Understanding Back End in a Web Application & JSON

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 create and design 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.

The Web Server

We talked about how the front-end consists of the information sent to a client so that a user can see and interact with a website, but where does the information come from? The answer is a web server.

The word "server" can mean a lot of things in computing, but we're going to focus on web servers specifically. A web server is a process running on a computer that listens for incoming request for information over the Internet and sends back responses.

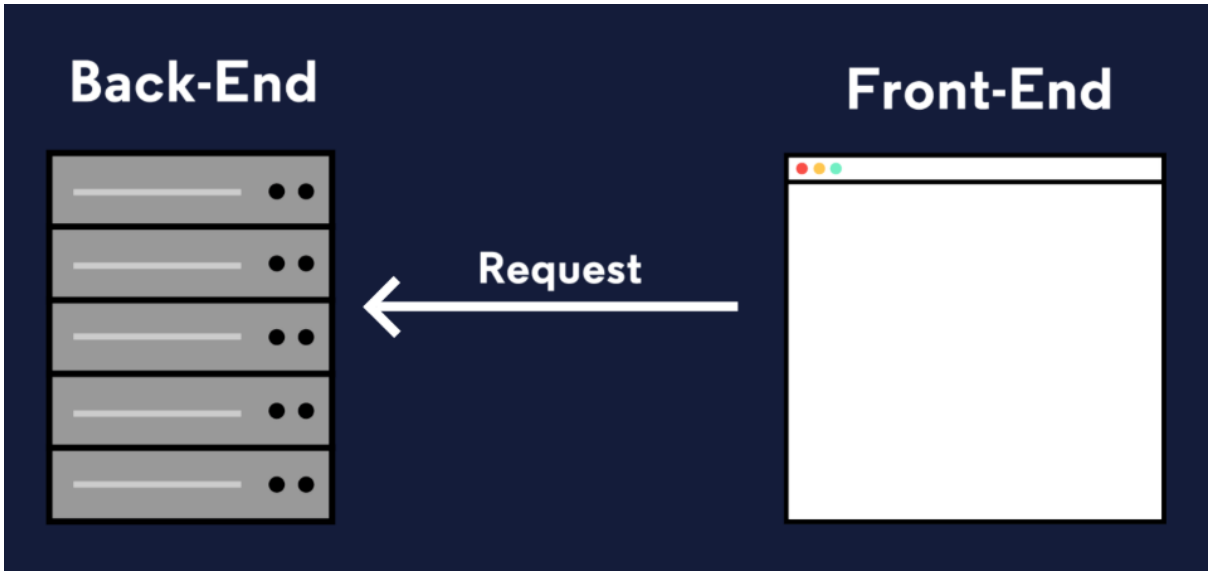
Each time a user visits a website in their browser, the browser sends a request to that site's web server. Every website has at least one web server. A large web platform may run many web servers across multiple locations around the world, but we can also run a simple web server on our own computer.

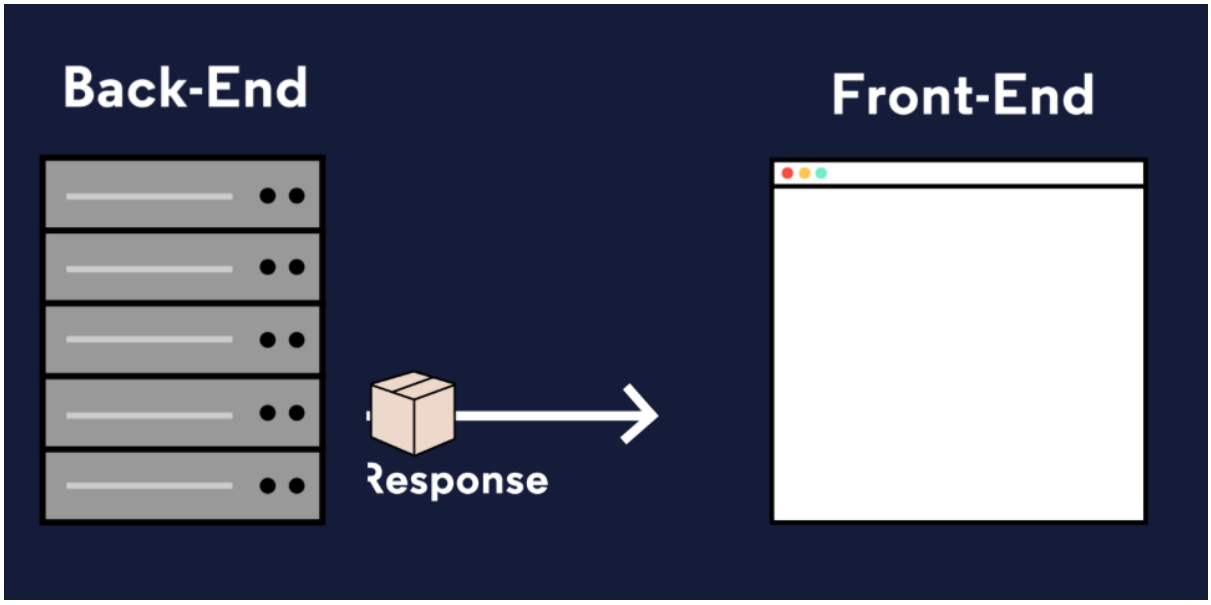
The specific format of a request (and the resulting response) is called the protocol. You might be familiar with the protocol used to access websites: HTTP. When a visitor navigates to a website on their browser,

similarly to how one places an order for takeout, they make an HTTP request for the resources that make up that site.
For simpler websites, the server mainly returns prebuilt files such as HTML, CSS, JavaScript, and images. This is often called a static website. That does not mean the site cannot be interactive. It means the server is mainly delivering files that were prepared ahead of time rather than generating custom content for each request. A static website might work well for a simple personal site with a short bio and photo gallery.

Modern web applications are often more dynamic. A user on a platform like X, YouTube, or Instagram expects personalized and constantly changing content. Supporting that kind of experience requires a more capable back-end that can process requests, work with data, and return different responses depending on the situation.

A static website is like ordering takeout, but a modern web application is more like dining in at a restaurant. A customer might order drinks, request substitutions, ask questions, or place multiple orders. Supporting that level of interaction requires a more complex back-end.





So What is the Back-end?
When a user visits an online store, the price of a product might change depending on their location, currency, or available promotions. For example, a website might show different prices for users in different countries. This type of behavior requires logic running on the back-end, which decides what information should be sent to the browser. Instead of sending the same files to every visitor, modern web applications often customize what they return depending on the request or the user. This is known as **dynamic content**.

When we eat at a restaurant, we order according to our preferences, make substitutions, or ask questions. The result is a dining experience unique to us. Behind the scenes, however, many things must happen to make the restaurant operate smoothly: ingredients are ordered, menus are planned, and employees' schedules are managed. Similarly, a web application's back-end performs many behind-the-scenes tasks beyond simply sending files to the browser.

The back-end contains business logic, which is the set of rules that determine how an application behaves. This logic processes requests retrieves or updates data, applies application rules, and returns appropriate responses to clients.

The collection of logic and services that handle these responsibilities is often called the **application server** or **application layer**. It can perform tasks such as sending confirmation emails after a purchase, validating user actions, processing payments, or running the algorithms that power search engines.

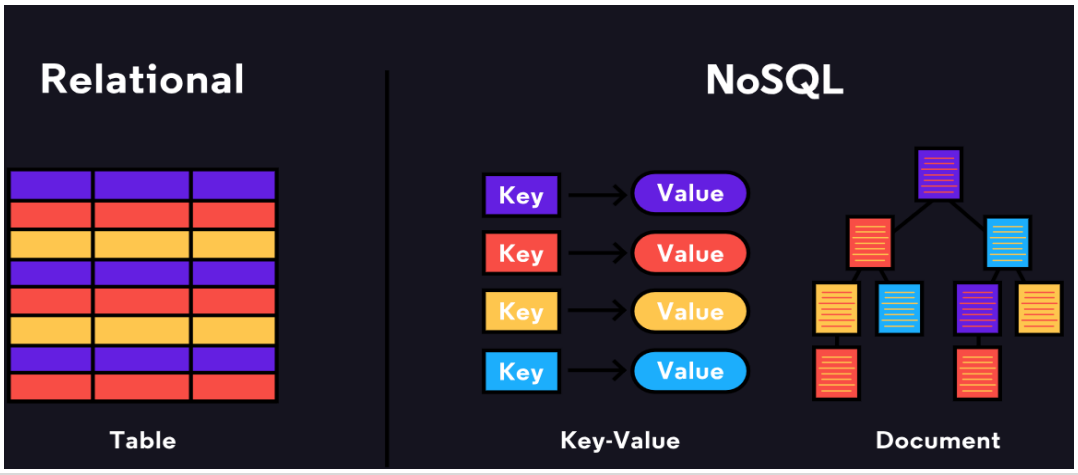
Storing Data
You've probably heard that data is a big deal. By some measures, 90% of the world's data has been generated in just the past two years. From a stored credit card number on an e-commerce site to the timestamp when you pause a video on a streaming platform, modern web applications collect and use large amounts of data. For that data to be useful, it must be organized and stored somewhere.

The back-ends of modern web applications include one or more databases. Databases are systems used to store and organize information so that it can be retrieved and updated efficiently. There are many types of databases, but they are commonly grouped into two broad categories: relational databases and non-relational databases (also known as NoSQL databases).

Relational databases organize data into tables made up of rows and columns. Each table typically represents a type of data, such as users, products, or orders. SQL (Structured Query Language) is commonly used to access and modify data stored in relational databases. Popular relational databases include MySQL and PostgreSQL.

Non-relational databases store data using other structures, such as key-value pairs, documents, or tree-like structures. These databases are often used when applications need flexible data formats or very large-scale data storage.
Popular NoSQL databases include MongoDB and Redis.

In addition to storing the data, the back-end must also include code that can read, create, update, and delete that data when requests are made by users or applications.



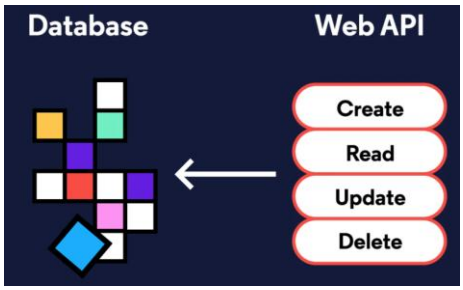
What is an API?
When a user navigates to a specific item for sale on an e-commerce site, the price listed for that item is stored in a database. When they purchase it, the database must be updated to reflect the correct inventory. Much of what the back-end does involves reading, creating, updating, or deleting data stored in a database.

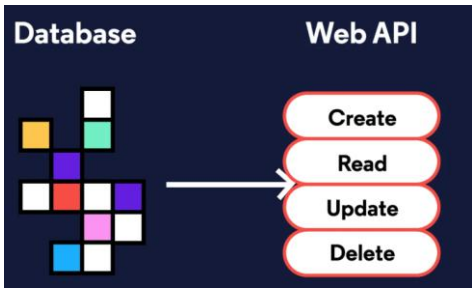
To provide consistent ways for applications to interact with this data, a back-end often exposes a **web API**. API stands for **Application Programming Interface**. In web development, a web API is a set of rules that allows applications to communicate with a server and interact with its data, usually through an HTTP request–response cycle.

Instead of simply requesting a webpage, these requests tell the server what action should be performed on the data. For example, an API request might create new data, retrieve existing data, update data, or delete data. The server processes the request, interacts with the database if needed, and then returns a response.

Let's revisit the example of making an online purchase. When a user submits an order, the browser loads an order confirmation page. At the same time, the front-end sends a request to the web API to record the order details and update the product inventory in the database.

Some web APIs are **public**, meaning other developers can use them to access certain features or data. For example, Instagram provides APIs that developers can use to interact with parts of the Instagram platform. Other APIs are **internal**, meaning they are only used by the application itself to allow different parts of the system to communicate.





Above showing a web API sending a request to a database and receiving data in response, then move on to the next exercise when ready.

Authorization and Authentication
Two other important responsibilities handled by server-side logic are **authentication** and **authorization**. These concepts help ensure that applications remain secure and that users can only access the information and actions they are allowed to use.

Authentication is the process of verifying the identity of a user. In other words, the application confirms that the user is who they claim to be. A common method of authentication is logging in with a username or email and a password. These credentials are securely stored in a database and checked when the user attempts to sign in. Web applications can also use external authentication providers. For example, you may have logged into an app using your Google, GitHub, or Facebook account. In this case, another service verifies your identity.

Once a user is authenticated, the application must determine what that user is allowed to do. This is called **authorization**. Authorization controls which resources and actions a user can access. For example, a user can edit their own profile or delete their own post, but they typically cannot edit or delete another user's content. Some actions may also be limited to administrators or specific roles.

Together, authentication and authorization help protect sensitive data and ensure that users only perform actions they are permitted to perform. When building a secure back-end, both processes are essential for managing user access and maintaining application security.

Different Back-end Stacks
Unlike the front-end, which is typically built using HTML, CSS, and JavaScript, there is much more flexibility in the technologies used to build the back-end of a web application. Developers can create back-end systems using many programming languages, such as PHP, Java, JavaScript, Python, and others.

Rather than building everything from scratch, developers usually rely on **frameworks**. A framework is a collection of tools and conventions that help structure an application and simplify common tasks such as routing requests, handling data, and organizing code.

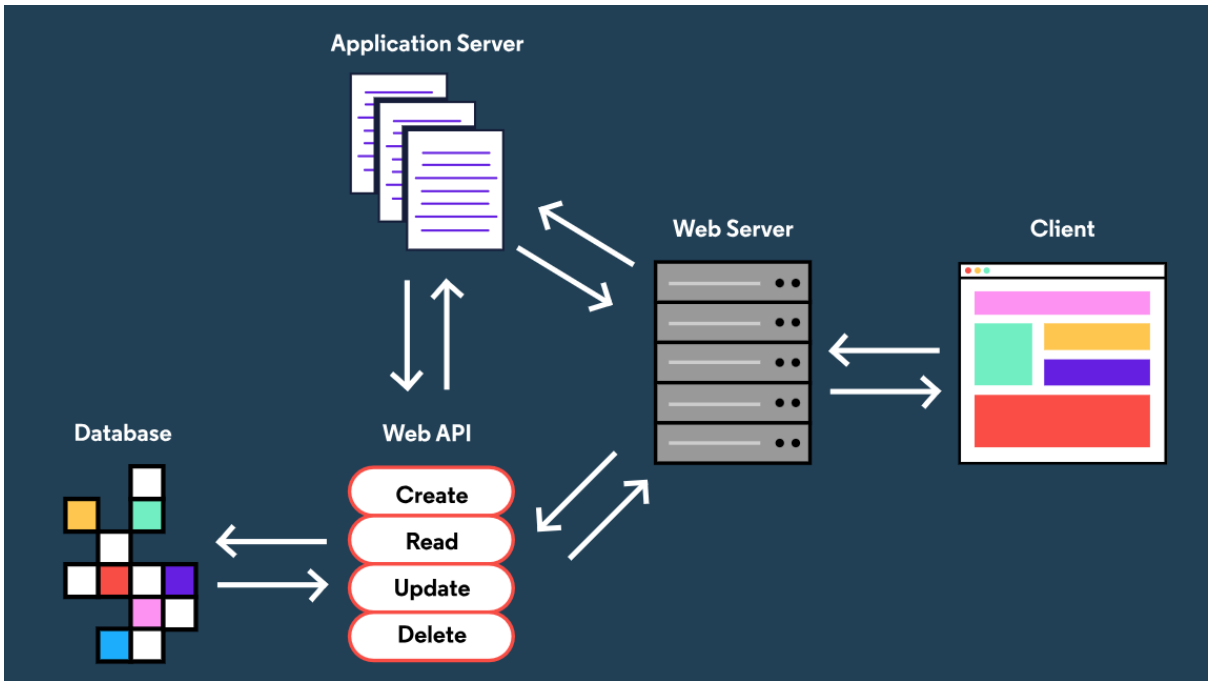
There are numerous back-end frameworks from which developers can choose. Here are a few examples:

Framework	Language
Laravel	PHP
Express.js	JavaScript (runs in the Node environment)
Ruby on Rails	Ruby
Spring	Java
JSF	Java
Flask	Python
Django	Python
ASP.NET	C#

The collection of technologies used to build the front-end and back-end of an application is known as a **technology stack**, or simply a **stack**. This is where the term full-stack developer comes from. A full-stack developer works on both the front-end and back-end parts of an application.

For example, the MEAN stack is a technology stack that includes MongoDB, Express.js, Angular, and Node.js. MongoDB is used to store data, Node.js runs the server-side environment, Express.js helps build the web server and APIs, and Angular is used to create the front-end interface.

Another common example is the LAMP stack, which consists of Linux, Apache, MySQL, and PHP. Linux is the operating system, Apache is the web server that handles requests, MySQL stores the application's data, and PHP is used to write the back-end application logic.

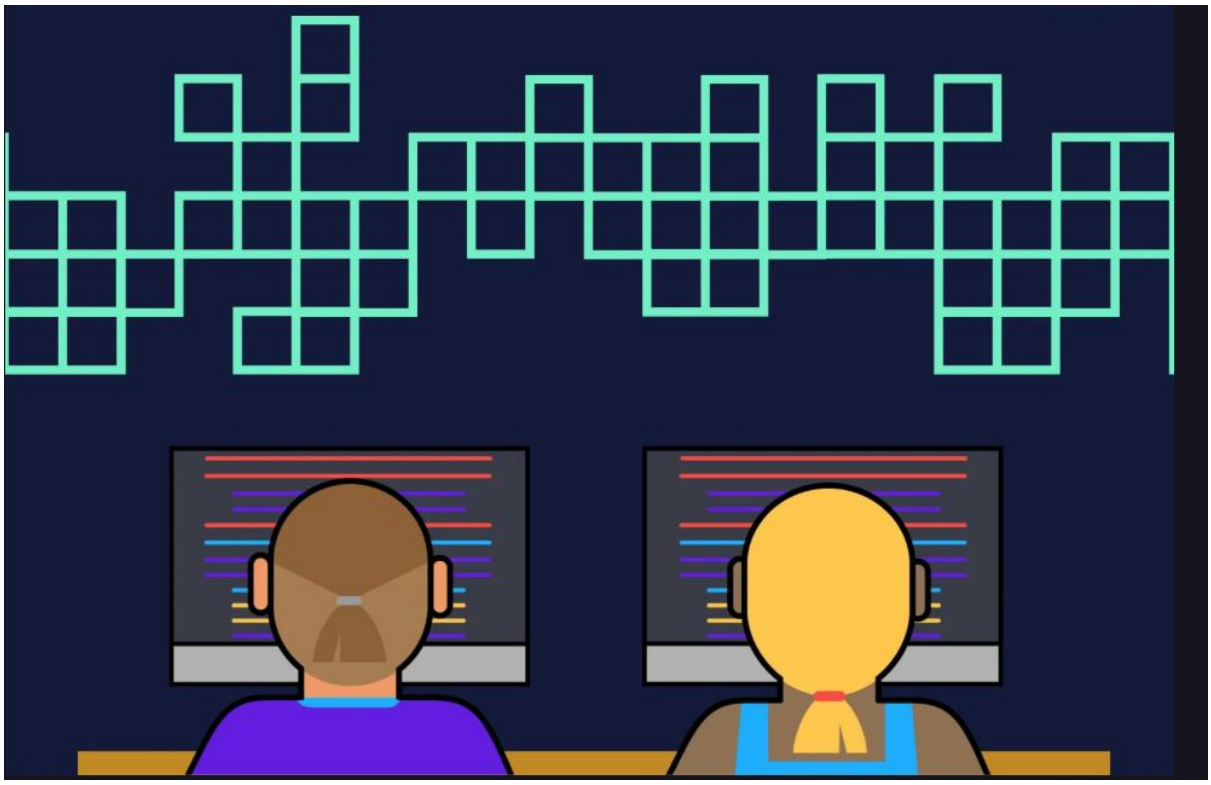


In order to deliver the front-end of a website or web application to a user, a lot needs to happen behind the scenes on the back-end! Understanding what makes up the back-end can be overwhelming because the back-end has a lot of different parts, and different websites or web applications can have dramatically 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 a 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!



JSON - JavaScript Object Notation

In a world inundated with data, it is becoming more important to know how to work with a variety of data. As programmers, we need to be able to transfer our populated data structures from any language we choose to a format that is recognizable and readable by other languages and platforms. Fortunately for us, there exists such a data-exchange format.

What is JSON?
JSON, or JavaScript Object Notation, is a popular, language-independent, standard format for storing and exchanging data. Adopted by ECMA International, an industry association founded in 1961 to standardize information and communication systems, JSON has become the de facto standard that facilitates storing and sending data between all programming languages.

Common Uses of JSON
JSON is heavily used to facilitate data transfer in web applications between a client, such as a web browser, and a server. A typical example where such data transfer occurs is when you fill out a web form. The form data is converted from HTML to JavaScript objects to JSON objects and sent to a remote web server for processing. These transactions could be as simple as entering a search engine query to a multi-page job application.

When companies make their data public for other applications, like Spotify sharing its music library or Google sharing its map data, the information is formatted in JSON. This way, any application, regardless of language, can collect and parse the data.

Some of the popular web APIs that utilize JSON in data exchanges are:

- Google Maps
- Google Auth 2.0 Authentication
- Facebook Social Graph API
- Spotify Music Web API
- LinkedIn Profile API

JSON Syntax
Since JSON is derived from the JavaScript programming language, its appearance is similar to that of JavaScript objects.

A sample JSON object is represented as follows:

```
{
  "student": {
    "name": "Rumaisa Mahoney",
    "age": 30,
    "fullTime": true,
    "languages": [ "JavaScript", "HTML", "CSS" ],
    "GPA": 3.9,
    "favoriteSubject": null
  }
}
```

Note the following syntax rules for JSON:

- The curly braces, {}, hold objects.
- The square brackets, [], hold arrays.
- Data is stored in key:value pairs separated by a colon, :.
- Every key:value pair is separated from another pair by a comma, ,. Similarly, every item in an array is delimited by a comma. Trailing commas are forbidden.
- Property names in JSON must be wrapped in double quotes, "", even though JavaScript names are not held by this stringency.
- JSON does not support comments.

The following object would be valid JavaScript, but is not valid JSON:

```
{
  "student": {
    name: "Rumaisa Mahoney",
    "age": 30,
    "fullTime": true,
    "languages": [ "JavaScript", "HTML", "CSS", ],
    "GPA": 3.9,
    'favoriteSubject': null
  }
}
```

1. The name property is not wrapping double quotes.
2. The "languages" array has a trailing comma after the final item, "CSS".
3. 'favoriteSubject' uses single quotes instead of double quotes.

JSON Data Types

A JSON data type must be one of the following:

- string (must be wrapped in double quotes, like property names)
- number (integer or floating point)
- object (key:value pair)
- array (comma-separated values)
- boolean (true or false, without quotes)
- null

Task: Try to find all the data types in this JSON example:

```
{
  "student": {
    "name": "Rumaisa Mahoney",
    "age": 30,
    "fullTime": true,
    "languages": [ "JavaScript", "HTML", "CSS" ],
    "GPA": 3.9,
    "favoriteSubject": null
  }
}
```

Notably, JSON doesn't cover every data type. Types that are not represented in JSON, such as dates, can be stored as a string and converted to a language-specific data structure. Here's an acceptable internationally-recognized date format in ISO 8601:

"2014-01-01T23:28:56.782Z"

This format contains parts which resemble a date and time. However, as a string, it is hard for a programming language to use as is. Conveniently, every programming language has built-in JSON facilities to convert this string into a more readable and usable format, such as:
Wed Jan 01 2014 13:28:56 GMT-1000 (Hawaiian Standard Time)

This pretty much covers the basic description of JSON, its popularity, and its syntax. Congratulations on reaching this milestone!