



Information Technology Systems

PHP Development Frameworks

THE QUEST FOR THE HOLY GRAIL

Made by: Ruben Marques

Student #: 20162078

Supervisor: Prof. José Vasconcelos

Barcarena,



Information Technology Systems

PHP Development Frameworks

THE QUEST FOR THE HOLY GRAIL

Made by: Ruben Marques

Student #: 20162078

Supervisor: Prof. José Vasconcelos

Barcarena,

Executive Summary

This work studies the most used PHP frameworks in the market and analyzes their pros and cons. After the analysis the author will be creating a requirement list of a new framework and a prototype of the new framework is proposed.

Key-Words: PHP, Frameworks, MVC, Development, Coding

Contents

Chapter 1 – Framework.....	5
1.1 Introduction	5
1.1.1 Investigation Question.....	6
1.1.2 Objectives.....	6
1.1.3 Document organization	6
1.2 Ethical Aspects.....	6
1.3 Scheduling.....	6
1.4 Resources and Cost.....	7
Chapter 2 - Literature Review.....	8
2.1 What is PHP.....	8
2.2 What is a Software Framework.....	8
2.3 Framework Comparison.....	9
2.3.1 Laravel Framework	10
2.3.2 Symfony Framework.....	11
2.3.3 CodeIgniter Framework	11
2.3.4 Yii Framework.....	12
2.3.5 CakePHP Framework.....	12
2.4 Common PHP Framework problems	12
2.5 What they have in common.....	13
2.6 Three-Tier architecture.....	15
2.7 MVC	16
2.7.1 The Model	18
2.7.2 The View.....	18
2.7.3 The Controller.....	18
Chapter 3 – Methodology.....	20
3.1 Methodology Approach.....	20
3.2 Design Science Research (DSR).....	20
3.2.1 Awareness of Problem	21
3.2.2 Suggestion	21
3.2.3 Development	22
3.2.4 Evaluation.....	22
3.2.5 Conclusion	23
3.3 SWEBOOK	23
Chapter 4 - Solution Design	24

4.1 Architecture Requirements.....	24
4.2 MVC adaptation	24
4.2.1 Hierarchical MVC	25
4.2.2 Definitions	26
4.3 Requirements	28
4.3.1 Non-Functional Requirements	28
4.3.2 Functional Requirements	28
4.3.3 Use Cases.....	29
4.3.4 Class Diagram Framework Concept.....	30
Chapter 5 - DragonPHP.....	32
5.1 How to install, DEPLOY and run it	32
5.2 Multiple Projects In a Single Framework	34
5.2.1 Multiple Projects Code.....	35
5.3 How does routing works	35
5.3.1 Custom Routing	36
5.3 Security and Permissions Manager	37
5.3.1 Permissions Manager.....	37
5.4 CRUD Generator and Definition Files	39
5.4.1 How to create or update our crud	41
5.4.2 Why creating CRUD with definition files.....	42
5.4.3 Definition files, more than just simple CRUD	42
5.5 Trying to automate documentation	43
5.6 How to write PHP/HTML code in DragonPHP	44
5.6.1 Interface code built with twig	44
5.6.2 Engine code PHP/MySQL.....	45
Chapter 6 - Initial Developers Opinion	47
Chapter 7 – Conclusion.....	48
Chapter 8 - Bibliography.....	49

Figure Index

Figure 1 - Percentages of websites using server-side programming languages (w3Techs, 2019)	5
Figure 2 - Schedule breakdown	7

Figure 4 - Distribution of Frameworks by usage (Google Trends, 2018).....	9
Figure 5 - Distribution of top Frameworks worldwide on the left and showing Portugal preferences on the right (Google Trends, 2018)	10
Figure 6 - MVC diagram.....	17
Figure 7 - DSR diagram (Vaishnavi, 2008).....	21
Figure 8 - H-D-MVC	25
Figure 9 - Functionalities Use Case	29
Figure 10 - Class Diagram framework Concept (without methods).....	30
Figure 11 - DragonPHP Zip File	32
Figure 12 - DragonPHP after unzip.....	32
Figure 13 - DragonPHP First Time access page	33
Figure 14 - DragonPHP Creating a new project	33
Figure 15 - DragonPHP List of available projects	34
Figure 16 - DragonPHP File Structure.....	35
Figure 17 - DragonPHP Controller Found	35
Figure 18 - DragonPHP Controller content.....	35
Figure 19 - DragonPHP Routing list.....	36
Figure 20 - DragonPHP Creating new route	37
Figure 21 - DragonPHP DB Schema for permissions	38
Figure 22 - DragonPHP list of profiles	38
Figure 23 - DragonPHP Insert form example	39
Figure 24 - DragonPHP controller example	39
Figure 25 - DragonPHP definition example 1.....	40
Figure 26 - DragonPHP definition example 2.....	40
Figure 27 - DragonPHP Form Builder	41
Figure 28 - DragonPHP Workflow Builder	42
Figure 29 - DragonPHP definition example 3.....	43
Figure 30 - DragonPHP Entities builder	43
Figure 31 - DragonPHP auto documentation.....	44
Figure 32 - DragonPHP skins structure.....	44
Figure 33 - DragonPHP skin template example	45
Figure 34 - DragonPHP custom controller php code example 1.....	45
Figure 35 - DragonPHP custom controller php code example 2.....	46

Chapter 1 – Framework

1.1 INTRODUCTION

PHP programming language for quite some time was not considered as a sufficiently serious language for large Web application development making it mostly used for small projects or prototypes, and thus, leaving serious platform and software development to more elitist languages like Java, Ruby or Python.

The three languages referenced above had a big advantage in regards to PHP which was the fact that each of those languages only had one serious framework to work on making their development most cohesive, more easy to understand and stronger. Java has Spring, Python has Django and Ruby has Ruby on Rails. On the other hand PHP has, and had several dozens of frameworks, and each making their code different and impossible to integrate between them.

The most famous frameworks for PHP, according to Google Trends (Google Trend, 2019), are Laravel, Symfony, CakePHP, Zend Framework and Code Igniter, but aside from Laravel which has a big market reference every other framework has a small market percentage. In fact there are 42 known different frameworks in PHP and hundreds of less known ones (https://en.wikipedia.org/wiki/Category:PHP_frameworks, 2018)

If we look only to the numbers above we would conclude that the three initial languages (Java, Python and Ruby) are more mature and best suited for web development due to their continuity and big community.

But if we look at what languages are actually being used over the internet we see that in 2019 PHP is the chosen language with 79% usage for server side programming.

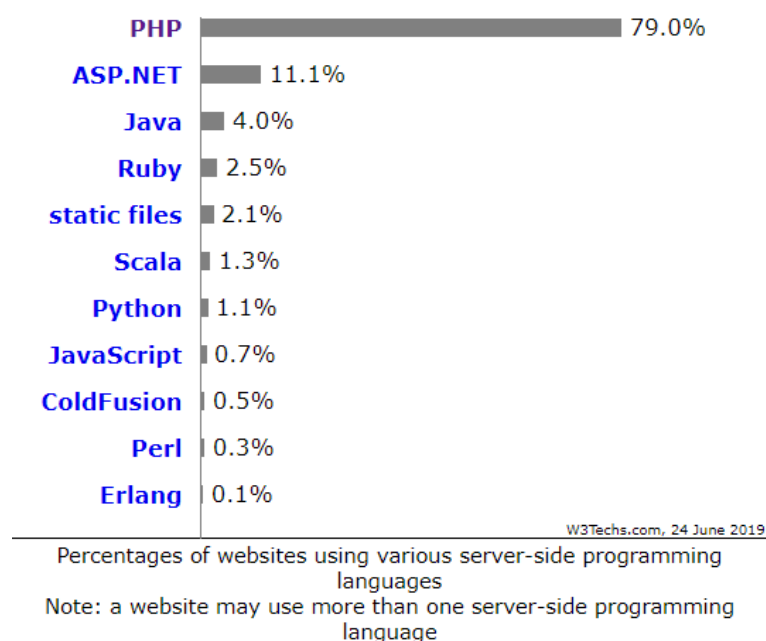


Figure 1 - Percentages of websites using server-side programming languages (w3Techs, 2019)

This discovery was the origin for this paper. The fact that PHP is still largely used it means that it is in fact a good language, but, the fact that every year a new framework appears on the market shows that, although there are 42 different frameworks for PHP, there is none which can seduce the market enough to be considered a standard.

Within this work we will be reviewing each of the top market frameworks, analyze them and understanding their pros and cons.

After this analysis we will be creating the requirements to build a new framework which can use all the good parts of the existing frameworks and improve them while still trying to keep it simple.

1.1.1 INVESTIGATION QUESTION

- Can a PHP Framework be developed without the drawbacks of the most popular frameworks in the market?

1.1.2 OBJECTIVES

- Study why there are so many PHP frameworks
- Knowing the most used frameworks in the market
- Understanding the pros and cons of each framework
- Understanding MVC pattern
- Knowing what the experience developers say about current frameworks
- Creating a new PHP Framework

1.1.3 DOCUMENT ORGANIZATION

This work is segmented in chapters and numbered topics to allow a good reading. We will start by a literature revision and then our solution architecture and requirements.

1.2 ETHICAL ASPECTS

There are no ethical aspects to consider in this work.

1.3 SCHEDULING

Below is the scheduling for the project which has been updated during the development of this document:

TASK NAME	START DATE	DAY OF THE MONTH	END DATE	DURATION (WORK DAYS)	DAYS COMPLETE	DAYS REMAINING	PERCENT COMPLETE
Framework							
Find Topic	10/1/2018	1	10/3/2018	2	2	0	100%
Literature Search	10/2/2018	2	10/5/2018	3	3	0	100%
Literature Review	10/3/2018	3	12/1/2018	59	59	0	100%
Paper Creation	11/1/2018	1	12/21/2018	50	50	0	100%
Requirements Creation							
Compile list of functionalities	12/1/2018	1	12/11/2018	10	10	0	100%
Interviews with developers to know what is needed	12/10/2018	10	12/13/2018	3	3	0	100%
Write requirements	12/12/2018	12	12/19/2018	7	7	0	100%
Document Revision	12/12/2018	12	12/21/2018	9	9	0	100%
Framework Development							
Create Framework Core	1/15/2019	15	2/2/2019	18	18	0	100%
Develop requested functionalities	2/1/2019	1	4/22/2019	80	80	0	100%
Backoffice skin development	2/23/2019	23	2/27/2019	4	4	0	100%
Results							
Analyze Results	5/1/2019	1	5/6/2019	5	5	0	100%
Write final consideration	5/10/2019	10	6/1/2019	22	22	0	100%

Figure 2 - Schedule breakdown

1.4 RESOURCES AND COST

There will not be any external cost or any resource used aside from the author work.

Chapter 2 - Literature Review

2.1 WHAT IS PHP

PHP is a server-side scripting language created by Rasmus Lerdorf in 1995 designed mainly for Web development, most specifically has a templating engine for Homepages and not has a programming language. Over time and due to community requests, PHP was extended to have web forms and to be able to communicate with databases.

Early PHP was not intended to be a new programming language, and grew organically, with Lerdorf noting in retrospect: *"I don't know how to stop it, there was never any intent to write a programming language [...] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way."* (<https://en.wikipedia.org/wiki/PHP>, s.d.)

The fact that PHP was not originally designed, but instead was developed organically has led to inconsistent naming of functions and inconsistent ordering of their parameters.

And because of that inconsistency the open source community started developing so many different frameworks for PHP.

2.2 WHAT IS A SOFTWARE FRAMEWORK

According to Riehle, Dirk (2000) a Software Framework represents the domain as an abstract design, consisting of abstract classes (or interfaces).

The abstract design is more than a set of classes, because it defines how instances of the classes are allowed to collaborate with each other at runtime. Effectively, it acts as a skeleton, or a scaffolding, that determines how framework objects relate to each other.

A framework comes with reusable implementations in the form of abstract and concrete class implementations. Abstract implementations are abstract classes that implement parts of a framework abstraction (as expressed by an abstract class or interface), but leave crucial implementation decisions to subclasses.

Frameworks use normally the principle of Design by Primitives. Design by Primitives bases a class implementation on a small set of primitive operations that are left open for implementation through subclasses. Concrete subclasses implement these operations so that they can be instantiated and used without further sub-classing.

Frameworks have key distinguishing features that separate them from normal libraries:

- **Inversion of control:** In a framework, unlike in libraries or in standard user applications, the overall program's flow of control is not dictated by the caller, but by the framework.
- **Extensibility:** A user can extend the framework - usually by selective overriding; or programmers can add specialized user code to provide specific functionality.

- **Non-modifiable framework code:** The framework code, in general, is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but should not modify its code.

How it beneficial to developers?

- Speeds up the development process by automatically creating core functionalities like CRUD.
- Follows MVC(Model-View-Controller) architecture.
- Offers many input and output filtering functions which adds an additional security layer to your web applications.
- Have their own specific debugging methods.
- Easy to organize your code and file.
- Better suitable for teamwork.

2.3 FRAMEWORK COMPARISON

From the longest list of available PHP Frameworks, here are the top 5 based on the recent analysis on Google Trends by November of 2018.

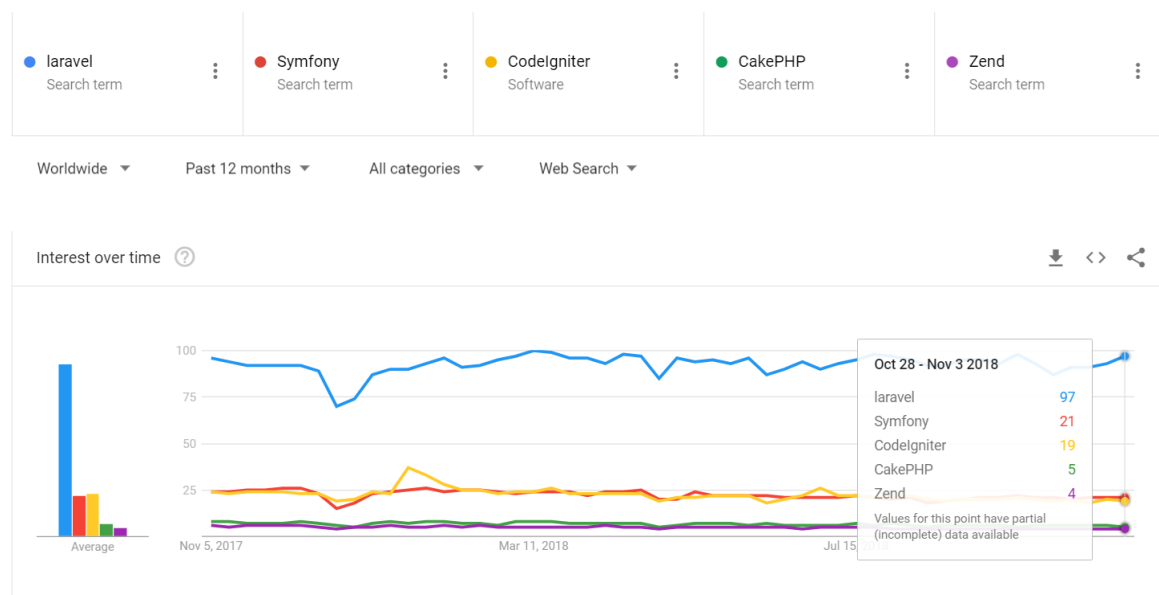


Figure 3 - Distribution of Frameworks by usage (Google Trends, 2018)

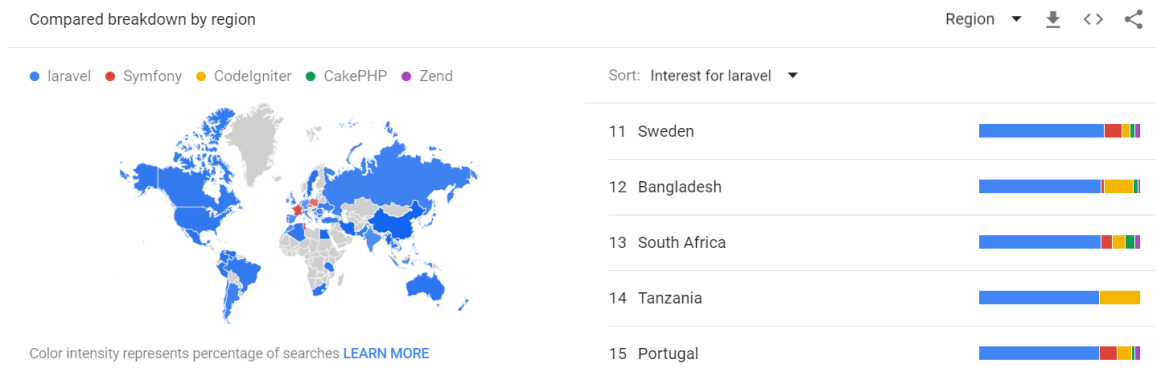


Figure 4 - Distribution of top Frameworks worldwide on the left and showing Portugal preferences on the right (Google Trends, 2018)

Based on the framework distribution seen in Figure 3 - Distribution of Frameworks by usage (Google Trends, 2018) a small description of each of the top frameworks were done. In the next chapters we will try to in a very high level detail about each of those frameworks.

2.3.1 Laravel Framework

From the Figure 4 - Distribution of top Frameworks worldwide on the left and showing Portugal preferences on the right (Google Trends, 2018), it's clear that Laravel holds the first place of all the other PHP Frameworks. Laravel is a free and open-source framework that follows an MVC Pattern, created by Taylor Otwell as a better alternative to Symfony.

Popular Versions: 5.1 ,5.5

Latest Version: 5.6

Required PHP Version: 5.5.9

Key benefits of Laravel

- Laravel Framework offers many specific packages such as,
- A lightweight template engine called as "Blade" – Helps you in different tasks like authentication, caching, sessions, routing, queuing and more.
- Artisan – A built-in command line tool, allows you to perform the most of the tedious and repetitive task.
- MVC Architecture – Allows you to speed up the development process and improve performance.
- Unit Testing – Lets you run many tests, ensure that the new changes won't affect any other thing in the web application.
- Eloquent ORM – Gives the ability to issue database queries in PHP syntax & avoids the hassle of writing SQL code.
- Security – Laravel offers you with a built-in security. It uses "Bcrypt Hashing Algorithm" to better secure your passwords.

2.3.2 Symfony Framework

The next most recognized PHP Framework in the list is Symfony. It is a reliable, modular and high performing framework developed by Sensio Labs with an aim to speed up the creation and maintenance of web applications.

Latest Version: 4.1 (Support up to Jan 2019)

Popular Version: 3.4 (Support up to Nov 2020)

Required PHP version: 5.5.9

Key benefits of Symfony

- Offers High Performance due to the use of Bytecode caching.
- PHP libraries are reusable, which helps in different tasks like routing, authentication, templating, object configuration and more.
- Well documented and excellent support system.
- Easy to use and maintain.

2.3.3 CodeIgniter Framework

The next framework in our list is CodeIgniter. It is a simple, flexible, lightweight and easy to use free PHP framework created by Rick Ellis. CodeIgniter is often known for its speed when compared to other Frameworks.

Latest version: 3.1.9

Legacy version: 2.x

Required PHP version: 5.4

Key benefits of CodeIgniter

- Offers a strong security as it includes built-in protection against XSS, CSRF such attacks.
- User-friendly interface lets you develop a dynamic, flexible and secure web application.
- Exceptional performance capabilities and outperforms most of the other frameworks.
- Provides you with a clear, easy to understand documentation, tutorials and user guides.
- Easy to spot error functions.

2.3.4 Yii Framework

The PHP Framework which occupies the 4th position in the list is Yii PHP Framework. It is a fast, secure and high-performance PHP Framework, started as an attempt to overcome the drawbacks of the PRADO Framework. Yii is known for its easy installation and extensibility.

Latest version: 2.0

Required PHP version: 5.1

Key benefits of Yii

- Generates code automatically for skeleton and CRUD application
- Works well with third-party applications
- Follows MVC design architecture and offers a strong caching support.
- Offers a short rapid development time
- One of the oldest PHP framework and provides support till data.

2.3.5 CakePHP Framework

Finally, the last Framework in our list is CakePHP. It is easy to learn with CRUD embedded in it. It started, when a polish programmer Michal Tatarynowicz wrote a minimal version of RAD in PHP.

Latest version: 3.6 (Red Velvet)

Required PHP version: 5.5.9

Key benefits of CakePHP

- Offers a built-in caching, authentication, database access, and more.
- Lets you prevent cross-site scripting and SQL Injection.
- Follows Zero configuration, so you no longer need to specify the library location or site URL, everything is auto-detected.
- Easy to run the test to check the critical points of your web application.

2.4 COMMON PHP FRAMEWORK PROBLEMS

Although no paper was found stating why the current frameworks are not good, or why it is hard to use them, there were several articles found in the internet written by experience developers stating several reasons why framework A or framework B are not good.

The most referred reasons are:

- **Big learning curve:** Most frameworks require you to read a lot of documentation to even do a simple CRUD;

- **Hard to install and run it:** One of the common complaints within the community and in the interviews done regarding the above frameworks is the fact that it requires some knowledge in order to install and do the first setup making the entrance barrier harder for them.
- **Limit control:** Although most frameworks provides methods to build some functionalities if the developer tries to change the behavior or small details of those functionalities it is close to impossible;
- **Requires a notebook to do simple commands:** Most frameworks currently works by building, enabling and / or configuring its framework via CLI (Command Line Interface) making it hard to remember every single command;
- **Not build to be a modular application:** Due to the nature of MVC it becomes very easy as time passes to start having bad coding across the application;

Maybe the most important comment comes from Rasmus Lerdorf, creator of PHP

He went on to say:

“While they all suck. Everyone needs a framework. What everyone doesn’t need is a general purpose framework. Nobody has a general problem. Everyone has a very specific problem they’re trying to solve. And a general purpose framework, while it can solve it, it usually solves it in a way that you get so many other things that you don’t need... that ends up being done on every request.”

He goes onto recommend using “targeted frameworks” for targeted problems:

“Usually, I tell people to look for a targeted framework. So, if you have a problem that looks a lot like a blogging problem. Maybe, WordPress should be your framework... if your problem is very close to something WordPress can handle, chances are, you’ll be using most of WordPress. There won’t be all these other general purpose things you won’t touch.”

2.5 WHAT THEY HAVE IN COMMON

By doing an analysis of the frameworks we conclude that there are some common technics and patterns which all of the frameworks follow or, at least they try to adopt and adapt.

The table below shows the characteristics and common functionalities of each framework (PHP Frameworks, 2018)

	CakePHP	Symfony	Laravel	Zend	CodeIgniter	Yii
PHP5	Yes			Yes	Yes	
PHP7	Yes	Yes	Yes	Yes	Yes	Yes
MVC	Yes	Yes	Yes	Yes	Yes	Yes
Multiple DB's	Yes	Yes	Yes	Yes	Yes	Yes
ORM	Yes	Yes	Yes	Yes		Yes
DB Objects	Yes	Yes	Yes	Yes	Yes	Yes
Templates			Yes	Yes	Yes	Yes
Caching	Yes	Yes	Yes	Yes	Yes	Yes
Validation	Yes	Yes	Yes	Yes	Yes	Yes
Ajax	Yes	Yes	Yes	Yes		Yes
Auth Module	Yes	Yes	Yes	Yes		Yes
Modules	Yes	Yes	Yes			Yes
EDP						Yes

MVC: Indicates whether the framework comes with inbuilt support for a Model-View-Controller setup.

Multiple DB's: Indicates whether the framework supports multiple databases without having to change anything.

ORM: Indicates whether the framework supports an object-record mapper, usually an implementation of ActiveRecord.

DB Objects: Indicates whether the framework includes other database objects, like a TableGateway.

Templates: Indicates whether the framework has an inbuilt template engine.

Caching: Indicates whether the framework includes a caching object or some way other way of caching.

Validation: Indicates whether the framework has an inbuilt validation or filtering component.

Ajax: Indicates whether the framework comes with inbuilt support for Ajax.

Auth Module: Indicates whether the framework has an inbuilt module for handling user authentication.

Modules: Indicates whether the framework has other modules, like an RSS feed parser, PDF module or anything else (useful).

EDP: Event Driven Programming.

2.6 THREE-TIER ARCHITECTURE

The objective of the three-tier architecture is to break an application/solution into its constituent parts, as follows:

- Presentation tier
- Application tier
- Data access tier

Presentation tier

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer which users can access directly (such as a web page, or an operating system's GUI).

Application tier (business logic, logic tier, or middle tier)

The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing.

Data access tier

The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. As with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability.

Web development usage

In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:

A front-end web server serving static content, and potentially some cached dynamic content. In web-based application, front end is the content rendered by the browser. The content may be static or generated dynamically.

A middle dynamic content processing and generation level application server (e.g., Symfony, Spring, ASP.NET, Django, Rails, Node.js).

A back-end database or data store, comprising both data sets and the database management system software that manages and provides access to the data.

Within the application tier, where our solution will be mainly focused, we will be using a variant of the design pattern MVC.

2.7 MVC

Web applications are composed of both Front end and back end technologies. Due to the evolution of the World Wide Web and the fast pace in which it evolved, developers need to use a large number of technologies to build a single Web Application. This resulted in complex and often, difficult to maintain and fix, solutions.

Due to this, a web application is generally built by a team of specialized developers, each working on its own technology and tier. HTML and CSS for the presentation layer, JavaScript for client-side interaction,

PHP (or ASP, Java, Python, Pearl, Ruby, etc.) for server-side logic and MySQL (or Oracle Database, Microsoft SQL Server, etc.) for data storage and management.

Each of these specialist needs to work with other specialized developers in such a way that their code pieces fit inside the overall design of the application.

For example, the client-side (data presentation) developer needs to alter the HTML and CSS code in such a way that he doesn't break the server-side developers' code that resides in the same file. Also, when a database developer alters the schema for an application the server-side developer may need to change a lot of code to make the application work.

The important thing to note here is that there is an acute need to separate presentation from logic and data storage in an application. There are some application design paradigms such as MVP (MVP, 2019) and MVVM (MVVM, 2019) offer solutions to this problem, but the focus is still on the MVC pattern.

The MVC pattern

In this section we'll review the present standing of the analysis during this field and take a glance at the literature behind the MVC pattern, describing the most useful elements of the pattern.

The MVC style pattern was first unreal by Trygve Reenskaug within the Nineteen Seventies at the Xerox Parc. According to him, "the essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer".

Later on, in 1988, the MVC paradigm was described in detail by Krasner and Pope in their article "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", published in the Journal of ObjectOriented Programming.

They stress out that there are enormous benefits to be had if one builds applications with modularity in mind. "Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit without having to know everything about the other units."

An application is split into 3 main categories: the model of the most application domain, the presentation of data therein model and user interaction.

The MVC pattern splits responsibilities into 3 main roles therefore with additional economical collaboration.

These main roles are development, style and integration.

The development role is taken on by skilled programmers that are chargeable for the logic of the application. They be sure of information querying, validation, process and additional.

The design role is for the developers that are chargeable for the appliance look and feel. They show information that is fed from the developers functioning on the primary role.

The integration role gathers developers with the responsibility to connect the work of the previous 2 roles.

The MVC style pattern is such a decent suitable net application development as a result of they mix many technologies typically split into a group of layers. Also, MVC specific behavior might be to send specific views to different types of user-agents.

“User interaction with an MVC application follows a natural cycle: the user takes an action, and in response the application changes its data model and delivers an updated view to the user. And then the cycle repeats. This is a very convenient fit for web applications delivered as a series of HTTP requests and responses.”

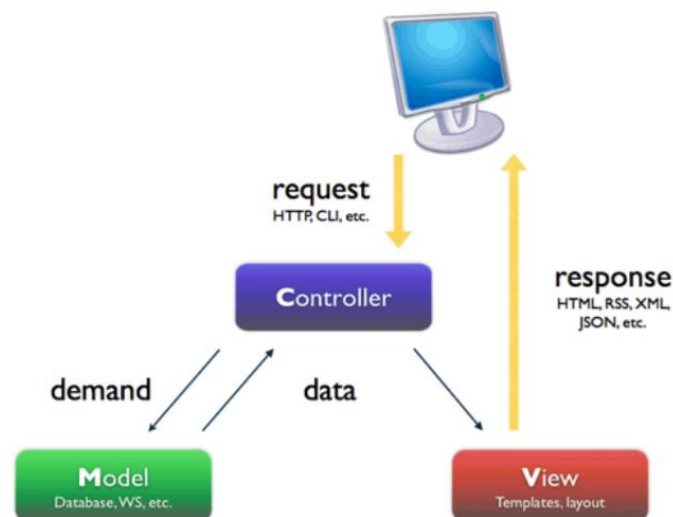


Figure 5 - MVC diagram

2.7.1 The Model

“The Model is the part of the system that manages all tasks associated with data: validation, session state and management, data supply structure (database). The Model greatly reduces the quantity of the code the developer has to write.

The Model layer is accountable with the business logic of associate application. it'll encapsulate strategies to access data (databases, files, etc.) and can build a reusable category library out there. sometimes a Model is constructed with information abstraction in mind, validation and authentication.

Moreover, the Model is created of categories that outline the domain of interest. These objects that belong to the domain typically times encapsulate information that's keep in databases, however conjointly embrace code that's accustomed manipulate this data and enforce business rules.

As a conclusion, the Model primarily handles information access abstraction and validation. The Model holds strategies for interaction with totally different information sources.”(Dragos-Paul Pop*, 2013)

2.7.2 The View

The view is accountable for graphical computer programming management. This implies all forms, buttons, graphic elements and every different HTML parts that are within the application. Views may also be accustomed generate RSS content for aggregators or Flash displays. By separating the presentation from the logic of the application we tend to greatly cut back the danger of errors occurrence once the designer decides to change the interface of that application. At the same time, the developers' job is greatly reduced because he will no longer need to see HTML code parts, style parts and graphical parts.

The view layer is what will ordinarily be referred to as internet style or templates. It controls the approach information is displayed and how the user interacts with it. It conjointly provides ways that for information gathering from the users. The technologies that are mainly employed in views are HTML, CSS and JavaScript. (Dragos-Paul Pop*, 2013)

2.7.3 The Controller

The Controller is to blame for event handling. These events may be triggered by either a user interacting with the application or by a system method. A controller accepts requests and prepares the information for a response. It is also responsible with establishing the format of that response. The Controller interacts with the Model so that he as to retrieve the required information and generates the read. This method is additionally called associate action or a verb. Once the request arrives at the server, the MVC framework dispatches it to a technique during a controller supported the URL.

The Controller binds all application logic and combines the show within the read with the practicality within the Model. it's accountable with information retrieval from the read and

with establishing the execution path for the application. The Controller can access the Model practicality and it'll interpret the information received so it may be displayed by the read. it's conjointly accountable with error handling.

A Controller manages the link between a read and a Model. It responds to user requests, interacts with the Model and decides that read ought to be generated and displayed.
(Dragos-Paul Pop*, 2013)

Chapter 3 – Methodology

3.1 METHODOLOGY APPROACH

There are several methods of investigation which allows a wide choice of methodology to be applied allowing the use of the one that best fits the area to be worked.

Thus, since there are several methods of investigation, there are also several methodological approaches that can be adopted in research projects. Their choice is made based on the nature of the problem and what is intended. Some of these methodological approaches are: Design Science Research (DSR), Research-Action Method, Delphi Method, Case Study Method, etc...

However, the one that fits this research project will be the DSR methodology which will enable the author to create and evaluate an artifact with the intention of solving identified problems.

Along side with DSR, the author also used SWEBOK, Software Engineering Body of Knowledge, most specifically the three areas of knowledge regarding Requirements, Design and Construction.

3.2 DESIGN SCIENCE RESEARCH (DSR)

Due to the nature of the research it was chosen to use the Design Science Research methodology adapted by Vijay K. Vaishnavi and William Kuechler Jr.

Design science research focuses on the development and performance of (designed) artifacts with the explicit intention of improving the functional performance of the artifact. Design science research is typically applied to categories of artifacts including algorithms, human/computer interfaces, design methodologies (including process models) and languages. Its application is most notable in the Engineering and Computer Science disciplines, though is not restricted to these and can be found in many disciplines and fields.

In design science research, as opposed to explanatory science research, academic research objectives are of a more pragmatic nature. Research in these disciplines can be seen as a quest for understanding and improving human performance. Such renowned research institutions as MIT's Media Lab, Stanford's Centre for Design Research, Carnegie-Mellon's Software Engineering Institute, Xerox's PARC and Brunel's Organization and System Design Centre use the Design Science Research approach. ([https://en.wikipedia.org/wiki/Design_science_\(methodology\)](https://en.wikipedia.org/wiki/Design_science_(methodology)), s.d.)

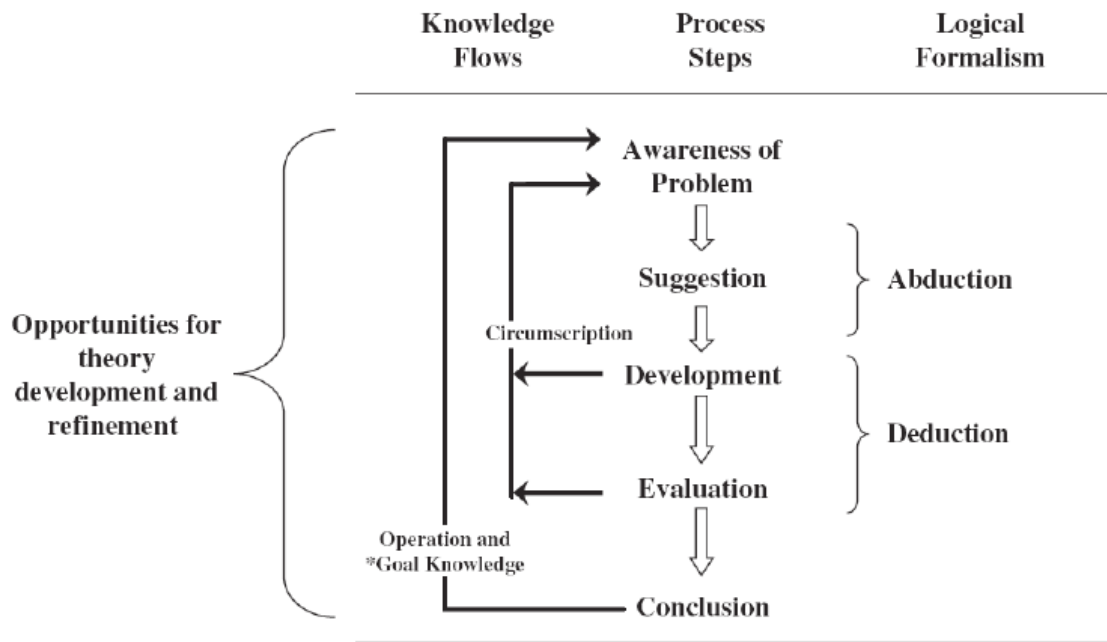


Figure 6 - DSR diagram (Vaishnavi, 2008)

Below are the process steps explanation described by Vijay K. Vaishnavi and William Kuechler Jr.

3.2.1 Awareness of Problem

An awareness of an interesting problem can come from multiple sources: new developments in industry or in a reference discipline. Reading in an allied discipline may also provide the opportunity for application of new findings to the researcher's field. The output of this phase is a Proposal, formal or informal, for a new research effort.

3.2.2 Suggestion

The Suggestion phase follows immediately behind the Proposal and is intimately connected with it, as the dotted line around Proposal and Tentative Design (the output of the Suggestion phase) indicates. Indeed, in any formal proposal for design science research, such as one to be made to the NSF (National Science Foundation) or an industry sponsor, a Tentative Design and likely the performance of a prototype based on that design would be an integral part of the Proposal. Moreover, if after consideration of an interesting problem, a Tentative Design does not present itself to the researcher, the idea (Proposal) will be set aside. Suggestion is an essentially creative step wherein new functionality is envisioned based on a novel configuration of either existing or new and existing elements. The step has been criticized as introducing no repeatability into the design science research method; human creativity is still a poorly understood cognitive process. However, the step has

necessary analogues in all research methods; for example, in positivist research, creativity is inherent in the leap from curiosity about organizational phenomena to the development of appropriate constructs that operationalize the phenomena and an appropriate research design for their measurement.

3.2.3 Development

The Tentative Design is further developed and implemented in this phase. Elaboration of the Tentative Design into complete design requires creative effort. The techniques for implementation will of course vary, depending on the artifact to be constructed. An algorithm may require construction of a formal proof. An expert system embodying novel assumptions about human cognition in an area of interest will require software development, probably using a high-level package or tool. The implementation itself can be very pedestrian and need not involve novelty beyond the state-of-practice for the given artifact; the novelty is primarily in the design, not the construction of the artifact.

3.2.4 Evaluation

Once constructed, the artifact is evaluated according to criteria that are always implicit and frequently made explicit in the Proposal (Awareness of Problem phase). Deviations from expectations, both quantitative and qualitative, are carefully noted and must be tentatively explained. That is, the evaluation phase contains an analytic sub-phase in which hypotheses are made about the behavior of the artifact. This phase exposes an epistemic fluidity that is in stark contrast to a strict interpretation of the positivist stance. At an equivalent point in positivist research, analysis either confirms or contradicts a hypothesis. Essentially, save for some consideration of future work as may be indicated by experimental results, the research effort is finished. For the design science researcher, by contrast, things are just getting interesting. Rarely, in design science research, are initial hypotheses concerning behavior completely borne out. Instead, the evaluation phase results and additional information gained in the construction and running of the artifact are brought together and fed back to another round of Suggestion (cf. the circumscription arrows of Figures 2.3 and 2.5). The explanatory hypotheses, which are quite broad, are rarely discarded; rather, they are modified to be in accord with the new observations. This suggests a new design, frequently preceded by new library research in directions suggested by deviations from theoretical performance. (Design science researchers seem to share Allen Newell's concept [from cognitive science] of theories as complex, robust nomological networks.) This concept has been observed by philosophers of science in many communities (Lakatos, 1978); and working from it, Newell suggests that theories are not like clay pigeons, to be blasted to bits with the Popperian shotgun of falsification. Rather, they should be treated like doctoral students. One corrects them when they err, and is hopeful they can amend their flawed behavior and go on to be evermore useful and productive (Newell, 1990).

3.2.5 Conclusion

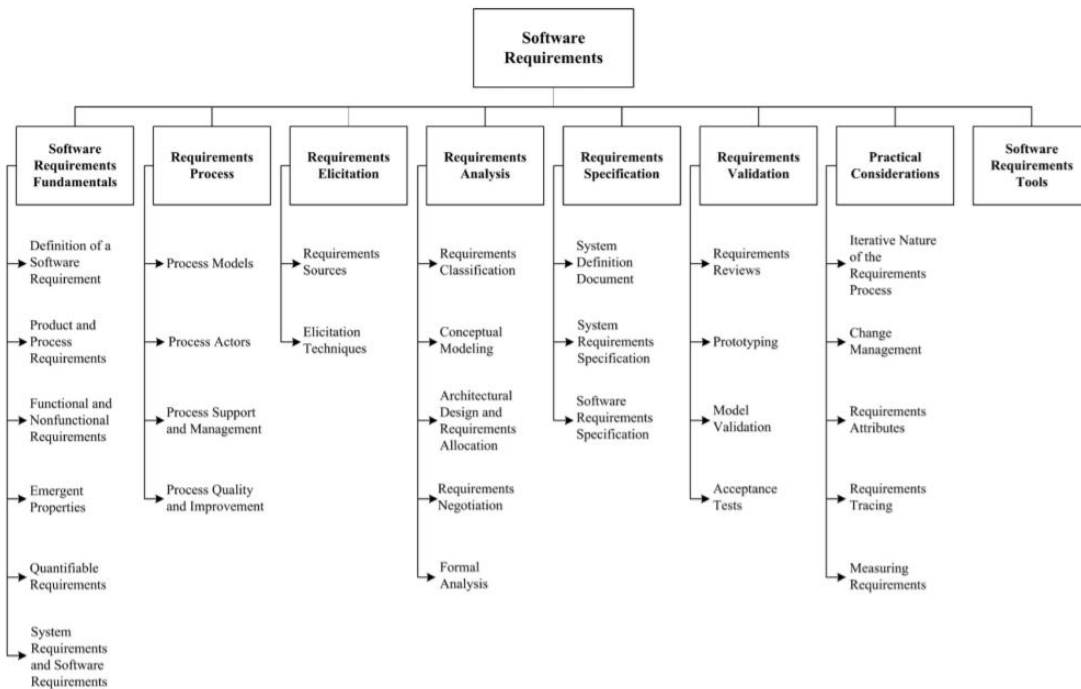
This phase is the finale of a specific research effort. Typically, it is the result of satisficing; that is, although there are still deviations in the behavior of the artifact from the (multiply) revised hypothetical predictions, the results are adjudged “good enough.” Not only are the results of the effort consolidated and “written up” at this phase, but the knowledge gained in the effort is frequently categorized as either “firm” — facts that have been learned and can be repeatedly applied or behavior that can be repeatedly invoked — or as “loose ends” — anomalous behavior that defies explanation and may well serve as the subject of further research.

3.3 SWEBOK

Software Engineering Body of Knowledge (SWEBOK Guide) was established between a partnership between IEEE and ACM with the aim of promoting Software Engineering and to create a set of guides about the areas of knowledge of SE.

The main objectives of SWEBOK are:

- Provide a consistent view of Software Engineering worldwide;
- Clear the boundaries of Software Engineering with respect to other disciplines such as computer science, project management, computing, mathematics, among others;
- Characterize the contents of the Software Engineering discipline;
- Provide access to the topics of the knowledge body of Software Engineering;
- Provide a basis for curriculum development and individual certification;
- Serve as support material.



Chapter 4 - Solution Design

In this chapter we will go thru the requirements needed for the solution and how it will be built.

4.1 ARCHITECTURE REQUIREMENTS

The development of the framework must be based on the **three-tier** architecture and every solution built upon the framework should maintain this design pattern. Within the application tier we will want to use an adaptation of the **MVC** pattern.

4.2 MVC ADAPTATION

In this work we have analyzed what is the MVC pattern and how important is the usage of the pattern on building scalable applications. And although it is a great design pattern and highly used by the development community, the author believes that there are ways to improve this pattern by introducing two new concepts: Hierarchy and Meta-Language.

Hierarchy must be built on top of the MVC pattern in order to avoid the big blocks of code that normally go with this pattern. The framework must be able to have several levels of each of the Model, View and Controller and each must be built using the extend paradigm of the object-oriented program.

Another of the problems commonly seen in the MVC pattern is that, although it allows the separation of code it also brings a problem for scalability and continuity of the applications, mainly because applications grow and adapt as time changes and as such, new code blocks

must be introduced and refactored. By adding a Meta language block in between we guarantee that this Meta Language will maintain the same across the years and allows the developers to refactor, enhance and improve their applications without the need to change or brake anything within this meta language blocks.

This will evolve the MVC pattern into H-D-MVC (Hierarchy Definition, Model View Controller).

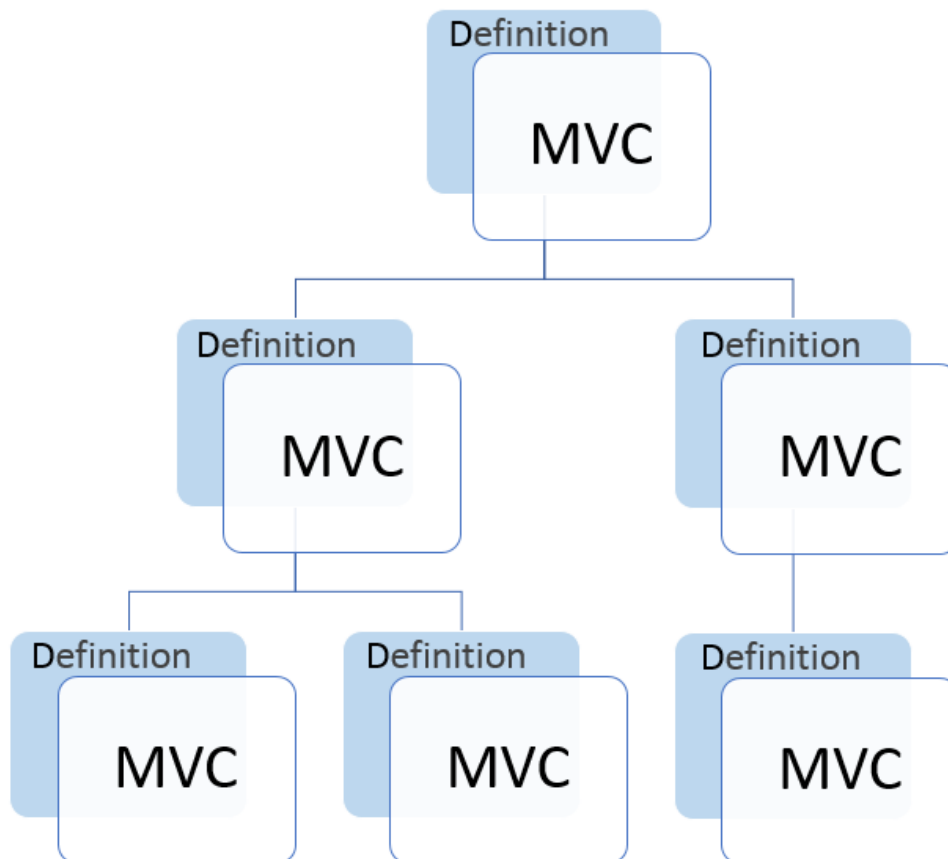


Figure 7 - H-D-MVC

4.2.1 Hierarchical MVC

According to Ahmad, Zainal Arifin and Dyna Marisa Khairina, due to the MVC architecture constraints, MVC developed into a more modular architecture called HMVC (Hierarchical Model-View-Controller). (Ahmad)

HMVC split into sections per module. Each module has its own MVC layer, so that when imported moved enough alone. Not only that, HMVC then allows a module has a derivative like hierarchy. HMVC first applied to desktop applications.

From there, the author decided to apply the HMVC to web framework.

HMVC (Hierarchical Model-View-Controller) Architecture consists of a layer of MVC (Model-View-Controller) which can have a hierarchical relationship derivative. HMVC is an MVC pattern but a hierarchy in which the MVC implementation stored in specific modules so that each module of the model, view, and controller itself.

HMVC architecture provide clear boundaries for each layer. Limiting layer is referred to as a module (module). Because HMVC is a development of the MVC architecture, the term MVC will continue to be used in its application. Each module has the MVC which stands independently. A module will not interfere with other modules. By doing so, the module is said to have clear boundaries.

When compared to the MVC architecture, HMVC architecture has several advantages, namely:

1. Modularity: HMVC makes it easier to interchange chunks of code through encapsulation of functionality.
2. Organization: HMVC allows each module to contain its own models, views, and controllers directories, keeping related files contained.
3. Extensibility: By promoting modularity, HMVC allows systems to be more easily extended by simply adding or replacing modules.
4. Reusability: Since modules can call each other, HMVC allows for code to be easily reused.

4.2.2 Definitions

The final concept to be introduced in the H-D-MVC is the Definition, a Markup Language file written in Yaml or Xml, similar to HTML which will be used as an abstraction layer to the MVC.

A description of a Markup Language can be: "A (document) markup language is a modern system for annotating a document in a way that is syntactically distinguishable from the text. The idea and terminology evolved from the "marking up" of paper manuscripts, i.e., the revision instructions by editors, traditionally written with a blue pencil on authors' manuscripts. In digital media this 'blue pencil instruction text' was replaced by tags, that is, instructions are expressed directly by tags or 'instruction text encapsulated by tags'." (Wikipedia, 2019)

The objective of adding definitions to the HMVC model is to allow a faster, more simplistic and backwards compatible approach to create MVC components. By using a Markup Language which works as an upper level of MVC we simplify the development process of the most common functionalities of the framework, while allowing its development to be done by any developer regardless of its experience.

This is done by usage of a natural language to define behaviors, looks and interactions of every page and forms in our framework.

In a more pragmatic view, a definition is composed of several tags in a specific order, and each tag has a very specific behavior. By using this markup language we can create a new

level of abstraction in our framework and guarantee that the result of a page developed with a Definition will always behave the same regardless of where it is run.

If we view an example taken from a definition in DragonPHP we can have a better glance of its meaning:

```
1  definition:
2  datasource:
3      table: users
4      pkey: id
5  form:
6      action: insert
7      title: Insert User
8  view:
9      skin: dragon
10     template: forms
11  fields:
12     id:
13         label: Id
14         data_type: integer
15         insertable: false
16         updatable: false
17     username:
18         label: Username
19         data_type: varchar
20         field_size: 100
21         edit_size: 50
22         validation:
23             required: true
24     password:
25         label: Password
26         data_type: password
27         field_size: 100
28         edit_size: 50
29         validation:
30             required: true
```

This definition, when called, will create an HTML form which will be used to insert a new user into our database. Since it is a layer to our MVC model what its being done by the framework is to tell to the Model where to save the information (Database, in a table called Users), which fields exists in our Users table (id, username and password) and their data type (If it is a text, integer, varchar, etc...).

It will tell the Controller to load a specific template file (Skin Dragon and Template forms), how to behave when accessing the page (Action: Insert).

Lastly it will tell our View how to construct the html code to be display, with, the corresponding validation rules.

The objective of the Definitions in DragonPHP is to allow developers to create any page without having to write any HTML, CSS, JavaScript or PHP code. The behavior of each tag is then done by default by the framework without the need of any interference of the developers, nevertheless, since this information is being sent to our MVC model, it means that if needed, developers can manipulate its data before rendering any page.

One of the main goals of having this definition files, is that, it allows DragonPHP to have several WYSIWYG tools to create forms and workflows with a clean interface and without any fear of having code problems.

4.3 REQUIREMENTS

By studying the top frameworks on the market and by the usage of surveys, below are the most important requirements found on Chapter 2 of this paper that the solution must have.

4.3.1 Non-Functional Requirements

Non-functional requirements will focus of the quality, scalability and usability of the framework.

- **Performance:** The framework must be built to handle hundreds of requests simultaneously and to handle / present several millions of records in its database within a reasonable amount of time.
- **Usability:** All functionalities built must be accessed via a web interface without the need of a command line interface (CLI). These functionalities must be built using the UX market standards.
- **Scalability:** Applications built on top of the framework must have the ability to enhance the system by adding new functionality at minimal effort.
- **Security:** The framework must be able to protect the applications built on top against XSS and SQL injections attacks.
- **Portability:** The framework and the applications built on top must be accessed and visualized in a desktop environment and on a mobile.
- **Singleton:** The objective of the framework is to allow the developers to build applications on top of the framework, and the framework must be able to encompass multiple different applications in the same framework instance.

4.3.2 Functional Requirements

Many of the functional requirements proposed for this work were gathered from the characteristics and functionalities studied on Chapter 2 of this paper, however with a interface and usability difference since most frameworks studied on Chapter 2 did the below functionalities without an interface, but rather via command line, or by code.

To have a fast application development using the framework, there are some functional requirements which are essential for the framework, such as:

- **Form Builder:** The developer must be able to build CRUD forms with a WYSIWYG (What you see is what you get) interface. This CRUD forms must be done in a fast and simple way to allow everyone, even non-experience developers, to build, create and maintain multiple forms.
- **Workflow Builder:** One of the most common problems developers face when developing applications is the usage of multiple workflow within their applications. The framework must provide an easy way to be able to create custom workflows based on BPM to allow the use of various methods to discover, model, analyze, measure, improve, optimize, and automate business processes.
- **Auto-Documentation:** The framework must be able to auto document as much as possible every application built on top of it by creating entity schemas, database schemas, and code comment analysis.
- **Database Abstraction:** The framework must allow the usage of any of the common databases on the market such as MySQL, Oracle, PostgreSQL, MariaDB.
- **Permission Management:** The framework must provide a functionality to manage permissions in a simple and intuitive way.
- **Pretty URLs:** The applications built on top of the framework must have a routing system to allow the pretty URL functionality.

4.3.3 Use Cases

Below is a use case of the mandatory functionalities that the framework needs to make available for the developers.

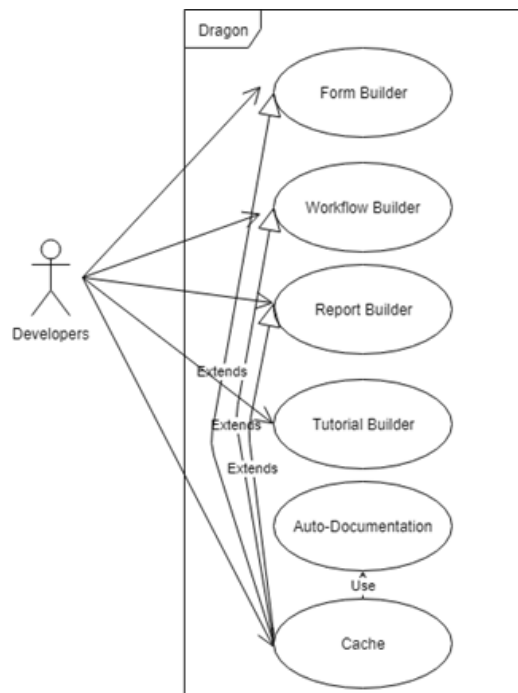


Figure 8 - Functionalities Use Case

4.3.4 CLASS DIAGRAM FRAMEWORK CONCEPT

In this diagram is a concept of how the framework should behave when a request is done.

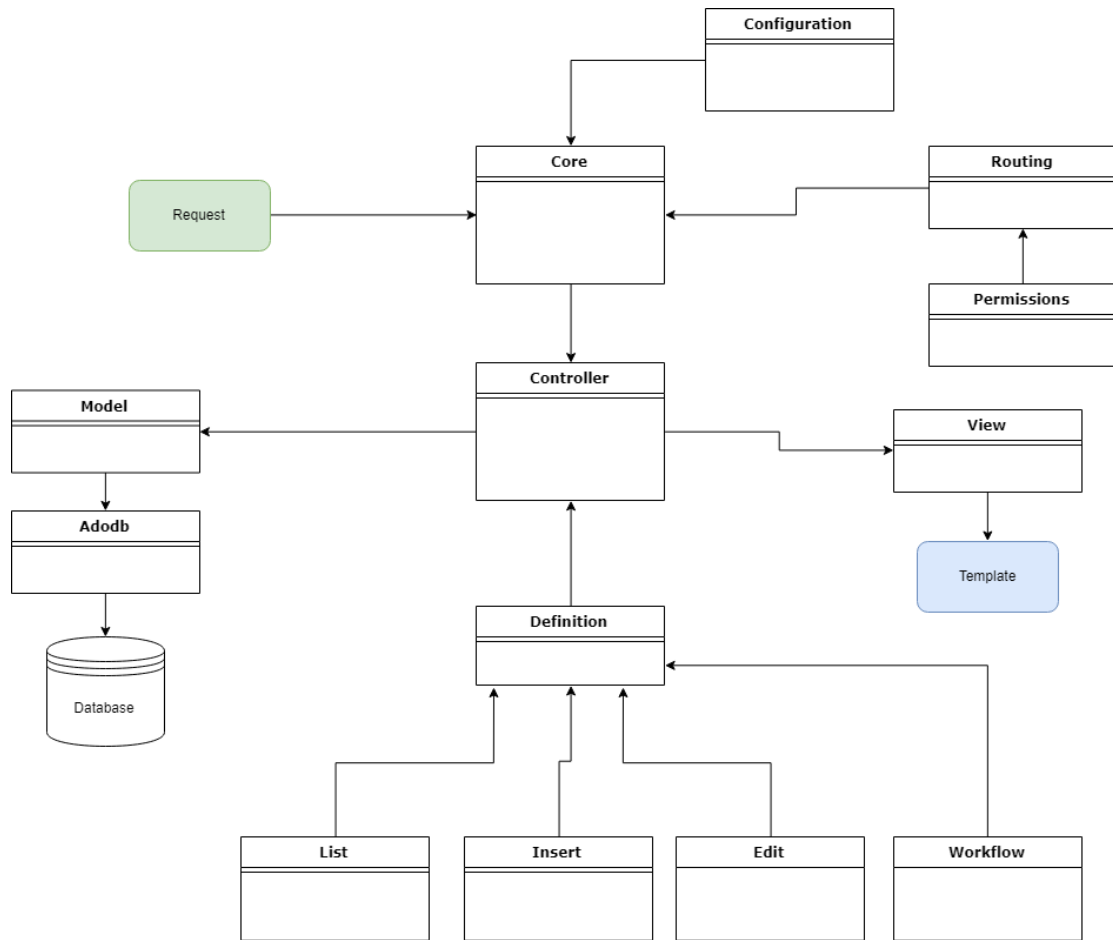


Figure 9 - Class Diagram framework Concept (without methods)

Chapter 5 - DragonPHP

Over the course of some months the author was able to create a prototype of his PHP Framework called DragonPHP.

In the next topics we will be going by each of the functionalities developed and how it tried to overcome the problems found with other frameworks while still being able to maintain most of the key elements that makes those frameworks good.

By the end of this paper we will conclude that it is in fact possible to create a framework that can be used and appeal to both junior and senior developers.

5.1 HOW TO INSTALL, DEPLOY AND RUN IT

One of the biggest complains found when speaking with junior developers was the fact that the top frameworks in the market are hard to install, and requires knowledge in several technologies (bash, composer, php, etc...)

This made some of the most junior developers to give up on those frameworks because they couldn't have it running properly.

With DragonPHP the author made the deployment as easy as possible.

- 1- Install one of the WebServer available on the internet such as:
 - a. Wamp: <http://www.wampserver.com/en/>
 - b. Xampp: <https://www.apachefriends.org/index.html>
 - c. Nginx: <https://www.nginx.com/>
- 2- Download and unzip DragonPHP framework to the www folder that the WebServer is serving:

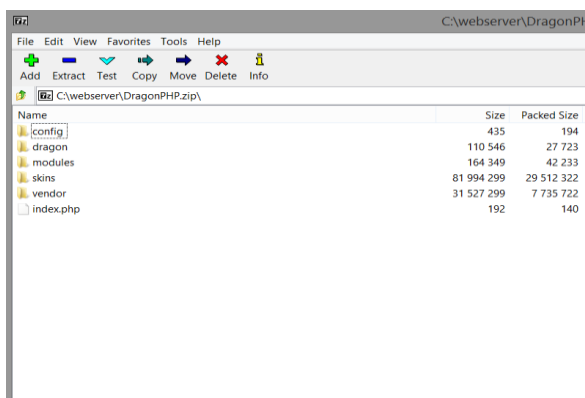


Figure 10 - DragonPHP Zip File

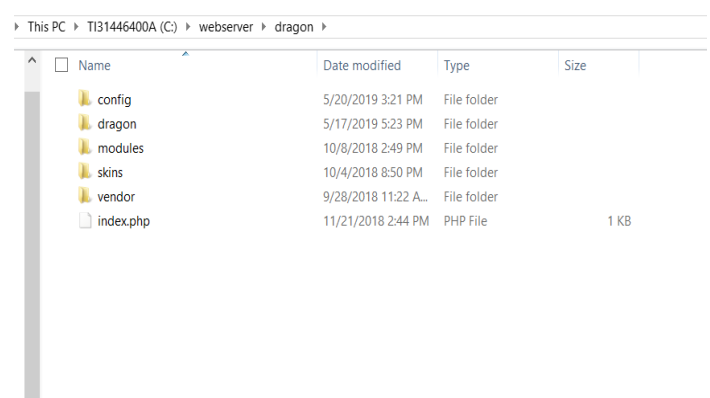


Figure 11 - DragonPHP after unzip

- 3- Open the browser and access the localhost. The framework will detect it is the first time running it and it will present two pages with simple configurations in order to

start working with the framework. In the first page the framework will ask for database connection information: Server Name, Username and Password:

Figure 12 - DragonPHP First Time access page

The information on the right side will be automatically populated and the user will not need to do anything.

After clicking on Save Config, and if the framework can connect to the database, the framework will ask for information regarding the first project:

Figure 13 - DragonPHP Creating a new project

After submitting the information the project will be configured and ready to be used.

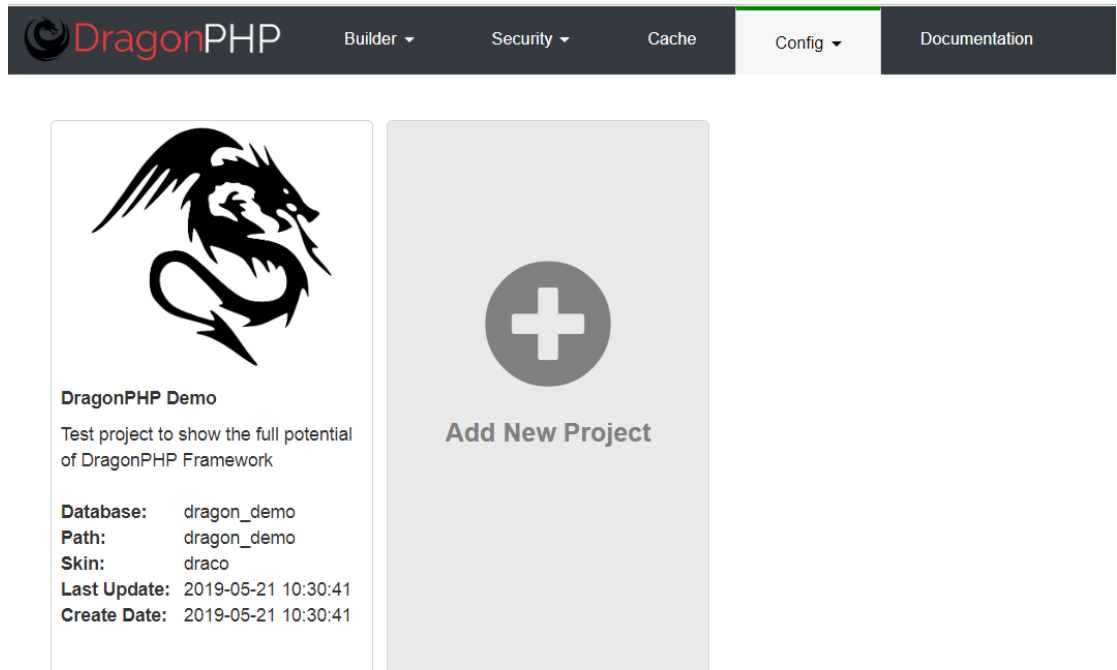


Figure 14 - DragonPHP List of available projects

5.2 MULTIPLE PROJECTS IN A SINGLE FRAMEWORK

DragonPHP was built in such a way that allows a developer to have a single instance of the Framework running while still having multiple different projects with different databases running in the same instance.

To view all active projects the developer may go to the framework development backoffice, i.e. <http://localhost/dev/> and access the option on the top: Config -> Projects

In that page there will be listed all projects currently configured in the framework. In order to witch between projects the developer will have the option "Make default project" beneath each project card.

5.2.1 Multiple Projects Code

Below is the file structure of DragonPHP after installing it:

The folders Config, Dragon and Tmp will be used by DragonPHP and that's where all the framework code resign.

With in the Specific folder (marked in red) we will have one folder for each project. Within that folder we will have specific code for that specific project.

Marked in blue we will have the folder Modules and Skins which are available for all projects.

This allows us when building an application using DragonPHP to have generic modules which are used in all projects, for example a module to manage Users, and to have within each project only

the code specific tailored to that project.

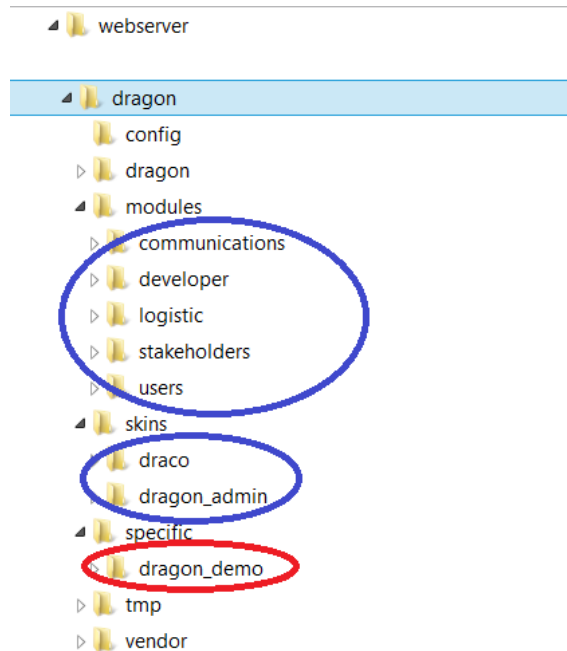


Figure 15 - DragonPHP File Structure

5.3 HOW DOES ROUTING WORKS

Routing is one of the most important parts of a framework, and in DragonPHP the author tried to make it simple and straight forward in terms of both configuration and usage.

By accessing a specific URL, i.e.: <http://localhost/pickup/list> the framework will, by default, break it in two or more sections. In this URL the first section is "pickup", which will be the controller, and the second section is the method "list".

In order to show the correct page the framework will first search for the controller file within the specific path, if found it will check if the method exists within that controller, if both exists then that is what is going to be present. If the framework fails in one of the searches then it falls back to the path /modules/

In this example the framework found the file pickup.controller.php within the specific path

with the corresponding method, and as such, that is what is going to be executed.

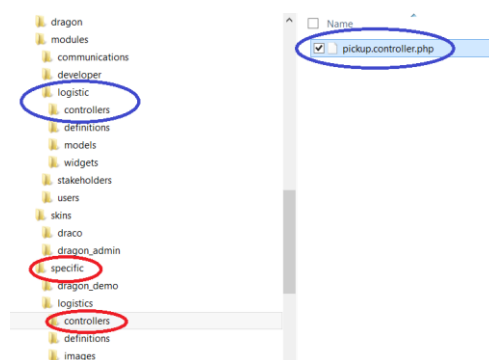


Figure 16 - DragonPHP Controller Found

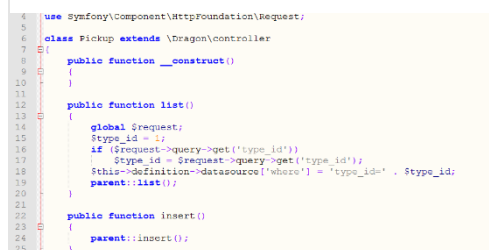


Figure 17 - DragonPHP Controller content

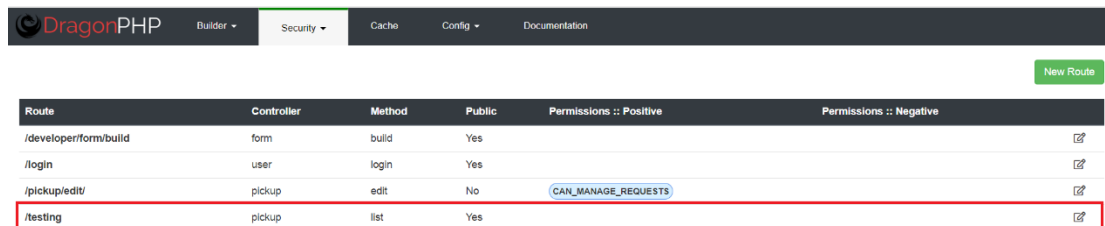
When the controller / method is found within a specific project, DragonPHP by default will use the concept of Inheritance in order to show the page.

This means that we can have a generic module created in the /modules/ path and only have specific code to our project in our project directory

5.3.1 Custom Routing

As seen in the previous section by default the framework will break the URL in two or more sections in order to search for a controller and a method to present the page.

But by using DragonPHP Custom Routing configurations we can configure custom paths and tell the framework where it should go. To access the Routing configurations the developer may go to the developer backoffice and access the Security -> Routing option found on the top bar.



Route	Controller	Method	Public	Permissions :: Positive	Permissions :: Negative
/developer/form/build	form	build	Yes		
/login	user	login	Yes		
/pickup/edit/	pickup	edit	No	CAN_MANAGE_REQUESTS	
/testing	pickup	list	Yes		

Figure 18 - DragonPHP Routing list

In the above screenshot we configured the url <http://localhost/testing> to access the controller “pickup” and the method “list”. This will mean that accessing <http://localhost/pickups/list> or <http://localhost/testing> will result in the same output, but, with a custom URL.

To add a new routing the developer can click on “New Route” and a form as the one below will appear with simple attributes:

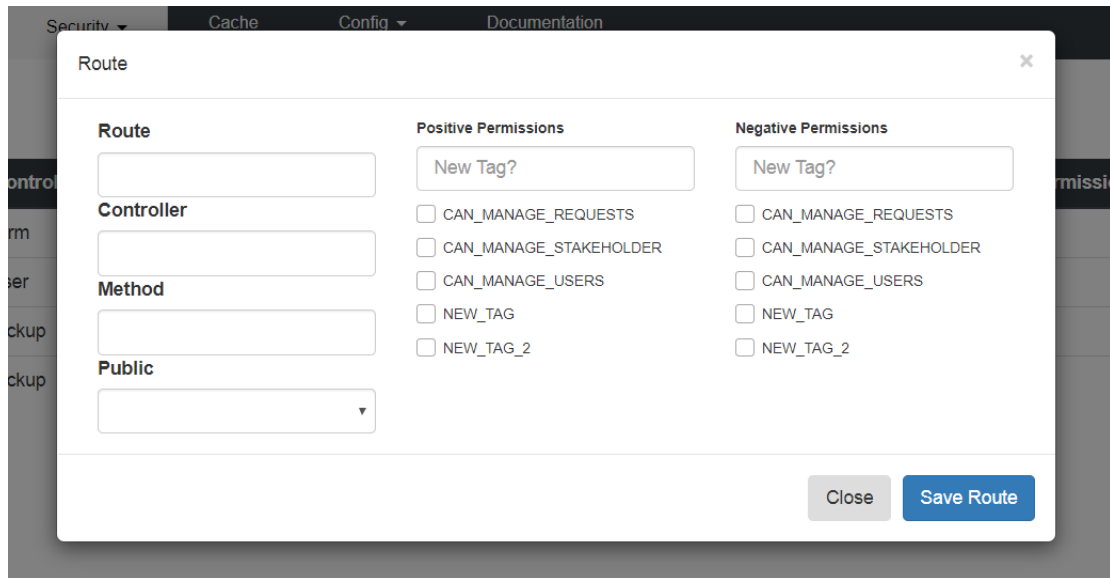


Figure 19 - DragonPHP Creating new route

5.3 SECURITY AND PERMISSIONS MANAGER

Security is always one of the most important parts of any framework. In DragonPHP every request will be parsed and several security parses will be done.

The most important security parses will be:

- 1- Does the logged user have access to view the url he's requesting?
- 2- By using several algorithms all GET and POST arguments will be parsed in search of SQL injections. If found the specified argument will be striped.
- 3- Except if specified by the developer every form created in DragonPHP will verify for XSS in their inputs and textarea elements. If any has any type of XSS the framework will strip those away before inserting them into the database.

5.3.1 Permissions Manager

As shown in the above figure (need to input the figure number!) when creating a new routing the developer may choose or add a new "permission tag" to that routing. That tag can be configured as a Positive or Negative permission.

This means that if the URL is not Public, if a user tries to access it the framework will:

- 1- Verify if the user has any tags associated to it which matches the Negative tags of the URL. If the response is true, than it means that the user does not have access to the URL.
- 2- If no negative tags were found, the framework will search for positive tags. If the user does not have any of the positive tags needed than the user does not have access to the URL.

This approach allows the developer to create a modular and flexible permission manager in each of his applications.

Below is a database structure schematic of how permissions works:

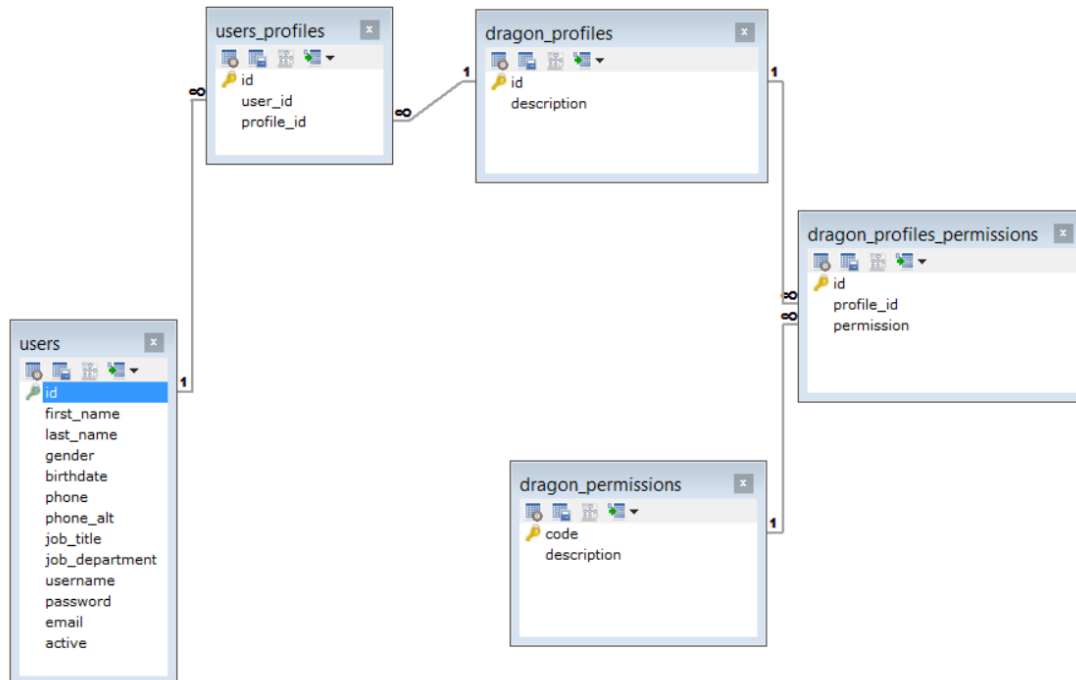


Figure 20 - DragonPHP DB Schema for permissions

This means that there are several Profiles, and Several permissions tags. Each profile is connected to one or more permission tags.

Each user can have one or more profiles.

In our example project we have the following configuration:

#	Profile	Permissions
1	Administrator	
2	Regular User	CAN_MANAGE_REQUESTS CAN_MANAGE_STAKEHOLDER

Figure 21 - DragonPHP list of profiles

Any profile without any permission tag is considered as a profile that can view and do everything in the application. This means that the profile Administrator can access every URL.

The profile Regular User can only view and access URL which have the positive permission “CAN_MANAGE_REQUESTS” and “CAN_MANAGE_STAKEHOLDERS”.

5.4 CRUD GENERATOR AND DEFINITION FILES

If we access the URL of our demo project <http://localhost/pickups/insert> we will have this next result:

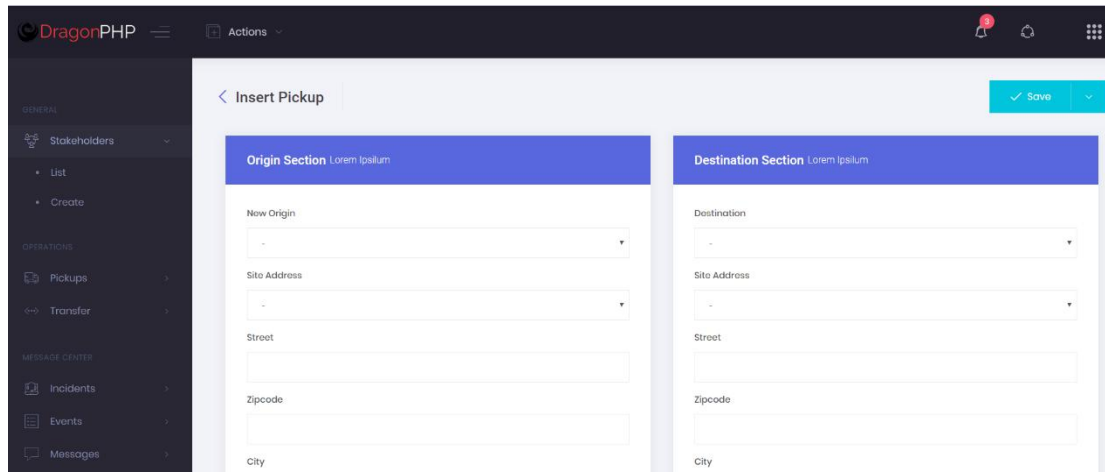


Figure 22 - DragonPHP Insert form example

This is the result of a simple Create action configured in the Framework. The way the framework worked was that it found a controller: pickup.controller.php, in that controller a method called insert, and it found a definition file called insert.pickups.yaml

Below is the code of each file:

1- Pickup.controller.php

```
1 <?php
2 namespace DragonController;
3
4 use Symfony\Component\HttpFoundation\Request;
5
6 class Pickup extends \Dragon\controller
7 {
8     public function __construct()
9     {
10
11
12     public function list()
13     {
14         parent::list();
15     }
16
17     public function insert()
18     {
19         parent::insert();
20     }
21
22     public function edit()
23     {
24         parent::edit();
25     }
26
27     public function delete()
28     {
29     }
30 }
```

Figure 23 - DragonPHP controller example

2- Insert.pickup.yaml

```
1 form_definition:
2   includes:
3     file: pickup.form.yaml
4   form:
5     action: insert
6     title: Insert Pickup
7   fields:
8     origin_id:
9       label: 'New Origin'
```

Figure 24 - DragonPHP definition example 1

As seen above both files seems to be very much stripped of any code. Since the method insert has no special code the framework parsed the definition insert.pickup.yaml file to know how to behave. In that file the framework understood that the page to be rendered was a “insert” (action: insert), has a specific title “Insert Pickup” and includes everything within the definition pickup.form.yaml.

If we view the contents of pickup.form.yaml we will now understand how the framework knew what to render.

```
1 form_definition:
2   entity:
3     main: Pickup
4   datasource:
5     table: pickups
6     pkey: id
7     order_by: id desc
8   form:
9     navigation_menu:
10      new_pickup:
11        title: New Pickup
12        url: /pickup/insert
13        css_class: flaticon-file
14        target: _blank
15      list_pickup:
16        title: List All Pickups
17        url: /pickup/list
18        css_class: flaticon-diagram
19        target: _blank
20   view:
21     layout:
22       tabl:
23         label: Collection
24         columns: 2
25         sections:
26           origin_section:
27             label: Origin Section
28           destination_section:
29             label: Destination Section
30           waste_details:
31             label: Waste Details
32             colspan: 2
33           carrier_section:
34             label: Carrier Section
35           collection_info:
36             label: Collection Info
37           waste_documents:
38             label: Documents
39   fields:
40     id:
41       label: Id
42       data_type: integer
43       insertable: false
44       updatable: false
45       section: section1
46     origin_id:
47       label: Origin
48       data_type: int
49       edit_object: select
50       lookup_table: stakeholders
51       lookup_text: short_name
52       lookup_value: id
53       section: origin_section
54     dependencies:
55       address:
56         field: origin_address_id
57         sql: select id as value, description as text from stakeholders_
58     contact_name:
59         field: origin_contact_name
60         sql: select users.first_name as text from users, stakeholders_u
61     contact phone:
62         field: origin_contact_phone
63         sql: select users.phone as text from users, stakeholders_users
64     contact_email:
65         field: origin_contact_email
66         sql: select users.email as text from users, stakeholders_users
67     carrier_id:
68       label: Carrier
69       data_type: int
70       edit_object: select
71       lookup_table: stakeholders
72       lookup_text: short_name
73       lookup_value: id
74       section: carrier_section
75     carrier_subcontractor_id:
76       label: Carrier Sub-Contractor
```

Figure 25 - DragonPHP definition example 2

One the left figure we told the framework to use the table pickups and that the primary key of the table is the id. We also specified that the page will have a submenu with two options

(New pickup and List All Pickups), we specified how the layout for the page would be generated, a single tab, with two columns and several sections, each with its label.

On the right side we have the definition of the fields to be generated. The field “ID”, since it is the primary key, its not meant to be inserted or updated (insertable / updatable both to false), the field origin_id will have the label “Origin” the data type of that field will be a integer and it will do a lookup to the table Stakeholders, making it a foreign key to that table. The edit_object to be generated will be a “HTML Select Box” and the field should be present in the section “origin_section”.

This file, pickup.form.yaml, will have in this case, the definition of all fields of that specific table. Definitions will also work with inheritance, which means that insert.pickups.yaml will have all code of itself and all code of pickups.form.yaml, but giving preference to the first.

This allows us to have a generic definition per table with all fields and behaviors and if we need to do small changes to a specific page we can include those specific conditions on the latest definition file without doing any change to the parent one.

5.4.1 How to create or update our crud

There is two ways to create or update the crud definition files. Or by manually accessing the file and changing its content, or by using the Form Builder tool available in the developer backoffice.

This tool is meant to be a WYSIWYG with simple point and click in order to create new fields or change the fields behavior:

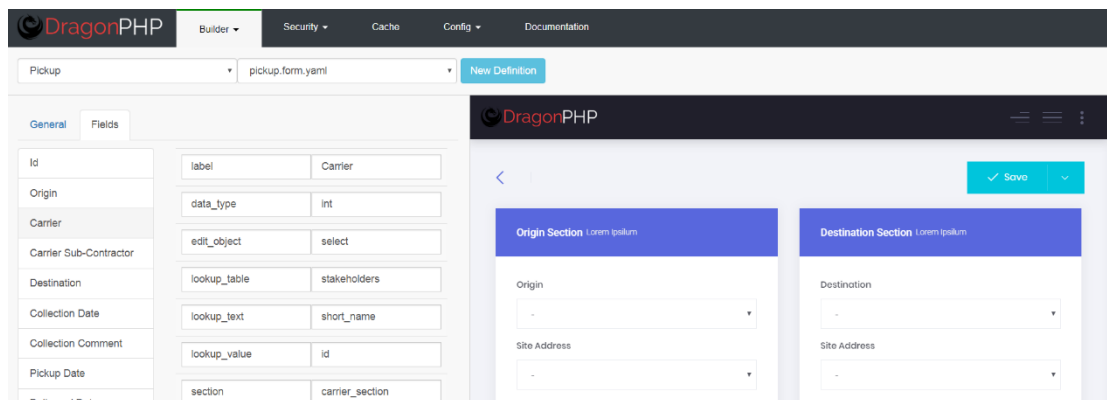


Figure 26 - DragonPHP Form Builder

In the left side we have the configuration of the definition files and on the right side we have a preview of how the page will appear to the end user.

5.4.2 Why creating CRUD with definition files

By using definition files to generate our CRUD we simplify the development of any application. Any non-developer or junior developer can create its own forms with little to no knowledge of PHP, HTML, CSS and Javascript.

This allows for a very rapid development of pages such as Lists, Insert Forms, Edit Forms, Delete Forms and Search Forms.

The fact that DragonPHP is using inheritance will also allow for modules to be created in a generic way and each project can have its tiny tweaks without any knowledge of PHP or need of hacks.

The last major point is the backwards compatibility. By using simple tags in the definition files, i.e. "label" or "data_type" allows the framework to constantly grow and improve how it is parsed and displayed while maintaining the fact that a label will always be a label, regardless of how it is generated. This means that a project can be developed and still be access after years of being developed and it will always behave the same.

5.4.3 Definition files, more than just simple CRUD

Aside from the CRUD seen above, this definition files can, and will be expanded to be able to create more sophisticated functionalities.

One of this functionalities is the workflow. By accessing the developers backoffice and go to Builder -> Workflow we have the next page:

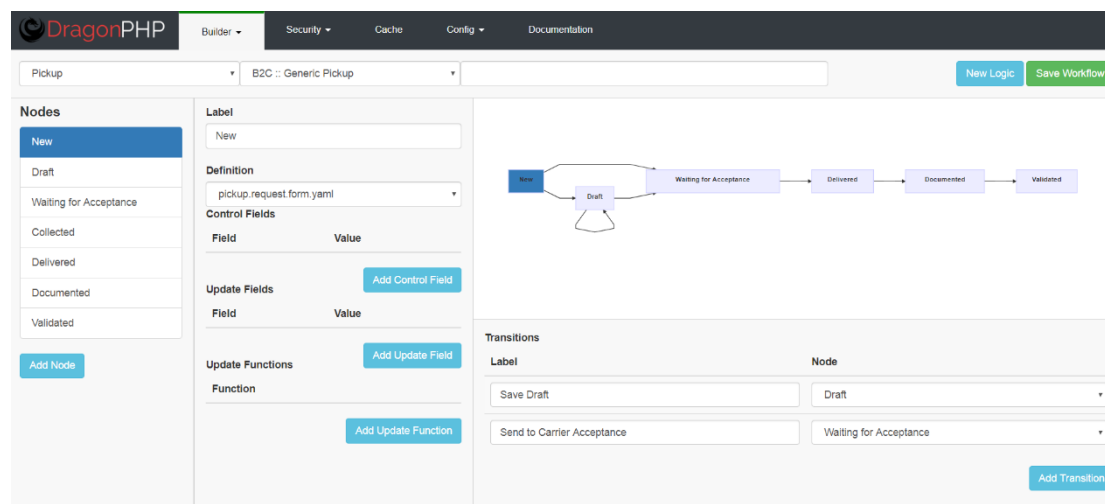


Figure 27 - DragonPHP Workflow Builder

This allows us to create workflow between pages in the application again with simple point and click. Just like the CRUD pages we can also access the specific definition file generated and change it manually:

```

1 form_definition:
2   datasource:
3     table: pickups
4     pkey: id
5   form:
6     action: workflow
7     title: 'Request Workflow'
8   workflow:
9     logic:
10      b2c_generic_pickup:
11        label: 'B2C :: Generic Pickup'
12        filter: ''
13        transitions:
14          new:
15            next0:
16              label: 'Save Draft'
17              node: draft
18            next1:
19              label: 'Send to Carrier Acceptance'
20              node: waiting_for_acceptance
21          draft:
22            next0:
23              label: 'Save Draft'
24              node: draft
25            next1:
26              label: 'Send to Carrier Acceptance'
27              node: waiting_for_acceptance
28          waiting_for_acceptance:
29            next0:
30              label: 'Accept Request'
31              node: delivered
32          delivered:
33            next0:
34              label: 'Mark as Delivered'
35              node: documented
36          documented:
37            next0:
38              label: 'Mark as Documented'
39              node: validated
40
41 b2c_generic_transfer:
42   label: 'B2C :: Generic Transfer'
43   filter: ''
44   control_fields:
45     type_id: 2
46   transitions:
47     new:
48       next0:
49         label: 'Go to Draft'
50         node: draft
51       next1:
52         label: 'Go to Documented'
53         node: documented
54     draft:
55       next0:
56         label: 'Save as Draft'
57         node: draft
58       next1:
59         label: 'Go to Documented'
60         node: documented
61     documented:
62       next0:
63         label: 'Go to Validation'
64         node: validated
65   nodes:
66     new:
67       label: New
68       definition: pickup.request.form.yaml
69       control_fields: { }
70       update_fields: { }
71       update_functions: { }
72     draft:
73       label: Draft
74       definition: pickup.draft.form.yaml
75       control_fields:
76         status_id: '1'
77       update_fields: { }
78       update_functions: { }
79     waiting_for_acceptance:

```

5.5 TRYING TO AUTOMATE DOCUMENTATION

The above functionalities aims to allow development of the most common functionalities found in any application easy and fast. But development is not only code, it is also documentation. DragonPHP tries to help developers by trying to automate the documentation. By parsing every file in the framework when it is being runned, and with a low burden to process and rendering (~0.01 secs) the framework is able to build documents as:

1- Entities Description and Relations:

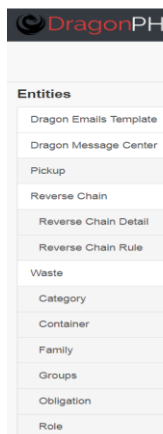


Figure 28 - DragonPHP definition example 3

Figure 29 - DragonPHP Entities builder

2- Tags created and used:

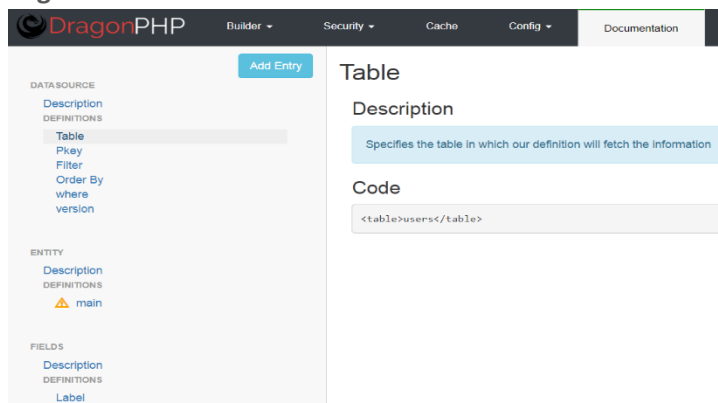


Figure 30 - DragonPHP auto documentation

In both examples the framework will try to autocomplete the documentation but still allowing the developer to add more information or change it accordingly.

Next steps for the documentation are to create a database schema and php class/method comments without the need of any interact from the developer.

5.6 HOW TO WRITE PHP/HTML CODE IN DRAGONPHP

Code in an application can be divided in two, interface (HTML/CSS/JS) and engine (PHP/MySQL).

Below is the explanation on how to change and write your own code for those purposes.

5.6.1 Interface code built with twig

Each project has its default skin, and each page calls a specific template file. By navigating in our source code to the folder /skins/ we encounter by default two available skins:

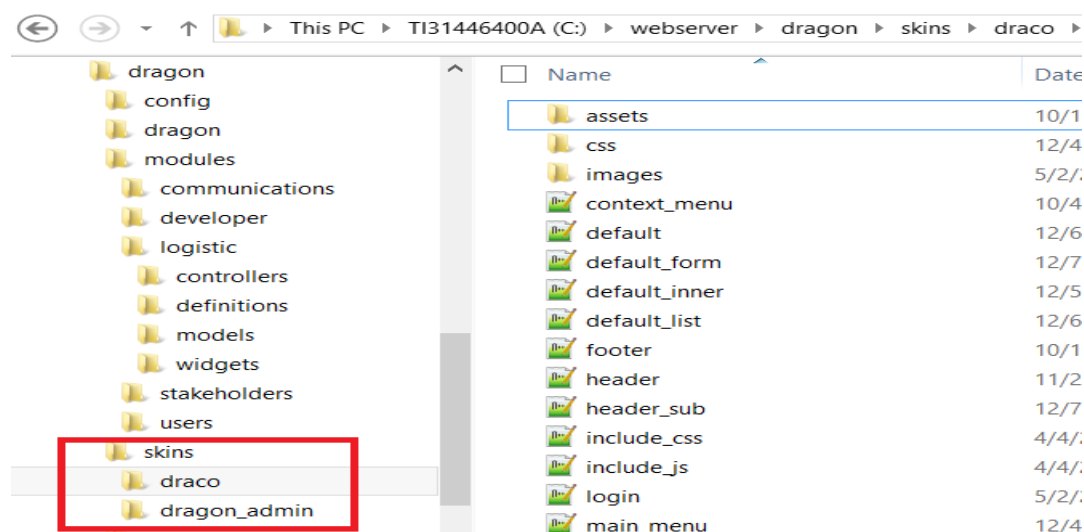


Figure 31 - DragonPHP skins structure

Draco will be the skin used for our projects backoffice. Developers may change it or create new skins. DragonPHP uses Twig templating engine in order to render the templates.

```

21 <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
22
23 {% include 'include_css.twig' %}
24 <link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/v/ju-1.12.1/jszip-2.5.0/dt-1.10.1
25 <!--end::Page Vendors Styles -->
26 <link rel="shortcut icon" href="/skins/draco/images/favicon.ico" />
27 <link href="/skins/draco/css/draco.style.css" rel="stylesheet" type="text/css" />
28 </head>
29
30 <!-- end::Head -->
31
32 <!-- begin::Body -->
33 <body class="m-page--fluid">
34 <!-- begin:: Page -->
35 <div class="m-grid m-grid--hor m-grid--root m-page">
36     {% include 'header.twig' %}
37     <!-- begin::Body -->
38     <div class="m-grid_item m-grid_item--fluid m-grid m-grid--ver-desktop m-grid--desktop m-body">
39         {% include 'main_menu.twig' %}
40         <div class="m-grid_item m-grid_item--fluid m-wrapper">
41             {% block body %}{% endblock %}
42         </div>
43     </div>
44     <!-- end:: Body -->
45     {% include 'footer.twig' %}
46 </div>
47
48 <!-- end:: Page -->
49
50 {% include 'sidebar.twig' %}
51
52 <!-- begin::Scroll Top -->
53 <div id="m_scroll_top" class="m-scroll-top">
54     <i class="la la-arrow-up"></i>
55 </div>
56 <!-- end::Scroll Top -->
57
58 {% include 'sidebar_nav.twig' %}
59 {% include 'include_js.twig' %}
60 </body>
61
62 <!-- end::Body -->
63 </html>

```

Figure 32 - DragonPHP skin template example

5.6.2 Engine code PHP/MySQL

As seen in section 5.4 when accessing a specific URL, DragonPHP will go over a specific method within a controller. Developers may add any necessary PHP / MySQL code on those Methods, below goes some examples:

1- Creating our own PHP code:

```

1 <?php
2 namespace DragonController;
3
4 use Symfony\Component\HttpFoundation\Request;
5
6 class Pickup extends \Dragon\controller
7 {
8     public function __construct()
9     {
10     }
11
12     public function list()
13     {
14         global $request;
15         $type_id = 1;
16         if ($request->query->get('type_id'))
17             $type_id = $request->query->get('type_id');
18         $this->definition->datasource['where'] = 'type_id=' . $type_id;
19         parent::list();
20     }
21
22     public function insert()
23     {
24         parent::insert();
25     }
26
27     public function update()
28     {
29         createVersion($this->pkey);
30         parent::update();
31         sendInformationtoWebServices($this->pkey);
32     }
33
34 }
35

```

Figure 33 - DragonPHP custom controller php code example 1

In this example we run a custom made PHP function, createVersion, before the update action is performed, and we call the function sendInformationtoWebServices after the update is done. And we leave the update of the record “parent::update()” to the framework to do all its job.

2- Passing values from the database to the visuals

```
1 <?php
2 namespace DragonController;
3
4 use Symfony\Component\HttpFoundation\Request;
5
6 class Pickup extends \Dragon\controller
7 {
8     public function __construct()
9     {
10    }
11
12    public function list()
13    {
14        global $request;
15        $type_id = 1;
16        if ($request->query->get('type_id'))
17            $type_id = $request->query->get('type_id');
18        $this->definition->datasource['where'] = 'type_id=' . $type_id;
19        parent::list();
20    }
21
22    public function insert()
23    {
24        global $skin, $db;
25        $sql = "select description from pickups where id=?";
26        $description = $db->Execute($sql, array($this->pkey));
27
28        $skin->description = $description;
29        parent::insert();
30    }
31 }
```

Figure 34 - DragonPHP custom controller php code example 2

In this example we use the global variable \$db, which is an object of the database connector instantiated by DragonPHP to grab the description of a pickup, and we send it back to the skin by calling the global \$skin and adding a new attribute to it.

Only after running our code, DragonPHP will generate the Insert form.

Chapter 6 - Initial Developers Opinion

In order to understand if DragonPHP prototype was able to answer the issues found with other frameworks, a small survey was sent to 10 developers, whereas 5 developers had less than 2 years of experience (Junior Developers) and the remaining 5 more than 2 years.

The preliminary results of the survey was satisfactory and it encourages the author to continue with the Framework and perfect the job.

Below is the result with calculated averages from all 10 responses:

Dragon PHP Survey Results (10 Surveys answered)	
Question	Average (1/5)
Was DragonPHP easy to install?	5
Were you able to create pages in a fast way?	5
Do you feel you still have the control of your code?	4
Do you use the form builder to generate your forms?	3
Do you feel auto-documentation is an essential part?	4
Was the security module easy to configure?	4
Is the available documentation good?	2
Would you suggest DragonPHP to another developer?	5

Chapter 7 – Conclusion

There are too many PHP frameworks in the market, and they all appeal to a specific group of developers or to a more junior developer or, to a more senior developer, but, almost none of them tries to appeal to both groups.

By studying what each top framework has and by interviewing developers the author was able to come with a new development pattern HDMVC which allows for faster and more agile development of applications in comparison with the MVC model.

DragonPHP was also able to remove the complexity of installing a framework by just unpacking its content to a WWW folder and without the need of any third-party software, making the entry difficulty for junior developers much lower.

By exchanging command line, hard coded configurations and the need of coding PHP pages with simple point and click pages DragonPHP enable developers with less experience to create sophisticated CRUD applications with minimum code.

Nevertheless, even thou DragonPHP makes it easier to develop applications and tries to overcome the coding problem, more experience developers may change and add its code to each page in a very simple and organized way.

The initial conclusions taken from the survey which was sent to developers is very positive, and clearly demonstrates that the overall opinion is that its conception and development appeals to developers.

More work is still needed to take DragonPHP to the open world, but after the initial impressions and feedback, DragonPHP will have a bright future.

Chapter 8 - Bibliography

Ahmad, Z. A. (n.d.). PHP FRAMEWORK DESIGN WITH HIERARCHICAL MODEL-VIEWCONTROLLER ARCHITECTURE.

Bartosz Porebski, K. P. (2011). *Building PHP Applications with Symfony, CakePHP and Zend Framework*.

Dragos-Paul Pop*, A. A. (2013). Designing an MVC Model for Rapid Web Application Development. *ScienceDirect*.

Glenn E. Krasner, S. T. (n.d.). *A Cookbook for Using View-Controller User the ModelInterface Paradigm in Smalltalk-80*.

Google Trend. (2019). Retrieved from PHP Frameworks: <https://trends.google.com/trends/explore?q=laravel,Symfony,%2Fm%2F02qgdj,CakePHP,Zend>

Guide to the Software Engineering Body of Knowledge Version 3.0. (n.d.).

<https://cakephp.org/>. (n.d.). Retrieved from CakePHP.

<https://code.tutsplus.com/tutorials/should-you-use-a-php-framework-five-pros-and-cons--cms-28905>. (n.d.). Retrieved from Should You Use a PHP Framework? .

<https://codeigniter.com/>. (n.d.). Retrieved from CodeIgniter.

https://en.wikipedia.org/wiki/Cascading_Style_Sheets. (n.d.). Retrieved from Cascading_Style_Sheets.

https://en.wikipedia.org/wiki/Category:PHP_frameworks. (2018). Retrieved from PHP Frameworks.

[https://en.wikipedia.org/wiki/Design_science_\(methodology\)](https://en.wikipedia.org/wiki/Design_science_(methodology)). (n.d.). Retrieved from Design_science_(methodology).

<https://en.wikipedia.org/wiki/HTML>. (n.d.). Retrieved from HTML.

<https://en.wikipedia.org/wiki/JavaScript>. (n.d.). Retrieved from JavaScript.

https://en.wikipedia.org/wiki/Multitier_architecture. (n.d.). Retrieved from Multitier_architecture.

<https://en.wikipedia.org/wiki/Mysql>. (n.d.). Retrieved from Mysql.

<https://en.wikipedia.org/wiki/PHP>. (n.d.). Retrieved from PHP.

<https://en.wikipedia.org/wiki/yaml>. (n.d.). Retrieved from Yaml.

<https://justpaste.it/gd76>. (n.d.). Retrieved from Why you should NOT use a web framework.

<https://laravel.com/>. (n.d.). Retrieved from Laravel.

<https://softwareengineering.stackexchange.com/questions/49488/when-not-to-use-a-framework>. (n.d.). Retrieved from When NOT to use a framework [closed].

<https://stackoverflow.com/questions/5925356/reasons-to-not-use-a-php-framework>. (n.d.). Retrieved from Reasons to NOT use a PHP Framework? [closed].

<https://symfony.com/>. (n.d.). Retrieved from Symfony.

<https://www.codementor.io/juzerali/5-reasons-bad-developers-will-give-for-not-using-an-application-framework-ejxogcgb7>. (n.d.). Retrieved from 5 Reasons Bad Developers Will Give for Not Using an Application Framework.

<https://www.johnmorrisonline.com/dont-use-php-frameworks/>. (n.d.). Retrieved from Why I don't use PHP frameworks.

<https://www.yiiframework.com/>. (n.d.). Retrieved from YiiFramework.

McArthur, K. (2008). *Pro PHP Patterns, Frameworks, Testing and More*.

MVP. (2019). Retrieved from <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

MVVM. (2019). Retrieved from <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>

Natalya Prokofyeva, V. B. (2017). Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems. *Procedia Computer Science* 104.

NIIT. (2001). *Special Edition Using C#*. Que.

PHP Frameworks. (2018). Retrieved from <http://phpframeworks.com>

Riehle, D. (2000). *Framework Design: A Role Modeling Approach*.

Vaishnavi, V. a. (2008). *Design Science Research Methods and Patterns*. 1st Edn.

w3Techs. (2019). Retrieved from https://w3techs.com/technologies/overview/programming_language/all

Web application development with Laravel PHP. (n.d.). *Jamal Armel*.

Wikipedia. (2019). Retrieved from Markup Language: https://en.wikipedia.org/wiki/Markup_language