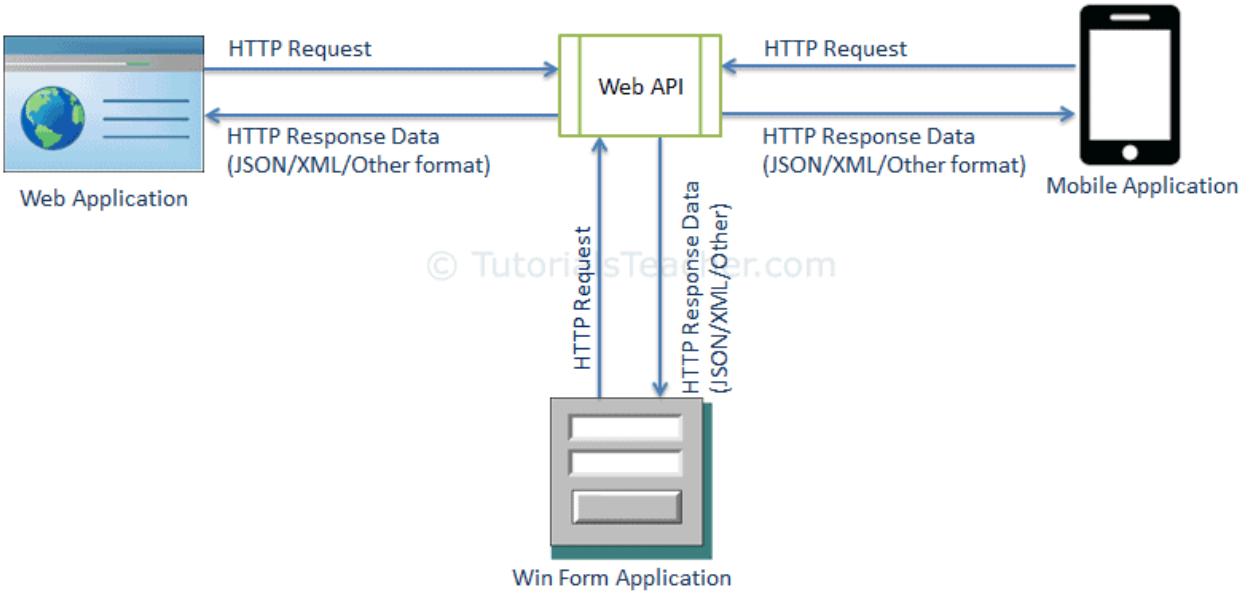
web API



Creating a web API

Appian makes it easy to create your web APIs by providing you with a series of templates to help you get started.

Selecting one of the templates will populate your web API with an example expression and test inputs, and convert it to use the proper HTTP method for that expression. If you already have your web API expression in mind, you can also create a web API from scratch.

Create a web API using a template

To create a web API using a template:

1. In your application, go to the **Build** view.

2. Click **NEW** > **Web API**.
3. In the **Create Web API** dialog, choose **Select a template** and select from one of the available templates.
4. Configure the following properties. Depending on your selected templat

What can APIs do?

There are a huge number of APIs available in modern browsers that allow you to do a wide variety of things in your code. You can see this by taking a look at the MDN APIs index page.

Common browser APIs

In particular, the most common categories of browser APIs you'll use (and which we'll cover in this module in greater detail) are:

- APIs for manipulating documents loaded into the browser. The most obvious example is the DOM (Document Object Model) API, which allows you to manipulate HTML and CSS — creating, removing and changing HTML, dynamically applying new styles to your page, etc. Every time you see a popup window appear on a page or some new content displayed, for example, that's the DOM in action. Find out more about these types of API in Manipulating documents.
- APIs that fetch data from the server to update small sections of a webpage on their own are very commonly used. This seemingly small detail has had a huge impact on the performance and behavior of sites — if you just need to update a stock listing or list of available new stories, doing it instantly without having to reload the whole entire page from

the server can make the site or app feel much more responsive and "snappy". The main API used for this is the Fetch API, although older code might still use the XMLHttpRequest API. You may also come across the term Ajax, which describes this technique. Find out more about such APIs in Fetching data from the server.

- APIs for drawing and manipulating graphics are widely supported in browsers — the most popular ones are Canvas and WebGL, which allow you to programmatically update the pixel data contained in an HTML <canvas> element to create 2D and 3D scenes. For example, you might draw shapes such as rectangles or circles, import an image onto the canvas, and apply a filter to it such as sepia or grayscale using the Canvas API, or create a complex 3D scene with lighting and textures using WebGL. Such APIs are often combined with APIs for creating animation loops (such as window.requestAnimationFrame()) and others to make constantly updating scenes like cartoons and games.
- Audio and Video APIs like HTMLMediaElement, the Web Audio API, and WebRTC allow you to do really interesting things with multimedia such as creating custom UI controls for playing audio and video, displaying text tracks like captions and subtitles along with your videos, grabbing video from your web camera to be manipulated via a canvas (see above) or displayed on someone else's computer in a web conference, or adding effects to audio tracks (such as gain, distortion, panning, etc.).
- Device APIs enable you to interact with device hardware: for example, accessing the device GPS to find the user's position using the Geolocation API.

- Client-side storage APIs enable you to store data on the client-side, so you can create an app that will save its state between page loads, and perhaps even work when the device is offline. There are several options available, e.g. simple name/value storage with the Web Storage API, and more complex database storage with the IndexedDB API.

Common third-party APIs

Third-party APIs come in a large variety; some of the more popular ones that you are likely to make use of sooner or later are:

- Map APIs, like Mapquest and the Google Maps API, which allow you to do all sorts of things with maps on your web pages.
- The Facebook suite of APIs, which enables you to use various parts of the Facebook ecosystem to benefit your app, such as by providing app login using Facebook login, accepting in-app payments, rolling out targeted ad campaigns, etc.
- The Telegram APIs, which allows you to embed content from Telegram channels on your website, in addition to providing support for bots.
- The YouTube API, which allows you to embed YouTube videos on your site, search YouTube, build playlists, and more.
- The Pinterest API, which provides tools to manage Pinterest boards and pins to include them in your website.

- The Twilio API, which provides a framework for building voice and video call functionality into your app, sending SMS/MMS from your apps, and more.
- The Disqus API, which provides a commenting platform that can be integrated into your site.
- The Mastodon API, which enables you to manipulate features of the Mastodon social network programmatically.
- The IFTTT API, which enables integrating multiple APIs through one platform.

How do APIs work?

Different JavaScript APIs work in slightly different ways, but generally, they have common features and similar themes to how they work.

They are based on objects

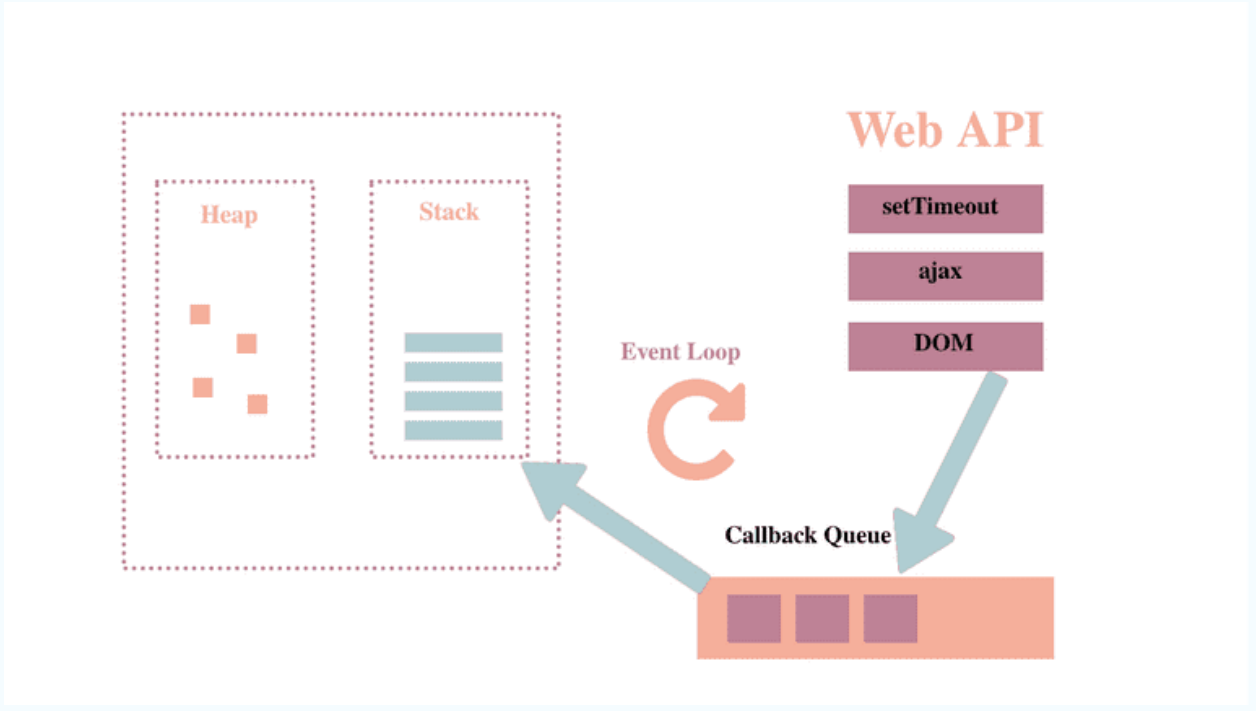
Your code interacts with APIs using one or more JavaScript objects, which serve as containers for the data the API uses (contained in object properties), and the functionality the API makes available (contained in object methods).

ASP.NET Web API Characteristics

1. ASP.NET Web API is an ideal platform for building RESTful services.
2. ASP.NET Web API is built on top of ASP.NET and supports ASP.NET request/response pipeline
3. ASP.NET Web API maps HTTP verbs to method names.

4. ASP.NET Web API supports different formats of response data. Built-in support for JSON, XML, BSON format.
5. ASP.NET Web API can be hosted in IIS, Self-hosted or other web server that supports .NET 4.0+.
6. ASP.NET Web API framework includes new HttpClient to communicate with Web API server. HttpClient can be used in ASP.MVC server side, Windows Form application, Console application or other apps.

Event Loop



What is the Event Loop?

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that a single JavaScript thread is used by default — by offloading operations to the system kernel whenever possible.

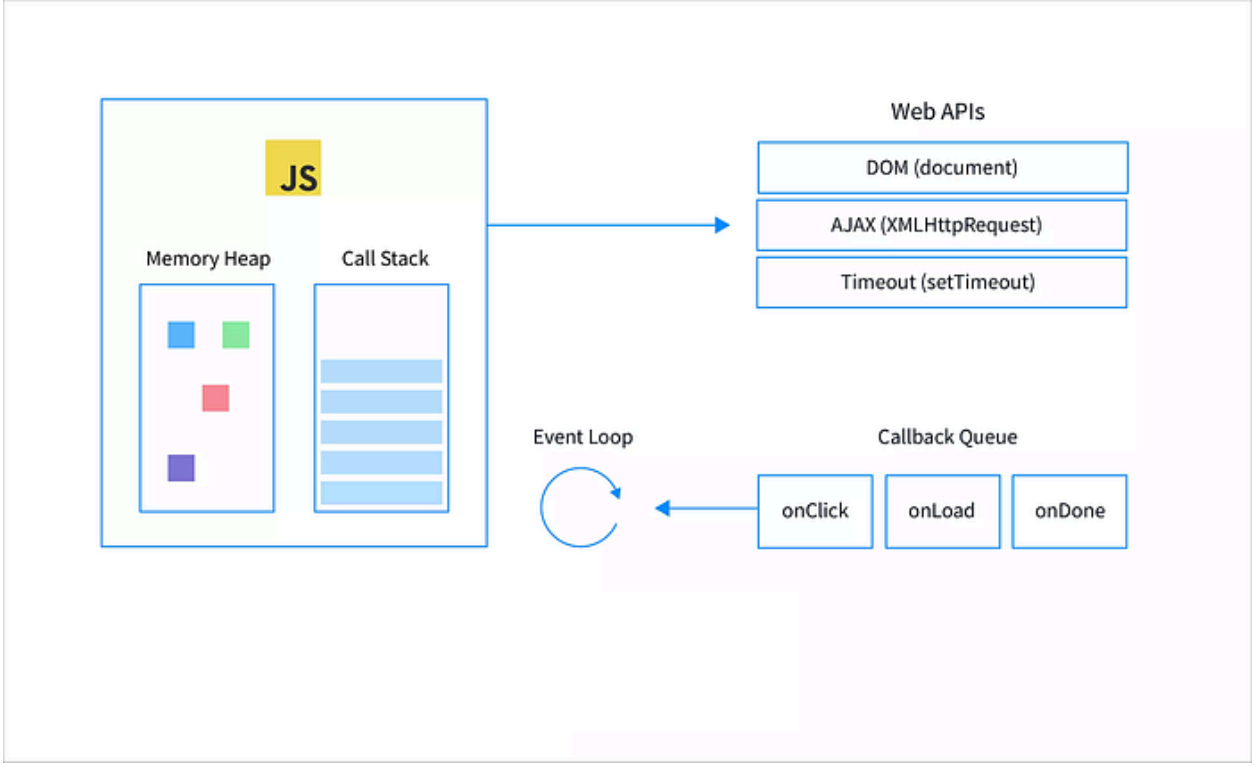
Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed. We'll explain this in further detail later in this topic.

Event Loop Explained

When Node.js starts, it initializes the event loop, processes the provided input script (or drops into the REPL, which is not

covered in this document) which may make async API calls, schedule timers, or call process.nextTick(), then begins processing the event loop.

The most important thing to understand in the diagram below is the interaction between the Call Stack, Callback Queue, and Web APIs. Before delving into the event loop, let me quickly mention the terms used here.



Event loop

Main Execution Context: When a JavaScript program starts running, it begins in the main execution context. Any synchronous code is executed line by line in this context.

Asynchronous Operations: Asynchronous operations, such as setTimeout, setInterval, or fetching data from a server, are encountered during execution. These operations are non-blocking, meaning they don't halt the execution of the main thread.

Web APIs: In JavaScript and web browsers, Web APIs refer to sets of functionalities provided by the browser environment to interact with web-related features and resources. These APIs include the Document Object Model (DOM) API for manipulating HTML and CSS, the XMLHttpRequest (XHR) or Fetch API for making HTTP requests, the Geolocation API for accessing user

location information, the localStorage and sessionStorage APIs for storing data locally in the browser, and many others.

Stack: A stack is a data structure that follows the Last-In-First-Out (LIFO) principle. This means that the last element added to the stack is the first one to be executed and removed. Think of it like a stack of plates where you can only add or remove plates from the top.

Call Stack: It stores function calls in the order they are invoked. When a function is called, it is added to the top of the stack, and when the function completes, it is removed from the stack. This allows JavaScript to keep track of function execution and manage the flow of code execution.

Queue: A queue is a data structure that follows the First-In-First-Out (FIFO) principle. This means that the first

element added to the queue is the first one to be executed and removed. Think of it like a line of people waiting for a service, where the person who arrived first is served first.

Event Queue: Asynchronous operations are scheduled to be executed later and are placed in the event queue once they are completed. Each operation in the queue contains a callback function to be executed when the operation is finished.