**Front and Back**

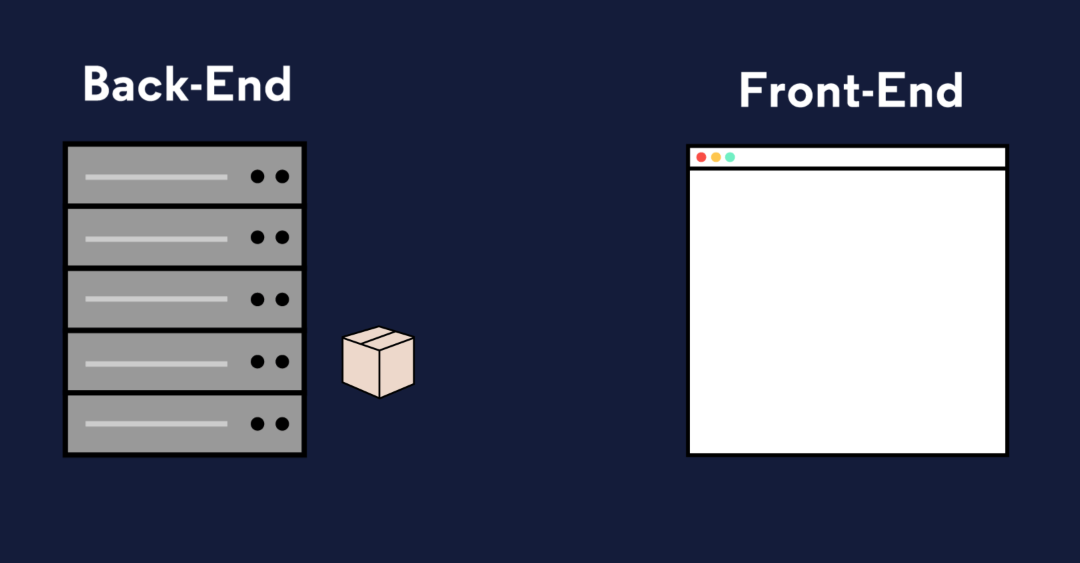
In this lesson, we’ll explain what makes up the back-end of a web application or website. The back-end can feel very abstract, but it becomes clearer when we explain it in terms of the front-end! To oversimplify a bit, the front-end is the parts of a webpage that a visitor can interact with and see.

Various tools and frameworks can be used to make a webpage, but, at its core, the front-end is composed of JavaScript, CSS, HTML, and other *static assets*, such as images or videos. Static assets are files that don’t change. When a visitor navigates to a webpage, these assets are sent to their browser.

Visiting a simple website is like ordering delivery from a restaurant: we place an order for our meal, and, once it’s delivered to us, we have it entirely in our possession. In this analogy, we can think of the front-end as everything that’s dropped off with the delivery: the containers, the utensils, and the food itself.

You’ll sometimes hear front-end development referred to as *client-side* development. Our instinct might be to think of the client as the human visitor or user of a website, but when referring to the client in web development, we’re usually referring to the non-human requester of content. In the case of visiting a website, the client is the browser, but in other circumstances, a client might be another application, a mobile device, or even a “smart” appliance!

While the front-end is the part of the website that makes it to the browser, the back-end consists of all the behind-the-scenes processes and data that make a website function and send resources to clients.



# So What is the Back-end?

When a user navigates to google.com, their request specifies the URL but not the filename for today’s [Google Doodle](https://en.wikipedia.org/wiki/Google_Doodle). The web application’s back-end will need to hold the logic for deciding which assets to send. Moreover, modern web applications often cater to the specific user rather than sending the same files to every visitor of a webpage. This is known as dynamic content.

When we eat at a restaurant, we’ll order to our tastes, make substitutions, etc; the result is a dining experience unique to us. Aside from that, there’s a lot happening behind the scenes to make a restaurant work: ingredients are ordered from suppliers, new menus are designed, and employees’ schedules are created. Similarly, to make a web application that runs smoothly, the back-end is doing a lot more than simply sending assets to browsers.

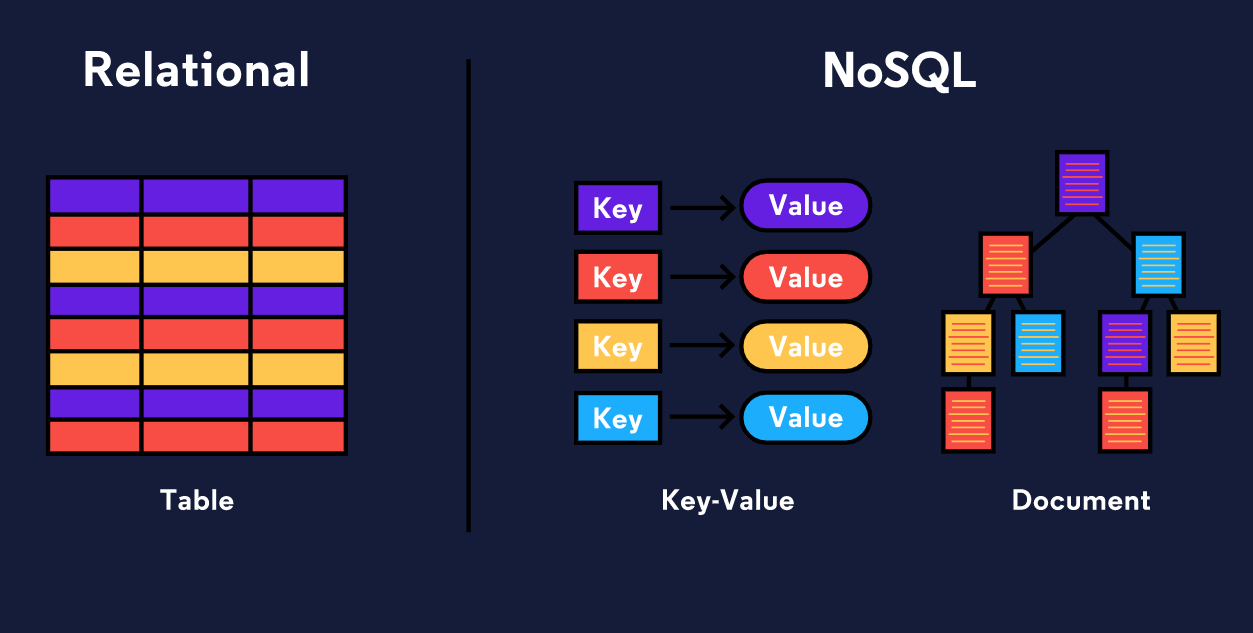
The collection of programming logic required to deliver dynamic content to a client, manage security, process payments, and myriad other tasks is sometimes known as the “application” or application server. The application server can be responsible for anything from sending an email confirmation after a purchase to running the complicated algorithms a search engine uses to give us meaningful results.

# Storing Data

You’ve probably heard that data is a big deal. By some measures, 90% of the world’s data has been generated in just the past two years! From a stored credit card number on an e-commerce site to the timestamp when you hit pause on Netflix, modern web applications collect a lot of data. For that data to be useful, it has to be organized and stored somewhere.

The back-ends of modern web applications include some sort of database, often more than one. Databases are collections of information. There are many different databases, but we can divide them into two types: [relational databases](https://www.codecademy.com/articles/what-is-rdbms-sql) and [non-relational databases (also known as NoSQL databases)](https://en.wikipedia.org/wiki/NoSQL). Whereas relational databases store information in tables with columns and rows, non-relational databases might use other systems such as key-value pairs or a document storage model. SQL, **S**tructured **Q**uery **L**anguage, is a programming language for accessing and changing data stored in relational databases. Popular relational databases include [MySQL](https://www.mysql.com/) and [PostgreSQL](https://www.postgresql.org/) while popular NoSQL databases include [MongoDB](https://www.mongodb.com/) and [Redis](https://redis.io/).

In addition to the database itself, the back-end needs a way to programmatically access, change, and analyze the data stored there.



# What is an API?

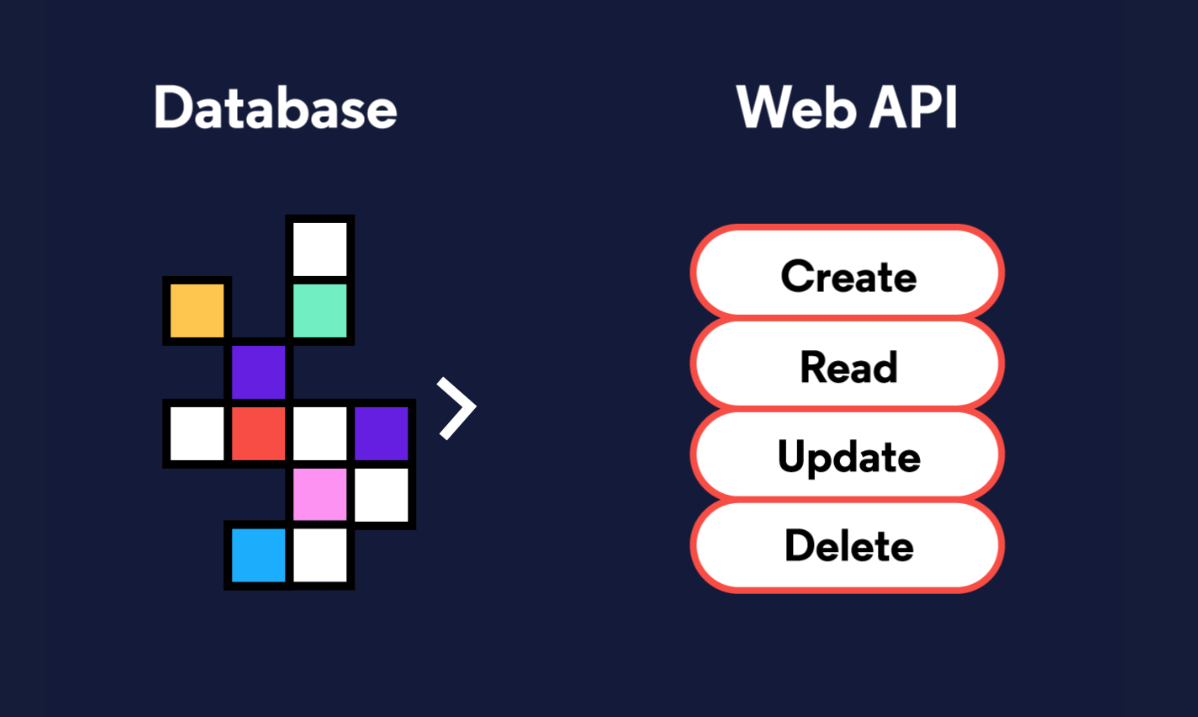
When a user navigates to a specific item for sale on an e-commerce site, the price listed for that item is stored in a database, and when they purchase it, the database will need to be updated with the correct inventory for that item type. In fact, much of what the back-end entails is reading, updating, or deleting information stored in a database.

In order to have consistent ways of interacting with data, a back-end will often include a web API. API stands for **A**pplication **P**rogram **I**nterface and can mean a lot of different things, but a web API is a collection of predefined ways of, or rules for, interacting with a web application’s data, often through an HTTP request-response cycle. Unlike the HTTP requests a client makes when a user navigates to a website’s URL, this type of request indicates how it would like to interact with a web application’s data (create new data, read existing data, update existing data, or delete existing data), and it receives some data back as a response.

Let’s walk through the example of making an online purchase again, but this time, we’ll imagine how the application’s web API might be used. When a user presses the button to submit an order, that will trigger a request to send them to a different view of the website, an order confirmation page, but an additional request will be triggered from the front-end, unseen by the user, to the web API so that the database can be updated with the information from the order.

Some web APIs are open to the public. [Instagram](https://www.instagram.com/developer/), for example, has an API that other developers can use to access some of the data Instagram stores. Others are only used by the web application internally— Codecademy, for example, has a web API that employees use to accomplish internal tasks.

## Image



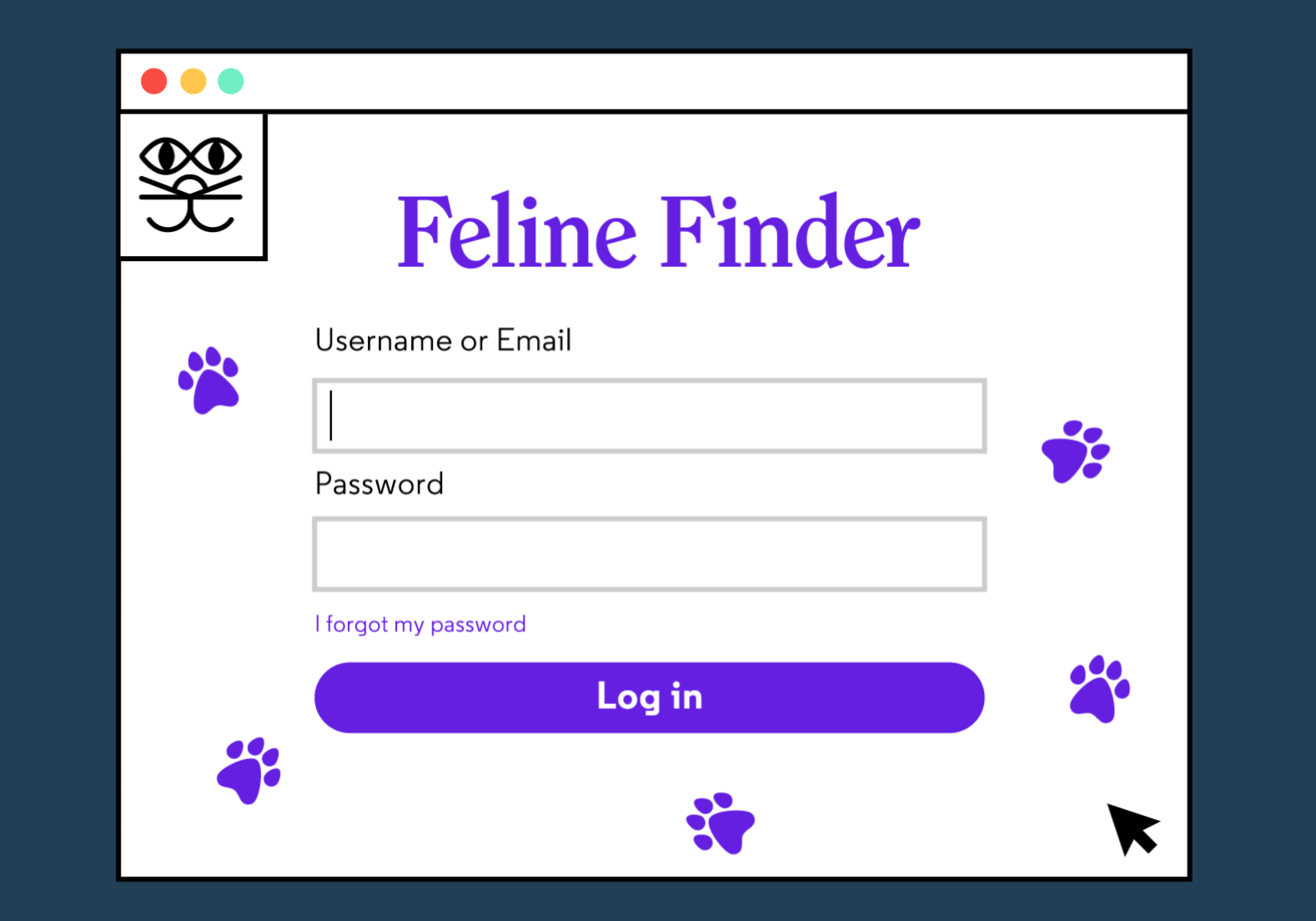
# Authorization and Authentication

Two other concepts we’ll want our server-side logic to handle are authentication and authorization.

Authentication is the process of validating the identity of a user. One technique for authentication is to use logins with usernames and passwords. These credentials need to be securely stored in the back-end on a database and checked upon each visit. Web applications can also use external resources for authentication. You’ve likely logged into a website or application using your Facebook, Google, or Github credentials; that’s also an authentication process.

Authorization controls which users have access to which resources and actions. Certain application views, like the page to edit a social media personal profile, are only accessible to that user. Other activities, like deleting a post, are often similarly restricted.

When building a robust web application back-end, we need to incorporate both authentication (Who is this user? Are they who they claim to be?) and authorization (Who is allowed to do and see what?) into our server-side logic to make sure we’re creating secure, personalized, and dynamic content.



# Different Back-end Stacks

Unlike the front-end, which must be built using HTML, CSS, and JavaScript, there’s a lot of flexibility in which technologies can be used in order to create the back-end of a web application. Developers can construct back-ends in many different languages like PHP, Java, JavaScript, Python, and more.

You don’t need to reinvent the wheel to create a robust back-end. Instead, most developers make use of frameworks which are collections of tools that shape the organization of your back-end and provide efficient ways of accomplishing otherwise difficult tasks.

There are numerous [back-end frameworks](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks#A_few_good_web_frameworks) from which developers can choose. Here are a few examples:

| **Framework** | **Language** |
| --- | --- |
| [Laravel](https://laravel.com/) | [PHP](http://www.php.net/) |
| [Express.js](https://expressjs.com/) | [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript) (runs in the [Node environment](https://nodejs.org/en/)) |
| [Ruby on Rails](https://rubyonrails.org/) | [Ruby](https://www.ruby-lang.org/en/) |
| [Spring](https://spring.io/) | [Java](https://www.oracle.com/java/) |
| [JSF](https://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html) | [Java](https://www.oracle.com/java/) |
| [Flask](http://flask.pocoo.org/) | [Python](https://www.python.org/) |
| [Django](https://www.djangoproject.com/) | [Python](https://www.python.org/) |
| [ASP.NET](https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet) | [C#](https://dotnet.microsoft.com/learn/csharp) |

The collection of technologies used to create the front-end and back-end of a web application is referred to as a stack. This is where the term full-stack developer comes from; rather than working in either the front-end or the back-end exclusively, a full-stack developer works in both.

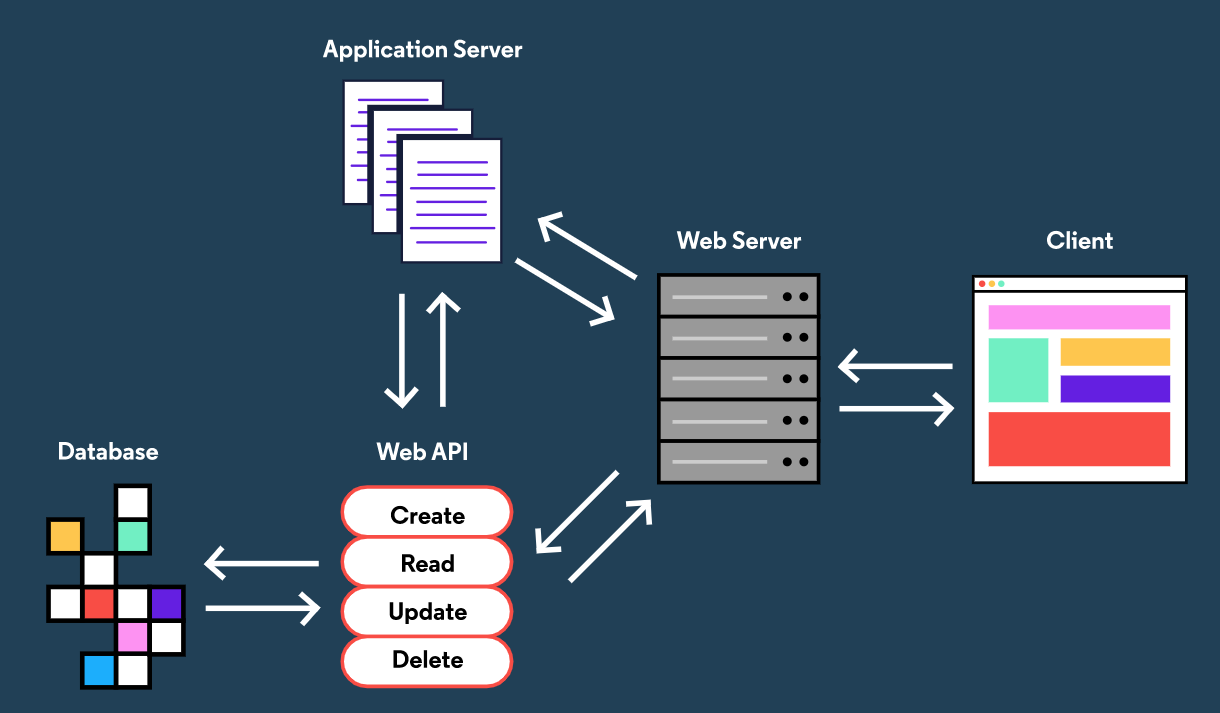
For example, [the MEAN stack](https://en.wikipedia.org/wiki/MEAN_(software_bundle)) is a technology stack for building web applications that uses **M**ongoDB, **E**xpress.js, **A**ngularJS, and **N**ode.js: MongoDB is used as the database, Node.js with Express.js for the rest of the back-end, and Angular is used as a front-end framework. While the [LAMP Stack](https://en.wikipedia.org/wiki/LAMP_(software_bundle)), sometimes considered the archetypal stack, uses **L**inux, **A**pache, **M**ySQL, and **P**HP.

# Review

In order to deliver the front-end of a website or web application to a user, a lot needs to happen behind the scenes on the back-end! Understanding what makes up the back-end can be overwhelming because the back-end has a lot of different parts, and different websites or web applications can have dramatically different back-ends. We covered a lot in this lesson, so let’s review what we learned:

* The front-end of a website or application consists of the HTML, CSS, JavaScript, and static assets sent to a client, like a web browser.
* A web server is a process running on a computer somewhere that listens for incoming requests for information over the internet and sends back responses.
* Storing, accessing, and manipulating data is a large part of a web application’s back-end
* Data is stored in databases which can be relational databases or NoSQL databases.
* The server-side of a web application, sometimes called the application server, handles important tasks such as authorization and authentication.
* The back-end of web application often has a web API which is a way of interacting with an application’s data through HTTP requests and responses.
* Together the technologies used to build the front-end and back-end of a web application are known as the stack, and many different languages and frameworks can be used to build a robust back-end.

Now that you have a sense for server-side web development and what the back-end is, you’re ready to dive in and learn about the different parts in more depth!



# Introduction

For a long time, the browser was the only place JavaScript code could be executed. Web developers had to use a different programming language on the front-end than the back-end. It also meant that, even as JavaScript evolved into a more robust and powerful language, it remained a front-end only language.

Though multiple attempts to create off-browser JavaScript environments have been attempted, [Node.js](https://nodejs.org/en/), invented by Ryan Dahl in 2009, found unprecedented popularity and is currently being used by numerous top-tier companies including Netflix, Uber, Paypal, and eBay. Node.js is a JavaScript runtime, or an environment that allows us to execute JavaScript code outside of the browser. A “runtime” converts code written in a high-level, human-readable, programming language and compiles it down to code the computer can execute.

Though Node was created with the goal of building web servers and web applications in JavaScript, it can also be used for creating command-line applications or desktop applications. In this lesson, we’ll explore some features of Node so you start to feel comfortable with running JavaScript in the Node environment and gain some familiarity with features unique to Node. For more advanced development, Node can be combined with any number of robust frameworks like the [Express.js framework](https://expressjs.com/) for creating effective web application back-ends.

There’s more to learn about Node than we could ever fit in one lesson. We’ll try to point to great resources like [MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript) and the [Node.js documentation](https://nodejs.org/api/). Take your time exploring and use the documentation.

**Instructions**

**1.**

In this lesson, we’ll be providing you a terminal with Node.js already installed. If you’d like to download Node on your local machine and follow along, check out [this article](https://www.codecademy.com/content-items/c4fe3060dbc61fc82d810c4ea06c29a8).

Let’s see what version of Node we have installed. Type node -v in the terminal and then press and hit enter (or return).

Node -v