Tracksz Development & Standards

Author: Jaymes H. Sorbel, jim@chrislands.com, (618) 910-8968

Date: March 21, 2020

Purpose: The purpose of this document is to provide guidance on the standards used when developing the Tracksz system. Ideally, any developer should look at a the code and see the same standard being used throughout.

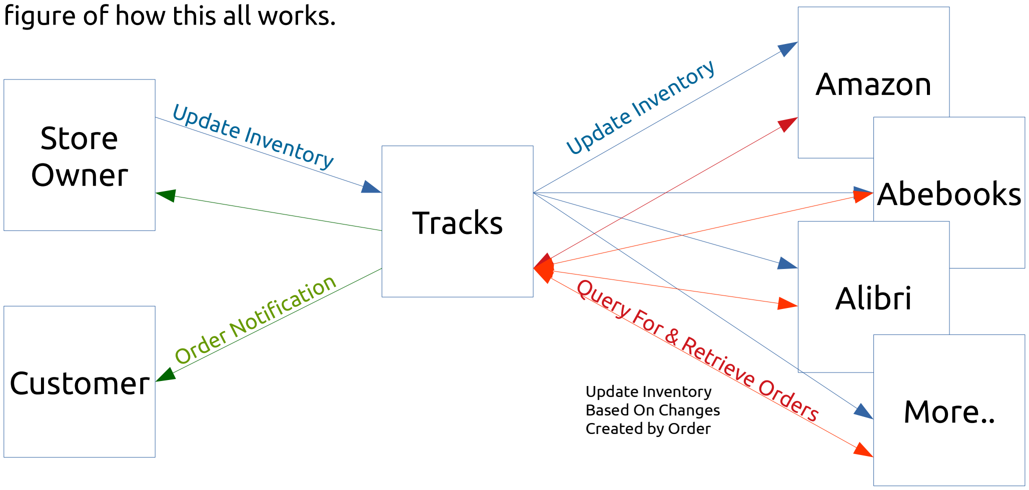
This document is not meant to be all encompassing. If there are any questions, you can either look at code previously written, example code in various areas of the framework, or by contacting the author directly.

What is Tracksz: Tracksz is on Online Inventory and Order Management system and is a service provided by ChrisLands Inc. The Tracksz user is a seller of products, a "Store Owner", that uses various online marketplaces to list their products. These marketplaces include but are not limited to Amazon.com, Barnes & Noble, Abebooks.com, Valorebooks.com, Alibris.com, ChrisLands.com and more.

Using the Tracksz system allows the user, "Store Owner", to manage their inventory online at Tracksz.com by uploading an inventory file or manually adding, editing or deleting inventory. Tracksz then takes any inventory changes and, through API calls, updates each Marketplace that the store is configured to support.

At scheduled intervals Tracksz, using API calls to each Marketplace the store supports, queries those Marketplaces for new orders. If an order is found, depending on settings, notifies the "Store Owner" and the customer about the order and about any order status change. If the order creates a change to the inventory, Tracksz updates all other Marketplaces with changes to the inventory.

Here is a simplistic figure of how this all works.



The Framework: After working with other frameworks, it was determined that some are bloated with many features not used and and they also do not provide adequate performance for enterprise level applications. Additionally, these frameworks are always being updated making it nearly impossible to keep your application on the current version.

The current Tracksz framework was put together by the author to only include what is absolutely needed. Nothing more. The goal is simplicity and performance. Each of the current packages included in the framework were researched extensively for performance, simplicity, and ratings. Also, since we control the framework, we control when the framework is updated and how.

Changes to the framework, packages added, packages removed, packages changed for different packages, and so on is allowed. However, any changes must be cleared and approved by Tracksz and there must be a well thought out reason to make any change. Any reason to make changes must include simplicity, performance, and rating arguments.

Packages required to support 3rd party APIs (order retrieval, inventory uploads, shipping, etc), can be added as need.

It is the intent of Tracksz to keep the framework lean, simple, and performance oriented.

Folder Structure: The folder structure of the framework follows other frameworks' folder structures. If you have worked with any framework you recognize the structure.

The current folder structure is:

```
-- .notes: Notes about Development, framework, database migration & design, things to remember and so on.
-- app:    The application code
   -- Application.php[1]: Loads services for Data Injection, Middleware and loads appropriate route file.
   -- Console:  Commands run from the command line or scheduled cron job commands
   -- Controllers: Logic Controllers
      --  ** Sub folders for logical separation of Controllers.[2] **
   -- Library: Helpful utilities such as Config.php and Email.php
   -- Middleware: Middleware to perform during get and post operations.  Used by routes
   -- Models: Models for database tables.  One for each table.[3]
   -- Services: Services used for Dependency Injections and instantiated by Application.php
      -- Routes: Route files[4].  Array in Application.php must be updated to find any new route files.
-- bootstrap: startup.php – start configurations and launches Application.php, .env file for secure data.
-- config: Configuration files. Base database sql file.
   -- migrations: Any database changes needs a migration file. Read .notes/Migrate.txt
-- public: Webroot of application. On the server it is not in this location.
-- resources:
   -- language: Language Files
   -- views: Contains the front end views, mostly HTML files.
-- vendor: Composer installed packages.
-- composer.json
-- compser.lock
```

[1] Must update $routeProviders array with any new route files in Application.php to load properly. Framework initiation.
public/index.php → bootstrap/startup (configure some features) → app/Application (services, middleware, routes) → selected route file

[2] Be logical when creating new files and sub-folders. Group like files and sub-folders under a logical parent folder. For instance all things pertaining to the Member Account should be under Account, everything for Inventory place under inventory, everything for orders place under orders, etc.

[3] Models should be "heavy" models. All SQL should be in a model and not a controller or a view.

[4] Route files should be created for each Parent level menu item in the control panel, such as Account, Store, Marketplaces, Inventory, Orders, etc. Anything under that Parent Menu link should be contained in this route file. Although there is no limit to how many route definitions can be defined in a single route file, it is preferable to keep the route file small. It is also easier to search for routes you need to work on.

Developers should not leave files in the framework needed for the their local development environment. If you add files to the Framework to support your personal local development environment, add those to the .gitignore file so they are not included in the repository.

.env should never be in the repository. It contains secure data. It is is advised to help keep secure data secure.

PHP Coding Standards: Tracksz will follow the PHP Standard Recommendations, PSR-1 and PSR-2, listed at PHP-Fig.org. Some key points:

Class Declarations:

```
<?php declare(strict_types = 1);

namespace Vendor\Model;


use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;


class Foo
{
}
```

* Class names are upper camel Case – TheClassName
?> not used at end of classes

Class file names are UpperCamelCase and must match the class in the file – FooController.php = class FooController

Functions:

```
public function fooBarBaz($arg1, &$arg2, $arg3 = [])
{
    // method body
}
```

* function names camelCase – theFunctionName
* function variables should imply what it is used for and may use _ (underscore) - $product_item

if, foreach, for, etc.

```
if ($a === $b) {
   bar();
} elseif ($a > $b) {
   $foo->bar($arg1);
} else {
   BazClass::bar($arg2, $arg3);
}
```

PSR-1: https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md
PSR-2: https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

Use Path\To\Class Area: This is where you alias the classes used within the current class. It provides a means to autoload when the class is first instantiated.

Often, when we create a new class, we copy an old class to get the correct structure. Ensure you clean out the classes you are not using in that current class. Or only add a "use Path\To\Class;" statement when you actually need that class.

Some editors identify what "use" classes that are not used in a the current class by showing them in a different color, such as gray. For instance, PHP Storm shows this:

```
use App\Library\Views;
use App\Models\Account\Store;
use App\Models\Account\Member;
use Delight\Auth\Auth;
use Delight\Cookie\Cookie;
use Delight\Cookie\Session;
use Laminas\Diactoros\ServerRequest;
use PDO;
```

The gray "use class" are not used in the current class. Delete them if you are done with the file. Some editors automatically add the "use path\to\class" at the head of the class when you enter $new_class = New Class(); If yours does, delete all of the current use when you create a new class and let your editor help.

Best Practices for Use:
- Do not import unused classes
- Place all use at first
- Avoid using the same alias for different classes
- Limit the number of use expression

https://www.exakat.io/6-good-practices-for-use/

Functions: In general functions should accomplish one task, should not extend for 100s of lines of code, and always return some kind of result.

Database Naming Convention, and Migration: While researching best practices, there were "many" to choose from. These are the ones Tracksz has chosen.

Database Name Convention.
- Singular names for tables: Product
- Singular names for columns: Title
- Schema name for tables prefix (E.g.: SchemeName.TableName) if needed for joins between databases.
- Pascal casing (a.k.a. upper camel case) for Database, Table and Column Names: ProductTitle

Database:
- Default Character Set = utf8mb4
- Default Collation = utf8mb4_unicode_ci

Whenever you add a table, modify a table, or even add a new database (for whatever reason) create a migration file and save into into the config/migration folder. Review the file in .notes/Migrate.txt

Database Access – Use SQL: Tracksz standard for accessing data from the database is to use PDO with raw SQL statements. It is unlikely that we'll ever switch from using MySQL therefore an abstraction layer may only add complexity. Using straight SQL is faster for performance than almost any other method. This is a performance issue.

Always Bind your variables using PDO::PARAM_* For example.

```
 $query  = 'SELECT Id, FullName, Address1, Address2, City, PostCode, ZoneId, CountryId,  MemberId, Company ';
$query .= 'FROM Address WHERE MemberId = :MemberId';
$stmt = $this->db->prepare($query);
$stmt->bindParam('MemberId', $form['MemberId'], PDO::PARAM_INT);
$stmt->execute();
```

SQL/Database Performance: Database performance often determines the performance of the application. Here are some things to consider when using SQL and creating Table Indexes.

1. Unless you need the entire table row, only select the fields you need. If you are going to use most or all of the columns in a row, then this:

```
SELECT * FROM Test
```

In most cases you do not need all the columns in a row. If you only need a few columns, it is better to do:

```
SELECT column1, column2, column5, column7 FROM Test
```

Selecting only the fields you need creates a smaller dataset size and is usually much faster.

2. If you do any logical, Join, ORDER BY, or GROUP BY operation on a column it must be indexed.

a. Select Statements. For example, Column1 is Indexed, Column2 is not indexed

```
SELECT Column1 FROM Test WHERE Column1 > 10;
```

Is faster than

```
SELECT Column2 FROM Test WHERE Column2 > 10;
```

b. Join Statements. t.Column1 and p.ProductId is indexed. t.Column2 is not.

```
SELECT t.Column1, p.ProductId FROM Test t, Product p
WHERE t.Column1 = p.ProductId
```

Is Faster than

```
SELECT t.Column2, p.ProductId FROM Test t, Product p
WHERE t.Column2 = p.ProductId
```

* of course the column selection in these examples can be any column you want. Only the operation performed determines the speed and those columns should be index.

3. Use Limit when you only need a subset of data rows for a single page. Often used for Pagination.

```
SELECT Column1 FROM Test WHERE Column1 > 10
LIMIT 20, 20
* This will get 20 rows from row 20
```

Is Faster than

```
SELECT Column1 FROM Test WHERE Column1 > 10
```

Plus in the second one you may get many rows you won't even need.

4. It is often faster to do multiple little queries than a Join, especially with big tables.

```
SELECT Column1 FROM Test
WHERE Column1 > 20;
* Build an array or string with the results, usually.
```

```
SELECT ProductID, ProductTitle FROM Product
WHERE ProductID IN (*array OR string from result above*)
```

Could be faster than:

```
SELECT t.Column1, p.ProductId FROM Test t, Product p
WHERE t.Column1 = p.ProductId
AND t.Column1 > 20;
```

5. When doing a join, if possible, list the smallest table and use the logical operator that returns the smallest subset of data first.

```
SELECT t.Column1, p.ProductId FROM Test t, Product p
WHERE t.Column1 > 20
AND t.Column1 = p.ProductId;
```

Maybe Faster than:

```
SELECT t.Column1, p.ProductId FROM Product p, Test t
WHERE t.Column1 = p.ProductId
AND t.Column1 > 20;
```

6. Always define your Joins. Never do a full join. It maybe better to read this.
https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php

7. Never use “%some string to find%” when searching for a row or rows from a table. Unless things have changed, if you index a field then use “%some string to find%” mySQL does not use the index. It does a row by row search.

If you use “some string to find%” (no leading %) it will use the Index.

8. Creating Multi Field Keys: Unless things have changed, if you create a key using two fields, such as:

```
KEY `keyName` (`Field1`, `Field2`);
```

The index is both fields. However, mysql also indexes the first field of the key. If you search on the first field it will use an index. If you search on the second field, it WILL NOT use an index. If you need to search on the second field by itself it must have a separate key defined.

Form Field Names and Database Table Column Names: To make writing SQL statements and preparing data for inserts, updates, and deletes easier, Tracksz input fields names use the same name as the table column name. Using the same form field name as the table column name, removes some of the need to manipulate data in the Model to get the right data in the right form to use for the SQL statement. For Example:

Table Column Name: **ProductTitle**

Form Input Field Name:

```
<input type="text" title="<?=_('Enter Store\'s Zip/Post Code')?>" class="form-control" id="ProductTitle"
name="ProductTitle" placeholder="12345" data-parsley-required-message="<?=_('Please insert store\'s zip/post
code.')?>" data-parsley-group="fieldset01" required data-parsley-length="[4, 10]" <?php if (isset($store['PostCode']))
echo ' value="' . $store['PostCode'] .'"' ?>>
```

Model Function:

```
public function updateProductTitle($form)
{
     $query  = 'UPDATE  Product SET ProductTitle = :product_title WHERE Id = :Id';
     $stmt = $this->db->prepare($query);
     $stmt->bindParam('product_title', $form['ProductTitle’], PDO::PARAM_INT);
     $stmt->bindParam('product_title', $form['Id’], PDO::PARAM_INT);
     $stmt→execute();
}
```

* Notice we didn’t have to put the $form[‘ProductTitle’] into another variable, manipulate it, or do anything with it. We used the field right from the form.

Duplicate Code: In general, duplicate code is not desirable. It makes maintenance of the code base harder. If you find you are writing the same code in multiple locations figure out a way to write it once and use it all locations. Every effort should be made not to use duplicate code.

Reused Code: It is common for developers to have a library of code they used before. This is great as it speeds up development. However, reused code needs to be edited to fit the current framework standards.

Sanitize, Validate, and Escape User Data: Data security is incredibly important. As such Tracksz will use multiple techniques.

1. Use Parsleyjs to validate a form before it is submitted.

```
        <form method="post" action="/inventory/defaults" id="inventory-defaults" data-parsley-validate="">
                <input type="text" title="<?=_('Enter Store\'s Zip/Post Code')?>" class="form-control" id="ProductTitle"
name="ProductTitle" placeholder="12345" data-parsley-required-message="<?=_('Please insert store\'s zip/post
code.')?>" data-parsley-group="fieldset01" required data-parsley-length="[4, 10]" <?php if
(isset($store['PostCode'])) echo ' value="' . $store['PostCode'] .'"' ?>>
        </form>
```

2. Validate in the Controller Function after a form has been successfully submitted.

```
   $form = $request->getParsedBody();
    unset($form['__token']); // remove CSRF token or PDO bind fails, too many arguments, Need to do every time.

    // Sanitize and Validate
    // - Sanitize First to Remove "bad" input, if any
    // - Validate Second, if Sanitize empties a field due to "bad" data that is required then Validate will catch it.

    $validate = new ValidateSanitize(); // use App\Library\ValidateSanitize\ValidateSanitize;
    $form = $validate->sanitize($form); // only trims & sanitizes strings (other filters available)

    $validate->validation_rules(array(
       'FullName'    => 'required|alpha_space|max_len,100|min_len,6',
       'Email'       => 'required|valid_email',
       'Telephone'   => 'required|integer',
       'Markets'     => 'alpha_space',
       'Features'    => 'alpha_space'
    ));

    // Add filters for non-strings (integers, float, emails, etc) ALWAYS Trim
    $validate->filter_rules(array(
       'Email'    => 'trim|sanitize_email',
       'Telephone'=> 'trim|sanitize_numbers',
    ));

    $validated = $validate->run($form);
    // For rest of function use $validated instead of $form as it is sanitized and validated.
   //  $validate is false if not validated

    if ($validated === false) {
       $validated['alert'] = 'Sorry, we could not add your contact message.  Please try again.';
       $validated['alert_type'] = 'danger';
       $this->view->flash($validated);
       return $this->view->redirect('/#contact');
    }
```

3. SQL Statements use →bind().

```
   $stmt->bindParam('MemberId', $form['MemberId'], PDO::PARAM_INT);
```

4. Escape output originally inputted by a user through form input. Tracksz uses PHP Plates for templates. It has a built in escape function. `<?=$this->e($name)?>`

Parsley: https://parsleyjs.org/
Validation Library: https://github.com/Wixel/GUMP
Plates Syntax: https://platesphp.com/v3/templates/syntax/
Plates Escape: https://platesphp.com/v3/templates/escaping/

Language: In order to prepare for various languages, Tracksz uses gettext. This is easy to use and easy to translate once the site is complete.

When outputting static text, text created during development (not from user input), use this syntax:

```
In Views:      <?=_(‘some text to output’)?>

In Controllers:

$validated['alert'] = _('That email address has already been submitted.  Thank You for your submission!');
```

Gettext’s _ (underscore) function is recognized by the PHP gettext extension.

Gettext: https://www.gnu.org/software/gettext/

The American Disability Act: There are many parts to the American Disability Act for web designers to consider. At present, Tracksz is focusing on just one part.

1. Include an appropriate title=”” attribute in all anchor links and form input fields.

```
<input type="text" title="<?=_('Enter Store\'s Zip/Post Code')?>" class="form-control" id="ProductTitle"
name="ProductTitle" placeholder="12345" data-parsley-required-message="<?=_('Please insert store\'s zip/post
code.')?>" data-parsley-group="fieldset01" required data-parsley-length="[4, 10]" <?php if (isset($store['PostCode']))
echo ' value="' . $store['PostCode'] .'"' ?>>

<a href="/account/panel" class="menu-link" title="<?=_('Tracksz Account Dashboard')?>">
```

2. In images <img...> use both title and the alt attributes.

```
<img src="/assets/images/logo3.png" title="Logo for Tracksz.com Multimarket Inventory and Order
Management Service" alt="Logo for Tracksz.com Multimarket Inventory and Order Management Service">
```

Documentation: Many times in code you’ll find a lack of documentation. Even the author of this document is not great at documentation. At a minimum, functions should be documented using PHPDOC standards for example

```
/**
 * editCardTable - Edit a card that was entered twice
 *          duplicated at gateway
 *
 * @param $form - original form name, address, zip
 * @param $card_values - values from stripe api results
 * @return view - member/card - failure or member/cards success
 */
```

Other Libraries: Other libraries available in app\LIbrary

1. Config.php – Loads the config/settings.php in bootstrap/startup.php. Available throughout the code base. Only static class in framework.

2. Email.php - wrapper for PHPMailer.com.

```
$mailer = new Email();
$message['html']  = $this→view→make('emails/changeemail');  // prepare html template
 $message['plain'] = $this→view→make('emails/plain/changeemail'); // prepare plain template
 $mailer->sendEmail($form['email'], Config::get('company_name'),
       _('Tracksz Change Email Request'), $message, ['url' => $url]);
```

PHPMailer: https://github.com/PHPMailer/PHPMailer

3. Encryption.php – used for all encryption and decryption. Uses sodium encryption library.

```
$encrypt = new Encryption();
$card_rows[$encrypt->safeDecrypt($card['GatewayPaymentId'], $key)] = $card;

$this->encrypt = new Encryption();
$gatewayId = $this->encrypt->safeDecrypt($card['GatewayPaymentId'], $this→member_key);
```

* each member has their own key for encryption and decryption

PHP Sodium: https://www.php.net/manual/en/book.sodium.php

4. Image.php – used for image resizing, correcting orientation and saving the image file.

5. Pageinate.php – pagination

6. Utils.php – a few functions used in various places in the code. Sanitizefilename, curlGet, curlPost, etc.

7. Views.php – this is used to create the view services. Tracksz uses PHP Plates templates system for the front end presentation layer.

Validate Your Completed HTML output.

Go to: https://validator.w3.org/

The easiest way is to view source, copy the source, and paste int the validator.

8. Zebra_Session.php – database session wrapper, started in bootstrap/startup.php

References: In order displayed in the document.

PHP PSR-1:
https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md

PHP PSR-2:
https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

6 Good Practices for “use” in PHP:
https://www.exakat.io/6-good-practices-for-use/

Full Join Article:
https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php

Parsley Form Validation:
https://parsleyjs.org/

Validation Library:
https://github.com/Wixel/GUMP

PHP Plates Template Engine:
https://platesphp.com/

PHP Plates Syntax:
https://platesphp.com/v3/templates/syntax/

PHP Plates Escape:
https://platesphp.com/v3/templates/escaping/

PHP League Containter for Data Injection:
https://container.thephpleague.com/3.x/

GetText:
https://www.gnu.org/software/gettext/

PHPMailer:
https://github.com/PHPMailer/PHPMailer

PHP Sodium:
https://www.php.net/manual/en/book.sodium.php

W3C Validator:
https://validator.w3.org/