**LAPORAN PRAKTIKUM**

**MATA KULIAH PEMROGRAMAN VISUAL**

***ANALISA VISUALISASI TAMPILAN APLIKASI KASIR RESTORAN MENGGUNAKAN PHP DAN MYSQL***



**Dosen Pengampu :**

Andi Iwan Nurhidayar, S.Kom., MT

**Disusun Oleh :**

Risa Amalia

18050623019

D3 MANAJEMEN INFORMATIKA

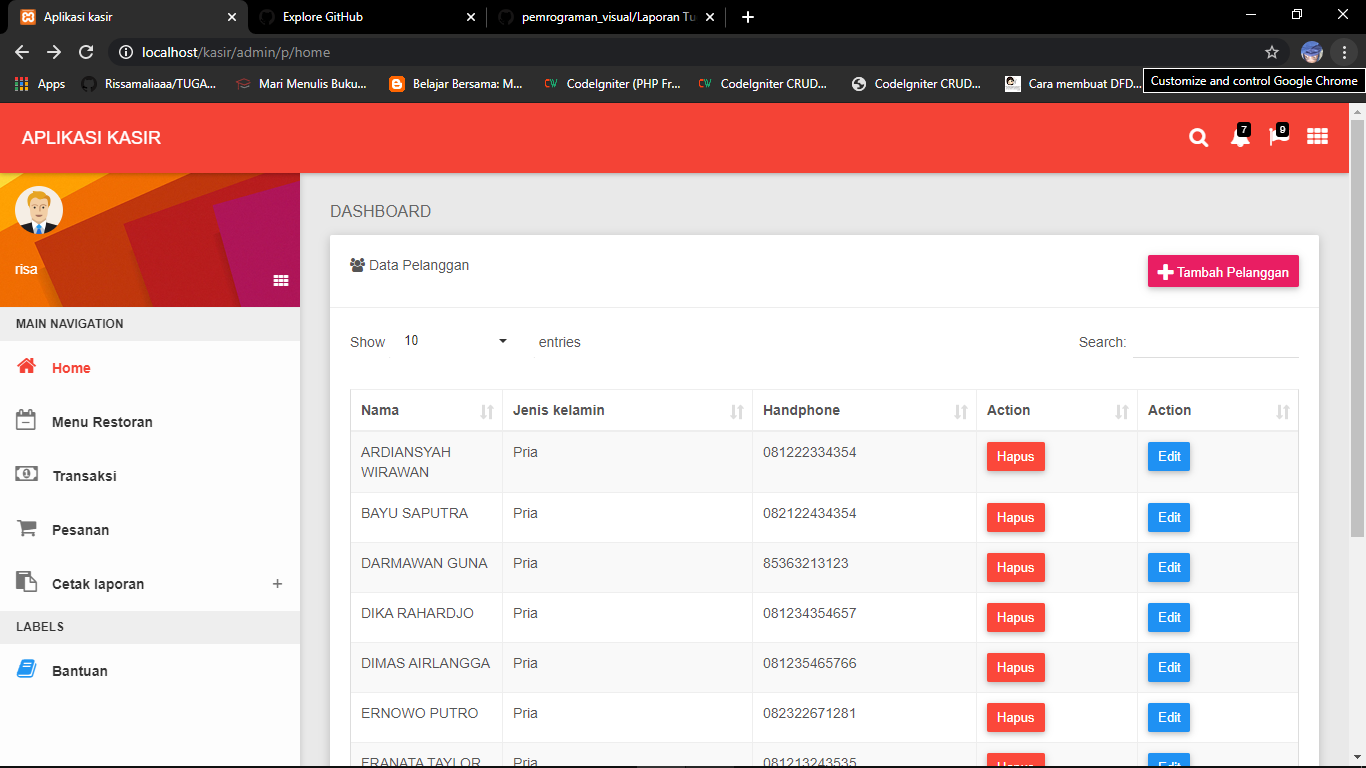
**JURUSAN TEKNIK INFORMATIKA**

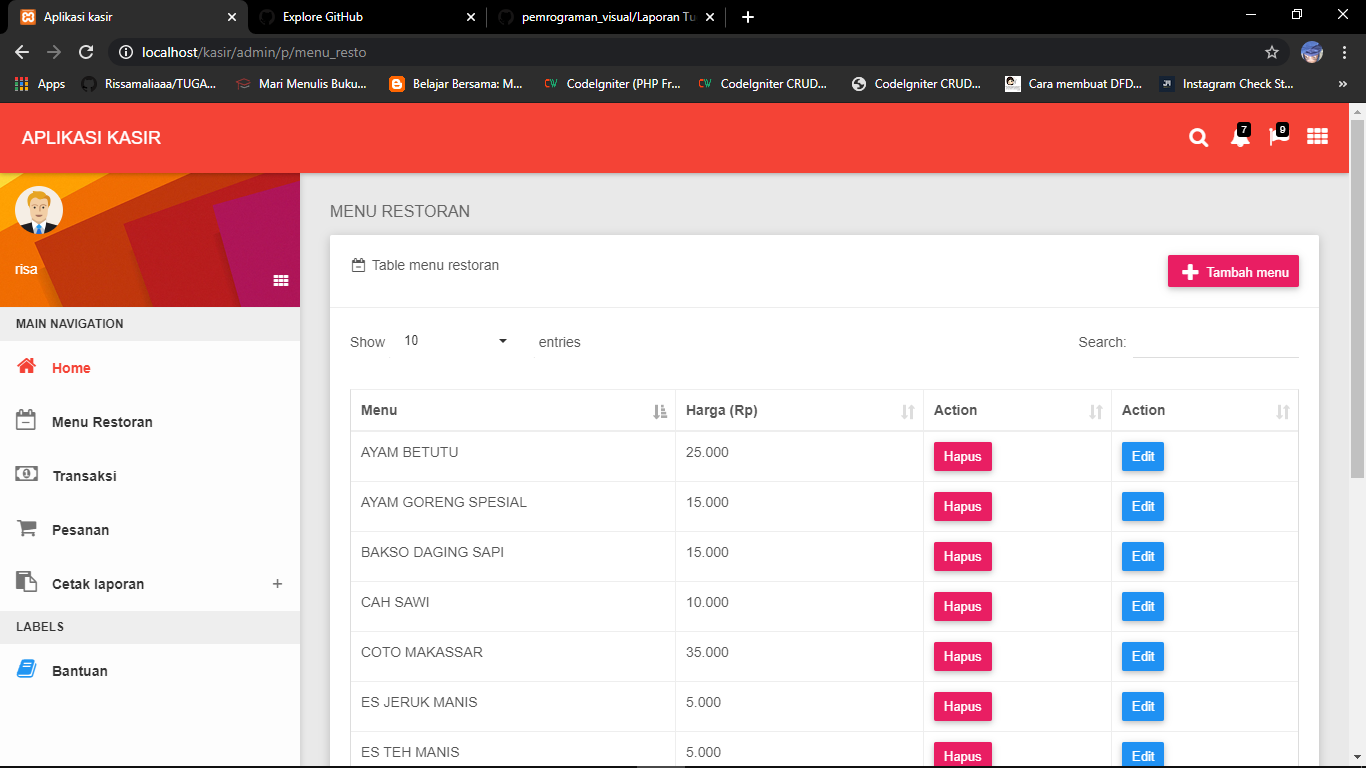
**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI SURABAYA**

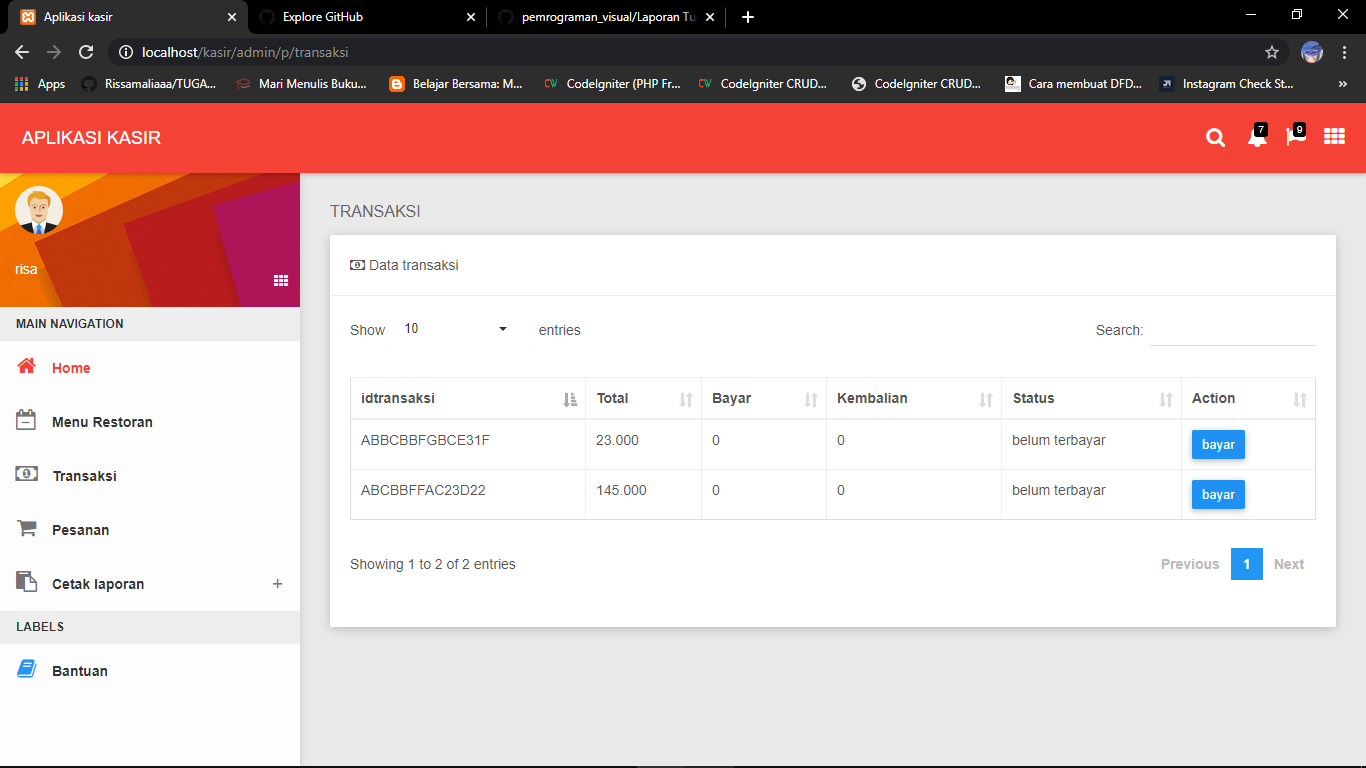
**SEMESTER GASAL – GENAP**

**2019 – 2020**

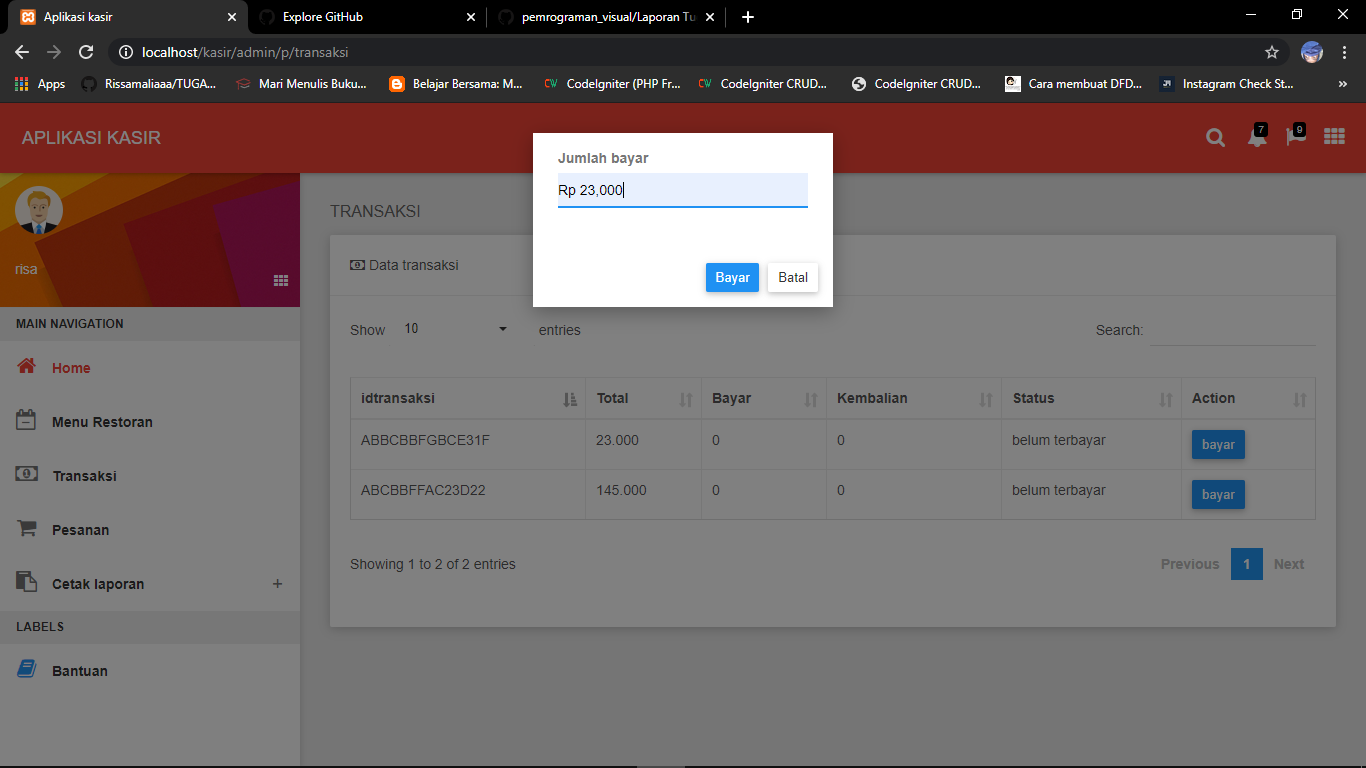
1. **Tampilan Menu Home**
2. **Tampilan Menu Restoran**



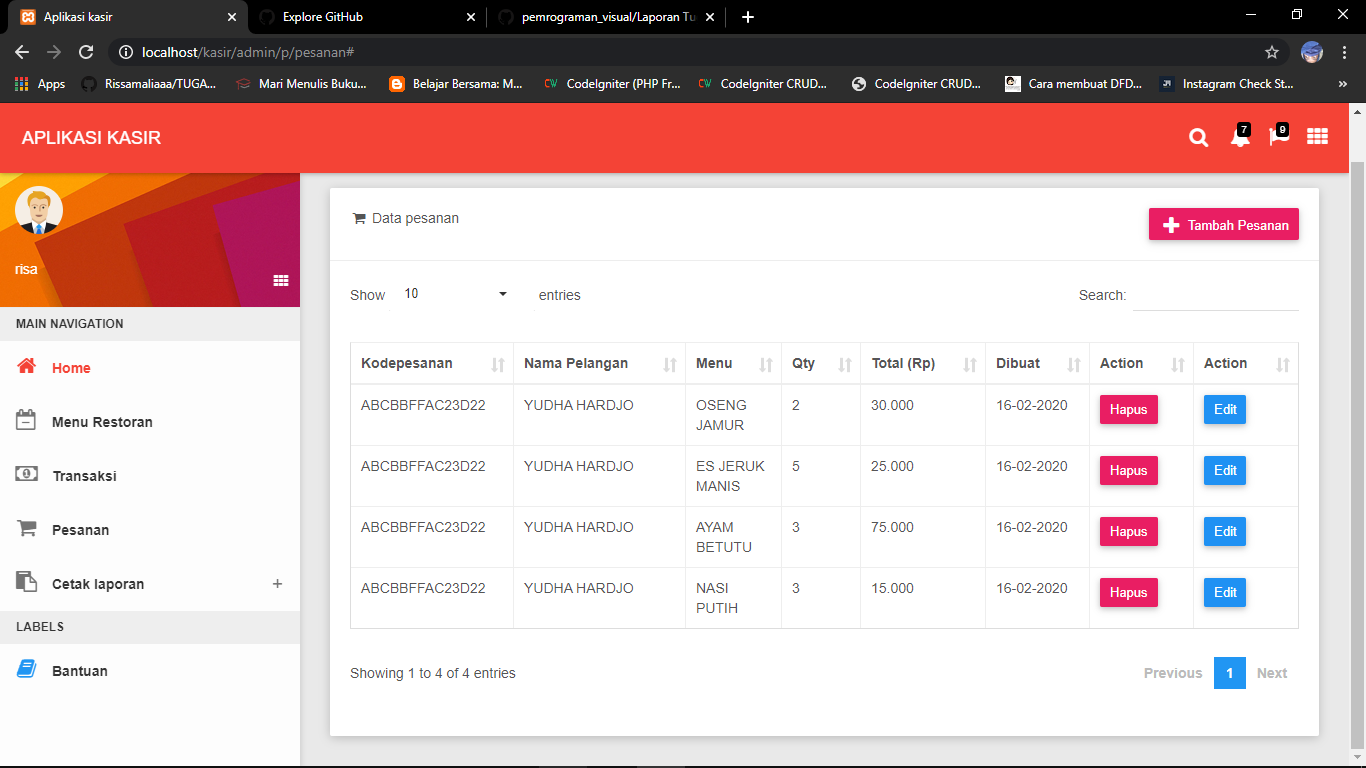
1. **Tampilan Halaman Transaksi**



1. **Tampilan Melakukan Pembayaran**

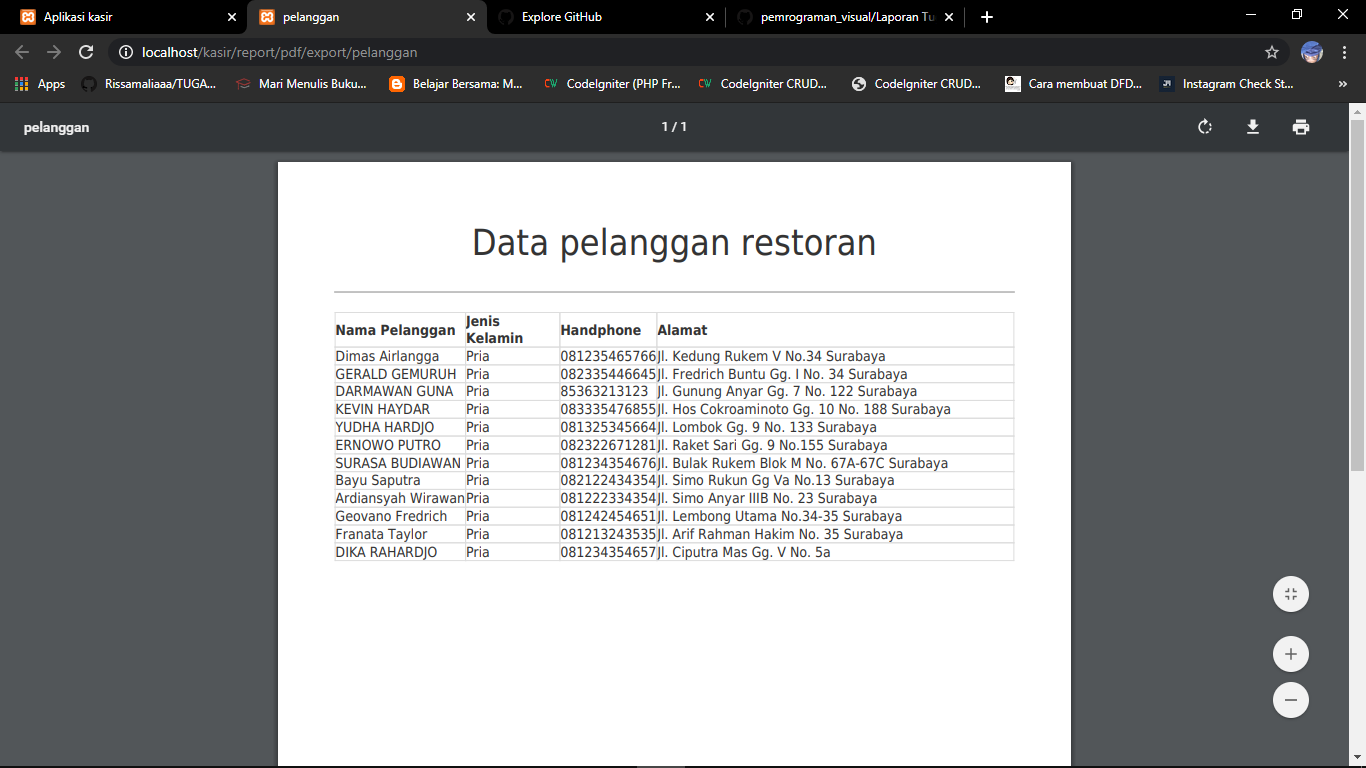


1. **Tampilan Halaman Pesanan**



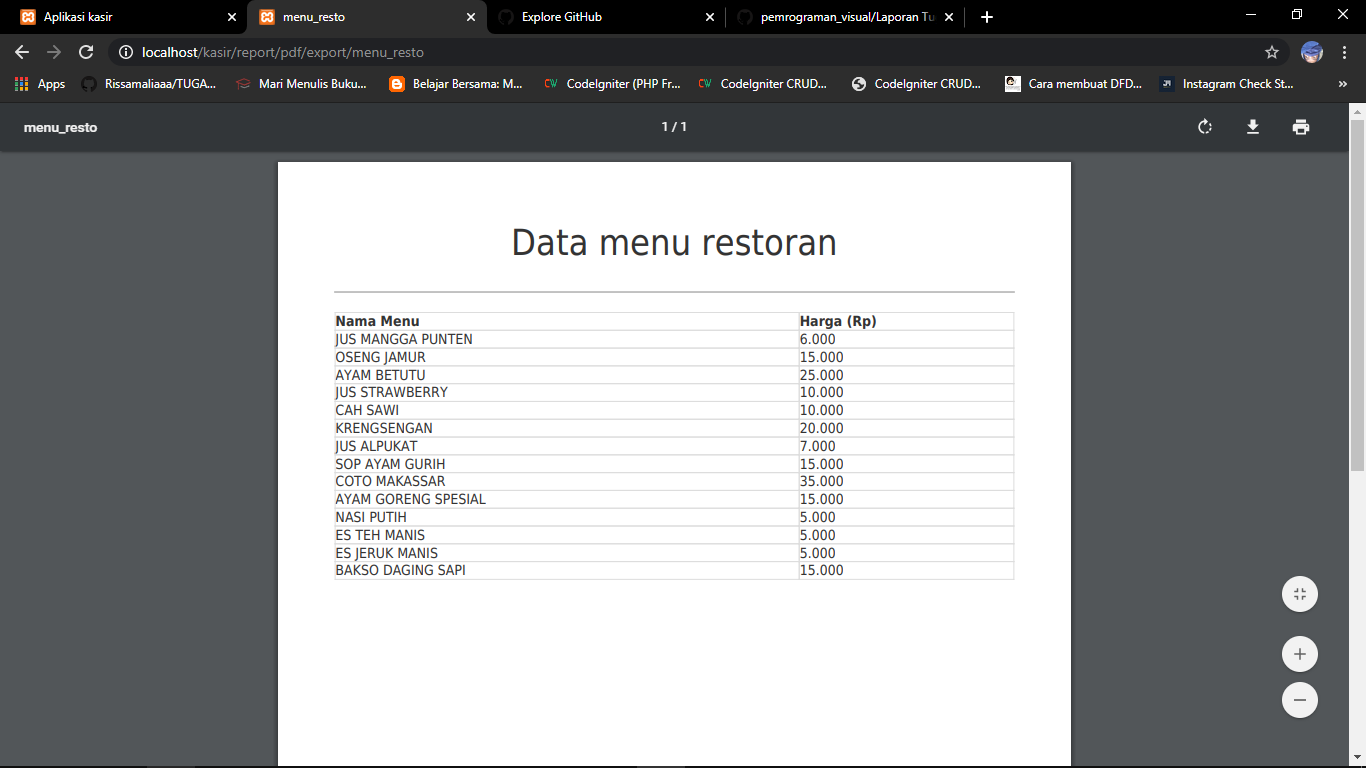
1. **Tampilan Cetak Laporan**
2. **Pelanggan Restoran**

* **Format PDF**



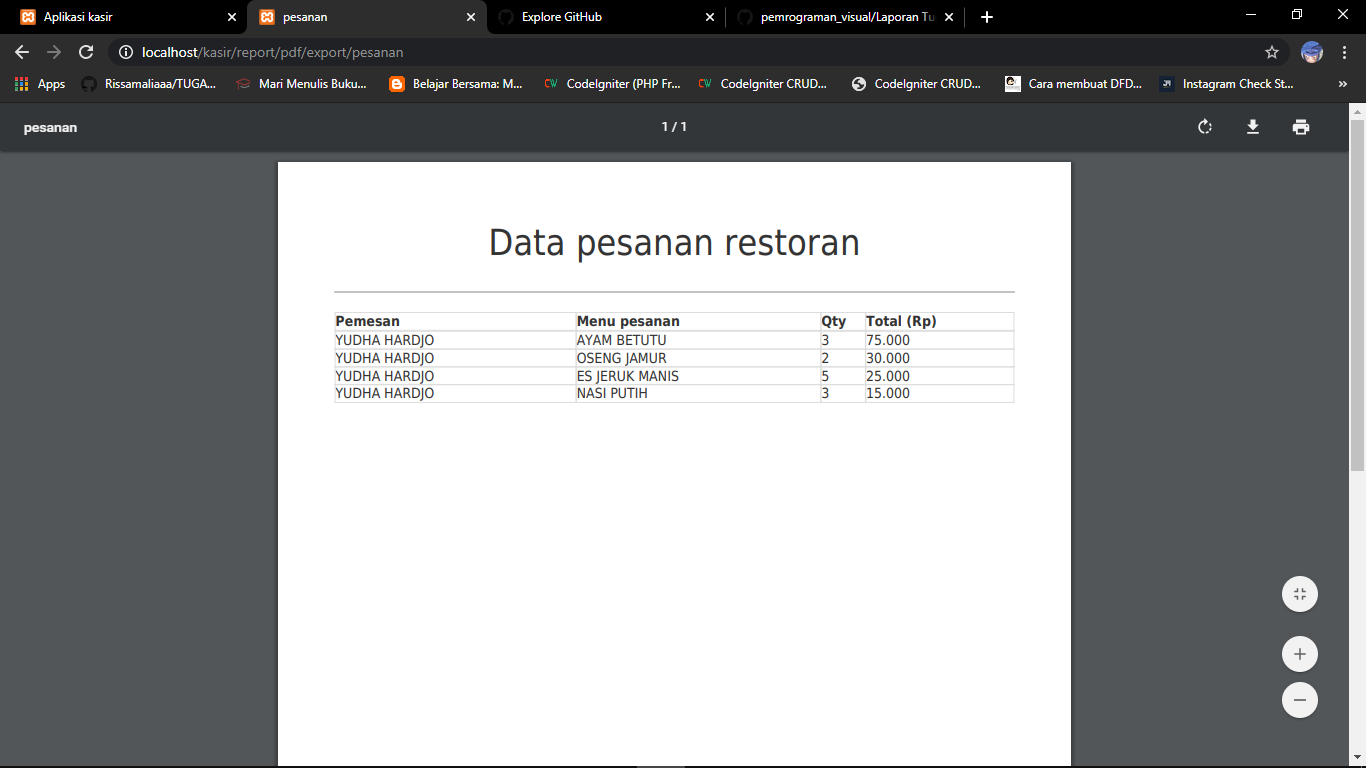
1. **Menu Restoran**

* **Format PDF**



1. **Pesanan Restoran**

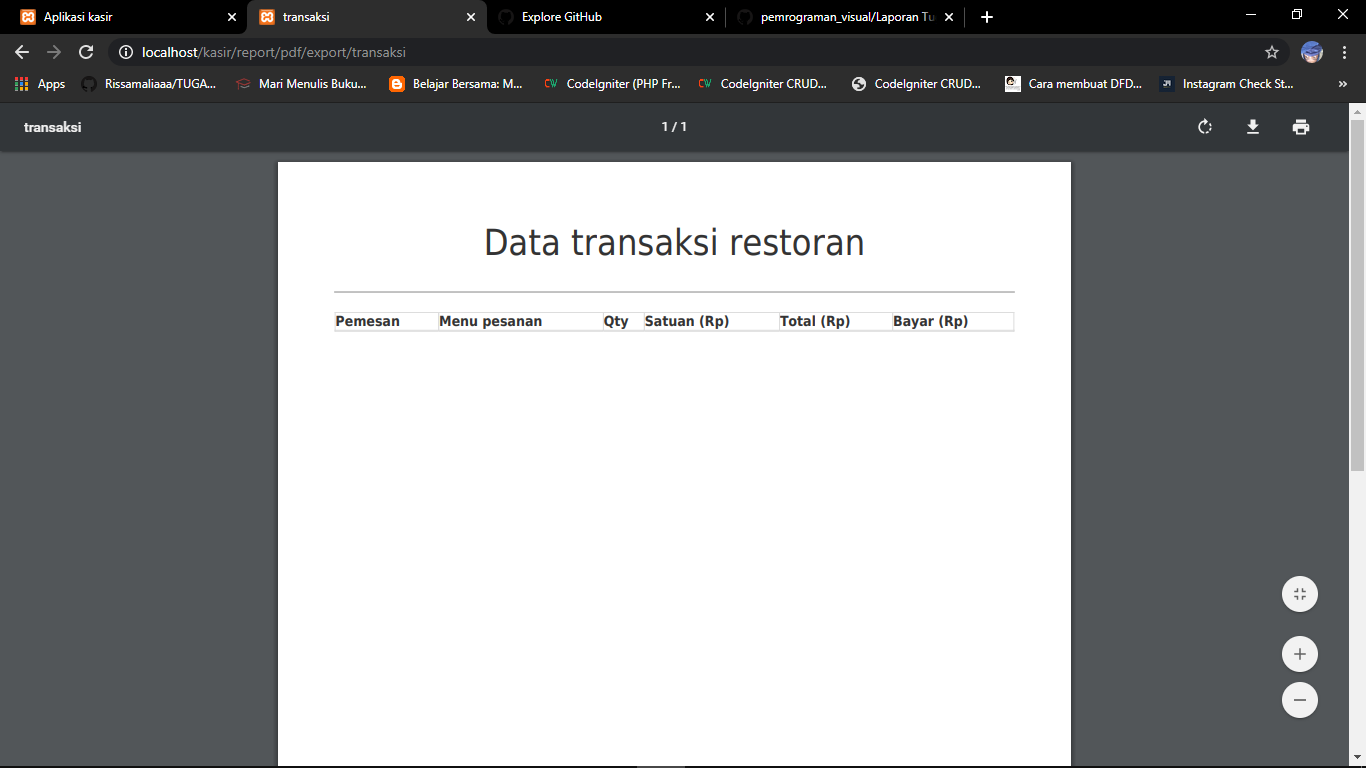
* **Format PDF**



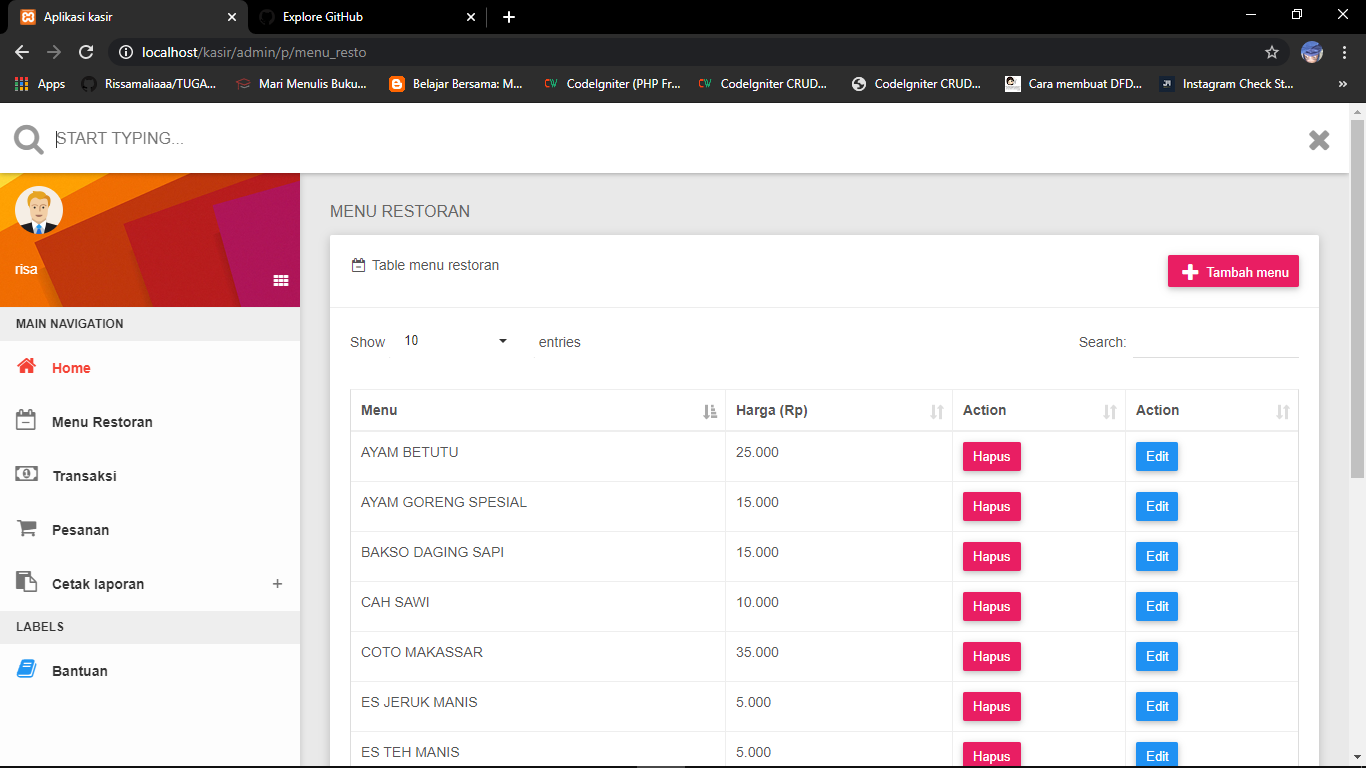
1. **Transaksi**

* **Format PDF**

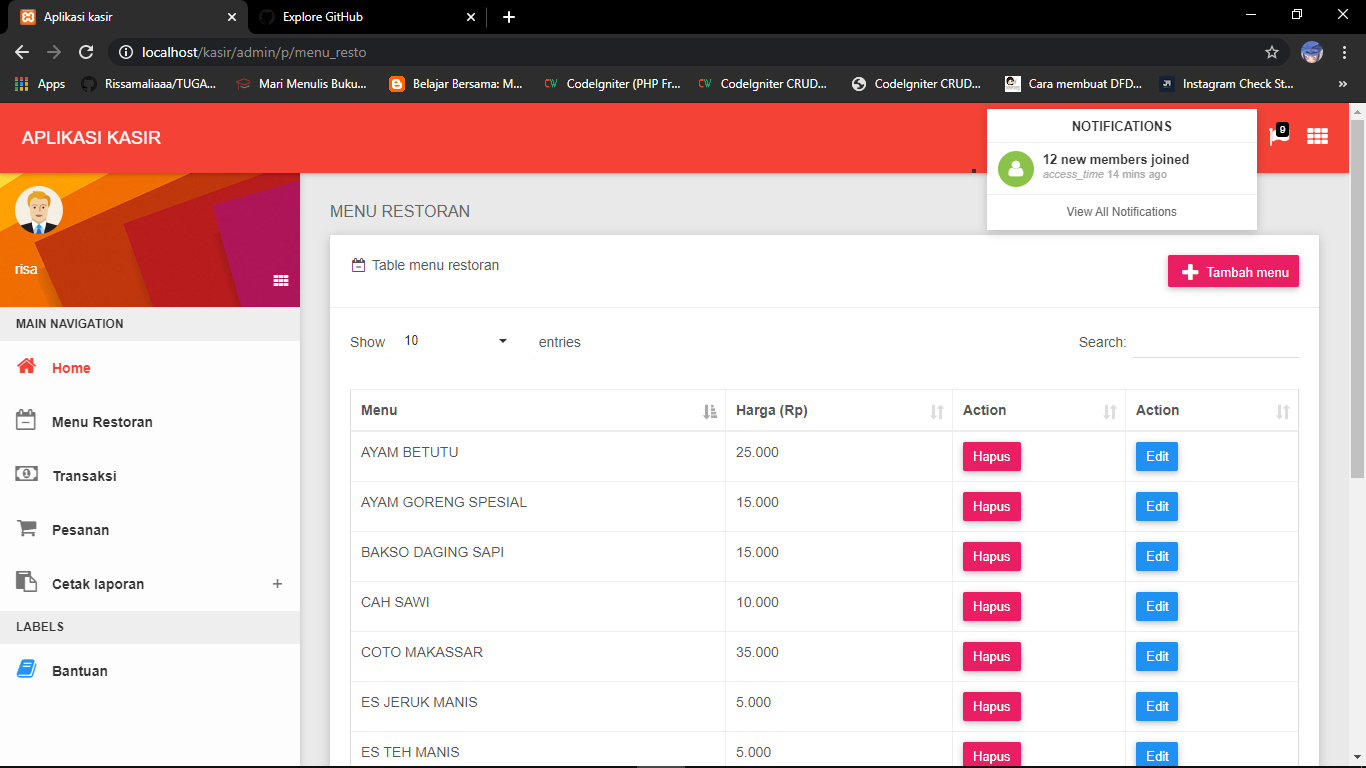
**( Contoh jika pelanggan belum membayar pesanan )**



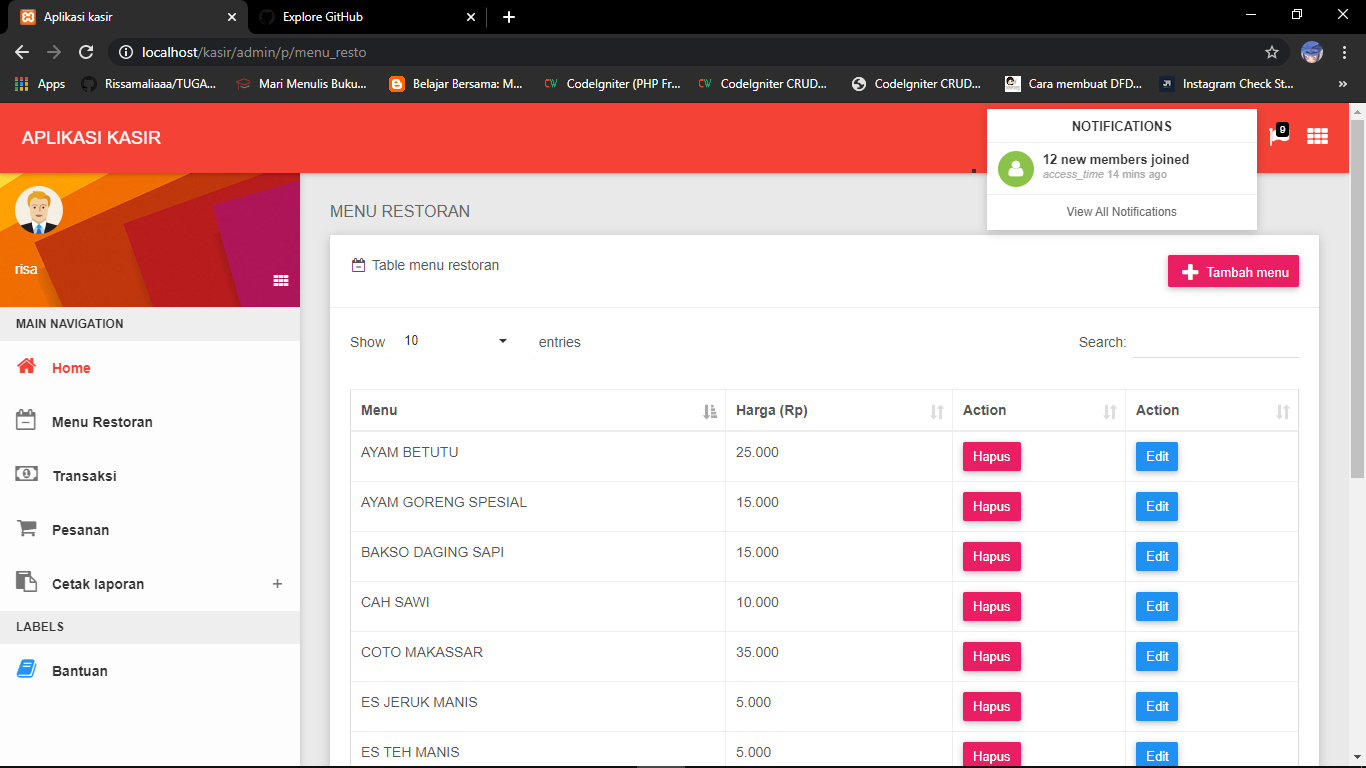
* **Tampilan Menu Search**



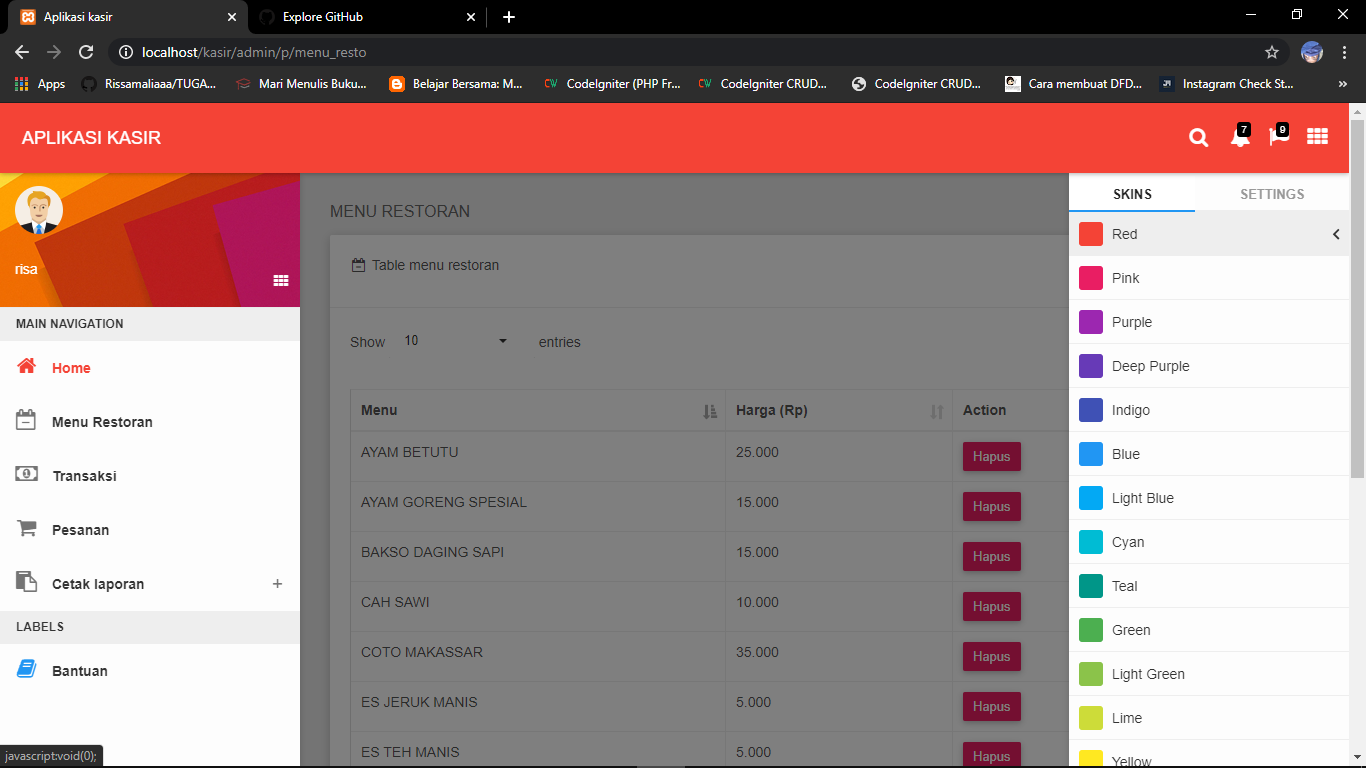
* **Tampilan Melihat Notifikasi**



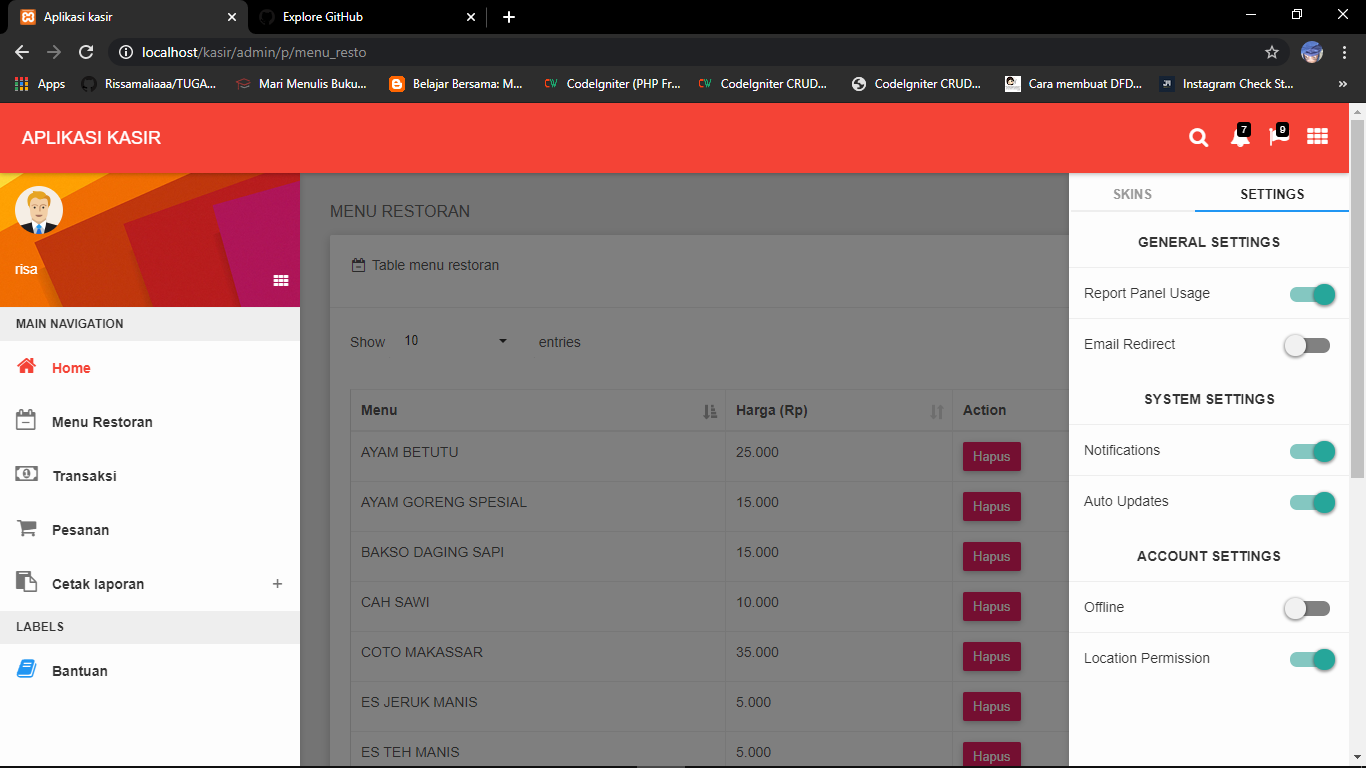
* **Tampilan Melihat Tasks**



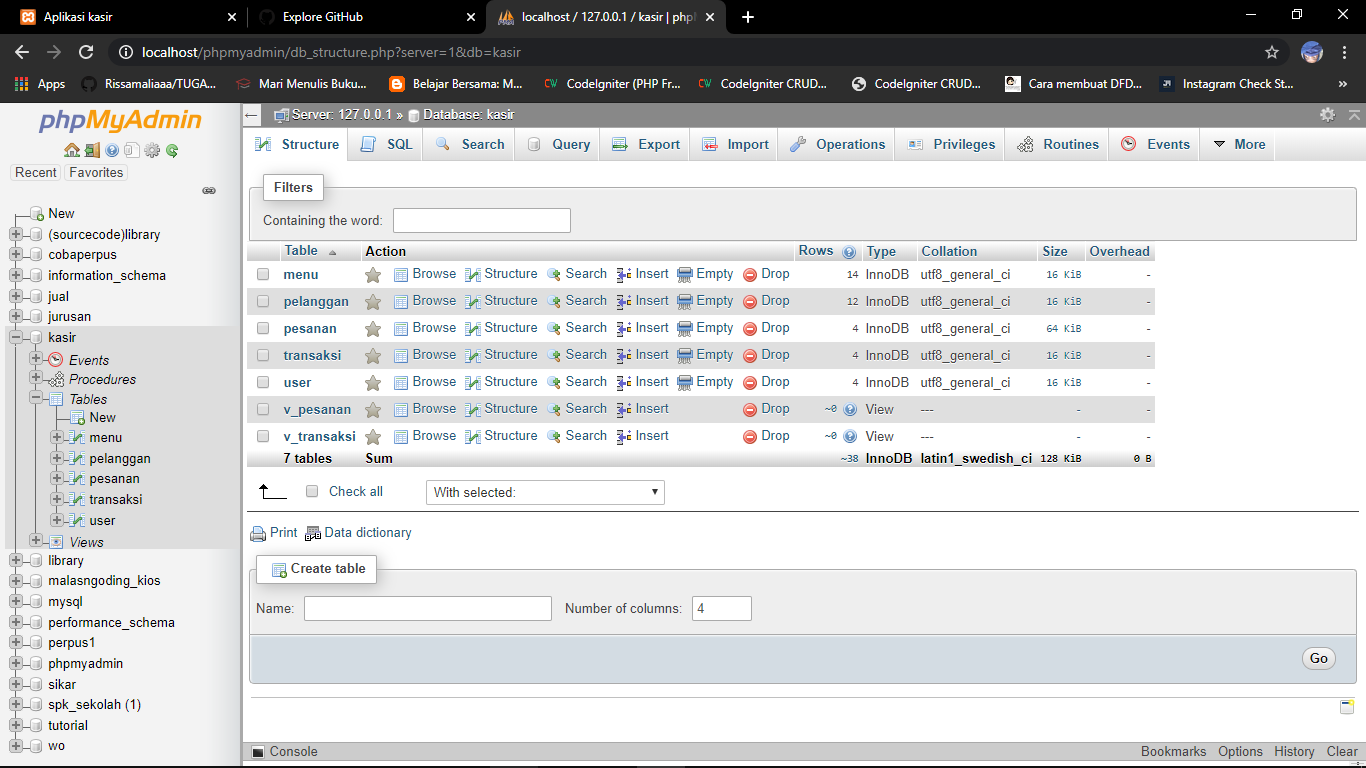
* **Tampilan Mengubah Tampilan Warna Taskbar**



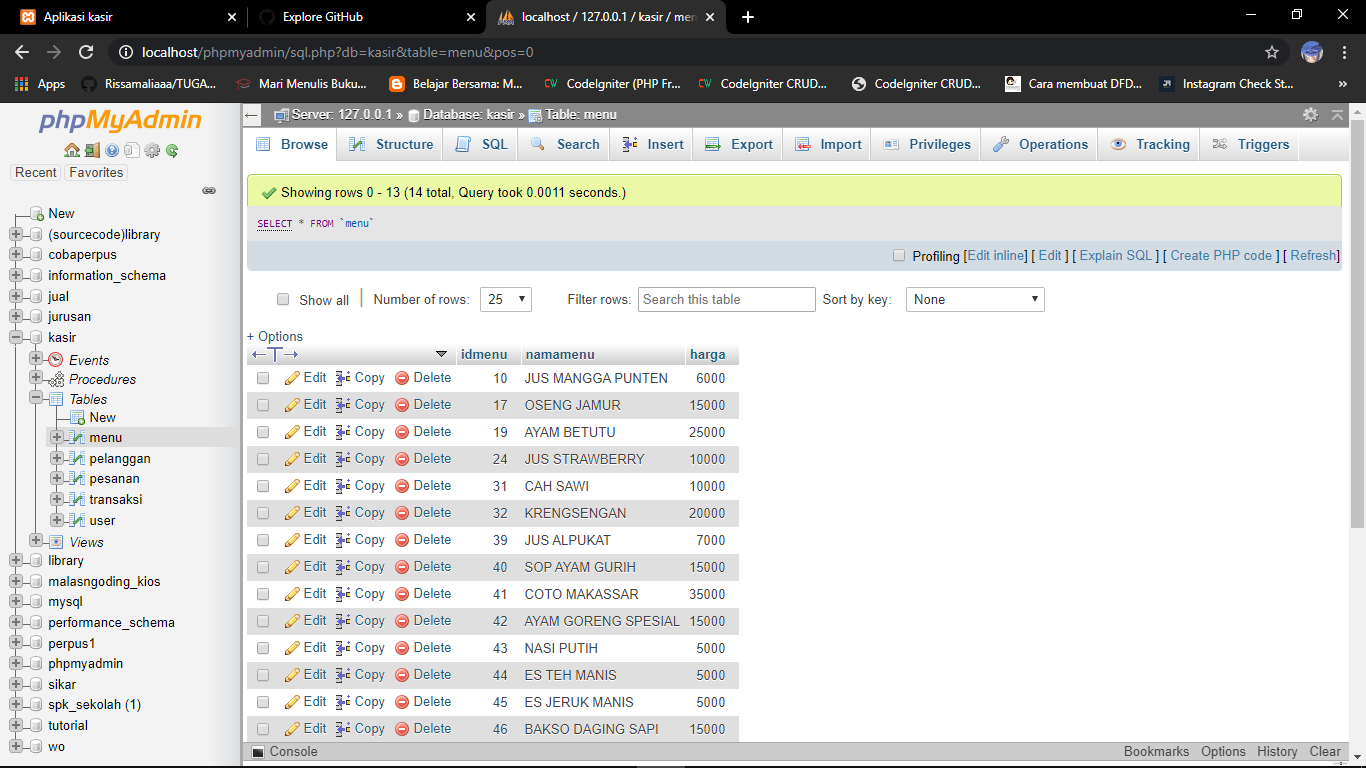
* **Tampilan Mengubah Setting**



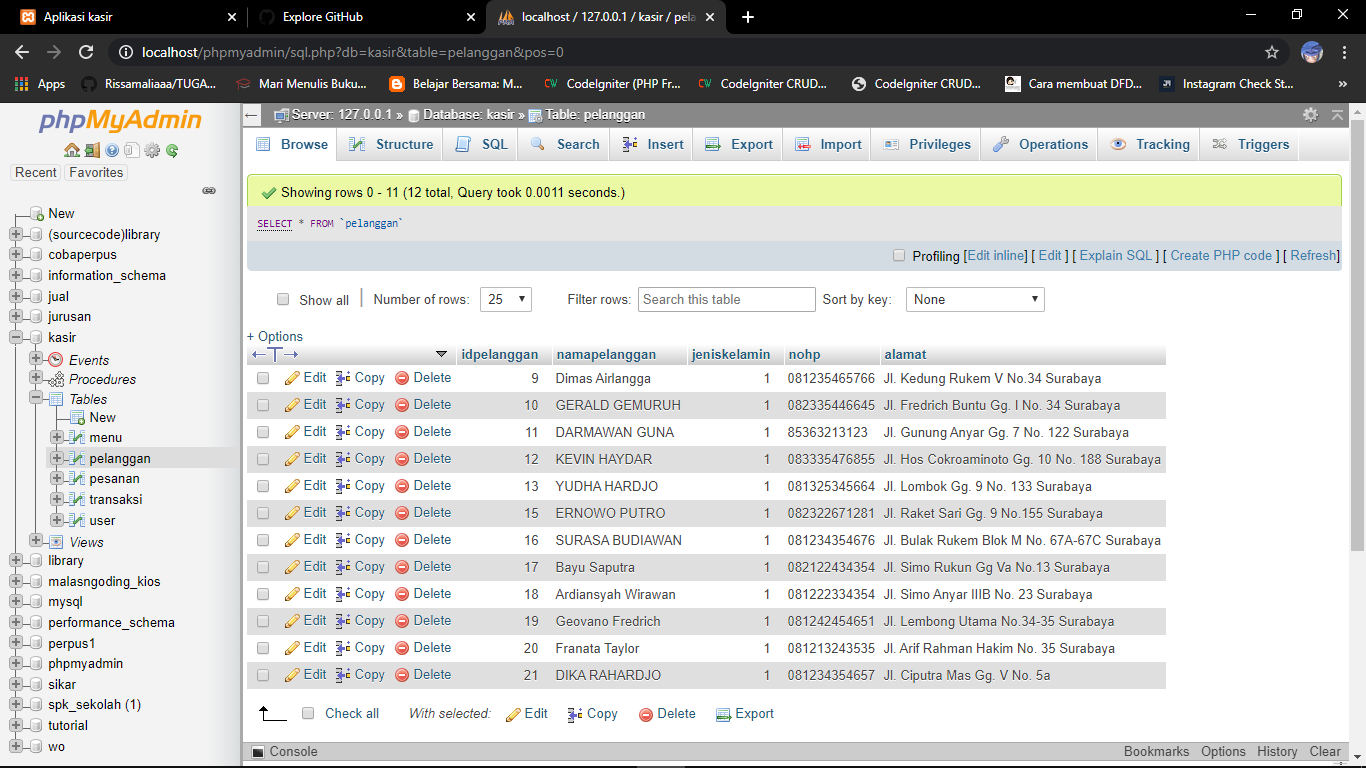
# **Tampilan Database**



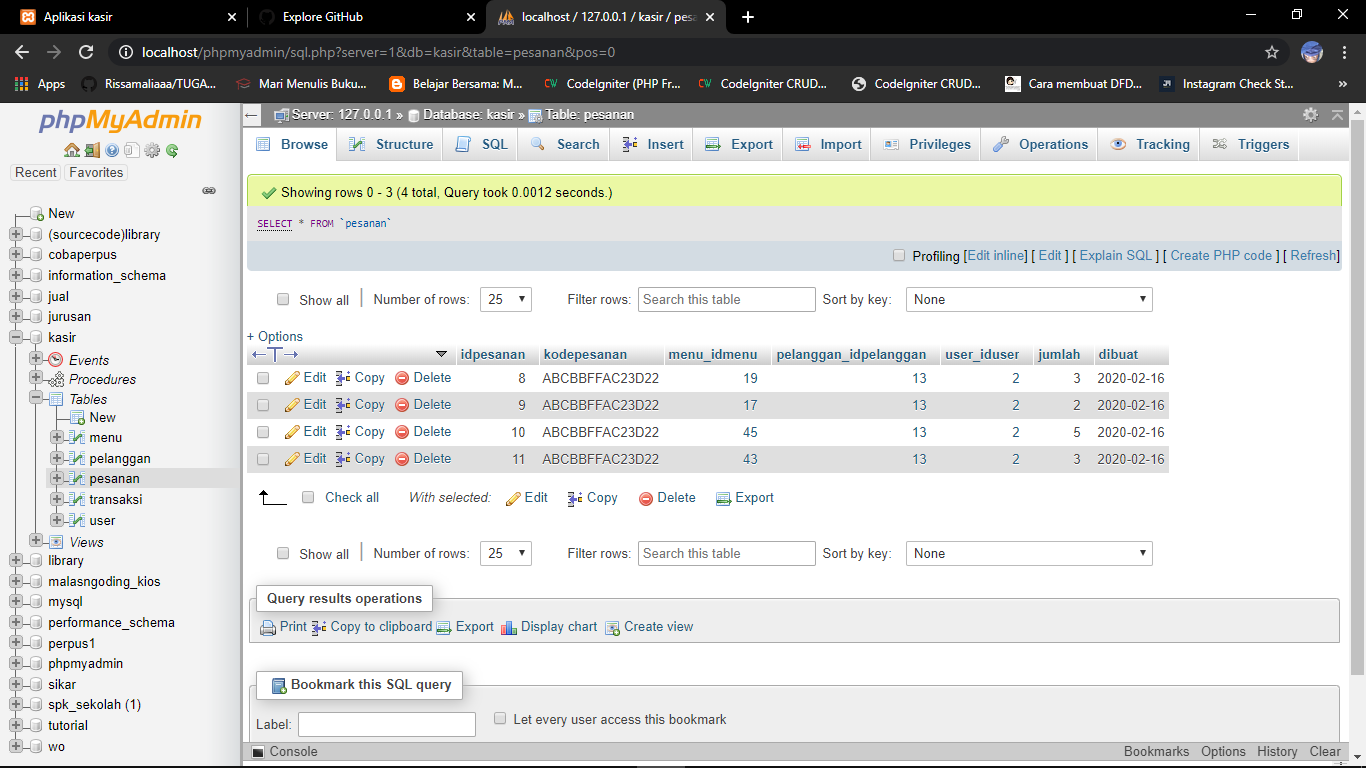
* **Table menu restoran**



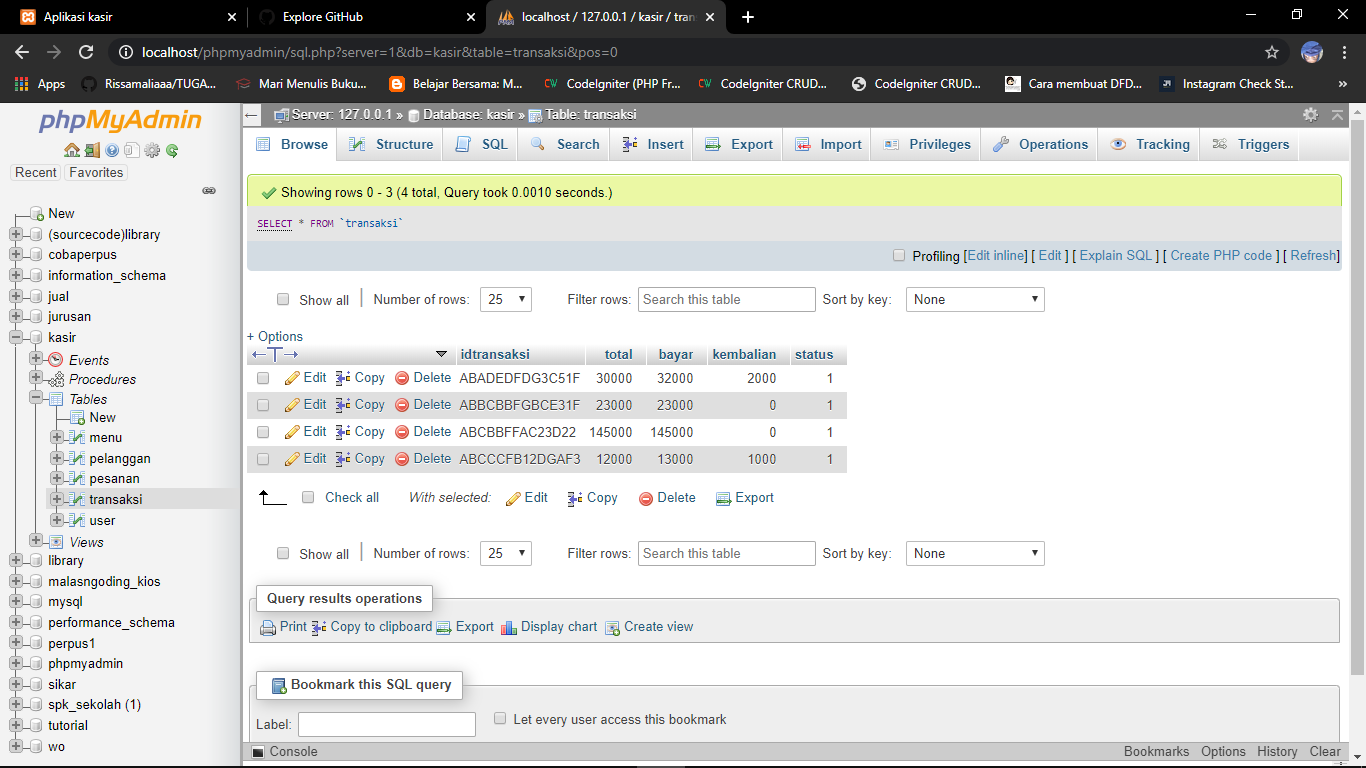
* **Tabel Pelanggan**



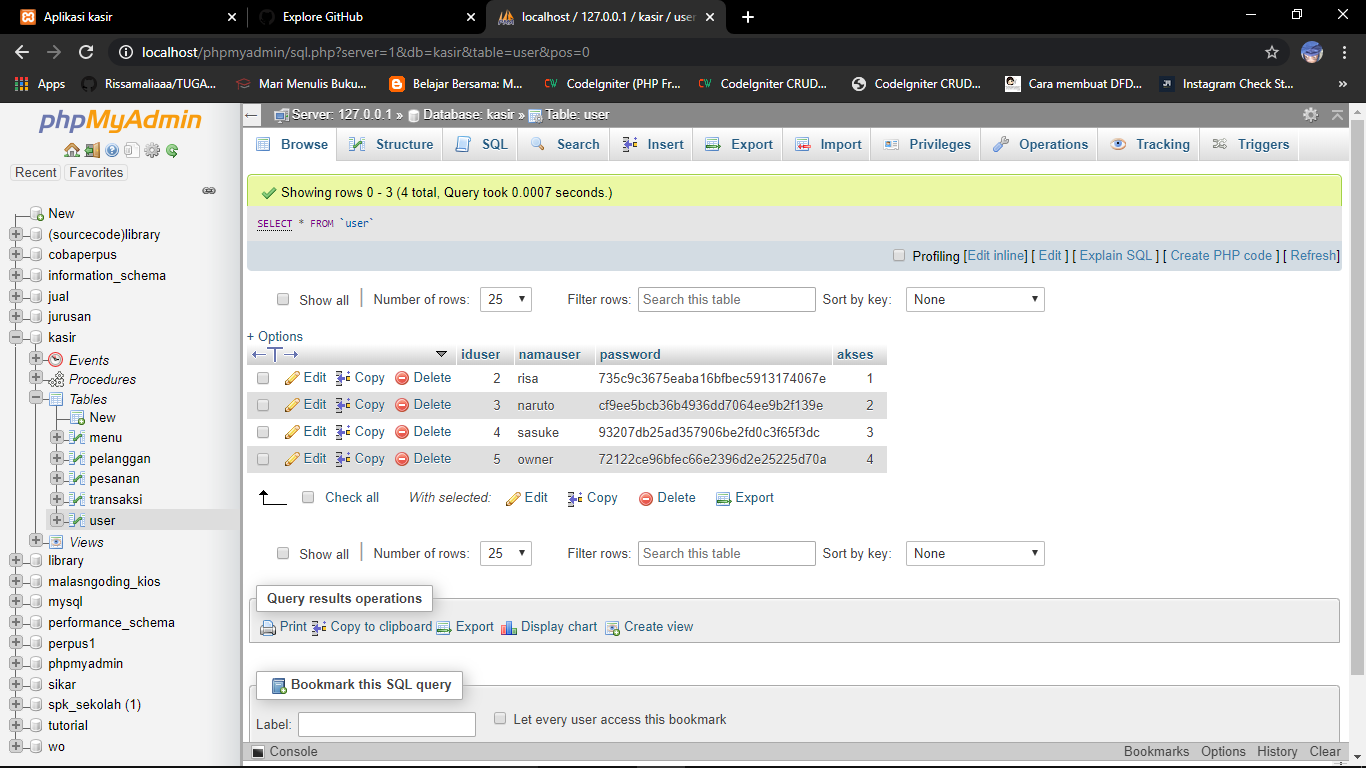
* **Tabel Pesanan**



* **Tabel Transaksi**



* **Tabel User Admin Restoran**



***CLASS DIAGRAM***

**SOURCE CODE**

1. ***Form Validation.php***

<?php

/\*\*

 \* CodeIgniter

 \*

 \* An open source application development framework for PHP

 \*

 \* This content is released under the MIT License (MIT)

 \*

 \* Copyright (c) 2014 - 2019, British Columbia Institute of Technology

 \*

 \* Permission is hereby granted, free of charge, to any person obtaining a copy

 \* of this software and associated documentation files (the "Software"), to deal

 \* in the Software without restriction, including without limitation the rights

 \* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

 \* copies of the Software, and to permit persons to whom the Software is

 \* furnished to do so, subject to the following conditions:

 \*

 \* The above copyright notice and this permission notice shall be included in

 \* all copies or substantial portions of the Software.

 \*

 \* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

 \* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

 \* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

 \* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

 \* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

 \* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

 \* THE SOFTWARE.

 \*

 \* @package CodeIgniter

 \* @author  EllisLab Dev Team

 \* @copyright   Copyright (c) 2008 - 2014, EllisLab, Inc. (https://ellislab.com/)

 \* @copyright   Copyright (c) 2014 - 2019, British Columbia Institute of Technology (https://bcit.ca/)

 \* @license https://opensource.org/licenses/MIT MIT License

 \* @link    https://codeigniter.com

 \* @since   Version 1.0.0

 \* @filesource

 \*/

defined('BASEPATH') OR exit('No direct script access allowed');

/\*\*

 \* Form Validation Class

 \*

 \* @package     CodeIgniter

 \* @subpackage  Libraries

 \* @category    Validation

 \* @author      EllisLab Dev Team

 \* @link        https://codeigniter.com/user\_guide/libraries/form\_validation.html

 \*/

class CI\_Form\_validation {

    /\*\*

     \* Reference to the CodeIgniter instance

     \*

     \* @var object

     \*/

    protected $CI;

    /\*\*

     \* Validation data for the current form submission

     \*

     \* @var array

     \*/

    protected $\_field\_data      = array();

    /\*\*

     \* Validation rules for the current form

     \*

     \* @var array

     \*/

    protected $\_config\_rules    = array();

    /\*\*

     \* Array of validation errors

     \*

     \* @var array

     \*/

    protected $\_error\_array     = array();

    /\*\*

     \* Array of custom error messages

     \*

     \* @var array

     \*/

    protected $\_error\_messages  = array();

    /\*\*

     \* Start tag for error wrapping

     \*

     \* @var string

     \*/

    protected $\_error\_prefix    = '<p>';

    /\*\*

     \* End tag for error wrapping

     \*

     \* @var string

     \*/

    protected $\_error\_suffix    = '</p>';

    /\*\*

     \* Custom error message

     \*

     \* @var string

     \*/

    protected $error\_string     = '';

    /\*\*

     \* Whether the form data has been validated as safe

     \*

     \* @var bool

     \*/

    protected $\_safe\_form\_data  = FALSE;

    /\*\*

     \* Custom data to validate

     \*

     \* @var array

     \*/

    public $validation\_data = array();

    /\*\*

     \* Initialize Form\_Validation class

     \*

     \* @param   array   $rules

     \* @return  void

     \*/

    public function \_\_construct($rules = array())

    {

        $this->CI =& get\_instance();

        // applies delimiters set in config file.

        if (isset($rules['error\_prefix']))

        {

            $this->\_error\_prefix = $rules['error\_prefix'];

            unset($rules['error\_prefix']);

        }

        if (isset($rules['error\_suffix']))

        {

            $this->\_error\_suffix = $rules['error\_suffix'];

            unset($rules['error\_suffix']);

        }

        // Validation rules can be stored in a config file.

        $this->\_config\_rules = $rules;

        // Automatically load the form helper

        $this->CI->load->helper('form');

        log\_message('info', 'Form Validation Class Initialized');

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set Rules

     \*

     \* This function takes an array of field names and validation

     \* rules as input, any custom error messages, validates the info,

     \* and stores it

     \*

     \* @param   mixed   $field

     \* @param   string  $label

     \* @param   mixed   $rules

     \* @param   array   $errors

     \* @return  CI\_Form\_validation

     \*/

    public function set\_rules($field, $label = '', $rules = array(), $errors = array())

    {

        // No reason to set rules if we have no POST data

        // or a validation array has not been specified

        if ($this->CI->input->method() !== 'post' && empty($this->validation\_data))

        {

            return $this;

        }

        // If an array was passed via the first parameter instead of individual string

        // values we cycle through it and recursively call this function.

        if (is\_array($field))

        {

            foreach ($field as $row)

            {

                // Houston, we have a problem...

                if ( ! isset($row['field'], $row['rules']))

                {

                    continue;

                }

                // If the field label wasn't passed we use the field name

                $label = isset($row['label']) ? $row['label'] : $row['field'];

                // Add the custom error message array

                $errors = (isset($row['errors']) && is\_array($row['errors'])) ? $row['errors'] : array();

                // Here we go!

                $this->set\_rules($row['field'], $label, $row['rules'], $errors);

            }

            return $this;

        }

        // No fields or no rules? Nothing to do...

        if ( ! is\_string($field) OR $field === '' OR empty($rules))

        {

            return $this;

        }

        elseif ( ! is\_array($rules))

        {

            // BC: Convert pipe-separated rules string to an array

            if ( ! is\_string($rules))

            {

                return $this;

            }

            $rules = preg\_split('/\|(?![^\[]\*\])/', $rules);

        }

        // If the field label wasn't passed we use the field name

        $label = ($label === '') ? $field : $label;

        $indexes = array();

        // Is the field name an array? If it is an array, we break it apart

        // into its components so that we can fetch the corresponding POST data later

        if (($is\_array = (bool) preg\_match\_all('/\[(.\*?)\]/', $field, $matches)) === TRUE)

        {

            sscanf($field, '%[^[][', $indexes[0]);

            for ($i = 0, $c = count($matches[0]); $i < $c; $i++)

            {

                if ($matches[1][$i] !== '')

                {

                    $indexes[] = $matches[1][$i];

                }

            }

        }

        // Build our master array

        $this->\_field\_data[$field] = array(

            'field'     => $field,

            'label'     => $label,

            'rules'     => $rules,

            'errors'    => $errors,

            'is\_array'  => $is\_array,

            'keys'      => $indexes,

            'postdata'  => NULL,

            'error'     => ''

        );

        return $this;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* By default, form validation uses the $\_POST array to validate

     \*

     \* If an array is set through this method, then this array will

     \* be used instead of the $\_POST array

     \*

     \* Note that if you are validating multiple arrays, then the

     \* reset\_validation() function should be called after validating

     \* each array due to the limitations of CI's singleton

     \*

     \* @param   array   $data

     \* @return  CI\_Form\_validation

     \*/

    public function set\_data(array $data)

    {

        if ( ! empty($data))

        {

            $this->validation\_data = $data;

        }

        return $this;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set Error Message

     \*

     \* Lets users set their own error messages on the fly. Note:

     \* The key name has to match the function name that it corresponds to.

     \*

     \* @param   array

     \* @param   string

     \* @return  CI\_Form\_validation

     \*/

    public function set\_message($lang, $val = '')

    {

        if ( ! is\_array($lang))

        {

            $lang = array($lang => $val);

        }

        $this->\_error\_messages = array\_merge($this->\_error\_messages, $lang);

        return $this;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set The Error Delimiter

     \*

     \* Permits a prefix/suffix to be added to each error message

     \*

     \* @param   string

     \* @param   string

     \* @return  CI\_Form\_validation

     \*/

    public function set\_error\_delimiters($prefix = '<p>', $suffix = '</p>')

    {

        $this->\_error\_prefix = $prefix;

        $this->\_error\_suffix = $suffix;

        return $this;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Get Error Message

     \*

     \* Gets the error message associated with a particular field

     \*

     \* @param   string  $field  Field name

     \* @param   string  $prefix HTML start tag

     \* @param   string  $suffix HTML end tag

     \* @return  string

     \*/

    public function error($field, $prefix = '', $suffix = '')

    {

        if (empty($this->\_field\_data[$field]['error']))

        {

            return '';

        }

        if ($prefix === '')

        {

            $prefix = $this->\_error\_prefix;

        }

        if ($suffix === '')

        {

            $suffix = $this->\_error\_suffix;

        }

        return $prefix.$this->\_field\_data[$field]['error'].$suffix;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Get Array of Error Messages

     \*

     \* Returns the error messages as an array

     \*

     \* @return  array

     \*/

    public function error\_array()

    {

        return $this->\_error\_array;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Error String

     \*

     \* Returns the error messages as a string, wrapped in the error delimiters

     \*

     \* @param   string

     \* @param   string

     \* @return  string

     \*/

    public function error\_string($prefix = '', $suffix = '')

    {

        // No errors, validation passes!

        if (count($this->\_error\_array) === 0)

        {

            return '';

        }

        if ($prefix === '')

        {

            $prefix = $this->\_error\_prefix;

        }

        if ($suffix === '')

        {

            $suffix = $this->\_error\_suffix;

        }

        // Generate the error string

        $str = '';

        foreach ($this->\_error\_array as $val)

        {

            if ($val !== '')

            {

                $str .= $prefix.$val.$suffix."\n";

            }

        }

        return $str;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Run the Validator

     \*

     \* This function does all the work.

     \*

     \* @param   string  $group

     \* @return  bool

     \*/

    public function run($group = '')

    {

        $validation\_array = empty($this->validation\_data)

            ? $\_POST

            : $this->validation\_data;

        // Does the \_field\_data array containing the validation rules exist?

        // If not, we look to see if they were assigned via a config file

        if (count($this->\_field\_data) === 0)

        {

            // No validation rules?  We're done...

            if (count($this->\_config\_rules) === 0)

            {

                return FALSE;

            }

            if (empty($group))

            {

                // Is there a validation rule for the particular URI being accessed?

                $group = trim($this->CI->uri->ruri\_string(), '/');

                isset($this->\_config\_rules[$group]) OR $group = $this->CI->router->class.'/'.$this->CI->router->method;

            }

            $this->set\_rules(isset($this->\_config\_rules[$group]) ? $this->\_config\_rules[$group] : $this->\_config\_rules);

            // Were we able to set the rules correctly?

            if (count($this->\_field\_data) === 0)

            {

                log\_message('debug', 'Unable to find validation rules');

                return FALSE;

            }

        }

        // Load the language file containing error messages

        $this->CI->lang->load('form\_validation');

        // Cycle through the rules for each field and match the corresponding $validation\_data item

        foreach ($this->\_field\_data as $field => &$row)

        {

            // Fetch the data from the validation\_data array item and cache it in the \_field\_data array.

            // Depending on whether the field name is an array or a string will determine where we get it from.

            if ($row['is\_array'] === TRUE)

            {

                $this->\_field\_data[$field]['postdata'] = $this->\_reduce\_array($validation\_array, $row['keys']);

            }

            elseif (isset($validation\_array[$field]))

            {

                $this->\_field\_data[$field]['postdata'] = $validation\_array[$field];

            }

        }

        // Execute validation rules

        // Note: A second foreach (for now) is required in order to avoid false-positives

        //   for rules like 'matches', which correlate to other validation fields.

        foreach ($this->\_field\_data as $field => &$row)

        {

            // Don't try to validate if we have no rules set

            if (empty($row['rules']))

            {

                continue;

            }

            $this->\_execute($row, $row['rules'], $row['postdata']);

        }

        // Did we end up with any errors?

        $total\_errors = count($this->\_error\_array);

        if ($total\_errors > 0)

        {

            $this->\_safe\_form\_data = TRUE;

        }

        // Now we need to re-set the POST data with the new, processed data

        empty($this->validation\_data) && $this->\_reset\_post\_array();

        return ($total\_errors === 0);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Prepare rules

     \*

     \* Re-orders the provided rules in order of importance, so that

     \* they can easily be executed later without weird checks ...

     \*

     \* "Callbacks" are given the highest priority (always called),

     \* followed by 'required' (called if callbacks didn't fail),

     \* and then every next rule depends on the previous one passing.

     \*

     \* @param   array   $rules

     \* @return  array

     \*/

    protected function \_prepare\_rules($rules)

    {

        $new\_rules = array();

        $callbacks = array();

        foreach ($rules as &$rule)

        {

            // Let 'required' always be the first (non-callback) rule

            if ($rule === 'required')

            {

                array\_unshift($new\_rules, 'required');

            }

            // 'isset' is a kind of a weird alias for 'required' ...

            elseif ($rule === 'isset' && (empty($new\_rules) OR $new\_rules[0] !== 'required'))

            {

                array\_unshift($new\_rules, 'isset');

            }

            // The old/classic 'callback\_'-prefixed rules

            elseif (is\_string($rule) && strncmp('callback\_', $rule, 9) === 0)

            {

                $callbacks[] = $rule;

            }

            // Proper callables

            elseif (is\_callable($rule))

            {

                $callbacks[] = $rule;

            }

            // "Named" callables; i.e. array('name' => $callable)

            elseif (is\_array($rule) && isset($rule[0], $rule[1]) && is\_callable($rule[1]))

            {

                $callbacks[] = $rule;

            }

            // Everything else goes at the end of the queue

            else

            {

                $new\_rules[] = $rule;

            }

        }

        return array\_merge($callbacks, $new\_rules);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Traverse a multidimensional $\_POST array index until the data is found

     \*

     \* @param   array

     \* @param   array

     \* @param   int

     \* @return  mixed

     \*/

    protected function \_reduce\_array($array, $keys, $i = 0)

    {

        if (is\_array($array) && isset($keys[$i]))

        {

            return isset($array[$keys[$i]]) ? $this->\_reduce\_array($array[$keys[$i]], $keys, ($i+1)) : NULL;

        }

        // NULL must be returned for empty fields

        return ($array === '') ? NULL : $array;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Re-populate the \_POST array with our finalized and processed data

     \*

     \* @return  void

     \*/

    protected function \_reset\_post\_array()

    {

        foreach ($this->\_field\_data as $field => $row)

        {

            if ($row['postdata'] !== NULL)

            {

                if ($row['is\_array'] === FALSE)

                {

                    isset($\_POST[$field]) && $\_POST[$field] = is\_array($row['postdata']) ? NULL : $row['postdata'];

                }

                else

                {

                    // start with a reference

                    $post\_ref =& $\_POST;

                    // before we assign values, make a reference to the right POST key

                    if (count($row['keys']) === 1)

                    {

                        $post\_ref =& $post\_ref[current($row['keys'])];

                    }

                    else

                    {

                        foreach ($row['keys'] as $val)

                        {

                            $post\_ref =& $post\_ref[$val];

                        }

                    }

                    $post\_ref = $row['postdata'];

                }

            }

        }

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Executes the Validation routines

     \*

     \* @param   array

     \* @param   array

     \* @param   mixed

     \* @param   int

     \* @return  mixed

     \*/

    protected function \_execute($row, $rules, $postdata = NULL, $cycles = 0)

    {

        // If the $\_POST data is an array we will run a recursive call

        //

        // Note: We MUST check if the array is empty or not!

        //       Otherwise empty arrays will always pass validation.

        if (is\_array($postdata) && ! empty($postdata))

        {

            foreach ($postdata as $key => $val)

            {

                $this->\_execute($row, $rules, $val, $key);

            }

            return;

        }

        $rules = $this->\_prepare\_rules($rules);

        foreach ($rules as $rule)

        {

            $\_in\_array = FALSE;

            // We set the $postdata variable with the current data in our master array so that

            // each cycle of the loop is dealing with the processed data from the last cycle

            if ($row['is\_array'] === TRUE && is\_array($this->\_field\_data[$row['field']]['postdata']))

            {

                // We shouldn't need this safety, but just in case there isn't an array index

                // associated with this cycle we'll bail out

                if ( ! isset($this->\_field\_data[$row['field']]['postdata'][$cycles]))

                {

                    continue;

                }

                $postdata = $this->\_field\_data[$row['field']]['postdata'][$cycles];

                $\_in\_array = TRUE;

            }

            else

            {

                // If we get an array field, but it's not expected - then it is most likely

                // somebody messing with the form on the client side, so we'll just consider

                // it an empty field

                $postdata = is\_array($this->\_field\_data[$row['field']]['postdata'])

                    ? NULL

                    : $this->\_field\_data[$row['field']]['postdata'];

            }

            // Is the rule a callback?

            $callback = $callable = FALSE;

            if (is\_string($rule))

            {

                if (strpos($rule, 'callback\_') === 0)

                {

                    $rule = substr($rule, 9);

                    $callback = TRUE;

                }

            }

            elseif (is\_callable($rule))

            {

                $callable = TRUE;

            }

            elseif (is\_array($rule) && isset($rule[0], $rule[1]) && is\_callable($rule[1]))

            {

                // We have a "named" callable, so save the name

                $callable = $rule[0];

                $rule = $rule[1];

            }

            // Strip the parameter (if exists) from the rule

            // Rules can contain a parameter: max\_length[5]

            $param = FALSE;

            if ( ! $callable && preg\_match('/(.\*?)\[(.\*)\]/', $rule, $match))

            {

                $rule = $match[1];

                $param = $match[2];

            }

            // Ignore empty, non-required inputs with a few exceptions ...

            if (

                ($postdata === NULL OR $postdata === '')

                && $callback === FALSE

                && $callable === FALSE

                && ! in\_array($rule, array('required', 'isset', 'matches'), TRUE)

            )

            {

                continue;

            }

            // Call the function that corresponds to the rule

            if ($callback OR $callable !== FALSE)

            {

                if ($callback)

                {

                    if ( ! method\_exists($this->CI, $rule))

                    {

                        log\_message('debug', 'Unable to find callback validation rule: '.$rule);

                        $result = FALSE;

                    }

                    else

                    {

                        // Run the function and grab the result

                        $result = $this->CI->$rule($postdata, $param);

                    }

                }

                else

                {

                    $result = is\_array($rule)

                        ? $rule[0]->{$rule[1]}($postdata)

                        : $rule($postdata);

                    // Is $callable set to a rule name?

                    if ($callable !== FALSE)

                    {

                        $rule = $callable;

                    }

                }

                // Re-assign the result to the master data array

                if ($\_in\_array === TRUE)

                {

                    $this->\_field\_data[$row['field']]['postdata'][$cycles] = is\_bool($result) ? $postdata : $result;

                }

                else

                {

                    $this->\_field\_data[$row['field']]['postdata'] = is\_bool($result) ? $postdata : $result;

                }

            }

            elseif ( ! method\_exists($this, $rule))

            {

                // If our own wrapper function doesn't exist we see if a native PHP function does.

                // Users can use any native PHP function call that has one param.

                if (function\_exists($rule))

                {

                    // Native PHP functions issue warnings if you pass them more parameters than they use

                    $result = ($param !== FALSE) ? $rule($postdata, $param) : $rule($postdata);

                    if ($\_in\_array === TRUE)

                    {

                        $this->\_field\_data[$row['field']]['postdata'][$cycles] = is\_bool($result) ? $postdata : $result;

                    }

                    else

                    {

                        $this->\_field\_data[$row['field']]['postdata'] = is\_bool($result) ? $postdata : $result;

                    }

                }

                else

                {

                    log\_message('debug', 'Unable to find validation rule: '.$rule);

                    $result = FALSE;

                }

            }

            else

            {

                $result = $this->$rule($postdata, $param);

                if ($\_in\_array === TRUE)

                {

                    $this->\_field\_data[$row['field']]['postdata'][$cycles] = is\_bool($result) ? $postdata : $result;

                }

                else

                {

                    $this->\_field\_data[$row['field']]['postdata'] = is\_bool($result) ? $postdata : $result;

                }

            }

            // Did the rule test negatively? If so, grab the error.

            if ($result === FALSE)

            {

                // Callable rules might not have named error messages

                if ( ! is\_string($rule))

                {

                    $line = $this->CI->lang->line('form\_validation\_error\_message\_not\_set').'(Anonymous function)';

                }

                else

                {

                    $line = $this->\_get\_error\_message($rule, $row['field']);

                }

                // Is the parameter we are inserting into the error message the name

                // of another field? If so we need to grab its "field label"

                if (isset($this->\_field\_data[$param], $this->\_field\_data[$param]['label']))

                {

                    $param = $this->\_translate\_fieldname($this->\_field\_data[$param]['label']);

                }

                // Build the error message

                $message = $this->\_build\_error\_msg($line, $this->\_translate\_fieldname($row['label']), $param);

                // Save the error message

                $this->\_field\_data[$row['field']]['error'] = $message;

                if ( ! isset($this->\_error\_array[$row['field']]))

                {

                    $this->\_error\_array[$row['field']] = $message;

                }

                return;

            }

        }

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Get the error message for the rule

     \*

     \* @param   string $rule    The rule name

     \* @param   string $field   The field name

     \* @return  string

     \*/

    protected function \_get\_error\_message($rule, $field)

    {

        // check if a custom message is defined through validation config row.

        if (isset($this->\_field\_data[$field]['errors'][$rule]))

        {

            return $this->\_field\_data[$field]['errors'][$rule];

        }

        // check if a custom message has been set using the set\_message() function

        elseif (isset($this->\_error\_messages[$rule]))

        {

            return $this->\_error\_messages[$rule];

        }

        elseif (FALSE !== ($line = $this->CI->lang->line('form\_validation\_'.$rule)))

        {

            return $line;

        }

        // DEPRECATED support for non-prefixed keys, lang file again

        elseif (FALSE !== ($line = $this->CI->lang->line($rule, FALSE)))

        {

            return $line;

        }

        return $this->CI->lang->line('form\_validation\_error\_message\_not\_set').'('.$rule.')';

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Translate a field name

     \*

     \* @param   string  the field name

     \* @return  string

     \*/

    protected function \_translate\_fieldname($fieldname)

    {

        // Do we need to translate the field name? We look for the prefix 'lang:' to determine this

        // If we find one, but there's no translation for the string - just return it

        if (sscanf($fieldname, 'lang:%s', $line) === 1 && FALSE === ($fieldname = $this->CI->lang->line($line, FALSE)))

        {

            return $line;

        }

        return $fieldname;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Build an error message using the field and param.

     \*

     \* @param   string  The error message line

     \* @param   string  A field's human name

     \* @param   mixed   A rule's optional parameter

     \* @return  string

     \*/

    protected function \_build\_error\_msg($line, $field = '', $param = '')

    {

        // Check for %s in the string for legacy support.

        if (strpos($line, '%s') !== FALSE)

        {

            return sprintf($line, $field, $param);

        }

        return str\_replace(array('{field}', '{param}'), array($field, $param), $line);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Checks if the rule is present within the validator

     \*

     \* Permits you to check if a rule is present within the validator

     \*

     \* @param   string  the field name

     \* @return  bool

     \*/

    public function has\_rule($field)

    {

        return isset($this->\_field\_data[$field]);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Get the value from a form

     \*

     \* Permits you to repopulate a form field with the value it was submitted

     \* with, or, if that value doesn't exist, with the default

     \*

     \* @param   string  the field name

     \* @param   string

     \* @return  string

     \*/

    public function set\_value($field = '', $default = '')

    {

        if ( ! isset($this->\_field\_data[$field], $this->\_field\_data[$field]['postdata']))

        {

            return $default;

        }

        // If the data is an array output them one at a time.

        //  E.g: form\_input('name[]', set\_value('name[]');

        if (is\_array($this->\_field\_data[$field]['postdata']))

        {

            return array\_shift($this->\_field\_data[$field]['postdata']);

        }

        return $this->\_field\_data[$field]['postdata'];

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set Select

     \*

     \* Enables pull-down lists to be set to the value the user

     \* selected in the event of an error

     \*

     \* @param   string

     \* @param   string

     \* @param   bool

     \* @return  string

     \*/

    public function set\_select($field = '', $value = '', $default = FALSE)

    {

        if ( ! isset($this->\_field\_data[$field], $this->\_field\_data[$field]['postdata']))

        {

            return ($default === TRUE && count($this->\_field\_data) === 0) ? ' selected="selected"' : '';

        }

        $field = $this->\_field\_data[$field]['postdata'];

        $value = (string) $value;

        if (is\_array($field))

        {

            // Note: in\_array('', array(0)) returns TRUE, do not use it

            foreach ($field as &$v)

            {

                if ($value === $v)

                {

                    return ' selected="selected"';

                }

            }

            return '';

        }

        elseif (($field === '' OR $value === '') OR ($field !== $value))

        {

            return '';

        }

        return ' selected="selected"';

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set Radio

     \*

     \* Enables radio buttons to be set to the value the user

     \* selected in the event of an error

     \*

     \* @param   string

     \* @param   string

     \* @param   bool

     \* @return  string

     \*/

    public function set\_radio($field = '', $value = '', $default = FALSE)

    {

        if ( ! isset($this->\_field\_data[$field], $this->\_field\_data[$field]['postdata']))

        {

            return ($default === TRUE && count($this->\_field\_data) === 0) ? ' checked="checked"' : '';

        }

        $field = $this->\_field\_data[$field]['postdata'];

        $value = (string) $value;

        if (is\_array($field))

        {

            // Note: in\_array('', array(0)) returns TRUE, do not use it

            foreach ($field as &$v)

            {

                if ($value === $v)

                {

                    return ' checked="checked"';

                }

            }

            return '';

        }

        elseif (($field === '' OR $value === '') OR ($field !== $value))

        {

            return '';

        }

        return ' checked="checked"';

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Set Checkbox

     \*

     \* Enables checkboxes to be set to the value the user

     \* selected in the event of an error

     \*

     \* @param   string

     \* @param   string

     \* @param   bool

     \* @return  string

     \*/

    public function set\_checkbox($field = '', $value = '', $default = FALSE)

    {

        // Logic is exactly the same as for radio fields

        return $this->set\_radio($field, $value, $default);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Required

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function required($str)

    {

        return is\_array($str)

            ? (empty($str) === FALSE)

            : (trim($str) !== '');

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Performs a Regular Expression match test.

     \*

     \* @param   string

     \* @param   string  regex

     \* @return  bool

     \*/

    public function regex\_match($str, $regex)

    {

        return (bool) preg\_match($regex, $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Match one field to another

     \*

     \* @param   string  $str    string to compare against

     \* @param   string  $field

     \* @return  bool

     \*/

    public function matches($str, $field)

    {

        return isset($this->\_field\_data[$field], $this->\_field\_data[$field]['postdata'])

            ? ($str === $this->\_field\_data[$field]['postdata'])

            : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Differs from another field

     \*

     \* @param   string

     \* @param   string  field

     \* @return  bool

     \*/

    public function differs($str, $field)

    {

        return ! (isset($this->\_field\_data[$field]) && $this->\_field\_data[$field]['postdata'] === $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Is Unique

     \*

     \* Check if the input value doesn't already exist

     \* in the specified database field.

     \*

     \* @param   string  $str

     \* @param   string  $field

     \* @return  bool

     \*/

    public function is\_unique($str, $field)

    {

        sscanf($field, '%[^.].%[^.]', $table, $field);

        return isset($this->CI->db)

            ? ($this->CI->db->limit(1)->get\_where($table, array($field => $str))->num\_rows() === 0)

            : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Minimum Length

     \*

     \* @param   string

     \* @param   string

     \* @return  bool

     \*/

    public function min\_length($str, $val)

    {

        if ( ! is\_numeric($val))

        {

            return FALSE;

        }

        return ($val <= mb\_strlen($str));

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Max Length

     \*

     \* @param   string

     \* @param   string

     \* @return  bool

     \*/

    public function max\_length($str, $val)

    {

        if ( ! is\_numeric($val))

        {

            return FALSE;

        }

        return ($val >= mb\_strlen($str));

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Exact Length

     \*

     \* @param   string

     \* @param   string

     \* @return  bool

     \*/

    public function exact\_length($str, $val)

    {

        if ( ! is\_numeric($val))

        {

            return FALSE;

        }

        return (mb\_strlen($str) === (int) $val);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Valid URL

     \*

     \* @param   string  $str

     \* @return  bool

     \*/

    public function valid\_url($str)

    {

        if (empty($str))

        {

            return FALSE;

        }

        elseif (preg\_match('/^(?:([^:]\*)\:)?\/\/(.+)$/', $str, $matches))

        {

            if (empty($matches[2]))

            {

                return FALSE;

            }

            elseif ( ! in\_array(strtolower($matches[1]), array('http', 'https'), TRUE))

            {

                return FALSE;

            }

            $str = $matches[2];

        }

        // PHP 7 accepts IPv6 addresses within square brackets as hostnames,

        // but it appears that the PR that came in with https://bugs.php.net/bug.php?id=68039

        // was never merged into a PHP 5 branch ... https://3v4l.org/8PsSN

        if (preg\_match('/^\[([^\]]+)\]/', $str, $matches) && ! is\_php('7') && filter\_var($matches[1], FILTER\_VALIDATE\_IP, FILTER\_FLAG\_IPV6) !== FALSE)

        {

            $str = 'ipv6.host'.substr($str, strlen($matches[1]) + 2);

        }

        return (filter\_var('http://'.$str, FILTER\_VALIDATE\_URL) !== FALSE);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Valid Email

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function valid\_email($str)

    {

        if (function\_exists('idn\_to\_ascii') && preg\_match('#\A([^@]+)@(.+)\z#', $str, $matches))

        {

            $domain = defined('INTL\_IDNA\_VARIANT\_UTS46')

                ? idn\_to\_ascii($matches[2], 0, INTL\_IDNA\_VARIANT\_UTS46)

                : idn\_to\_ascii($matches[2]);

            if ($domain !== FALSE)

            {

                $str = $matches[1].'@'.$domain;

            }

        }

        return (bool) filter\_var($str, FILTER\_VALIDATE\_EMAIL);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Valid Emails

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function valid\_emails($str)

    {

        if (strpos($str, ',') === FALSE)

        {

            return $this->valid\_email(trim($str));

        }

        foreach (explode(',', $str) as $email)

        {

            if (trim($email) !== '' && $this->valid\_email(trim($email)) === FALSE)

            {

                return FALSE;

            }

        }

        return TRUE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Validate IP Address

     \*

     \* @param   string

     \* @param   string  'ipv4' or 'ipv6' to validate a specific IP format

     \* @return  bool

     \*/

    public function valid\_ip($ip, $which = '')

    {

        return $this->CI->input->valid\_ip($ip, $which);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Alpha

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function alpha($str)

    {

        return ctype\_alpha($str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Alpha-numeric

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function alpha\_numeric($str)

    {

        return ctype\_alnum((string) $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Alpha-numeric w/ spaces

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function alpha\_numeric\_spaces($str)

    {

        return (bool) preg\_match('/^[A-Z0-9 ]+$/i', $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Alpha-numeric with underscores and dashes

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function alpha\_dash($str)

    {

        return (bool) preg\_match('/^[a-z0-9\_-]+$/i', $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Numeric

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function numeric($str)

    {

        return (bool) preg\_match('/^[\-+]?[0-9]\*\.?[0-9]+$/', $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Integer

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function integer($str)

    {

        return (bool) preg\_match('/^[\-+]?[0-9]+$/', $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Decimal number

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function decimal($str)

    {

        return (bool) preg\_match('/^[\-+]?[0-9]+\.[0-9]+$/', $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Greater than

     \*

     \* @param   string

     \* @param   int

     \* @return  bool

     \*/

    public function greater\_than($str, $min)

    {

        return is\_numeric($str) ? ($str > $min) : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Equal to or Greater than

     \*

     \* @param   string

     \* @param   int

     \* @return  bool

     \*/

    public function greater\_than\_equal\_to($str, $min)

    {

        return is\_numeric($str) ? ($str >= $min) : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Less than

     \*

     \* @param   string

     \* @param   int

     \* @return  bool

     \*/

    public function less\_than($str, $max)

    {

        return is\_numeric($str) ? ($str < $max) : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Equal to or Less than

     \*

     \* @param   string

     \* @param   int

     \* @return  bool

     \*/

    public function less\_than\_equal\_to($str, $max)

    {

        return is\_numeric($str) ? ($str <= $max) : FALSE;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Value should be within an array of values

     \*

     \* @param   string

     \* @param   string

     \* @return  bool

     \*/

    public function in\_list($value, $list)

    {

        return in\_array($value, explode(',', $list), TRUE);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Is a Natural number  (0,1,2,3, etc.)

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function is\_natural($str)

    {

        return ctype\_digit((string) $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Is a Natural number, but not a zero  (1,2,3, etc.)

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function is\_natural\_no\_zero($str)

    {

        return ($str != 0 && ctype\_digit((string) $str));

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Valid Base64

     \*

     \* Tests a string for characters outside of the Base64 alphabet

     \* as defined by RFC 2045 http://www.faqs.org/rfcs/rfc2045

     \*

     \* @param   string

     \* @return  bool

     \*/

    public function valid\_base64($str)

    {

        return (base64\_encode(base64\_decode($str)) === $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Prep data for form

     \*

     \* This function allows HTML to be safely shown in a form.

     \* Special characters are converted.

     \*

     \* @deprecated  3.0.6   Not used anywhere within the framework and pretty much useless

     \* @param   mixed   $data   Input data

     \* @return  mixed

     \*/

    public function prep\_for\_form($data)

    {

        if ($this->\_safe\_form\_data === FALSE OR empty($data))

        {

            return $data;

        }

        if (is\_array($data))

        {

            foreach ($data as $key => $val)

            {

                $data[$key] = $this->prep\_for\_form($val);

            }

            return $data;

        }

        return str\_replace(array("'", '"', '<', '>'), array('&#39;', '&quot;', '&lt;', '&gt;'), stripslashes($data));

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Prep URL

     \*

     \* @param   string

     \* @return  string

     \*/

    public function prep\_url($str = '')

    {

        if ($str === 'http://' OR $str === '')

        {

            return '';

        }

        if (strpos($str, 'http://') !== 0 && strpos($str, 'https://') !== 0)

        {

            return 'http://'.$str;

        }

        return $str;

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Strip Image Tags

     \*

     \* @param   string

     \* @return  string

     \*/

    public function strip\_image\_tags($str)

    {

        return $this->CI->security->strip\_image\_tags($str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Convert PHP tags to entities

     \*

     \* @param   string

     \* @return  string

     \*/

    public function encode\_php\_tags($str)

    {

        return str\_replace(array('<?', '?>'), array('&lt;?', '?&gt;'), $str);

    }

    // --------------------------------------------------------------------

    /\*\*

     \* Reset validation vars

     \* Prevents subsequent validation routines from being affected by the

     \* results of any previous validation routine due to the CI singleton.

     \*

     \* @return  CI\_Form\_validation

     \*/

    public function reset\_validation()

    {

        $this->\_field\_data = array();

        $this->\_error\_array = array();

        $this->\_error\_messages = array();

        $this->error\_string = '';

        return $this; }}

1. ***Database.php***
2. <?php
3. defined('BASEPATH') OR exit('No direct script access allowed');
4. /\*
5. | -------------------------------------------------------------------
6. | DATABASE CONNECTIVITY SETTINGS
7. | -------------------------------------------------------------------
8. | This file will contain the settings needed to access your database.
9. |
10. | For complete instructions please consult the 'Database Connection'
11. | page of the User Guide.
12. |
13. | -------------------------------------------------------------------
14. | EXPLANATION OF VARIABLES
15. | -------------------------------------------------------------------
16. |
17. |   ['dsn']      The full DSN string describe a connection to the database.
18. |   ['hostname'] The hostname of your database server.
19. |   ['username'] The username used to connect to the database
20. |   ['password'] The password used to connect to the database
21. |   ['database'] The name of the database you want to connect to
22. |   ['dbdriver'] The database driver. e.g.: mysqli.
23. |           Currently supported:
24. |                cubrid, ibase, mssql, mysql, mysqli, oci8,
25. |                odbc, pdo, postgre, sqlite, sqlite3, sqlsrv
26. |   ['dbprefix'] You can add an optional prefix, which will be added
27. |                to the table name when using the  Query Builder class
28. |   ['pconnect'] TRUE/FALSE - Whether to use a persistent connection
29. |   ['db\_debug'] TRUE/FALSE - Whether database errors should be displayed.
30. |   ['cache\_on'] TRUE/FALSE - Enables/disables query caching
31. |   ['cachedir'] The path to the folder where cache files should be stored
32. |   ['char\_set'] The character set used in communicating with the database
33. |   ['dbcollat'] The character collation used in communicating with the database
34. |                NOTE: For MySQL and MySQLi databases, this setting is only used
35. |                as a backup if your server is running PHP < 5.2.3 or MySQL < 5.0.7
36. |                (and in table creation queries made with DB Forge).
37. |                There is an incompatibility in PHP with mysql\_real\_escape\_string() which
38. |                can make your site vulnerable to SQL injection if you are using a
39. |                multi-byte character set and are running versions lower than these.
40. |                Sites using Latin-1 or UTF-8 database character set and collation are unaffected.
41. |   ['swap\_pre'] A default table prefix that should be swapped with the dbprefix
42. |   ['encrypt']  Whether or not to use an encrypted connection.
43. |
44. |           'mysql' (deprecated), 'sqlsrv' and 'pdo/sqlsrv' drivers accept TRUE/FALSE
45. |           'mysqli' and 'pdo/mysql' drivers accept an array with the following options:
46. |
47. |               'ssl\_key'    - Path to the private key file
48. |               'ssl\_cert'   - Path to the public key certificate file
49. |               'ssl\_ca'     - Path to the certificate authority file
50. |               'ssl\_capath' - Path to a directory containing trusted CA certificates in PEM format
51. |               'ssl\_cipher' - List of \*allowed\* ciphers to be used for the encryption, separated by colons (':')
52. |               'ssl\_verify' - TRUE/FALSE; Whether verify the server certificate or not
53. |
54. |   ['compress'] Whether or not to use client compression (MySQL only)
55. |   ['stricton'] TRUE/FALSE - forces 'Strict Mode' connections
56. |                           - good for ensuring strict SQL while developing
57. |   ['ssl\_options'] Used to set various SSL options that can be used when making SSL connections.
58. |   ['failover'] array - A array with 0 or more data for connections if the main should fail.
59. |   ['save\_queries'] TRUE/FALSE - Whether to "save" all executed queries.
60. |               NOTE: Disabling this will also effectively disable both
61. |               $this->db->last\_query() and profiling of DB queries.
62. |               When you run a query, with this setting set to TRUE (default),
63. |               CodeIgniter will store the SQL statement for debugging purposes.
64. |               However, this may cause high memory usage, especially if you run
65. |               a lot of SQL queries ... disable this to avoid that problem.
66. |
67. | The $active\_group variable lets you choose which connection group to
68. | make active.  By default there is only one group (the 'default' group).
69. |
70. | The $query\_builder variables lets you determine whether or not to load
71. | the query builder class.
72. \*/
73. $active\_group = 'default';
74. $query\_builder = TRUE;
75. $db['default'] = array(
76. 'dsn'   => '',
77. 'hostname' => 'localhost',
78. 'username' => 'root',
79. 'password' => 'root',
80. 'database' => 'kasir',
81. 'dbdriver' => 'mysqli',
82. 'dbprefix' => '',
83. 'pconnect' => FALSE,
84. 'db\_debug' => (ENVIRONMENT !== 'production'),
85. 'cache\_on' => FALSE,
86. 'cachedir' => '',
87. 'char\_set' => 'utf8',
88. 'dbcollat' => 'utf8\_general\_ci',
89. 'swap\_pre' => '',
90. 'encrypt' => FALSE,
91. 'compress' => FALSE,
92. 'stricton' => FALSE,
93. 'failover' => array(),
94. 'save\_queries' => TRUE
95. );

***3. Index.php***

<?php

/\*\*

 \* CodeIgniter

 \*

 \* An open source application development framework for PHP

 \*

 \* This content is released under the MIT License (MIT)

 \*

 \* Copyright (c) 2014 - 2019, British Columbia Institute of Technology

 \*

 \* Permission is hereby granted, free of charge, to any person obtaining a copy

 \* of this software and associated documentation files (the "Software"), to deal

 \* in the Software without restriction, including without limitation the rights

 \* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

 \* copies of the Software, and to permit persons to whom the Software is

 \* furnished to do so, subject to the following conditions:

 \*

 \* The above copyright notice and this permission notice shall be included in

 \* all copies or substantial portions of the Software.

 \*

 \* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

 \* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

 \* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

 \* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

 \* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

 \* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

 \* THE SOFTWARE.

 \*

 \* @package CodeIgniter

 \* @author  EllisLab Dev Team

 \* @copyright   Copyright (c) 2008 - 2014, EllisLab, Inc. (https://ellislab.com/)

 \* @copyright   Copyright (c) 2014 - 2019, British Columbia Institute of Technology (https://bcit.ca/)

 \* @license https://opensource.org/licenses/MIT MIT License

 \* @link    https://codeigniter.com

 \* @since   Version 1.0.0

 \* @filesource

 \*/

/\*

 \*---------------------------------------------------------------

 \* APPLICATION ENVIRONMENT

 \*---------------------------------------------------------------

 \*

 \* You can load different configurations depending on your

 \* current environment. Setting the environment also influences

 \* things like logging and error reporting.

 \*

 \* This can be set to anything, but default usage is:

 \*

 \*     development

 \*     testing

 \*     production

 \*

 \* NOTE: If you change these, also change the error\_reporting() code below

 \*/

    define('ENVIRONMENT', isset($\_SERVER['CI\_ENV']) ? $\_SERVER['CI\_ENV'] : 'development');

/\*

 \*---------------------------------------------------------------

 \* ERROR REPORTING

 \*---------------------------------------------------------------

 \*

 \* Different environments will require different levels of error reporting.

 \* By default development will show errors but testing and live will hide them.

 \*/

switch (ENVIRONMENT)

{

    case 'development':

        error\_reporting(-1);

        ini\_set('display\_errors', 1);

    break;

    case 'testing':

    case 'production':

        ini\_set('display\_errors', 0);

        if (version\_compare(PHP\_VERSION, '5.3', '>='))

        {

            error\_reporting(E\_ALL & ~E\_NOTICE & ~E\_DEPRECATED & ~E\_STRICT & ~E\_USER\_NOTICE & ~E\_USER\_DEPRECATED);

        }

        else

        {

            error\_reporting(E\_ALL & ~E\_NOTICE & ~E\_STRICT & ~E\_USER\_NOTICE);

        }

    break;

    default:

        header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);

        echo 'The application environment is not set correctly.';

        exit(1); // EXIT\_ERROR

}

/\*

 \*---------------------------------------------------------------

 \* SYSTEM DIRECTORY NAME

 \*---------------------------------------------------------------

 \*

 \* This variable must contain the name of your "system" directory.

 \* Set the path if it is not in the same directory as this file.

 \*/

    $system\_path = 'system';

/\*

 \*---------------------------------------------------------------

 \* APPLICATION DIRECTORY NAME

 \*---------------------------------------------------------------

 \*

 \* If you want this front controller to use a different "application"

 \* directory than the default one you can set its name here. The directory

 \* can also be renamed or relocated anywhere on your server. If you do,

 \* use an absolute (full) server path.

 \* For more info please see the user guide:

 \*

 \* https://codeigniter.com/user\_guide/general/managing\_apps.html

 \*

 \* NO TRAILING SLASH!

 \*/

    $application\_folder = 'application';

/\*

 \*---------------------------------------------------------------

 \* VIEW DIRECTORY NAME

 \*---------------------------------------------------------------

 \*

 \* If you want to move the view directory out of the application

 \* directory, set the path to it here. The directory can be renamed

 \* and relocated anywhere on your server. If blank, it will default

 \* to the standard location inside your application directory.

 \* If you do move this, use an absolute (full) server path.

 \*

 \* NO TRAILING SLASH!

 \*/

    $view\_folder = '';

/\*

 \* --------------------------------------------------------------------

 \* DEFAULT CONTROLLER

 \* --------------------------------------------------------------------

 \*

 \* Normally you will set your default controller in the routes.php file.

 \* You can, however, force a custom routing by hard-coding a

 \* specific controller class/function here. For most applications, you

 \* WILL NOT set your routing here, but it's an option for those

 \* special instances where you might want to override the standard

 \* routing in a specific front controller that shares a common CI installation.

 \*

 \* IMPORTANT: If you set the routing here, NO OTHER controller will be

 \* callable. In essence, this preference limits your application to ONE

 \* specific controller. Leave the function name blank if you need

 \* to call functions dynamically via the URI.

 \*

 \* Un-comment the $routing array below to use this feature

 \*/

    // The directory name, relative to the "controllers" directory.  Leave blank

    // if your controller is not in a sub-directory within the "controllers" one

    // $routing['directory'] = '';

    // The controller class file name.  Example:  mycontroller

    // $routing['controller'] = '';

    // The controller function you wish to be called.

    // $routing['function'] = '';

/\*

 \* -------------------------------------------------------------------

 \*  CUSTOM CONFIG VALUES

 \* -------------------------------------------------------------------

 \*

 \* The $assign\_to\_config array below will be passed dynamically to the

 \* config class when initialized. This allows you to set custom config

 \* items or override any default config values found in the config.php file.

 \* This can be handy as it permits you to share one application between

 \* multiple front controller files, with each file containing different

 \* config values.

 \*

 \* Un-comment the $assign\_to\_config array below to use this feature

 \*/

    // $assign\_to\_config['name\_of\_config\_item'] = 'value of config item';

// --------------------------------------------------------------------

// END OF USER CONFIGURABLE SETTINGS.  DO NOT EDIT BELOW THIS LINE

// --------------------------------------------------------------------

/\*

 \* ---------------------------------------------------------------

 \*  Resolve the system path for increased reliability

 \* ---------------------------------------------------------------

 \*/

    // Set the current directory correctly for CLI requests

    if (defined('STDIN'))

    {

        chdir(dirname(\_\_FILE\_\_));

    }

    if (($\_temp = realpath($system\_path)) !== FALSE)

    {

        $system\_path = $\_temp.DIRECTORY\_SEPARATOR;

    }

    else

    {

        // Ensure there's a trailing slash

        $system\_path = strtr(

            rtrim($system\_path, '/\\'),

            '/\\',

            DIRECTORY\_SEPARATOR.DIRECTORY\_SEPARATOR

        ).DIRECTORY\_SEPARATOR;

    }

    // Is the system path correct?

    if ( ! is\_dir($system\_path))

    {

        header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);

        echo 'Your system folder path does not appear to be set correctly. Please open the following file and correct this: '.pathinfo(\_\_FILE\_\_, PATHINFO\_BASENAME);

        exit(3); // EXIT\_CONFIG

    }

/\*

 \* -------------------------------------------------------------------

 \*  Now that we know the path, set the main path constants

 \* -------------------------------------------------------------------

 \*/

    // The name of THIS file

    define('SELF', pathinfo(\_\_FILE\_\_, PATHINFO\_BASENAME));

    // Path to the system directory

    define('BASEPATH', $system\_path);

    // Path to the front controller (this file) directory

    define('FCPATH', dirname(\_\_FILE\_\_).DIRECTORY\_SEPARATOR);

    // Name of the "system" directory

    define('SYSDIR', basename(BASEPATH));

    // The path to the "application" directory

    if (is\_dir($application\_folder))

    {

        if (($\_temp = realpath($application\_folder)) !== FALSE)

        {

            $application\_folder = $\_temp;

        }

        else

        {

            $application\_folder = strtr(

                rtrim($application\_folder, '/\\'),

                '/\\',

                DIRECTORY\_SEPARATOR.DIRECTORY\_SEPARATOR

            );

        }

    }

    elseif (is\_dir(BASEPATH.$application\_folder.DIRECTORY\_SEPARATOR))

    {

        $application\_folder = BASEPATH.strtr(

            trim($application\_folder, '/\\'),

            '/\\',

            DIRECTORY\_SEPARATOR.DIRECTORY\_SEPARATOR

        );

    }

    else

    {

        header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);

        echo 'Your application folder path does not appear to be set correctly. Please open the following file and correct this: '.SELF;

        exit(3); // EXIT\_CONFIG

    }

    define('APPPATH', $application\_folder.DIRECTORY\_SEPARATOR);

    // The path to the "views" directory

    if ( ! isset($view\_folder[0]) && is\_dir(APPPATH.'views'.DIRECTORY\_SEPARATOR))

    {

        $view\_folder = APPPATH.'views';

    }

    elseif (is\_dir($view\_folder))

    {

        if (($\_temp = realpath($view\_folder)) !== FALSE)

        {

            $view\_folder = $\_temp;

        }

        else

        {

            $view\_folder = strtr(

                rtrim($view\_folder, '/\\'),

                '/\\',

                DIRECTORY\_SEPARATOR.DIRECTORY\_SEPARATOR

            );

        }

    }

    elseif (is\_dir(APPPATH.$view\_folder.DIRECTORY\_SEPARATOR))

    {

        $view\_folder = APPPATH.strtr(

            trim($view\_folder, '/\\'),

            '/\\',

            DIRECTORY\_SEPARATOR.DIRECTORY\_SEPARATOR

        );

    }

    else

    {

        header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);

        echo 'Your view folder path does not appear to be set correctly. Please open the following file and correct this: '.SELF;

        exit(3); // EXIT\_CONFIG

    }

    define('VIEWPATH', $view\_folder.DIRECTORY\_SEPARATOR);

/\*

 \* --------------------------------------------------------------------

 \* LOAD THE BOOTSTRAP FILE

 \* --------------------------------------------------------------------

 \*

 \* And away we go...

 \*/

require\_once BASEPATH.'core/CodeIgniter.php';