Package util

The util package is primarily responsible for reading and validating an .env file to set up the server and database configuration. If future changes to the design require that stand-alone functions be written that are available to multiple parts of the back end, those functions will also be included in the util package.

*Note*: The capitalization of classes is not arbitrary. Go exports members which are capitalized, while members which are not are only available in that scope.

**dbCredentials**

**Description**: dbCredentials holds information the Server needs to connect to the Database, such as the Port, Password, and Database name.

**Data Members**: string Host, string Port, string User, string Password, string DBName

**Class Methods**: ConnectString(): string, loadDBEnv()

**serverConfig**

**Description**: serverConfig holds information the Server needs to run, such as the port it runs at and the cookie name that will be used.

**Data Members**: string Port, bool DoesLogAll, string JWTKey, string JWTCookieName

**Class Methods:** loadServerEnv()

**config**

**Description**: config holds both the Server and Database configurations and loads them from the .env file. It is exported as the singleton Config (with a capital C, for Go exports) and accessible around the program.

**Data Members**: dbCredentials DB, serverConfig Server

**Class Methods**: Load(string)

Functional

**dbCredentials**

**ConnectString**() : string

*Output*: ConnectString forms and returns a Postgres connection string from its members, in the form “host=%s, port=%s, user=%s, password=%s, dbname=%s, sslmode=disable”.

*Returns*: A string which can be used by the SQL package to connect to a Postgres instance.

**loadDBEnv**(),

*Output*: loadDbEnv makes calls to os.GetEnv for each .env variable needed to populate dbCredential’s members. It validates that each exists and loads their values into dbCredential’s members.

*Error*: If loadDBEnv fails to find any env variables it needs, an error message is printed to the console and the process shuts down. The server cannot connect to the database, and therefore cannot run, without being able to connect to the Database.

**config**

**Load**(string)

*Input*: A path to the .env file.

*Output*: The imported godotenv package function Load() is called with the path as a parameter. This makes the environment variables available. Then dbConfig.loadDBEnv() and serverConfig.loadServerEnv() are called to validate the configuration.

*Error*: If gotdotenv fails to find an env file an error message is printed to the console and the process shuts down. The server and database cannot run without a configuration.

**serverConfig**

**loadServerEnv**()

*Output*: loadServerEnv makes calls to os.GetEnv for each .env variable needed to populate serverConfig’s members. It validates that each exists and loads their values into serverConfig’s members.

*Error*: If loadServerEnv fails to find an env variable, an error message is printed to the console and the process shuts down. The server cannot run without a proper configuration.