DESIGN CHALLENGE SOLUTION

Snapdocs @ Aug 31, 2020

by

Sudhakar Balakrishnan

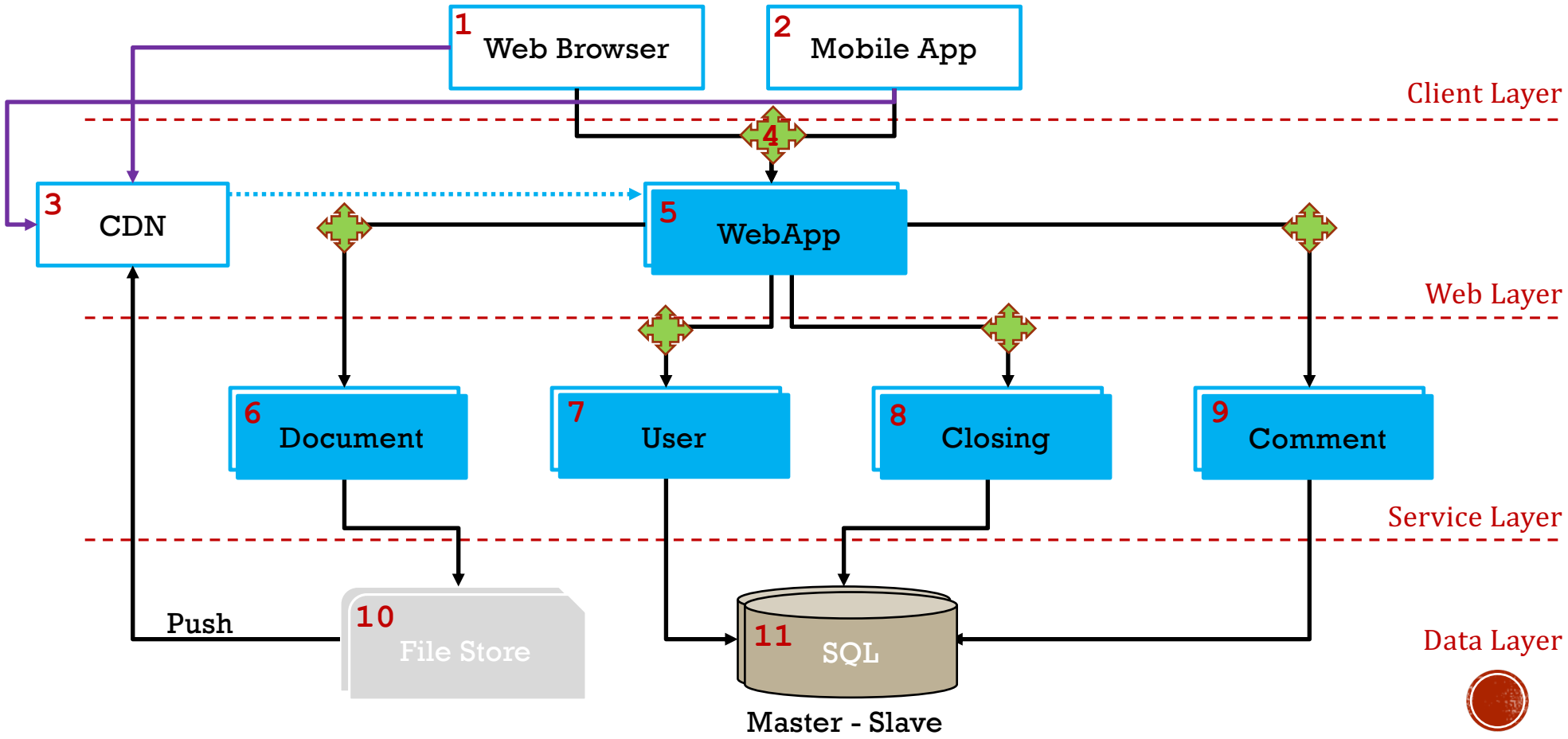
FUNCTIONAL REQUIREMENTS

- User – The system supports 4 types of users. CRUD operations supported.
- Closing – Allow users to create a Closing, and return a closingPkgId as response.
- Document – System allow users to upload documents once Closing request is in place. Doc owners are allowed to delete their doc. Documents are not supposed to be edited or altered in any way.
- Comment – Users can add his/her comments to specific documents on file. Any user can perform CRUD operations on the comments.
- Note: *Multiple users should be able to work on the closing package at the same time.*

NON-FUNCTIONAL REQUIREMENTS

- System should be reliable and horizontally scalable as the company grows.
- Application tend to be read heavy.
- Capacity planning:
  - Expected 100 closing request daily (or) 3K monthly (or) 36K yearly
  - Each closing deals with one document with avg size of 5MB. File store space needs – 500MB daily (or) 15GB monthly (or) 180GB yearly.
  - Typically on average 5 users are associated with a closing request.
    - 1 notary, 1 title, 1 lender, 2 consumers.
  - So, on average 500 users may login to the system daily.

SYSTEM DESIGN



DESIGN SUGGESTIONS

- #1. There is no clear requirements defined for official Browser support. It is a good practice to officially support min of 2 browsers. Ensure the UI design is responsive and the testing include device testing.
- #2. There was no requirements on Mobile. Knowing about customers, most user often check the status, follow up with stakeholders, et all. Providing a mobile client will be beneficial for customers to remotely manage their closing process.
- #3. Initially we may choose not to use a CDN provider. It's a good idea as the product grows globally, where the internet performance may varies. Also a good option to consider, if the current user base grow and the latency for loading document considerably increase.
- #4. A Load balancer is crucial to ensure the WebApp does not become single point of failure. With the LB being external, its good practice to replicate the setup with multiple LB to ensure user never experience server failure errors. And to protect the platform from security issues, it is essential to place external LB behind product like Akamai WAF/DDoS protection.
- #5. WebApp is the first application module to handle the request. It is a good practice to enable Caching on Web Server as the system is read heavy. A typical WebApp would include filters to handle security, authentication, instrumentation, logs, etc. The WebApp would process in the incoming request and call one of the services to complete its tasks. With web and services logic isolated, it gives system flexibility to scale both layers based on system load. Since the WebApp supports file upload, ensure the uploaded files are scanned for security vulnerabilities before forwarding request to document service.

DESIGN SUGGESTIONS

- #6, #7, #8, #9 – the domain service handles all operations related to their business logic. If the initial version does not include CDN, ensure the document service is scaled based on the system load and to reduce latency use Async calls when non essential tasks are performed. The individual stateless domain services would have their own Load balancers to allow scaling the services horizontally based on system usage.
- #10 – If a CDN provider is used use the Push method to sync newly created documents to the CDN server.
- #11 – the initial solution may use a MySQL or Postgres database with a Master-Slave setup. If the database performance is impacted due to heavy reads, a read replica can be created. Alternatively cache can be also be used.
- And lastly to support multiple users work on the same closing package at the same time:
  - User sessions are managed by the web application and no need to lock resources when users open/work on the same package.
  - CRUD on documents – Users are allowed to upload/create or delete their own document, so no special handling required.
  - CRUD on comments – Use database locks to check and commit, the comments scheme operations. This ensures the comment operations are allowed only when its linked document record is active and not simultaneously delete by its owner.

API DESIGN

1. User API

HTTP POST: /v1/users - Create a new user

```
{
   "firstName": "James",
   "lastName": "Bond",
   "emailId": "james.bond@movie.com",
   "password": "6b2c3127fa2022564065de5d900ab29f",
   "phoneNum": "+44-20-1000-0007"
}
```

HTTP GET: /v1/users - Read one or more users

HTTP PUT: /v1/users - Update one or more users

HTTP DELETE: /v1/users - Delete one or more users

HTTP GET: /v1/users/{id} - Read/return a specific user

HTTP PUT: /v1/users/{id} - Update a specific user info

HTTP DELETE: /v1/users/{id} - Delete a specific user

API DESIGN

2. Closing API

```
HTTP POST: /v1/closings
```

3. Document API

```
HTTP POST: /v1/documents
HTTP GET: /v1/documents
HTTP DELETE: /v1/documents
```

4. Comment API

```
HTTP POST: /v1/comments
HTTP GET: /v1/comments
HTTP PUT: /v1/comments
HTTP DELETE: /v1/comments
```

SERVICE OBJECTS (POJO)

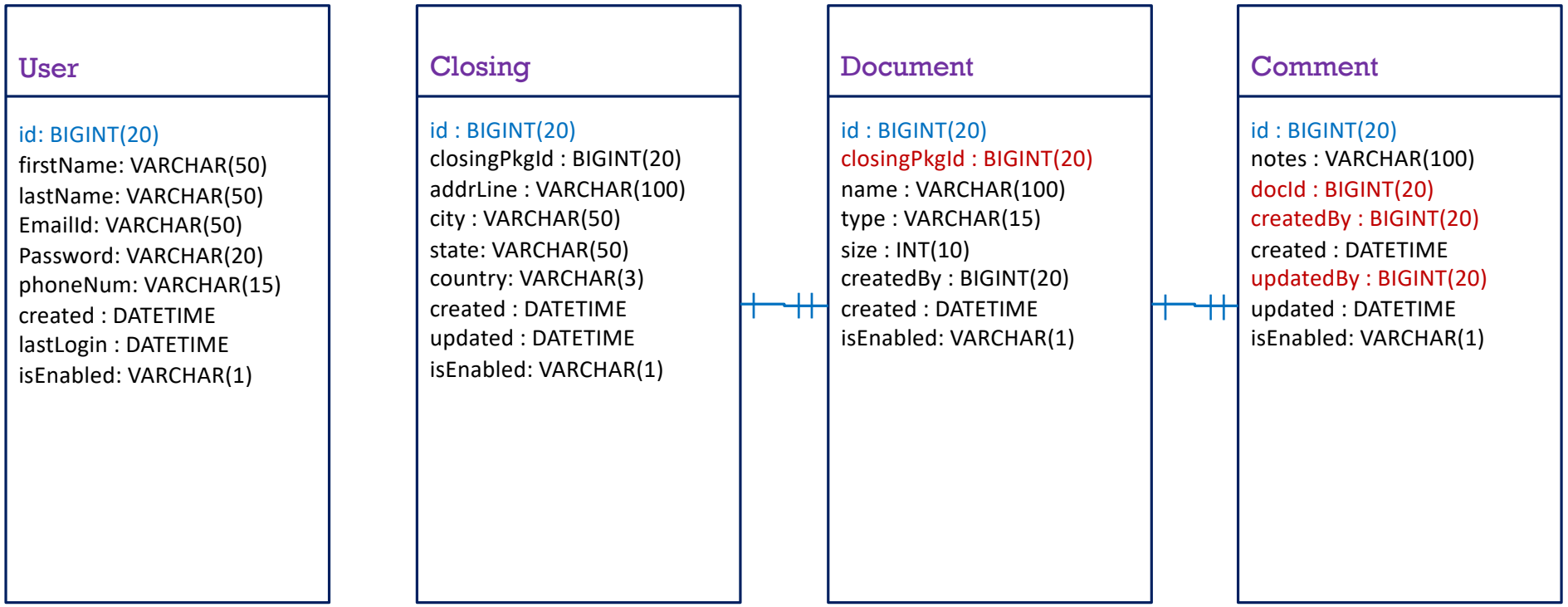
User {
Id: Long
firstName: String
lastName: String
emailId: String
phoneNum: String
}

Closing {
Id: Long
closingPkgId: Long
addrLine: String
city: String
state: String
country: String
}

Document {
Id: Long
closingPkgId: Long
docName: String
docType: String
docSize: String
}

Comment {
Id: Long
description: String
docId: Long
createdBy: Long
created: Date
updatedBy: Long
updated: Date
}

DB DESIGN


User
id: BIGINT(20)
firstName: VARCHAR(50)
lastName: VARCHAR(50)
EmailId: VARCHAR(50)
Password: VARCHAR(20)
phoneNum: VARCHAR(15)
created : DATETIME
lastLogin : DATETIME
isEnabled: VARCHAR(1)
Closing
id : BIGINT(20)
closingPkgId : BIGINT(20)
addrLine : VARCHAR(100)
city : VARCHAR(50)
state: VARCHAR(50)
country: VARCHAR(3)
created : DATETIME
updated : DATETIME
isEnabled: VARCHAR(1)
Document
id : BIGINT(20)
closingPkgId : BIGINT(20)
name : VARCHAR(100)
type : VARCHAR(15)
size : INT(10)
createdBy : BIGINT(20)
created : DATETIME
isEnabled: VARCHAR(1)
Comment
id : BIGINT(20)
notes : VARCHAR(100)
docId : BIGINT(20)
createdBy : BIGINT(20)
created : DATETIME
updatedBy : BIGINT(20)
updated : DATETIME
isEnabled: VARCHAR(1)