Aug 10th, 12:00 AM

Deploying APIs in the Cloud: A Novel Approach to the MIS Infrastructure Course

David Schuff
Temple University, schuff@temple.edu

Mart Doyle
Temple University, mdoyle@temple.edu

Deploying APIs in the Cloud: A Novel Approach to the MIS Infrastructure Course

Completed Research

David Schuff
Temple University
schuff@temple.edu

Mart Doyle
Temple University
mdoyle@temple.edu

Abstract

Among the most in-demand technology jobs for 2020 include cloud architect and software developer. The typical IT infrastructure position will require programming knowledge, while the typical developer will require infrastructure knowledge. The creation and deployment of APIs is an ideal vehicle to teach both programming and cloud infrastructure in an integrated way. This paper describes the design and structure of a new undergraduate cloud infrastructure course that takes this integrated approach.

Keywords

Cloud computing, APIs, software development, curriculum design, pedagogy

Introduction

According to CIO Magazine, two of the 10 most in-demand technology jobs for 2020 include cloud architect and software developer (White, 2020). However, these skills are increasingly seen as part of an integrated whole. A "full stack" developer is described as someone with front and back-end development knowledge; this can also include DevOps knowledge to implement and deploy the code they write (Tylosky, n.d.; Alter, 2018). This combination of skills is also consistent with White's (2020) definition a cloud architect as someone responsible for not just the infrastructure but also "deploying, managing, and supporting cloud applications" (White, 2020). In part due to DevOps, the lines between infrastructure and development are blurring. The typical IT infrastructure position will require programming knowledge, while the typical developer will require infrastructure knowledge. Future IT professionals must be able to integrate these two skills in order to effectively manage modern software applications.

A practical example of the intersection of software development and cloud infrastructure is the proliferation of software services that communicate through web application programming interfaces (APIs). Web APIs facilitate communication between front and back-end components of a web-based application (Wikipedia, n.d.; Olsen and Moser, 2013), and therefore serves as a conceptual and practical framework for building cloud-based software applications that function at scale over the web.

Conceptual understanding and the ability to consume APIs has been a topic in IS curricula (Olsen and Moser, 2013; Hosack et al., 2012; Wyner and Lubin, 2011). However, we propose the creation and deployment of APIs is an ideal vehicle to teach both programming and cloud infrastructure in an integrated way that gives students a holistic view how cloud-based applications work. This paper describes the design and structure of a new, unique undergraduate cloud infrastructure course introduced last year into the MIS curriculum at Temple University in Philadelphia, Pennsylvania. It is required of all MIS majors and builds skills in constructing a cloud infrastructure using Amazon Web Services, developing an API using Node.js, and then deploying that API on a cloud infrastructure.

The Environment

Temple University is a large, public, urban institution with over 39,000 students. Its primary mission is to educate the regional undergraduate population through its over 400 degree programs. There are 17

schools and colleges including liberal arts, business, education, law, media and communication, music and dance, and engineering. The business school has over 8,500 students and nine academic departments. The Department of Management Information Systems has over 430 undergraduate majors and 330 undergraduate minors.

The department recently went through a major curriculum revision, shifting its emphasis to API-based software development, cloud infrastructure, user experience design, and cybersecurity. One of the major changes was the IT infrastructure course, which had focused on traditional physical networking concepts and virtualization. Some cloud topics in that course used the Microsoft Azure platform.

In order to address changes in technology and place a greater emphasis on the cloud, the course was re-imagined as a cloud infrastructure course. There are two software development courses in the curriculum: the first covered developing web-based interfaces using JavaScript and consuming APIs; the second covered software services and APIs using JavaScript and Node.js. We decided the cloud infrastructure course would act as a bridge between the two software development courses. This achieved three pedagogical goals. First, it would reinforce the programing concepts students learned in their first programming course. Second, it would allow for the introduction of API programming so students would have something to deploy on a cloud infrastructure. Third, it would give students a "head start" on the second programming course, which required them to build more sophisticated services and APIs.

Course Goals

The course addresses several key learning objectives:

1. **Understand** the concepts of modern cloud computing.
2. **Build** a cloud application deployment infrastructure using Amazon Web Services (AWS).
3. **Analyze** and configure a cloud infrastructure for scaling and redundancy.
4. **Develop** a simple RESTful API using Node.js.
5. **Deploy** an API to a cloud infrastructure using instance-based and application-based methods.

From these broad goals, we developed nine specific learning goals for the course that could be evaluated through assignments and exams:

1. **Understand TCP/IP-based networking concepts** – This includes topics such IP addressing, routing, network tools such as ping and tracert, and ports. These topics are explained within the context of both physical and virtual, cloud-based networks.
2. **Understand the economics of cloud services –** Students learn the value proposition of the cloud and how businesses accrue costs when using cloud-based services. They learn how to set up cloud-based networks that minimize costs while maintaining a target service level.
3. **Create and secure an AWS Virtual Private Cloud** – Students set up a virtual network that is redundant and secure. They implement redundancy through multiple subnets and implement security through AWS security groups, public and private subnets, and operating system level firewalls.
4. **Create and configure Elastic Compute Cloud (EC2) Virtual Machines** – This includes choosing the operating system, configuring network options, storage, and security groups. Students then connect to and test their virtual machine. They also create custom Amazon Machine Images (AMIs) that will be used for server scaling.
5. **Create a robust server infrastructure using load balancing and auto-scaling** – This involves creating AWS launch configurations and auto-scaling groups containing rules for dynamically increasing or decreasing on demand the number of application and web servers based.
6. **Configure and deploy a database using the Relational Database Service (RDS)** – Students create and configure a relational database server using RDS. They also connect to the database using client software, create several schemas, and populate those schemas with data.
7. **Create a RESTful API using Node.js** – Students build an API that responds to a HTTP GET or POST request from a web page using the Express framework. They also create the web client to test

the API. Students augment their API with the capability of receiving parameters from an HTTP GET or POST request and performing actions based on the value of those parameters. This includes using those parameter values to construct a SQL query.

8. **Deploy a web-based application to a set of EC2 instances** – Students deploy their functioning API and client application to web and application servers and create an AMI based on that configuration. Then they apply load balancing and auto-scaling to ensure the application runs optimally under high and low loads and continues to run when one of the servers fail.
9. **Deploy a web-based application using Elastic Beanstalk –** As an alternative deployment method, students deploy their API on the AWS Elastic Beanstalk managed deployment service instead of their own application server. They compare and contrast the two deployment methods.

Technology Choices

The two key technologies used in the course are Amazon Web Services (AWS) as the cloud platform and Node.js (and JavaScript) for the development environment. We chose AWS for several reasons. First, it has a well-developed, wizard-driven user interface. It is easy to use, allowing students to focus on what they are building instead of navigating the interface. Second, it has a robust "free-tier" allowance, giving students a lot of room to experiment before they start incurring charges. Third, it has services such as Elastic Beanstalk and Lambda that are well-suited to the scalable deployment of software services; this gives the flexibility to introduce new topics such as microservices as the course (and the industry) evolves.

The decision to use Node.js was based on several factors. First, using Node.js allowed for the use of a single language – JavaScript – across all the programming courses, reducing the time required to teach syntax. JavaScript is one of the few languages used on both the client-side and server-side. Second, JavaScript is among the top five most popular programming languages in many lists (Putano, 2019; Chan, 2019). Third, Node.js is a popular environment and it is open-source. Along with a free editor such as Notepad++ or Visual Studio Code, students use realistic, widely-used development tools at no cost.

Course Structure

The course is divided into three multi-week modules:

- Module 1: Building a robust, scalable AWS cloud infrastructure
- Module 2: Building a web-based application with a RESTful API
- Module 3: Deploying the web-based application to the cloud

Each class topic is divided into two 80-minute sessions. The first session is a discussion-driven "lecture" that covers key concepts and requires students to integrate readings and videos to demonstrate understanding. The second session is a lab-driven, experiential in-class activity that walks them through building a component of their cloud infrastructure (in Module 1), creating their application (in Module 2), and deploying their application (in Module 3). All in-class exercises build on previous exercises and by the end of the course students build a complete cloud infrastructure and working application.

The in-class activities are also supplemented with experiential learning outside of class. In the first course module, students complete a series of "Qwiklab" exercises (see http://qwiklabs.com). The Qwiklabs introduce an aspect of AWS through a highly-controlled "sandbox" experience. For the second part of the course, the additional experiential activities are Node.js exercises that refresh the students' JavaScript knowledge from prior courses. We designed the course to be delivered to medium class sizes (40 to 60 students) with multiple sections and multiple instructors. The in-class activities are designed to be completed individually but working together in small groups of two or three students is encouraged.

A semester-long, four-part class project uses concepts introduced in the lecture and in-class activities. The project requires students to independently build a second cloud infrastructure, develop a second web-based, API-driven application, and deploy that application to their infrastructure. The following table summarizes how the nine learning goals are covered throughout the course. The rest of this section describes each module, how it addresses the learning goals, and that module's in-class activities.

Learning Goal	Module 1	Module 2	Module 3
Understand TCP/IP-based networking concepts	✓		
Understand the economics of cloud services	✓		
Create and secure an AWS Virtual Private Cloud	✓		
Create and configure Elastic Compute Cloud (EC2) Virtual Machines	✓		
Create a robust server infrastructure using load balancing and auto-scaling	✓		
Configure and deploy a database using the Relational Database Service (RDS)	✓		
Create a RESTful API using Node.js		✓	
Deploy a web-based application to a set of EC2 instances			✓
Deploy a web-based application using Elastic Beanstalk			✓

Table 1. Learning Goal Mapping by Module

Module 1: Building a robust, scalable AWS cloud infrastructure

This module builds skills in support of six learning goals: "understand basic TCP/IP-based networking concepts," "understand the economics of cloud services," "create and secure an AWS Virtual Private Cloud," "create and configure Elastic Compute Cloud (EC2) virtual machines," "create a robust server infrastructure using load balancing and auto-scaling," and "configure and deploy a database using the Relational Database Service (RDS)." Students spend the first half of the course learning how to create a fault-tolerant, scalable network infrastructure in the cloud. They create software images so virtual servers can be created on demand. Security is implemented through the server (using security groups) and operating system (using the Windows firewall). The economics of cloud services is explored using the pricing models for these services. Figure 1 depicts the AWS virtual network students build, and an overview of several of those exercises is described in Table 2.

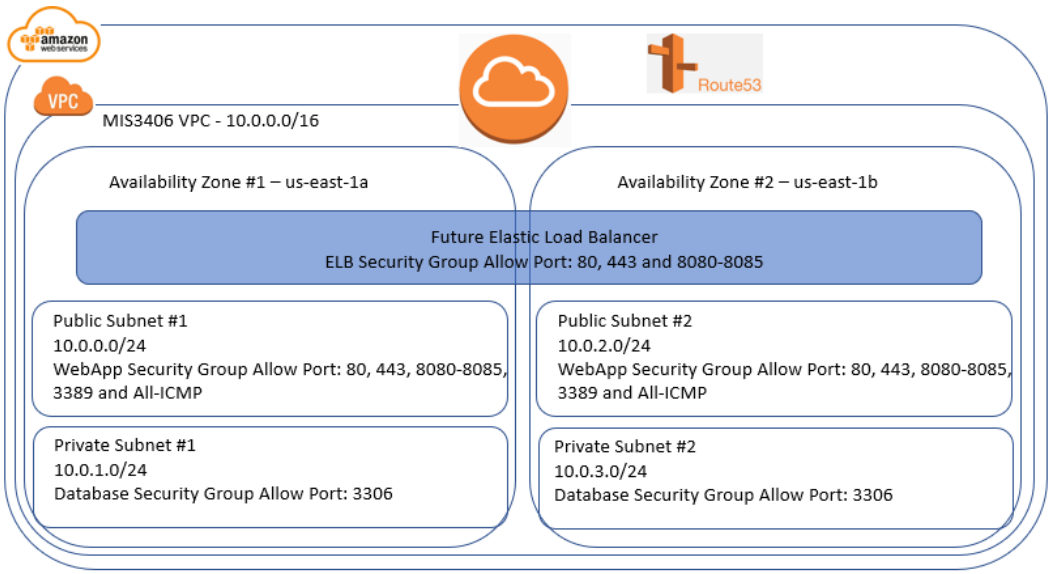

Figure 1. AWS Virtual Network Created in Module 1

In-Class Activity	Overview	Objectives
Creating Your First Instance	In this in-class activity you will create your first instance (e.g., virtual machine) using Amazon Web Services (AWS).	1. Ensure that you can access your AWS account 2. Create your first instance 3. Login to your first instance
Creating Your VPC	In this in-class activity you will get create your first Virtual Private Cloud (VPC). The VPC will span two Availability Zones (AZ). It will include four subnets, two in each AZ. One of the subnets in each AZ will be a public subnet, that is, accessible from the Internet using an Internet Gateway and public IP addresses.	Create a VPC that spans two Availability Zones (AZ) with: 1. A "Public" network segment in each AZ. 2. A "Private" network segment in each AZ. 3. Explore the routing tables for each network segment.
Creating Your Own AMI	In this in-class activity you will create your own Amazon Machine Image (AMI) in the cloud using Amazon Web Services (AWS). You will create and tailor this instance, updating a few settings like opening ports on the Windows Firewall and installing software that could be used as a single instance for an application. You will then create an AMI from this instance. The AMI will be stored in S3. Finally, we will create and launch additional instances from this AMI, demonstrating auto scaling.	1. Learn how to create and use a custom AMI. 2. Learn how to create clones from the AMI.

Table 2. Selected In-Class Exercises from Module 1

Module 2: Building a web-based application with a RESTful API

This module builds skills in support of the learning goal "create a RESTful API using Node.js." Once students have learned how to build a cloud infrastructure on which to run a web application, they learn how to construct a simple web API and web client. The course assumes a working knowledge of JavaScript (a prerequisite for this course is an introductory course in JavaScript), but does not assume prior knowledge of Node.js. Students build a multi-user number guessing game with a web-based form. A server-side application, implemented as a RESTful API, randomly generates a number, keeps track of the guesses, and determines whether the guess is correct. The API runs on a Node.js server using the Express framework. Students start with a basic application that responds to a message from the client through an HTTP POST request. Then they add the guessing game functionality along with the ability to pass parameters from the client to the server. Finally, they incorporate the ability to make database calls using the "sync-mysql" module into the application to track guesses and the "right" answer. An overview of the in-class exercises used in this module is detailed in Table 3.

In-Class Activity	Overview	Objectives
Restful APIs with Node.js	In this in-class activity you will create the first version of a client/server application which uses a RESTful API. In this version of this program, your laptop will play the role of both the client and the server and no data will be passed from the client to the server. At this point this will not be much of a game but will be a stepping-stone to a complete client/server application with data being passed between the client and the server with the server running in the cloud.	Create your first client/server program that uses a RESTful API and provides a stepping-stone for a more feature rich game. Both the client and the server will run on your laptop.
Building out an API – Taking Parameters	In this in-class activity you will create the second version of a client/server application which uses a RESTful API. In this version, your laptop will play the role of both the client and server and data will be passed from the client to the server.	Create your second client/server program that passes parameters using a RESTful APIs and provides a stepping-stone for a more scalable version of the game.
Building out an API – Retrieving Data from an RDS Instance	In this in-class activity you will create the third version of a client/server application which uses a RESTful API. In this version, your laptop will play the role of both the client and server. By the time you have completed this in-class activity, your game will be implemented so that it can scale using AWS cloud services AWS. The main change for GuessANumber3.js is that instead of referencing an array to find the random number for the user to guess associated with that game, we will store and retrieve that number using a MySQL database.	Create your third client/server program that passes parameters using a RESTful API and stores data in an RDS instance instead of arrays providing another stepping-stone for a more scalable version of the game.

Table 3. In-Class Exercises from Module 2

Module 3: Deploying the web-based application to the cloud

This module builds skills in support of two learning goals: "deploy a web-based application to a set of EC2 instances" and "deploy a web-based application using Elastic Beanstalk." Students deploy the guessing game they created in module 2 to the infrastructure they created in module 1. First, students deploy their application to the web and application virtual servers using EC2 and the AMIs created in module 1. This allows students to see how their cloud infrastructure scales and supports a web-based multi-tier application. Second, students use the Elastic Beanstalk service to deploy the server-side API. This allows them to see how cloud services can be bundled to minimize infrastructure management. Table 4 provides an overview of the in-class exercises used in this module.

In-Class Activity	Overview	Objectives
Deploying Cloud Software – Instance-Based	In this in-class activity you will create the fourth version of a client/server application which uses a RESTful API. In this version of this program, you will deploy your web page and your Node.js code to your EC2 instance running in the cloud. In addition, the database will be accessed from your RDS instance. By the time you have completed this in-class activity, your game will run entirely in the AWS cloud and will scale.	Create your next version of the GuessANumber game and learn to: 1. Deploy GuessANumber4.html to the IIS web server running on your EC2 instance. 2. Deploy GuessANumber.js to the EC2 instance. 3. Communicate between your browser and your IIS web server running on your EC2 instance. 4. Communicate between your HTML client page and your GuessANumber.js Node.js server running on your EC2 instance. 5. Communicate between your GuessANumber.js application and the MySQL RDS instance you created in an earlier activity. 6. Use qckwinsvc to run GuessANumber.js as a Windows service on the EC2 instance, making the startup of the game server automatic when the instance starts.
Deploying Cloud Software – Application-Based	In this in-class activity you will create the final version of a client/server application which uses a RESTful API. You will build upon what you have done in class with GuessANumber4. In this version of this program, you will move your Node.js code into the Elastic Beanstalk service. By the time you have completed this in-class activity, your game will operate as before, but the application servers will scale and execute your software using Elastic Beanstalk.	Create your next version of the GuessANumber game and learn how to configure GuessANumber to run under ElasticBeanstalk.

Table 4. In-Class Exercises from Module 3

Table 5 provides a summary of all twelve in-class exercises, and how they map to the learning objectives.

Module	In-Class Exercise	Learning Goals
1	Creating Your First Instance	Understand basic TCP/IP-based networking concepts Understand the economics of cloud services Create and configure Elastic Compute Cloud (EC2) virtual machines
	Creating a VPC	Create and secure an AWS Virtual Private Cloud
	Securing a VPC	Create and secure an AWS Virtual Private Cloud
	Creating Your Own AMI	Create and configure Elastic Compute Cloud (EC2) virtual machines
	Load Balancing, Redundancy, and Auto Scaling	Create a robust server infrastructure using load balancing and auto scaling
	Cloud Storage – Elastic Block Store	Understand the economics of cloud services Create and configure Elastic Compute Cloud (EC2) virtual machines
	Cloud Storage – Relational Database Service	Configure and deploy a database using the Relational Database Service (RDS)
2	RESTful APIs with Node.js	Create a RESTful API using Node.js
	Building out an API – Taking Parameters	Create a RESTful API using Node.js
	Building out an API – Retrieving Data from an RDS Instance	Create a RESTful API using Node.js
3	Deploying Cloud Software – Instance-Based	Deploy a web-based application to a set of EC2 instances
	Deploying Cloud Software – Application-Based	Deploy a web-based application using Elastic Beanstalk

Table 5. Mapping Learning Goals to In-Class Exercises

Class Project

The class project is a four -part, individual assignment that requires students to bring together what they have learned throughout the course. The format of the project largely follows what the students built through the in-class exercises, and each part of the project builds on the previous part:

- In part 1 of the project, build a Virtual Private Cloud in AWS with security groups and public and private subnets.
- In part 2 of the project, build AMIs, launch configurations, load balancers, and auto-scaling groups that will distribute and scale their server capacity up or down depending on load.
- In part 3 of the project, create a "local" version of a web-based, API-driven application. They create the client and server and ensure it runs on their own computer.
- In part 4 of the project, deploy the application to the infrastructure built in parts 1 and 2 of the project.

The application is a fare calculator for a toll highway in Pennsylvania (see Figure 2). Users must select the interchanges where they begin and end their trip and whether they are using EZPass or cash using an HTML form. The user's choices are sent to an API that queries a database and returns the correct toll amount. The toll is displayed in the browser through an alert.

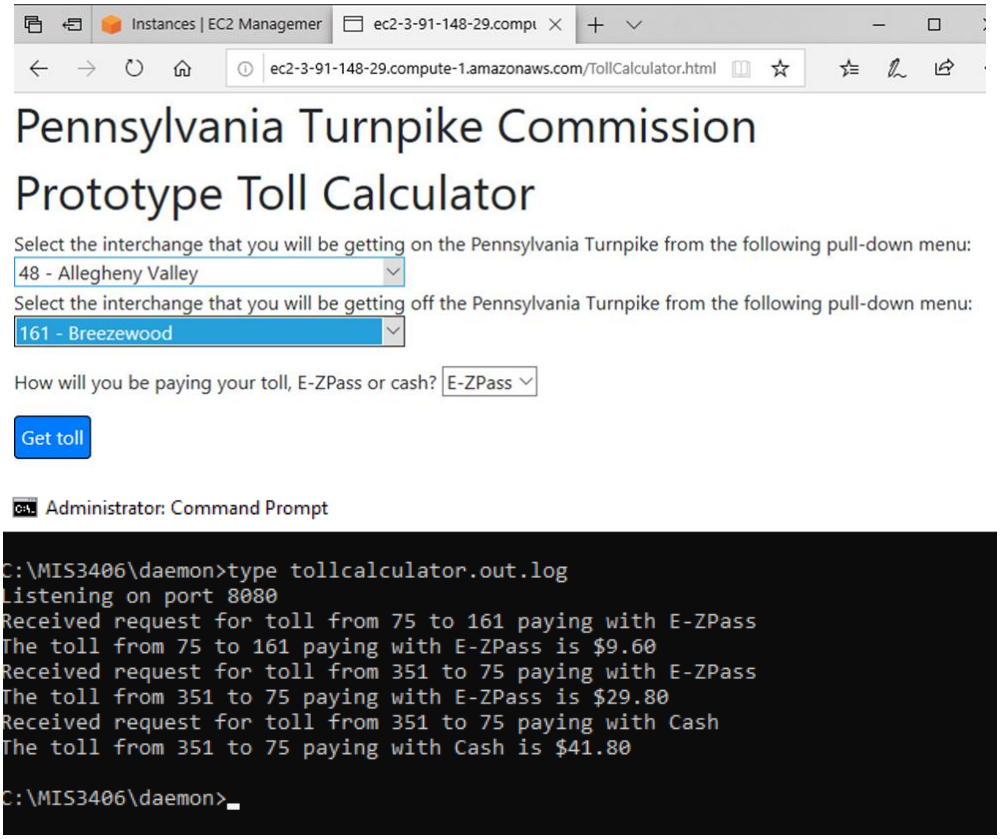

Figure 2: Client and Server Screenshots from the Completed Class Project

The project is more complex than the in-class exercises. In the in-class exercises, the web server and application server are hosted on the same virtual machine. For the project, they implement the web and application servers on separate virtual machines. This added level of complexity requires the students go beyond mimicking the in-class exercises and apply what they have learned to a new scenario.

Each part of the project reinforces one or more of the course learning goals (see Table 6). The exception is the final learning goal – the project does not require students to deploy their web-based application to Elastic Beanstalk. It was decided that there was not time in the course to make this a project deliverable – future versions of the course may integrate an Elastic Beanstalk implementation into the project.

Project	Description	Learning Goals
Part 1	Create a VPC with properly configured network and security groups.	Understand basic TCP/IP networking concepts Understand the economics of cloud services Create and configure Elastic Compute Cloud (EC2) virtual machines Create and secure an AWS Virtual Private Cloud

Part 2	Configure load balancers, launch configurations, auto scaling groups, and RDS	Create a robust server infrastructure using load balancing and auto scaling Configure and deploy a database using the Relational Database Service (RDS)
Part 3	Create the prototype of the application to display toll information between two toll highway interchanges.	Create a RESTful API using Node.js
Part 4	Deploy the prototype Turnpike Prototype Toll Calculator to the AWS VPC created in part 2 so that it runs and can scale.	Deploy a web-based application to a set of EC2 instances

Table 6. Mapping Course Learning Goals to Class Project

Conclusions

Deploying cloud-based, API-driven software applications requires a complex set of interrelated skills that are also in high demand by industry. Cloud architects must understand software development, and software developers must understand infrastructure. This DevOps mindset of tighter integration between software development and operations should be reflected in how we teach these topics in the classroom. We propose a pedagogical approach that integrates networking, cloud infrastructure, and API-based software development. This holistic methodology enables students to see the complete picture of an end-to-end software service solution within the scope of a single course.

The result of this integrated approach is a departure from traditional networking and cloud computing courses because the network infrastructure principles are taught within the context of application development and deployment. Building the course around API deployment has the dual advantage of being a fundamental programming skill for modern web-based application development and a network-centric paradigm that serves as the motivating backdrop for a modern infrastructure course.

References

Alter, S. 2018. "System Interaction Theory: Describing Interactions between Work Systems," *Communications of the Association for Information Systems* (42:9), pp. 233-267.

Chan, R. 2019. "The 10 Most Popular Programming Languages, According to the Microsoft-Owned GitHub," BusinessInsider.com. Retrieved February 25, 2020 from https://www.businessinsider.com/most-popular-programming-languages-github-2019-11.

Hosack, B., Lim, B., and Vogt., W. Paul. 2012. "Increasing Student Performance Through the Use of Web Services in Introductory Programming Classrooms: Results from a Series of Quasi-Experiments," *Journal of Information Systems Education* (23:4), pp. 373-383.

Olsen, T., and Moser, K. 2013. "Teaching Web APIs in Introductory and Programming Classes: Why and How," in *Proceedings of the AIS SIG-ED IAIM 2013 Conference*. 16.

Putano, B. 2019. "A Look at 5 of the Most Popular Programming Languages of 2019," Stackify.com. Retrieved February 25, 2020 from https://stackify.com/popular-programming-languages-2018/.

Tylosky, T. n.d. "What is a full-stack developer?" Thinkful.com. Retrieved February 25, 2020 from https://www.thinkful.com/blog/what-is-a-full-stack-developer-2/.

White, Sarah K. 2020. "The 10 most in-demand tech jobs for 2020 – and how to hire for them," CIO.com. Retrieved February 25, 2020 from https://www.cio.com/article/3235944/hiring-the-most-in-demand-tech-jobs-for-2018.html.

Wikipedia contributors. 2020. "Application programming interface," Wikipedia. Retrieved February 25, 2020 from https://en.wikipedia.org/wiki/Application_programming_interface.

Wyner, G., and Lubin, B. 2011. "From Hello World to Interface Design in Three Days: Teaching Non-technical Students to Use an API," in *Proceedings of the 2011 Americas Conference for Information Systems*, 407.