productivedev

FULL-STACK WEB DEVELOPMENT

USING

.NET CORE,

VUE.JS (TS)

POSTGRESQL

Course Instructor: Wes Doyle

About the Instructor

Hi! I'm Wes - I'm a professional full-stack developer with over 12 years of engineering experience in the life sciences, financial tech, search, and healthcare industries. When I'm not developing, I love to play chess, hike, and learn languages. I'm interested in natural language processing and human learning, and my passion is helping others learn new skills.

The Course

In this course, we'll build a **feature-complete** web application for a coffee roaster called Solar Coffee. We'll build a **management dashboard** consisting of a **PDF invoice generator, dynamic graphs, and an order fulfillment feature.**

As part of this course, you'll learn key full-stack development skills, including:

- Web API development with **.NET Core**
- Front-End development with **Vue.js and TypeScript**
- Using **SQL** with a **PostgreSQL** database
- Unit testing with **Jest**, **xUnit**
- End-to-end testing with **Cypress**
- Finding and fixing bugs
- Web application architecture

High-Level Architecture

It will help to think of the architecture of our application in terms of layers:

- The Presentation Layer
- The Application Layer
- The Data Layer

This is sometimes called the **3-tiered** or **n-tiered architecture**, and is a very common pattern in web development.   There are many other layers to our application, but at a high level, it's worth understanding where each piece of the application we'll build together fits into this model.

Let's look at a summary of our 3-tiered architecture implementation.

Tools of the Trade

For a Powerful Development Environment

(JUST USE WHAT YOU ENJOY WORKING WITH)

The list of suggestions below is for reference - everything listed below are popular, powerful tools. However, to maximize your developer happiness, you should explore and find the tools you love using the most - and master them!

Console

For Windows Users:

cmder
https://cmder.net/

Git For Windows
https://gitforwindows.org/

For Mac Users:

iterm2 *
https://www.iterm2.com/

Developing in C# + .NET

Visual Studio Community Edition
https://visualstudio.microsoft.com/vs/community/

Rider *
https://www.jetbrains.com/rider/

Developing in TS / JavaScript

Visual Studio Code
https://code.visualstudio.com/

WebStorm *
https://www.jetbrains.com/webstorm/

Developing IN SQL

pgAdmin (PostgreSQL specific) *
https://www.pgadmin.org/

DataGrip *
https://www.jetbrains.com/datagrip/

psql *
https://www.postgresql.org/docs/10/app-psql.html

* indicates a tool used by the author in this course

3-Tiered Architecture

Presentation Layer

This is what our users interact with directly. The code in this layer runs *client-side,* or in the browser. Often, it's served to the browser by a server like nginx. It communicates with the application layer by making **HTTP** requests. In our case, this layer is developed in **Vue.js**.

Application Layer

This is where our core business logic lives. This layer contains software that listens for **HTTP** requests from our client and responds accordingly, often performing create, read, update, or destroy (**CRUD**) operations on our database - however, it can do anything we want to run *server-side*! In our case, this layer is built in **.NET Core**.

Data Layer

This is where our data is persisted. We have **SQL relations** (tables) that map directly to our core domain models. Our database responds to queries originating from our Application Layer. Here, we're using **PostgreSQL**.

Section 1: Introduction

Setting up a solid ***development environment***

You'll want to have a good development environment set up before we begin writing code.

All of our programming languages can be run on any platform, but the choice of **IDE** (integrated development environment) for each tier of the project is up to you!

Here are some suggested tools of the trade - but feel free to explore and use what you like best!

Section 2: Back-End Developement

The web API and SQL database that will ***drive our business logic***

Let's dive into creating the back-end of our web application.

In this section we'll walk through setting up a **web API in .NET Core**. We'll be using a lightweight implementation of the **MVC** pattern and separating concerns between multiple class libraries.

It's important to understand that our back end application runs independently of our front end application.

The web API we'll be building will respond to **HTTP requests** from the front end application, handle all of our core business logic, and perform create, read, update, and delete (**CRUD**) operations on our data, which lives in a **PostgreSQL** (**SQL**) **database**.

We'll use EntityFramework Core as an **ORM**, or object-relational mapping, which maps objects in C# to corresponding tables in our SQL database.

Section 3: Front-End Developement

The web application that will provide ***visibility*** *and* ***management tools*** *for our company.*

We'll build all of the code that our users will interact with in the browser using **Vue.js**.

Vue is a web framework that makes it easy and fun to develop component-based, modular code that we can use to compose a user interface for our application.

We'll be using **TypeScript**, which allows us to optionally use static typing with Javascript. Ultimately, our application is transpiled to vanilla JS using **Babel**. Luckily, the Vue CLI makes all of this relatively effortless!

Section 4: Full-Stack Integration

Getting all of the pieces wired together, debugging, and feature development

Now that the basic building blocks have been laid out, we'll continue on with development by adding core features to our application.

We'll also encounter and fix several important bugs along the way.

Learning to debug software locally, finding the source of bugs, and fixing them is a core skill for any software developer. It's often overlooked in online courses, but it will be an important part of our journey!

Section 5: Unit and Automation Testing

Test-driving our business logic and our user behavior

We'll look at how to use **xUnit** to write unit and integration tests for our back-end code.

Jest is our test framework of choice for the front end.

Finally, we'll use the powerful **Cypress** library to set up end-to-end tests, which will drive our browser as we program user interaction tests directly into our codebase!