MIP Dashboard
Developer Documentation

MIP Developer Documentation

Media Impact Project Dashboard
The Media Impact Project (MIP) Dashboard is an open-source dashboard used together with the MIP Data Repository as part of the Media Impact Project's Media Impact Measurement System.

The MIP Dashboard and the following documentation were created for use on the Google Cloud Platform. Some additional work may be required to run the MIP Dashboard on other platforms.

Dashboard Overview

The MIP Dashboard application is a PHP based application that runs on Google App Engine using the popular Laravel Framework for modular, agile feature development. The MIP Dashboard uses a Google Cloud based MYSQL database for application performance and cost reasons. The Dashboard includes automated ETL processes that run queries to extract the necessary data from the data source and enter it into the MYSQL database to be available for the Dashboard to display. Currently, there the dashboard has connectors for pulling data from Google BigQuery and Google Analytics into MYSQL and displaying on the Dashboard.

State of Project
The MIP Dashboard is freely available for use under the Apache 2.0 open source license. The Media Impact Project is a division of the Norman Lear Center at the USC Annenberg School for Communication and Journalism. It is funded by a grant from the Bill & Melinda Gates Foundation, with additional funding from the John S. and James L. Knight Foundation and the Open Society Foundations.

License

MIP Dashboard and other documentation is published under the Apache Software License, the popular open source license (Apache 2.0 license).

“ASL, which is widely used in the open-source software community and has been approved by the Open Source Initiative, is a permissive license that is conducive to commercial development and proprietary redistribution. Code that is distributed

under the ASL and other permissive licenses can be integrated into closed-source proprietary products and redistributed under a broad variety of other terms."

- **http://arstechnica.com/uncategorized/2007/11/why-google-chose-the-apache-software-license-over-gplv2/**

Prepare Your Local Environment

1. Install PHP5.x, python 2.7, gcloud sdk, git, ssh, etc.
2. Install Composer (https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx)
3. Run command 'composer install'
4. Create new .env file based on the.env.local for your local development env. you need modify some settings special database settings. e.g. DB_HOST DB_DATABASE DB_USERNAME DB_PASSWORD
5. The first time you run the project, you will need to run `php artisan migrate` to initialize the database. Then add your user account details into /database/seeds/DatabaseSeeder.php e.g.

```
php
DB::table('users')->insert([
    'name' => 'john.admin',
    'email' => 'john.admin@mediaimpactproject.org',
    'password' => bcrypt('admin123!@#'),
]);

$user = App\Models\User::where('email', '=', 'john.admin@mediaimpactproject.org')->first();
$role = App\Models\Role::where('name', '=', 'SuperUser')->first();
```

```
$user->roles()->attach($role);
$user->save();
```

then run `php artisan db:seed` to add your account and role
6. Change the email settings in the .env file. You can use `MAIL_DRIVER=log` for debug emails. do not need real email smtp service. you can check the log file in path `/storage/logs/laravel.log` to see what you send content in email.

7. Install library `composer install`

8. We used PHP intl see need run `brew install php55-intl` to install this extensions. need restart php-fpm or restart your computer. The php.ini for google app engine already adds these extensions, so don't worry about this when you deploy to GAE.

PHP Larvel and the .env file

About Larvel See more [here](https://laravel.com/docs/5.1)

.env file for production
> APP_ENV=production
> APP_DEBUG=false
> APP_KEY= PUT YOUR APP KEY HERE
> APP_HOST_DOMAIN=http://yourhosturldomain.org
CHANGE THIS LINE TO REFLECT YOUR ACTUAL HOST URL/DOMAIN
> APP_LOG=syslog
>
> DB_CONNECTION=cloudsql
> DB_SOCKET=/cloudsql/project-name:us-central1:sql-database-name
> DB_HOST=
> DB_DATABASE=
> DB_USERNAME=
> DB_PASSWORD=

These need to be set to reflect the cloud sql data for your google cloud SQL database.

>
> MAIL_DRIVER=gae
>
> FILESYSTEM=gae
> SESSION_DRIVER=memcached
> CACHE_DRIVER=memcached
> QUEUE_DRIVER=gae

>
> CACHE_SERVICES_FILE=false
> CACHE_CONFIG_FILE=false
> CACHE_ROUTES_FILE=false
> CACHE_COMPILED_VIEWS=false
>
> REDIS_HOST=127.0.0.1
> REDIS_PASSWORD=
> REDIS_PORT=6379

APP_KEY is very important as the user password is based on this.
APP_HOST_DOMAIN is used in email template. Change this in order to set the reset password email and invite email links to have the correct domain.
DB_SOCKET or DB_HOST are database address. For Google App Engine use DB_SOCKET. For local dev, you will likely use DB_HOST.

Background Services
The Background Services are separate Google App Engine applications written in python and deployed separately from the main app. They are run as cron jobs. The source code for these are in the folder `background`

1. Edit mysqlClient.py for your local mysql passwd setting
2. Add: passwd='passwordgoeshere') to mysqlClient.py file in the Background folder
3. Add the your .p12 file to the background folder (mip-analytics.p12). This is your Google Analytics key file.
4. Run ‘gcloud.cmd auth login’ to generate the necessary files for using the google cloud services & api

Then run the background debug to install/test/check for dependencies
1. open the python interpreter
2. run the command 'Import debug'
3. run the command 'debug.run_last_week()'

Because we used some third party libraries, you will need to install these libraries into folder `lib` for Mac. There have a issue for Homebrew [see more here](http://stackoverflow.com/questions/24257803/distutilsoptionerror-must-supply-either-home-or-prefix-exec-prefix-not-both)

Create a pip config file `~/.pydistutils.cfg` with these content
```
[install]
prefix=
```

```

Then run the pip install command `pip install -t lib -r requirements.txt`

See more [here](https://cloud.google.com/appengine/docs/python/tools/using-libraries-python-27)

**How to run Background services backlog or history for a particular client locally**

1. Background services should be run on the master branch if you are running locally.
2. Change the settings flag to only enable one client to sync.
3. open the python interpreter
4. run the command 'Import debug'
5. Run the command ‘debug.run_history()
6. Check your mysql database for the results.

**Notes about the Background service:**
+ we get big query hive sql from mysql table settings.
+ the insert big query result to mysql sql is saved in query.py
+ for performance issue we only saved top 100 stories in mysql others are saved as csv file on google storage. the storage path in config.py that means for stage and production you must changed this value before deploy
+ history.py are quick code for run job.py history method
+ storiesClear.py this is a quick script code for get top 100 stories and save to another table.

**Deploying to Google Cloud**

This application currently runs on the Google Cloud Platform. If you wish to run this on a different platform, you will need to make adjustments accordingly.

**Google Cloud Platform Setup**
1. Create a project.
2. Make sure you have enabled billing for this project.
3. Enable all necessary APIs.
4. create and save your .p12/json key for project. NOTE: Do not commit your keys.
5. create an SQL database for the project. set the root password.

**Before Deploying**
1. Adjust SQL and BigQuery settings  (stage, live, dev, etc.)
2. Adjust yaml settings for deployment (stage, live, dev, etc.). NOTE: remember that version names can only have one dash in them. stick to version names without dashes for ease of use.
```

3. run the command to deploy the project 'appcfg.py -A project-name update ./'

Command Line
+ `php artisan clear-compiled`
+ `php artisan optimize`
+ `php artisan config:cache`
+ `php artisan route:cache`
+ `php artisan view:clear`
These commands is been used to optimize the framework for better performance. The compiled file are stored to `bootstrap/cache/` folder. Need to manual run these command sometimes when found something wrong.

Cache
You can use you local cache server like Memcached or Redis as you like. see more [here](https://laravel.com/docs/5.1/cache#configuration)
these are the cache setting exmaple in the `.env` file
```
CACHE_DRIVER=redis
SESSION_DRIVER=redis

REDIS_HOST=127.0.0.1
REDIS_PORT=6379
```
some useful Redis cache command
+ Connect to Redis `redis-cli` then you will see `127.0.0.1:6379>`
+ Test connection `ping` then you will see `PONG` if connection success
+ List all keys `keys *`
+ Another way to list all keys `scan 0`
+ Show value `get key` replace the key to what you need checked cache

Menu
Middleware `app/Http/Middleware/ParseCurrentControllerAndAction` is for get current Controller and Action will be used in menu view
BaseController `app/Controllers/AuthenticatedBaseController` register 2 middleware: routeInfo and auth. Any need authenticate Controller can inherit from this base controller

Client Info
Middleware `app/Http/Middleware/InjectClientInfo` is for inject user client info into request and view. you can use $client in View and use $request->input('client') to get current user client info
For SuperAdmin will auto inject the first client. and allow SuperAdmin select client from view.
For safe used these variable better check before use it in view
```php
@if(isset($client))
```

```

**Other Tools**
Google Analytics Query Tools [https://ga-dev-tools.appspot.com/query-explorer/](https://ga-dev-tools.appspot.com/query-explorer/)

**GCloud Library**
there was 2 kind of google libaray. one is gogole api libaray another is this. for now seem this library is not good enough but we have more options. keep eyes on these.
https://googlecloudplatform.github.io
(PHP)[https://googlecloudplatform.github.io/google-cloud-php/#/]
(Python)[https://googlecloudplatform.github.io/gcloud-python/#/]

**A Quick Walk Through Files in the Application**
app > Helpers > FormatterHelper.php
We can use this helper in View. We need the following line in the view
> @inject('formatter', 'App\Helpers\FormatterHelper')
for now, most page use ajax to get data. To format dates, we used moment.js. To format numbers, we use new Intl ECMAScript Internationalization API, which Safari does not support. So we added a shim script
<script src="//cdn.polyfill.io/v2/polyfill.min.js?features=Intl.~locale.en"></script>
This script will check brower have support Intl and add fallbacks.

app > Http > Controllers > DataController.php
This controller is our most important controller. One area of improvement would be to split to multiple controllers for each of the data-tabs: data-users data-stories data-newsletter data-quality controller

app > Http > Middleware > InjectClientInfo.php
We use this to add current user client info to $request

app > Http > Middleware > ParseCurrentControllerAndAction.php
This is used in the side menu to get active menu

app > Models
These are Eloquent Model, an ORM solution [see more here](https://laravel.com/docs/5.1/eloquent)

App > Providers > AuthServiceProvider.php
Register Admin Role and SuperAdmin Role here

background
```

These are Python background cron jobs and related code.

background > analyticsClient.py
This calls GA data helper function

background > bigQueryClient.py
This calls big query helper function. One area to improve is where the weekly stories data has too many records for the cron job. e.g. SCPR data stories. some week have 50000+ records. might make the cron job crash, so this should be refactored.

background > cloudStorage.py
access google cloud storage helper functions

background > job.py
cron job main functions are in this folder

background > main.py
cron job web access entry

background > mySqlClient.py
mysql database access helper function

background > query.py
mysql sql config e.g. insert into mysql sql for clients

public > scripts > main.js
Most of the common javascript is here. The rest is in the View

vendor
3rd party javascript libraries are here

resources > views > data
We split these views by clients. To add a new client, add a new folder and views. One area of improvement would be to add a default view and check if a special view exists otherwise use standard view. This logic improvement would need be upgraded within DataController.php

resources > views > errors
We can add more error pages here e.g. not authed error page. 404 not found page etc.

resources > views > layouts > main.blade.php
This is our main template. like index.html. All the rest of the css and js files are here.
This is where we could add a GA tracking code.

resources > views > layouts > offCanvas.blade.php
This is where the side menu is.

A Note on Controllers
You can easy inherit AuthenticatedBaseController this base controller already added user check
```php
public function __construct()
{

    $this->middleware('auth');

}
```
or you can add this middleware by yourself
other useful middleware are
```php
    $this->middleware('routeInfo');
    $this->middleware('clientInfo');
```
These middleware are all locationed at path app/Http/Middleware/
1. routeInfo are used by the menu to get current active menu
2. clientInfo are used to get current user client info from $request

A Few Note on Views
We have 2 sections
@section('content') put html dom node here
@section('script') put this page javascript code here

We already add main.js in public/scripts/ for some common scripts
+ datepicker init.
+ datatable reload
+ download button event
+ tracking

For datetable reload, we used an event to trigger this common javascript event. For data page(users stories newsletter) we needed add current page datatable instance to global windows variable `ReportDataTable` the property name is the table id. you can see it in this code
```html
<div class="panel">
    <div class="top-bar">
        @include('widgets.daterange', ['min_date' => $min_date, 'max_date' => $max_date])
    </div>
```

```
    <table id="dataNewsLetter" class="report tiny hover">
    </table>
```
The main.js already adds the datepicker change event (meaning, the client changed displayed date), this will trigger parent panel custom event `change.daterange` and there is code to handle this event in main.js
```javascript
$(document).on('change.daterange', '.panel', function () {...});
```
this code is to get the table id and try to get datatable instance in gloabl variable `ReportDataTable` and try to reload data

For datepicker to set init date range, the main.js will check the hidden input name = 'defaultDateRange' html element value. you can use template widgets.daterange to render these html or add by yourself. See more: resources/views/widgets/daterange.blade.php
For stories, we only allow the client to select one week so we do not use a common daterange.