of work that was previously done manually by the client. During this release I was responsible for the successful integration with external systems while overseeing another developer to help make those changes.

After the initial release of HAB Formula, the client wanted the ability to drop an excel file suiting a specific format onto a file server which can then be read into our system. Since this behavior could be done while formulas are in progress, it followed a two-step staging process and then would explicitly be loaded into formulas if desired.

Relevant tech details: Used dependency injection to allow multiple implementations for file retrieval, validation, and parsing. Internally used Aspose for reading excel sheets into data sets for parsing. Knockout for some UI binding & validation. WebForms.

HRSA EHB Funding Memo Re-Design

In addition to upgrading to our new internal technology platform, this re-design introduced several advanced new configurations for the manner in which a "Funding Memo" is created. The "Funding Memo" is essentially a description of the funding that is to be award to several potential grantees. With all of the new configurations introduced, the UI had to be flexible to accommodate any mix of configurable parameters. As one can imagine, it was also critical that all of the funding specifications calculated the desired total. I was also responsible for onboarding a few new developers throughout the project.

Relevant tech details: WebForms.

HRSA EHB Post Award Re-Design

This project was a substantial enhancement to the project I first worked on in the company. We upgraded the solution to the new internal technology platform and implemented several workflow-related enhancements. On this team I was responsible to effectively coordinate two more senior developers to leverage the existing code base while introducing enhancements.

Relevant tech details: Migrated Classic ASP (in Visual Basic) to WebForms.

HRSA EHB Awards Re-Design

The awards process in EHB (Enterprise Handbooks) is the phase in which money and terms and conditions are created to be given to the appropriate grantee. My role in this project was to create the entire "funding information" module, which allowed the responsible HRSA employee to distribute money. It was a large team of about 10 developers and occasionally required a team effort in order to integrate modules.

Relevant tech details: WebForms.

HRSA EHB Post Award 2.0

As an entry-level software engineer, my responsibilities were to create the internal review module for "EHB Post Award" module. Grantees receiving funds from HRSA would submit applications to meet the terms and conditions imposed upon them, which would then be reviewed by HRSA employees. This behavior also had to be implemented for another module (prior approvals) in which a similar review process was necessary, but based on a different subject matter. For this reason, the code base and data model had to be flexible enough to accommodate both modules.

Relevant tech details: WebForms.