566 E-Guard Report

Ting Zhou

# Related Software and Libraries

1. MAMP or XAMPP is used as Apache server and MySql DB setup. The program under *E-Guard/E-Guard-Server* is running on the Apache Server under the directory *localhost/E-Guard/E-Guard-Server.*
2. I use the

2. Chrome.tabs is the main Chrome API I used to control the chrome's page. Every time user access to a new page, the extension's background.js will check if the URL is allowed.

## Installation Steps:

1. Create a DB in Phpmyadmin called e-guard. Import the \*DB.sql\* from the directory \*'{server root directory}/E-Guard/E-Guard-Database'\*

2. Load the extension directory "./E-Guard/E-Guard-Client" from the Chrome Extension Page.

3. You can click the E-Guard Extension -> Options to get to the configuration page.

4. You can uninstalll it by click the uninstall icon from the Chrome Extension page.

# User Interface

### Chrome Extension Popup Page

This is the Client side of my E-Guard chrome extension. I implemented 3 visualization pages for visualizing the time spent on visited websites, Bar Chart, Table, Chart.

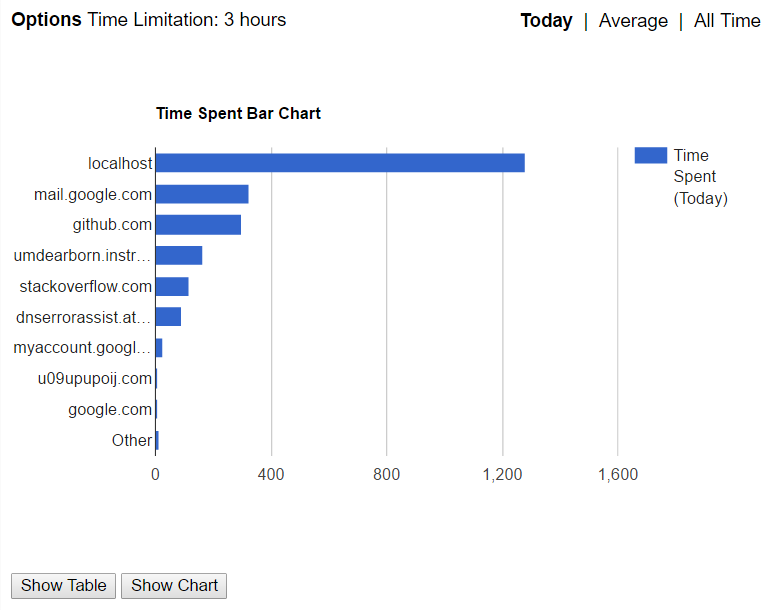


Figure Bar Table

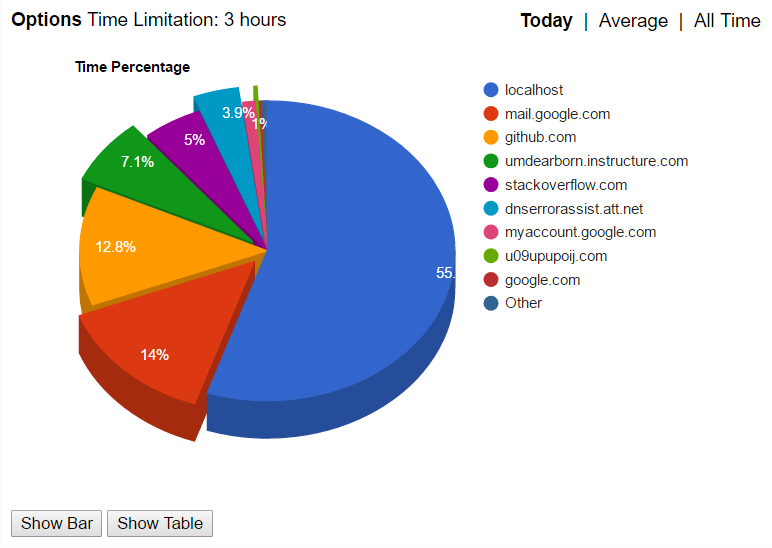


Figure Chart

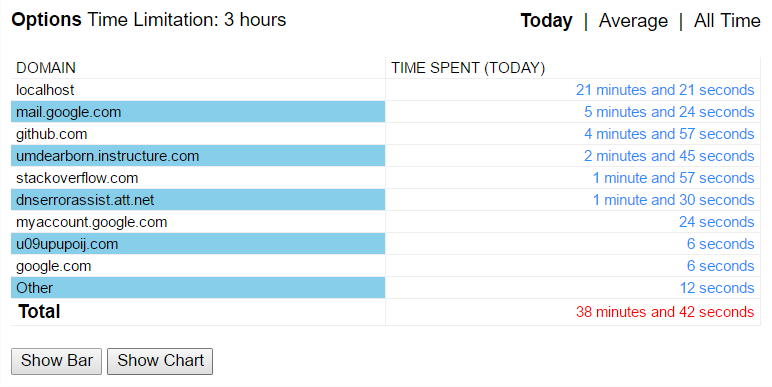


Figure Table

### Configuration Page

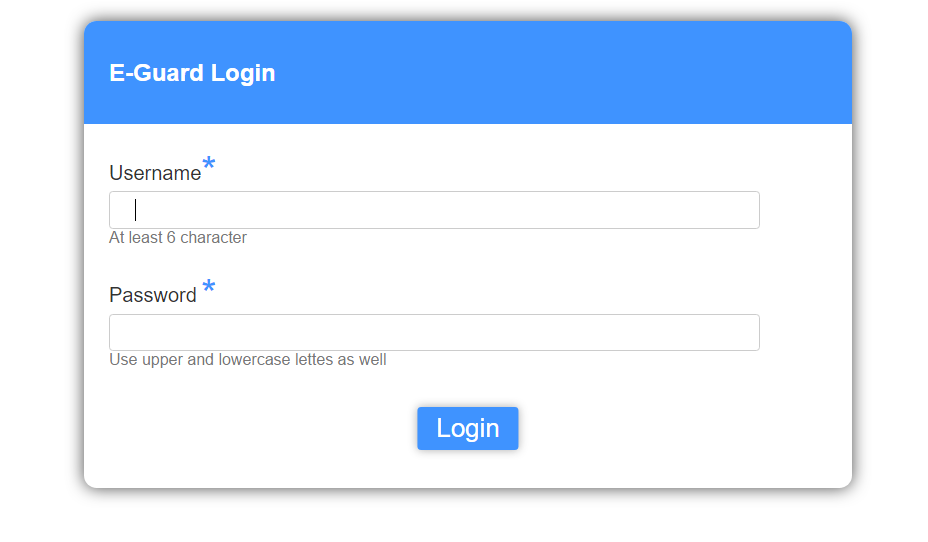


Figure User Login

Username: ztlevi

Password: ztlevi

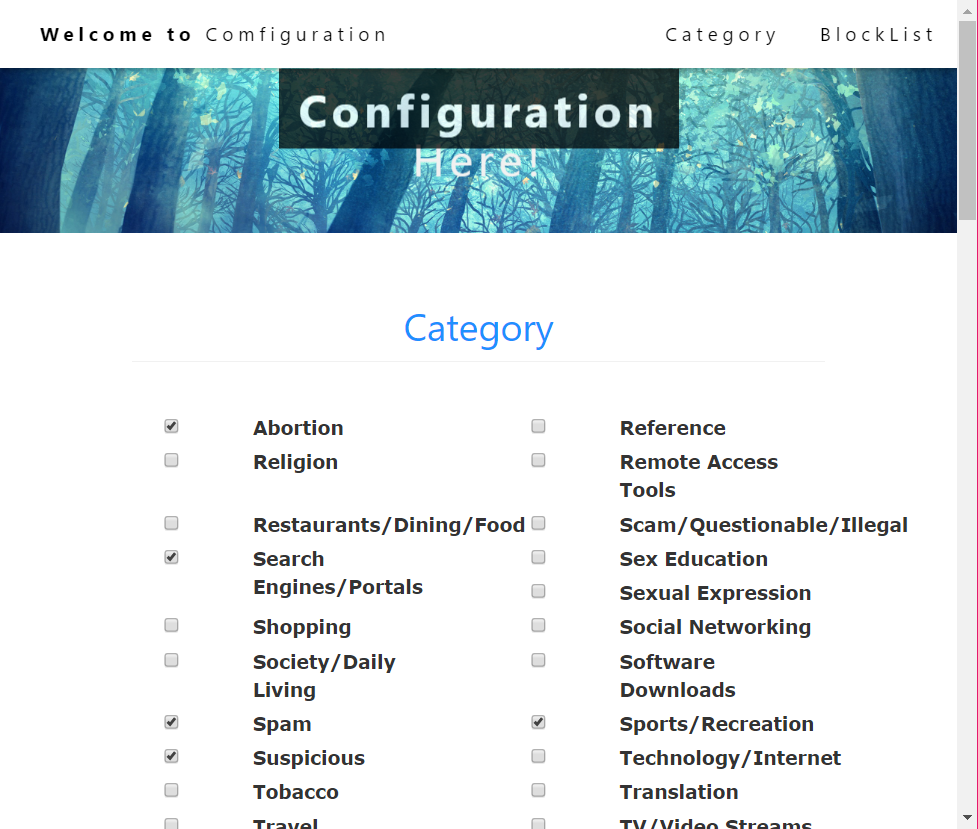


Figure Pick Categories

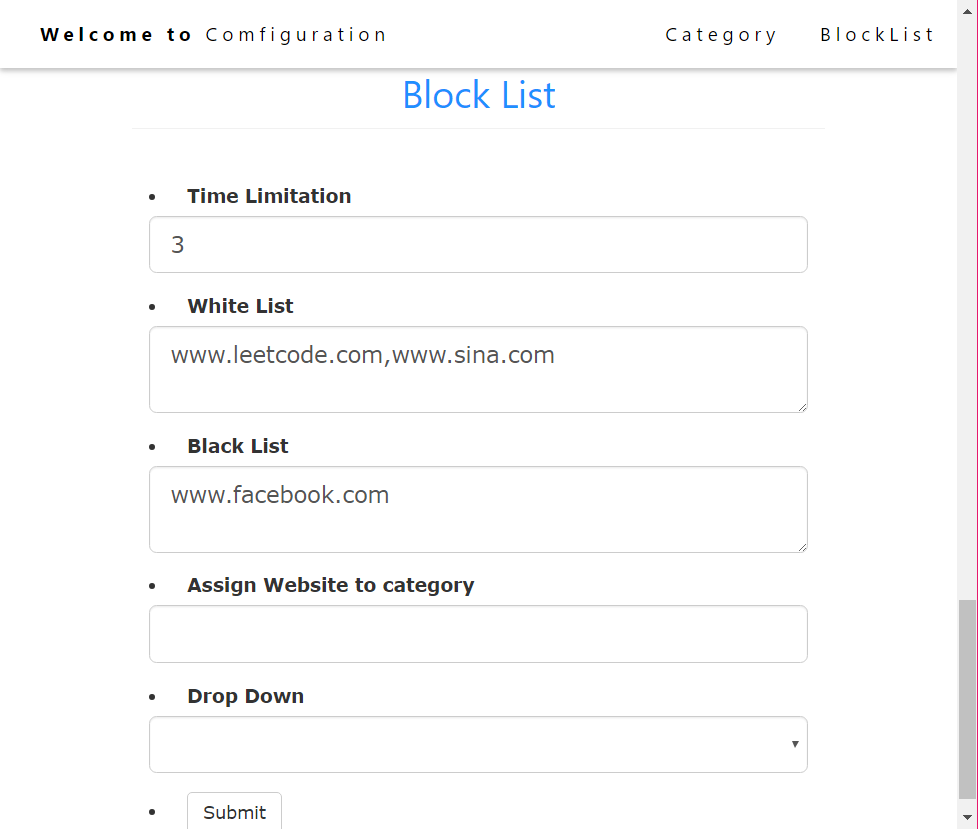


Figure Block List

### Database Structure

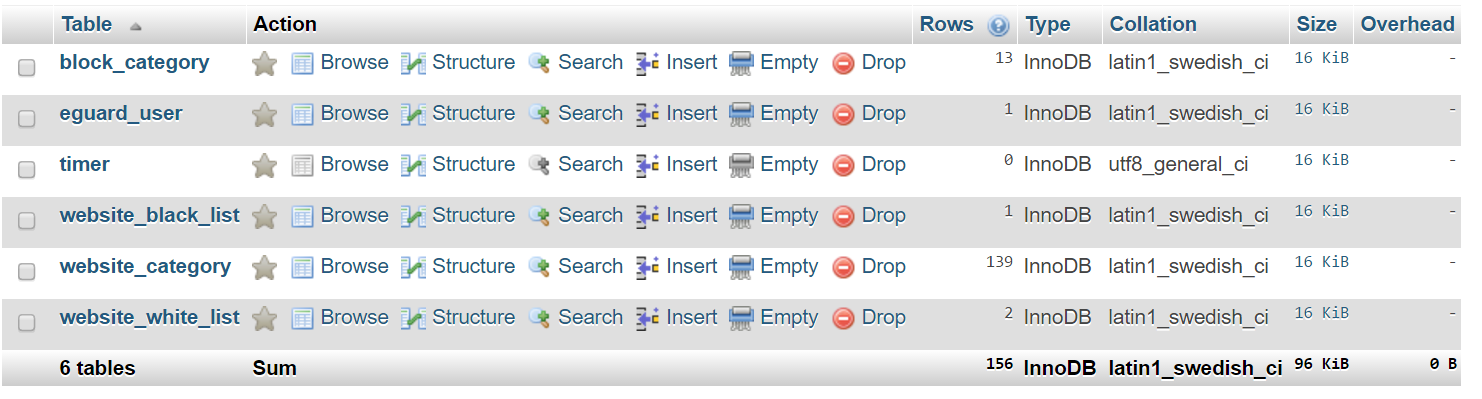


Figure MySql

# Class Diagrams

## Singleton

### UML

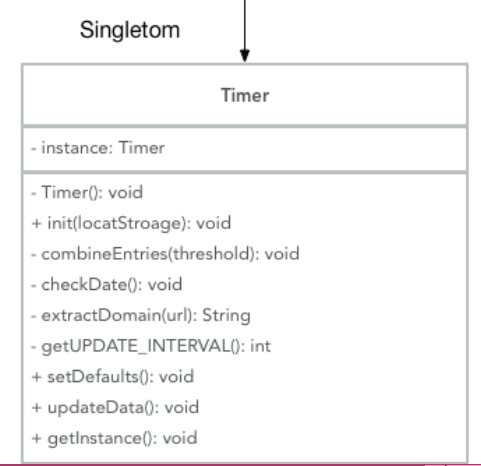


Figure Singleton UML

### Source Code:

//declare Timer CLass

var Timer = (function(){

  var instance;

  function init(localStorage){

  // Interval (in seconds) to update timer

      var UPDATE\_INTERVAL = 3;

      var limitedTime = parseInt(localStorage["timeLimitation"]);

      // Add sites which are not in the top threshold sites to "other" category

      // WARNING: Setting the threshold too low will schew the data set

      // so that it will favor sites that already have a lot of time but

      // trash the ones that are visited frequently for short periods of time

      var combineEntries = function (threshold) {

          var domains = JSON.parse(localStorage["domains"]);

          var other = JSON.parse(localStorage["other"]);

          // Don't do anything if there are less than threshold domains

          if (Object.keys(domains).length <= threshold) {

              return;

          }

          // Sort the domains by decreasing "all" time

          var data = [];

          for (var domain in domains) {

              var domain\_data = JSON.parse(localStorage[domain]);

              data.push({

                  domain: domain,

                  all: domain\_data.all

              });

          }

          data.sort(function (a, b) {

              return b.all - a.all;

          });

          // Delete data after top threshold and add it to other

          for (var i = threshold; i < data.length; i++) {

              other.all += data[i].all;

              var domain = data[i].domain;

              delete localStorage[domain];

              delete domains[domain];

          }

          localStorage["other"] = JSON.stringify(other);

          localStorage["domains"] = JSON.stringify(domains);

      }

​

      // Check to make sure data is kept for the same day

      var checkDate = function () {

          var todayStr = new Date().toLocaleDateString();

          var saved\_day = localStorage["date"];

          if (saved\_day !== todayStr) {

              // Reset today's data

              var domains = JSON.parse(localStorage["domains"]);

              for (var domain in domains) {

                  var domain\_data = JSON.parse(localStorage[domain]);

                  domain\_data.today = 0;

                  localStorage[domain] = JSON.stringify(domain\_data);

              }

              // Reset total for today

              var total = JSON.parse(localStorage["total"]);

              total.today = 0;

              localStorage["total"] = JSON.stringify(total);

              // Combine entries that are not part of top 500 sites

              combineEntries(500);

              // Keep track of number of days web timer has been used

              localStorage["num\_days"] = parseInt(localStorage["num\_days"]) + 1;

              // Update date

              localStorage["date"] = todayStr;

          }

      }

​

      // Extract the domain from the url

      // e.g. http://google.com/ -> google.com

      var extractDomain = function (url) {

          var re = /:\/\/(www\.)?(.+?)\//;

          return url.match(re)[2];

      }

​

      var inBlacklist = function (url) {

          if (!url.match(/^http/)) {

              return true;

          }

          var blacklist = JSON.parse(localStorage["blacklist"]);

          for (var i = 0; i < blacklist.length; i++) {

              if (url.match(blacklist[i])) {

                  return true;

              }

          }

          return false;

      }

      return {

          getUPDATE\_INTERVAL : function () {

              return UPDATE\_INTERVAL;

          },

​

      //set Time limitation

          setTimer : function(time){

              limitedTime = time\*60\*60;

​

          },

​

          getTimer : function(){

            return limitedTime;

          },

​

      // Set default settings

          setDefaults : function () {

              // Set blacklist

              if (!localStorage["blacklist"]) {

                  localStorage["blacklist"] = JSON.stringify(["example.com"]);

              }

              // Set number of days Web Timer has been used

              if (!localStorage["num\_days"]) {

                  localStorage["num\_days"] = 1;

              }

              // Set date

              if (!localStorage["date"]) {

                  localStorage["date"] = new Date().toLocaleDateString();

              }

              // Set domains seen before

              if (!localStorage["domains"]) {

                  localStorage["domains"] = JSON.stringify({});

              }

              // Set total time spent

              if (!localStorage["total"]) {

                  localStorage["total"] = JSON.stringify({

                      today: 0,

                      all: 0

                  });

              }

              if (!localStorage["timeLimitation"]){

                  localStorage["timeLimitation"] = 7200;

              }

              // Limit how many sites the chart shows

              if (!localStorage["chart\_limit"]) {

                  localStorage["chart\_limit"] = 9;

              }

              // Set "other" category

              // NOTE: other.today is not currently used

              if (!localStorage["other"]) {

                  localStorage["other"] = JSON.stringify({

                      today: 0,

                      all: 0

                  });

              }

          },

          // Update the data

          updateData : function () {

              // Only count time if system has not been idle for 30 seconds

              chrome.idle.queryState(30, function (state) {

                  if (state === "active") {

                      // Select single active tab from focused window

                      chrome.tabs.query({'lastFocusedWindow': true, 'active': true}, function (tabs) {

                          if (tabs.length === 0) {

                              return;

                          }

                          var tab = tabs[0];

                          // Make sure 'today' is up-to-date

                          checkDate();

                          if (!inBlacklist(tab.url)) {

                              var domain = extractDomain(tab.url);

                              // Add domain to domain list if not already present

                              var domains = JSON.parse(localStorage["domains"]);

                              if (!(domain in domains)) {

                                  domains[domain] = 1;

                                  localStorage["domains"] = JSON.stringify(domains);

                              }

                              var domain\_data;

                              if (localStorage[domain]) {

                                  domain\_data = JSON.parse(localStorage[domain]);

                              } else {

                                  domain\_data = {

                                      today: 0,

                                      all: 0

                                  };

                              }

                              domain\_data.today += UPDATE\_INTERVAL;

                              domain\_data.all += UPDATE\_INTERVAL;

                              localStorage[domain] = JSON.stringify(domain\_data);

                              // Update total time

                              var total = JSON.parse(localStorage["total"]);

                              total.today += UPDATE\_INTERVAL;

                              if (total.today > parseInt(window.localStorage["timeLimitation"])){

                                  chrome.windows.getCurrent(function(window){

                                      alert("Exceed the time limitation! \nPlease contact the administrator to modify limited time.");

                                      chrome.windows.remove(window.id);

                                  });

                              }

                              total.all += UPDATE\_INTERVAL;

                              localStorage["total"] = JSON.stringify(total);

                              // Update badge with number of minutes spent on

                              // current site

                              var num\_min = Math.floor(domain\_data.today / 60).toString();

                              if (num\_min.length < 4) {

                                  num\_min += "m";

                              }

                              chrome.browserAction.setBadgeText({

                                  text: num\_min

                              });

                          } else {

                              // Clear badge

                              chrome.browserAction.setBadgeText({

                                  text: ""

                              });

                          }

                      });

                  }

              });

          }

      }

  }

  return{

  getInstance: function(localStorage) {

​

      if ( !instance ) {

          instance = init(localStorage);

      }

​

      return instance;

  }

  }

})();

## Façade

### UML

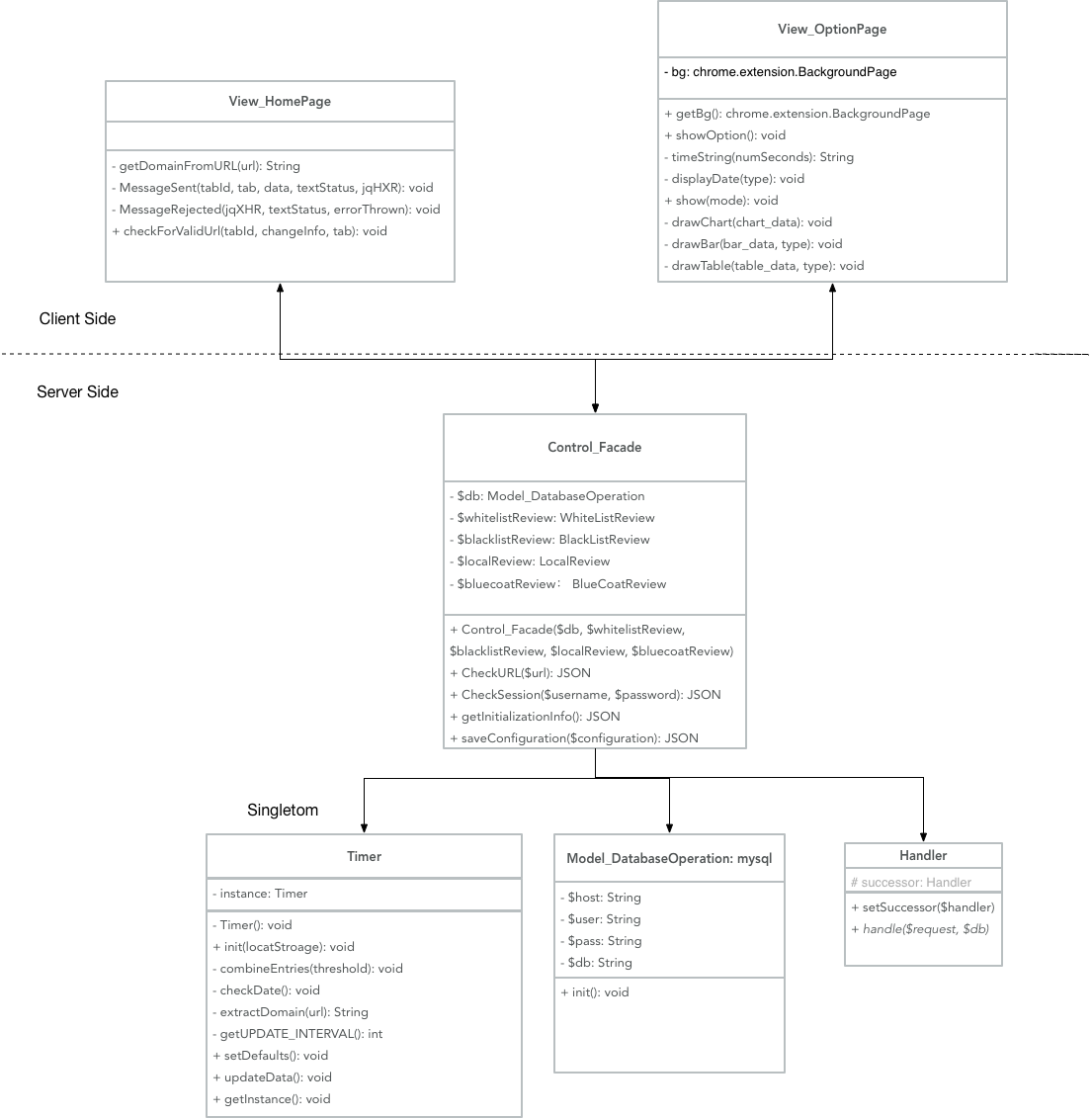


Figure Control\_Facade

### Source Code:

class Control\_Facade {

  private $db = null;

  private $whitelistReview = null;

  private $blacklistReview = null;

  private $localReview = null;

  private $bluecoatReview = null;

​

  // initialization for Control\_Facade

  public function \_\_construct(PDO $db, WhiteListReview $whitelistReview,

                              BlackListReivew $blacklistReview, LocalReview $localReview, BlueCoatReview $bluecoatReview){

      $this->db = $db;

      $this->whitelistReview = $whitelistReview;

      $this->blacklistReview = $blacklistReview;

      $this->localReview = $localReview;

      $this->bluecoatReview = $bluecoatReview;

  }

​

  // set the chain of responsibilities by using setSuccessor()

  public function CheckURL($url){

      $this->whitelistReview->setSuccessor($this->blacklistReview);

      $this->blacklistReview->setSuccessor($this->localReview);

      $this->localReview->setSuccessor($this->bluecoatReview);

      $this->bluecoatReview->setSuccessor($this->localReview);

      return $this->whitelistReview->handle($url, $this->db);

  }

​

  // Valid the user login

  public function CheckSession($username, $password){

      $query = ("SELECT \* FROM eguard\_user WHERE Username = '{$username}' "

                . "AND Password = '{$password}'");

      $result = $this->db->query($query);

      if ($result->rowCount() != 0) {

          exit(json\_encode("allow"));

      }

      else{

          exit(json\_encode("deny"));

      }

  }

  public function getInitalizationInfo(){

      $query\_blockedCategory = ("SELECT Category FROM block\_category");

      $configuration->block\_category = $this->db->query($query\_blockedCategory)->fetchAll(PDO::FETCH\_COLUMN, 0);

      $query\_timer = ("SELECT Limitation FROM timer");

      $configuration->timer = $this->db->query($query\_timer)->fetchAll(PDO::FETCH\_COLUMN, 0);

      $query\_white\_list = ("SELECT URL FROM website\_white\_list");

      $configuration->white\_list = $this->db->query($query\_white\_list)->fetchAll(PDO::FETCH\_COLUMN, 0);

      $query\_black\_list = ("SELECT URL FROM website\_black\_list");

      $configuration->black\_list = $this->db->query($query\_black\_list)->fetchAll(PDO::FETCH\_COLUMN, 0);

      exit(json\_encode($configuration));

  }

​

  public function saveConfiguration($configuration){

      $blockedCategories = $configuration->blockedCategories;

      $timer = $configuration->timer;

      $whitelist = $configuration->whitelist;

      $blacklist = $configuration->blacklist;

      $website = $configuration->website;

      $category = $configuration->category;

      $query\_blockedCategory = ("delete FROM block\_category");

      $result = $this->db->exec($query\_blockedCategory);

      for($numOfBlockedCategories = 0;

          $numOfBlockedCategories < sizeof($blockedCategories);

          $numOfBlockedCategories++)

      {

          $query\_blockedCategory = "INSERT INTO block\_category (`Category`) VALUES ('{$blockedCategories[$numOfBlockedCategories]}');";

          $result = $this->db->exec($query\_blockedCategory);

      };

      $query\_timer = "UPDATE timer SET Limitation = $timer LIMIT 1;";

      $result = $this->db->exec($query\_timer);

      preg\_match\_all('/.+?,/', $whitelist , $whitelist);

      $query\_white\_list = "DELETE FROM website\_white\_list";

      $result = $this->db->exec($query\_white\_list);

      for($numOfWhitelist = 0; $numOfWhitelist < sizeof($whitelist[0]); $numOfWhitelist++){

          $whitelist\_URL = str\_replace(array(","),"",$whitelist[0][$numOfWhitelist]);

          $query\_white\_list = "INSERT INTO website\_white\_list (`URL`) VALUES ('{$whitelist\_URL}')";

          $result = $this->db->exec($query\_white\_list);

      }

      preg\_match\_all('/.+?,/', $blacklist , $blacklist);

      $query\_black\_list = "DELETE FROM website\_black\_list";

      $result = $this->db->exec($query\_black\_list);

      for($numOfBlacklist = 0; $numOfBlacklist < sizeof($blacklist[0]); $numOfBlacklist++){

          $blacklist\_URL = str\_replace(array(","),"",$blacklist[0][$numOfBlacklist]);

          $query\_black\_list = "INSERT INTO website\_black\_list (`URL`) VALUES ('{$blacklist\_URL}')";

          $result = $this->db->exec($query\_black\_list);

      }

      if(!empty($website) && !empty($category)){

          $query = "DELETE FROM website\_category WHERE URL = '{$website}'";

          $delete\_result = $this->db->exec($query);

          $query\_newWebsite = "INSERT INTO website\_category (`URL`,`Category`) VALUES ('{$website}','{$category}');";

          $result = $this->db->exec($query\_newWebsite);

      }

      exit(json\_encode("success!"));

  }

}

​

## Chain of responsibility

### UML

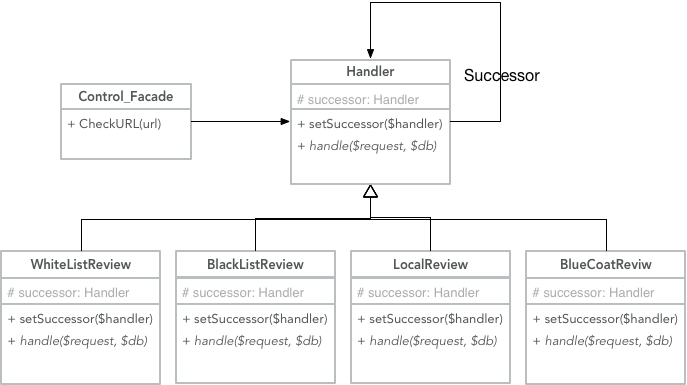


Figure CoR

### Source Code:

class Control\_Facade {

// set the chain of responsibilities by using setSuccessor()

  public function CheckURL($url){

      $this->whitelistReview->setSuccessor($this->blacklistReview);

      $this->blacklistReview->setSuccessor($this->localReview);

      $this->localReview->setSuccessor($this->bluecoatReview);

      $this->bluecoatReview->setSuccessor($this->localReview);

      return $this->whitelistReview->handle($url, $this->db);

  }

}

​

// chain of responsibilities

abstract class Handler{

​

  protected $successor = null;

​

  public function setSuccessor(Handler $handler){

      $this->successor = $handler;

  }

​

  abstract public function handle($request, PDO $db);

}

​

class WhiteListReview extends Handler{

​

  public function handle($request,PDO $db){

      $query = "SELECT URL FROM website\_white\_list Where URL = '{$request}' LIMIT 10";

      $result = $db->query($query);

      if ($result->rowCount() != 0) {

          $datarow = $result->fetch();

          /\* free result set \*/

          exit(json\_encode("allow"));

      }

      else{

​

          $this->successor->handle($request, $db);

      }

  }

}

​

class BlackListReivew extends Handler{

  public function handle($request, PDO $db){

      $query = "SELECT URL FROM website\_black\_list Where URL = '{$request}' LIMIT 10";

      $result = $db->query($query);

      if ($result->rowCount() != 0) {

          $datarow = $result->fetch();

          exit(json\_encode("deny"));

      }

      else{

          $this->successor->handle($request, $db);

      }

  }

}

​

class LocalReview extends Handler{

  public function handle($request, PDO $db){

      $query = "SELECT Category FROM website\_category WHERE URL = '{$request}' LIMIT 10";

      $result = $db->query($query);

      if ($result->rowCount() != 0){

          $datarow = $result->fetch();

          $query = "SELECT Category FROM block\_category WHERE Category = '{$datarow[Category]}'";

          $result = $db->query($query);

          if ($result->rowCount() != 0){

              exit(json\_encode("deny"));

          }

          else{

              exit(json\_encode("allow"));

          }

      }

      else{

          $this->successor->handle($request, $db);

      }

  }

}

​

class BlueCoatReview extends Handler{

  public function handle($request, PDO $db){

      $post\_data = array(

          'url' => $request

      );

      $postdata = http\_build\_query($post\_data);

      $options = array(

          'http' => array(

              'method' => 'POST',

              'header' => 'Content-type:application/x-www-form-urlencoded',

              'content' => $postdata,

              'timeout' => 15 \* 60

          )

      );

      $context = stream\_context\_create($options);

      // query the bluecoat to check the website category

      $result = file\_get\_contents('http://sitereview.bluecoat.com/rest/categorization', false, $context);

      $jsonResult = json\_decode($result, true);

      if(preg\_match\_all('/>.+?<\/a>/', $jsonResult['categorization'] , $categorization)){

          for($categorizationIndex=0; $categorizationIndex < sizeof($categorization[0]); $categorizationIndex++){

              // trim the categorization string

              $categorization[0][$categorizationIndex] = str\_replace(array("</a>",">"),"",$categorization[0][$categorizationIndex]);

              $query = "INSERT INTO website\_category (`URL`, `Category`) VALUES ('{$request}','{$categorization[0][$categorizationIndex]}');";

              $result = $db->exec($query);

          }

          if (strcmp($categorization[0][0],"Uncategorized")==0){

              if (require './PHPMailer/PHPMailerAutoload.php')

                  echo "Seccess load PHPMailer";

              //Create a new PHPMailer instance

              $mail = new PHPMailer;

              //Tell PHPMailer to use SMTP

              $mail->isSMTP();

              //Enable SMTP debugging

              // 0 = off (for production use)

              // 1 = client messages

              // 2 = client and server messages

              $mail->SMTPDebug = 2;

              //Ask for HTML-friendly debug output

              $mail->Debugoutput = 'html';

              //Set the hostname of the mail server

              $mail->Host = 'smtp.gmail.com';

              //Set the encryption system to use - ssl (deprecated) or tls

              $mail->Port = 587;

              $mail->SMTPSecure = 'tls';

              //Whether to use SMTP authentication

              $mail->SMTPAuth = true;

              //Username to use for SMTP authentication - use full email address for gmail

              $mail->Username = "ztlevitest@gmail.com";

              //Password to use for SMTP authentication

              $mail->Password = "helloTest";

              //Set who the message is to be sent from

              $mail->setFrom('ztlevitest@gmail.com', 'Ting Zhou');

              //Set an alternative reply-to address

              $mail->addReplyTo('ztlevitest@gmail.com', 'Ting Zhou');

              //Set who the message is to be sent to

              $mail->addAddress('ztlevtest@yahoo.com', 'Ting Zhou');

              $query = ("SELECT Username, Email FROM eguard\_user");

              $result = $db->query($query);

              $user = $result->fetch();

              $mail->AddAddress("{$user['Email']}", "{$user['Username']}");

              //Set the subject line

              $mail->Subject = 'PHPMailer GMail SMTP test';

              //Read an HTML message body from an external file, convert referenced images to embedded,

              //convert HTML into a basic plain-text alternative body

              $mail->Body = 'Hello!<br>' . $request . 'is uncategorized, please go to E-Guard option page and assign it to one category!<br>Thanks';

              // $mail->msgHTML(file\_get\_contents('contents.html'), dirname(\_\_FILE\_\_));

              //Replace the plain text body with one created manually

              $mail->AltBody = 'This is a plain-text message body';

              //Attach an image file

              // $mail->addAttachment('images/phpmailer\_mini.png');

              //send the message, check for errors

              if (!$mail->Send()) {

                  echo "Mailer Error: " . $mail->ErrorInfo;

              } else {

                  echo "Message sent!";

              }

          }

          $this->successor->handle($request, $db);

      }

  }

}

​