Design Pattern Documentation

By the LCJJ SE:PT Development Team

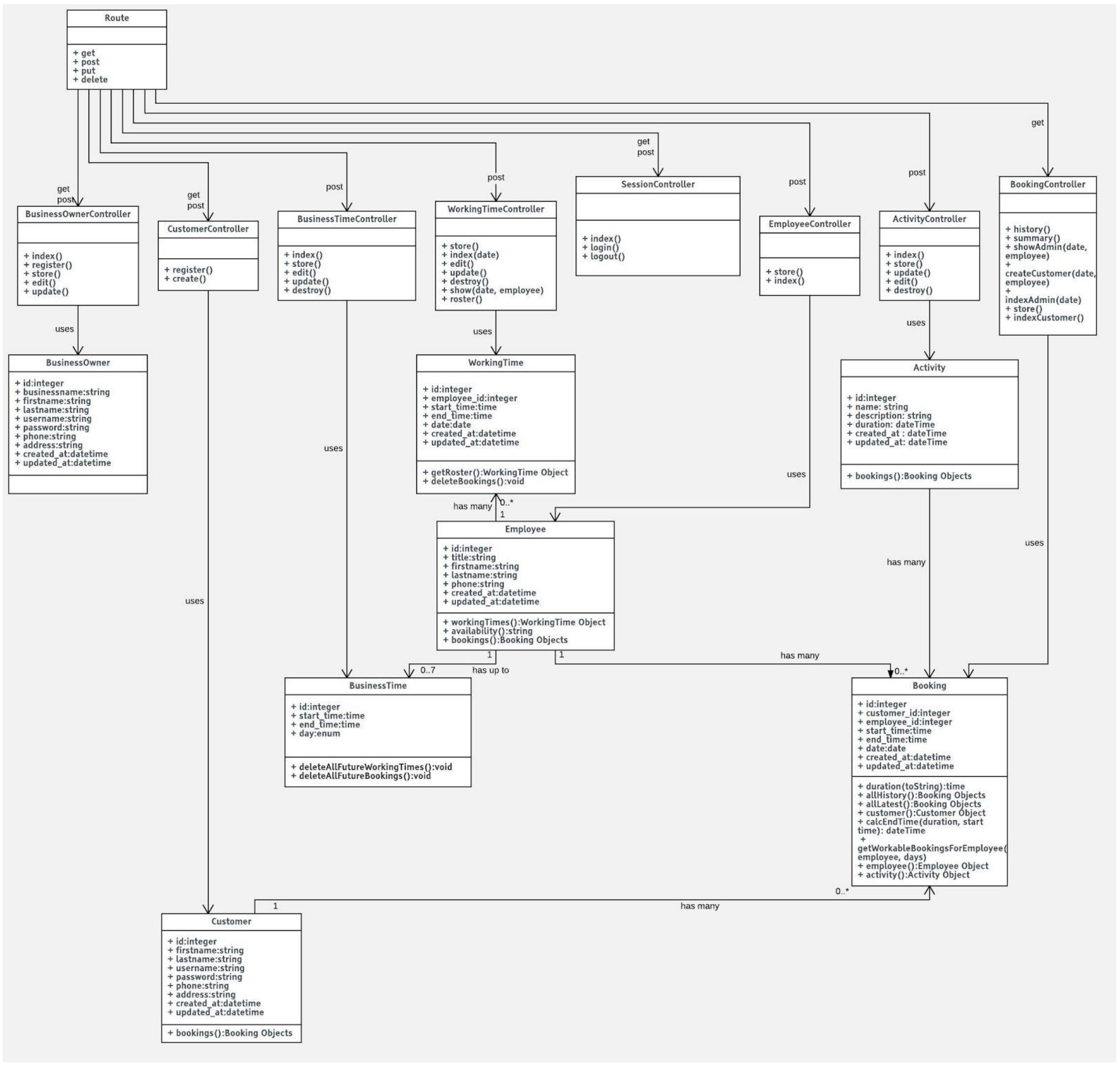


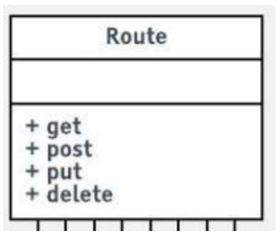
Figure 1.0 Class Diagram of the Appointment Booking System (full size picture is in the documents folder)

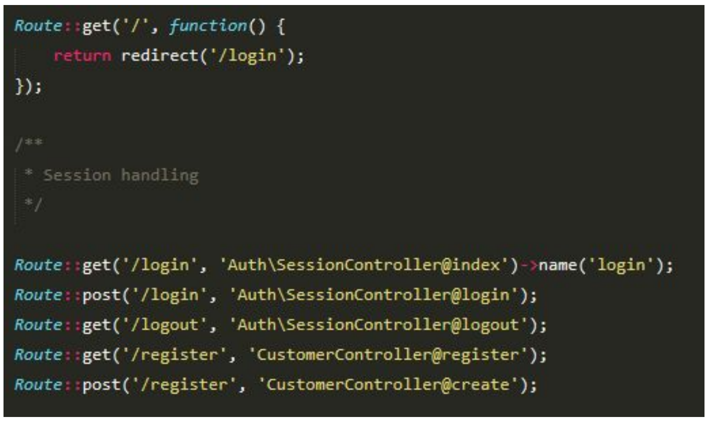
The design was strictly chosen to use the MVC pattern since we were using a PHP framework to develop our application.

Design Patterns

Model View Controller

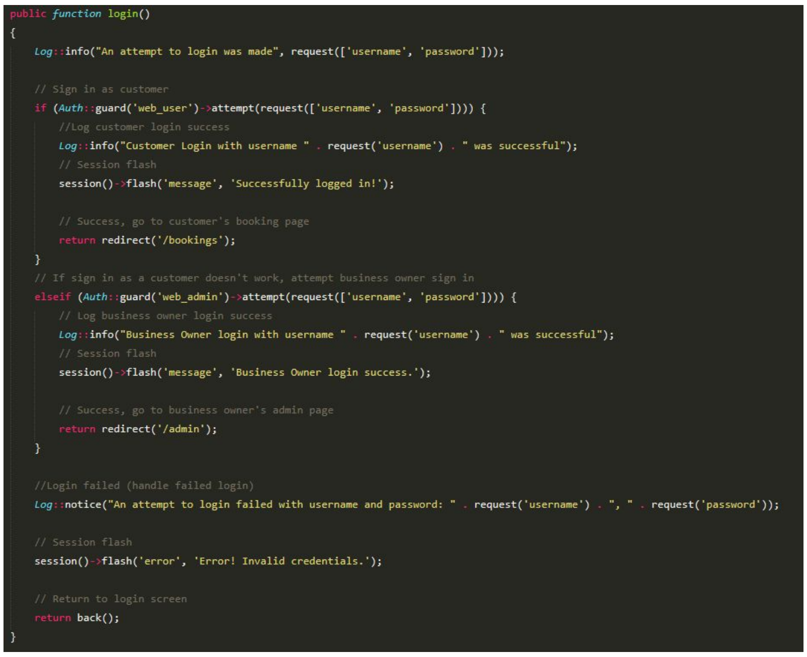
MVC which is short for the Model-View-Controllers is a software architecture, where the logic of the software is separated from the presentation of the software. We are able to achieve this using Laravel's control classes which serve to split the views and the functional code.





```php
Route::get('/', function() {
    return redirect('/login');
});

/**
 * Session handling
 */

Route::get('/login', 'Auth\SessionController@index')->name('login');
Route::post('/login', 'Auth\SessionController@login');
Route::get('/logout', 'Auth\SessionController@logout');
Route::get('/register', 'CustomerController@register');
Route::post('/register', 'CustomerController@create');
```

Figure 2.0 routes file from routes/web.php

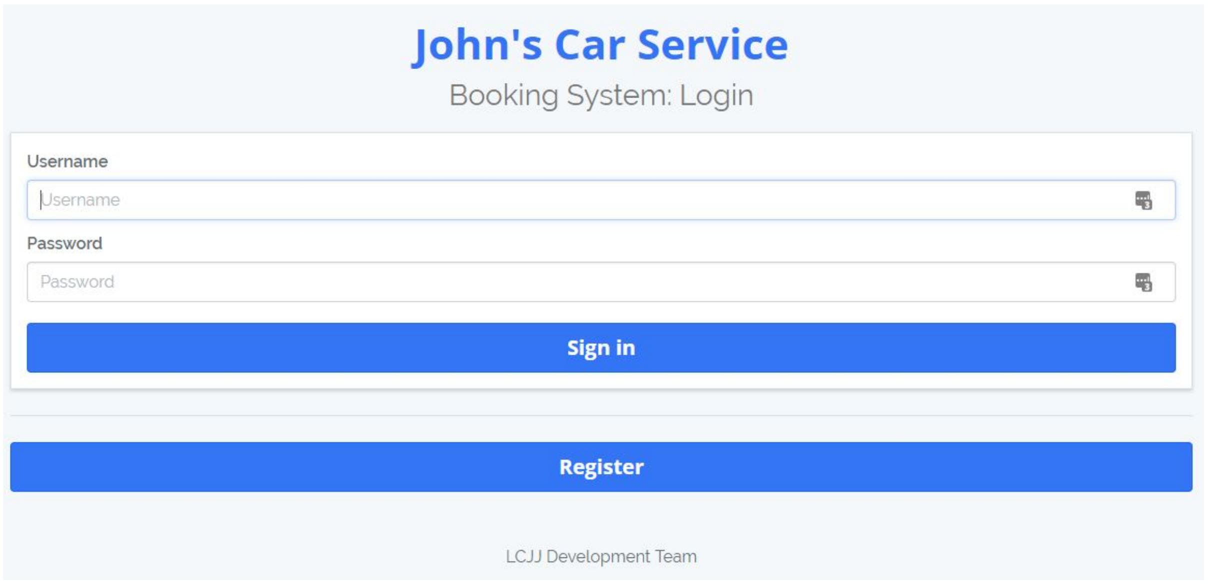


```php
public function login()
{
    Log::info("An attempt to login was made", request(['username', 'password']));

    // Sign in as customer
    if (Auth::guard('web_user')->attempt(request(['username', 'password']))) {
        //Log customer login success
        Log::info("Customer Login with username " . request('username') . " was successful");
        // Session flash
        session()->flash('message', 'Successfully logged in!');

        // Success, go to customer's booking page
        return redirect('/bookings');
    }
    // If sign in as a customer doesn't work, attempt business owner sign in
    elseif (Auth::guard('web_admin')->attempt(request(['username', 'password']))) {
        // Log business owner login success
        Log::info("Business Owner login with username " . request('username') . " was successful");
        // Session flash
        session()->flash('message', 'Business Owner login success.');

        // Success, go to business owner's admin page
        return redirect('/admin');
    }

    //Login failed (handle failed login)
    Log::notice("An attempt to login failed with username and password: " . request('username') . ", " . request('password'));

    // Session flash
    session()->flash('error', 'Error! Invalid credentials.');

    // Return to login screen
    return back();
}
```

Figure 2.1 login function of session controller

Users access the application through a browser by sending requests. Main request methods users post are POST and GET where the route file defines specific paths to call functions in controllers. An example would be:

1. User requests GET for /login
2. Routes file calls index function in the session controller (figure 2.0)
3. The index function calls the view method where it produces a HTML file to present to the user (figure 2.1)

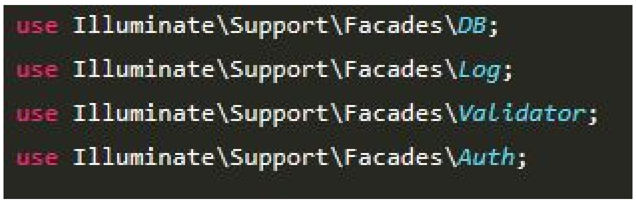


Another example of this in our application is editing working times for employees.

1. User visits /admin/{employee ID}/edit
2. User submits form containing edited working time data and sends to the applicatio
3. Routes file directs data from user to working time controller and calling the update method with form data

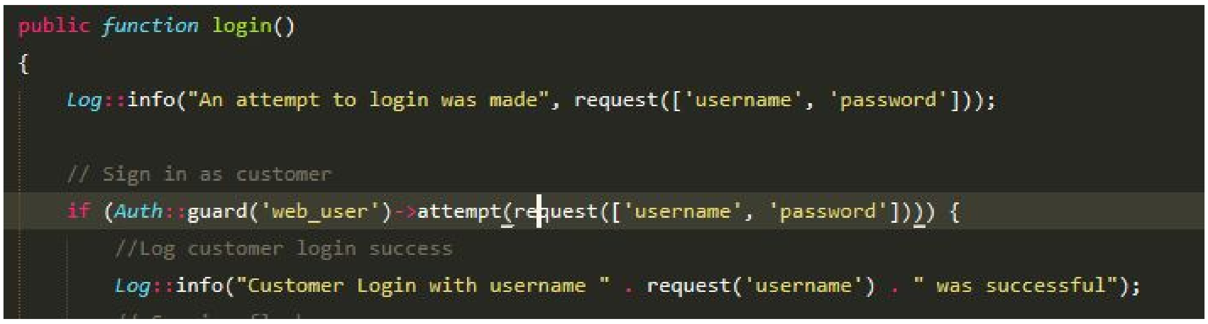
Facade

We have used a collection of facade methods to reduce complexity in the application. Laravel provides us with these methods for session auth, logging and validation.



```
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Auth;
```

Figure 3.0 referencing certain facade classes



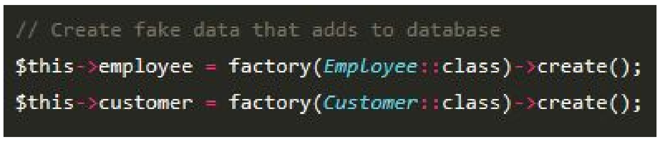
```
public function login()
{
    Log::info("An attempt to login was made", request(['username', 'password']));

    // Sign in as customer
    if (Auth::guard('web_user')->attempt(request(['username', 'password']))) {
        //Log customer login success
        Log::info("Customer Login with username " . request('username') . " was successful");
```

Figure 3.1 use of facades in the login function of the session controller

As shown in figure 3.1, the facade function calls a static method guard from the Auth class and then calls the attempt method to check user authentication. Laravel provides us with the Auth class with all logic relating to user authentication in the framework foundation code.

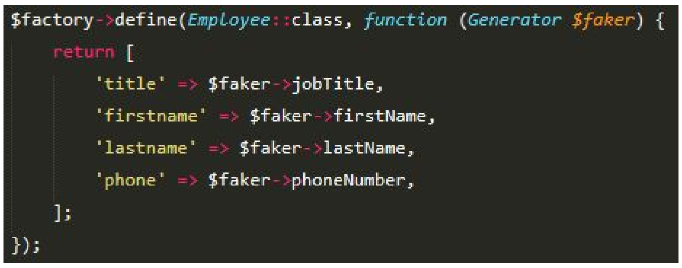
Factory

We use the factory design pattern with test cases as we create quick instances of models such as employees or customers



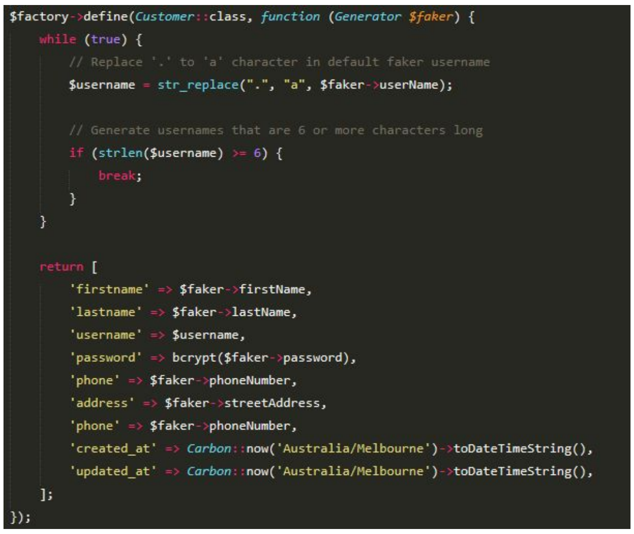
```php
// Create fake data that adds to database
$this->employee = factory(Employee::class)->create();
$this->customer = factory(Customer::class)->create();
```

Figure 4.0 factory helper methods called from an integration test file



```php
$factory->define(Employee::class, function (Generator $faker) {
    return [
        'title' => $faker->jobTitle,
        'firstname' => $faker->firstName,
        'lastname' => $faker->lastName,
        'phone' => $faker->phoneNumber,
    ];
});
```

Figure 4.1 employee factory method



```php
$factory->define(Customer::class, function (Generator $faker) {
    while (true) {
        // Replace '.' to 'a' character in default faker username
        $username = str_replace(".", "a", $faker->userName);

        // Generate usernames that are 6 or more characters long
        if (strlen($username) >= 6) {
            break;
        }
    }

    return [
        'firstname' => $faker->firstName,
        'lastname' => $faker->lastName,
        'username' => $username,
        'password' => bcrypt($faker->password),
        'phone' => $faker->phoneNumber,
        'address' => $faker->streetAddress,
        'phone' => $faker->phoneNumber,
        'created_at' => Carbon::now('Australia/Melbourne')->toDateTimeString(),
        'updated_at' => Carbon::now('Australia/Melbourne')->toDateTimeString(),
    ];
});
```

Figure 4.2 customer factory method

Laravel has a feature called model factories where we can call helper functions (figure 4.0) to directly create or generate model data from a configuration file (figures 4.1, 4.2).