**Introduction to Software Development**

1. **Overview of Web and Cloud Development**

When you’re starting out as a Web Developer, it can be difficult to determine what you need to learn and what order you should learn it in. Understanding how familiar websites are constructed and delivered to you is a good starting point. Let’s review the basics of how you interact with a website. You launch an internet browser – there are lots available: Google Chrome, Microsoft Edge, Mozilla Firefox, and Apple Safari are some of the most popular. The browser has an address bar, into which you enter a URL, like www.IBM.com. The browser then contacts the server with the name and requests the information that makes up the website.

The server then sends a response, which contains the data that the client requires to display the website. For most websites, the server will return: HTML, which defines the structure of the page, but doesn’t look very attractive CSS, which adds style and flair to the page and JavaScript, which adds interactivity and dynamic content. Content displayed by websites can contain elements that are either previously stored on the server (called “static”) or generated each time they are requested by the client (called “dynamic”). Dynamic elements can involve information coming from other systems and applications, such as databases. Most websites contain static and dynamic elements to provide the best user experience.

Cloud Applications are similar to Websites in that they request content that a server returns. Cloud Apps are built to work seamlessly with a Cloud-based back-end infrastructure, Cloud-based data storage and data processing, and other Cloud services, making them very scalable and very resilient.

The environment for building websites and Cloud Applications is divided into two primary areas: front-end and back-end. The front-end deals with everything that happens at the client-side – everything the user can see and interact with. You can choose to specialize in front-end coding, using HTML, CSS, JavaScript and related frameworks, libraries, and tools. The back-end deals with everything that happens on the server before the code and data are sent to the client. The back-end coding usually handles the logic and functionality that make the website or app work, and the authentication

processes that keep data secure. Back-end developers may also work with relational or noSQL databases, even collaborating with database administrators in bigger projects. Full-stack developers have skills, knowledge, and experience in both front-end and back-end environments.

Whichever way you choose to specialize, you will need the appropriate tools to help you work. The first tool most developers add to their resources is a code editor. Developers also need tools to integrate, build, compile, and debug code. Integrated Development Environments or IDEs incorporate some of these additional capabilities beyond just code editing and make it easier to build and manage your code. Good IDEs support multiple languages and integrate with management and storage tools like Git and GitHub. Other useful features are custom extensions and themes for supporting your working style and environment. Examples for code editors and IDEs include Sublime Text, Atom, Vim, VS Code, Visual Studio, Eclipse and NetBeans.

In this brief overview, you learned about: The basic communication between client and servers. How websites are built and displayed. Front-end development is about what happens on the client side. Back-end development is about what happens on the server. Full-stack development includes both front-end and back-end development. IDEs will help you create and manage your code.

1. **Learning Front End Development**

Internet websites offer lots of different services, one of the most popular being online shopping. When you explore an online shopping website, navigating through pages, choosing different product categories, or comparing products, you are interacting with the front-end of a website. Let us see how the front-end of a website is developed by front-end developers. For this, we need to understand how a website is made. To create a website, web developers usually use Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. These languages are designed to work in conjunction with each other. HTML is used to create the physical structure of a website.

The physical structure contains elements such as text, links, images/videos, page dividers and buttons. The HTML code ensures a proper formatting of all text and image elements so that browsers display the page consistently. A website is like a house which has only been constructed. Just like we need interior designers to add style to a space, we need front-end developers to add the necessary glamor and appeal to a website. When you order products from any website, you realize that the pages have a pleasing font, attractive colors, and are easy to navigate. Developers use CSS to create stylish websites.

CSS provides front-end developers with a standard method to define, apply, and manage different sets of style characteristics for a website and each of its components. CSS ensures uniformity in look and feel, style, colors, fonts, designs and layouts. So, HTML is used to create the structure and CSS is used to design it and make it appealing. CSS is also used to create websites that have cross browser compatibility which means that they are compatible with multiple browsers and multiple devices such as PC, mobile devices, iPads etc. Online shopping websites are intuitive, interactive and quick to load. This is where JavaScript comes into the picture. JavaScript is an object-oriented programming language that is used in conjunction with HTML and CSS to add interactivity to a website.

For example, you use HTML to add a login button to a page, and CSS to style that button. You then use JavaScript to add log-in functionality to that button. A new front-end development language is Syntactically Awesome Style Sheets called SASS. It is an extension of CSS that is compatible with all versions of CSS. SASS enables you to use things like variables, nested rules, and inline imports to keep things organized. SASS allows you to create style sheets faster and more easily. Another language that is being used now is Learner Style Sheets or LESS.

LESS enhances CSS, adding more styles and functions. It is backwards compatible with CSS. Less.js is a JavaScript tool that converts the LESS styles to CSS styles. Using all these languages, websites are designed as reactive and responsive. Reactive or adaptive websites display the version of the website designed for a specific screen size. For example, a website can provide more information if opened on a PC than when opened on a mobile device. Responsive design of a website means that it will automatically resize to the device it is being accessed from.

For example, if you open up a products website on your mobile device, it will adapt itself to the small size of the screen and still show you all the features. A JavaScript framework is an application framework that is written in JavaScript. Programmers can manipulate the different functions, use them wherever required and can create device responsive applications. A few examples of several frameworks being used are:

Angular framework: an open-source framework being maintained by Google. Angular frameworks allow websites to render the HTML pages quickly and efficiently. It has built-in tools for routing and form validation.

React.js has been developed and maintained by Facebook. It is a JavaScript library that builds and renders components for a web page. It is not a complete suite of tools. For example, routing is not a part of this framework and will need to be added using a third-party tool. React.js only helps build and drop components into a page.

Vue.js is maintained by the community and its main focus is the view layer which includes user interface, buttons, and visual components. It is flexible, scalable and integrates well with other frameworks. It is very adaptable. It can be a library, or it can be the framework.

The task of a front-end developer evolves continuously. The technologies are upgraded constantly and so front-end developers needs to keep upgrading the websites that they create. The websites that they create should work in multiple browsers, multiple operating systems and multiple devices.

1. **The importance of back end Development**

Welcome to the importance of back-end development. A front-end developer creates websites and Cloud applications using HTML, CSS, and JavaScript to create what the user sees and interacts with in the client's software. A back-end developer creates and manages all the resources that are needed to respond to the requests that the user makes through the client. The back-end developers tasks focus on enabling this server infrastructure or back-end to process requests, supply data, and provide other services securely. Front-end and back-end developers must work together very closely. Each needs to understand the requirements of the solution and how their respective parts will interact before the development process can begin. Throughout the life cycle of the website or Cloud app, front-end and back-end developers collaborate to resolve issues and add functionality.

How does the work of a back-end developer affect you as you are browsing the Internet or using a Cloud App? Think about it like this. When you're shopping online, what happens to the data you enter? Your login information, your product searches, your payment info. The back-end processes all of these things. The back-end developers write and maintain the parts of the application that process the inputs. Let's think about your experience as you explore an online shopping site and make a purchase.

As you search for products, your search request is submitted to a web application, which then retrieves the data from a separate database and serves it back to the client for display. To facilitate this, a back-end developer must understand the language that the web application uses, how to query the database for the correct data, and how to bring the two together. Even a simple task like navigating around the site can require the skills of a back-end developer. Many sites have restricted areas that are only available to users who have an account and have logged in. User account management, authentication and authorization can be the responsibility of the back-end developer too. Once you've decided what you want to purchase, you must add it to your card and make a payment. The purchase process requires you to submit sensitive information such as your address and credit card number.

The back-end developer must ensure that this data is securely handled and stored. Front-end client interactions, whether a request for data, like an image, accepting input from a user filling out a form, or securing sensitive information like a credit card number, all require different services from the backend server. Each request needs to interact with the back-end in a different way. Back-end developers use APIs, routes, and endpoints to process incoming requests. An API is code that works with data, usually using JSON or XML. APIs have set rules and structure. A route is a path to a website or page that the user interacts with.

Routes generally take user input and show results based on the input and end point maybe an API or may simply be a path. When a request from the front-end arrives at the back-end, it is routed to the correct service. If the back-end has an endpoint defined for the request by using routing, the request will be addressed and replied to. If the endpoint is missing, the server returns a 404 error. Back-end developers must create and maintain this server side routing. Along with back-end APIs, routes effectively allow the front-end client to plug into the correct socket at the back-end. APIs provide a mechanism for Cloud apps, mobile apps, and other types of software to access resources from the back-end.

To perform all this back-end development, you will need to be familiar with at least one back-end language and its associated frameworks. Among the most popular languages today is JavaScript, which was originally designed to run in web browsers, adding extra interactivity and dynamic content to web pages. JavaScript is also being used on the back-end with new releases adding server-side functionality. JavaScript has many frameworks, but two of the most well-known are Node.js and Express. Python is another popular language. It's very flexible and easy to learn. Python has wide functionality.

It can be used for everything from creating webpages to connecting to a database, to performing data analysis. Two well-known Python frameworks are Django and Flask. Back-end developers often work with data and databases. You will also benefit from learning some SQL. To help handle requests from databases, back-end developers can use object relational mapping tools or ORM to connect to the databases and retrieve the correct data. Although an ORM can hide some of the complexity of querying databases, it's useful to understand the fundamentals of databases so that you can troubleshoot any issues that arise. The day-to-day tasks of a back-end developer focus on the behind the scenes functionality that keeps websites, Cloud apps, and mobile apps up and running.

Back-end development covers a wide range of technologies, from managing user accounts, authentication and authorization to ensuring that sensitive data is stored and transferred securely. Back-end developers also work with databases, retrieving, processing, and storing data as required. Life for a back-end developer is very challenging and ever-changing.

1. **Teamwork and squads**

By definition, a team is a group of people working together towards a common aim. Within a team, you’ll find a range of different people with different skills, experience, and talents. Each person can give their attention and effort to the things that they are good at, and by working alongside others on tasks outside their current repertoire, they can expand their skillset. Working in a team promotes creativity. Collaborating with others gives you the opportunity to discuss ideas and challenge one another’s thinking about a subject.

Good teamwork is empowering: positive attitudes and behaviors can impact the rest of the team and create positive results. Working well as a team doesn’t always come naturally, but there are some things you can consider to help your team succeed. Each member needs to trust and respect the other members of the team. This generally comes with time, but depends on all of the members contributing equitably. Defining and agreeing on goals for a project is essential so that the whole team knows what they are working towards. And you also need to define and agree on roles to avoid any duplication of effort or missed tasks. Working with each members strengths is important to make the most of the talent within your team, as is celebrating success and analyzing problems.

Communication is vital in a team environment. Ensure that you choose a method that works for everyone so you know that the whole team is seeing and responding to information. So, what does teamwork look like in software engineering? Teams often start projects with a kick-off meeting where they plan how they will complete the project, assign tasks, and agree on goals. Throughout the lifetime of a project, you’re likely to have whole team and/or sub team meetings to review progress and plans. Design and code reviews can be requested at the team level and undertaken by whoever has availability at that point in time. Team members might present walkthroughs of their sections of responsibility to the rest of team so that the whole team has oversight of all parts of the project.

And, key team members will likely present walkthroughs to stakeholders at various times during the project. When a project is complete, retrospective meetings may be held to review what went well and what could be improved in future projects. You may have a mentor who may or may not be in your current project team. You might also be asked to be a mentor. Sometimes team mentoring is used so everyone can learn from each other. Some groups also have teams working on internal projects such as defining code standards, maintaining or updating legacy cross-project code, or reviewing potential new software for usefulness to the team. Good teamwork can bring many benefits to a project.

Working alongside others can encourage creativity and enable you to take advantage of each person’s strengths while also allowing them to gain knowledge and skills from other members of the team. When working as part of a team, software engineers are more likely to adhere to corporate coding standards and regularly document their code. The additional accountability that teamwork creates results in better quality code, fewer bugs, and more maintainable code. From a software engineer’s point of view, working as a team can reduce stress because there’s always someone to turn to and get help from. And having someone to discuss problems with can help you to increase your understanding and resolve more issues by yourself. By working in a team, each member has a greater idea of the bigger picture, resulting in a more coherent overall solution. Some organizations that follow Agile development methodologies may call a team a squad.

Typically, a squad is a small team of up to 10 developers. It is likely to consist of: A squad leader who acts as the anchor developer and coach for the squad. And a few software engineers who develop and implement the product features and test cases. It may also include one or two user experience developers or designers. In some squads the developers may work together in pairs to practice pair programming.

1. **Pair programming**

Pair programming is an extension of teamwork where two developers work side-by-side at one computer. They can either be physically at the same computer or work virtually via video link or shared screens. The first is the preferred option, but virtual pair programming can also be productive. Pair programming is a type of Agile development where two developers can plan and discuss their ideas continually as they create a solution, generally resulting in a better end product. There are various styles of pair programming:

Driver/navigator style is the most common style, where one developer is the driver, typing in the code, and the other is the navigator, reviewing the code as it’s written and giving directions where to go next. The navigator also keeps an eye on the bigger picture of the overall solution. When working in this way, it’s important to regularly swap roles to keep both of the pair engaged across the whole task.

Ping-pong style incorporates test-driven development. For each task, one developer writes a failing test and then the second developer writes code to pass that test. For each new task, they swap roles, so regularly changing who writes the test and who writes the implementation. The two developers work together at the end of each task refactoring the successful code to refine and improve it. Strong style pair programming is a good way for junior software engineers to learn from more experienced ones.

The defining rule for this is that for an idea to go from your head to the computer, it must go through someone else’s hands. So, the more experienced of the pair is the navigator and the driver learns from witnessing their implementation and thought processes. For this to work well, the driver shouldn’t challenge any ideas until the full implementation is complete so as not to interrupt the flow of ideas from the navigator. Now pair programming has many benefits. It’s a good way to share knowledge and skills from one developer to another or between the two and a great way for a new team member to get up to speed on a project. As well as enhancing the technical skills of the pair, it’s also a good way of building soft skills such as communication and problem solving. Having two sets of eyes on the code often results in fewer typos, logic errors, and bugs.

And it enables code reviews to be done on the fly. While this doesn’t replace formal code reviews, it does enable another layer of review. Having two people thinking about a problem can result in multiple initial ideas, but is likely to result in the optimal approach being chosen earlier in the process. And although pair programming can take longer than individuals writing the same solutions, it’s likely to result in better code with less time spent reviewing, testing, and bug fixing. There are also a few challenges to overcome. Working in a pair requires long periods of focus which can be exhausting for the two programmers. And personal or other work commitments can impact the pair schedule.

Sometimes one of the pair can end up controlling the entire process, resulting in a more typist/programmer pairing which doesn’t benefit from any of the positives of pair programming. And at other times, individual personalities may not work well together. When multiple sets of pair programmers are present, their discussions can result in a noisy environment for the other workers in the room.

1. **Introducing Application Development Tools**

Getting your Cloud App from the ideas stage to fully formed, written, and deployed is a long process, but there are many tools which will help you along the way. A cloud application developer’s workbench includes: Version Control, Libraries, and Frameworks. When many developers are working on the same project, knowing what order changes were made, thereby creating a new version of the source code, becomes overridingly important. Version control systems keep track of what changes were made when and by whom and resolve any conflicts between changes. For developing your code version control can be useful even when you are the sole contributor on a project. Properly used, it can give you a way to revert to an older version of your code if something goes wrong and gives you some basic information about how the code developed over time.

Version control functionality is generally tied to the storage system you are using, which is why a code repository is recommended, even for beginners. Git and GitHub are extremely popular for source code storage and management. Git stores files in repositories where you can track changes, split code into different branches for more focused development, and then merge them back into the main body of code. Libraries are collections of code, like standard programs and subroutines, that you can use within your code. For example, you might want to include a navigational feature, like a carousel – a code library can supply you with the code for that so that you don’t have to spend the time and energy creating one from scratch. Being able to reuse code in this way makes developing your app much quicker and easier. Multiple code libraries can be integrated into your existing project.

As you discover a need for a specific function or feature, you can research an appropriate library. You determine when to call the required method as needed. The control returns to the program flow once the subroutine is finished. When you use a code library, you are in control. Code libraries are generally used to solve a specific problem or add a specific feature set. Either way, there are lots for you to choose from, so do your research. Here are some examples of code libraries:

* jQuery is a JavaScript library that simplifies DOM manipulation.
* Email-validator is a small library that checks an email address is correctly constructed and valid.
* Apache Commons Proper is a repository of reusable Java components.

Frameworks provide a standard way to build and deploy applications. You can think of a framework as being a skeleton that you can extend by adding your own code, providing a scaffold on which to build your apps. The framework you intend to use must be determined early in your development planning and used right from the beginning. New frameworks can’t be incorporated into an existing project. Your chosen framework dictates the architecture of your program and controls the program flow.

The framework determines which subroutines and methods will be called when. When working with a framework, there is a specific structure that you must follow. The framework calls on your code, rather than you calling on the framework. Frameworks are less flexible than libraries, allowing you less control, but they do provide good standardization and can help you create efficient code. To use an analogy, if you are a carpenter building a house, the framework is the frame that you add to – bricks on the outside, plasterboard on the inside, and so on. The frame acts as a guide for how the house is constructed. Here are some examples of frameworks:

* AngularJS is a JavaScript-based framework for dynamic web applications.
* Vue.js is a JavaScript framework focused on the user interface.
* Django is a framework that uses Python for web development.

Frameworks define the workflow that you must follow, unlike libraries, which allow you to call functions as and when required. When using a framework, it can sometimes feel like you, as a developer, are not in full control of the development process. This sense of the framework and its predefined workflow controlling the development process is referred to as inversion of control. Frameworks that have a lot of control are known as opinionated – they have opinions on how their workflow should be used and remove a lot of the decisions you would otherwise have to make about how code is written, the location of files, and even file names. Frameworks often include their own libraries, which they call when needed. Inversion of control allows you to create standardized apps, and takes away a lot of the tedious configuration work, so you can focus on the code for your app.

1. **More Application Development Tools**

Let’s look at some tools which can help you get your app built and deployed: CI/CD, Build Tools, Packages, and Package Managers.

CI/CD refers to the practices of continuous integration and either continuous delivery or continuous deployment. CI/CD is a best practice for devops teams enabling developers to deliver frequent changes reliably. Implemented through a build-automation server, Continuous Integration (CI) ensures that all the code components work together smoothly. A CI build environment enables you to integrate newly developed code frequently, at least every day, if not every hour, depending on how quickly the project changes. Continuous delivery (CD) begins where CI ends. The CI process automatically builds and tests your code, then CD deploys all code changes in a build to a testing or staging environment.

A build tool transforms your source code into the binaries needed for installation. Build tools organize your source code, set compile flags, and manage dependencies. They are most important in environments where there are many inter-connected projects, with multiple developers contributing to each project. In these environments, it can be very difficult to keep track of what changes were made, in what order, what dependencies exist, and what needs to be incorporated in the next build, so automation is key to keeping everything running smoothly. Build automation can automate a wide variety of tasks that developers do in their day-to-day activities like: Downloading dependencies. Compiling source code into binary code. Packaging that binary code. Running tests. And deployment to production systems.

You can initiate a build from the command line or from an IDE. There are two categories of Build Tools widely in use: Build-automation utilities, which generate build artifacts like executables, by compiling and linking source code. Build-automation servers, which execute build-automation utilities on a scheduled or triggered basis. Some examples of build tools are:

* Webpack – a module bundler for JavaScript. Babel – a JavaScript compiler.
* And Web Assembly - a binary instruction format that runs in your browser.

Now that you have your app developed and tested, you’re ready to deploy. But how does that happen? The app needs to be simple and trouble-free for the user to install, so a commonly used technique is to collect all the necessary files and bundle them together into a package. Packages are archive files that contain the app files, instructions for installation, and any metadata that you choose. They have their own metadata too, including the package description, package version, and any dependencies, like other packages that need to be installed beforehand. Once you have bundled your app into a package, you can use a package manager to distribute it.

Package managers take care of the tasks of finding, installing, maintaining or uninstalling. software packages at the user's request. Package management systems:

* Coordinate with file archivers to extract package archives.
* Verify checksums and digital certificates to ensure the integrity and authenticity of the package.
* Locate, download, install, or update existing software from a software repository.
* And manage dependencies to ensure a package is installed with all packages it requires.

Some commonly used package managers for each of the major platforms are listed here: On Linux - Debian Package Management System (DPKG). Red Hat Package Manager (RPM).

On Windows - Chocolatey. On Android - Package Manager. On MacOS - Homebrew and MacPorts.

Any libraries or utility code that is developed as part of the application is managed with the cloud application package managers. Here are some examples of package managers for popular languages: For Node.js/Javascript - npm. For Java - Gradle and Maven. For Ruby - RubyGems.

For Python - Pip and Conda.

1. **Software Stacks**

A software stack is a combination of technologies that includes software and programming languages. Developers use a software stack to create applications and solutions such as web and mobile apps. The set of individual technologies is stacked in a hierarchy and work together to support the execution of an application. The higher levels in the stack provide tasks or services for the user and the lower levels interact with the computer hardware. Software stacks typically include: Front-end technologies such as programming languages, frameworks, and user interface tools.

And back-end technologies such as programming languages, frameworks, web servers, app servers, operating systems, messaging applications, and databases. You might hear the term technology stack used in place of software stack. However, a technology stack is a broader term that includes hardware and infrastructure like virtual machines, containers, storage, and load balancers, as well as the software stack. The simplest implementation of a software stack consists of a presentation layer, a business logic layer, and a data layer. However, more complex applications use more complex stacks, which could include software for virtualization, scheduling and orchestration, runtime environments, database connectivity, networking, and security. The software and services that make up a stack can be from a variety of sources: from internal resources, to third party providers, to cloud providers. There’s no formal definition of the structure of a stack, the only rule being that the software and services included must support an application’s development, functionality, or deployment.

When you’re using a software stack, you don’t have to use all of the available layers, you only need to use those which are relevant to your solution. There are many different examples of software stacks, The Python-Django stack, uses the popular Python programming language alongside the Django web framework. This combination is all open source and commonly used for large-scale, fast-changing web applications. The Ruby on Rails stack, uses the Ruby programming language with a server-side web application framework. Ruby on Rails is great with JSON or XML for data transfer and HTML, CSS, and JavaScript for front-end development. And the ASP.NET stack includes Microsoft technologies such as the ASP.NET MVC framework, the IIS web server, SQL Server, and Azure.

The LAMP stack runs on the Linux operating system. It uses the Apache HTTP or Web server, MySQL databases, and the PHP programming language. LAMP is an example of an early incarnation of a software stack designed for building websites and cloud applications. All its constituent parts are open source and loosely coupled, so it’s easy to swap different options into the stack. For example, you could choose to use PostgreSQL instead of MySQL for your database server, changing the LAMP stack to be the LAPP stack. Similarly, you could use the Python programming language instead of PHP. The MEAN stack uses a MongoDB database with an Express.js web application server framework, the Angular.js framework for front-end JavaScript development, and the Node.js platform for server-side scripting.

The MEAN software and services are platform agnostic, free, and open source. There are other stacks related to the MEAN stack, including: The MERN stack which replaces Angular.js with React, and is a flexible and high-performing framework for developing front-ends. And the MEVN stack which replaces Angular.js with Vue.js. Vue is a lighter-weight JavaScript framework with less capabilities, but it can provide better performance than Angular.js. Let’s consider some advantages and disadvantages of three different software stacks: MEAN, MEVN, and LAMP. MEAN is a free and open-source JavaScript software stack used for building web applications. The biggest advantage of the MEAN stack is that all of the parts use JavaScript, so developers only need to know a single language. The stack is also open source which means the cost is lucrative to businesses and there is a lot of documentation and re-usable code for developers to use. Development can happen quickly because Node.js has a huge collection of free, reusable module libraries.

However, the MEAN stack may not be well-suited for large-scale applications. When using Express.js, the business logic often resides on the server preventing the reuse of some services like batching operations. And MongoDB is great for unstructured data, but it doesn’t provide the same level of functionality as a relational database. The MEVN stack is a web stack like MEAN, but it uses Vue.js instead of Angular.js for user interfaces. MEVN and MEAN stacks have similar advantages, but Vue.js is a much newer technology and doesn't have as many reusable libraries as Angular.js. Like MEAN and MEVN, the software and services in the LAMP stack are open source meaning there are lots of reusable chunks of code available to the developers. And because LAMP is one of the oldest software stacks it’s easy to find support and reusable solutions. However, because the Linux operating system is an integral part of the stack, it isn’t as flexible as MEAN and MEVN which are platform agnostic. MySQL is a relational database, so the stack cannot take advantage of unstructured data. The other disadvantage of the LAMP stack is that the back-end runs on PHP, Perl, or Python, whereas the front-end uses JavaScript and HTML, making it harder for developers to switch back and forth than the MEAN and MEVN stacks which use JavaScript throughout.

In this video, you learned that: Software stacks are a combination of technologies for creating applications and solutions. Software stacks can range from simple three-layered systems to many layers. There are numerous types of software stacks for different developers and environments. The biggest advantage of the MEAN stack is that all of the parts use JavaScript, so developers only need to know a single language. And, the LAMP stack on Linux works well for relational data.

Summary & Highlights

Congratulations! You have completed this module. At this point, you know:

* How websites are built and displayed, and how they communicate with the back-end servers.
* How different front-end technologies work together to create reactive and responsive websites.
* How back-end development covers a wide range of technologies including business logic, security, and database access.
* Effective teamwork can result in better quality code with fewer bugs, better-skilled team members, and less stress for everyone.
* Pair programming is a great way to share knowledge and skills between developers, resulting in better solutions and improved efficiency.
* You can use developer tools to track who makes what changes to your code withversion control software, access libraries of reusable code, and use frameworks to build and deploy applications in a standard way.
* CI/CD tools, build tools, packages, and package managers help you build and distribute your applications.
* A software stack is a combination of technologies for creating applications and solutions.