Jetzy VNEXT

# Overview

The Jetzy VNEXT backend will replace the legacy Jetzy .Net backend and is designed to address a number of standardization and performance issues with the legacy Jetzy Backend. The framework has been moved from .net to Elixir/Phoenix[[1]](#footnote-1) + MySQL & Redis[[2]](#footnote-2). Search and complex geo querying has been moved to Sphinx Search[[3]](#footnote-3). Elixir’s greater request concurrency and lower cost per virtual thread, normalized and optimized MySQL schema and the use of Sphinx and in memory redis caching will work together to address performance and scalability issues present in the legacy platform. Logic is moved from MSSQL stored procedures to more straight forward MySQL queries coupled with hierarchical caching of data and greater post processing performed in Elixir and Sphinx to simplify the path for increasing system capacity via horizontally scaling (spinning up additional web, redis and sphinx servers).

## Technology

The VNext Backend runs on a number of Noizu Labs libraries that have already been battle tested and capable of handling billions of requests plus processing calls per month. (upwards of 500 of millions of calls per day).

Along with Jetzy specific extensions.

<https://github.com/JetzyTechTeam/application-server/tree/master/jetzy_elixir> - Main Project  
  
<https://github.com/noizu/SimplePool> - Worker Pool Management Library (support for moving workers between nodes, health monitoring, built in scaffolding to ping workers, fetch worker state, kill workers, migrate workers, etc.)

<https://github.com/noizu/ElixirScaffolding> - Lower Level ORM wrappers and scaffolding. Reduce lines of code needed to hookup CRUD apis.Conversation between :mnesia (or other persistence stores – redis, mysql, datastore, riak, mssql) records and domain objects. Built in permission checks, automatic audit trails of database operations, etc. FastGlobal compiled module caching wrappers for spanning multiple nodes, etc.  
  
<https://github.com/noizu/RuleEngine> - Extensible database entity driven rule engine. Allows user defined runtime behavior changes via API. Such as conditions for granting users badges, data processing, etc.

<https://github.com/noizu/KitchenSink> - API Driven extensible CMS, Transactional/Templated emails with automated argument injection ( replace –{site.name}- with variable site name, etc.), Smart Tokens for url logins, restricted access to resources, time limited access to resources, etc.

<https://github.com/noizu/ElixirCore> - Low level libraries used by other projects. Context Entities. Module options with defaults, required params, etc.

## Database

### MySQL Database

MySQL database redesigned[[4]](#footnote-4) for faster querying. Normalized[[5]](#footnote-5) where appropriate, denormalized where performance gains can be seen with extended data removed from primary tables (vertical partitioning)[[6]](#footnote-6) to improve query times. For example extended user info removed from main user table. Joins set up to always run on integers instead of strings or guids. Cross table querying on universal ref ids with lookup tables to convert ref ids back to primary keys on source tables supported. Multi column indexing configured to reduce number of total indexes required while covering more common queries than multiple single column indexes. Schema versioning tracked in source control via Liquibase[[7]](#footnote-7).

MySQL ECTO ORM entities can be found here: <https://github.com/JetzyTechTeam/application-server/tree/master/jetzy_elixir/lib/jetzy/schema/mysql>

### MSSQL database

To support gradual migration from legacy MSSQL stored procedures to MySQL a patched erlang OTP version with a modified ODBC driver[[8]](#footnote-8) has been implemented to make access of GUID containing tables possible from within the ECTO ORM framework with out requiring bare queries or work arounds.

MSSQL ECTO ORM entities can be found here: <https://github.com/JetzyTechTeam/application-server/tree/master/jetzy_elixir/lib/jetzy/schema/mssql>

### Erlang Mnesia Database

The lightning fast Mnesia NoSQL database is used for tracking low volume data. Topic definitions, User records, etc. While dynamic content where is inappropriate to keep entirely in memory (like posts, comments) are stored in MySQL. For simplified data recovery definitive copy of entities are written to MySQL while reads are made against the Mnesia database functioning as a nearly project wide in memory cache of records.

### Redis

Temporary read caches of user content is commonly stored in Redis for fast retrieval with only slightly more overhead than Mnesia reads with a simpler cache invalidation/expiration flow.

@TODO mysql to ref and some other non expiring cache data is currently stored in Redis but for performance reasons will be switched to storage in mnesia tables to further improve performance and reduce points of failure.

### Sphinx

Sphinx is a full text search engine accessible from a MySQL compatible connection. It provides full and partial text searches, common verb prefix and suffix substition, geo spatial filtering, attribute filtering and a number of other useful features. MySQL and MSSQL full text searching performance is poor and queries may often need to span multiple tables making matters even worse. Sphinx allows the pre-generation of documents via query or xml pipe output thar are then indexed in a much faster and more queryable data structure than provided by full text search in mysql/mssql. Sphinx processes may be spanned across multiple servers making horizontal scale out more straight forward and weighting mechanisms may be adjusted making it great for generating items such as feeds.

#### Example Sphinx Document

Note I’ll be extending these slightly to include user interactions and comment tallies. So they can be used as metrics when querying and weighting queries.

|  |
| --- |
| <sphinx:document id="3">  <!-- Full text searchable field -->  <content>Teddy Bear Restaurant is open for delivery 12:00 - 24:00 7 days a week. </content>  <!—User ID by MySQL primary key instead of Firebase or MSSQL GUID -->  <author\_id>1</author\_id>  <!-- record type: text, image, video, link -->  <post\_type\_id>0</post\_type\_id>  <!—comma separated list of topic ids -->  <topics>4</topics>  <!— Location id – e.g. id=1 might be central park. -->  <location\_id>1</location\_id>  <moderation\_status\_id>0</moderation\_status\_id>  <visibility>0</visibility>  <!— private groups/friend groups etc. to control visibility – 0 default (no group) -->  <group/>  <!— visibility time range for time sensitive posts. -->  <scheduled\_from/>  <scheduled\_until/>  <!—Geo Graphic Region to optimize geo distance queries by also restricting queries by zone before performing distance calculations and sorting. -->  <zone>1</zone>  <latitude>11.538361</latitude>  <longitude>104.915963</longitude>  </sphinx:document> |

## Security, Authentication & Auditing

Apis calls may be authenticated using Firebase JWT[[9]](#footnote-9) tokens, self signed JWT tokens (for legacy user accounts) or legacy jetzy style authentication headers. Context tokens for each in coming call are created at the api endpoint and passed through the system as calls are made. Automated audit trails are written to mnesia for tracking record deletion, creation and updates, and individual requests may be traced as they flow through the system and nodes if needed to debug troublesome edge cases.  
  
All Elixir Scaffolding entities and entity repos support an extensible interface (protocol in elixir parlance) for verifying caller permission. Before updating a record, running a special background job, etc. the user permissions for the passed in context token can be checked to confirm the caller has the right to perform the action, query the record, etc.   
  
Multiple login credentials may be linked to a single underlying user account allowing for facebook, google, twitter, email/pass, and firebase login to the same underlying user account. <https://github.com/JetzyTechTeam/application-server/blob/master/jetzy_elixir/lib/jetzy/schema/mysql/user_credentials_table.ex>

## Caching

Generic libraries allow any entity to be persisted to or retrieved from Redis Cache with out requiring additional coding. Any entity that supports Json encoding and Decoding may be cached to Redis.

For result set caching the preference is to return sets of IDs in turn individually fetched from cache although in some cases entire data sets may be cached.

To support more advanced cache invalidation schemes we will be (future work) implementing a redis/elixir version of my fragmented key library which makes it straight forward to invalidate the cache of all entities related to a changed record with out needing to explicitly call redis delete to invalidate. <https://github.com/noizu/fragmented-keys>

# API Layer

## Restful

The VNext Api Endpoints endeavor to follow RestFul api best practices. <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>

Api naming conventions are standardized over the paths used by the legacy system.

## Versioning

The API interface is versioned by url: api-vnext.jetzyapp.com/api/1.0/comments , api-vnext.jetzyapp.com/api/2.0/comments etc. Major api structure changes or required data structures will use different api versions.

Unlike the legacy system all apis are served by the same underlying backend. Instead of releasing and maintaining multiple versions of the APIs we instead adhere to the open-closed principle[[10]](#footnote-10). Entities accepted or returned by the APIs may have new fields added with out changing the api version but removal of fields or changes to existing fields require new api versions to avoid breaking changes in existing API clients. Internally entities implement json encoders that can accept a version parameter to change the output (and input) json structure served or accepted from API.

## Authentication

Firebase or Self Signed Authorization headers may be used to sign api calls and is the preferred method. Api calls may additionally be signed using the same header authentication scheme used api calls on the old system.

See examples for more details.

## Entity Expansion & Paging

Some common framework libraries make it straight forward to add paging, and entity expansion to CRUD api calls. (note some of these behavior providers aren’t currently hooked up to the existing APIs due to some extra complexities from the mix of Mnesia and MySQL backed records, paging, results per page specifically aren’t hooked up on the current production version.).

### Query Parameters

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Results Per Page | rpp | Positive Integer |
| Note: Specify how many records to include in result set | | |

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Page Number | pg | Positive Integer |
| Note: Specify page of result sets to use. | | |

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Continuation Token | next | Continuation Identifier String |
| Note: Future Work Paging of results with out risk of shifting entries resulting in duplicated records. Not currently implemented. Api responses supporting this would include a continuation token along side results sets that can be feed in to the api call to obtain the next chunk of entries. | | |

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Expands all Refs | expand-all-refs | true|false |
| Note: Expand all foreign keys references by entities (with recursion protection). A post entity might return {owner: “ref.user.1234”} with out this option but with this option will include the nested user object. | | |

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Expands Refs | expand-refs | Comma separated list of entities. |
| Note: Selectively expand some refs. Expand location details for instance with out expanding user details. | | |

|  |  |  |
| --- | --- | --- |
| Name | Param | Values |
| Expand From | expand-from | [entity:cache|disk|sql|mysql|mssql] |
| Note: (Future Work: Not Yet Implemented – Defaults to from cache currently.) specify if entities should be fetched from cache (Redis or Mnesia), disk (Mnesia), directly from sql (MySQL/MSSQL or Mnesia) depending on entity type, or specifically mysql (MySQL) or mssql (MSSQL) for entities that support both | | |
| Example: ?expand-from=user:cache,location:disk,moment:mssql | | |

## Calls & Examples

### Authentication APIs

### ADMIN CRUD APIs

### CLIENT CRUD APIs

#### Active User

#### Other Calls

### QUERY APIs

# Chats

Details Pending. @See WebSockets & Phoenix Channels.

# Web Sockets| Phoenix Channels.

Phoenix out of the box provides support for push events using Phoenix Channels/Web Sockets. Client APIs are available in Java, Swift and Objective C and other languages. Coupled with elixirs superior concurrency handling Web Sockets allow for the backend to push events/notifications to mobile clients with out the need for the client to poll for new comments, new user interactions (likes/dislikes/hearts/etc.)

## Phoenix Channel Documentation

[Intro] <https://whatdidilearn.info/2018/03/04/using-channels-in-phoenix.html>

[Official Documentation] <https://hexdocs.pm/phoenix/channels.html>

## Phoenix Channel Clients

* JavaScript
  + [phoenix.js](https://github.com/phoenixframework/phoenix/blob/v1.4/assets/js/phoenix.js)
* Swift (iOS)
  + [SwiftPhoenix](https://github.com/davidstump/SwiftPhoenixClient)
* Java (Android)
  + [JavaPhoenixChannels](https://github.com/eoinsha/JavaPhoenixChannels)
* Kotlin (Android)
  + [JavaPhoenixClient](https://github.com/dsrees/JavaPhoenixClient)
* C#
  + [PhoenixSharp](https://github.com/Mazyod/PhoenixSharp)
* Elixir
  + [phoenix\_gen\_socket\_client](https://github.com/Aircloak/phoenix_gen_socket_client)

1. https://www.phoenixframework.org/ [↑](#footnote-ref-1)
2. https://redis.io/ [↑](#footnote-ref-2)
3. http://sphinxsearch.com/ [↑](#footnote-ref-3)
4. <https://github.com/JetzyTechTeam/application-server/tree/master/jetzy_elixir/liquibase> [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Database_normalization> [↑](#footnote-ref-5)
6. <http://cloudgirl.tech/data-partitioning-vertical-horizontal-hybrid-partitioning/> (Vertical Partitioning) [↑](#footnote-ref-6)
7. <https://www.liquibase.org/> [↑](#footnote-ref-7)
8. <https://github.com/noizu/otp> [↑](#footnote-ref-8)
9. https://jwt.io/introduction/ [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/Open%E2%80%93closed\_principle [↑](#footnote-ref-10)