Cloud Computing Systems - Backend for an auction system

Frederico Luz*
fc.luz@campus.fct.unl.pt
MIEI, DI, FCT, UNL

Rui Genovevo†
r.genovevo@campus.fct.unl.pt
MIEI, DI, FCT, UNL

ABSTRACT

The project will consist in the design and implementation of the backend for an auction system like EBay and companion testing scripts. The project will be implemented in Azure, using the resources available to students under the Azure student program.

ACM Reference Format:
Frederico Luz and Rui Genovevo. 2022. Cloud Computing Systems - Backend for an auction system. In *The Projects of CSS - first delivery, 2022, Faculdade de Ciências e Tecnologia, NOVA University of Lisbon, Portugal*. ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

The goal of this project is to understand how services available in cloud computing platforms can be used for creating applications that are scalable, fast, and highly available. The system will manage auctions. Users can create auctions and bid on open auctions. User can also pose question about the product of an auction. A question can only be answered by the user that created the auction, and there can be only one answer for each question.

2 DESIGN

For implementing its features, the system maintains the following information:

- **Users**: information about users, including the nickname, name, (hash of the) password, photo;
- **Media**: manages images used in the system;
- **Auction**: information about auctions, including for each auction a title, a description, an image, the owner (the user that created the auction), the end time of the auction (i.e. until when bid can be made), the minimum price, the winner bid for auctions that have been closed, the status of the auction (open, closed, deleted).
- **Bid**: Each bid must include the auction it belongs to, the user that made the bid, the time it was created and the value of the bid.
- **Questions**: auctions' questions and replies. Each question must include the auction it refers to, the user that posed the question, the text of the message and its reply.
- **Winner**: user with highest bid in an auction when it ends.

The base version consists in the basic version of the backend service, that includes the base REST endpoints associated with Users, Media, Auction, Bid, Questions.
You must decide how the service information is stored in the underlying storage services / database services, creating the necessary objects/documents.
Besides the base endpoints associated with the previous resources, the service provides the following endpoints:

- List of auctions about to close;
- List of auctions of a given user.

The system uses the following Azure services:

- App service: for creating the REST servers
- Blob storage: for storing BLOB
- Cosmos DB database: for storing structure data
- Azure Cache for Redis service
- Azure Functions: to perform some periodic computation

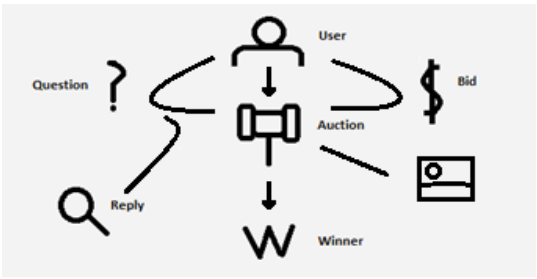

Figure 1: Workflow of the project

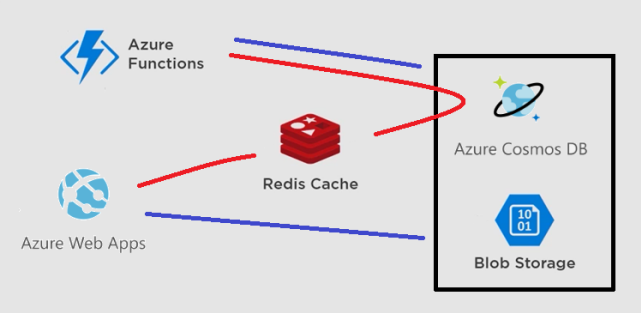

Figure 2: Systems of the project

Azure Functions:
- perform timed functions directly on the database (e.g., close expired auctions).
- send request to update Redis Cache with database information.

Azure WebApps:
- perform operations directly on the database.
- check cache for necessary information on beginning of operation and update cache if needed on operation finished.
- check validity of session with authentication on cache.

*Student number 51162
†Student number 45576

CCS22/23, 2022, Faculdade de Ciências e Tecnologia, NOVA University of Lisbon, Portugal

3 IMPLEMENTATION

3.1 Basic Operations

The system must support, at least, the following basic operations using standard REST rules for defining endpoints:

• **Users** (/rest/user):

-writeUser @POST @Path("/"): create a new user. The inputs for this endpoint its a json like:

```
{
    "id" : "1",
    "name" : "user",
    "nickname" : "usernickname",
    "pwd" : "password123",
    "photoId" : "storeageimage.com/image112"
}
```

-delUser @DELETE() @Path("/id"): delete user using its id and a cookie of the session.

After a user is deleted, auctions and bids from the user appear as been performed by a default "Deleted User" user.

-updateUser @PUT() @Path("/id"): update user using its id. The inputs for this endpoint its a cookie for the session and a json like:

```
{
    "id" : "1",
    "name" : "user",
    "nickname" : "usernickname",
    "pwd" : "password123",
    "photoId" : "storeageimage.com/image112"
}
```

-getUser @GET @Path("/id"): get user using its id.

-getAuctionsOfUser @GET @Path("/id/auctions"): get auctions of a user using its id.

-getOpenAuctionsOfUser @GET @Path("/id/auctionsopen"): get auctions of a user that are open using its id and a query param ?status="OPEN".

-getAuctionsUserFollow @GET @Path("/id/auctions/following"): get auction that a user had a bid for using its id

-auth @POST @Path("/auth"): make login using user nickname and password. The inputs for this endpoint its a json like:

```
{
    "user": "usernickname",
    "pwd": "password123"
}
```

• **Media** (/rest/media): upload media, download media.

-UploadImages @POST @Path("/"): upload an image to storage using a byte array as input data.

-DownloadImages @GET @Path("/id"): download an image using its id.

Operations that create an entity with an associated media object (e.g., users and auctions) must execute two consecutive calls, the first for uploading the image and the second for the corresponding operation.

• **Auction** (/rest/auction):

-createAuction @POST @Path("/"): create an auction using a cookie for the session and a json for the input of the endpoint like this:

```
{
    "id" : "1",
    "title" : "Picasso Painting",
    "description" : "Selling Portrait of Dora Maar",
    "imageId" : "storeageimage.com/picasso1",
    "ownerId" : "1",
    "endTime": "2022-11-22T21:30:00.000Z",
    "minPrice" : 1500.0
}
```

-updateAuction @PUT() @Path("/id") update an auction using its id. The inputs for the endpoints are a cookie for the session and a json like:

```
{
    "id" : "1",
    "title" : "Picasso Painting",
    "description" : "Selling Portrait of Dora Maar",
    "imageId" : "storeageimage.com/picasso1",
    "ownerId" : "1",
    "endTime": "2022-11-22T21:30:00.000Z",
    "minPrice" : 1500.0,
    "winnerId" : "1",
    "status" : "CLOSED"
}
```

-getAuctionsToClose @GET() @Path("/auctionsToClose"): get auctions that are less than an hour about to close

• **Bid** (/rest/auction/id/bid):

-createBid @POST @Path("/"): create a new bid for an auction using the auction id as path param, a cookie for the session and a json for the input of the endpoint like this:

```
{
    "id" : "1",
    "auctionId" : "11",
    "userId" : "1",
    "time": "2022-11-21T20:14:00.000Z",
    "value": 1700.0
}
```

-listBids @GET() @Path("/"): list all the bids of an auction using the id of the auction as path param.

• **Questions** (/rest/auction/id/question):

-createQuestion @POST @Path("/"): create a question for a specific auction using the auction id for the path param and a json for the input of the endpoint like this:

```
{
    "id" : 1,
    "auctionId" : "11",
    "userId" : "1",
    "message" : "im sending a question"
}
```

-listQuestions @GET @Path("/list"): list all the questions of an auction using the auction id for the path param.

-replyQuestion @POST @Path("/QuestionId/reply"): reply to a specific question of a specific auction using the auction id and the question id for the path param. The input for the endpoit is a json like this:

```
{
    "reply" : "replying to a question"
}
```

A question can only be replied by the creator of the auction and can only be replied one time.

3.2 Application-level caching

Caching is fundamental for improving the performance of a system by avoiding slow database operations and by avoiding repeating the same computations multiple times.
Improve the base version of backend by using application-level caching. To this end, use the Azure Cache for Redis service.

Cache is always active in authentication purposes for security reasons (e.g., only the session with owner id can edit the respective auction), but can be deactivated at variable USE_CACHE in management, for methods that also store information in cache (e.g., users and auctions). That's the comparison used in experimental evaluation.

There are 4 different replication uses:

• **Authentication**: The network always makes use of authentication caching, a user's request to the application server is only forwarded when he is authorized and verified as having the necessary permission to access the requested content. This authentication decision is cached when method auth is called and a session cookie is stored, reducing the need to constantly look for that information which changes infrequently. Authentication is a relatively expensive operation to perform, so by reducing the number of lookups for authentication, the overall performance of the web application is improved and delays in delivery times are reduced.

Also should monitor the session timeout value used – this is the period of time after which inactive users must re-authenticate to regain access. Longer authentication cache timeout values can increase security risks. However, smaller timeout values can affect performance, since the server needs to access the user registry more frequently.

• **Update**: Copies of data are stored in designated locations, either locally or on the server. This saves resources trying to run difficult queries on databases or those loading large images from the server every time a website loads.

• **Check**: Browsers will check designated cache locations before requesting the resource from the server, and if it's found, it will be loaded from the cache instead of having it downloaded, which would be significantly slower.

• **Trigger**: Special update for information/lists that need to be up to date and are important/relevant for the system to maintain atomicity.

3.3 Azure Functions

Use Azure functions to implement some functionality in your application (**e.g.**, closing auctions, define auction winner).

Azure Functions is a serverless solution that allows to write less code, maintain less infrastructure, and save on costs. The cloud infrastructure provides the up-to-date resources needed to keep applications running. Allows to implement system's logic into readily available blocks of code. These code blocks are called "functions". Different functions can run anytime you need to respond to critical events (triggers).

Most common types of triggers available in Azure:

Timer Trigger - is called on a predefined schedule. We can set the time for execution of the Azure Function using this trigger.

Blob Trigger - will get fired when a new or updated blob is detected. The blob contents are passed on as input to the function.

CosmosDB Trigger - uses the Azure Cosmos DB Change Feed to listen for inserts and updates across partitions.

HTTP Trigger - gets fired when the HTTP request comes.

• **CloseExpiredAuctions** (TimeTrigger): Verifies auctions close date/time with system. If auction expired (close time <= system time) then state of auction is changed from open to close, no more bids can be made on this auction and triggers method to declared auction winner (user with highest bid).

• **DefineAuctionWinner** (HTTPTrigger): When an auction expires and is closed then this event triggers, checks the list of bids for the highest value and returns the user, declaring the winner of that auction.

• **GarbageCollector** (TimeTrigger): System Garbage Collection (GC) is a memory recovery feature built into programming languages such as C and Java. A GC-enabled programming language includes one or more garbage collectors (GC engines) that automatically free up memory space that has been allocated to objects no longer needed by the program.

• **SyncStorage** (TimeTrigger): Storage synchronization is a method of keeping files that are stored in several different physical locations up to date. Cloud often offers software that helps with this process and is increasingly important in it.

3.4 Geo-Replicated

Extends system for having an efficient geo-replicated deployment in data centers.

Note: Cosmos DB allows to specify the data centers in which data should be replicated. The same does not occur for Blob Storage. A way to address this issue is to replicate the data in the blob store in multiple data centers using Azure functions.

Objective to tolerate both data center failures and complete region failures, it is possible to read from other region – potential better latency, with data replicated synchronously across Azure availability zones in the primary region, then replicated asynchronously to the secondary region.

Synchronous vs. asynchronous replication

• Synchronous replication in the primary zone for establishing the "official" state of the blob.

• Asynchronous replication to secondary zone, allowing to return the result of a write operation to the client without incurring in inter-region latency.

Azure Replication

CosmosDB - Azure as an option to replicate database itself.
BlobStorage - TimerTrigger function (SyncStorage) to replicate resources between blobs.

4 EVALUATION

Use of developed scripts for testing backend service, using artillery. A base version is provided that extends to test all operations of the system.

Evaluation of the system comparing the following settings, using artillery:

a) Application deployed in region West Europe, with caching;

b) Application deployed in region West Europe, without caching;

c) Application deployed in some other region outside of Europe, with or without caching;

d) Evaluate your work by deploying your system in multiple data centers and having clients calling the different replicas.

5 CONCLUSION

Caching vs No Caching (same region)

Caching improves application performance by storing critical pieces of data in memory for low latency access. Cached information may include the results of I/O- intensive database queries or the results of computationally-intensive calculations, minimizes the number of queries that are sent to server, which takes longer to process than cached results. This are the main reason for caching and how it improves performance, load time and lower network costs.

Different Regions (no caching)
Some of the disadvantages of replication are...

- Replicating data requires you to invest in numerous hardware and software, with a complete technical setup to ensure a smooth replication process.
- Task of replication without any bugs, errors, etc., requires you to set up a reaction pipeline.
- A large amount of data flows from your data source to the destination database, to ensure a smooth flow of information and prevent any loss of data, having sufficient bandwidth is necessary.

Since Azure has high availability and reliability (due to size and infrastructure), it ensures a platform for solving most of this problems. Making, that way, more valuable for the user the advantages of replication, such as...

- Providing lower-latency data access in different geographic regions.
- No additional charge for using cross-region replication.
- Physical datacenter separation reduces the likelihood of natural disasters, civil unrest, power outages, or physical network outages affecting both regions at once. (Enhances the resilience and reliability of systems by storing data at multiple sites across the network)

Multiple data centers (clients calling the different replicas)
As explained before, more than anything 3 points are very important:

Asynchronous System - able to deal with multiple requests simultaneously, thus completing more operations in a much shorter period of time.

Load Balancing - evenly distribute network traffic to prevent failure caused by overloading a particular resource.

Region Availability - isolated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity.

REFERENCES