**Introduction**

In this lesson, you will be introduced to the concept of caching and why it is important for application performance.

**We'll cover the following**

* + [What is caching?](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#What-is-caching?)
  + [Caching dynamic data](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#Caching-dynamic-data)
  + [Caching static data](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#Caching-static-data)

Before getting on with the lesson, I want to ask you a question. When you visit a website and request certain information from the server, how long do you wait for the response?

5 seconds? 10? … 15 seconds? … 30? I know, I know, I am pushing it … 45 seconds? What? Still no response?

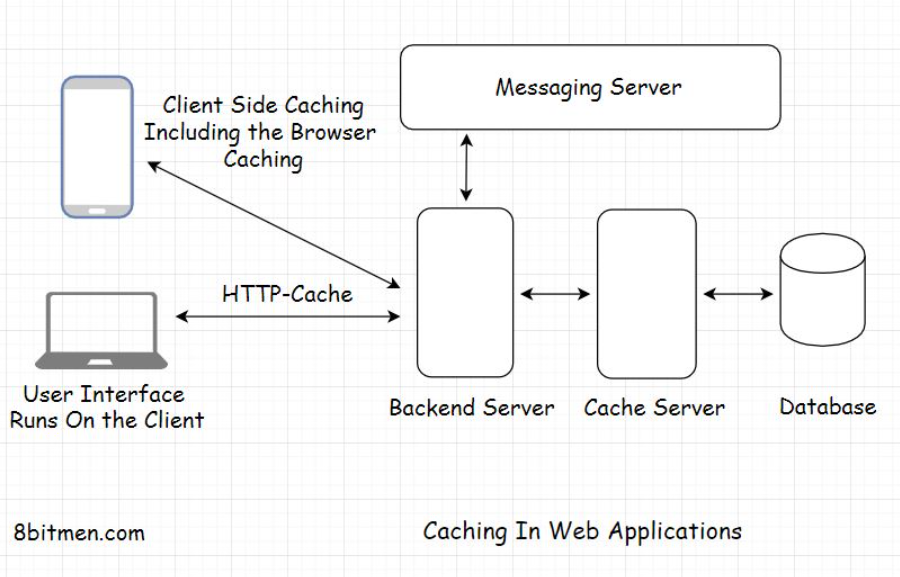
What option are you left with than to bounce off and visit another website for your information. We are impatient creatures, and we want our answers quick. Our application needs to have minimum latency and implementing caching enables us to stop users from bouncing off to other websites.

**What is caching?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#What-is-caching?)

Caching is key to the performance of any kind of application. It ensures low latency and high throughput. An application with caching will undoubtedly do better than an application without caching, simply because a cache intercepts all the requests darting towards the database and provides the response in no time.

Intercepting the database requests allows the database to free its resources to work with other requests, requesting uncached data, for open connections or write operations.

Implementing caching in a web application means copying frequently accessed data from the database, which is disk-based hardware, and storing it in RAM (Random access memory) for quick response.



RAM provides faster access than disk-based hardware, ensuring low latency and high throughput. Throughput means the number of network calls or request-response cycles between the client and the server within a stipulated time.

When an application server requests data from the database, it can be called the client and the database would be the server.

A cache can always handle more read requests than a database since it stores the data in a key-value pair and does not have to do much computation when returning the data in contrast to a database.

Frequently requested data queried from the database with the help of several table joins can be cached to avert the same joins query to be run every time the same data is requested. This increases throughput, improves performance and saves resources.

## Caching dynamic data[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#Caching-dynamic-data)

With caching, we can cache both the static and the dynamic data. Dynamic data changes more often and has an expiry time or a TTL (Time to live). After the TTL ends, the data is purged from the cache, and the newly updated data is stored in it. This process is known as cache invalidation.

Though the data TTL should be long enough to make effective use of caching, caching won’t help much if the data changes too often, for instance, the price of a stock, the score of a cricket or a baseball match.

## Caching static data[#](https://www.educative.io/module/lesson/web-application-architecture-101/g70LmlkOB6D#Caching-static-data)

Static data consists of images, font files, CSS, and similar files. It also includes data such as customer data, their name, age, address, social id, photos, etc. This is the kind of data that doesn’t change often and can easily be cached either on the client-side in their browser, CDN or on the server, depending on the sensitivity.

In the next lesson, we will cover how to decide if we really need a cache in our application.

**Do I Need A Cache?**

In this lesson, we will discuss how to tell if we need caching in our application.

**We'll cover the following**

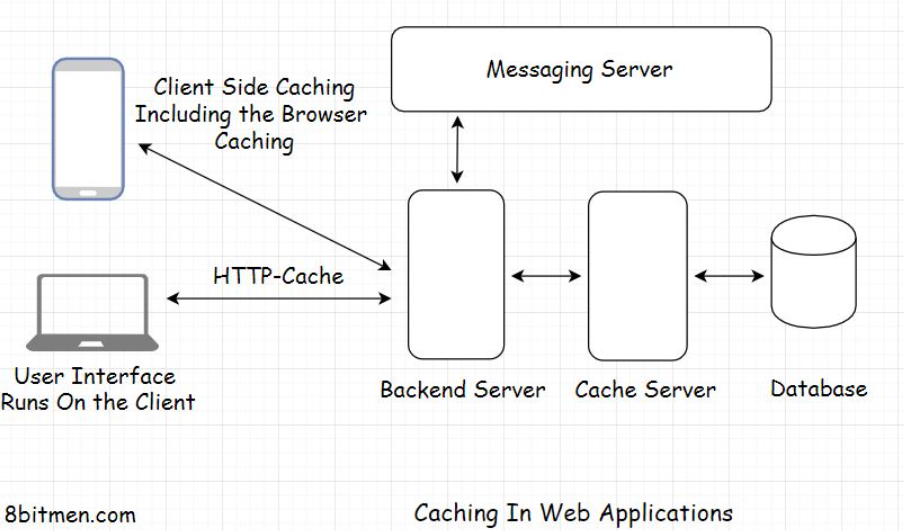
* + [Different components in the application architecture where the cache can be used](https://www.educative.io/module/lesson/web-application-architecture-101/qZ8X56w9rNG#Different-components-in-the-application-architecture-where-the-cache-can-be-used)

It’s always a good idea to use a cache as opposed to not using it, especially if we have static data in our application. You’ll come across very few use cases where caching doesn’t help. We also need to remember that caching should be implemented wisely. If not, it can cause data inconsistency issues.

It can be used at any layer of the application, with any component and there are no ground rules as to where it can and cannot be applied.

The most common usage of caching is database caching. Caching helps alleviate the stress on the database by intercepting the requests being routed to it.

## Different components in the application architecture where the cache can be used[#](https://www.educative.io/module/lesson/web-application-architecture-101/qZ8X56w9rNG#Different-components-in-the-application-architecture-where-the-cache-can-be-used)



Across the architecture of our application, we can use caching at multiple places, right from the browser to the database component. Caching is used in the client browser to cache static data. It is used with the database to intercept all the data requests. It is used in the REST API implementation, also in cross-module communication in a microservices architecture, etc.

Besides these components, I suggest you look for patterns. We can always cache the frequently accessed content on our website, be it from any component. There is no need to repeatedly poll any component for the same data when it can be cached.

Think of joins in relational databases. They are notorious for making the response slow. More joins mean more latency. A cache can avert the need for running joins every time by storing the data in demand. Imagine how much this mechanism would speed up our application.

Also, even if the database goes down for a while, the users won’t notice it since the cache would continue to serve the data requests.

Caching is also the core of the HTTP protocol. We can store user sessions in a cache. Key-value data stores are primarily used to implement caching in web applications.

# Reducing the Application Deployment Costs via Caching

In this lesson, we will discuss a real-world example of how effective caching can help us reduce the deployment cost of our application.

**We'll cover the following**

* + [Real-world use cases](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Real-world-use-cases)
    - [Stock market-based multiplayer game](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Stock-market-based-multiplayer-game)
    - [Polyhaven - a 3D asset library](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Polyhaven---a-3D-asset-library)

## Real-world use cases[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Real-world-use-cases)

### Stock market-based multiplayer game[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Stock-market-based-multiplayer-game)

In this lesson, I will share an insight from a stock market-based multiplayer game that I developed and deployed on the cloud.

The game had stocks of numerous companies listed on the stock market and the algorithm would trigger the stocks’ price movement based on certain parameters every second, if not before.

Initially, I persisted the updated stock price in the database as soon as the prices changed to create a stock price movement timeline at the end of the day. However, the number of database writes for the stocks price movement for the whole day was very high, having the potential to create a crater in my pocket.

Eventually, I decided not to persist the updated price every second in the database and rather use a cache (Memcache) to persist the updated stock prices. I then scheduled a batch operation that would run throughout the day every few hours to update the database with the stock prices.

In the cloud, writing to Memcache was cheaper than writing to the database by quite an extent. The cache served all the stock price read-write requests, and the database got the updated values when the batch operation ran.

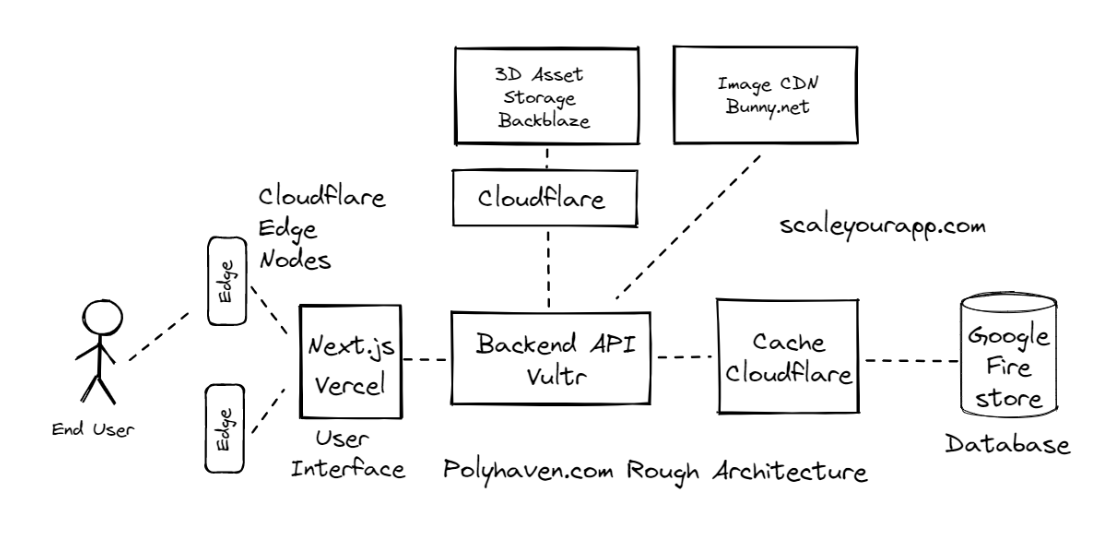
This tweak may not be ideal for a real-life Fintech app. However, it helped me save a truckload of money and allowed me to run the game for a more extended period of time.

So, this is one instance where you can leverage the caching mechanism to cut down costs. There are use cases where you might not want to persist every little information in the database and instead use the cache to store that not-so-mission-critical information.

### Polyhaven - a 3D asset library[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2WR49N0ron#Polyhaven---a-3D-asset-library)

This is a very rudimentary example of how an application with static data can leverage caching to significantly cut down its deployment costs.

I recommend reading [this blog article that I wrote](https://www.scaleyourapp.com/application-hosting-how-polyhaven-manages-5-million-page-views-and-80tb-traffic-a-month-for-400/) on how Polyhaven, a 3D asset library, managed 5 million page views and 80TB traffic a month for less than 400 USD by leveraging caching. If it weren’t for it, the storage cost for all that data on a cloud object-based storage would cost them approx. 4K USD a month.



In the next lesson, let’s look into some of the caching strategies we can leverage to enhance the app’s performance further.

**Best Usage of Caching by PolyHaven:**

===================================

https://scaleyourapp.com/application-hosting-how-polyhaven-manages-5-million-page-views-and-80tb-traffic-a-month-for-400/

- The caching layer of the app is powered by Cloudflare

\* https://www.cloudflare.com/

- The cache layer is further optimized by a Cloudflare product Argo, this improves the cache hit ratio to 93%.

\* https://blog.cloudflare.com/argo/

- All the 3D asset data is stored on Backblaze cloud storage. Empowered by a partnership of Cloudflare and Backblaze

\* https://www.backblaze.com/

- The images on the website that are thumbnails of the assets, renders, previews, etc., are stored using another CDN service Bunny.net. Besides providing hosting, the CDN service also dynamically resizes and compresses the images for the website costing 27 USD per month

\* https://bunny.net/

- The site is built using Next.js powered by Vercel servers

\* https://nextjs.org/

- The 3D assets are stored on the cloud object storage and the website database that is Google Firestore (a serverless document database) stores info on the 3D assets, download logs and so on.

- A separate API powered by Vultr compute connects the frontend to the database and gives them control over caching the database reads.

**Caching Strategies**

In this lesson, we will discuss some of the commonly used caching strategies.

**We'll cover the following**

* + [Cache aside](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Cache-aside)
  + [Read-through](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Read-through)
  + [Write-through](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Write-through)
  + [Write-back](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Write-back)

Here are some of the commonly used caching strategies in application development: *cache aside, read-through, write-through* and *write-back*. Every caching strategy has a specific use case.

Let’s find out what they are.

**Cache aside**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Cache-aside)

Cache aside is the most commonly used caching strategy. In this approach, the cache works along with the database intending to reduce the hits on it as much as possible.

The data is lazy-loaded in the cache. When the user sends a request for particular data, the system first looks for it in the cache. If present, it is simply returned. This is called a cache hit.

If not, this is called a cache miss. In this case, the application fetches the data from the database and returns it to the user, also updating the cache for future requests.

When it comes to data write, it is directly written to the database. Now, this could cause data inconsistency between the cache and the database. To avoid this, the data on the cache has a TTL (Time to live). After it expires, the data is invalidated from the cache.

This caching strategy works best with read-heavy workloads. The data that does not get updated too frequently, like customer data (name, account number, etc.) is cached using the cache aside strategy. We can assign a long TTL to this kind of data.

**Read-through**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Read-through)

This strategy is similar to the cache aside strategy. A subtle difference is that the cache in a read-through strategy always automatically stays consistent with the database.

The cache library, or the framework, takes the onus of maintaining consistency with the database. On the contrary, in the cache aside strategy, we have to write explicit logic to update the cache.

Application data in this strategy is also lazy-loaded in the cache only when the user requests it. Also, the data model of the cache has to be consistent with the database since it is updated automatically by the library.

**Write-through**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Write-through)

In this strategy, the cache sits in line with the database. Every write goes through the cache before updating the database.

This maintains high data consistency between the cache and the database. However, it adds a little latency during the write operations since the data has to be additionally written to the cache. Well, this is a trade-off.

This caching strategy works well for use cases where we need strict data consistency between the cache and the database. It is generally used with other caching strategies to achieve optimized performance.

**Write-back**[#](https://www.educative.io/module/lesson/web-application-architecture-101/B1V5DKonx2X#Write-back)

This caching strategy helps optimize application hosting costs significantly. In it, the data is directly written to the cache instead of the database, and the cache, after some delay, as per the business logic, writes data to the database.

This is what I pulled off in my stock market game. If there are quite a number of writes in the application, developers can reduce the frequency of database writes to cut down the load on it and the associated write operation costs.

A risk in this approach is if the cache fails before the DB is updated, the data might get lost. Again, this strategy is clubbed with other caching strategies to get the best of all the worlds.

Folks, with this, we are done with the caching mechanism in web applications. Now let’s move on to the world of message queues.