kora 3.0.0 Technical Documentation

Database Structure

This section will go through the database structure of kora, giving an overview of the purpose for each table.

kora is built and tested using a minimum requirement of **MYSQL 5.7.20**, though it is possible that earlier versions of MYSQL 5.7 may be compatible. For best results, it is strongly encouraged to follow this minimum requirement.

User Tables

- **Users**: The main users table containing hashed pws, user data, preferences, and active/admin status
- **Password_resets**: Manages tokens for when a user requests a password reset
- **Dashboard_blocks**: The individual Blocks that make up a user's dashboard
- **Dashboard_sections**: Section groupings of Blocks for a user's dashboard
- **Project_custom**: Defines the order of Projects on a user's Custom tab
- **Form_custom**: Defines the order of Forms on a user's Custom tab

Permissions Tables

- **Tokens**: Kora Tokens, their title, and their permission sets
- **Project_token**: Linker table connecting Projects to individual tokens
- **Project_groups**: Project permission groups and their permission sets
- **Project_group_users**: Linker table connecting users to Project permissions groups
- **Form_groups**: Form permission groups and their permission sets
- **Form_group_users**: Linker table connecting users to Form permissions groups

Model Tables

- **Projects**: Project info, admin permission group, and the active state of the Project
- **Forms**: Form info, admin permission group, preset state, and layout of Form
    - **Layout**: Array with 2 parameters. The first is *pages*, which defines the name of each page and the layout of the Form fields in that page. The second is *fields* which contains each Field and all properties/settings that define that Field.

Form/Field Modifier Tables

- **Associations**: Association between the data Form and the searching Form respectively for use in Associator Fields
- **Field_value_presets**: Collection of all the Field Value Presets and their shared status, used to populate record data with commonly used values

Record Tables

- **Records_{*form_id*}**: The records for each Form, including high level Record information, the Record owner, and columns for each Field in the Form. These columns, and their database type, are defined by the Field models
- **{*Combo_flid*}_{*form_id*}**: These tables represent all data for every Record in a particular Combo List Field, for a single Form. The data in the Records table for this Field will have an array of ids that point to the rows in this table
- **Reverse_associator_cache**: Represents a cached data set containing the connections between Associator Fields (in all kora Forms) to the Records they point at
- **Record_presets**: Records that are a preset template to help create other Records
- **Revisions**: History of a Record, including a copy of old and new versions for restoration

Progress Tables

- **Failed_records**: Logs any errors for failed records during record importing
- **Exodus_overall_progress**: Tracks the progress of projects migrated from kora 2.* to kora 3.0.0
- **Exodus_partial_progress**: Tracks the per-scheme progress of projects migrated from kora 2.* to kora 3.0.0
- **Jobs**: Laravel table for tracking queued Jobs for kora emailing process
- **Failed_jobs**: Laravel table for documenting failed Jobs

MISC Tables

- **Global_cache**: Used to track recent searches for a user in the global search box
- **Migrations**: Laravel table for managing/tracking DB table installs
- **Scripts**: List of installed scripts when updating kora
- **Versions**: Tracks the current version of kora being used

Backend Code Structure

This section will go through the folder structure of kora's code, highlighting particular classes and features along the way. Unless it is useful to the understanding of kora, features of Laravel itself will be ignored.

kora is built using Laravel 5.6 with a minimum requirement of **PHP 7.1.3**

.env

- The configuration file for an individual installation of kora. This is created during the installation process. The default structure of this file can be viewed in '.env.example'. NOTE: While it is possible in the code to access these variables via the env() command, it is not recommended. All env variables should be accessed using the Laravel config() function. See 'CONFIG' section below to learn more.

App

- Eloquent Model classes in Laravel represent the Model portion of an MVC framework. Every major object in kora is represented in one of these php classes.
- Before explaining the Models, here are the files that don't represent objects in kora:
    - **helpers.php** - This is a simple php file with helper classes. Typically, these are functions that are used frequently in the View files. Since View files require the full namespace of a function to be used, this file masks those calls into simpler functions for cleanliness.
    - **Search.php** - This file handles keyword search, for a Form, at a fundamental level. There are also special functions to help handle the use of special characters in a search.
- **KoraFields/BaseField.php** - Base abstract class for all Field types in kora. It contains generic functions for all Fields, as well as several abstract functions to implement a Field type. This documentation will cover the purposes of each abstract function to help with the understanding of how a Field is constructed. See _________ for more information on how to build a new Field.
    - **getFieldOptionsView** - Returns the view file for the Field's options page.
    - **getAdvancedFieldOptionsView** - Returns the view file for the Field's advance creation form.
    - **getAdvancedSearchInputView** - Returns the view file for the Field's advance search input.
    - **getFieldInputView** - Returns the view file for the Field's Record Data input.
    - **getFieldDisplayView** - Returns the view file for the Field's Record Data display.

    - **addDatabaseColumn** - Creates the DB column in the Records table associated with this Field
    - **getDefaultOptions** - Returns the default options value for a new Record Field.
    - **updateOptions** - Updates the options for a particular Field.
    - **validateField** - Validates the submitted Field data against the options for that Field.
    - **processRecordData** - Prepares the data for entry into the DB.
    - **processRevisionData** - Prepares the data for display in a Record Revision.
    - **processImportData** - Prepares the imported data for entry into a Record.
    - **processImportDataXML** - Prepares the imported XML data for entry into a Record.
    - **processImportDataCSV** - Prepares the imported CSV data for entry into a Record.
    - **processDisplayData** - Prepares the data for display in a Record.
    - **processXMLData** - Prepares the data for display in a XML formatted Record.
    - **processLegacyData** - Prepares the data for display in the koraSearch legacy functions.
    - **massAssignRecordField** - Assigns data to a particular Field for several Records.
    - **massAssignSubsetRecordField** - Assigns data to a particular Field for a subset of Records.
    - **keywordSearchTyped** - Performs a keyword search on this Field and returns any results.
    - **setRestfulAdvSearch** - Updates the request for an API search to mimic the advanced search structure.
    - **advancedSearchTyped** - Performs an advanced search on this field and returns any results.
- **KoraFields/FileTypeField.php -** Second abstract class that extends the BaseField. This class adds additional base functionality for File Type Fields.
- **KoraFields/{*type_name*}.php**
    - TextField, RichTextField, BooleanField, IntegerField, FloatField, ListField, MultiSelectListField, GeneratedListField, ComboListField, DateField, DateTimeField, HistoricalDateField, DocumentsField, GalleryField, PlaylistField, VideoField, ModelField, GeolocatorField, AssociatorField
    - These classes implement BaseField (or FileTypeField) and add additional functionality related to their Field type.
- Other Models - These are brief descriptions of what each model file represents. This documentation also lists some of the unique functions available in these models that may be of interest.
    - **Association.php** - Represents associations between two Forms. This connection allows one Form to associate to records in the other.

- **FieldValuePreset.php** - Represents a Field Value Preset, which is a set of data that can be reused for various Record Fields.
    - **STOCKPRESETS** - The built-in presets that are installed upon installation.
- **Form.php** - Represents a Form, which is the structure that defines a set of data.
    - **validFieldTypes/ validFilterFields/ validAssocFields/ fieldModelMap/ jsonFields/ enumFields** - Several static arrays that define and configure various Field types in kora.
    - **getRecordsForExport*** - A set of functions for getting Records from a Form. Each one outputs in a different format.
- **FormGroup.php** - Represents the permission groups that allow access and modification of a Form, its structure, and its data.
- **Project.php** - Represents a Project, which is a high-level project in Kora 3 that contains 1 to many Forms.
- **ProjectGroup.php** - Represents the permission groups that allow access and modification of a Project and its settings.
- **Record.php** - Represents an object of data within a Form, including the data for each Field in the Form, and metadata such as owner and timestamps. This class has a custom class _construct to assign the model its Form Records table.
    - **isKIDPattern** - Important validation feature to determine if a provided Kora ID (projectID-formID-recordID) is properly formatted.
- **RecordPreset.php** - Represents a template Record that can be used to create new Records without starting from scratch.
- **Revision.php** - Represents a snapshot of Record, through the life of the Record. This includes the creation, modification, and deletion of a Record, including the reversal of any of these states.
- **Script.php** - Represents update scripts that have been executed by a particular Kora 3 installation.
- **Token.php** - Represents the token strings that authenticate access to Projects via the Restful API.
- **User.php** - Represents a User, and their information, within the installation.
    - **verifyRegisterRecaptcha**/ **finishRegistration**/ **sendPasswordResetNotification**/ **refineLoginCredentials** - These particular functions are modifications to Laravel's authenticate system. Every time composer is updated for kora to support a new version of Laravel, the authentication system is updated and reset. NOTE: When this happens, these 4 function calls need to be re-implemented into the Laravel authentication system.
- **Version.php** - Represents the current version of a particular kora installation.

App/Commands

- Commands (now called Jobs) are Laravel classes for firing off queued workers on the server allowing us to run time consuming functions at the system level.
- The main functionality in kora is sending emails since they may take time, and kora requires no success response for email routines. The MailCommand class is used as an abstract function for defining and grouping emails by type.
- Each email requires an operation name, and an array of options to configure a particular email.

App/Console

- Laravel has a powerful command line editing tool called php artisan for building projects. Not only that, but we can add functionality to that tool by creating Console Command classes in the sub 'app/Console/Commands' folder.
- NOTE: Any newly created Commands must be registered in the $commands array for 'app/Console/Kernel.php'.
- These are the custom CLI functions that have been added to kora including their parameters:
    - **ConvertField.php** (***php artisan kora:convert-field***) – This script converts an existing Field into a new type (i.e. convert a List Field to a Generated List Field). Note that data can be lost when converting to certain Field types, where the old data is incompatible with the new Field. In all of these cases, the old Record data is maintained in the database for retrieval.
        - **fid** - Form ID
        - **flid** - Field ID to generate thumbnails for
        - **type** - The new field type
    - **DisableRollbacks.php** (***php artisan kora:disable-rollbacks***) – This script turns off the rollback option for every existing Record Revision.
    - **ExodusScript.php** (***php artisan kora:exodus***) - This script allows us to run the Exodus migration tool from command line. It was created to help migrate larger kora 2 projects since we can avoid any timeout/memory limitations that exist in PHP.
        - **dbhost** - Hostname of the kora 2 DB
        - **dbuser** - User of the kora 2 DB
        - **dbname** - Database name of the kora 2 DB
        - **dbpass** - Password of the kora 2 DB
        - **project** - Project ID of the project to migrate from kora 2
        - **fileDir** - Local directory path pointing to the kora 2 files directory (or copy of said files directory

    - **FileUrlFix.php** (***php artisan kora:file-url-fix***) - This script takes the given domain of the installation, and browses record data for all File Type Fields, in order to update/repair the file URLs. This is useful when the wrong URL was assigned, or the hosted domain of the kora installation becomes different than the URL referenced in the record data.
        - **domain** – URL to the home page of kora (i.e. https://www.kora.com)
    - **GenerateThumbs.php** (***php artisan kora:generate-thumbs***) - When using a URL to access a Record file, an optional thumbnail size can be specified. This thumbnail is generated as needed. This script allows automatic creation of thumbnails in a Form, in order to decrease load times of thumbnails externally.
        - **fid** - Form ID
        - **flid** - Field ID to generate thumbnails for
        - **size** - Size of thumbnail to generate (i.e. 100x100)
    - **InstallKora.php** (***php artisan kora:install***) - The installation process for kora requires the user to access kora through the browser to finish installation. To avoid that, or to automate the install process, this script allows you finish the installation via command line. See Github for more information on installation.
    - **RebuildRecordPresets.php** (***php artisan kora: record-preset-fix***) – This script rebuilds all Record Presets, where the original Record still exists.
    - **RecordFileZipExport.php** (***php artisan kora:record-file-zip***) - Generates a zip file of all record files for a particular form. Outputs server location of generated zip.
        - **fid** - Form ID
    - **ReverseAssocCache.php** (***php artisan kora:assoc-cache***) – This script builds and caches all Record associators in order to make reverse connections between the Records.
    - **UpdateKora.php** (***php artisan kora:update***) - The update process for kora requires the user to access kora through the browser to finish the update. To avoid that, or to automate the update process, this script allows you finish the installation via command line.

App/FieldHelpers

- This folder contains special classes that add functionality to kora.
- **gPoint.php** - This class allows us to convert Latitude & Longitude coordinates into addresses, and vice versa. When we create Locations in a Geolocator field, when one or the other coordinate type is provided, we convert that respectively to the other type. NOTE: gPoint is licensed under the terms of the GNU General Public License as published by the Free Software Foundation.

- **koraSearch.php -** This file primarily allows us to replicate the KORA_Search and KORA_Clause classes from kora 2. In order for a project to use this, it must be transferred to kora 3 via the Exodus migration tool. It basically converts the kora 2 search into a kora 3 API call, and uses CURL PHP to call kora. The returned data is formatted the same as expected from kora 2. The file can be included into a site's php code just as in Kora 2.
- **UploadHelper.php** - This class is used in our File Type Fields to help with uploading and sorting multiple files into records, while tracking progress. NOTE: jQuery File Upload Plugin is licensed under the MIT License.

App/Http

- These files handle all Http protocols and methods in Laravel.
- **Kernel.php -** This class defines the entire middleware stack for Http. Any request that runs in Laravel will hit these classes before executing. The kora API can even have its own defined stack, separate from the main web application stack. There is also some custom route Middleware that can be registered here.
- **routes.php** - Defines all URI routes for Laravel and their Http request type. They are grouped by 'web' and 'api' middleware groups. This affects which stack is used in the 'app/Http/Kernel.php' file.

App/Http/Controllers

- Controller classes in Laravel represent the Controller portion of an MVC framework. A bulk of the work for kora is done in these files. Listed are each kora related controller and some of the unique functions available in these controllers.
- **Auth/RegisterController.php** - This controller handles the registration of new Users as well as their validation and creation. By default this controller uses a trait to provide this functionality without requiring any additional code.
  - **validator**/ **create -** These functions have been modified to fit the structure of a User in kora.
- **Auth/UserController.php -** This controller handles User based functions, including modifying User information and permissions. It also has several constants that define User permission sets.
- **AdminController.php** - This controller handles administrative functions for kora.
- **AdvancedSearchController.php -** This controller handles advanced searches for a Form.
- **AssociationController.php** - This controller handles management of Form Associations for use in Associator Fields.
- **AssociationSearchController.php** - This controller handles Record searching for individual Associator Fields in Record creation.
- **DashboardController.php** - This controller handles the User dashboard system.

- **ExodusController.php** - This controller handles the Exodus migration of kora 2 data to kora 3.
    - **startExodus** - Handles the migration of User, Tokens, Projects and Forms (called Schemes in kora 2).
    - **finishExodus** - Handles the building of associations between Records in the migrated data.
- **ExodusHelperController.php** - Helper functions for migrating Controls and Record data.
    - ***EXODUS_CONVERSION_SIZE*** - Defines the amount of Association Controls written to one file to prevent arrays from getting too big.
    - **migrateControlsAndRecords** - Handles the migration of Fields (called Controls in kora 2) and Records.
- **ExportController.php** - This controller handles the export process of kora structures and data.
    - **exportProject** - Exports a Project and its structure. Exported in JSON format called .kProj
    - **exportForm** - Exports a Form and its structure. Exported in JSON format called .kForm
    - **exportRecords** - Exports a Form's Record data. Exported in either JSON or XML formats.
- **FallbackController.php** - This controller handles routing of unknown Routes to a custom 404 page.
- **FieldAjaxController.php** - This controller handles ajax requests within kora/Laravel to access a Route for specific field related functions. This class merely calls the requested Field function, and then returns its result. We do this so Field classes can maintain their functions, but since routing in Laravel is used to call Controllers instead of Models for best practice, we use this Controller as the go between.
- **FieldController.php** - This controller handles the creation and management of Fields in kora.
- **FieldValuePresetController.php** - This controller handles preset values that can be used in various Field options.
- **FormController.php** - This controller handles creation and manipulation of Form models.
- **FormGroupController.php** - This controller handles groups that manage User permissions for Forms.
- **FormSearchController.php** - This controller handles Form searches in kora.
- **ImportController.php** - This controller handles import of Project/Form structures as well as Record data.
- **ImportMultiFormController.php** - This controller handles the import process for importing Records into multiple Forms.
- **InstallController.php** - This controller handles finishing the installation process, and the modification on non-critical .env values.
    - **INSTALLED_VERSION** - Current version of kora.
    - **DIRECTORIES** - List of directories created during installation.

- **PageController.php** - This controller handles the page layout of Forms.
- **ProjectController.php** - This controller handles Projects within kora.
- **ProjectGroupController.php** - This controller handle permission groups for Projects.
- **ProjectSearchController.php** - This controller handles search for a Project and global search.
- **RecordController.php** - This controller handles Record creation and manipulation.
- **RecordPresetController.php** - This controller handles creation and management of Record Presets.
- **RestfulController.php** - This controller handles API requests to kora. To learn more about the API, see the kora API Documentation.
- **RevisionController.php** - This controller handles Record revisions to preserve history of a Record.
- **TokenController.php** - This controller handles creation and management of API authentication Tokens.
- **UpdateController.php** - This controller handles version management of kora.
    - **UPDATE_PAGE** - The URL to check for the latest kora updates.
- **WelcomeController.php** - This controller renders the home pages for the application and navigate the User based on their status, or whether or not they are logged in.

App/Http/Middleware

- Route Middleware classes for kora and Laravel. These are essentially rules that must be true for a particular Http route to be accessed (i.e. user must be logged in to access the Projects page).
- NOTE: Any newly created Middleware must be registered in the $routeMiddleware array for 'app/Http/Kernel.php'.
- These are the custom Middleware classes that have been added to kora:
    - **IsActive.php** - Requires the current user to be using an Active account.
    - **IsAdmin.php** - Requires the current user to be a kora system administrator.
    - **IsInstalled.php** - Requires the kora installation to be installed.

App/Http/Requests

- HTML forms typically have certain validation rules that need to be met. For example, when registering a new user, a username is required and must be unique. Validation Requests in Laravel are used to define those rules for a set of fields in a specific HTML form.
- These are the custom Request classes that have been added to Kora 3:
    - **BlockRequest.php** - Defines the requirements for a Dashboard Block in Kora
    - **FieldRequest.php** - Defines the requirements for a Field in Kora
    - **FormRequest.php** - Defines the requirements for a Form in Kora
    - **ProjectRequest.php** - Defines the requirements for a Project in Kora

    - **UserRequest.php** - A modified version of the Laravel default. Defines the requirements for a User in Kora

Config

- These Configuration files contain associative arrays of configurations for Laravel and kora. Most of these are predefined and fixed for use in kora. Some of them are used to retrieve setting specific values for an individual installation (i.e. APP_ENV, DB_HOST, RECAPTCHA_PRIVATE_KEY). These values are retrieved by the env() function which pulls the value from the global '.env' file. Some env() calls have default values in case the '.env' file is unreadable at runtime. NOTE: As mentioned in the '.ENV' section, env variables should be accessed by using config(), to call the config variable that accesses it.

Database/Migrations

- Migration classes in Laravel define database tables and their structure. They include functions for installing and removing the table, allowing for the addition of other custom PHP code to execute, such as adding default table values or even adding/deleting groups of tables together.
    - **CreateRecordsTable** - This migration has several rules for modifying Field columns and supports several types of DB columns. There are also functions for managing Combo List Field tables.

Public

- Holds public web files, javascript files and resources
- **.htaccess** - For kora, apart from Laravel usage, this file has optional rules for RewriteBase (URL usage) and php_value (values related to file upload limits).
- assets - Contains several files and assets for the Kora 3 site. Here are the relevant subfolders:
    - **css** - Contains the generated app.css file from the SCSS files.
    - **css/vender** - Vender css files for included third party plugins.
    - **fonts** - Font files for front-end of kora.
    - **fonts/kora-icons** - Custom font icon files for kora. An included 'readme.md' file explains how to add new icons to the files.
    - **images** - Different front end image artifacts used in kora.
    - **javascripts** - Javascript classes for each front-end page and object.
    - **javascripts/production** - Just like CSS, Javascript files are compiled into a single minified Javascript file for performance improvements.
    - **logos** - Kora 3 logo files.

Resources/Assets

- Contains all the SCSS files for generating the front-end CSS in kora.
- NOTE: In order to build/re-build the production 'app.css' file, a compiler is required (i.e. Compass) and must be installed manually.
- The primary .scss file is 'scss/app.scss'.

Resources/Views

- View classes in Laravel represent the View portion of an MVC framework. They are built using Blade PHP template files.
- Each folder, and included templates, represent one page or major object within the application. Any partial templates or reusable views are kept in the 'partials' directory.
- **app.blade.php** - This is the main template file for the kora application. All views in kora are built off this template.

Scripts

- These files are used in the update process for kora. The primary update process is to pull in the latest changes from Github. However, there may be changes to kora (i.e. adding/deleting tables, modifying tables or table data, etc) that must occur after the system files are updated. These scripts will run those changes. The database itself will keep track of which scripts have been executed.

Storage/App

- All created files for system use or data use in kora exists in this folder.
- NOTE: It is highly recommended to create regular backups of this data along with your database.
- These are the main folders within 'storage/app':
  - **exodus/kidConversions** - Conversion files for kora 2 KIDs to kora 3 KIDs used in Exodus migrations to maintain Record Associations.
  - **exodus/assocData** - The Record Association data from an Exodus migration to be converted.
  - **exports** - Record exports for a Form are compiled here before being downloaded by the user.
  - **files** - The Record files for data in any File Type Field. They are organized by the following pattern: 'files/{Project ID}/{Form ID}/{Record ID}/{the files}
  - **profiles** - Stores profile pictures for all Users.

    - **tmpFiles -** For most file operations (i.e. Record manipulation, Record importing, etc), we store files temporarily here if they need to exist before their final destination.

Storage/Logs

- Error and information Logs for Laravel and kora are stored here. Generally, this a great place to debug errors. All errors are recorded here, as well as information from certain processes in kora.

Prepping kora for Update

This section will explain how to prep a kora update for going live. Complete this step once you have pushed all changes to the **development** branch.

1. Update **INSTALLED_VERSION** constant in App/Http/Controllers/**InstallController.php** to the new version number.
2. Build script in **scripts** folder
   a. Script name should be a variant of the version number to preserve order (i.e. version 3.11.4 would have the script, 031104_update.php)
   b. Script should first check if current version is less than the scripts version
   c. Build the script that will run inside that version check (i.e. these can include anything from DB changes, to data manipulation. Basically, anything that can't be accomplished by the user running git pull).
   d. Inside the version check, we should also update the **versions** table in the DB.
   e. Outside the version check, store the script as hasRun in the **scripts** DB table if it isn't already.
3. Push these changes to the **development** branch, and then push all **development** branch changes into the **master** branch.
4. Add the update information to **http://matrix-msu.github.io/kora/**. NOTE: this actually triggers the update.
   a. The site exists at the **gh-pages** branch of the kora repository.
   b. Copy and remove the current JSON array from the body of **index.html** and add it to the **history.json** file.
   c. Add the new JSON update structure to **index.html** (see example below).
   d. Push changes to the **gh-pages** branch.
   e. NOTE: Once pushed, the site will automatically update, but may take several minutes to reflect these changes.

```
{
        "version": "8.19.90",
        "notes": "Changes some things about kora",
        "features": [
                "Added virtual reality component",
                "All data can be viewed in a Tesla"
        ],
        "bugs": [
                "Fixed an issue where audio playback needed more cowbell"
        ]
}
```