

Caching and performance - Course material

<https://github.com/heig-vd-dai-course>

[Markdown](#) · [PDF](#)

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Based on the original course by O. Liechti and J. Ehrensberger.

This work is licensed under the [CC BY-SA 4.0](#) license.



Table of contents

- [Table of contents](#)
- [Objectives](#)
- [Caching](#)
 - [Types of caching](#)
 - [CDN](#)
 - [Where to cache?](#)
- [Managing cache with HTTP](#)
 - [Expiration model](#)
 - [Validation model](#)
 - [Is it possible to use both models?](#)
- [Managing cache with proxies](#)
- [Managing cache with key-value stores](#)
- [Practical content](#)
 - [Update the Main.java class to cache the results](#)
 - [Update the AuthController.java to cache the results](#)
 - [Update the UsersController.java to cache the results](#)
 - [Test the caching system with curl](#)
 - [Test the caching system with a browser](#)
 - [Go further](#)
- [Conclusion](#)
 - [What did you do and learn?](#)
 - [Test your knowledge](#)
- [Finished? Was it easy? Was it hard?](#)
- [What will you do next?](#)
- [Additional resources](#)
- [Sources](#)

Objectives

In this last and final chapter of this course, you will learn about caching and performance.

You will learn about caching, how it can be used to improve the performance of a system, where to cache, how you can manage cache with HTTP and how to implement it in your application.

Caching

Caching is the process of storing data in a cache. A cache is a temporary storage where data is stored so that future requests for that data can be served faster.

Caching can be used to improve the performance of a system by serving cached data instead of processing a request again. Caching significantly improves the performance of a system because it avoids processing the same request multiple times.

This has several advantages:

- The client will receive the response faster, especially when the client itself (browser) has cached the response.
- The server does not have to process the request (query the database, process the data, compose the response, etc).
- The network does not have to carry the messages along the entire path between client and server.

It however introduces some complexity because it is difficult to know when to invalidate a cache. If a cache is not invalidated, it can serve stale data (= outdated data).

Types of caching

Caching can be done on the client-side or on the server-side:

- **Client-side caching** (private caches): once a client has received a response from a server, it can store the response in a cache. The next time the client needs the same resource, it can use the cached response instead of sending a new request to the server.
- **Server-side caching** (shared caches): the server stores data in a cache with the help of a reverse proxy or by the web application. The next time the server needs the same resource, it can use the cached response instead of processing the request again.

CDN

Content delivery networks (CDNs) are a type of cache that can be used to serve static content (e.g. images, videos, etc.) to clients.

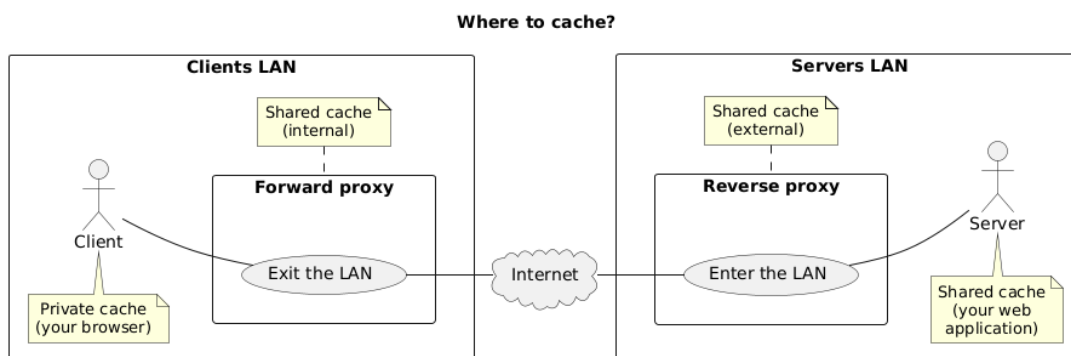
A CDN is a geographically distributed network of proxy servers and their data centers.

A CDN can be used to improve the performance of a system by serving static content to clients from the closest server for clients all around the world.

Where to cache?

Caching can be done on the client-side, on the server-side, or on a CDN.

Private caches are caches that are only used by one client. Public caches are caches that are used by multiple clients.



The best would be to cache at each level of the system to ensure the best performance. But it is not always possible or faisable.

In the context of this course, we will focus on server-side caching.

Managing cache with HTTP

Managing cache is challenging because it is difficult to know when to invalidate a cache. If a cache is not invalidated, it can serve stale data.

There are two main caching models:

- **Expiration model:** the cache is valid for a certain amount of time.
- **Validation model:** the cache is valid until the data is modified.

Expiration and validation are two mechanisms that can be used to control caching.

Expiration is the process of specifying how long a response can be cached.

Validation is the process of checking if a cached response is still valid.

Both can be used at the same time to improve the performance of the system.

Much more details about caching with HTTP can be found on MDN Web Docs: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>.

Tip

MDN Web Docs is a great resource to learn about web technologies. It is maintained by the Mozilla Foundation and is considered a reliable source of information.

If you ever have a question about a web technology, you can check MDN Web Docs to find the answer.

Expiration model

The expiration model is the simplest caching model. It is described in [RFC 2616](#).

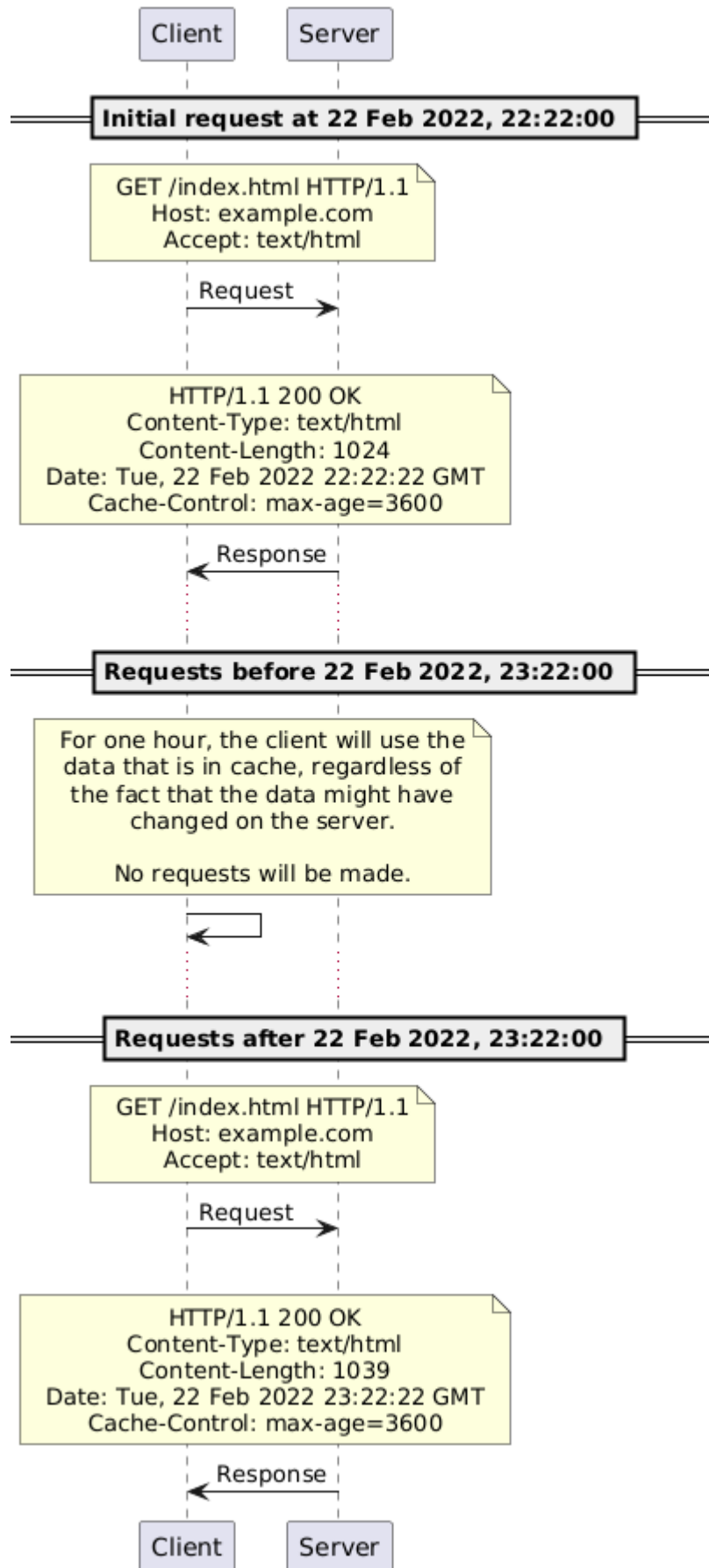
The cache is invalidated after a certain amount of time. The cache can be invalidated after a certain amount of time because the data is not expected to change.

The expiration model can be used to cache static content (e.g. images, videos, etc.) or to cache responses from servers to improve the performance of the system.

The expiration model can be implemented with the following header:

- Cache-Control: max-age=<number of seconds>: specifies the maximum amount of seconds a resource will be considered fresh. and responses.

Expiration model



Validation model

The validation model is more complex than the expiration model. It is described in [RFC 2616](#).

The cache is invalidated when the data is modified. The cache can be invalidated when the data is modified because the data is expected to change.

The validation model can be used to cache responses from servers to improve the performance of the system.

The main idea of the validation model is:

1. Send a request to the server to check if the data has changed.
2. If the data has not changed, the server can return a 304 Not Modified response to the client.
3. If the data has changed, the server can return a 200 OK response to the client with the new data.

The request to check if the data has changed is called a **conditional request**.

When clients want to update a resource, they can send a conditional request to the server to check if the data has changed since the last time it was modified.

If the data has not changed since the last modification. The server accepts the changes, update the resource and returns a 200 OK response.

If the data has changed, it means someone else has updated the data prior to our changes. The server cannot accept the changes and returns a 412 Precondition Failed response to the client.

Tip

Some common terms used in the context of caching are **cache hit** and **cache miss**.

A **cache hit** occurs when the cache contains the requested data and can return it without accessing the origin server.

A **cache miss** occurs when the cache does not contain the requested data and must access the origin server to get it.

You will see these terms used in the context of caching in the following diagrams and explanations.

There are two types of conditional requests:

- **Based on the Last-Modified header:** allows a 304 Not Modified to be returned if content is unchanged since the last time it was modified (= a cache hit)
- **Based on the ETag header:** allows a 304 Not Modified to be returned if content is unchanged for the version/hash of the given entity (= a cache hit)

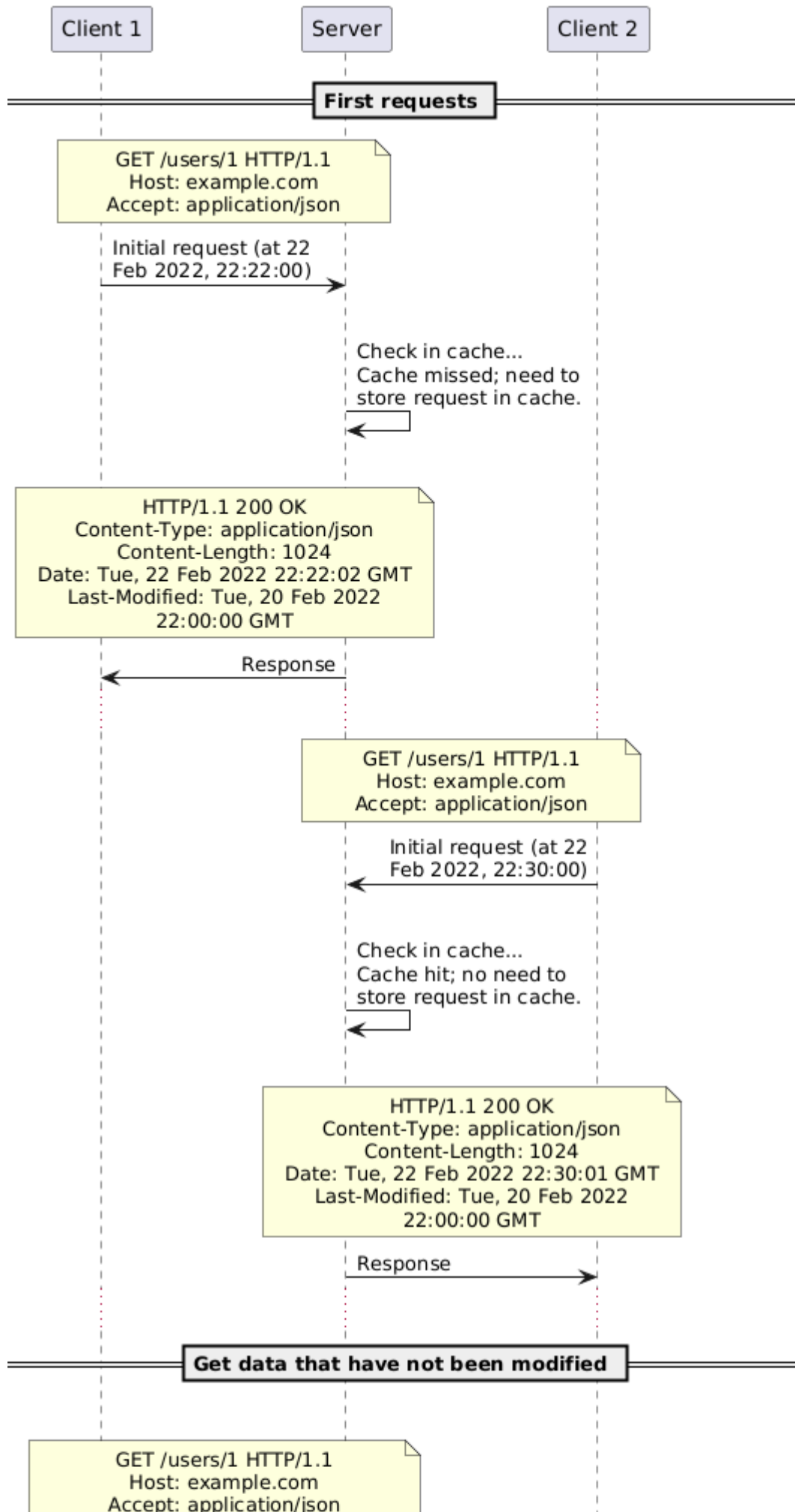
Based on the Last-Modified header

With HTTP, the validation model based on the Last-Modified header can be implemented with the following headers:

- Last-Modified: indicates the date and time at which the resource was last updated.
- If-Modified-Since: returns a 304 Not Modified if content is unchanged since the last known time (= a cache hit).
- If-Unmodified-Since: returns a 412 Precondition Failed if content has changed since the last known time (= a cache miss) **when you try to update/delete the resource.**

The Last-Modified header is used to check if the data has changed since the last time it was modified.

Validation model based on the Last-Modified header



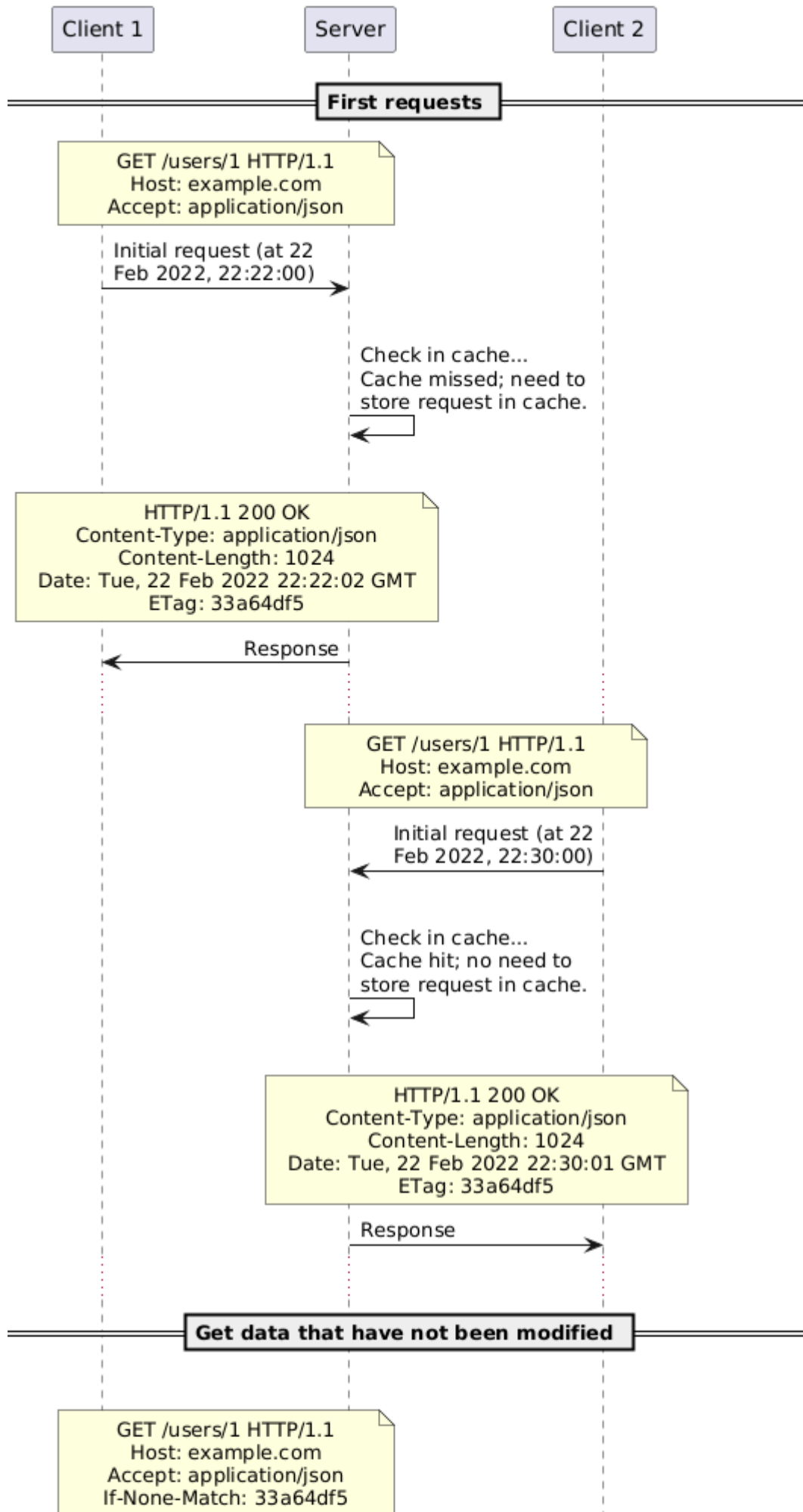
Based on the ETag header

With HTTP, the validation model based on the ETag header can be implemented with the following headers:

- ETag: provides the current entity tag for the selected representation. Think of it like a version number or a hash for the given resource.
- If-None-Match: returns a 304 Not Modified if content is unchanged for the entity specified (ETag) (= a cache hit).
- If-Match: returns a 412 Precondition Failed if content is changed for the entity specified (ETag) (= a cache miss) **when you try to update/delete the resource.**

The ETag header is used to check if the data has changed since the last time it was modified.

Validation model based on the ETag header



Is it possible to use both models?

Yes, it is possible to use the expiration model and the validation model at the same time.

It can improve the performance even more because a client will not even attempt to send a request to the server if the expiration time is not reached. But once the expiration time is reached, the client will send a validation request to the server to check if the data has changed.

Managing cache with proxies

A forward/reverse proxy can be used to manage cache with HTTP as well. A forward/reverse proxy can cache responses from clients/servers to improve the performance of the system.

Traefik, for example, can be used to cache responses from servers. It is available as a middleware in their Enterprise version. You can learn more about it in their documentation at <https://doc.traefik.io/traefik-enterprise/middlewares/http-cache/>.

As it is out of the scope/reach for this course, we will not go into details about how to configure Traefik to cache responses from servers. We will, however, implement it on the server side with Javalin.

Managing cache with key-value stores

A key-value store is a type of database that stores data as a collection of key value pairs.

A key-value store can be used to manage cache with HTTP. A key-value store can cache responses from clients/servers to improve the performance of the system.

Redis, for example, can be used as a key-value store to cache responses from servers. You can learn more about it in their documentation at <https://redis.io/documentation>.

As it is out of the scope/reach for this course, we will not go into details about how to configure Redis to cache responses from servers. We will, however, implement it on the server side with Javalin.

Practical content

In this practical content, you will implement the validation model based on the Last-Modified header in your application.

You will need the results of the practical content from chapter [HTTP and curl](#).

If you do not have the results of the practical content from chapter HTTP and curl, you can use the solution mentioned in the HTTP and curl chapter. Clone the solution to have the project ready for this practical content.

Update the Main.java class to cache the results

Update your Main.java class to add a Map to cache results to your application:

```
diff --git a/23-caching-and-performance/src/main/java/ch/heigvd/dai/Main.java b/23-caching-and-performance/src/main/java/ch/heigvd/dai/Main.java
```

```
index d4aae20..cc64e48 100644
```

```
--- a/23-caching-and-performance/src/main/java/ch/heigvd/dai/Main.java
```

```
+++ b/23-caching-and-performance/src/main/java/ch/heigvd/dai/Main.java
```

```
@@ -1,36 +1,49 @@
```

```
package ch.heigvd.dai;
```

```
import ch.heigvd.dai.auth.AuthController;
```

```
import ch.heigvd.dai.users.User;
```

```
import ch.heigvd.dai.users.UsersController;
```

```
import io.javalin.Javalin;
```

```
+import java.time.LocalDateTime;
```

```
import java.util.concurrent.ConcurrentHashMap;
```

```
public class Main {
```

```
    public static final int PORT = 8080;
```

```
    public static void main(String[] args) {
```

```
        - Javalin app = Javalin.create();
```

```
        + Javalin app =
```

```
+     Javalin.create(  
+         // Add custom configuration to Javalin  
+         config -> {  
+             // This will allow us to parse LocalDateTime  
+             config.validation.register(LocalDateTime.class, LocalDateTime::parse);  
+         });  
  
// This will serve as our database  
ConcurrentHashMap<Integer, User> users = new ConcurrentHashMap<>();  
  
+ // This will serve as our cache  
+ //  
+ // The key is to identify the user(s)  
+ // The value is the last modification time of the user(s)  
+ ConcurrentHashMap<Integer, LocalDateTime> usersCache = new  
+     ConcurrentHashMap<>();  
+  
// Controllers  
- AuthController authController = new AuthController(users);  
- UsersController usersController = new UsersController(users);  
+ AuthController authController = new AuthController(users, usersCache);  
+ UsersController usersController = new UsersController(users, usersCache);  
  
// Auth routes  
app.post("/login", authController::login);  
app.post("/logout", authController::logout);  
app.get("/profile", authController::profile);  
  
// Users routes  
app.post("/users", usersController::create);  
app.get("/users", usersController::getMany);  
app.get("/users/{id}", usersController::getOne);  
app.put("/users/{id}", usersController::update);  
app.delete("/users/{id}", usersController::delete);  
  
app.start(PORT);  
}  
}
```

In this code snippet, we have added a Map to cache results to your application.

The key is to identify the user(s) and the value is the last modification time of the user(s).

Javalin does not support `LocalDateTime` (the class representing a date) by default. We have added a custom configuration to Javalin to parse `LocalDateTime`.

Update the `AuthController.java` to cache the results

Update the `AuthController.java` to use the `Map` to cache results to your application:

```
diff --git a/21-http-and-curl/src/main/java/ch/heigvd/dai/auth/AuthController.java b/21-http-and-curl/src/main/java/ch/heigvd/dai/auth/AuthController.java
```

```
index 08c8670..83db13b 100644
```

```
--- a/21-http-and-curl/src/main/java/ch/heigvd/dai/auth/AuthController.java
```

```
+++ b/21-http-and-curl/src/main/java/ch/heigvd/dai/auth/AuthController.java
```

```
@@ -1,55 +1,81 @@
```

```
package ch.heigvd.dai.auth;
```

```
import ch.heigvd.dai.users.User;
```

```
import io.javalin.http.*;
```

```
+import java.time.LocalDateTime;
```

```
import java.util.concurrent.ConcurrentHashMap;
```

```
public class AuthController {
```

```
    private final ConcurrentHashMap<Integer, User> users;
```

```
+    private final ConcurrentHashMap<Integer, LocalDateTime> usersCache;
```

```
+
```

```
-    public AuthController(ConcurrentHashMap<Integer, User> users) {
```

```
+    public AuthController(
```

```
+        ConcurrentHashMap<Integer, User> users,
```

```
+        ConcurrentHashMap<Integer, LocalDateTime> usersCache) {
```

```
        this.users = users;
```

```
+        this.usersCache = usersCache;
```

```
    }
```

```
    public void login(Context ctx) {
```

```
        User loginUser =
```

```
            ctx.bodyValidator(User.class)
```

```

        .check(obj -> obj.email != null, "Missing email")
        .check(obj -> obj.password != null, "Missing password")
        .get();

for (User user : users.values()) {
    if (user.email.equalsIgnoreCase(loginUser.email)
        && user.password.equals(loginUser.password)) {
        ctx.cookie("user", String.valueOf(user.id));
        ctx.status(HttpStatus.NO_CONTENT);
        return;
    }
}

throw new UnauthorizedResponse();
}

public void logout(Context ctx) {
    ctx.removeCookie("user");
    ctx.status(HttpStatus.NO_CONTENT);
}

public void profile(Context ctx) {
    String userIdCookie = ctx.cookie("user");

    if (userIdCookie == null) {
        throw new UnauthorizedResponse();
    }

    Integer userId = Integer.parseInt(userIdCookie);

+ // Get the last known modification date of the user
+ LocalDateTime lastKnownModification =
+     ctx.headerAsClass("If-Modified-Since", LocalDateTime.class).getOrDefault(null);
+
+ // Check if the user has been modified since the last known modification date
+ if (lastKnownModification != null &&
+     usersCache.get(userId).equals(lastKnownModification)) {
+     throw new NotModifiedResponse();
+ }
+

```

```

    User user = users.get(userId);

    if (user == null) {
        throw new UnauthorizedResponse();
    }

+   LocalDateTime now;
+   if (usersCache.containsKey(user.id)) {
+       // If it is already in the cache, get the last modification date
+       now = usersCache.get(user.id);
+   } else {
+       // Otherwise, set to the current date
+       now = LocalDateTime.now();
+       usersCache.put(user.id, now);
+   }
+
+   // Add the last modification date to the response
+   ctx.header("Last-Modified", String.valueOf(now));
    ctx.json(user);
}
}

```

In this code snippets, we have updated the AuthController.java to:

1. Use the Map to cache the results of your application
2. Store results in the cache
3. Return the Last-Modified header
4. Validate the cache with the If-Modified-Since header
5. Validate the cache with the If-Unmodified-Since header

Update the UsersController.java to cache the results

Update the UsersController.java to use the Map to cache the results to your application:

```

diff --git a/21-http-and-curl/src/main/java/ch/heigvd/dai/users/UsersController.java
      b/21-http-and-curl/src/main/java/ch/heigvd/dai/users/
      UsersController.java

```

```

index 76bca68..e66cc00 100644

```

```

--- a/21-http-and-curl/src/main/java/ch/heigvd/dai/users/UsersController.java
+++ b/21-http-and-curl/src/main/java/ch/heigvd/dai/users/UsersController.java
@@ -1,117 +1,221 @@

```

```
package ch.heigvd.dai.users;

import io.javalin.http.*;
+import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;

public class UsersController {
    private final ConcurrentHashMap<Integer, User> users;
    private final AtomicInteger userId = new AtomicInteger(1);

+ private final ConcurrentHashMap<Integer, LocalDateTime> usersCache;
+
+ // This is a magic number used to store the users' list last modification date
+ // As the ID for users starts from 1, it is safe to reserve the value -1 for all users
+ private final Integer RESERVED_ID_TO_IDENTIFY_ALL_USERS = -1;
+
- public UsersController(ConcurrentHashMap<Integer, User> users) {
+ public UsersController(
+     ConcurrentHashMap<Integer, User> users,
+     ConcurrentHashMap<Integer, LocalDateTime> usersCache) {
    this.users = users;
+ this.usersCache = usersCache;
}

public void create(Context ctx) {
    User newUser =
        ctx.bodyValidator(User.class)
            .check(obj -> obj.firstName != null, "Missing first name")
            .check(obj -> obj.lastName != null, "Missing last name")
            .check(obj -> obj.email != null, "Missing email")
            .check(obj -> obj.password != null, "Missing password")
            .get();

    for (User user : users.values()) {
        if (user.email.equalsIgnoreCase(newUser.email)) {
            throw new ConflictResponse();
        }
    }
}
```

```
}

User user = new User();

user.id = userId.getAndIncrement();
user.firstName = newUser.firstName;
user.lastName = newUser.lastName;
user.email = newUser.email;
user.password = newUser.password;

users.put(user.id, user);

+ // Store the last modification date of the user
+ LocalDateTime now = LocalDateTime.now();
+ usersCache.put(user.id, now);
+
+ // Invalidate the cache for all users
+ usersCache.remove(RESERVED_ID_TO_IDENTIFY_ALL_USERS);
+
+ ctx.status(HttpStatus.CREATED);
+
+ // Add the last modification date to the response
+ ctx.header("Last-Modified", String.valueOf(now));
+
+ ctx.json(user);
}

public void getOne(Context ctx) {
    Integer id = ctx.pathParamAsClass("id", Integer.class).get();

+ // Get the last known modification date of the user
+ LocalDateTime lastKnownModification =
+     ctx.headerAsClass("If-Modified-Since", LocalDateTime.class).getOrDefault(null);
+
+ // Check if the user has been modified since the last known modification date
+ if (lastKnownModification != null &&
+     usersCache.get(id).equals(lastKnownModification)) {
+     throw new NotModifiedResponse();
+ }
+ }
```

```
User user = users.get(id);

if (user == null) {
    throw new NotFoundResponse();
}

+ LocalDateTime now;
+ if (usersCache.containsKey(user.id)) {
+     // If it is already in the cache, get the last modification date
+     now = usersCache.get(user.id);
+ } else {
+     // Otherwise, set to the current date
+     now = LocalDateTime.now();
+     usersCache.put(user.id, now);
+ }
+
+ // Add the last modification date to the response
+ ctx.header("Last-Modified", String.valueOf(now));
+ ctx.json(user);
}

public void getMany(Context ctx) {
+ // Get the last known modification date of all users
+ LocalDateTime lastKnownModification =
+     ctx.headerAsClass("If-Modified-Since", LocalDateTime.class).getOrDefault(null);
+
+ // Check if all users have been modified since the last known modification date
+ if (lastKnownModification != null
+     && usersCache.containsKey(RESERVED_ID_TO_IDENTIFY_ALL_USERS)
+     && usersCache.get(RESERVED_ID_TO_IDENTIFY_ALL_USERS).equals(lastKnownModification)) {
+     throw new NotModifiedResponse();
+ }
+
+ String firstName = ctx.queryParam("firstName");
+ String lastName = ctx.queryParam("lastName");

+ List<User> users = new ArrayList<>();

+ for (User user : this.users.values()) {
+     if (firstName != null && !user.firstName.equalsIgnoreCase(firstName)) {
```



```
        continue;
    }

    if (lastName != null && !user.lastName.equalsIgnoreCase(lastName)) {
        continue;
    }

    users.add(user);
}

+ LocalDateTime now;
+ if (usersCache.containsKey(RESERVED_ID_TO_IDENTIFY_ALL_USERS)) {
+     // If it is already in the cache, get the last modification date
+     now = usersCache.get(RESERVED_ID_TO_IDENTIFY_ALL_USERS);
+ } else {
+     // Otherwise, set to the current date
+     now = LocalDateTime.now();
+     usersCache.put(RESERVED_ID_TO_IDENTIFY_ALL_USERS, now);
+ }
+
+ // Add the last modification date to the response
+ ctx.header("Last-Modified", String.valueOf(now));
+ ctx.json(users);
+ }

public void update(Context ctx) {
    Integer id = ctx.pathParamAsClass("id", Integer.class).get();

+ // Get the last known modification date of the user
+ LocalDateTime lastKnownModification =
+     ctx.headerAsClass("If-Unmodified-Since", LocalDateTime.class).getOrDefault(null);
+
+ // Check if the user has been modified since the last known modification date
+ if (lastKnownModification != null && !
+     usersCache.get(id).equals(lastKnownModification)) {
+     throw new PreconditionFailedResponse();
+ }
+
    User updateUser =
        ctx.bodyValidator(User.class)
```

```
.check(obj -> obj.firstName != null, "Missing first name")
.check(obj -> obj.lastName != null, "Missing last name")
.check(obj -> obj.email != null, "Missing email")
.check(obj -> obj.password != null, "Missing password")
.get();

User user = users.get(id);

if (user == null) {
    throw new NotFoundResponse();
}

user.firstName = updateUser.firstName;
user.lastName = updateUser.lastName;
user.email = updateUser.email;
user.password = updateUser.password;

users.put(id, user);

+ LocalDateTime now;
+ if (usersCache.containsKey(user.id)) {
+     // If it is already in the cache, get the last modification date
+     now = usersCache.get(user.id);
+ } else {
+     // Otherwise, set to the current date
+     now = LocalDateTime.now();
+     usersCache.put(user.id, now);
+
+     // Invalidate the cache for all users
+     usersCache.remove(RESERVED_ID_TO_IDENTIFY_ALL_USERS);
+ }
+
+ // Add the last modification date to the response
+ ctx.header("Last-Modified", String.valueOf(now));
+ ctx.json(user);
}

public void delete(Context ctx) {
    Integer id = ctx.pathParamAsClass("id", Integer.class).get();
```

```
+ // Get the last known modification date of the user
+ LocalDateTime lastKnownModification =
+     ctx.headerAsClass("If-Unmodified-Since", LocalDateTime.class).getOrDefault(null);
+
+ // Check if the user has been modified since the last known modification date
+ if (lastKnownModification != null && !
+     usersCache.get(id).equals(lastKnownModification)) {
+     throw new PreconditionFailedResponse();
+ }
+
+ if (!users.containsKey(id)) {
+     throw new NotFoundResponse();
+ }

+ users.remove(id);

+ // Invalidate the cache for the user
+ usersCache.remove(id);
+
+ // Invalidate the cache for all users
+ usersCache.remove(REERVED_ID_TO_IDENTIFY_ALL_USERS);
+
+ ctx.status(HttpStatus.NO_CONTENT);
+ }
+ }
```

In this code snippets, we have updated the `UsersController.java` to:

1. Use the Map to cache the results of your application
2. Store results in the cache
3. Return the Last-Modified header
4. Validate the cache with the If-Modified-Since header
5. Validate the cache with the If-Unmodified-Since header

Test the caching system with curl

Now that you have implemented the validation model based on the Last-Modified header in your application, you can test the caching system.

To test the caching system, you can use the following steps:

1. Create a new user as client 1.

2. Get the user as client 1.
3. Get the user as client 2.
4. Update the user as client 1.
5. Update the user as client 2 using the old Last-Modified header.
6. Get all users as client 1.
7. Create a new user as client 2.
8. Get all users as client 1 using the old Last-Modified header.

Use the following commands to test the caching system with curl to simulate multiple clients and see if the cache is working as expected.

Create a new user as client 1

Create a new user as client 1

```
curl -i \  
  -X POST \  
  -H "Content-Type: application/json" \  
  -d '{"firstName":"John","lastName":"Doe","email":"john.doe@example.com","password":"secret"}' \  
  http://localhost:8080/users
```

The output should be similar to the following:

```
HTTP/1.1 201 Created  
Date: Fri, 06 Dec 2024 20:28:19 GMT  
Content-Type: application/json  
Last-Modified: 2024-12-06T21:28:19.140804844  
Content-Length: 95
```

```
{"id":  
1,"firstName":"John","lastName":"Doe","email":"john.doe@example.com","password":"secret"}
```

The Last-Modified header indicates the date and time at which the resource was last modified.

Get the user as client 1

Now that you have created a user as client 1, you can get the user as client 1, using the If-Modified-Since header:

Get the user as client 1

```
curl -i \  
  -X GET \  
  -H "If-Modified-Since: 2024-12-06T21:28:19.140804844"
```

```
-H "If-Modified-Since: 2024-12-06T21:28:19.140804844" \  
http://localhost:8080/users/1
```

The output should be similar to the following:

```
HTTP/1.1 304 Not Modified  
Date: Fri, 06 Dec 2024 20:50:17 GMT  
Content-Type: text/plain
```

The 304 Not Modified response indicates that the resource has not been modified since the time specified in the If-Modified-Since header.

Get the user as client 2

Now that you have gotten the user as client 1, you can get the user as client 2:

```
# Get the user as client 2  
curl -i \  
-X GET \  
http://localhost:8080/users/1
```

The output should be similar to the following:

```
HTTP/1.1 200 OK  
Date: Fri, 06 Dec 2024 20:50:43 GMT  
Content-Type: application/json  
Last-Modified: 2024-12-06T21:28:19.140804844  
Content-Length: 95
```

```
{"id":  
1,"firstName":"John","lastName":"Doe","email":"john.doe@example.com","password":"secret"}
```

As this is the first time client 2 has requested the user, the server has returned a 200 OK response with the user and the Last-Modified header corresponding to the date and time at which the resource was last modified.

Update the user as client 1

Now that you have gotten the user as client 2, you can update the user as client 1:

```
# Update the user with ID 1 as client 1
```

```
curl -i \  
  -X PUT \  
  -H "Content-Type: application/json" \  
  -H "If-Unmodified-Since: 2024-12-06T21:28:19.140804844" \  
  -d '{"firstName":"Jane","lastName":"Doe","email":"jane.doe@example.com","password":"secret"}' \  
  http://localhost:8080/users/1
```

The output should be similar to the following:

```
HTTP/1.1 200 OK  
Date: Fri, 06 Dec 2024 20:52:30 GMT  
Content-Type: application/json  
Last-Modified: 2024-12-06T21:52:30.542486768  
Content-Length: 95
```

```
{"id":  
1,"firstName":"Jane","lastName":"Doe","email":"jane.doe@example.com","password":"secret"}
```

The user has been updated successfully. The Last-Modified header indicates the date and time at which the resource was last modified, updated since the last time it was modified.

Update the user as client 2 using the old Last-Modified header

Now that you have updated the user as client 1, you can try to update the user as client 2:

```
# Update the user with ID 1 as client 2 using the old Last-Modified header
```

```
curl -i \  
  -X PUT \  
  -H "Content-Type: application/json" \  
  -H "If-Unmodified-Since: 2024-12-06T21:28:19.140804844" \  
  -d '{"firstName":"Jeanne","lastName":"Doe","email":"jeanne.doe@example.com","password":"secret"}' \  
  http://localhost:8080/users/1
```

The output should be similar to the following:

```
HTTP/1.1 412 Precondition Failed  
Date: Fri, 06 Dec 2024 20:55:04 GMT  
Content-Type: text/plain  
Content-Length: 19
```

Precondition Failed

The 412 Precondition Failed response indicates that the resource has been modified since the time specified in the If-Unmodified-Since header. The client cannot update the resource because the resource has been modified since the last time the client requested it. The client has to get the resource again to update it.

Get all users as client 1

Get all users as client 1:

```
# Get all users as client 1
```

```
curl -i \  
-X GET \  
http://localhost:8080/users
```

The output should be similar to the following:

```
HTTP/1.1 200 OK  
Date: Fri, 06 Dec 2024 21:05:23 GMT  
Content-Type: application/json  
Last-Modified: 2024-12-06T22:02:24.421558585  
Content-Length: 97
```

```
[[{"id":  
1,"firstName":"Jane","lastName":"Doe","email":"jane.doe@example.com","password":"secret"}]]
```

The list of users has been returned successfully. The Last-Modified header indicates the date and time at which the resource was last modified.

If you try to get all users again, you will see that the response is cached with a 304 Not Modified response:

```
# Get all users as client 1 using the Last-Modified header
```

```
curl -i \  
-X GET \  
-H "If-Modified-Since: 2024-12-06T22:02:24.421558585" \  
http://localhost:8080/users
```

Create a new user as client 2

Create a new user as client 2:

```
# Create a new user as client 2
curl -i \
  -X POST \
  -H "Content-Type: application/json" \
  -d '{"firstName":"Alice","lastName":"Doe","email":"alice.doe@example.com","password":"secret"}' \
  http://localhost:8080/users
```

The output should be similar to the previous one you have seen when you created a new user as client 1.

Get all users as client 1 using the old Last-Modified header

Get all users as client 1 using the old Last-Modified header:

```
# Get all users as client 1 using the old If-Modified-Since header
curl -i \
  -X GET \
  -H "If-Modified-Since: 2024-12-06T22:02:24.421558585" \
  http://localhost:8080/users
```

The output should be similar to the following:

```
HTTP/1.1 200 OK
Date: Fri, 06 Dec 2024 21:08:11 GMT
Content-Type: application/json
Last-Modified: 2024-12-06T22:08:11.793635324
Content-Length: 195
```

```
[[{"id":
1,"firstName":"Jane","lastName":"Doe","email":"jane.doe@example.com","password":"secret"},
{"id":
2,"firstName":"Alice","lastName":"Doe","email":"alice.doe@example.com","password":"secret"}]]
```

As the cache for the list of users has been invalidated (using the RESERVED_ID_TO_IDENTIFY_ALL_USERS magic ID), the server has returned a 200 OK response with the list of users and the Last-Modified header corresponding to the date and time at which the resource was last modified.

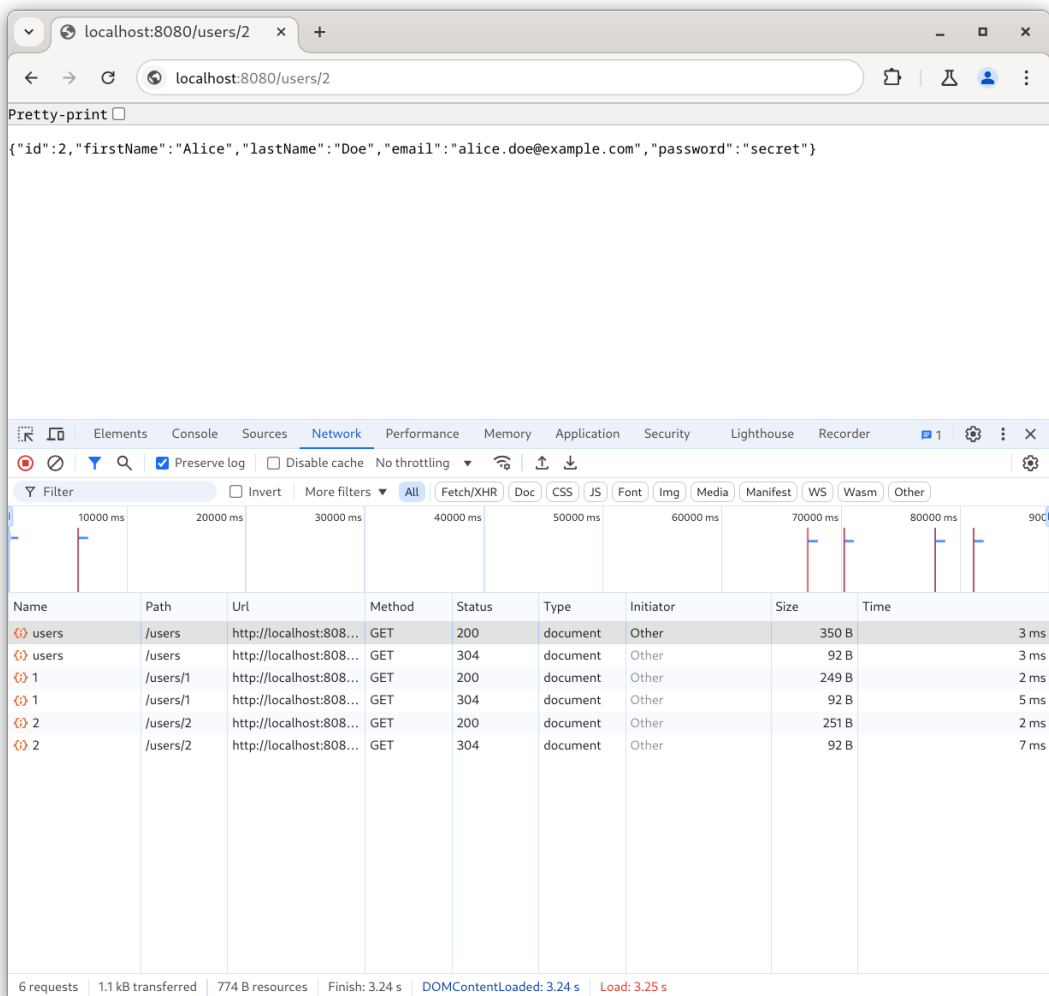
Test the caching system with a browser

You can also test the caching system with a browser to see if the cache is working as expected.

We recommend creating the users with curl as it is easier than with a browser. Then, you can use the browser to test the caching system of the GET requests.

To check if the cache is working as expected, open the developer tools of your browser and check the Network tab as seen in the chapter [HTTP and curl](#).

You can use the same steps as with curl to test the caching system with a browser.



The screenshot shows a browser window at localhost:8080/users/2. The developer tools Network tab is open, displaying a list of requests. The first request to /users/2 is a 200 OK response with a 350 B document. Subsequent requests for /users/1 and /users/2 are 304 Not Modified responses, indicating successful caching.

| Name | Path | Url | Method | Status | Type | Initiator | Size | Time |
|-------|----------|-------------------------|--------|--------|----------|-----------|-------|------|
| users | /users | http://localhost:808... | GET | 200 | document | Other | 350 B | 3 ms |
| users | /users | http://localhost:808... | GET | 304 | document | Other | 92 B | 3 ms |
| 1 | /users/1 | http://localhost:808... | GET | 200 | document | Other | 249 B | 2 ms |
| 1 | /users/1 | http://localhost:808... | GET | 304 | document | Other | 92 B | 5 ms |
| 2 | /users/2 | http://localhost:808... | GET | 200 | document | Other | 251 B | 2 ms |
| 2 | /users/2 | http://localhost:808... | GET | 304 | document | Other | 92 B | 7 ms |

6 requests | 1.1 kB transferred | 774 B resources | Finish: 3.24 s | DOMContentLoaded: 3.24 s | Load: 3.25 s

Go further

This is an optional section. Feel free to skip it if you do not have time.

- Are you able to add the expiration model to the validation model to use both models at the same time?
- Are you able to replace the Last-Modified validation model with the ETag validation model in your application?

Conclusion

What did you do and learn?

In this chapter, you have learned about caching mechanisms that are offered by HTTP.

You have discovered the expiration and validation caching models.

You have implemented the validation model based on the Last-Modified header in your application to improve the performance of the system.

You have tested the caching system with curl and a browser to simulate multiple clients and see if the cache is working as expected:

- If you run your application to get a user or all users with the If-Modified-Since header, the application will check the cache and response with a 304 Not Modified status code. Not even the database is queried, more performance!
- If you run your application to create a new user, the application will create a new record in the cache, invalidate the RESERVED_ID_TO_IDENTIFY_ALL_USERS magic ID cache, and response with a 201 Created status code including the new Last-Modified header
- If you run your application to update or delete a user with an old If-Unmodified-Since header, the application will check the cache and response with a 412 Precondition Failed status code - the client cannot update or delete the resource because the resource has been modified since the last time the client modified it

Test your knowledge

At this point, you should be able to answer the following questions:

- What is a cache?
- What is the expiration model?
- What is the validation model?
- What are the two types of conditional requests?

- What are the headers used to implement the validation model based on the Last-Modified header?
- What are the headers used to implement the validation model based on the ETag header?

Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

Note

Vous pouvez évidemment poser toutes vos questions et/ou vos propositions d'améliorations en français ou en anglais.

N'hésitez pas à nous dire si vous avez des difficultés à comprendre un concept ou si vous avez des difficultés à réaliser les éléments demandés dans le cours. Nous sommes là pour vous aider !

→ [GitHub Discussions](#)

You can use reactions to express your opinion on a comment!

What will you do next?

We are arriving at the end of the third part of the course. An evaluation will be done to check your understanding of all the content seen in this third part.

Additional resources

Resources are here to help you. They are not mandatory to read.

- *None yet*

Missing item in the list? Feel free to open a pull request to add it!

Sources

- Main illustration by [Richard Horne](#) on [Unsplash](#)