Introduction

This application supports routing. I copied the router class and other classes from my framework. All routes are defined in the \routes directory. But since routing wasn't part of the task, I won't explain how it works. I copied some classes from my framework besides the Router class, like the Paginator class.

Configuration.

All configuarations are defined in the config\config.php file.

```php
<?php
    define('DEBUG',true);
    define('DB_NAME','db_contacts');
    define('DB_USER','root');
    define('DB_PASSWORD','');
    define('DB_HOST','127.0.0.11:3309');
    define('DEFAULT_CONTROLLER','Home'); //default controller
    define('ENVIROMENT','Development');
    define('SITE_NAME','contacts');
    define('FILES_DIR',ROOT .DS."files");
```

The application assumes the it will be ran on a local server like wampserver where it's accessed like so: localhost:8080/contacts/routename. If your local server allows you to access the app like so: sitename.dev (or something like that), change the ENVIROMENT constant to 'live' for the url() function to work properly.

I created a view and a controller that consumes the /contacts api. That view is accessible on this route: /list , and this is what it looks like:

Saving Contacts

I read all records from the csv file, for every record found I call the createContact method.

```php
public function createContact($contact_arr,$emailAddressDomain){

    $date = date_create($contact_arr['date']);
    $date = date_format($date,"Y-m-d ").$contact_arr['time'];

    $contact = new Contact();
    $contact->id = $contact_arr['id'];
    $contact->title =  $contact_arr['title'];
    $contact->first_name = $contact_arr['first_name'];
    $contact->last_name = $contact_arr['last_name'];
    $contact->email = $contact_arr['email'];
    $contact->date =  $date;
    $contact->note = $contact_arr['note'];
    $contact->tz = $contact_arr['tz'];
    $contact->image =  $this->createContactCard($contact);
    $contact->email_ip = gethostbyname($emailAddressDomain);

    $contact->save();

}
```

Figure 1 | the createContactMethod

Inside the createContact method, I instantiate a new contact model. The contact model extends a Parent model defined in the core directory, and therefore inherits all methods from the Parent model including the save method.

```php
<?php
 namespace models;

 use core\database\model;

class Contact extends Model
{

  protected $table =  'contacts';
  public $primary_key = 'id';
    public function __construct()
    {
        $table = 'contact';
        parent::__construct($table);
    }
}
```

Figure 2 | The contact Model

The save Method

```php
public function save(){

    $fields = $this->get_fields();
    //determine whether to updae or insert
    if(property_exists($this,'id') && $this->find($this->id) == true){
        return $this->update($this->id,$fields);
    }else{
       $this->insert($fields);
       return  $this->id;
    }
}
```

The save Method get all fields that are assigned to the contact model,

(like so, $contact->fieldname) , and pass them to the insert method. The insert method calls an insert method defined on the query builder class.

Figure 3 | The insert method from the Parent Model

```
public function insert($fields){

    if(empty($fields)) return false;

    return $this->query_builder->insert($fields);

}
```

An instance of the queryBuilder class is created when a model is being instantiated. The insert method from the queryBuilder class makes use of the grammer class to construct a sql insert statement, and pass that insert statement to the PDO connection class.. The connection class execute the insert statement.

```
public function insert($values){

    $insert_statement = $this->grammer->compileInsert($this,$this->model->get_fields());
    $this->connection->insert($insert_statement,$values);

}
```

The /contacts endpoint

I use the database Façade to get contacts from the database. The DB façade allows you to call methods from the query builder class statically, even though they are not defined statically on the query Builder class. That enables me to call methods from the query builder class without creating an instance of the query builder class.

Figure 4 | getContacts method from Api class

```
$per_page = 50; //number of records to display per page
$paginationType = 'ajax';
$contacts = DB::table('contacts')
            ->paginate($per_page,$paginationType);

$data = [
    'results' =>  $contacts->results,
    'pagination_links' => $contacts->get_pagination_links()
];

echo json_encode($data);
```

I call the table method, which sets the table to work with. The table method returns $this (instance of the query builder class), that enables me to chain the paginate method to the table method.

The paginate method calculate a limit an offset for the current page, get records and pass the results to an instance of a Paginator class.

Figure 5 | The paginate method

```
function paginate($per_page,$pagination_type = ''){

    $total = $this->getCountForPagination();
    $page =isset($_GET['page']) ? $_GET['page'] : 1;
    $results = $total ? $this->forPage($page, $per_page)->get() : [];

    return new Paginator($total,$per_page,$pagination_type,$results);
}
```

Figure 6 | The get() that get results from the database

```
public function get(){

   if(is_null($this->columns)) $this->columns = ['*'];
   $select_statement = $this->grammer->compileSelect($this);


   $result = $this->connection->get($select_statement);

   if(!empty($this->model)){

      return $this->return_results_objects($result);

    }else{

      return $this->return_result_as_array($result);
    }

 }
```

The get method checks if the columns array of the QueryBuilder class is null or not, if it's null it uses * (all) as a default column. When you call the select method, the columns array will hold columns that are passed as parameters to the select method. The columns array is used by the Grammer class to construct a select statement. The where method works the same way as the select method, but the grammer class uses the parameters that are passed to it to construct a where clause.

Figure 7 | The select, where and get methodss

```
$times_zones = DB::table('contacts')
            ->select('tz','id')
            ->selectRaw('CONCAT(title ," ", first_name," ",
            ->where("tz", $time_zone)
            ->get();
```

Avoiding SQL Injection

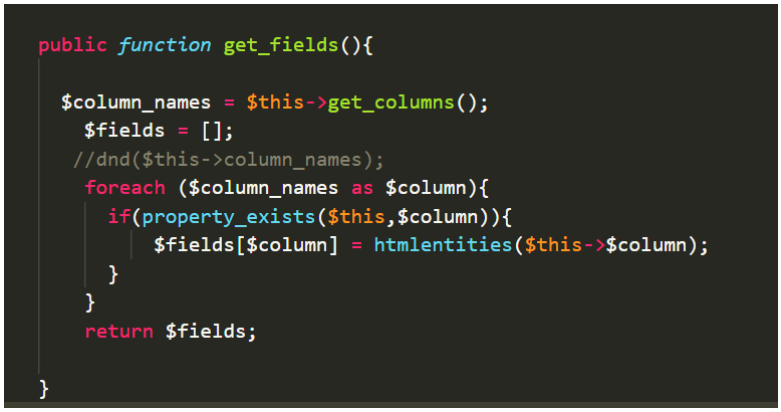
I use PDO prepared statements to avoid sql injection

Figure 8 | from the connection class

```
public function statement($query, $bindings = [])
{
        $statement = self::$connection->prepare($query);

        foreach ($bindings as $key=>$val) {

          if(!empty($val)) $statement->bindValue($key,$val);

        }
        extract($bindings);

        return $statement->execute();

}
```

And I use htmlentities to convert all some chars to html entities before saving data to the db.

Figure 9 | from the parent model, get propers that are assigned to a model and convert chars to html entities



```
public function get_fields(){

  $column_names = $this->get_columns();
    $fields = [];
   //dnd($this->column_names);
    foreach ($column_names as $column){
      if(property_exists($this,$column)){
          $fields[$column] = htmlentities($this->$column);
      }
    }
    return $fields;

}
```