Applying the 3-layer Model in the Construction of a Framework to Create Web Applications

Daniel SANCHEZ
Universidad Distrital Francisco Jose de Caldas,
Bogotá, Colombia, danielssj88@gmail.com

Oscar MENDEZ
Universidad Distrital Francisco Jose de Caldas,
Bogotá, Colombia, oscfrayle@gmail.com

Hector FLOREZ
Universidad Distrital Francisco Jose de Caldas,
Bogotá, Colombia, haflorezf@udistrital.edu.co

ABSTRACT

Currently, there are a lot of frameworks to build web applications working with the architectural pattern MVC (Model View Controller) [1]. One interesting approach is to use the 3-layer model [2], which allows identifying and separating the final application in different layers that facilitates its construction and maintenance. The purpose of this paper is to present our approach of a framework [3] for developing PHP web applications using the 3-layer model. This approach integrates different technologies and design patterns in order to provide one tool that supports the community in the creation of PHP web applications providing build-in tools and applying good practices focused on the pursue of proper development times. In addition, the approach aims to handle common issues in the industry like efficiency, maintainability, and security.

Keywords: Multilayer Architecture, Model View Controller, 3-layer Model.

1. INTRODUCTION

According to [4], software are not only programs but all associated documents and configuration data needed to make these programs to operate correctly. Usually a software system consists of several separated programs, configuration files that are used to run these programs, and a documentation system for describing the software. Thus, good software meet its goals when these attributes highlights the functionality required by the user, while being maintainable, reliable and easy to use.

That is the reason why this work seeks to characterize as a multilayer framework to create web applications in PHP that allows to improve own processes of software development from its architectural and conceptual design. This architecture provides new ways of perceiving and applying own methodological of the area such as object-oriented programming and application design patterns. Design patterns have become very important due to they have changed the area of software engineering in favor of creating truly elegant designs [5].

Object-oriented software design is difficult, and it is even more design it reusable [5]. Thus, object oriented software design, it is necessary in to find the relevant objects, factoring them in classes with the right granularity, define classes, inter-faces and inheritance hierarchies, and establish key relation-ships between these classes and objects. This paper proposes the development of a framework to support PHP software development, which has been validated through the creation of one specific web application named CTS Virtual, which is a software to create surveys that focus on research about the perception of the Colombians about science, technology and society.

The development and documentation of CTS Virtual and our framework does not intend to put the developers in the position to decide between the different solutions available in the market for the construction of web applications, what it looks for is to present an experience that presents the readers to one the many possible alternatives in terms of the use and implementation of software architectures.

The paper is structured as follows. Section II presents the context in order to explain the related concepts used in the construction of the framework. Section III presents related work. Section IV presents the implementation of the framework in the practical case of CTSVirtual. Finally Section V presents the conclusions.

2. CONTEXT

The software industry is in constant search of new ways of solving existing problems. One good approach to improve the development time and costs in the construction of software is the reuse of source code, given the amount of web applications, we found similar functionality that supports the fact to use and develop build-in libraries, which allows performing a specific task that can be integrated into multiple projects such as libraries, which are able to perform generic features improving the performance and security of applications that motivates the reduction of time and effort that ultimately reduces manufacturing costs.

In this section, we provide a brief explanation regarding important concepts related to the construction of our approach.

A. Separation of Concern

Separation of concern (SoC) is a software development concept that separates a computer program into different sections or concerns, in which each concern has a different purpose. By separating these sections, each one can encapsulate information that can be developed and updated independently. 3-layer development is an example of SoC in which the user interface (UI) is separated from both the business layer and the data access layer. Although the addition of SoC adds some complexity to the applications design, the benefits outweigh the extra complexity [6].

In the process of building a framework, defining the architecture has a great importance because it is the basis for the development of the logic components and their interaction, taking in mind that these components need to be reused in the applications that are implemented on it. For the specific case of the development of CTS Virtual the definition of architecture took place when the research about the main features of the pattern of three layers and the Model View Controller was made. This research generated the concepts used in the construction of the web framework, which is mainly profiled to characterize in an independent way the general aspects of each of these patterns.

B. Model View Controller

The design pattern Model View Controller or MVC defines a software architecture that separates data from the user interface and events into different parts: the Model contains application data, the View manages the user interface, and the Controller is responsible for receiving user requests (events or actions performed on the user interface). When the model is modified then the view is updated; in addition, the controller selects the view to show the actual responses and the model to assign to it [7].

Moreover, according to [8], the MVC pattern consists of 3 different components, where the model acts as the domain that software is built around, the view as the visual representation of a model, and the controller responsible for processing input, acting upon the model, and deciding on what action should be performed, such as rendering a view or redirecting to another page for supporting a business models. Figure 1 presents the component diagram of the MVC pattern.

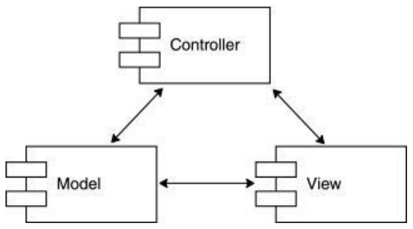

Fig. 1. Model View Controller component diagram.

C. 3-layer model

Another alternative is to use the 3-layer model, which separates the presentation logic, business logic and data access in 3 different layers. Using the definition of layer, each layer supports the layer n+1 having in mind that the lower layer in this case is data access, so the presentation layer does not have access to the data access layer. This model is presented in the Figure 2.

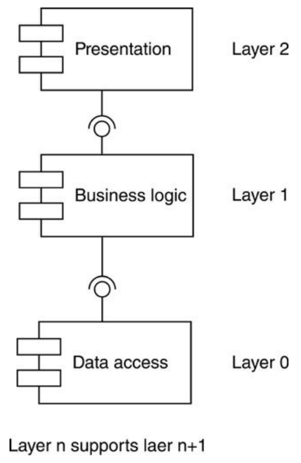

Fig. 2. 3-layer model.

One of the main advantages of this model is the low coupling between the layers because this characteristic allows easily to modify functionality in the application without having in mind all the components of the application [9].

3. RELATED WORK

In the industry there are several frameworks to create web applications using PHP such as Symfony, CakePHP and Zend Framework. The majority of them use MVC as an architectural design pattern [10].

Nevertheless, by interacting with frameworks such as CakePHP, CodeIgniter, and Kohana, their maturity and thus the complex and heavy that they can become at the time of development of specific technological tools and specific objectives is not the best, This is because they are equipped with lots of libraries for the final result that are useless and impractical for the context. Moreover, there are some projects that do not need the whole structure of a framework like Laravel or Symfony, either because they are very simple, they have very specific characteristics or require extreme speed of execution. This statement refers to the need to take concrete and lightweight architectures that meet the specific objectives of the project that is intended to be developed and that allows to be scalable to adding new functionality. In this approach there are another alternatives like Lumen [11] which is a reduced version of the bigger framework Laravel [12].

Hence, the idea arises to create a framework with reduced functionality in which the proposed architecture is designed to build a set of logical tools that provide the necessary components to developers, who make part of building instruments focused on this field. In addition, the approach uses the 3-layer model that is very well accepted in the academy; thus, the academic community can use it in different projects.

4. SOLUTION STRATEGY

The development of this framework, which has been validated in the construction of the project CTS Virtual, is characterized by defining three levels or layers of functionality: presentation, logic and persistence, where its operation is more simple and linear fulfilling the purpose to separate the presentation of persistence and to avoid logic code within interfaces.

It is important to mention that in the case of the construction of the multi-layer architecture for creating web applications and specifically for CTS Virtual, the use of Smarty [13] as template manager plays a vital role because it allows to separate the interaction between the code in the client side (HTML, CSS, Javascript) with the code of the server side (PHP), giving a new level of abstraction that modularize the presentation layer, allowing one code more clear and readable. The complete architecture is shown in the Figure 3.

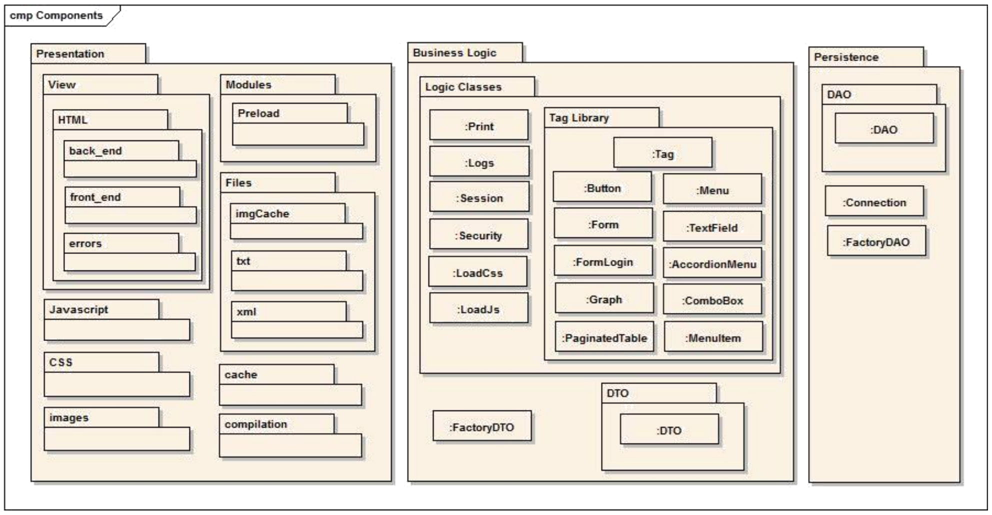

Fig. 3. Multilayer model.

A. Persistence layer

In this model, the data access layer is responsible for encapsulating all the logic necessary to communicate between the application and what is being used to persist data, such as an engine database, serialized files, or other storage system. In this case, the storage system is a database; thus, our data layer manage the SQL statements sent to the database. For that task it is used the DAO (Data Access Object) pattern by the library ADOdb [14] in what we have independence between the database engine and the code implemented to connect to the database. In the case of CTS Virtual we used the class hierarchy presented in Figure 4.

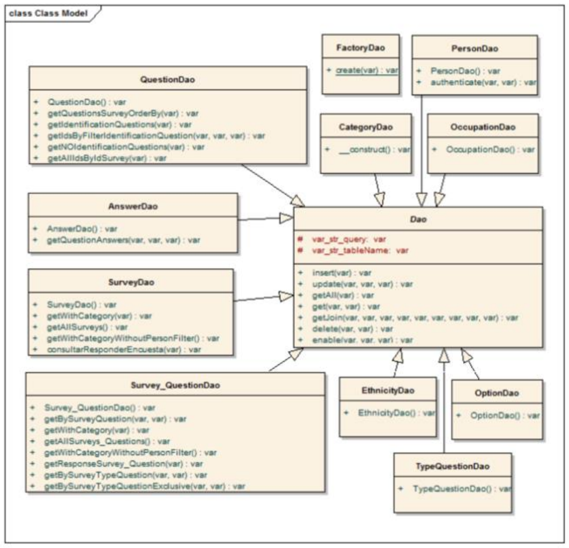
class Class Model
FactoryDao
+ create(var) : var
PersonDao
+ PersonDao() : var
+ authenticate(var, var) : var
QuestionDao
+ QuestionDao() : var
+ getQuestionsSurveyOrderBy(var) : var
+ getIdentificationQuestions(var) : var
+ getIdsByFilterIdentificationQuestion(var, var, var) : var
+ getNOIdentificationQuestions(var) : var
+ getAllIdsByIdSurvey(var) : var
CategoryDao
+ __construct() : var
OccupationDao
+ OccupationDao() : var
AnswerDao
+ AnswerDao() : var
+ getQuestionAnswers(var, var, var) : var
Dao
# var_str_query: var
# var_str_tableName: var
+ insert(var) : var
+ update(var, var, var) : var
+ getAll(var) : var
+ get(var, var) : var
+ getJoin(var, var, var, var, var, var, var, var, var) : var
+ delete(var, var) : var
+ enable(var, var, var) : var
SurveyDao
+ SurveyDao() : var
+ getWithCategory(var) : var
+ getAllSurveys() : var
+ getWithCategoryWithoutPersonFilter() : var
+ consultarResponderEncuesta(var) : var
Survey_QuestionDao
+ Survey_QuestionDao() : var
+ getBySurveyQuestion(var, var) : var
+ getWithCategory(var) : var
+ getAllSurveys_Questions() : var
+ getWithCategoryWithoutPersonFilter() : var
+ getResponseSurvey_Question(var) : var
+ getBySurveyTypeQuestion(var, var) : var
+ getBySurveyTypeQuestionExclusive(var, var) : var
EthnicityDao
+ EthnicityDao() : var
OptionDao
+ OptionDao() : var
TypeQuestionDao
+ TypeQuestionDao() : var

Fig. 4. DAO classes - class diagram.

The factory pattern was used to create the DAO objects in the next layer (i.e. the logic layer) that encapsulates the operations made against the database, and all the classes inherits from the class DAO, which has the definition of the generic methods that return the SQL statement as:

- *insert*: Generates dynamically insert queries, using a data array that receives as parameter.

```
/**
* @param Array $arr_data entity data in the
    database.
**/
function insert($arr_data){ ... }
```

- *update*: Generates dynamically update queries, using a data array that receives as parameter.

```
/**
 * @param Array $arr_data data in the entity
     of the database.
 * @param String $var_str_key attribute name
     to compare.
 * @param String $var_str_value attribute
     value to compare.
 */
function update($arr_data, $var_str_key,
    $var_str_value){ ... }
```

- *getAll*: Generates dynamically simple select queries of all columns in a table.

```
/**
* @param String $var_str_order a field name
    in the database by which to sort.
*/
function getAll($var_str_order){ ... }
```

- *get_by_attribute_and_value*: Generates the select query based on one filter with one merely parameter

```
/**
* @param String $var_str_key attribute name
    to compare.
* @param String $var_str_value attribute
    value to compare.
*/
function getAll($var_str_key, $var_str_value)
    { ... }
```

- *getJoin*: Method that generates dynamically Join sentence between tables.

```
/**
* @param String $var_str_nameTable1 name of
    the first table.
* @param String $var_str_nameTable2 name of
    the second table.
* @param Array $arr_attributesTable1
    attributes to get from the first table.
* @param Array $arr_attributesTable2
    attributes to get from the second table.
* @param String
    $var_str_attributoJoinOnTable1 attribute
    to join within the first table.
* @param String
    $var_str_attributoJoinOnTable2 attribute
    to join within the second table.
* @param String $var_str_order $var_str_order
    attribute to order the query.
* @param String $var_str_key attribute to
    order the query (optional).


* @param String $var_str_value value to order
    the query (optional).
*/
function queryJoin($var_str_nameTable1,
    $var_str_nameTable2,
    $arr_attributesTable1,
    $arr_attributesTable2,
    $var_str_attributoJoinOnTable1,
    $var_str_attributoJoinOnTable2,
    $var_str_order,
    $var_str_key = "",$var_str_value = "")
{ ... }
```

- *delete*: Generates dynamically delete queries comparing the attribute and values that receives as parameter.

```
/**
* @param String $var_str_key attribute name
    to compare.
* @param String $var_str_value value to
    compare.
*/
function delete($var_str_key, $var_str_value)
{ ... }
```

- *enable*: Enable or disable one register updating its value 'enable' to 0 or 1 depending the case.

```
/**
* @return String returns the exception (if
    exists).
* @param String $var_str_key attribute name
    to compare.
* @param String $var_str_value value to
    compare.
* @param Int $var_int_enable 0 enable, 1
    disable.
**/
public function enable($var_str_key,
    $var_str_value, $var_int_enable) { ... }
```

These methods have the logic implemented to interact with the entities in the database. Each of this classes inherits from the abstract class *DAO*, decreasing the implementation only to the next steps:

1) Inherits from the class *DAO* in the constructor to define in the variable *var_str_tableName* of the *DAO* superclass the desired name of the entity in the database that to map whit such class. For example, the class *CategoryDAO* that made operations with the entity in the database category.

```
public function __construct() {
  $this->var_str_tableName = "category";
```

2) Later on, in the class *FactoryDAO* is made the instantiation of the objects of the classes *DAO* created to be used in the business logic layer.

```
/**
* Class "FactoryDao" which lets you create
    all objects in the persistence layer
    that implement the dao pattern. This
    class applied the Factory pattern for
    dynamic creation of objects with the "
    create" method by sending the name of
    the object you want to create, to assist
     in the creation of dynamic objects and
    centralized business logic.
*/
class FactoryDao{
    /**
     * creates and returns an object with
    the name of the class that receives as
    parameter.
     * @access public
     * @return Object Dao, specific object
    type you requested.
     */
    public static function create(
    $var_str_type){
        return new $var_str_type();
    }
}
```

This class receives the name of the class as a string to create and return the new instance to use in the next layers.

B. Business logic layer

The business logic layer uses the pattern *DTO* (Data Transfer Object) to transfer the data from the data access to the presentation layer. For this task, the same approach that was used in the *DAO* pattern is used; thus, there is one abstract class *DTO* that implements the generic functionalists to those classes. For this, it is necessary to follow the next steps:

1) Creating the class that inherits from the class *DTO* in the constructor is necessary to define in the variable *var_str_nameDao* of the superclass *DTO* the name of the desired *DAO* class to map with such class. For instance, the class *CategoryDTO* that makes operations with its corresponding *DAO* class, i.e. the *CategoryDAO* class.

```
public function CategoryDto() {
  $this->var_str_nameDao = "CategoryDao";
}
```

2) Afterwards, in the class *FactoryDTO* is made the instantiation of the objects of the classes *DTO* created to be used in the presentation layer. For this task, it is necessary to declare the constants with the name of the classes in *DTO* that are going to be instantiated in the method create. This method receives the name of the class as a string to create and return the new instance to use in the next layers

```
/**
* This class contains constants that stores
    the name of the classes in DTO.
**/
class FactoryDto{
        const PERSON_DTO=”PersonDto”;
        const CATEGORY_DTO=”CategoryDto”;
        const SURVEY_DTO=”SurveyDto”;
        const QUESTION_DTO=”QuestionDto”;
        const TYPE_QUESTION_DTO=”TypeQuestionDto
        ”;
        const OPTION_DTO=”OptionDto”;
        const ANSWER_DTO=”AnswerDto”;
        const SURVEY_QUESTION_DTO=”
        Survey_QuestionDto”;
        const OCCUPATION_DTO=”OccupationDto”;

        /**
         * @return Object Dto, specific object
        type you requested.
         */
        public static function create(
        $var_str_type){
            return new $var_str_type();
        }
    }
```

3) In the business logic layer there are another classes such as:
   a) Tag Library: The tag library was created to print HTML elements like forms, menu and so on; these elements works with the structure and the libraries used in the framework, for instance using this to create the menu generates the HTML structure that works with JQuery and takes the look and feel configured for the application. Figure 2 into the business logic layer shows the Tag Library classes.
   b) General Use Classes: These are classes for general use presented in the Figure 2 into the business logic layer. The functionality of those classes are:
      LoadCss: This class has a set of constants defined and a static method to get the routes of the stylesheets dynamically. **LoadJs:** It accomplish the same function that LoadCss class, but in this case it made the load of the Javascript files and JQuery plugins.
      Print: This class works with one tag library returning a string with the HTML code generated to assign to the Smarty templates that are mentioned later in this paper.
      Logs: Class responsible to print the server logs in a text/plain file to make the instrumentation and auditory in the application.
      Security: The injections of malicious code are one of the security issues more common in the web applications [15] to prevent this our framework has one personalized class used to perform the filter and validate the inputs in the URLs and form fields with the purpose to avoid injections of malicious code as XSS injections or SQL injections.
      Session: Class responsible to manage the sessions of the applications, avoiding security failures like session fixation what is common in applications that uses server sessions based in cookies to manage the authentication and authorization.
      Util: Class that have useful operations and common to use in all of the application.

C. Presentation Layer

To improve the separation between the client logic (HTML) and the server logic (PHP) the template manager Smarty [13] was used. In addition, the separation of the code this library allows printing code with its own tags and the storage in memory (Smarty cache) to future queries. The presentation layer has the structure presented in the first part of the Figure 2. The directories in the presentation layer are:

- **files:** It has the files generated or the general use for the application like the XML, TXT, PDFs and generated images files.
- **cache:** Directory used by Smarty to save the cache of the templates
- **compilation:** This directory is used for Smarty to save the templates compiled that are localized.
- **modules:** This directory contains the PHP scripts that receive the requests of the forms to call the functionalities in the business logic, the preload of the Smarty variables to show in the view, and to process the AJAX requests.
- **view:** It contains the folders of static files to present in the view. The HTML directory contains the HTML files with the Smarty variables to present the information.

D. Configuration File

Another point to highlight in this framework in the centralization of the configurations in one configuration file that has all the routes, the separator to use in the directories system like ’\’ for windows systems and ’/’ for Unix systems, plus the session, database connection, smarty configurations and so on, even the theme to display in the app with Jquery UI. This file appears in the root of the app, but its location can be changed and just its its path need to be updated in the index.php file.

E. Route

One of the main features of one web framework is the ability to route calls made to its components from the URL, maintaining a clear structure that is easy reading for the users and search engines; it is why our framework runs routing using the index.php file as an access point to respond to the requests from the client-side in a way more agile. This in function of SEO (Search Engine Optimization), as a feature for accessibility and usability in web applications [16].

5. CONCLUSIONS

In the industry this kind of architectures allow having well-defined three roles working within a group developing a project that can work together taking advantage of the Soc (Separation of concerns) [6] that create this architecture.

These work roles can be the designer or front-end developer of the GUI that would work on the presentation layer, the developer of the business logic which is responsible for performing calculations and information processing, and the developer of the data access logic which work directly in the layer persistence and deal with the issues the the data query. In this manner, none of these roles need to know how to work the other roles, merely knowing the way to communicate with the next layer; this approach allows us to generate standards of work into the develop group.

The develop of CTS Virtual as a case under study allowed us to determine the feasibility of developing a framework focused on a software architecture based on a three-layer model, considering the details of performance and security in the integrity of the information; as well as it required to implement functionalities like data access, template management, forms building, and so on what made vital part of the core of the framework as an agile and light architecture to create web applications.

It is clear that there is a great variety of PHP frameworks to build web applications, which are backed by large communities and companies that invest resources at all levels in order to improve their constant evolution, however this article focuses on the experience of developing an architecture that allowed the construction of CTS Virtual under a framework that focused on the solution of specific problems, and which in turn is projected as an opportunity to document an architecture and an experience around the software development.

Scalability is presented as a common feature of the frameworks mentioned before, This was materialized in our framework in a file called config.php that allowed to centralize similarly all those utilities and libraries with their respective configurations to increase the possibility of scaling in a simple way web applications, providing greater functionality and speed for the development and implementation of the products obtained from the basis of this architecture.

9. REFERENCES

[1] A. Leff and J. T. Rayfield, **“Web-application development using the model/view/controller design pattern,”** in Enterprise Distributed Object Computing Conference, 2001. EDOC ’01. Proceedings. Fifth IEEE International, 2001, pp. 118–127.

[2] F. Buschmann, K. Henney, and D. C. Schmidt, **Pattern oriented software architecture**, ser. Wiley series in software design patterns. Chichester, England and Hoboken, N.J.: John Wiley, 2007.

[3] A. Pasetti**, Software frameworks and embedded control systems**, ser. Lecture notes in computer science. Berlin and New York: Springer, 2002, vol. 2231.

[4] I. Sommerville and M. I. A. Galipienso, **Ingeniería del software**. Pearson Educacion, 2005.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, **“Design patterns: Abstraction and reuse of object-oriented design,”** in European Conference on Object-Oriented Programming. Springer, 1993, pp. 406–431.

[6] W. Penberthy, Exam Ref 70-486 Developing ASP. NET MVC 4 Web Applications (MCSD): **Developing ASP. NET MVC 4 Web Applications**. Pearson Education, 2013

[7] N. Kupp and Y. Makris, **“Applying the model-view controller paradigm to adaptive test,”** IEEE Design & Test of computers, vol. 29, no. 1, pp. 28–35, 2012.

[8] W. Romsaiyud, **“Applying mvc data model on hadoop for delivering the business intelligence,”** in ICT and Knowledge Engineering (ICT and Knowledge Engineering), 2014 12th International Conference on. IEEE, 2014, pp. 78–82

[9] H. Florez, **“Multilayer arquitecture though ajax and orm,”** Vinculos, vol. 7, no. 1, pp. 3–16, 2013.

[10] B. Porebski, K. Przystalski, and L. Nowak, **Building PHP Applications with Symfony, CakePHP, and Zend Framework**. John Wiley and Sons, 2011.

[11] Lumen. **Lumen laravel**. [Online]. Available: https://lumen.laravel.com/

[12] Laravel. Laravel. [Online]. Available: https://laravel.com/

[13] Smarty. **Smarty: Template engine**. [Online]. Available: http://www. smarty.net/.

[14] ADOdb. Adodb: **Database abstraction layer for php**. [Online]. Available: http://adodb.org/dokuwiki/doku.php

[15] M. Johns, **“Code-injection vulnerabilities in web applicationsexemplified at cross-site scripting,”** It-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik, vol. 53, no. 5, pp. 256–260, 2011.

[16] J. B. Killoran, **“How to use search engine optimization techniques to increase website visibility,”** IEEE transactions on professional communication, vol. 56, no. 1, pp. 50–66, 2013