Large-Scale and Multi-Structured Databases

Distribooked

A Distributed Library Management System

GABRIELE GIUDICI

ALESSANDRO CIARNIELLO

GABRIELE BILLI CIANI

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Application Highlights

Distributed Library Management System

Book Discovery

- Browse comprehensive book catalogue
- Search by title, author, most popular, and by category
- Find book details with (nearby) libraries with copies

User Actions

- Reserve/Borrow up to 5 books at a time
- Save books to favourites
- Track reserved, borrowed, and read books

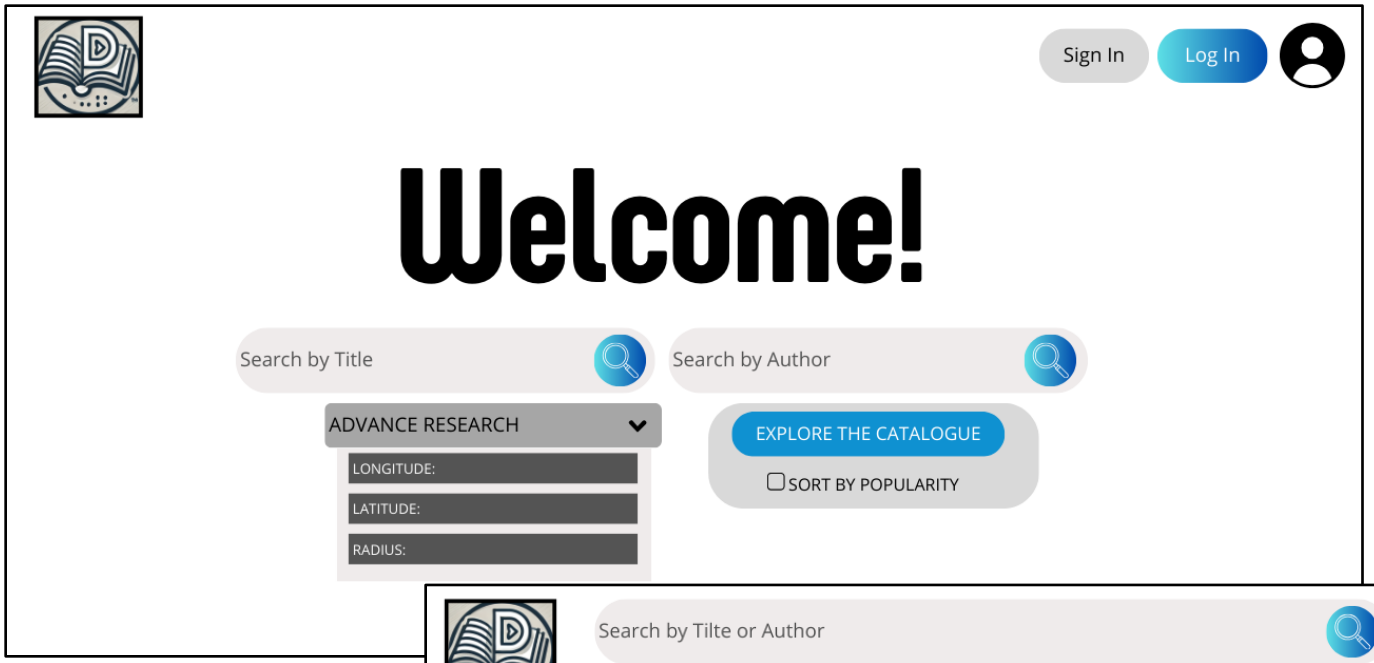
Book Availability & Access

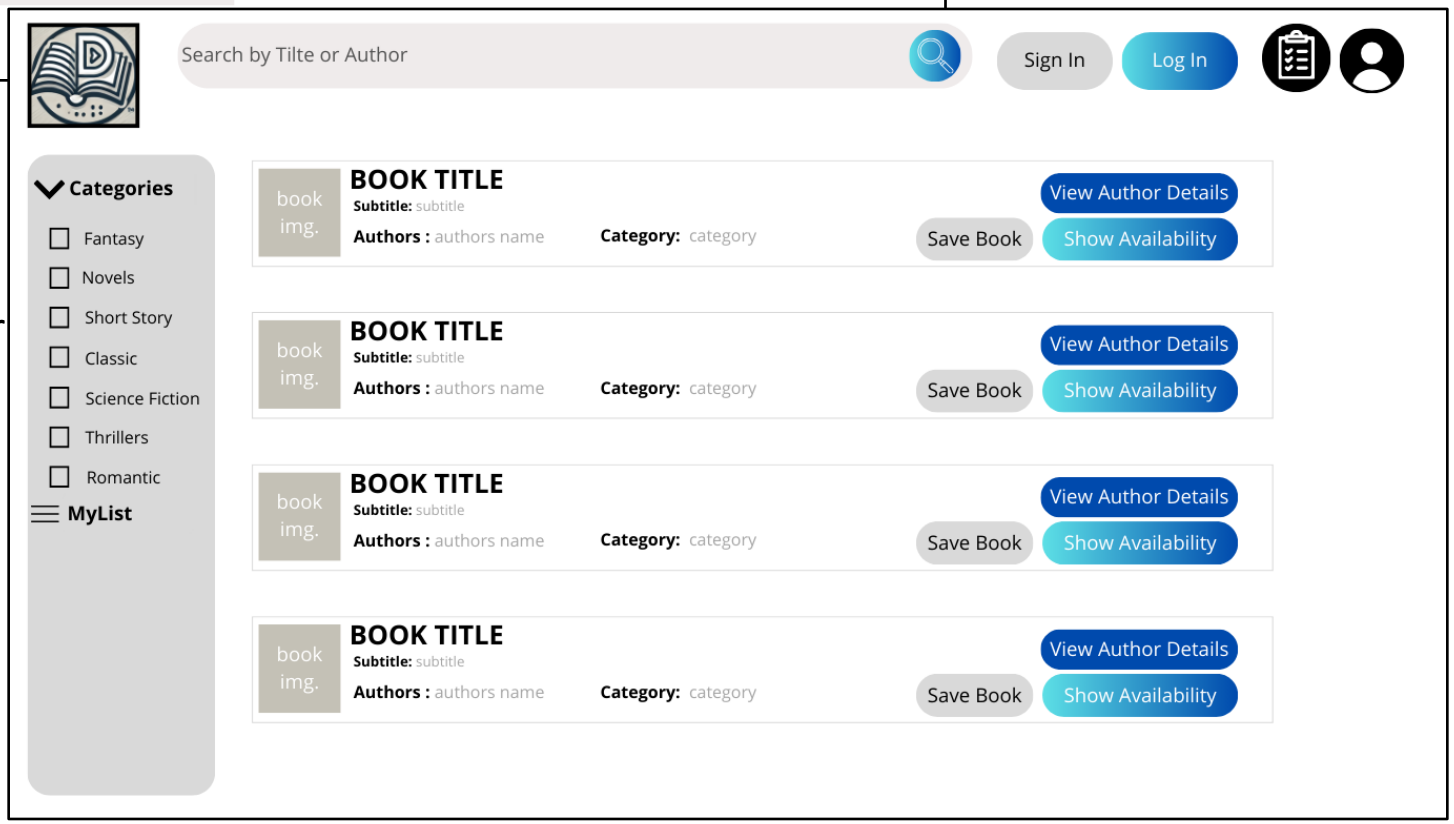
- View updated number of available copies of books in libraries
- Reservation system with 3-day pickup and 30-day return period

Administrative Capabilities

- Search and manage reservations (by library or user)
- Add new books and manage inventory
- Track overdue books and returns

Actors and Mockups





- Unregistered user
- Registered user
- Administrator

Unregistered User

Book Details & Availability

book img.

BOOK TITLE
Subtitle: subtitle

Authors : authors name

Date of Publishing: date

Language: language

Category: category

Publisher: publisher

isbn: isbn code

AVAILABILITY

LIBRARY NAME
Library Address: address
Library Distance: distance
Number of copies | View Library Details | Reserve

LIBRARY NAME
Library Address: address
Library Distance: distance
Number of copies | View Library Details | Reserve

LIBRARY NAME
Library Address: address
Library Distance: distance
Number of copies | View Library Details | Reserve

Author Details

Author img.

Name: author name

About the Author:

short biography

Year of Death: date

BOOKS:

Page 1/2

book img.
BOOK TITLE
Subtitle: subtitle
Category: category
Other Authors: authors name

book img.
BOOK TITLE
Subtitle: subtitle
Category: category
Other Authors: authors name

book img.
BOOK TITLE
Subtitle: subtitle
Category: category
Other Authors: authors name

Search Authors

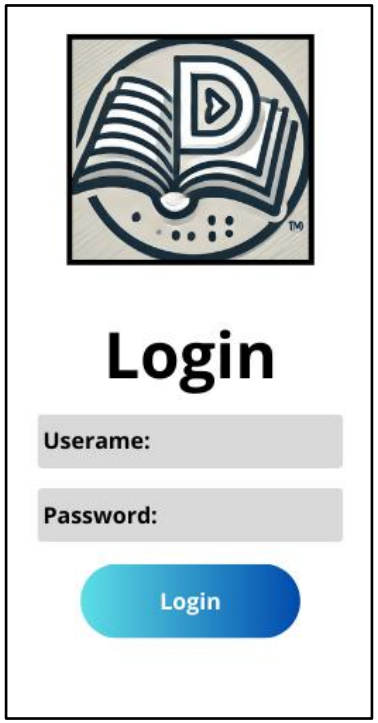
author img. **AUTHOR NAME** View Author Details

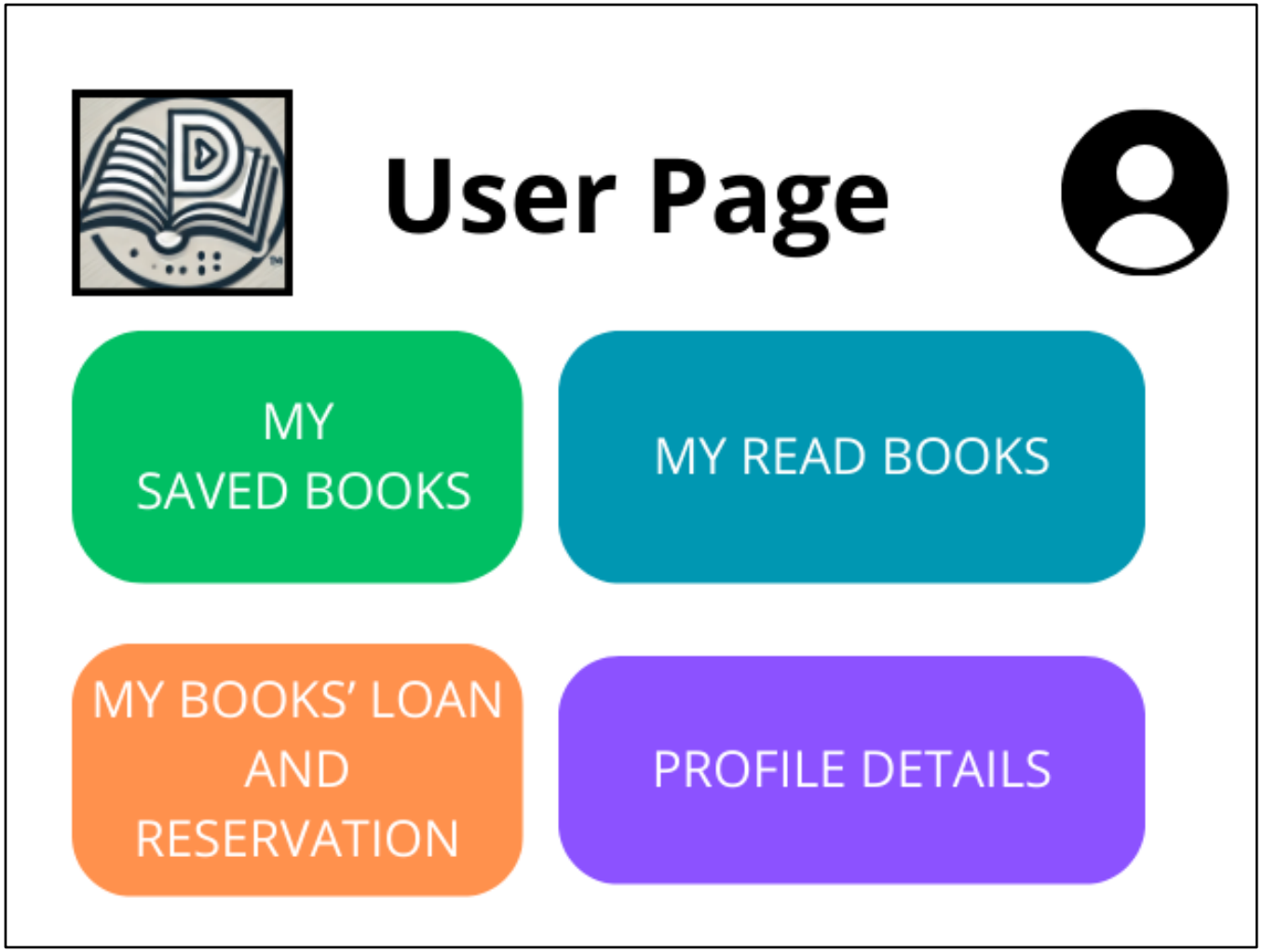
author img. **AUTHOR NAME** View Author Details

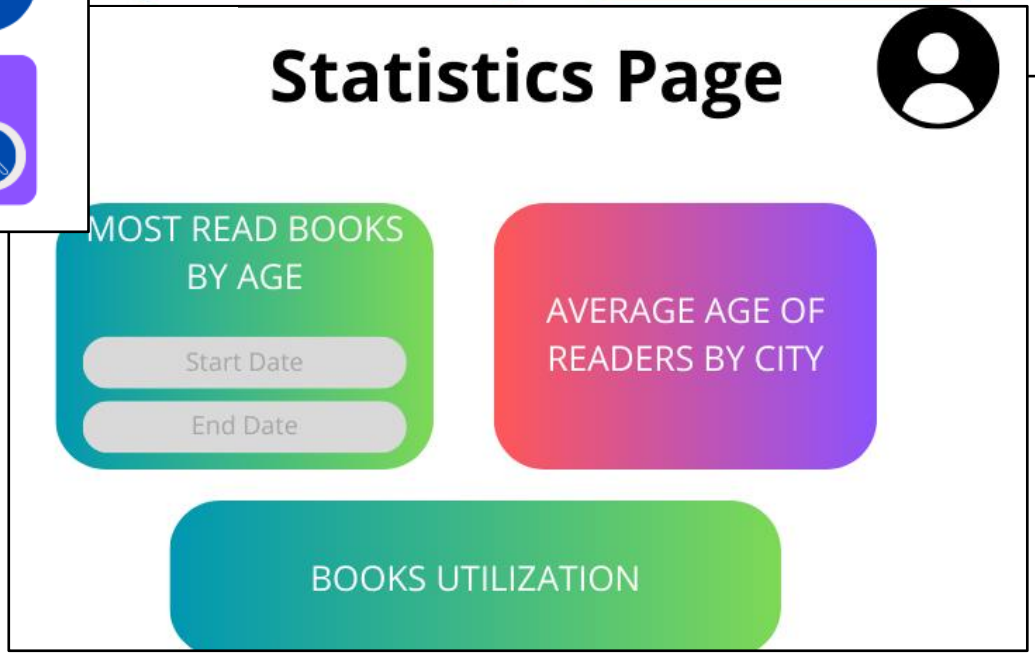
author img. **AUTHOR NAME** View Author Details

author img. **AUTHOR NAME** View Author Details

Registered User







Administrator

Dataset Description (i)

To build our database we started from **3** different **real datasets:**

Source	Description	Volume
Amazon Reviews Dataset collected in 2023 by McAuley Lab	A large-scale data set of item metadata for various products, including books.	Metadata of 4.4 million books; .jsonl.gz file (5.82 GB)
Project Gutenberg Metadata Dataset	A data set containing metadata for all books downloadable from gutenberg.org	.csv file (19.3 MB)
Anagrafe delle Biblioteche Italiane, Open Data	Open data registry of Italian libraries, including addresses, coordinates, and other details.	.csv file (3.4 MB)

- **Users data**: artificially generated using Python's Faker library → personal information about users (510 KB)

➢ **<u>Variety in data sources</u>**

Books Data Processing

- **Filter**ed Amazon dataset to retain only book-related records
- **Merge**d Amazon and Project Gutenberg metadata based on titles
- **Filled missing attributes** using data from both sources

➢ Final book dataset formatted in JSON (88MB)

Library Data Processing

- **Clean**ed dataset by removing irrelevant attributes and excessive null values
- Restricted to Pisa library branches

➢ Converted to JSON format (13KB)

User Data Processing

- Generated synthetic users with realistic names, addresses, emails, and phone numbers
- **Passwords hashed** using bcrypt → **non-functional requirement**

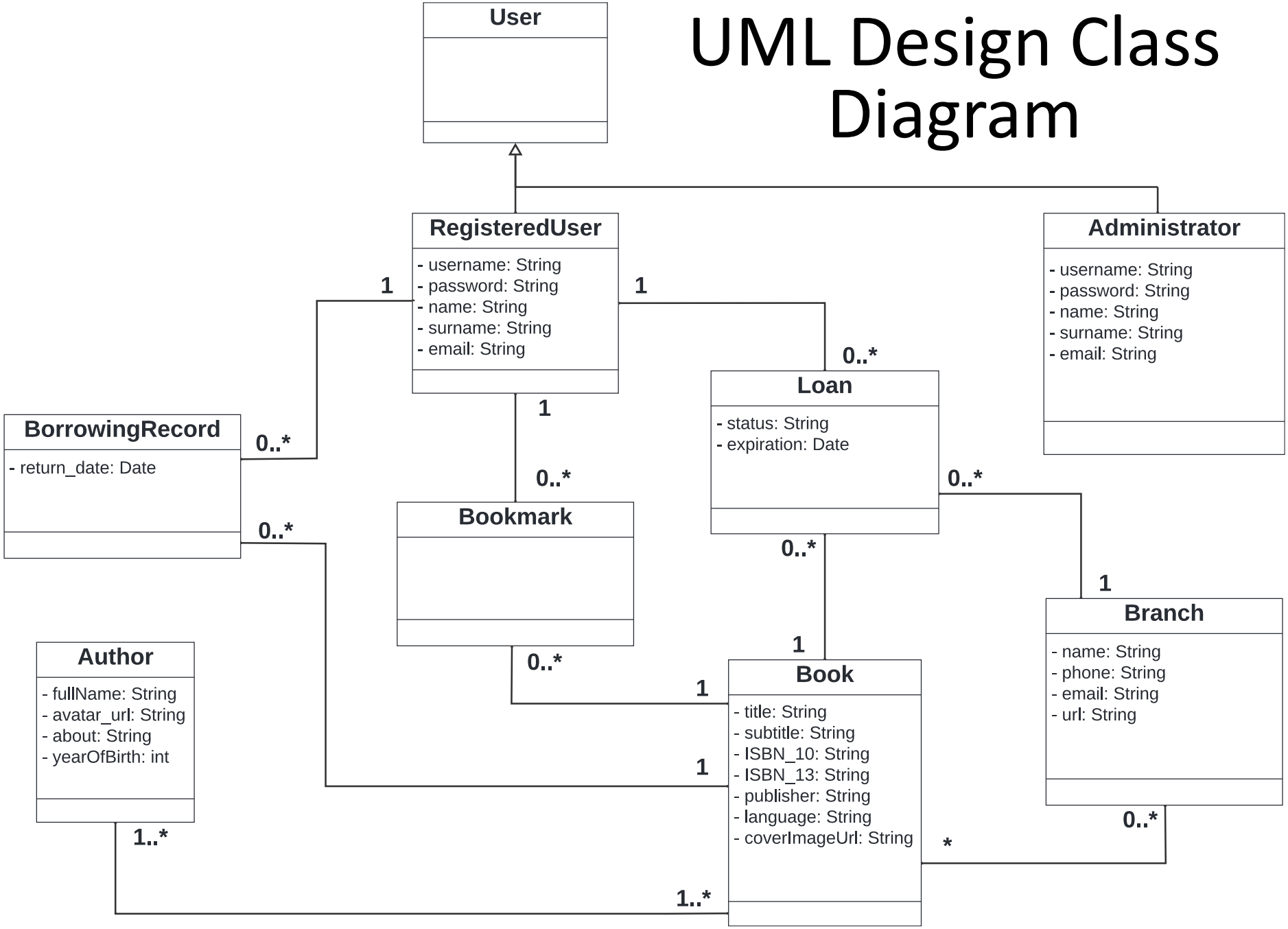
Book Distribution Across Libraries

- Books assigned to the 26 library branches using a round-robin method.
- Additional randomisation introduced for multiple copies across branches.

Library Log Simulation

- **Historical data generated** for 2022-2024.
- Each user assigned a random number of borrowed books per year (max 80 per year), and a random number of saved books (max 50)

UML Design Class Diagram



Application non-functional requirements

- Access to authorised functionalities must be protected by a **secure authentication mechanism**
- User **credentials** must be securely **stored using encryption**
- The system must be developed using **object-oriented programming** languages
- The system must prioritise **high availability** without compromising **data integrity** for critical operations, such as book reservations and loan management
- The system must **minimise data loss** to preserve records of past readings and ensure consistency in book availabilities and loan management
- The system must support a **throughput of at least 50 operations** per second, considering a **typical workload distribution** that includes authentication, catalogue browsing, and book management activities

Document DB Design (i)

MongoDB

Authors

```
{
    "_id": ObjectId("679cb31ab477993c5cdc08da"),
    "fullName": "Thomas Mann",
    "yearOfBirth": "1875",
    "yearOfDeath": "1955",
    "avatarUrl":
        "https://m.media-amazon.com/images/I/91bDRRiA2fL._SY600_
        .jpg",
    "about": "Paul Thomas Mann was a German novelist...",
    "books": [
        {
            "_id": ObjectId("679cb33db477993c5cdcdf5c"),
            "title": "Gladius Dei; Schwere Stunde",
            "authors": [
                {
                    "id": ObjectId("679cb31ab477993c5cdc08da"),
                    "fullName": "Thomas Mann"
                }
            ]
        },
        ...
        {
            "_id": ObjectId("679cb34cb477993c5cdd5e5c"),
            "title": "Royal Highness",
            "subtitle": "Paperback - January 27, 2019",
            "categories": ["Books", "Literature & Fiction",
               "Literary"],
            "coverImageUrl":
               "https://m.media-amazon.com/images/I/51NSftca6AL.
               _SX348_BO1,204,203,200_.jpg",
            "authors": [
                {
                    "id": ObjectId("679cb31ab477993c5cdc08da"),
                    "fullName": "Thomas Mann"
                },
                {
                    "id": ObjectId("679cb31ab477993c5cdc3af0"),
                    "fullName": "Curtis A. Cecil [Translator]"
                },
                ...
            ]
        }
    ]
}
```

Books

```
{
    "_id": ObjectId("679cb33db477993c5cdcdf9c"),
    "title": "Der Tod in Venedig",
    "publicationDate": "2004-04-01",
    "language": "German",
    "authors": [
        {
            "_id": ObjectId("679cb31ab477993c5cdc08da"),
            "fullName": "Thomas Mann"
        }
    ],
    "readingsCount": 0,
    "branches": [
        {
            "_id": ObjectId("679cb364d125ba32463b9746"),
            "libraryName": "Biblioteca dell'Istituto Domus
               Mazziniana",
            "location": { "type": "Point", "coordinates":
               [10.3980345, 43.7114097] },
            "address": {
                "street": "Via Giuseppe Mazzini 71",
                "city": "Pisa",
                "province": "Pisa",
                "postalCode": "56125",
                "country": "Italia"
            },
            "numberOfCopies": 5
        },
        ...
    ]
}
```

Branches


```
{
  "_id": ObjectId("679cb364d125ba32463b9744"),
  "isilCode": "IT-PI0112",
  "name": "Biblioteca Universitaria",
  "address": {
    "street": "Piazza San Matteo in Soarta 2",
    "city": "Pisa",
    "province": "Pisa",
    "postalCode": "56126",
    "country": "Italia"
  },
  "location": { "type": "Point", "coordinates": [ 10.4074101,
     43.7146166 ] },
  "phone": "+39 050573749",
  "email": "bu-pi@cultura.gov.it",
  "url": "https://www.bibliotecauniversitaria.pi.it/it/index.html"
}
```


Users


```
{
  "_id": ObjectId("679cb391d5e81efe4645569e"),
  "username": "victo13",
  "name": "Victoria",
  "surname": "Disdero",
  "dateOfBirth": "1983-10-13",
  "password": "82746999021240O7b5fd493834602ec",
  "userType": "USER",
  "email": "disderovictoria@libero.it",
  "address": {
    "street": "Stretto Livio, 765 Piano 3",
    "city": "Pisa",
    "postalCode": "56126",
    "province": "Pisa",
    "country": "Italy"
  },
  "location": {
    "type": "Point",
    "coordinates": [10.410596867140791, 43.70232171686015]
  },
  "readings": [
    {
      "id": "679cb354b477993c5cdda049",
      "title": "The Remedy for Unemployment",
      "authors": [
        {
          "_id": ObjectId("679cb31ab477993c5cdbfa9e"),
          "fullName": "Alfred Russel Wallace"
        }
      ],
      "returnDate": "2024-10-03",
      "branch": { "type": "Point", "coordinates": [10.3975611,
         43.7193769] }
    },
    ...
  ],
  "savedBooks": [
    {
      "id": "679cb362b477993c5cde074e",
      "title": "The truth about Ireland",
      "authors": [
        {
          "_id": ObjectId("679cb31ab477993c5cdc8652"),
          "fullName": "Alexander Corkey"
        }
      ]
    },
    ...
  ]
}
```


This design allows us to focus on the core functionalities of our application:

- ❑ Book exploration with **direct view of availabilities** in different branches
- ❑ Book **exploration by author**

- ➢ **Embedding** allows us to support these frequent queries efficiently

Document DB Design (iii)

Average Age of Active Users by City

- Filters users who borrowed books in the last year.
- Groups by city, calculates **average age and user count**.

```
db.users.aggregate([
    {
    $match: {
        "readings.returnDate": {
        $gte: new Date(new Date().setFullYear(new
            Date().getFullYear() - 1))
            .toISOString()
            .split('T')[0]
        }
    }
    },
    {
    $group: {
        _id: "$address.city",
        average_age: {
        $avg: {
            $dateDiff: {
            startDate: { $toDate: "$dateOfBirth" },
            endDate: "$$NOW",
            unit: "year"
            }
        }
        },
        total_users: { $sum: 1 }
    }
    }
]);
```

Other queries we implemented:

Book Utilization Analysis

- Computes **total readings per book**.
- Categorizes **underutilized** (many copies, low reads) and **overutilized** (few copies, high reads).

Most Read Books by Age Group

- Filters readings by year, groups by **age group**.

Finding Books in Nearby Libraries

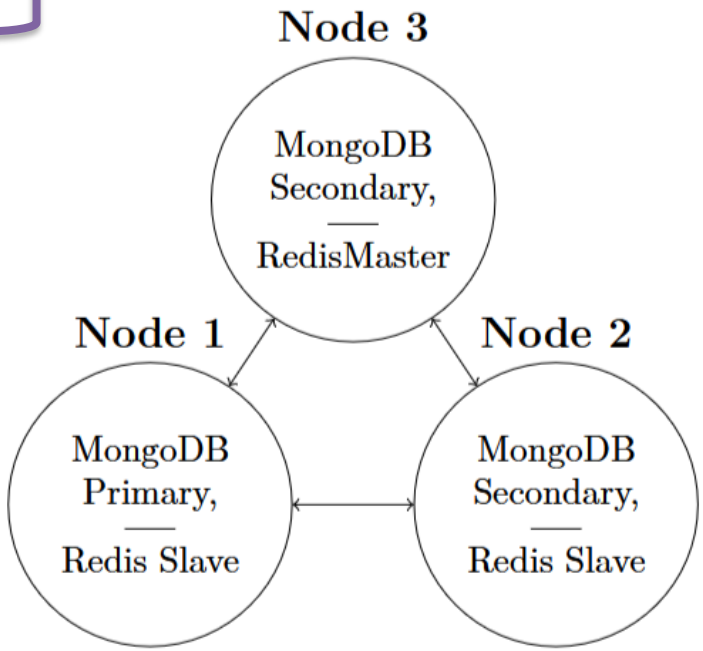
- Filters libraries within **geo-radius**.

MongoDB Replica Set Configuration

❑ **w = majority**
⟹ eventual consistency but data loss is minimised

❑ **readPreference = secondaryPreferred**
⟹ serving stale data from Mongo is acceptable

High-availability & Data Loss minimisation (**non-functional requirements**) are supported



➢ Tends to the **AP side of the CAP Triangle** → eventual consistency, but with the compromise of w = majority

MongoDB Indexes

```
db.books.createIndex(
    { title: 1 },
    { collation: { locale: "en", strength: 2, alternate: "shifted"
       } }
)
```

- Over 100,000 documents examined → 1 examined key
- **30ms → near-zero**

➢ Allows case-insensitive search and ignore punctuation/white spaces

```
db.books.createIndex({ categories: 1 })
```

30ms → 2ms

```
db.authors.createIndex({ fullName: 1 })
```

10ms → near-zero

```
db.books.createIndex({ readingsCount: -1 })
```

28ms → nearly-instant sorting

➢ Support frequently-issued queries

```
db.books.createIndex({ "branches.location": "2dsphere" })
```

➢ Required for the $geoWithin operation

Key-value DB Design (i)

REDIS

Consistent naming convention: `EntityName:⟨EntityId⟩:EntityAttribute`

- **Key-Value Pair**: `book:⟨bookId⟩:lib:⟨libraryId⟩:avail`, stores the current number of available copies of a book in a specific library
- **Hashes**: Each hash contains multiple fields related to a common purpose
  - Efficient retrieval of the whole hash and operation on specific hash entries

`lib:⟨libraryId⟩:res`
`lib:⟨libraryId⟩:loans`
`lib:⟨libraryId⟩:overdue`
	`user:⟨userId⟩:book:⟨bookId⟩:start` → reservation/loan timestamp

`user:⟨userId⟩:active`	`lib:⟨libraryId⟩:book:⟨bookId⟩:info` → small JSON object: `{ "status": "LOANED", "deadlineDate": 1738272760031, "bookId": "679a76b930f95ec8c6e7c608", "libraryId": "679a76df5cf745180198d6a1", "timestamp": 1738272760031 }`

Key-value DB Design (ii)

- **Sorted Sets (ZSets)**: Used for managing expiration-based operations (tracking reservation and loan deadlines)
  - `zset:res-exp` & `zset:loan-exp`:
    `user:⟨userId⟩:book:⟨bookId⟩:lib:⟨libraryId⟩:exp` → expiration timestamp
  - Used instead of TTL messages (non-fault-tolerant)
  - **Worker periodically retrieves entries with expired value** and performs required operation (e.g. increase availability if user doesn't pick up book within 3 days after reservation)
  - **Retrieving only KV pairs whose value is below current timestamp** very efficient in Redis' sorted sets

- **Streams**: Used to support event-driven architecture (asynchronous processing for some operations, explained later)

Key-value DB Design (iii)

Example of Queries

- **Lua scripts**: transaction-like execution of operations
  - ✓ Allows to perform complex operations (MULTI/EXEC not enough)
  - ✓ Cached into Redis, invocation is quick (only parameters are passed)
    - ➢ This **prevents race conditions** in book reservation

```lua
-- Check if user reservation exists
if redis.call('HEXISTS', userResLoansKey, userReservationField) == 0 then
    return cjson.encode({["err"] = "User reservation does not exist"})
end

-- Check if library reservation exists
if redis.call('HEXISTS', libraryResKey, libraryReservationField) == 0 then
    return cjson.encode({["err"] = "Library reservation does not exist"})
end

-- Delete reservation from user reservations hash
redis.call('HDEL', userResLoansKey, userReservationField)

-- Delete reservation from library reservations hash
redis.call('HDEL', libraryResKey, libraryReservationField)

-- Remove the reservation from the expiration ZSet
redis.call('ZREM', expirationZSetKey, zsetMember)

-- Increment availability
redis.call('INCR', availabilityKey)

-- Return success
```

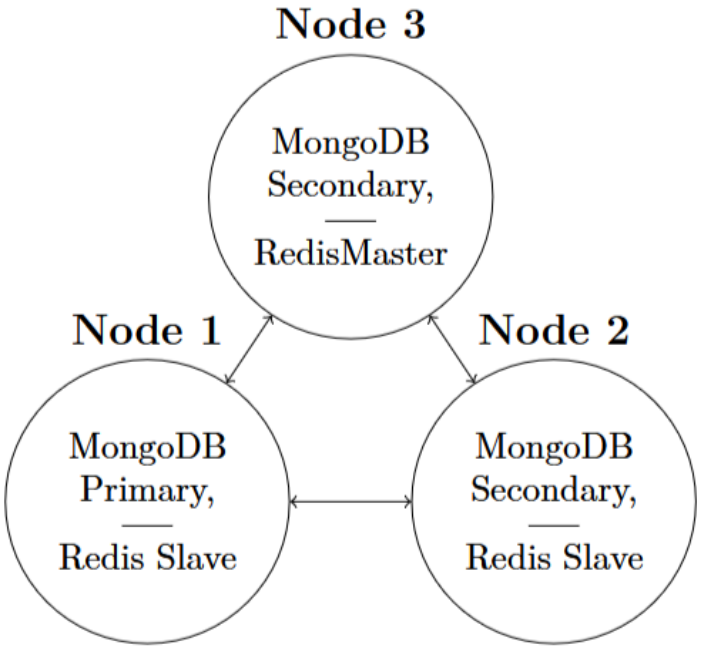
Example: **Book Reservation**

- Check whether user can reserve book
- Check book's availability
- Insert reservation into library's hash
- Insert reservation into user's hash
- Decrease book's availability
- Create TTLs and entry in Zset

➢ **Data integrity** + **Avoid race conditions** (**non-functional requirements**)

Redis Replica Set Configuration

- **Redis Sentinel** mechanism to handle failover
- **min-replicas-to-write 1** (min-replicas-max-lag 10)
- **AOF: appendfsync always**
- **Periodic RDB snapshots** (interval depends on number of writes)
- Key eviction policy: **NO eviction** (Redis Dump is far from saturating main memory [~20MB])



- support **data integrity**, **data loss minimisation** & **high-availability** (non-functional requirements)
- Tends to the **CP side of the CAP Triangle** → consistency but a certain degree of availability via replication

Discussion on Data Sharding

REDIS SHARDING STRATEGY

- Keep **all availability, reservation, and loan-related keys/hashes for a library in the same shard** ⟹ Ensures atomicity for operations modifying multiple Redis structures (reservations, loans)
- Challenge: User activity hash not always in the same shard as library's structures (users can reserve books in all libraries) ⟹ **remove user hashes and persist them asynchronously to MongoDB**

→ **write load evenly distributed**

MONGODB SHARDING STRATEGY

- **Read-heavy workload** ⟹ sharding not needed, replica sets handle scaling more efficiently
- **Sharding adds overhead** for queries like title-based searches and statistics ⟹ **Cross-shard merging** would slow down retrieval.

Inter- and Intra-DB Consistency (i)

Outbox Pattern for Asynchronous Processing

Outbox Collection in MongoDB

```
{
    "_id": "679bb09e4a78c01e693af661",
    "type": "DECREMENT_BOOK_COPIES",
    "payload": {
        "libraryId": "679a76df5cf745180198d68e",
        "bookId": "679bafe94a78c01e693af65e"
    },
    "status": "COMPLETED",
    "retryCount": 0,
    "createdAt": "2025-01-30T17:02:22.532Z",
    "updatedAt": "2025-01-30T17:02:22.532Z",
    "nextRetryAt": "2025-01-30T17:02:22.532Z",

}
```

- **Payload** tailored for the specific processing needed
- **Status**: pending, in progress, completed, failed, retry scheduled
- Reliable execution mechanism: **retries with exponential backoff**

- **OutboxWorker** (server-side): periodically retrieves and processes pending tasks
- The worker invokes the corresponding **OutboxTaskProcessor** implementing the **specific logic** required for handling that **task type**

Inter- and Intra-DB Consistency (ii)

Operations on Multiple MongoDB Collections

- Example: **new book addition by the admin**

1. **Check** whether all specified authors already have a document in the Authors collection, otherwise return appropriate error
2. **Book document is added** to the Books collection
3. **Outbox task** for updating authors' documents is **created** with the new book details
4. **Eventually the OutboxWorker handles the task** by invoking the appropriate processor
5. Processor inserts embedded book details in all involved authors' documents
6. On failure, the task is retried up to five times; if it still fails, it is marked as failed, requiring human intervention (only in case of major system issues).

Eventual consistency ⟹ **data integrity** is preserved without sacrificing **high availability** (non-functional requirements)

Inter- and Intra-DB Consistency (iii)

Consistency Between MongoDB and Redis

- **Redis Streams** used to achieve safe propagation of Redis updates to MongoDB without immediate dual-access ⟹ **preserve Redis speed**

➢ Example: **user returns loaned book** to the library

1. Admin uses the "Complete a Loan" function
2. **Entry is removed from library's and user's hashes**, **book availability is increased**, entry from Zset is removed (30-day expiration)
3. **Message is published in `stream:completed-loans`** with read book information (user, library, book)
4. **Worker on server-side** eventually **reads the message, creates outbox task** related to the event and sends acknowledgment to the stream
5. **Outbox logic**: OutboxWorker eventually handles the outbox task invoking the right processor
6. Processor inserts read book entry in user's document

Eventual consistency ⟹ **data integrity** is preserved without sacrificing **high availability** (non-functional requirements)

Swagger UI REST APIs documentation

Distribooked API 1.0.0 OAS 3.1

API documentation for 'Distribooked'

Servers

http://localhost:8080 - Generated server url

Authorize

Loans Operations related to loans, for admins only.

Usage statistics Operations related to statistics, for admins only.

Library-related searches Endpoints available to everyone, without authentication.

Catalogue Management Operations related to catalogue management, for admins only.

Author-related searches Endpoints available to everyone, without authentication.

Reservations Operations related to book reservations, for registered users.

Book-related searches Endpoints available to everyone, without authentication.

User Operations related to users, for registered users.

Authentication Operations users/admins authentication.

Code	Description	Links
201	User registered successfully Media type: */* Controls Accept header. Example Value | Schema	*No links*

```
{
  "timestamp": "2025-01-31T12:00:00.000Z",
  "status": 201,
  "message": "User registered successfully",
  "data": {
    "id": "889a770c426571df71448d64",
    "username": "johndoe"
  },
  "path": "/api/v1/auth/register"
}
```

400	Invalid registration details Media type: */* Example Value | Schema	*No links*

```
{
  "type": "string",
  "title": "string",
  "status": "4xx, 5xx",
  "detail": "string",
  "instance": "/api/v1/resource/{id}",
  "timestamp": 1738325243029,
  "error": "string"
}
```

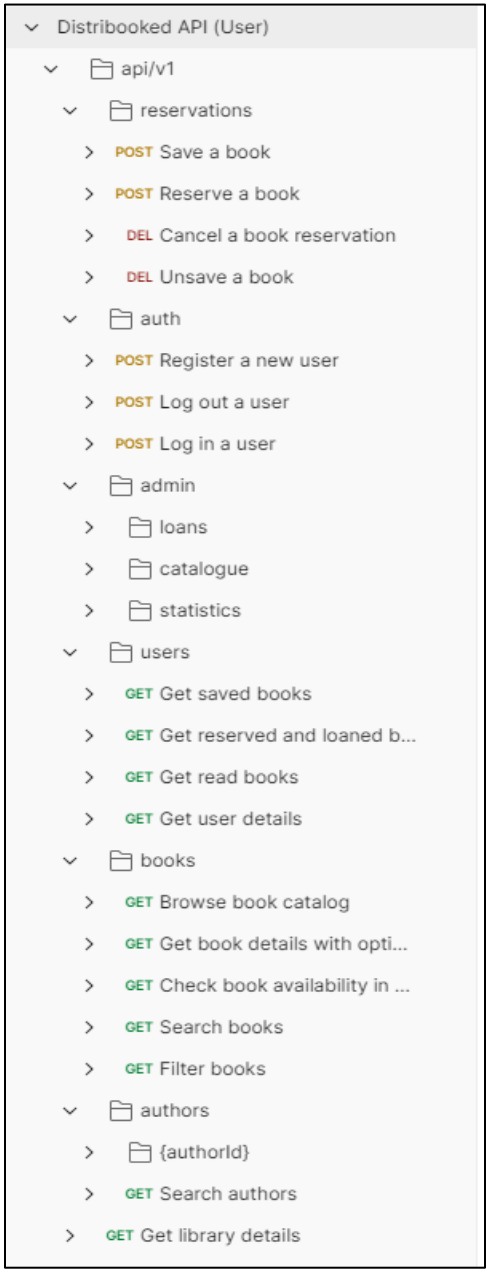
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

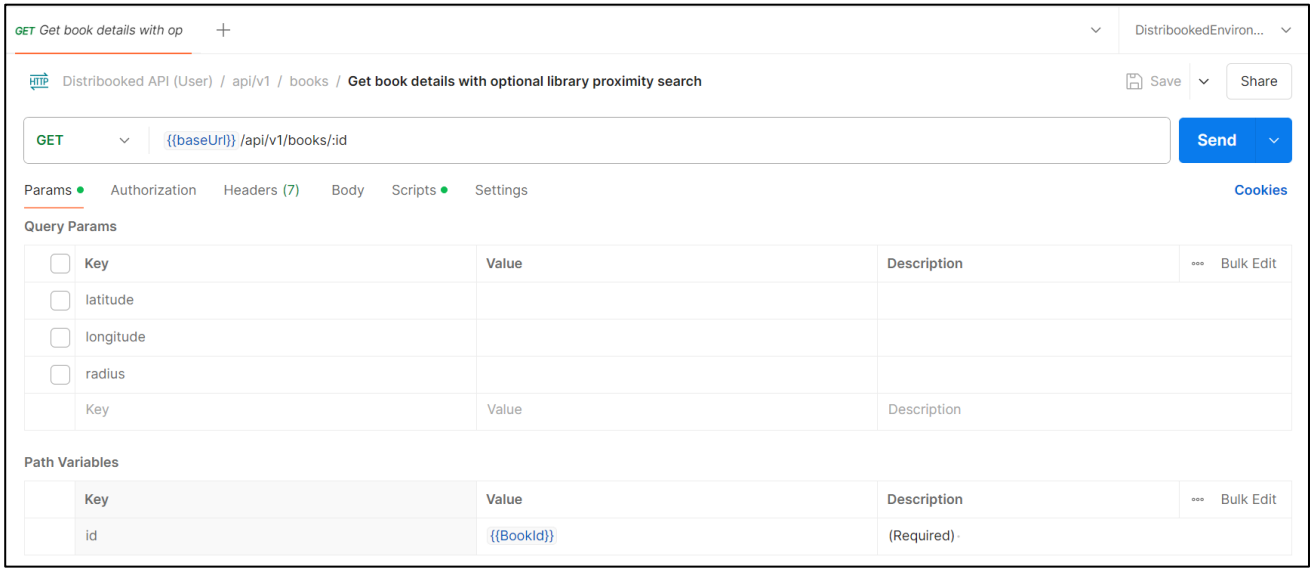
IN SUPREMÆ DIGNITATIS 1343

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Live Demo with Postman





Load tests in stress scenarios

150 parallel requests (50 logins, 50 book searches, and 50 reservations) → **average response time 0.732 second**

- This supports the last **non-functional requirement** (“throughput of at least 50 operations per second, considering a typical workload distribution”)

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0