SETUP INSTRUCTIONS

1. Clone the repository git clone
2. Composer install
3. mv .env.example .env
4. php artisan cache:clear
5. composer dump-autoload
6. php artisan key:generate
7. npm install
8. npm run dev

To make use of the caching, set up your cache & session driver to redis.
Sample .env file below

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:LEiVcOuebagvp1lmi2j0oFmsU6H1HJL/tPhGWIcT/Qs=
APP_DEBUG=true
APP_URL=http://overflow-assessment.site

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=overflow-tippla
DB_USERNAME=homestead
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=redis
FILESYSTEM_DRIVER=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=redis
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_CLIENT=predis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

POPULATE SAMPLE DATA

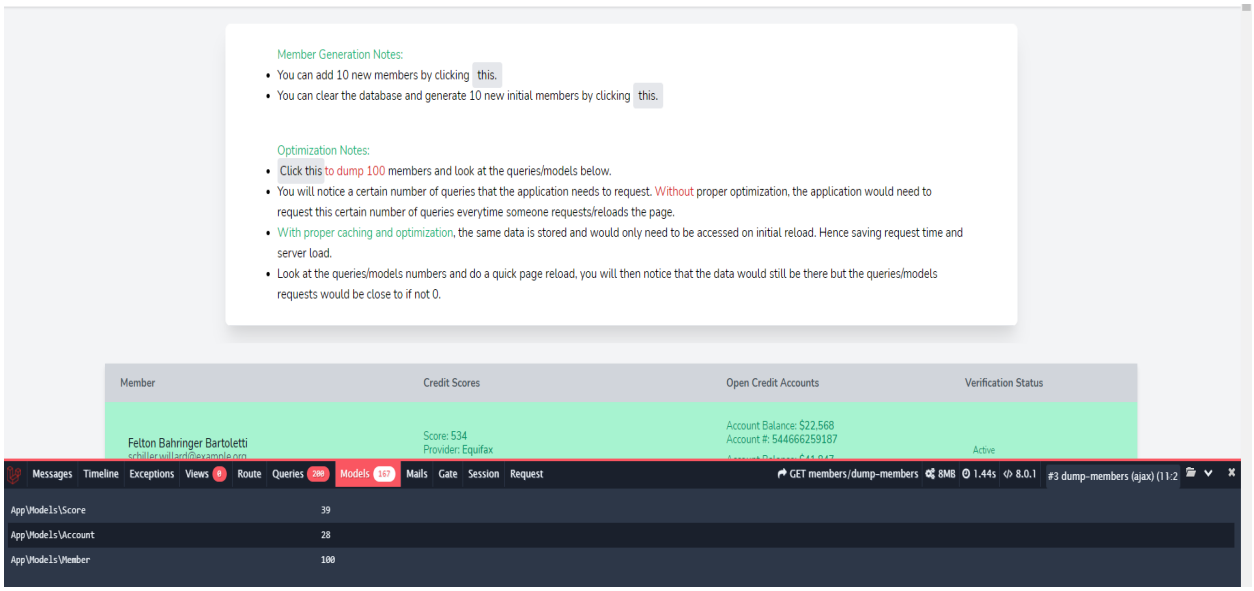
9. php artisan migrate:fresh --seed
10. Admin Credentials :
    User : admin@overflow-assessment.site
    Password : 12345678

CACHING NOTES

See : https://laravel.com/docs/8.x/cache

Debugbar is used to monitor the applications performance

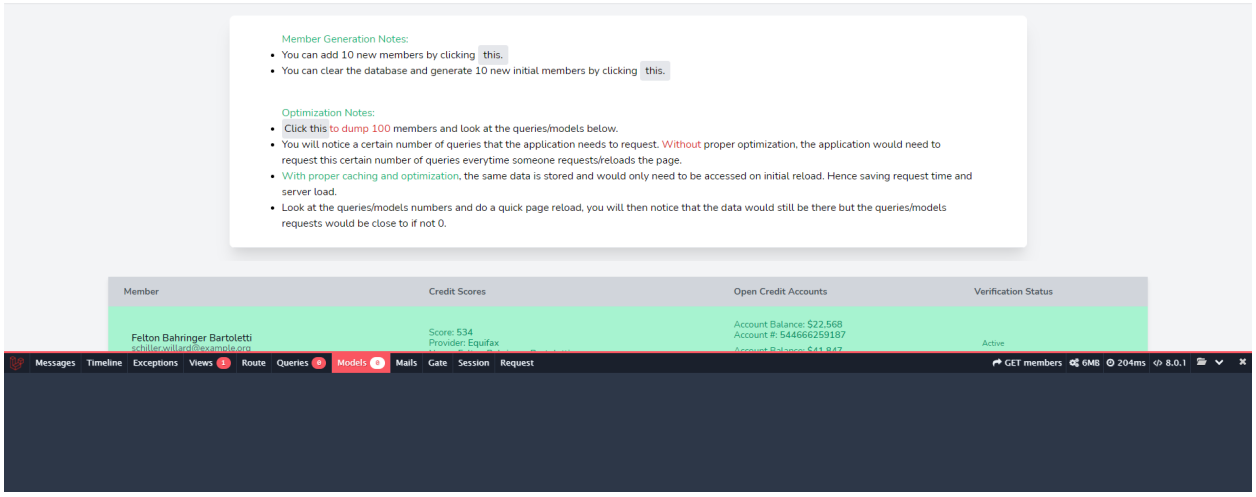
Figure 1 : Shows the amount of initial queries and requests the application has.



These are the amount of queries and requests that the application has every time it loads the page.

Without caching, the application would need to request this certain number of queries everytime someone requests/reloads the page.

Figure 2 : Shows the amount of queries and requests the application has with data stored in cache.



The same data is stored and would only need to be accessed on initial reload. Hence saving request time and server load.

Look at the queries/models numbers and do a quick page reload, you will then notice that the data would still be there but the queries/models requests would be close to if not 0.

Note : In this case, Auth User is also stored on the Cache

API RESOURCE NOTES
See : https://laravel.com/docs/8.x/eloquent-resources

Since it was mentioned that the application would mostly be dealing with credits and accounts, it would be ideal to have a secured way of sending and receiving data, one way of doing so would be through resources.

Implementing resources allows you to ***easily and efficiently*** manipulate data that you will be exposing.

See figures below :

Figure 3 : Shows the default response that Laravel sends, as you can see, it shows all the data attributes/columns**.** This is not ideal because it exposes all of the data and sends out information that is not part of the request.

With API resource you can reduce the data that you will send and turn it into something like this.

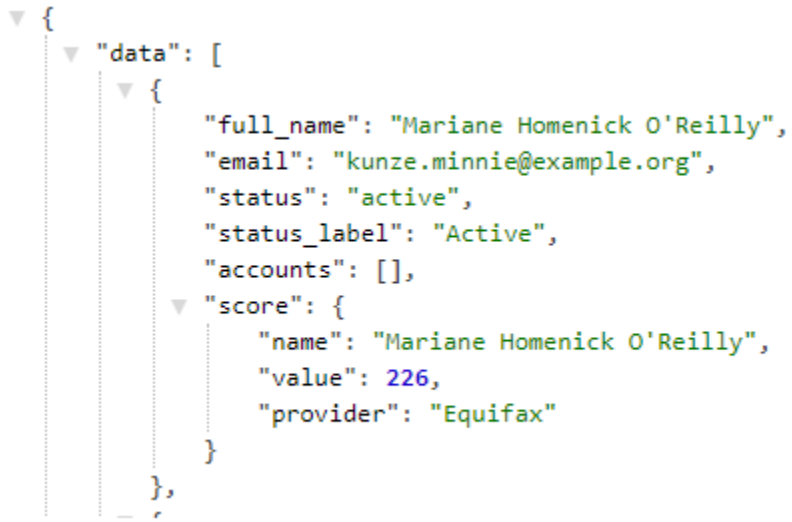


Figure 4 : Data sent is now filtered, more secure and less data sent.