Web Fundamentals

1) The World Wide Web is a series of web pages connected by hyperlinks which are spread all over the World. Clicking on these links allow up hop from one web page to another or we could directly key the url into our web browser.

   All manner of users & their computers/servers contribute to the World Wide Web rather than a sole company. Just like "the Cloud" you can't really map out the World Wide Web since websites are constantly being added and removed from it. It was created by Sir Tim Berners-Lee in 1989, he based the link structuring and referencing idea on some Victorian book, I think it was an obscure encyclopaedia. Main reason it took off compared to rival systems is Sir Tim insisted on making his idea free rather than a proprietary concept. Sir Tim admitted to earning nothing from having invented the www system we're all familiar with.

   Web pages have URIs which are Universal Resource Identifiers that identify specific files such as audio, visual and document files eg pngs, mp4s and pdfs. If you want to locate these resources you use URLs which are Universal Resource Locators - the web address that you want to click on to locate something.

   For example here's a file I wrote teaching myself about arrays in Rust.
   https://github.com/markpackham/rustLearn/blob/master/src/arrays.rs

   To access it we have the https protocol (essentially a "Secure" certificate approved version of the Hyper Text Transfer Protocol). We need standardized protocols so one computer can communicate to another about the material they wish to grab from one another without all manner of conflicts. Another protocol example is FTP which is File Transfer Protocol which you'd use if you wished to moves files to and from you server and local device.

   github.com is the domain name of where all the content is hosted.

   The actual Rust file itself is the arrays.rs part at the very end of the link.

   In this example we don't see the port being used but that's pretty common, you usually don't want everyone knowing what ports you do and don't have open or they're ones you're already familiar with such as port 80 for http and port 443 for encrypted networks.

   Most web pages are created with Html, that's Hypertext Markup Language which end with the file name .html and have <html></html> as the opening and closing tags within the file.

2) Front-end and Back-end functional differences

Front-end – code usually written in JS aka “JavaScript”, WebAssembly (for people less keen on using JavaScript although it’s still young in relative terms), Html & CSS (Cascading Style Sheets) and the focus is on the site visitor’s web browser where they have the power to interact with the material presented in front of them (eg clicking on a video so it plays or a web page that shows more text). The front end focus is content that is presented to the end user. Going off stereotypes front end devs tend to focus more on making stuff pretty and more user friendly (eg a crazy amount of time is dedicated to positioning banner images just the right number of pixels to the left or right in CSS).

Front end devs often use frameworks such as React, Vue, Svelte and Angular so there are standards on how files are structured and efficiently cached/compressed. If you have a powerful computer with excellent caching & space then having as much stuff done on the front end/client side as possible is great performance wise (the less requests you need to make to the server the better) – less so if you’re using a very weak mobile device with storage issues.

Back-end code is usually written in Python, PHP, JavaScript using NodeJS, Java and Ruby. You encounter frameworks like Django, Laravel, Express, Spring and Ruby on Rails. The back end is the behind-the-scenes magic that the end user doesn’t see & shouldn’t be able to access. The last thing you’d ever want is a random site visitor chucking SQL queries at your database or being able to retrieve sensitive data from your servers.

Back end devs focus on handling requests from the front end and responding to them with the necessary parts from the back end such as retrieving a relevant bit of data from the database or carrying out some elaborate business logic that would be too slow or insecure to perform on the front end.

A full stack developer is someone who carries out both back-end and front-end tasks. This is more common nowadays given JavaScript can be used both in the front end and in it’s Node.js form at the back end. JavaScript also looks a great deal like PHP, Java, C++, C# & C so it’s not as hard for a front end dev to learn backend programming. The creation of Typescript also makes JavaScript more palatable to people who love Strict Typing such as Java programmers.

3) What happens at the backend when I enter a Google query.

From the front end I can see the text I enter into Google. As I’m typing Google’s running through various algorithms looking at the words I’m using and trying to predict what I’m asking it. The backend of Google is fetching common search recommendations from databases and passing them onto the front end to assist me with what I’m asking Google.

When I hit return after entering my query the query is sent to Google's back-end servers who attempt to find relevant information on sites I should be visiting from their databases, the information is then sent from the back end to the front end with the list of recommended sites that I should visit. Google factors in my IP address and language I'm using so the primary information presented to me will be in English and have a bias towards the United Kingdom. A lot of the complex decision making is done at the back end as well as the bulk of the security so I can't just inject malicious code into Google's servers when carrying out a query.

4) MERN

Mongo DB a non-relational (non ER) document focused database that uses JSON based data storage (so a lot more forgiving if you have to make changes to the database design than SQL databases like MySQL). When you query Mongo DB it looks like you're dealing with JavaScript objects. MongoDB lives on the server side so a typical user shouldn't be able to directly access it.

Express is an unopinionated server sided web framework, it's very similar to Python's Flask or Ruby's Sinatra (I think it was heavily influenced by Sinatra). By unopinionated I mean it allows a great deal of flexibility in how you configure it compared to something with very strict rules on how it should be structured and configured like Ruby on Rails. You'd have Express handle requests & responses between the front end such as React and the back end such as MongoDB. Express is viewed as part of the server side/back end.

React is a front-end JavaScript based framework that calls itself a library but is more elaborate than a typical JavaScript library like jQuery. If we were basing an application off a Module View Controller design then React would be the View aspect or the presentational aspect that the client would be interacting with. It was created by Facebook and is the most wildly used front end framework (quickly dethroning Google's more complex, opinionated & older Angular front end framework). There's also a mobile focused version called React Native. React is viewed as part of the client side/front end.

Node.js is the runtime environment for JavaScript that lets JavaScript go beyond the browser and carry out server-side tasks. Like Java's virtual machine you can install Node.js on various operating systems be they Windows, Mac or Linux.

In the MERN stack the site visitor interacts with the React end. React then passes requests through Express which retrieves the relevant information from the Mongo DB databases, Express then passes that information back to the site visitor via React. React and Express run off Node.js.