Cheat Sheet:

Routing in Laravel:

```
Route::get( uri: '/staff_section',[SomeOneController::class,'Type'])->middleware( middleware: 'auth');;
Route::post( uri: '/staffAdd',[StaffController::class,'staffAdd'])->middleware( middleware: 'auth');;
Route::get( uri: '/staffData',[StaffController::class,'staffData'])->middleware( middleware: 'auth');;
Route::post( uri: '/editstaffd',[StaffController::class,'editstaffd'])->middleware( middleware: 'auth');;
Route::post( uri: '/staffedit',[StaffController::class,'staffedit'])->middleware( middleware: 'auth');;
Route::post( uri: '/staffview',[StaffController::class,'staffview'])->middleware( middleware: 'auth');;
Route::post( uri: '/staffdel',[StaffController::class,'staffdel'])->middleware( middleware: 'auth');;



Route::get( uri: '/manage_room',[RoomController::class,'Room'])->middleware( middleware: 'auth');;
Route::post( uri: '/roomAdd',[RoomController::class,'RoomAdd'])->middleware( middleware: 'auth');;
Route::post( uri: '/editroom',[RoomController::class,'editroom'])->middleware( middleware: 'auth');;
Route::post( uri: '/roomdelete',[RoomController::class,'roomdelete'])->middleware( middleware: 'auth');;
Route::get( uri: '/RoomDetails',[RoomController::class,'RoomDetails'])->middleware( middleware: 'auth');;
Route::post( uri: '/CustomerShow',[RoomController::class,'CustomerShow'])->middleware( middleware: 'auth');;
Route::post( uri: '/room_type',[RoomController::class,'room_type'])->middleware( middleware: 'auth');;
Route::post( uri: '/checkout',[RoomController::class,'checkout'])->middleware( middleware: 'auth');;


Route::get( uri: '/Reservation',[CustomerController::class,'Reservation'])->middleware( middleware: 'auth');;
Route::post( uri: '/ReservationAdd',[CustomerController::class,'ReservationAdd'])->middleware( middleware: 'auth');;
Route::post( uri: '/price',[CustomerController::class,'price'])->middleware( middleware: 'auth');;


Route::post( uri: '/payment_Ok',[CustomerListController::class,'payment_Ok'])->middleware( middleware: 'auth');;
Route::get( uri: '/payment_Ok1',[CustomerListController::class,'payment_Ok1'])->middleware( middleware: 'auth');;
Route::get( uri: '/PaymentClear',[CustomerListController::class,'PaymentClear1'])->middleware( middleware: 'auth');;
Route::get( uri: '/customerList',[CustomerListController::class,'customerList1'])->middleware( middleware: 'auth');;
Route::get( uri: '/customer',[CustomerListController::class,'customer1'])->middleware( middleware: 'auth');;
```

For loop in PHP:

```
for ($i = 1; $i <= 10; $i++) {
  echo $i . PHP_EOL;
}
```

In this code, a for loop is used to iterate over an array of numbers. The loop starts with the for keyword, followed by the loop counter declaration and initialization (let i = 0), the termination condition (i < numbers.length), and the increment statement (i++). The code inside the loop simply logs each number to the console. The variable names are clear and concise, making it easy to understand what the code is doing. By following good coding practices, such as using meaningful variable names and writing clean, organized code

Proper Name:

```
$('.reservation').click(function() {

let RoomType = $("#room_type option:selected").text();
let RoomNo = $("#room_no option:selected").text();
let check_in_date = $('#check_in_date').val();
let check_out_date = $('#check_out_date').val();
let email = $('#email').val();
let first_name = $('#first_name').val();
let last_name = $('#last_name').val();
let contact_no = $('#contact_no').val();
const id_card_id = $("#id_card_id option:selected").text();
let id_card_no = $('#id_card_no').val();
let address = $('#address').val();
let transaction = $('#transaction').val();
let payment = $('#payment').val();
let pr = $('#total_price').html();
let p = pr-payment;
let day = $('#staying_day').html();

    //console.log(RoomType+RoomNo+check_in_date+check_out_date+email+
    // first_name+last_name+contact_no+id_card_id+address+transaction+id_card_no);
```

Comments:

```
                showToast('Reservation Data Added Failed!', {
                    duration: 5000, // The time interval after notification disappear
                    background: '#20b2aa', // Background color for toast notification
                    color: '#f1f1f1', //Text color
                    borderRadius: '15px', //Border Radius,
                    marginTop: '89px'
                });
            }
        }).catch(function(error) {



        showToast('Reservation Data Added Failed!', {
            duration: 5000, // The time interval after notification disappear
            background: '#ff2323', // Background color for toast notification
            color: '#000000', //Text color
            borderRadius: '7px', //Border Radius,
            marginTop: '89px'
        });
    })
```

In this code, comments are used to explain the purpose and function of each part of the code. The comments are written in plain English, making the code easy to understand

and follow. They are preceded by //, which indicates that the rest of the line is a comment and should not be executed as part of the code. This helps to improve the code's readability and maintainability, making it easier to update and modify in the future.

HTML form: Here's an example of clean and well-organized HTML code for a form

```
<form role="form"  data-toggle="validator" method="post" action="">
    <div class="row">
        <div class="form-group col-lg-6">
            <label for="complain_name">Complainant Name</label>
            <input id="complain_name" type="text" class="form-control"
                   placeholder="Complainant Name" name="complainant_name" required>
            <div class="help-block with-errors"></div>
        </div>

        <div class="form-group col-lg-6">
            <label for="complain_type">Complaint Type</label>
            <input id="complain_type" type="text" class="form-control"
                   placeholder="Complaint Type" name="complaint_type" required>
            <div class="help-block with-errors"></div>
        </div>

        <div class="form-group col-lg-12">
            <label for="complain">Please Describe Your Complaints</label>
            <textarea id="complain" class="form-control"
                      name="complaint" placeholder="Complaint" required></textarea>
        </div>

    </div>

    <button type="button" id="complain_button"
            class="btn btn-lg btn-success"
            name="createComplaint" >Submit</button>
    <button type="button" id="complain_rest_button"
            class="btn btn-lg btn-danger" >Reset</button>
</form>
```

HTML Table:

```
<table id="dataTable" class="table table-striped table-bordered scroll">
    <caption id="tableDescription">Description of the data table</caption>
    <thead>
        <tr>
            <th>Room No</th>
            <th>Room Type</th>
            <th>Room Free</th>
            <th>Booking Status</th>
            <th>Check Out</th>
            <th>Action</th>

        </tr>
    </thead>
    <tbody class="room_table">

    </tbody>
</table>
```

This code uses the <table> element to create the table, and the <thead> and <tbody> elements to organize the table's header and body content, respectively. The <th> elements are used for the header cells, and the <td> elements for the body cells.

CSS Attributes:

```
.reauth-email {
  display: block;
  color: #404040;
  line-height: 2;
  margin-bottom: 10px;
  font-size: 14px;
  text-align: center;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
```

Use of Function:

```
//For Room Table
function getRoomData() { //aita


    axios.get('/RoomDetails')
        .then(function(response) {
            //alert("Robin");

            if (response.status === 200) {


                //$('.room_table').DataTable().destroy();

                $('.room_table').empty();

                //  $('#dataTable').DataTable();

                var dataJSON = response.data;


                // alert(book)

                $.each(dataJSON, function (i, item) {

                    let book, checkout, view, edit
                    if (dataJSON[i].booking_status === 0) {
                        book = '<button class="btn btn-success">Book Room</button>';
                        checkout = '';
                        del = '<i class=""></i>';
                        edit = '<i class="fa fa-pencil"></i>';
                        view = '';
```

The code inside the function is organized and easy to read, with clear and concise variable names. The function returns the result of the calculation, which can then be stored in a variable and used elsewhere in the code. This example demonstrates the importance of writing clean and well-organized code, making it easier to understand, maintain, and debug.