Storage Architecture

Presented by

Abdallah Hussein Hefzy

What is storage?

- Any place where data is stored, whether temporarily or permanently.
- Storage doesn't just mean databases; it can also include memory, browsers, files, or databases.
- Each type of storage has different characteristics in terms of speed, data retention time, and cost.
- Choosing the right storage method is an engineering decision that affects performance, security, and scalability.

Why is storage important?

- Without storage, the application will lose its data after every request or server restart.
- Choosing an inappropriate storage method can lead to:
- Slower performance.
- Increased costs.
- Security issues.

- **Examples**:
- Storing access tokens in local storage may expose them to XSS attacks.
- Using a database as a cache in a large application unnecessarily increases the load on it.

HTTP is Stateless... what does that mean?

- Every request that arrives at the server is treated by Laravel as if it's seeing the client for the first time.
- The server has no memory to link a request to subsequent requests from the same user.
- If the user logs in to the first request, the server won't remember this in the next request... unless we store the state somewhere.
- This is precisely where the need for multiple storage mechanisms arises: cookies in the browser, sessions, or Redis on the server.

Temporary vs Persistent Storage

- Volatile example: Online Users Counter —If the server goes down, we can simply start counting again.
- Persistent example: Order data in a food delivery app —It must remain even if the server restarts or crashes.
- Practical rule: If data loss will cause problems for the user or the business → Persistent
- If data loss will be automatically resolved upon recompilation →Volatile is the more economical choice.

Browser Storage

Cookies

- Purpose: To automatically send a small piece of data with each request to the server (such as a Session ID or Token).
- Storage Location: Within the browser, and it's automatically sent in the HTTP headers with every request to the same domain.
- Lifetime: You can control it —Session Cookie (disappears when the browser is locked) or Persistent Cookie (with a specified expiration date).
- Advantages: Sent automatically, supports HttpOnly (blocks JavaScript access) and Secure (HTTPS only).
- Limitations: Very small size (4KB), and it increases the size of each request because it's sent automatically.
- Laravel use case: To store the laravel_session cookie that links the user to their session on the server.

Local Storage

- Purpose: To store data in the browser that remains even after the browser is closed and reopened.
- Storage Location: Only within the browser; it is not sent automatically with requests (it must be read and sent manually by JavaScript).
- Lifetime: Permanent until the user deletes it manually or the code deletes it.
- Advantages: Much larger storage space than cookies10-5) MB (and easier to handle than JavaScript.
- Limitations: Not secure for sensitive data (any JavaScript on the page can read it, even from an XSS attack).
- Laravel use case: To store user interface preferences (theme, language) in a Vue/React SPA built on top of the Laravel API.

Session Storage

- Purpose: To store temporary data specific to a single browser tab.
- Storage Location: Within the browser, tied to the specific tab—not shared across different tabs, even on the same website.
- Lifetime: Expires immediately upon closing the tab or browser.
- Advantages: Ideal for temporary data, such as the steps of a multi-step form before final submission.
- Limitations: Same security limitations as Local Storage, and it's quickly deleted, making it unsuitable for long-term data.
- Laravel use case: To retain the state of a multi-step checkout form until the user clicks 'Complete Order.'

IndexedDB

- There's a fourth browser storage type called IndexedDB —it allows storing much larger amounts of data) not limited to 10-5MB like Local Storage ,(and complex data (Objects, Files) in a structured way, like a small database inside the browser itself.
- You'll mostly encounter it in Offline Apps and PWA (Progressive Web Apps) when the app needs to work without internet.
- Not covering it in depth now, just good to know it exists.

Server Storage

RAM (Memory)

- Purpose: Fastest available storage location —Data is stored directly in server memory while the process is running.
- Advantages: Instant access speed (nanometers), no disk I/O.
- Limitations: Data is lost immediately if the process is closed or the server is restarted, and it is not shared across multiple servers in load balancing mode.
- Laravel use case: The Array Cache Driver is only suitable for local development —not suitable for production if you have more than one server.

Database

- Purpose: Permanent and structured storage of essential application data (Users, Orders, Products, etc.)
- Advantages: Ensures data integrity through Relationships and Constraints, and supports Transactions (the entire process succeeds or fails as a single unit).
- Limitations: Slower than Redis or RAM because it uses disk space, and not ideal for highly recurring temporary data like caches.
- NoSQL in brief: Databases like MongoDB are schema-less, useful for variable-form data, but Laravel primarily integrates with SQL Databases via Eloquent ORM.
- Laravel use case: Storing users, orders, products—any data that needs to remain and be interconnected.

Redis

- Purpose: A high-speed in-memory database primarily used for caching, sessions, and queues.
- Advantages: Speed close to RAM, but with optional disk persistence and shared across all servers.
- Limitations: Data is essentially ephemeral and not a suitable replacement for a full relational database.
- Supports advanced data types: Strings, Hashes, Lists, Sets —not just a simple Key-Value.
- Laravel use case: CACHE_STORE=redis , SESSION_DRIVER=redis , QUEUE_CONNECTION=redis.

File Storage

- Purpose: Storing files on disk —images, PDF documents, log files
- Advantages: Simple, straightforward, and suitable for local development or small projects
- Limitations: If you have more than one server (load balancing), files will not be available to all servers unless you use shared storage space.
- Laravel use case: FilesyStem_Disk=local
- Then in production once the project grows: FilesyStem_Disk=s3

Clarifications and Core Differences

- **Quick note —Memcached**
- There's an older alternative called Memcached —supports Key-Value caching only, without Redis's advanced features (Persistence, Pub/Sub, complex data types). In practice, Redis is almost the only choice in modern Laravel projects.

- **Quick note —HTTP Cache (Browser Cache)**
- Don't confuse this with Redis. HTTP Cache is an automatic cache the browser itself performs for static files (images, CSS, JS), based on HTTP headers like `Cache-Control` dna `ETag` ni uoy yb dellortnoc si ehcaC sideR :ecnereffid yeK .) edoc`Cache::remember()` yllacitamotua sneppah ehcaC PTTH elihw ,(.edoc fo enil elgnis a gnitirw tuohtiw

Cache Patterns

- Cache Aside (most common): The application checks the cache first. If not found (Miss), it goes to the Database, then stores the result in the cache for next time. Laravel supports this easily via Cache::remember.()
- Read Through: Reading always happens through the Cache. If data isn't there, the Cache layer itself (not your application code) fetches it from the Database.
- Write Through: Every write operation is written to the Cache and Database at the same time, keeping data always in sync —at the cost of slightly slower writes.

Cloud Storage

- Amazon S a aiv metsyseliF levaraL htiw yltcerid setargetni ,nommoc tsom ehT :3 .revird edam-ydaer
- Object Storage: Each file is stored as an independent object with a unique key, instead of a complex hierarchical structure.
- Advantages: Scalable almost infinitely, accessible from anywhere via a direct URL.
- Laravel use case: `Storage::disk('s — (htap$)lru<-('3`same upload code works regardless of which driver is active.
- Quick note: Alternatives like Google Cloud Storage and Azure Blob Storage exist too —useful if your infrastructure is already on GCP or Azure.
- **Quick note —CDN**
- In large projects, files on S3 are usually not read directly by the user —they're distributed first through a CDN for faster loading, reduced load on S.noitubirtsid labolg dna ,3

We've Covered All Types of Storage Now Let's see How Laravel Manages Them

Laravel Drivers

- Laravel separates the code you write from the actual data storage location using a layer called Driver.
- The same line, Cache::put(), can store data in Redis, a file, or a database—depending on the Driver specified in the .env file.
- Benefit: You can change the storage location without altering a single line of business logic.
- This configuration is done through the config/cache.php, config/session.php, config/queue.php, and config/filesystems.php files.

```
. //env: CACHE_STORE=file
Cache::put('key', 'value') ;
→ //Stored in a file on disk
. //env: CACHE_STORE=redis
Cache::put('key', 'value') ;
→ //Same line, stored in Redis
```

Where does Laravel store the sessions?

- SESSION_DRIVER specifies where user session data (session) is stored.
- file: Stored in files within storage/framework/sessions —suitable for local development only.
- database: Stored in the sessions table —useful if you need to track and manage sessions from the database.
- redis: The best choice for production, especially if you have multiple load-balanced servers.
- The cookie (laravel_session) loads only the session ID, not the data itself — the data is actually stored on the server.

Where does Laravel store its cache and queue?

- CACHE_STORE specifies where to store temporary, fast data (frequent query results, settings, etc.).
- Common values: file, database, redis, array (for testing only; data is lost after each request).
- QUEUE_CONNECTION specifies where to store deferred jobs before they run in the background.
- Common values: sync (runs immediately, for development), database, redis — Redis is the fastest and most widely used database in production.

Where does Laravel store uploaded files?

- FILESYSTEM_DISK specifies where user-uploaded files (profile pictures, documents, etc.) are stored.
- local: Stored in storage/app —suitable for local development only.
- public: Stored in storage/app/public and directly accessible via a link after creating storage:link.
- s — (ecivres elbitapmoc a ro)3 S nozamA ni derotS :3 the standard production choice for scalable applications.
- The same upload code works with any driver —the only change is in the .env file.

```
. //env: FILESYSTEM_DISK=s3
$path = $request->file('image')
<-    store('products', 's;('3

$url = Storage::disk('s('3
<-    url($path);
```

Security across storage types

- Always use HTTPS —any data sent without it can be read in transit.
- Enable HttpOnly and Secure on any cookie carrying sensitive data —blocks JavaScript access to it.
- Never store sensitive data (passwords, tokens, payment info) in Local Storage — any JavaScript (including from an XSS attack) can access it.
- If sensitive data must be stored, encrypt it (Encryption at Rest) before storing, not after.
- General rule: the closer the data is to the user (browser), the less sensitive it should be.

With Best Wishes to You All

Thanks