Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 15

Managing State

Pearson

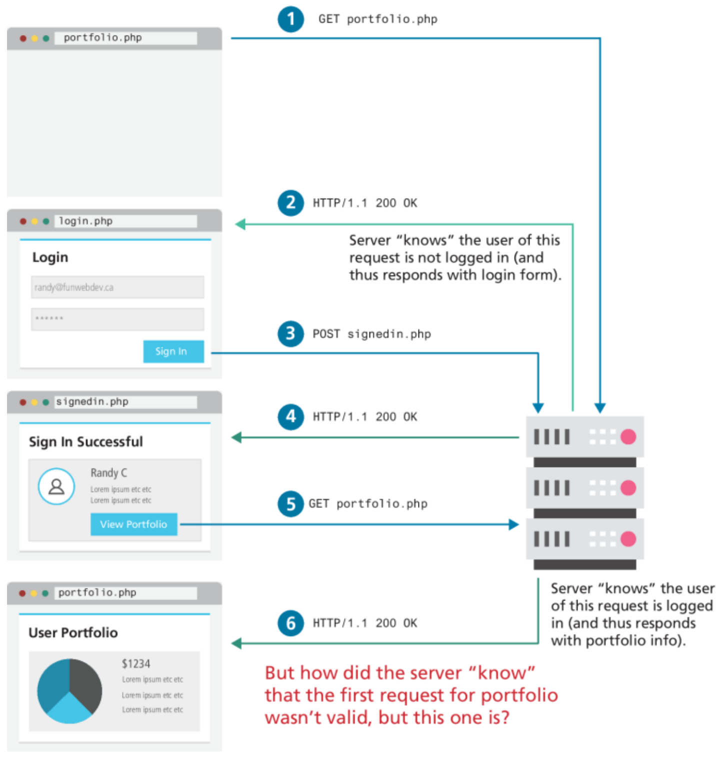
In this chapter you will learn . . .

- Why state is a problem in web application development
- What cookies are and how to use them
- What session state is and what are its typical uses and limitations
- What server cache is and why it is important in real-world websites

The Problem of State in Web Applications

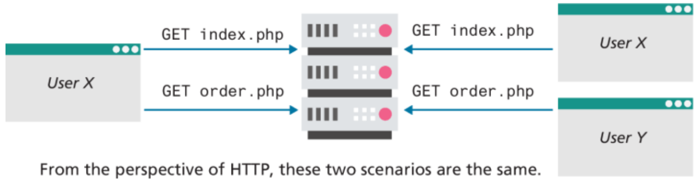
How can one request share information with another request?

The question is: how did the server "know" when the user was and was not logged in? →

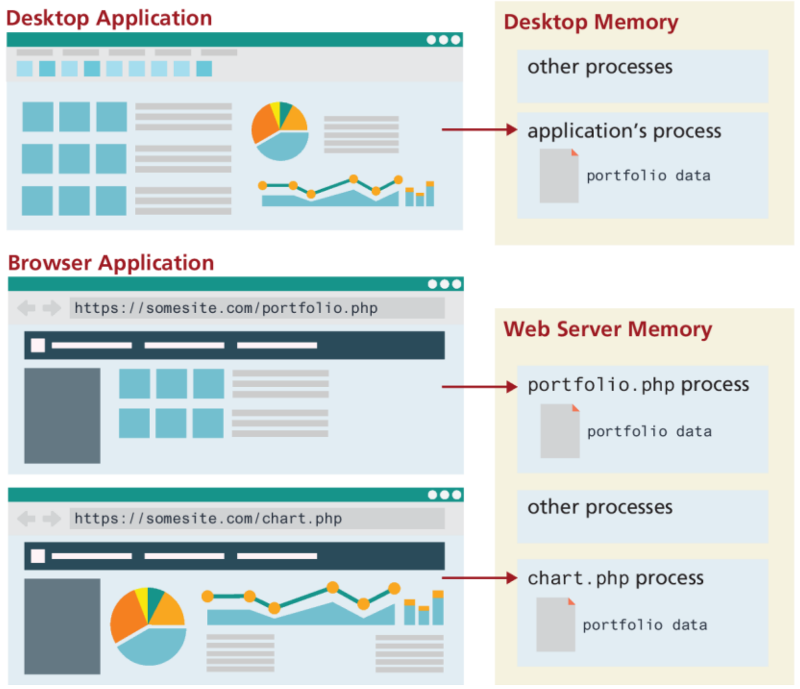


The Problem of State in Web Applications (ii)

Unlike a desktop application, A web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is essentially a request to run a separate program



From the perspective of HTTP, these two scenarios are the same.



Passing Information in HTTP

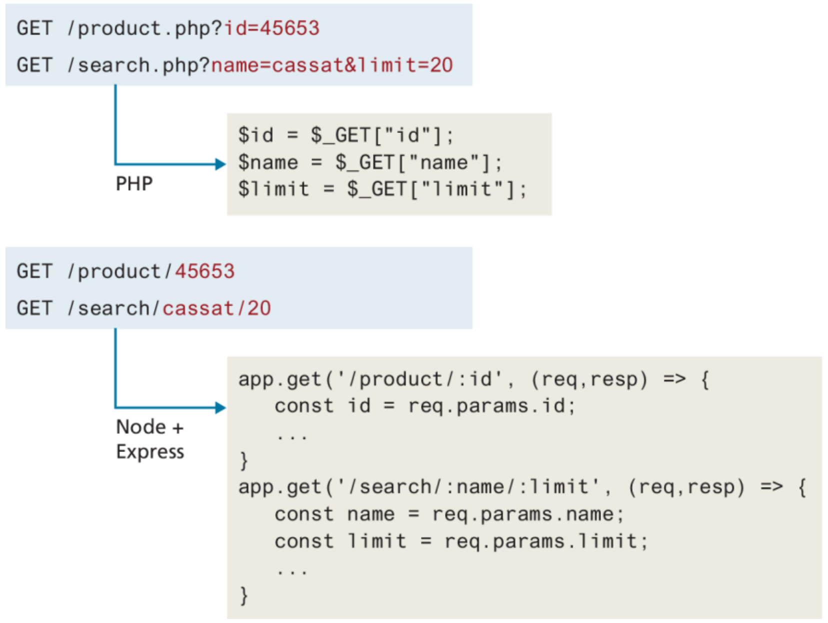
In HTTP, we can pass information using:

- URL
- HTTP header
- Cookies

Passing Information via the URL

Recall a web page can pass query string information from the browser to the server using one of the two methods:

- a query string within the URL **(GET) →**
- a query string within the HTTP header (POST).



```
GET /product.php?id=45653
GET /search.php?name=cassat&limit=20
```

```
$id = $_GET["id"];
$name = $_GET["name"];
$limit = $_GET["limit"];
```

```
GET /product/45653
GET /search/cassat/20
```

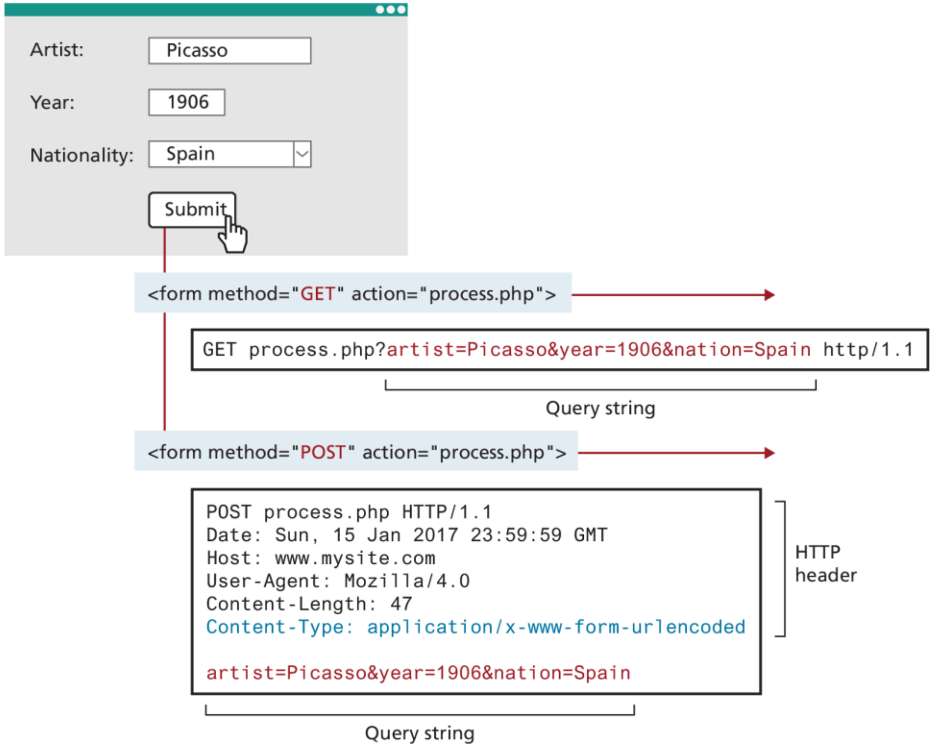
```
app.get('/product/:id', (req,resp) => {
   const id = req.params.id;
   ...
}
app.get('/search/:name/:limit', (req,resp) => {
   const name = req.params.name;
   const limit = req.params.limit;
   ...
}
```

Passing Information via HTTP Header

You can see that the form data sent using the POST method is sent as a query string after the HTTP header.

Think of this data being passed via the HTTP header

Another way that a browser can send data to the server is via JSON data


Artist:
Picasso
Year:
1906
Nationality:
Spain
Submit
<form method="GET" action="process.php">
Query string
<form method="POST" action="process.php">
HTTP header
Query string

```
GET process.php?artist=Picasso&year=1906&nation=Spain http/1.1
```

```
POST process.php HTTP/1.1
Date: Sun, 15 Jan 2017 23:59:59 GMT
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 47
Content-Type: application/x-www-form-urlencoded

artist=Picasso&year=1906&nation=Spain
```

Cookies

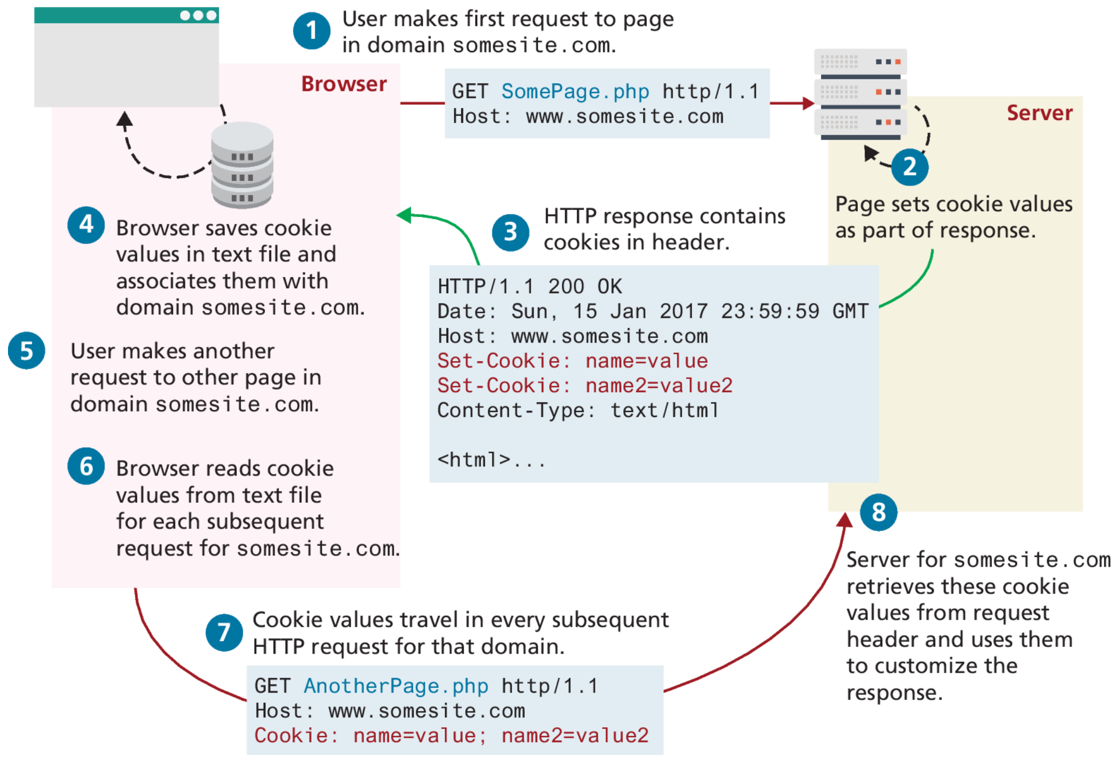
Cookies are a client-side approach for persisting state information.

They are name=value pairs that are saved within one or more text files that are managed by the browser.

While cookies can be used for any state-related purpose, they are principally used as a way of maintaining continuity over time in a web application. One typical use of cookies in a website is to "remember" the visitor so that the server can customize the site for the user.

Cookies are also frequently used to keep track of whether a user has logged into a site.

How Do Cookies Work?



Browser
Server
1
User makes first request to page in domain somesite.com.
2
Page sets cookie values as part of response.
3
HTTP response contains cookies in header.
4
Browser saves cookie values in text file and associates them with domain somesite.com.
5
User makes another request to other page in domain somesite.com.
6
Browser reads cookie values from text file for each subsequent request for somesite.com.
7
Cookie values travel in every subsequent HTTP request for that domain.
8
Server for somesite.com retrieves these cookie values from request header and uses them to customize the response.

```
GET SomePage.php http/1.1
Host: www.somesite.com
```

```
HTTP/1.1 200 OK
Date: Sun, 15 Jan 2017 23:59:59 GMT
Host: www.somesite.com
Set-Cookie: name=value
Set-Cookie: name2=value2
Content-Type: text/html

<html>...
```

```
GET AnotherPage.php http/1.1
Host: www.somesite.com
Cookie: name=value; name2=value2
```

Using Cookies in PHP

Cookies in PHP are *created* using the setcookie() function and are *retrieved* using the $_COOKIES superglobal associative array,

It is important to note that cookies must be written before any other page output.

```
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
```

LISTING 15.2 Writing a cookie

```
if ( !isset($_COOKIE['username']) ) { echo "this cookie doesn't exist"; }
else {
  echo "The username retrieved from the cookie is:“. $_COOKIE['username’];
}

// loop through all cookies in request
foreach ($_COOKIE as $name => $value) {
                echo "Cookie: $name = $value";
}
```

LISTING 15.3 Reading a cookie

Using Cookies in Node and Express

Cookie support in Node and Express (cookie-parser) needs to be installed using npm.

- save JSON data as a cookie value
- read and write *signed* cookies

```
const express = require('express');
const app = express();
const cookieParser = require('cookie-parser');
app.use( cookieParser() );
app.get('/', (req, resp) => { // retrieve a single named cookie
    console.log( req.cookies.username );
    const entries = Object.entries(req.cookies);
    for (const [name, value] of entries) {// loop through all cookies
        console.log(`${name} = ${value}`)
    }
    // now write new cookie as part of response
    const opts = {
        maxAge: 24 * 60 * 60 * 1000, // 1 day
        httpOnly: true
    }
    resp.cookie( 'theme', 'dark', opts );
    resp.send('content sent to browser');
});
```

LISTING 15.4 Using cookies in Node and Express

Persistent Cookie Best Practices

Due to the limitations of cookies your site’s correct operation should not be dependent upon them.

Almost all login systems are dependent upon IDs sent in session cookies

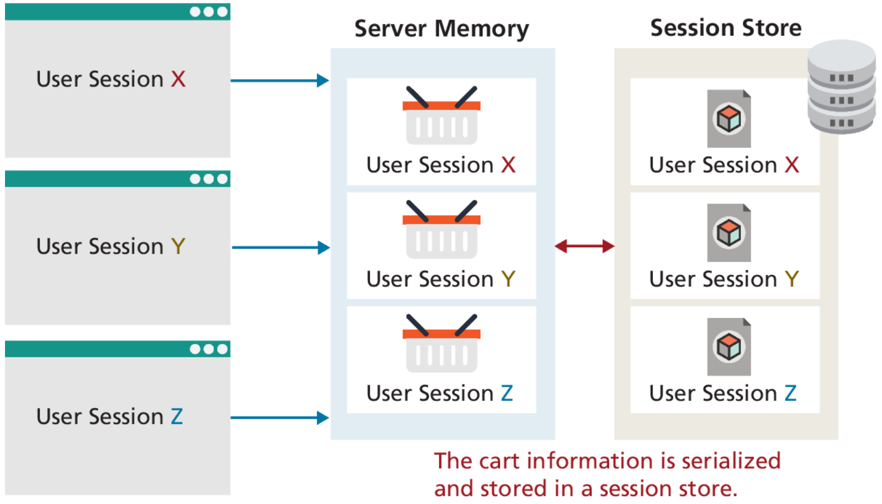
Cookies containing sensitive information should have a short lifetime

A login cookie might contain the username but not the password. Instead, the login cookie would contain a random token used by the site’s back-end database. Every time the user logs in, a new token would be generated and stored in the database and cookie.

Session State

Session state is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

Session state is dependent upon some type of **session store**, that is, some type of storage area for session information.

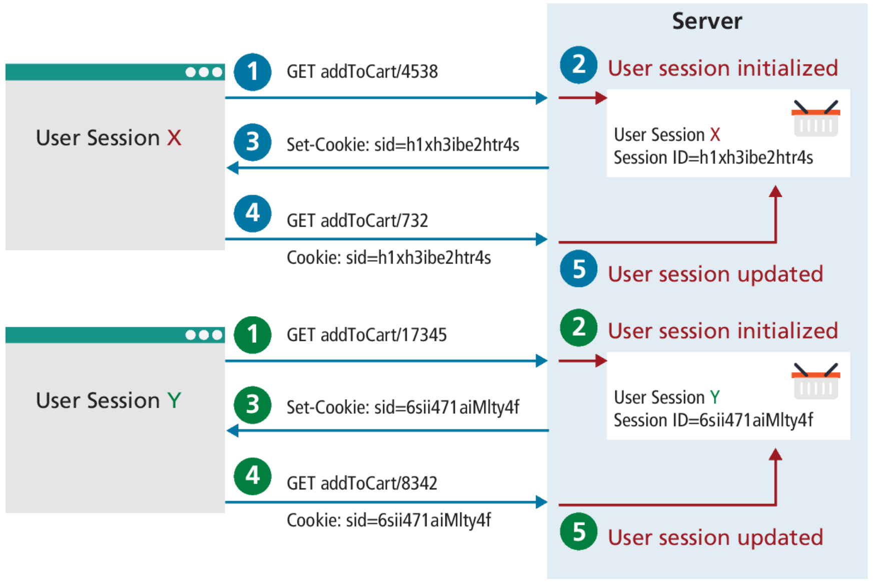


How Does Session State Work?

Since HTTP is stateless, some type of user/session identification system is needed.

Sessions in PHP and Express are identified with a unique session ID.

This session ID transmitted back and forth between the user and the server via a session cookie



Server
User Session X
1 GET addToCart/4538
2 User session initialized
User Session X
Session ID=h1xh3ibe2htr4s
3 Set-Cookie: sid=h1xh3ibe2htr4s
4 GET addToCart/732
Cookie: sid=h1xh3ibe2htr4s
5 User session updated
User Session Y
1 GET addToCart/17345
2 User session initialized
User Session Y
Session ID=6sii471aiMlty4f
3 Set-Cookie: sid=6sii471aiMlty4f
4 GET addToCart/8342
Cookie: sid=6sii471aiMlty4f
5 User session updated

Session Storage and Configuration

In the example shown in Figure 15.8, each user’s session information is kept in serialized files

The decision to save sessions to files rather than in memory addresses the issue of memory usage that can occur on shared hosts as well as persistence between restarts.

One downside to storing the sessions in files is a degradation in performance compared to memory storage.

Session State in PHP

Session in PHP can be accessed via the $_SESSION variable, but unlike the other superglobals, you have to take additional steps first.

You must call the **session_start()** function at the beginning of the script

If the session object does not yet exist one might generate an error, redirect to another page, or create the required object

```
<?php
include_once("ShoppingCart.class.php");
session_start();
// check for existence of session object before accessing
if ( !isset($_SESSION["Cart"]) ) {
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 15.8 Checking session existence

Session State in Node

There are two commonly used session packages for Express: express-session and cookie-session.

The express-session package supports different session stores, whether they be memory, files, external caches, or databases.

Listing 15.9 demonstrates how the express-session package could implement a simple favorites list.

```
const session = require('express-session');
// configure session middleware
app.use(session({
    secret: process.env.SESSION_SECRET,
    saveUninitialized: true,
    resave: true,
    cookie: { secure: true, htttpOnly: true }
}));

app.get('/addToFavorites/:prodid', function(req, resp) {
    if (req.session.cart) {
        const favorites = req.session.favorites;
        favorites.push( req.params.prodid );
    } else {
        req.session.favorites = [ req.params.prodid ];
    }
    // send message or do something else
    …
}
```

LISTING 15.9 Using express-session

Express-store mechanisms

The default express-session store mechanism is server memory, which isn't suitable for production environments. There are dozens of compatible session store packages that allow you to use a wide range of databases and cloud services for your session store. For instance, to configure MongoDB along with Mongoose, you can simply modify your session configuration as follows:

```
const mongoose = require('mongoose');
mongoose.connect(connectionOptions);
const MongoStore = require('connect-mongo')(session);
app.use(session({
    ...
    store: new MongoStore({ mongooseConnection: mongoose.connection })
}));
```

Caching

As you learned back in Chapter 2, your browser uses caching to speed up the user experience. In Chapter 10, you learned about the Web Storage API, which provides a JavaScript-accessible cache managed by the browser.

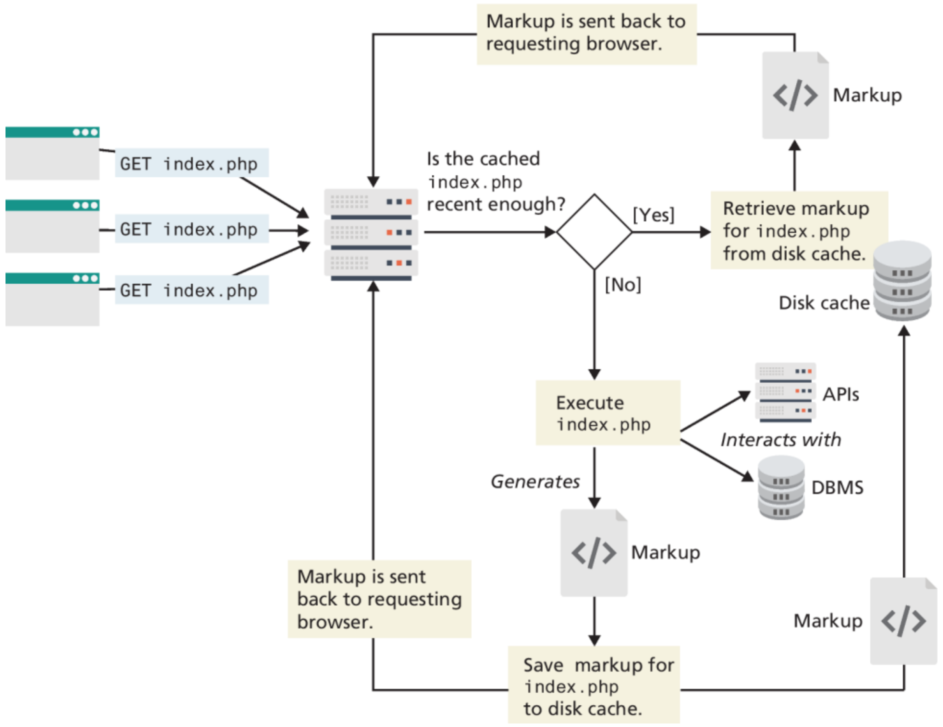
Caching is just as important on the server-side.

Every time a PHP page is requested, it must be fetched, parsed, and executed by the PHP engine. Dynamic generation of that page may become unsustainable under high traffic load.

One way to address this problem is to **cache** the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page.

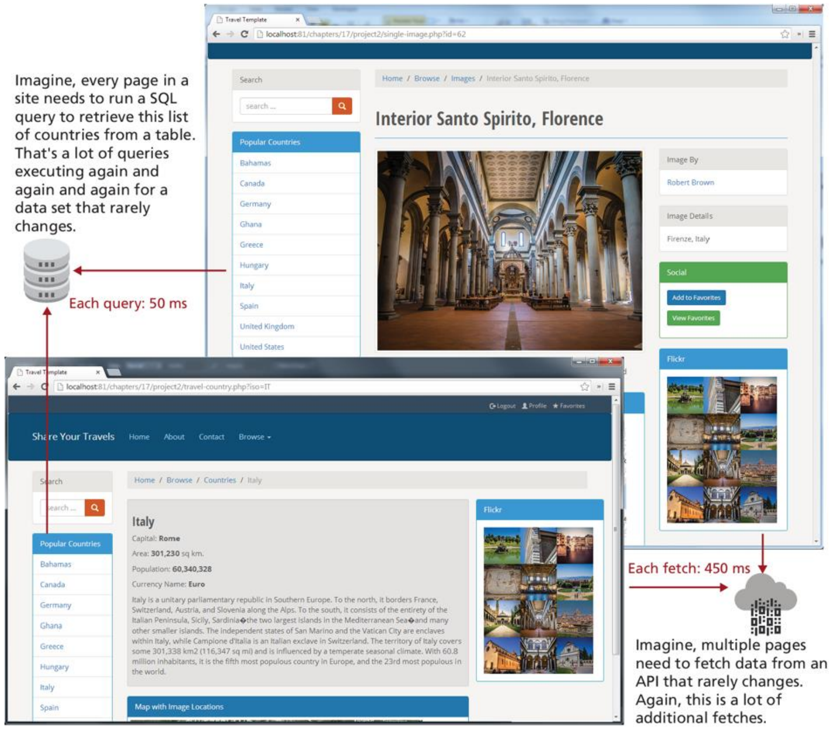
Page Output Caching

Page output caching saves the rendered output of a page (or part of a page) and reuses the output instead of reprocessing the page when a user requests the page again.



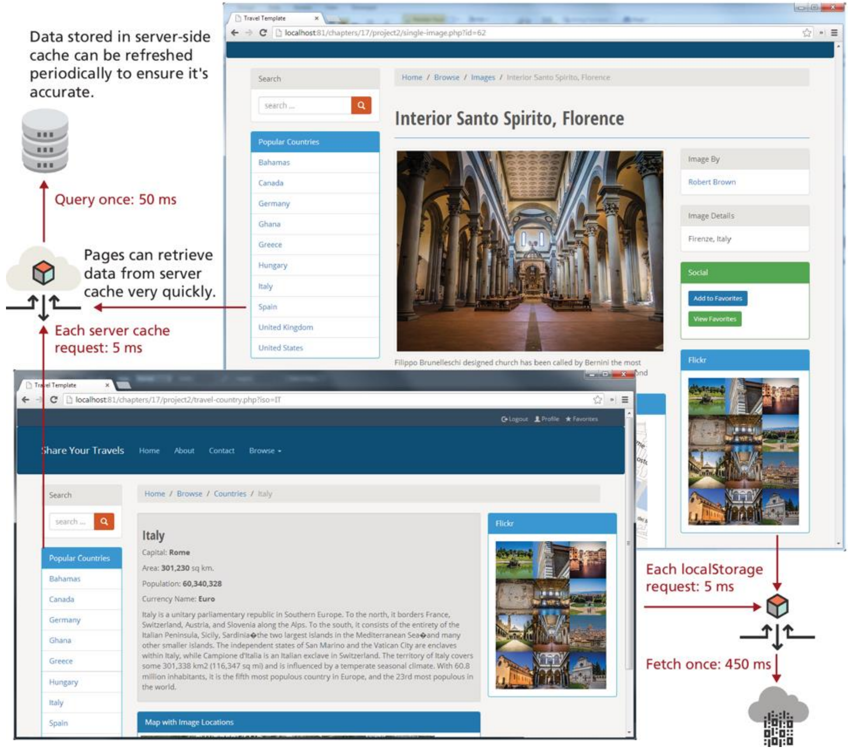
Markup is sent back to requesting browser.
Markup
GET index.php
GET index.php
GET index.php
Is the cached index.php recent enough?
[Yes]
Retrieve markup for index.php from disk cache.
[No]
Disk cache
Execute index.php
APIs
Interacts with
DBMS
Generates
Markup
Markup is sent back to requesting browser.
Markup
Save markup for index.php to disk cache.

Use case for caching



Imagine, every page in a site needs to run a SQL query to retrieve this list of countries from a table. That's a lot of queries executing again and again and again for a data set that rarely changes.

Imagine, multiple pages need to fetch data from an API that rarely changes. Again, this is a lot of additional fetches.



Data stored in server-side cache can be refreshed periodically to ensure it's accurate.

Pages can retrieve data from server cache very quickly.

Each server cache request: 5 ms
Each localStorage request: 5 ms
Fetch once: 450 ms

Application Data Caching

One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.

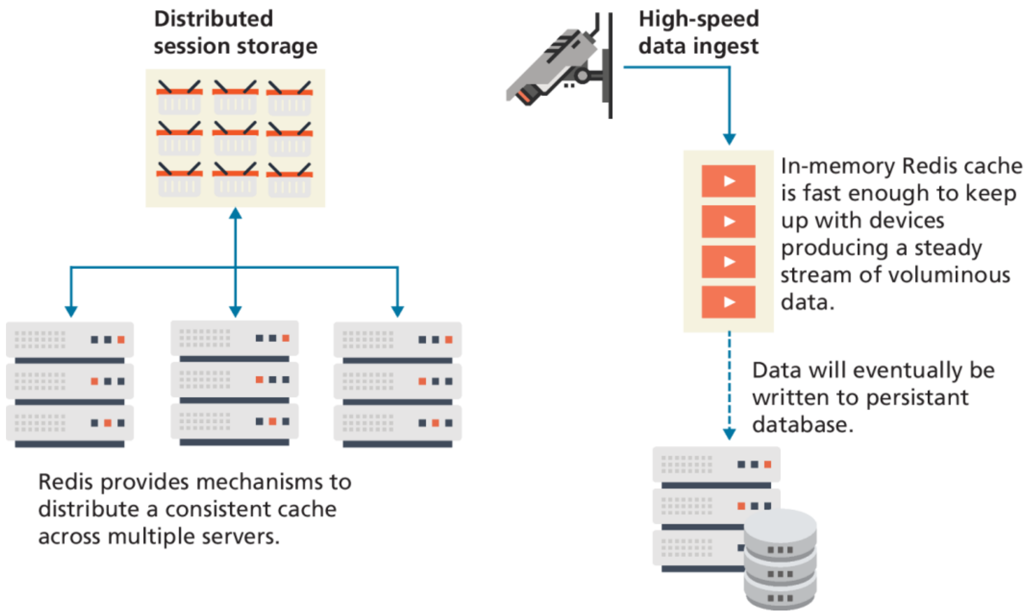
Application data caching allows the developer to programmatically cache data commonly used collections of data.

Then other pages that also need that same data can use the cache version rather than reretrieve it from its original location

Redis as Caching Service

Redis is a popular in-memory key/value noSQL database that is frequently used for distributed caching.

Redis is an **in-memory database**. This means its speed of search and retrieval is very fast.


Distributed session storage
Redis provides mechanisms to distribute a consistent cache across multiple servers.
High-speed data ingest
In-memory Redis cache is fast enough to keep up with devices producing a steady stream of voluminous data.
Data will eventually be written to persistant database.

Key Terms

application data caching	persistent cookies	write-though cache
cache	serialization	
cookies	session cookie	
data eviction algorithms	session state	
deserialization	session store	
HttpOnly	URL rewriting	
page output caching	write-back cache	

Copyright