P1Group20 Design Document

- Jerry Jiang, jj71, jryjng
- Charlie Wells, crw16, wellsch
- Jingwu Wang, jw133, Jingwu-01

Design Principles

Single Responsibility Principle:
Each package has a single, clearly defined role. Each package only contains functions and structs relevant to its functionality, required helpers for its functionality, and test functions. For example, the document package only contains methods to create, manipulate, and obtain the data of a document, the paths package only has the responsibility of processing pathname strings from requests, authentication handles all authentication, etc.

Interface Segregation Principle:
The owldb database is composed of 3 elements: documents, collections that store documents, and collection holders that store collections. Each of these 3 elements are represented as an interface. In those interface definitions, only get operations and HTTP handler methods are required. Functionality such as document and collection subscription, document patching, document metadata, document overwriting, and document children are segregated into other interfaces - Subscribable, Patchable, Overwritable, HasMetadata, and Postable. Therefore, the client does not need to implement subscriptions, patching, allow overwriting of documents, and other operations if they deem those unnecessary for their own implementation of documents and collections.

Open Closed Principle:
As mentioned previously, the owldb database utilizes three interfaces to represent the data structures present within the database: documents (IDocument), collections (ICollection), and collection holders (ICollectionHolder). In our implementation, these three types are referenced instead of our concrete implementations of their interfaces. This allows the client to add or substitute their own implementation of these data structures without having to modify any of the original source code.

Concurrency

Concurrency is managed in packages such as skiplist, authentication, and subscribe.

Skiplist:
We use skip lists to quickly store and retrieve data in our system by names. Specifically, we use skip lists to store documents and collections. To manage concurrency in skip lists, each node in our skip lists has a mutual exclusion lock to prevent concurrent modification. In addition, each node and the root skip list data structure has atomic boolean and integers to track if the data has been modified without having to lock to allow concurrent reads.

Authentication:
The authentication package implements an “Authentication” struct that stores user sessions, leveraging “sync.Map” for thread-safe concurrent access to session data. “sync.Map” is a concurrent data structure that ensures multiple goroutines could simultaneously read from and write to the session data without causing race conditions. Key methods like “ValidateToken”, “login”, and “logout” manipulate this concurrent map safely through sync.Map methods like “Load”, “Delete”, and “Store”. Session information is immutable once created, further enhancing concurrency safety. The generation of cryptographically secure tokens is stateless, which ensures its safety for concurrent calls.

Subscribe:
The subscribe package manages concurrency with Go channels, which enables real-time updates to subscribers for document/collection changes. The “Subscribe” struct has two channels, “UpdateCh” for data updates and “DeleteCh” for data deletions. The `ServeSubscriber` method runs in its own goroutine, continuously monitoring these channels and the client's connection state. It uses a `select` statement to simultaneously listen for various events. When a data update or deletion occurs, corresponding messages are sent to the subscriber. To maintain an active connection, a comment is dispatched every 15 seconds. If the client disconnects, the goroutine detects this through the request's context and terminates, ensuring efficient concurrency handling without leaving hanging resources.

Architectural Diagram

This design diagram depicts all the interactions between different components within the container that is owlDB. Users of owlDB are other software systems, which make REST API requests to the system, which are dispatched to the proper handlers by a ServeMux.

Authentication is handled by the authentication handler, which contains a map of login tokens to user credentials and when their tokens will expire, and is embedded via its interface in the DB request handler, which calls its validate token method on user requests.

DB Requests are handled by the DB Request handler, which uses the path processor to locate the proper resource, and then calls the method from the proper interface (collection or document) on that resource. The collection and document packages implement several different interfaces for interacting with each other, and they also both contain skiplists either directly or indirectly, and contain slices of subscribers. The methods in those packages and the subscriber package are wholly responsible for writing non-options replies to the client, and they also perform the proper processing and actualization of the user requests.

All error messages are written to the user via the error response method, which properly formats them. The patcher package contains a struct, patch visitor, which implements the visitor interface, and has a method for applying patches to document bodies with it.