Bazar Bookstore 2 – Program Output

D. Samer Arandi

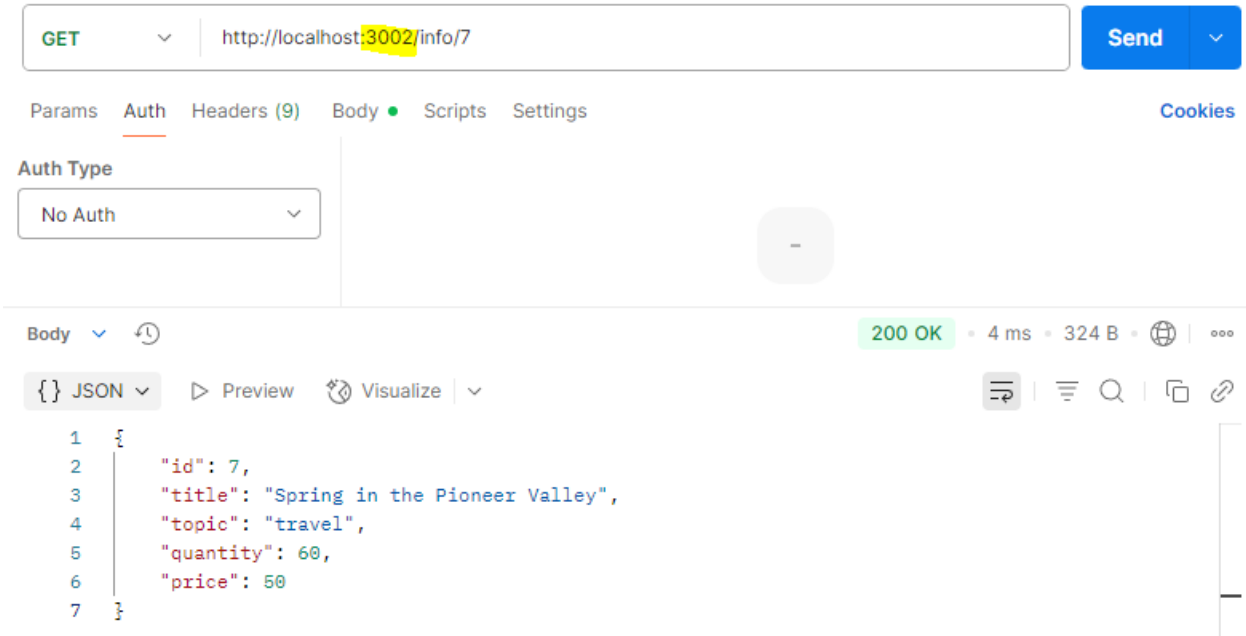
Rand Johari	**12027653**
Abeer Kharouf	**12028125**

🧠 Caching Strategy:

To improve performance and reduce the load on the catalog servers, we implemented an **in-memory caching system** within the front-end server. Below are the main components of the caching mechanism:

1. **Cache Hits on Repeated Requests:**
   When a client requests information about a specific book (e.g., **/info/:id**), the front-end first checks the cache:

a) If the data is not cached or expired, the front-end fetches it from the appropriate catalog replica and stores it in the cache (❌ cache miss).



In the logs below, we can observe that the front-end server started at 20:15:28, and shortly after that, a book info request was issued at 20:15:46 for book ID

Because this was the first request after the server started, the cache was still empty. As a result, the front-end issued a request to the appropriate catalog replica and retrieved the book details directly from the source.

```
2025-06-07 20:15:28 Front-end server running on port 3100
2025-06-07 20:15:46 Book details:
2025-06-07 20:15:46 ID: 7
2025-06-07 20:15:46 Topic: travel
2025-06-07 20:15:46 Title: Spring in the Pioneer Valley
2025-06-07 20:15:46 Quantity: 60
2025-06-07 20:15:46 Price: 50
```

b) If the data is already cached and still valid, it is returned immediately without contacting the catalog server (✅ cache hit).

Just a few seconds later, at 20:15:50, the same book ID is requested again, this time, the response is served directly from the cache, avoiding a backend call.

```
2025-06-07 20:15:28 Front-end server running on port 3100
2025-06-07 20:15:46 Book details:
2025-06-07 20:15:46 ID: 7
2025-06-07 20:15:46 Topic: travel
2025-06-07 20:15:46 Title: Spring in the Pioneer Valley
2025-06-07 20:15:46 Quantity: 60
2025-06-07 20:15:46 Price: 50
2025-06-07 20:15:50 Serving from cache: info:7
```

c) Each cached item has a Time-To-Live **(TTL)** of **1 minute**. After that, it is considered expired and automatically removed upon access.

At 20:17:18, we see the same book being requested again, this time there's no "**Serving from cache**" log, which means the item had expired from cache due to TTL. The frontend automatically fetched it again from the catalog service (cache miss due to expiration).

```
2025-06-07 20:15:50 Serving from cache: info:7
2025-06-07 20:17:18 Book details:
2025-06-07 20:17:18 ID: 7
2025-06-07 20:17:18 Topic: travel
2025-06-07 20:17:18 Title: Spring in the Pioneer Valley
2025-06-07 20:17:18 Quantity: 60
2025-06-07 20:17:18 Price: 50
```

2. **Cache Hits on Repeated Requests:**

   To simulate realistic constraints, we set the cache capacity to **only 4 items**.

   In the logs below info:7 (book ID 7) was removed from the cache **to make space** for a new book request (in this case, book ID 3).

```
2025-06-07 20:41:21 Front-end server running on port 3100
2025-06-07 20:41:28 Book details:
2025-06-07 20:41:28 ID: 7
2025-06-07 20:41:28 Topic: travel
2025-06-07 20:41:28 Title: Spring in the Pioneer Valley
2025-06-07 20:41:28 Quantity: 60
2025-06-07 20:41:28 Price: 50
2025-06-07 20:41:30 Book details:
2025-06-07 20:41:30 ID: 6
2025-06-07 20:41:30 Topic: academic
2025-06-07 20:41:30 Title: Why theory classes are so hard
2025-06-07 20:41:30 Quantity: 72
2025-06-07 20:41:30 Price: 50
2025-06-07 20:41:31 Book details:
2025-06-07 20:41:31 ID: 5
2025-06-07 20:41:31 Topic: project management
2025-06-07 20:41:31 Title: How to finish Project 3 on time
2025-06-07 20:41:31 Quantity: 29
2025-06-07 20:41:31 Price: 50
2025-06-07 20:41:33 Book details:
2025-06-07 20:41:33 ID: 4
2025-06-07 20:41:33 Topic: undergraduate school
2025-06-07 20:41:33 Title: Cooking for the Impatient Undergrad
2025-06-07 20:41:33 Quantity: 56
2025-06-07 20:41:33 Price: 50
2025-06-07 20:41:35 Cache full. Removing least recently used: info:7
2025-06-07 20:41:35 Book details:
2025-06-07 20:41:35 ID: 3
2025-06-07 20:41:35 Topic: undergraduate school
2025-06-07 20:41:35 Title: Xen and the Art of Surviving Undergraduate School
2025-06-07 20:41:35 Quantity: 71
2025-06-07 20:41:35 Price: 50
```

To further confirm that **book 7 (info:7)** was actually removed from the cache due to LRU:

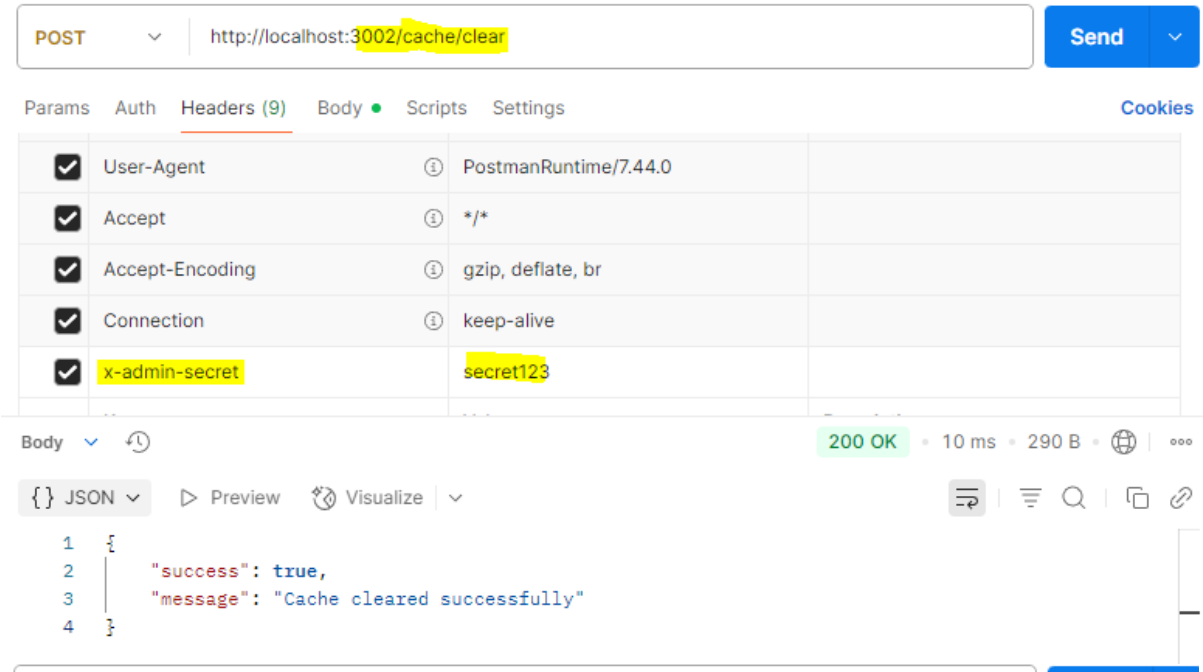
We can observe that when we re-request book ID 7,the front-end does **not** serve it from the cache (no Serving from cache message appears). Instead, it has to **fetch it again**, and due to the cache already being full, it **evicts another item**:

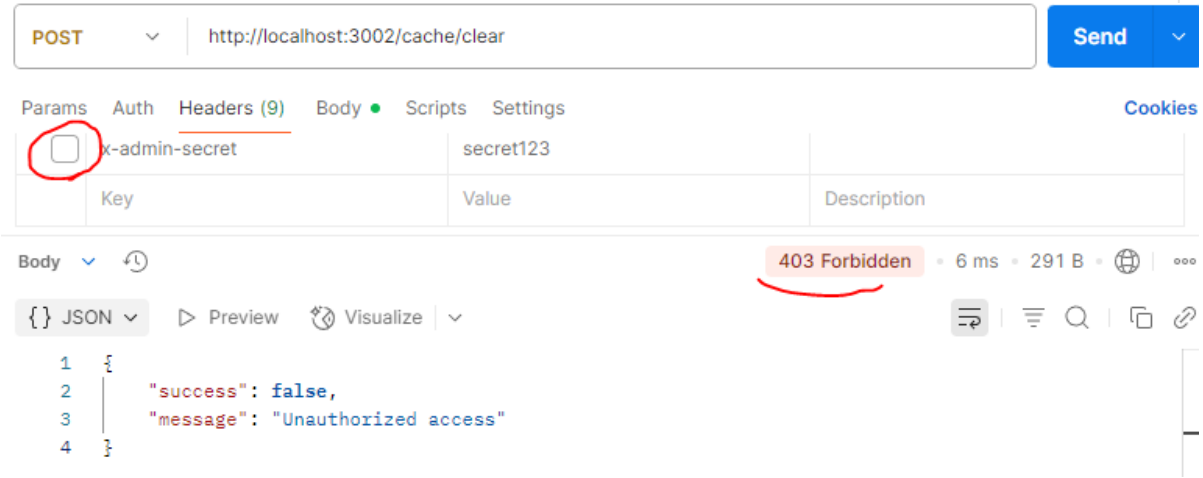
```
2025-06-07 20:41:28 ID: 7
2025-06-07 20:41:28 Topic: travel
2025-06-07 20:41:28 Title: Spring in the Pioneer Valley
2025-06-07 20:41:28 Quantity: 60
2025-06-07 20:41:28 Price: 50
2025-06-07 20:41:30 Book details:
2025-06-07 20:41:30 ID: 6
2025-06-07 20:41:30 Topic: academic
2025-06-07 20:41:30 Title: Why theory classes are so hard
2025-06-07 20:41:30 Quantity: 72
2025-06-07 20:41:30 Price: 50
2025-06-07 20:41:31 Book details:
2025-06-07 20:41:31 ID: 5
2025-06-07 20:41:31 Topic: project management
2025-06-07 20:41:31 Title: How to finish Project 3 on time
2025-06-07 20:41:31 Quantity: 29
2025-06-07 20:41:31 Price: 50
2025-06-07 20:41:33 Book details:
2025-06-07 20:41:33 ID: 4
2025-06-07 20:41:33 Topic: undergraduate school
2025-06-07 20:41:33 Title: Cooking for the Impatient Undergrad
2025-06-07 20:41:33 Quantity: 56
2025-06-07 20:41:33 Price: 50
2025-06-07 20:41:35 Cache full. Removing least recently used: info:7
2025-06-07 20:41:35 Book details:
2025-06-07 20:41:35 ID: 3
2025-06-07 20:41:35 Topic: undergraduate school
2025-06-07 20:41:35 Title: Xen and the Art of Surviving Undergraduate School
2025-06-07 20:41:35 Quantity: 71
2025-06-07 20:41:35 Price: 50
2025-06-07 20:48:16 Cache full. Removing least recently used: info:6
2025-06-07 20:48:16 Book details:
2025-06-07 20:48:16 ID: 7
2025-06-07 20:48:16 Topic: travel
2025-06-07 20:48:16 Title: Spring in the Pioneer Valley
2025-06-07 20:48:16 Quantity: 60
2025-06-07 20:48:16 Price: 50
```

3. 🛠 Admin-Controlled Cache Clearing:

We also implemented an **admin endpoint** that allows cache to be fully cleared at any time:
http://localhost:3002/cache/clear
This endpoint requires a secret admin token. It is particularly useful for evaluation purposes. When triggered, all items are removed from the cache.

🔐 **In the request headers, we included:**
x-admin-secret: secret123





After sending the request, we received the response:"**Cache cleared successfully**". Immediately afterward, we issued info requests for the same books that had previously been cached. In the terminal logs, we noticed that the following message did **not** appear: **Serving from cache**. This confirms that the data was **fetched from the original catalog replica** rather than the cache — proving that the cache was successfully cleared and the frontend fell back to querying the source servers.

```
2025-06-07 20:57:10 Cache manually cleared by admin
2025-06-07 20:57:40 Book details:
2025-06-07 20:57:40 ID: 7
2025-06-07 20:57:40 Topic: travel
2025-06-07 20:57:40 Title: Spring in the Pioneer Valley
2025-06-07 20:57:40 Quantity: 60
2025-06-07 20:57:40 Price: 50
2025-06-07 20:57:46 Book details:
2025-06-07 20:57:46 ID: 5
2025-06-07 20:57:46 Topic: project management
2025-06-07 20:57:46 Title: How to finish Project 3 on time
2025-06-07 20:57:46 Quantity: 29
2025-06-07 20:57:46 Price: 50
```

4. Cache Consistency and Behavior:

a) Cache Usage for Search and Info Queries

The cache is **only active** for **read-only endpoints**:

- GET /info/:id — fetch details of a book.
- GET /search/:topic — search books by topic.

We verified that caching is working correctly for both endpoints through repeated experiments:

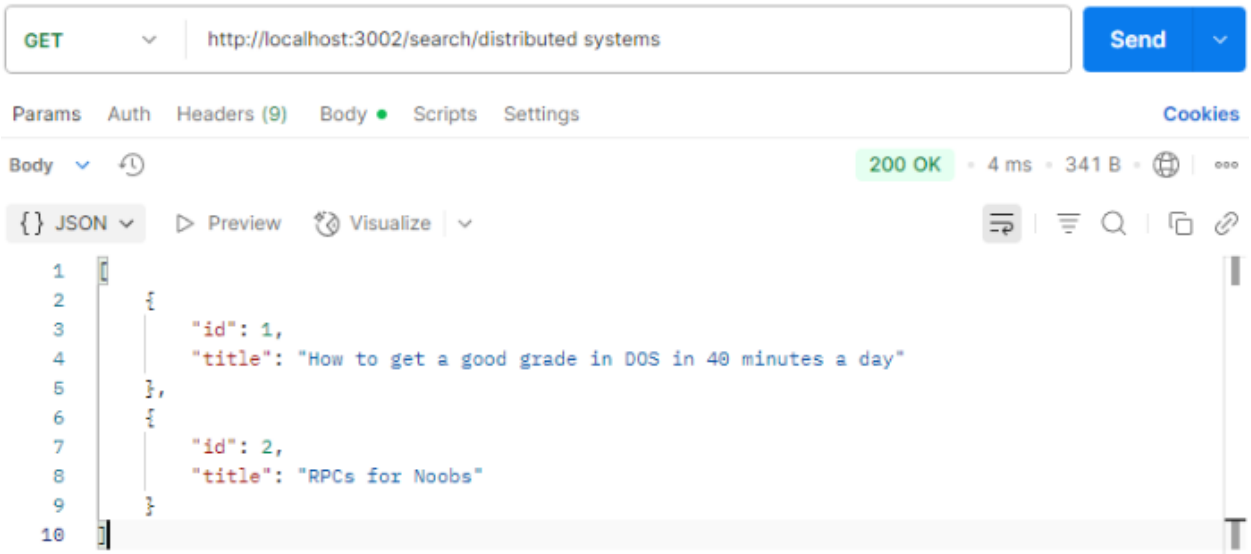
Example: GET /info/7

- First request at 21:32:39 fetched from the catalog (miss).

- Second request at 21:32:44 served from cache (**Serving from cache**: info:7).

Example: GET /search/distributed systems

- First request at 21:33:13 returned book list for the topic.
- Repeating the request at 21:33:18 gave: **Serving from cache**




```
2025-06-07 21:32:39 Book details:
2025-06-07 21:32:39 ID: 7
2025-06-07 21:32:39 Topic: travel
2025-06-07 21:32:39 Title: Spring in the Pioneer Valley
2025-06-07 21:32:39 Quantity: 60
2025-06-07 21:32:39 Price: 50
2025-06-07 21:32:44 Serving from cache: info:7
2025-06-07 21:33:13 Search results for topic 'distributed systems':
2025-06-07 21:33:13 - [1] How to get a good grade in DOS in 40 minutes a day
2025-06-07 21:33:13 - [2] RPCs for Noobs
2025-06-07 21:33:18 Serving from cache: search:distributed systems
```

b) Cache Consistency During Write Operations

At 21:34:23, we issued a **purchase request** for book ID **7**. Since this book was in the cache, our system correctly triggered a **cache invalidation**

POST http://localhost:3002/purchase/7 Send

Params Auth Headers (9) Body Scripts Settings Cookies

Body 200 OK 590 ms 301 B

{} JSON Preview Visualize

```
{
    "success": true,
    "message": "Book purchased successfully by order1"
}
```

```
2025-06-07 21:34:23 Invalidate request body: { id: 7 }
2025-06-07 21:34:23 🧹Cache invalidated for book ID: 7
2025-06-07 21:34:23 ✅Bought book with ID: 7
```

Then, at 21:35:37, an **admin update** was issued for book ID **2**. Again, we observe the system pushing an invalidation:

PUT http://localhost:3002/admin/update/2 Send

Params Auth Headers (9) Body Scripts Settings Cookies

raw JSON Beautify

```
{
    "price": 50,
    "quantity":7}
```

Body 200 OK 32 ms 337 B

{} JSON Preview Visualize

```
{
    "result": {
        "message": "Update complete and cache cleared",
        "updated": {
            "id": 2,
            "price": 50,
            "quantity": 7
        }
    }
}
```

```
2025-06-07 21:35:37 Invalidate request body: { id: 2 }
2025-06-07 21:35:37 🛠️Admin updated book ID 2 via http://catalog2:3000
```

To verify that invalidation was successful, we re-issued GET /info/7 and GET /info/2. Both calls were not served from cache — we confirmed this because there was **no "Serving from cache" message**. Instead, the catalog was contacted again, and the updated data (e.g., quantity dropped from 60 → 59) was retrieved.

```
2025-06-07 21:32:39 Book details:
2025-06-07 21:32:39 ID: 7
2025-06-07 21:32:39 Topic: travel
2025-06-07 21:32:39 Title: Spring in the Pioneer Valley
2025-06-07 21:32:39 Quantity: 60
2025-06-07 21:32:39 Price: 50
2025-06-07 21:32:44 Serving from cache: info:7
2025-06-07 21:33:13 Search results for topic 'distributed systems':
2025-06-07 21:33:13 - [1] How to get a good grade in DOS in 40 minutes a day
2025-06-07 21:33:13 - [2] RPCs for Noobs
2025-06-07 21:33:18 Serving from cache: search:distributed systems
2025-06-07 21:34:23 Invalidate request body: { id: 7 }
2025-06-07 21:34:23 Cache invalidated for book ID: 7
2025-06-07 21:34:23 Bought book with ID: 7
2025-06-07 21:35:37 Invalidate request body: { id: 2 }
2025-06-07 21:35:37 Admin updated book ID 2 via http://catalog2:3000
2025-06-07 21:58:49 Book details:
2025-06-07 21:58:49 ID: 7
2025-06-07 21:58:49 Topic: travel
2025-06-07 21:58:49 Title: Spring in the Pioneer Valley
2025-06-07 21:58:49 Quantity: 59
2025-06-07 21:58:49 Price: 50
2025-06-07 21:58:57 Cache full. Removing least recently used: info:5
2025-06-07 21:58:57 Book details:
2025-06-07 21:58:57 ID: 2
2025-06-07 21:58:57 Topic: distributed systems
2025-06-07 21:58:57 Title: RPCs for Noobs
2025-06-07 21:58:57 Quantity: 7
2025-06-07 21:58:57 Price: 50
```

🔄 Load Balancing in Replicated Order & Catalog Servers:

a) Order replica:

At 22:09:23, book purchase request:

- From the logs of **Order1**, we observe:

```
2025-06-07 22:09:23 📥Received OK from replica 2 for book 6
2025-06-07 22:09:23 [order1] 🔍Contacting catalog: http://catalog2:3000/info/6
2025-06-07 22:09:23 ✅[order1] Bought book: Why theory classes are so hard
```

  This request was routed to **catalog2** and successfully completed.

- From the logs of **Order2**, at the **same timestamp:**

```
2025-06-07 22:09:23 ✅Granted access to replica 1 for book 6
```

Then, at 22:09:28, another request was handled:

- **Order2** now handled book 5**:**

```
2025-06-07 22:09:28 📥Received OK from replica 1 for book 5
2025-06-07 22:09:28 [order2] 🔍Contacting catalog: http://catalog1:3000/info/5
2025-06-07 22:09:28 ✅[order2] Bought book: How to finish Project 3 on time
```

This request was routed to **catalog1** and successfully completed.

- **Order1**, on the other hand:

```
2025-06-07 22:09:28 ✅Granted access to replica 2 for book 5
```

a) Catalog replica:

At 22:27:18, book purchase request:

- **Catalog2** initiated an update for book **5**:

```
2025-06-07 22:27:18 ⏳Wants to update book 5 — Clock: 16
2025-06-07 22:27:18 📥Received OK from replica 1 for book 5
2025-06-07 22:27:18 🔒All OKs received. Safe to update book 5
2025-06-07 22:27:18 🧹Sent invalidate for book 5
2025-06-07 22:27:18 ✅Update complete for book 5
```

- At the exact same time, **Catalog1** received a for the same book:

```
2025-06-07 22:27:18 ✅Granted access to replica 2 for book 5
```

Then, at 22:27:29, the behavior switched roles:

- **Catalog1** initiated an update for book **7**:

```
2025-06-07 22:27:29 ⏳Wants to update book 7 — Clock: 18
2025-06-07 22:27:29 📥Received OK from replica 2 for book 7
2025-06-07 22:27:29 🔒All OKs received. Safe to update book 7
2025-06-07 22:27:29 🧹Sent invalidate for book 7
2025-06-07 22:27:29 ✅Update complete for book 7
```

- Simultaneously, **Catalog2** received the :

```
2025-06-07 22:27:29 ✅Granted access to replica 1 for book 7
```

This confirms that requests are distributed across different catalog & order replicas, and results were returned successfully.

📝 Catalog Replica Synchronization (Sync):

CSV catalog 1 and catalog 2 before admin update request:

```
books.csv M ×
backend > catalog > catalog2 > books.csv > data
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,5,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,70,50
7,Spring in the Pioneer Valley,travel,46,50
```

```
books.csv M ×
backend > catalog > catalog1 > books.csv > data
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,5,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,70,50
7,Spring in the Pioneer Valley,travel,46,50
```

a) At 22:43:50, an **admin update** was issued to book **ID 2** using **catalog2** ("RPCs for Noobs") via catalog2. The update reduced the book's quantity from **5 → 41**. This can be confirmed in the logs:

- **Frontend log** shows:

```
2025-06-07 22:43:50 Invalidate request body: { id: 2 }
2025-06-07 22:43:50 🛠️ admin updated book ID 2 via http://catalog2:3000
```

- **Catalog2** performed the update**:**

```
2025-06-07 22:43:50 ⏳ Wants to update book 2 — Clock: 20
2025-06-07 22:43:50 📥 Received OK from replica 1 for book 2
2025-06-07 22:43:50 🔒 All OKs received. Safe to update book 2
2025-06-07 22:43:50 🧹 Sent invalidate for book 2
2025-06-07 22:43:50 ✅ Update complete for book 2
```

- As part of the consistency mechanism, **Catalog1** was synchronized immediately**:**

```
2025-06-07 22:43:50 ✅Granted access to replica 2 for book 2
2025-06-07 22:43:50 🔄Received sync request for book ID 2 from other replica
2025-06-07 22:43:50 📝Sync data: { quantity: 41, price: 50, clock: 21 }
2025-06-07 22:43:50 ⏱Logical clock updated to 21 from incoming sync
```

After synchronization:

books.csv M ×

backend > catalog > catalog2 > books.csv > data

```csv
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,41,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,70,50
7,Spring in the Pioneer Valley,travel,46,50
```

books.csv M ×

backend > catalog > catalog1 > books.csv > data

```csv
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,41,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,70,50
7,Spring in the Pioneer Valley,travel,46,50
```

b) At 23:05:11, a **purchase request** was made for **book ID 6** through the frontend. The purchase decreased the book's quantity from **70 → 69.**

- **Frontend log** shows:

```
2025-06-07 23:05:11 Invalidate request body: { id: 6 }
2025-06-07 23:05:11 ✅Bought book with ID: 6
```

- **Catalog1** performed the update**:**

```
2025-06-07 23:05:11 📦Quantity: 70
2025-06-07 23:05:11 ⏳Wants to update book 6 — Clock: 22
2025-06-07 23:05:11 📥Received OK from replica 2 for book 6
2025-06-07 23:05:11 🔒All OKs received. Safe to update book 6
2025-06-07 23:05:11 🧹Sent invalidate for book 6
2025-06-07 23:05:11 ✅Update complete for book 6
```

- As part of the consistency mechanism, **Catalog2** was synchronized immediately:

```
2025-06-07 23:05:11 ✅ Granted access to replica 1 for book 6
2025-06-07 23:05:11 🔄 Received sync request for book ID 6 from other replica
2025-06-07 23:05:11 📝 Sync data: { quantity: 69, price: 50, clock: 23 }
2025-06-07 23:05:11 ⏱ Logical clock updated to 23 from incoming sync
```

After synchronization:

books.csv M

backend > catalog > catalog2 > books.csv > data

```csv
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,41,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,69,50
7,Spring in the Pioneer Valley,travel,46,50
```

books.csv M

backend > catalog > catalog1 > books.csv > data

```csv
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,21,60
2,RPCs for Noobs,distributed systems,41,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,71,50
4,Cooking for the Impatient Undergrad,undergraduate school,56,50
5,How to finish Project 3 on time,project management,52,50
6,Why theory classes are so hard,academic,69,50
7,Spring in the Pioneer Valley,travel,46,50
```