LifeScale – Go/Gin Back-end

Environment Variables

Environment variables are stored in the *.env* file (example values can be found in the *.env.example* file).

The struct definition for the environment variables can be found in */env/env.go*. This file also contains the function that loads the environment variables.

Environment Variable Descriptions:

- **DATABASE_STRING –** The string used to connect to the MySQL database
- **HOST –** The hostname of this application
- **PORT –** The port that this application should be hosted on
- **PATH_PREFIX –** The start of all route urls (e.g. "/api/v1" in "/api/v1/user/register")
- **JWT_KEY –** The secret key used when generating JWTs
- **JWT_EXPIRATION_TIME_MIN –** How many minutes should a JWT stay valid?
- **EMAIL_PORT –** The email address to send any emails from
- **EMAIL_USERNAME –** The username for the SMTP email host
- **EMAIL_PASSWORD –** The password for the SMTP email host
- **EMAIL_HOST –** The SMTP email host
- **EMAIL_PORT –** The port to use for SMTP connections
- **TLS_CERT_PATH –** The SSL certificate for secure connections
- **TLS_KEY_PATH –** The SSL key for secure connections
- **CORS_ORIGIN –** The origin URL of the application's front-end

If any of the email-related variables are blank (or the EMAIL_PORT is 0), then the password-reset-request route will return an error, stating that the feature is not currently available.

If any of the TLS-related variables are blank, a non-secure HTTP router will be used instead of an HTTPS router.

If the CORS origin is not provided, all origins are allowed.

Migrating The Database

Once you have created a database for the application, and set the connection string environment variable, you will need to run the migrations.

You can do this by running the */migration/migration.go* file.

Custom Utilities

There are a number of utility functions that are used throughout this application.

Regular utilities are found in */custom_utils.* Test utilities are found in */custom_test_utils.*

IDResolver:

IDs in the front-end are strings (so this application can use a NoSQL database in the future), but we're using numeric IDs for now. This function takes pointers to both a string ID, and a numeric ID.

If given pointers to a numeric ID greater than 0, and an empty string ID, this will populate the string ID with the number's value.

Given pointers to a string ID, and a number ID equal to 0, this will populate the number with the numeric value of the string.

This can return an error if a string can't be converted to a number, or if both values are blank (empty string, and 0)

StringSanitiser:

Sanitises a string from any HTML tags, or other dangerous text (SQL injection has to be handled by the ORM, not by this)

Emailer:

An instance of the *Emailer* struct can be used to send emails from the application. You can (and should) create any new Emailer instances with the *MakeEmailer* function. This function will set the necessary properties, determine if the properties are all valid, and then set the *IsValid* property.

DeleteEntity (Test Utility):

Runs a DELETE request. If the response status code is not in the 200 range, then the returned error will be a string of the code (e.g. “404”).

GetFreshTestDatabase (Test Utility):

Generates a pointer to a *gorm.DB* instance, connected to a refreshed SQLite database (all data truncated out).

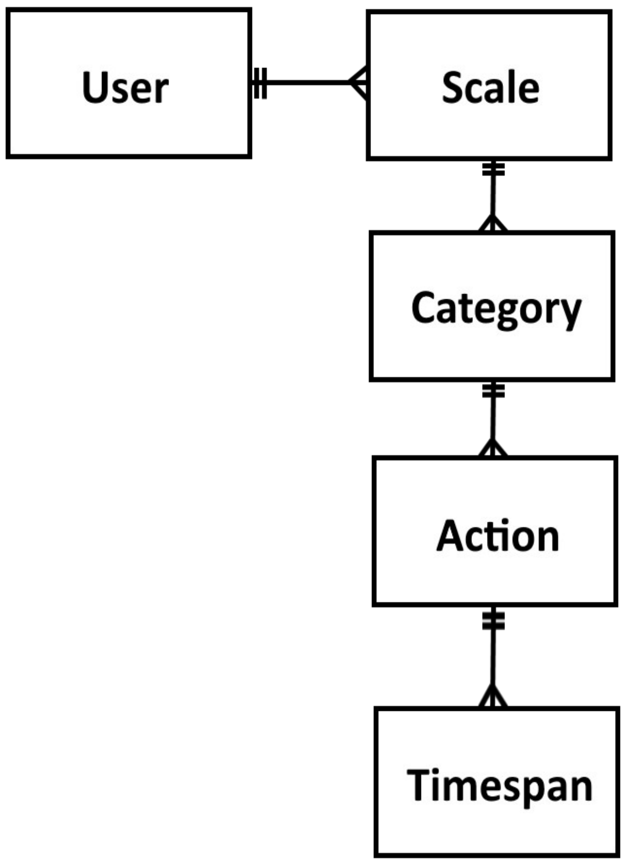
Running Tests

To run the tests for this application, you will need to run the below command. This will ensure that only one test is ran at a time (which is important, as the tests use the same SQLite database, and can therefore interfere with each other's data).

```
npm run test:unit
```

Data-Structure/Models

This application has a reasonably simple data structure:



Model Methods:

The models for the entities in the above diagram all have the following 4 methods:

- **Validate –** Will return an error if any of the data in the entity is not valid (in some cases, this will also check the inner-entities)
- **ValidateAuthorisation –** Will return an error if the authorised user is not the owner of this entity (doesn't check inner-entities)
- **ResolveID –** The front-end used string IDs (so a NoSQL database could be used in the future). This method will populate either the numeric or string ID, with the value from the other (also does the same for inner-entities)
- **Sanitise –** Sanitises the string values in this entity (also does the same for inner-entities)

Models:

- **PasswordChange –** Holds the data required to change a password
- **User**
- **Scale**
- **Category**

- **Action**
- **Timespan**

Services

The “service” structs are used to create/read/update/delete data in the database.

UserService:

- **Get –** Gets a user with the given ID (If the "id" paramater is an empty string, the "email" parameter will be used instead). If the "getRelatedScales" parameter is true, the user's scales will also be retrieved
- **Create –** Inserts a User (not including its child entities) into the database
- **Update –** Replaces a user's record in the database (using the ID property of the "user" parameter). If the "allowPasswordUpdate" parameter is true, this method will attempt to perform a password update

ScaleService:

- **Get –** Retrieves a Scale with the provided "id".

  The "timespanOption" parameter determines how many timespans are returned along with the scale. Unless the "NoTimespans" value is used, categories, actions and timespans can be included with the scale. If the "AllTimespans" value is used, all timespans under the scale are returned, within their categories/actions. If the "DisplayDayCountTimespans" value is used, only the timespans that fall within the scale's DisplayDayCount period will be included
- **Create –** Inserts a Scale (not including its child entities) into the database
- **Update –** Replaces a Scale's record in the database, using the ID property in the "newScaleData" parameter. If updating the DisplayDayCount value, all timespans that fall inside the new date limit will also get returned, with their categories/actions
- **Delete –** Deletes a Scale from the database, using the "scaleID" parameter

CategoryService:

- **Get –** Retrieves a Category with the provided "id"
- **Create –** Inserts a Category (not including its child entities) into the database
- **Update –** Replaces a Category's record in the database, using the ID property in the "newCategoryData" parameter
- **Delete –** Deletes a Category from the database, using the "categoryID" parameter

ActionService:

- **Get –** Retrieves an Action with the provided "id"
- **Create –** Inserts an Action (not including its child entities) into the database
- **Update –** Replaces an Action's record in the database, using the ID property in the "newActionData" parameter
- **Delete –** Deletes an Action from the database, using the "actionID" parameter

TimespanService:

- **Get –** Retrieves a Timespan with the provided "id"
- **Create –** Inserts a Timespan into the database
- **Delete –** Deletes a Timespan from the database, using the "timespanID" parameter

Routing

There is a “Setup” function in */routing/routing.go*. This function is used to setup the application's services and route handlers. It takes the following parameters:

- **r (*gin.Engine) –** A pointer to an instance of a Gin engine
- **db (*gorm.DB) –** A pointer to a database session, opened with the Gorm ORM
- **envVars (env.EnvVars) –** EnvVars holds the environment variables that are stored in the .env file

Routes

There is a variable in the environment variables that defines a prefix for every route listed below (e.g. “/api/v1”).

No-Auth-Required Endpoints:

- **/user (POST) –** This is used to register a new user. It returns the new user's record (the request body should match *models.User*)
- **/login (POST) –** This is used to log in to the application. It returns an instance of *handlers.JwtOutput*. One of the properties of the JwtOutput is the user's record, and this record will also contain all of the user's scales (but none of the entities within those). The request body should match *models.User* – but only requires email/password.
- **/passwordreset (POST) –** This is used to trigger a password reset. It will change the user's password to a random string, and then trigger an email (the request body should match *models.PasswordRequest*)

Auth-Required Endpoints:

- **/tokenrefresh (GET) –** This is used to get a fresh JWT, when you already have a valid one that is about to expire. If the current token is not set to expire in the next 30 seconds, this will respond with an error.
- **/user/changepassword (PUT) –** This is used to change the user's password (the request body should match *models.PasswordChange*)
- **/user (PUT) –** This is used to edit a user's record, but you cannot change a password through this route (the request should match *models.User*)

- **/scale/:scaleid (GET) –** This is used to retrieve a scale. This will also retrieve the entities within the scale (except in the case of timespans – only the timespans that fall within the scale's DisplayDayCount will be returned).
- **/scale (POST) –** This is used to create a new scale. This will return the new scale record (the request body should match *model.Scale*)
- **/scale/:scaleid (PUT) –** This is used to update a scale. If updating the DisplayDayCount value, all timespans that fall inside the new date limit will also get returned with their categories/actions (the request body should match *model.Scale*)
- **/scale/:scaleid (DELETE) –** This is used to delete a scale.

- **/scale/:scaleid/category (POST) –** This is used to create a category (the request body should match *model.Category*)
- **/scale/:scaleid/category/:categoryid (PUT) –** This is used to edit a category (the request body should match *model.Category*)
- **/scale/:scaleid/category/:categoryid (DELETE) –** This is used to delete a category.

- **/scale/:scaleid/category/:categoryid/action (POST) –** This is used to create an action (the request body should match *model.Action*)
- **/scale/:scaleid/category/:categoryid/action/:actionid (PUT) –** This is used to edit an action (the request body should match *model.Action*)
- **/scale/:scaleid/category/:categoryid/action/:actionid (DELETE) –** This is used to delete an action.

- **/scale/:scaleid/category/:categoryid/action/:actionid/timespan (POST) –** This is used to create a timespan (the request body should match *model.Timespan*)
- **/scale/:scaleid/category/:categoryid/action/:actionid/timespan/:timespanid (DELETE) –** This is used to delete a timespan.

Route Handlers

The “handler” structs provide route-handling/middleware methods, related to the different entities in the database.

AuthHandlerProvider:

- **CreateAuthMiddleware –** Creates and returns a middleware function. The created function checks that the request contains a valid JWT, for a valid user. This user is added to the context with the “auth-user” key.
- **SignInHandler –** Allows a user to sign in (returns a JWT, along with other data)
- **ChangePassword –** Used to change the authenticated user's password
- **PasswordResetRequestHandler –** Used to trigger a password change (to a random value), and an email containing this new password. If any of the email-related environment variables are not set, this will respond with an error, stating the feature is not currently available.
- **RefreshTokenHandler –** Used to refresh a JWT that is soon to expire. If the token is not set to expire in the next 30 seconds, this will return an error response

UserHandlerProvider:

- **RegistrationHandler –** Used to register a new User
- **UpdateHandler –** Used to update a User (this is suitable for a PUT request, but not a PATCH)

ScaleHandlerProvider:

- **RetrievalHandler –** Used to get a Scale from the database.

  If the "daycounttimespansonly" URL query parameter exists, and is true, only the timespans that fall within the scale's DisplayDayCount period will be included, with their categories/actions. If this parameter doesn't exist, all the timespans will be returned
- **CreateHandler –** Used to create a Scale in the database
- **UpdateHandler –** Used to update a Scale (this is suitable for a PUT request, but not a PATCH). If updating the DisplayDayCount value, all timespans that fall inside the new date limit will also get returned, with their categories/actions
- **DeleteHandler –** Used to delete a Scale from the database

CategoryHandlerProvider:

- **CreateHandler –** Used to create a Category in the database
- **UpdateHandler –** Used to update a Category (this is suitable for a PUT request, but not a PATCH)
- **DeleteHandler –** Used to delete a Category from the database

ActionHandlerProvider:

- **CreateHandler –** Used to create an Action in the database
- **UpdateHandler –** Used to update an Action (this is suitable for a PUT request, but not a PATCH)
- **DeleteHandler –** Used to delete an Action from the database

TimespanHandlerProvider:

- **CreateHandler –** Used to create a Timespan in the database
- **DeleteHandler –** Used to delete a Timespan from the database