**King Abdul Aziz University**

**Faculty of Computing and Information Technology**

**Department of Computer Science**

**Senior Project CPCS-499**

Developing an Online Developers Community& auto UML Generator (Report-II)

02.10.2018

## Prepared by

Hazar Gul Nazari (1414322)

Supervisors

Prof. Isa Fathi & Prof. Kamal Jambi

**Table of Contents**

[Summary of previous work 2](#_Toc530098160)

[Aim of the project 2](#_Toc530098161)

[Problem definition 2](#_Toc530098162)

[Project scope 2](#_Toc530098163)

[Expected outcome 3](#_Toc530098164)

[Target Users 3](#_Toc530098165)

[Suggested solution 3](#_Toc530098166)

[Functional Requirements 3](#_Toc530098167)

[Non-Functional Requirements 3](#_Toc530098168)

[Data Requirements 4](#_Toc530098169)

[Implementation Languages 4](#_Toc530098170)

[Software 4](#_Toc530098171)

[Methods and Approaches 4](#_Toc530098172)

[Planning 4](#_Toc530098173)

[Analysis 4](#_Toc530098174)

[Design 5](#_Toc530098175)

[Implementation 5](#_Toc530098176)

[Maintenance 5](#_Toc530098177)

[Activities and Actions 5](#_Toc530098178)

[Use Case Diagram 5](#_Toc530098179)

[Flow Chart 6](#_Toc530098180)

[Class Diagram 7](#_Toc530098181)

[Tasks Completed 7](#_Toc530098182)

[Interface 8](#_Toc530098183)

[References 13](#_Toc530098184)

[List of references:- 13](#_Toc530098185)

[Book: 13](#_Toc530098186)

[Code ( Copy and Paste all of your project code…..) 14](#_Toc530098187)

# Summary of previous work

## Aim of the project

The aim of this project is to create an application which is an online community for developers. This online community will allow the users to share, ask their ideas or UML diagram by generating UML diagram from the source code automatically. In the future, this project will allow to all types of OOP (Objected-Oriented Programming) languages to generate their UML source code.

## Problem definition

This project is an online software application which focuses on creating an auto UML generator to convert the PHP 5.0 source code into UML diagram. In the future, it can be expanded easily, due to providing such a platform for other languages as well, such as (Java, C++, and other languages that support OOP concept). The problem is, in developing a PHP parser to extract all the Class and Interface information from the source code and finding the solution on how to illustrate the extracted information into diagram by using HTML5 Canvas that could be easier to understand for the user. Also, another part of the problem is to create such a platform to allow users to create a specific group where the developers can share and ask the ideas on a specific topic such as a group for PHP where PHP developers can join and chat with each other about specific PHP topic.

## Project scope

The scope of this project is to:

* Create an auto UML generator from PHP 5.0 source code
* Creating a special communication tool for developers
* Make a platform for future extension of auto UML diagram for other OOP (Object Oriented Programming) languages

## Expected outcome

A full responsive web application that can generate UML diagram from PHP 5 source code, the users can download, share and interact with the generated UML diagram. Besides, the application provides a special communication tool for developers where they could find a specific group and join it for discussion.

## Target Users

The target users of this application are the software developers, but in general, anyone can use it for his or her own purpose of generating UML diagram from PHP source code and to communicate with other developers.

## Suggested solution

* Creating a PHP syntax parser for extracting class and interface details from the source code
* Using HTML5 Canvas to draw the extracted information from the PHP source code into visually UML diagram
* Creating the application interface for the chat application

## Functional Requirements

* Generate UML diagram
* Download generated UML diagram
* Share generated UML diagram
* Create new group
* Ranking the group
* Putting group in favorite list
* Remove group from favorite list
* Register new account / Sign in account
* Search community
* Send & receive message on chatroom

## Non-Functional Requirements

* Response time
* Throughput
* Reliability
* Maintainability

## Data Requirements

The user should upload the files which contain the PHP source code, which through the uploaded source code, the UML diagram will be generated.

## Implementation Languages

* PHP
* MySQL
* JavaScript
* jQuery libraries
* HTML
* CSS

## Software

* NetBeans 8.2
* Windows 10
* XAMP Server (PHP and MySQL compiler)
* Chrome Browser

# Methods and Approaches

## Planning

I planned for the project by using the **work break down structure** (WBS), scheduled the project by Gantt chart & time estimates to manage & organize my work in order to complete the project.

## Analysis

At First, I tackled the analysis and parsing part of PHP syntax analyzer and figure out how to design and implement it to extract the Class and Interface information from the PHP files. The compiler course which I took last year, really helped me in this part of the project.

Second, I figured out about the illustration part of the UML diagram on the client side (Web Browser), which is the most important and difficult part of this project. The generated UML diagram will be illustrated in the browser so, therefore, I planned to make use of HTML 5 Canvas to convert the extracted information from the source code into UML diagram in the client side.

Third, it’s the application’s user interface part (GUI) and also figure about the chat application part which is about creating the database for the entire application.

## Design

I designed the flow chart, use case diagram and class diagram for this project which you can see on the above section of this page.

## Implementation

This project is an application which is an implementation based project, therefore I started coding the project and there are 3 parts in coding the coding section:

1. Implementation of the PHP syntax parser.
2. Implementation of the UML illustration part by HTML 5 Canvas.
3. Implementation of Chat Application.
4. Implementation of home page
5. Implementation of profile & profit pages
6. Implementation of creating new community or group page

## Maintenance

After completing the code, I will test the program in order to remove the errors.

# Activities and Actions

## Use Case Diagram

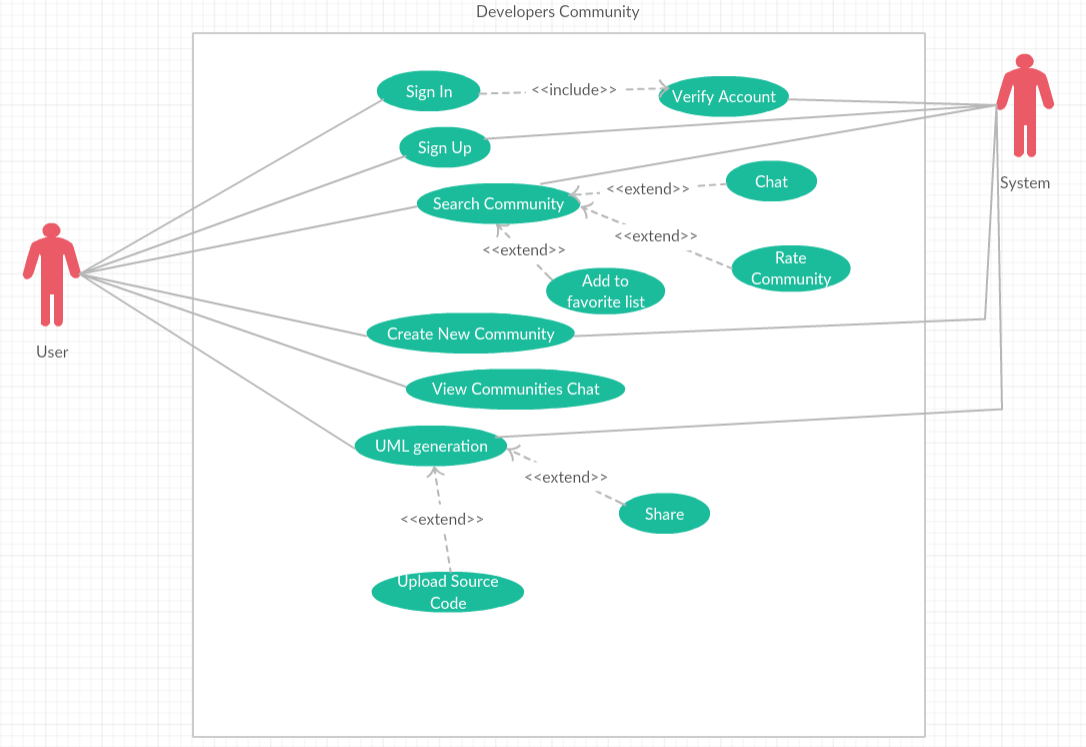


Figure this use case shows how the application works

## Flow Chart

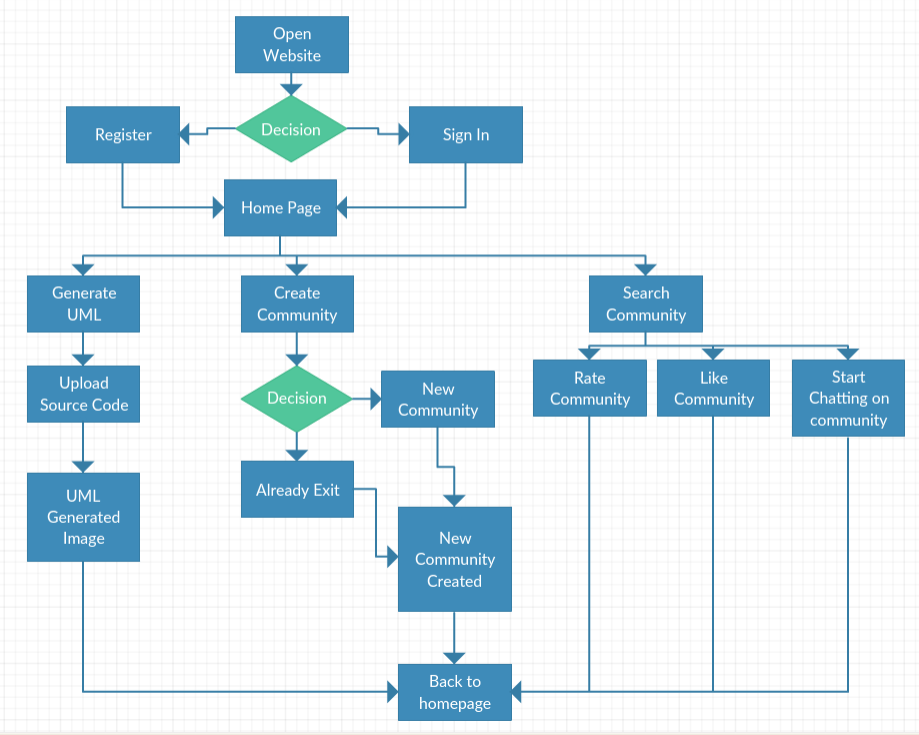


Figure this flow chart explains how the entire system will work

## Class Diagram

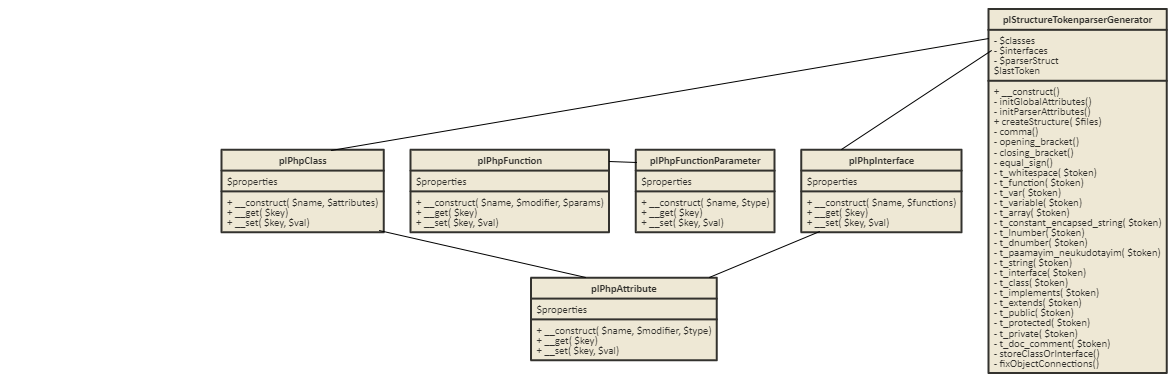


Figure 3 shows the PHP syntax parser’s class diagram

## Tasks Completed

* PHP syntax parser.
* Printing UML diagram from source code
* Generating UML diagram from source code
* Chat Application
* Searching Community
* Creating, editing user account

## Interface

Here are some of the interface of the project.

1. **The Main Page**: In the main page, there are many options for the users to navigate the entire application. Such as, searching a specific group, entering to a specific group’s chat room, creating a new group, putting a specific group into favorite list, signing in/up, and the most important one, generating UML diagram.



Figure shows the main page of the application

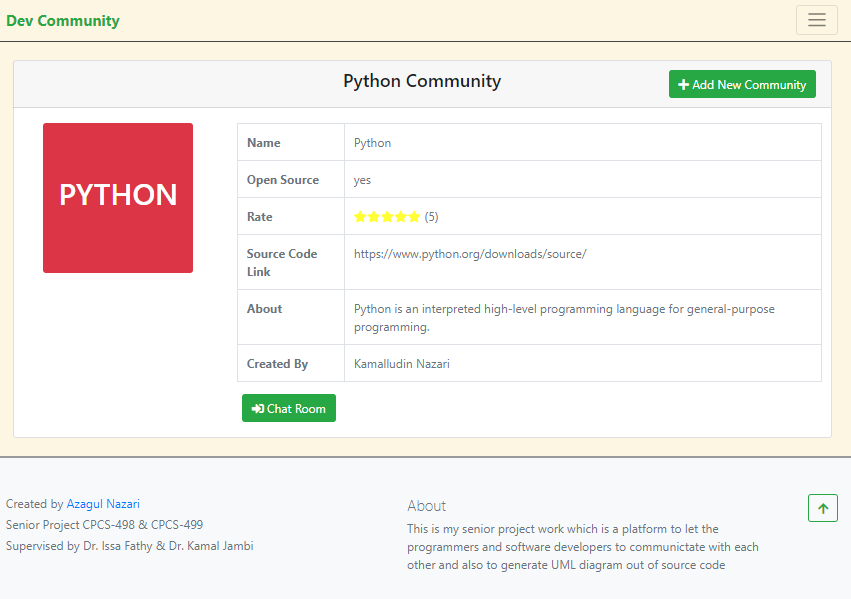
1. **Community Information Page**: this page is to get an overall information about a specific group (language/library) and it also allows the user to rate it or to join the chatroom.

Figure shows the community info page

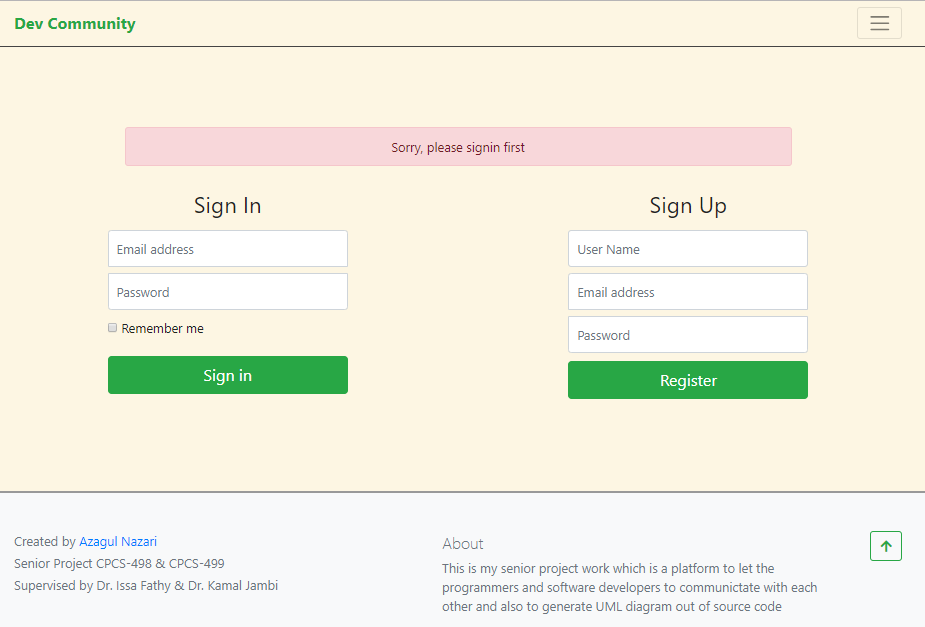
1. **Sign in/up page**: this page is to let the user to log in or to register a new account in the system. 

Figure shows the sign in and register page

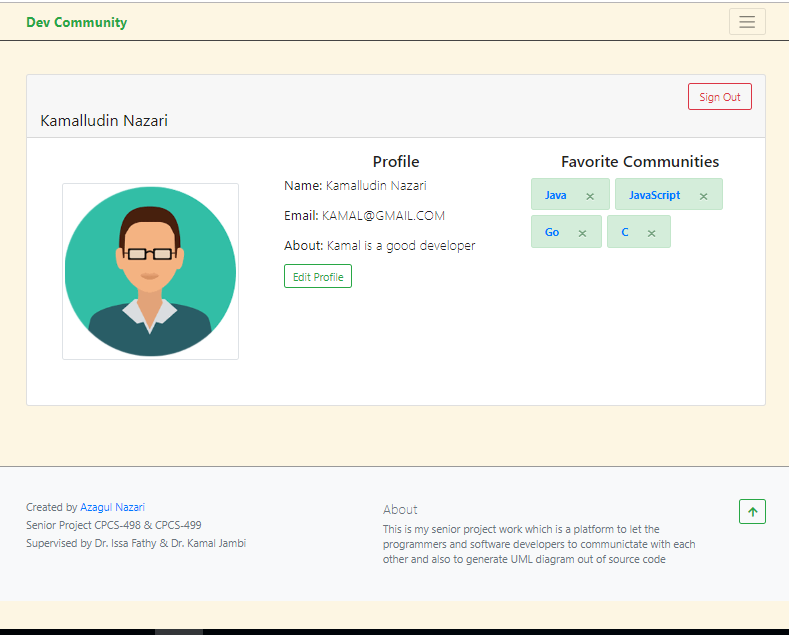
1. **Profile page**: the profile page is to visit the profile page of the user. The user can see his/her favorite group list and also he/she can edit his profile information such as email, password, details and profile pic.

Figure shows the user profile information

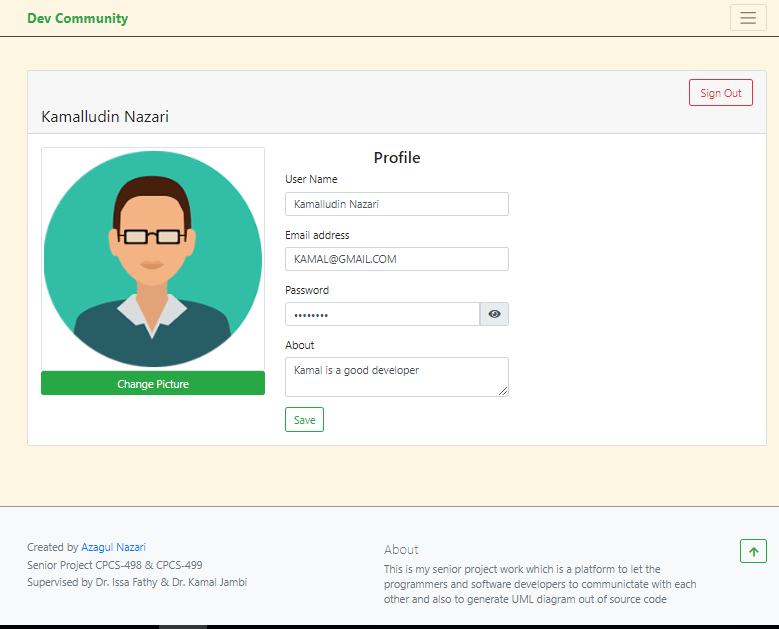
1. **Profile Edit page**: If the user wants to bring some changes to his/her profile, he can go to this page and edit his profile info. Note that only the authorized users can edit his profile. 

Figure shows the profile edit page

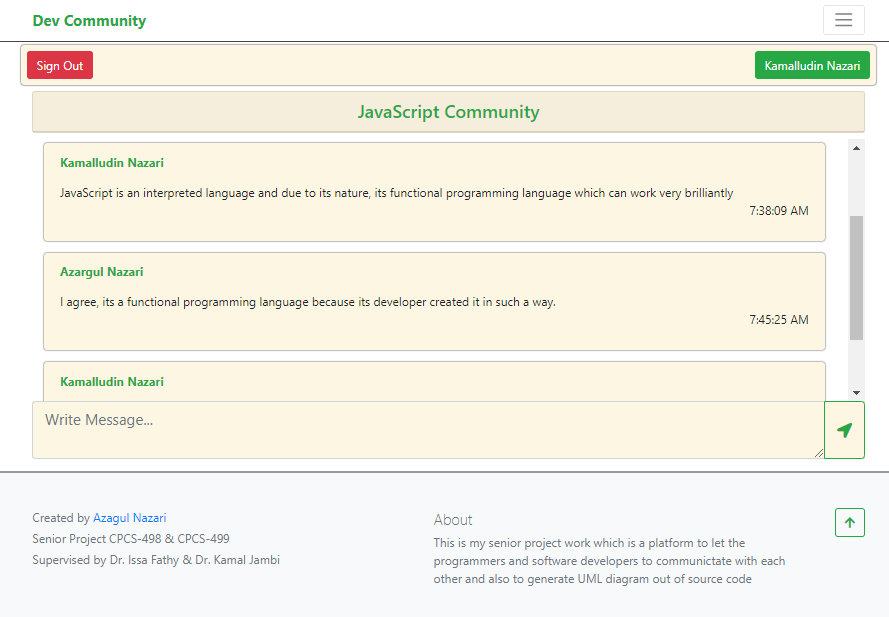
1. **Group chatroom page**: this page is to let the developers to chat in the same group which they can send messages on the group.

Figure shows the chat room for the JavaScript Community

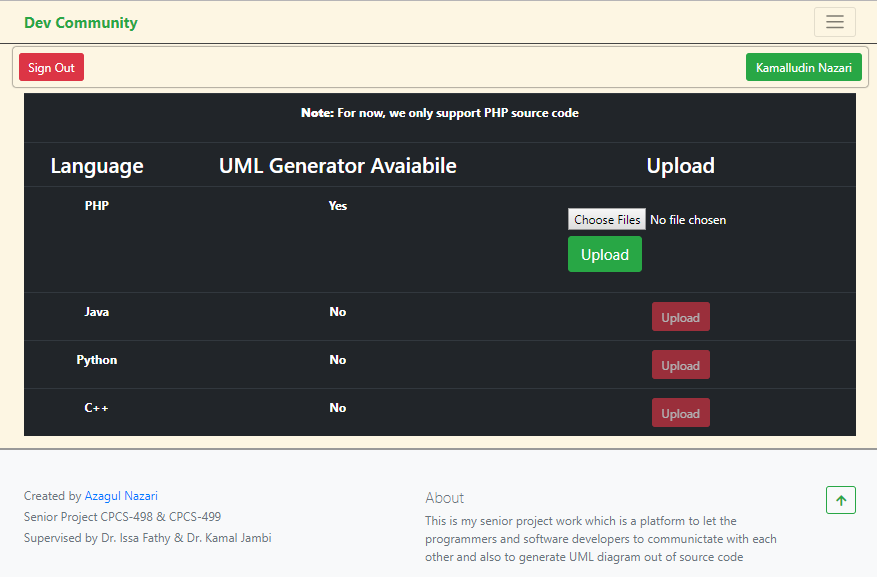
1. **Upload file page:** The upload page is to let the users to upload PHP files for generating their UML diagram.

Figure Shows the PHP file upload page

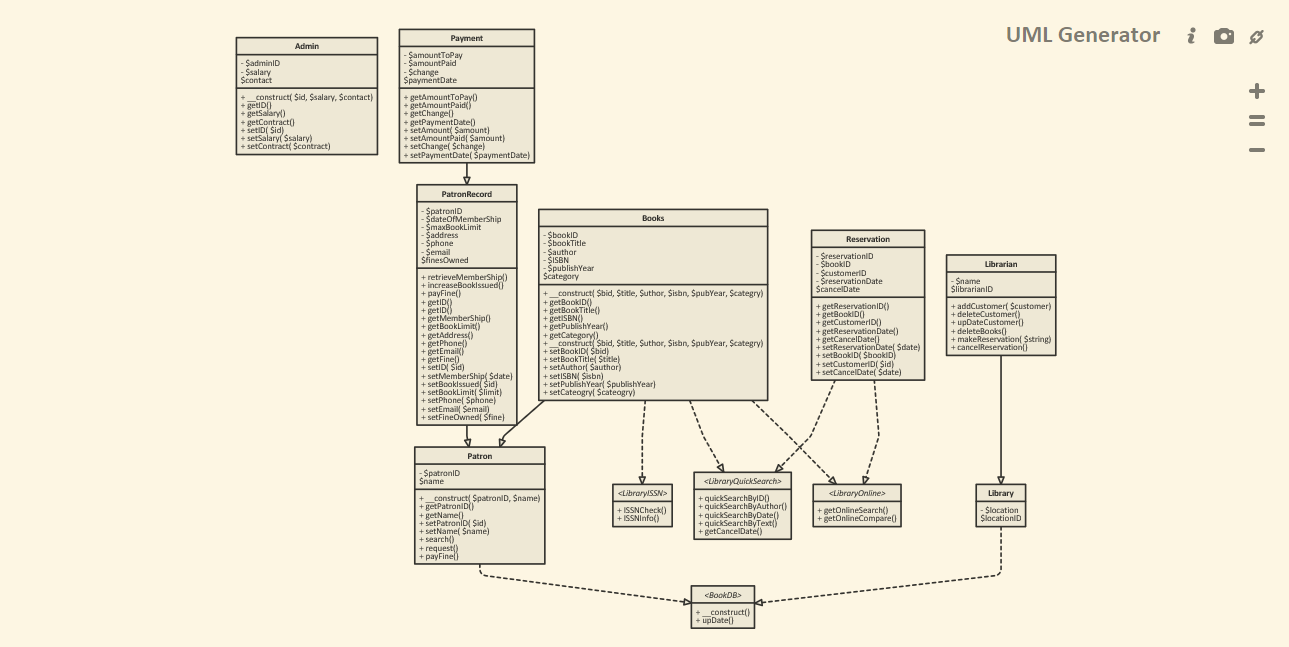
1. **UML Generator page**: To see the result of uploaded PHP files or simple to see the UML diagram which the user uploaded on the previous diagram, they can see it on this page, after uploading process is done. The user can zoom in/out, download UML image, or to share the link with other developers. 

Figure shows the generated UML diagram

# References

## List of references:-

## Book:

1. “Learning PHP, MySQL & JavaScript With jQuery, CSS & HTML5”: [Robin Nixon](https://www.ebooks.com/searchapp/searchresults.net?page=1&term=Nixon%2C+Robin&restrictBy=author), [O'Reilly Media](https://www.ebooks.com/searchapp/searchresults.net?page=1&term=PublisherID:36485&publisherName=O%27Reilly+Media), 2018.
2. “HTML5 Canvas Native Interactivity and Animation for the Web”: [Steve Fulton](https://www.ebooks.com/searchapp/searchresults.net?page=1&term=Fulton%2C+Steve&restrictBy=author); [Jeff Fulton](https://www.ebooks.com/searchapp/searchresults.net?page=1&term=Fulton%2C+Jeff&restrictBy=author), [O'Reilly Media](https://www.ebooks.com/searchapp/searchresults.net?page=1&term=PublisherID:36485&publisherName=O%27Reilly+Media), 2013.

# Code (Copy and Paste all of your project code…..)

**ClassInformation** class is to store the extracted information about classes from the source code

**<?php**

**class ClassInformation**

**{**

**private $properties; // store properties of a class**

**public function \_\_construct( $name, $attributes = array(), $functions = array(), $implements = array(), $extends = null )**

**{**

**$this->properties = array(**

**'name' => $name,**

**'attributes' => $attributes,**

**'functions' => $functions,**

**'implements' => $implements,**

**'extends' => $extends,**

**);**

**}**

**// getter**

**public function \_\_get( $key )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Sorry Doesn't exist");**

**}**

**return $this->properties[$key];**

**}**

**// setter**

**public function \_\_set( $key, $val )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**$this->properties[$key] = $val;**

**}**

**}**

**InterfaceInformation** class is to store the extracted information about interfaces from the source code

**<?php**

**class InterfaceInformation**

**{**

**private $properties; // to store the properties of an interface**

**public function \_\_construct( $name, $functions = array(), $extends = null )**

**{**

**$this->properties = array(**

**'name' => $name,**

**'functions' => $functions,**

**'extends' => $extends,**

**);**

**}**

**// getter**

**public function \_\_get( $key )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**return $this->properties[$key];**

**}**

**// setter**

**public function \_\_set( $key, $val )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**$this->properties[$key] = $val;**

**}**

**}**

**FunctionInformation** class is to store the extracted information about functions of a class or an interface from the source code

**<?php**

**class FunctionInformation**

**{**

**private $properties; // To store the properties of a function**

**public function \_\_construct( $name, $modifier = 'public', $params = array() )**

**{**

**$this->properties = array(**

**'name' => $name,**

**'modifier' => $modifier,**

**'params' => $params,**

**);**

**}**

**// getter**

**public function \_\_get( $key )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**return $this->properties[$key];**

**}**

**// setter**

**public function \_\_set( $key, $val )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**$this->properties[$key] = $val;**

**}**

**}**

**Attribute** class is to store the extracted information about attributes of a class or an interface from the source code

**<?php**

**class Attribute**

**{**

**private $properties; // To store properties of an attribute**

**public function \_\_construct( $name, $modifier = 'public', $type = null )**

**{**

**$this->properties = array(**

**'name' => $name,**

**'modifier' => $modifier,**

**'type' => $type,**

**);**

**}**

**// getter**

**public function \_\_get( $key )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**return $this->properties[$key];**

**}**

**// setter**

**public function \_\_set( $key, $val )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**$this->properties[$key] = $val;**

**}**

**}**

**FunctionParameter** class is to store the extracted information about a function’s parameters information from the source code

**<?php**

**class FunctionParameter**

**{**

**private $properties; // To store the parameters information of a function**

**public function \_\_construct( $name, $type = null )**

**{**

**$this->properties = array(**

**'name' => $name,**

**'type' => $type,**

**);**

**}**

**// getter**

**public function \_\_get( $key )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**return $this->properties[$key];**

**}**

**// setter**

**public function \_\_set( $key, $val )**

**{**

**if ( !array\_key\_exists( $key, $this->properties ) )**

**{**

**throw new Exception("Doesn't Exist");**

**}**

**$this->properties[$key] = $val;**

**}**

**}**

**StructureTokenparserGenerator** class is the most important class for syntax parsing of a PHP file. This class uses all the previous class to generate information about classes and interfaces from the uploaded PHP source code.

**<?php**

**require 'interface.php';**

**require 'class.php';**

**require 'function.php';**

**require 'functionParameter.php';**

**require 'attribute.php';**

**class StructureTokenparserGenerator**

**{**

**private $classes; // to store classes**

**private $interfaces; // to store interfaces**

**private $parserStruct; // to store parser struct**

**private $lastToken; // to store the last token**

**public function \_\_construct()**

**{**

**// Initializing Global Attributes**

**$this->initGlobalAttributes();**

**// Initializing parser attributes**

**$this->initParserAttributes();**

**}**

**// Initialization global attributes**

**private function initGlobalAttributes()**

**{**

**// for storing classes**

**$this->classes = array();**

**// For storing interfaces**

**$this->interfaces = array();**

**}**

**// Intializig parser attributes**

**private function initParserAttributes()**

**{**

**// the intial parser attribute is empty evertything (class, interface, function)**

**$this->parserStruct = array(**

**'class' => null,**

**'interface' => null,**

**'function' => null,**

**'attributes' => array(),**

**'functions' => array(),**

**'typehint' => null,**

**'params' => array(),**

**'implements' => array(),**

**'extends' => null,**

**'modifier' => 'public',**

**'docblock' => null,**

**);**

**// to store the last tokens**

**$this->lastToken = array();**

**}**

**// to create the structure by given the array of files**

**public function createStructure( array $files )**

**{**

**// everytime the global attributes is initialized**

**$this->initGlobalAttributes();**

**// loop through all files**

**foreach( $files as $file )**

**{**

**// for every file, the intial parser attributes is renewed**

**$this->initParserAttributes();**

**$tokens = token\_get\_all( file\_get\_contents( $file ) ); // get the file in string format**

**// Loop through all tokens**

**foreach( $tokens as $token )**

**{**

**// Split into Simple and complex token**

**if ( is\_array( $token ) !== true )**

**{**

**switch( $token )**

**{**

**case ',':**

**$this->comma();**

**break;**

**case '(':**

**$this->opening\_bracket();**

**break;**

**case ')':**

**$this->closing\_bracket();**

**break;**

**case '=':**

**$this->equal\_sign();**

**break;**

**default:**

**// Ignore everything else**

**$this->lastToken = null;**

**}**

**}**

**else if ( is\_array( $token ) === true )**

**{**

**switch ( $token[0] )**

**{**

**case T\_WHITESPACE:**

**$this->t\_whitespace( $token );**

**break;**

**case T\_FUNCTION:**

**$this->t\_function( $token );**

**break;**

**case T\_VAR:**

**$this->t\_var( $token );**

**break;**

**case T\_VARIABLE:**

**$this->t\_variable( $token );**

**break;**

**case T\_ARRAY:**

**$this->t\_array( $token );**

**break;**

**case T\_CONSTANT\_ENCAPSED\_STRING:**

**$this->t\_constant\_encapsed\_string( $token );**

**break;**

**case T\_LNUMBER:**

**$this->t\_lnumber( $token );**

**break;**

**case T\_DNUMBER:**

**$this->t\_dnumber( $token );**

**break;**

**case T\_PAAMAYIM\_NEKUDOTAYIM:**

**$this->t\_paamayim\_neukudotayim( $token );**

**break;**

**case T\_STRING:**

**$this->t\_string( $token );**

**break;**

**case T\_INTERFACE:**

**$this->t\_interface( $token );**

**break;**

**case T\_CLASS:**

**$this->t\_class( $token );**

**break;**

**case T\_IMPLEMENTS:**

**$this->t\_implements( $token );**

**break;**

**case T\_EXTENDS:**

**$this->t\_extends( $token );**

**break;**

**case T\_PUBLIC:**

**$this->t\_public( $token );**

**break;**

**case T\_PROTECTED:**

**$this->t\_protected( $token );**

**break;**

**case T\_PRIVATE:**

**$this->t\_private( $token );**

**break;**

**case T\_DOC\_COMMENT:**

**$this->t\_doc\_comment( $token );**

**break;**

**default:**

**// Ignore everything else**

**$this->lastToken = null;**

**// And reset the docblock**

**$this->parserStruct['docblock'] = null;**

**}**

**}**

**}**

**// One file is completely scanned here**

**// Store interface or class in the parser arrays**

**$this->storeClassOrInterface();**

**}**

**// Fix the class and interface connections**

**$this->fixObjectConnections();**

**// Return the class and interface structure**

**return array\_merge( $this->classes, $this->interfaces );**

**}**

**private function comma()**

**{**

**// Reset typehints on each comma**

**$this->parserStruct['typehint'] = null;**

**}**

**private function opening\_bracket()**

**{**

**// Ignore opening brackets**

**}**

**private function closing\_bracket()**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// The function declaration has been closed**

**// Add the current function**

**$this->parserStruct['functions'][] = array(**

**$this->parserStruct['function'],**

**$this->parserStruct['modifier'],**

**$this->parserStruct['params'],**

**$this->parserStruct['docblock']**

**);**

**// Reset the last token**

**$this->lastToken = null;**

**//Reset the modifier state**

**$this->parserStruct['modifier'] = 'public';**

**// Reset the params array**

**$this->parserStruct['params'] = array();**

**$this->parserStruct['typehint'] = null;**

**// Reset the function name**

**$this->parserStruct['function'] = null;**

**// Reset the docblock**

**$this->parserStruct['docblock'] = null;**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function equal\_sign()**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the equal sign**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_whitespace( $token )**

**{**

**// Ignore whitespaces**

**}**

**private function t\_function( $token )**

**{**

**switch( $this->lastToken )**

**{**

**case null:**

**case T\_PUBLIC:**

**case T\_PROTECTED:**

**case T\_PRIVATE:**

**$this->lastToken = $token[0];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_var( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_VAR**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_variable( $token )**

**{**

**switch( $this->lastToken )**

**{**

**case T\_PUBLIC:**

**case T\_PROTECTED:**

**case T\_PRIVATE:**

**// A new class attribute**

**$this->parserStruct['attributes'][] = array(**

**$token[1],**

**$this->parserStruct['modifier'],**

**$this->parserStruct['docblock'],**

**);**

**$this->lastToken = null;**

**$this->parserStruct['modifier'] = 'public';**

**$this->parserStruct['docblock'] = null;**

**break;**

**case T\_FUNCTION:**

**// A new function parameter**

**$this->parserStruct['params'][] = array(**

**$this->parserStruct['typehint'],**

**$token[1]**

**);**

**break;**

**}**

**}**

**private function t\_array( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_ARRAY**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_constant\_encapsed\_string( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_CONSTANT\_ENCAPSED\_STRING**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_lnumber( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_LNUMBER**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_dnumber( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_DNUMBER**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_paamayim\_neukudotayim( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case T\_FUNCTION:**

**// just ignore the T\_PAAMAYIM\_NEKUDOTAYIM**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_string( $token )**

**{**

**switch( $this->lastToken )**

**{**

**case T\_IMPLEMENTS:**

**// Add interface to implements array**

**$this->parserStruct['implements'][] = $token[1];**

**// We do not reset the last token here, because**

**// there might be multiple interfaces**

**break;**

**case T\_EXTENDS:**

**// Set the superclass**

**$this->parserStruct['extends'] = $token[1];**

**// Reset the last token**

**$this->lastToken = null;**

**break;**

**case T\_FUNCTION:**

**// Add the current function only if there is no function name already**

**// Because if we know the function name already this is a type hint**

**if ( $this->parserStruct['function'] === null )**

**{**

**// Function name**

**$this->parserStruct['function'] = $token[1];**

**}**

**else**

**{**

**// Type hint**

**$this->parserStruct['typehint'] = $token[1];**

**}**

**break;**

**case T\_CLASS:**

**// Set the class name**

**$this->parserStruct['class'] = $token[1];**

**// Reset the last token**

**$this->lastToken = null;**

**break;**

**case T\_INTERFACE:**

**// Set the interface name**

**$this->parserStruct['interface'] = $token[1];**

**// Reset the last Token**

**$this->lastToken = null;**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_interface( $token )**

**{**

**switch( $this->lastToken )**

**{**

**case null:**

**// New initial interface token**

**// Store the class or interface definition if there is any in the**

**// parser arrays ( There might be more than one class/interface per**

**// file )**

**$this->storeClassOrInterface();**

**// Remember the last token**

**$this->lastToken = $token[0];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_class( $token )**

**{**

**switch( $this->lastToken )**

**{**

**case null:**

**// New initial interface token**

**// Store the class or interface definition if there is any in the**

**// parser arrays ( There might be more than one class/interface per**

**// file )**

**$this->storeClassOrInterface();**

**// Remember the last token**

**$this->lastToken = $token[0];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_implements( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->lastToken = $token[0];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_extends( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->lastToken = $token[0];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_public( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->lastToken = $token[0];**

**$this->parserStruct['modifier'] = $token[1];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_protected( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->lastToken = $token[0];**

**$this->parserStruct['modifier'] = $token[1];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_private( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->lastToken = $token[0];**

**$this->parserStruct['modifier'] = $token[1];**

**break;**

**default:**

**$this->lastToken = null;**

**}**

**}**

**private function t\_doc\_comment( $token )**

**{**

**switch ( $this->lastToken )**

**{**

**case null:**

**$this->parserStruct['docblock'] = $token[1];**

**break;**

**default:**

**$this->lastToken = null;**

**$this->parserStruct['docblock'] = null;**

**}**

**}**

**private function storeClassOrInterface()**

**{**

**// First we need to check if we should store interface data found so far**

**if ( $this->parserStruct['interface'] !== null )**

**{**

**// Init data storage**

**$functions = array();**

**// Create the data objects**

**foreach( $this->parserStruct['functions'] as $function )**

**{**

**// Create the needed parameter objects**

**$params = array();**

**foreach( $function[2] as $param)**

**{**

**$params[] = new FunctionParameter( $param[1], $param[0] );**

**}**

**$functions[] = new FunctionInformation(**

**$function[0],**

**$function[1],**

**$params**

**);**

**}**

**$interface = new InterfaceInformation(**

**$this->parserStruct['interface'],**

**$functions,**

**$this->parserStruct['extends']**

**);**

**// Store in the global interface array**

**$this->interfaces[$this->parserStruct['interface']] = $interface;**

**}**

**// If there is no interface, we maybe need to store a class**

**else if ( $this->parserStruct['class'] !== null )**

**{**

**// Init data storage**

**$functions = array();**

**$attributes = array();**

**// Create the data objects**

**foreach( $this->parserStruct['functions'] as $function )**

**{**

**// Create the needed parameter objects**

**$params = array();**

**foreach( $function[2] as $param)**

**{**

**$params[] = new FunctionParameter( $param[1], $param[0] );**

**}**

**$functions[] = new FunctionInformation(**

**$function[0],**

**$function[1],**

**$params**

**);**

**}**

**foreach ( $this->parserStruct['attributes'] as $attribute )**

**{**

**$type = null;**

**// If there is a docblock try to isolate the attribute type**

**if ( $attribute[2] !== null )**

**{**

**// Regular expression that extracts types in array annotations**

**$regexp = '/^[\s\*]\*@var\s+array\(\s\*(\w+\s\*=>\s\*)?(\w+)\s\*\).\*$/m';**

**if ( preg\_match( $regexp, $attribute[2], $matches ) )**

**{**

**$type = $matches[2];**

**}**

**else if ( $return = preg\_match( '/^[\s\*]\*@var\s+(\S+).\*$/m', $attribute[2], $matches ) )**

**{**

**$type = trim( $matches[1] );**

**}**

**}**

**$attributes[] = new Attribute(**

**$attribute[0],**

**$attribute[1],**

**$type**

**);**

**}**

**$class = new ClassInformation(**

**$this->parserStruct['class'],**

**$attributes,**

**$functions,**

**$this->parserStruct['implements'],**

**$this->parserStruct['extends']**

**);**

**$this->classes[$this->parserStruct['class']] = $class;**

**}**

**$this->initParserAttributes();**

**}**

**private function fixObjectConnections()**

**{**

**foreach( $this->classes as $class )**

**{**

**$implements = array();**

**foreach( $class->implements as $key => $impl )**

**{**

**$implements[$key] = array\_key\_exists( $impl, $this->interfaces )**

**? $this->interfaces[$impl]**

**: $this->interfaces[$impl] = new InterfaceInformation( $impl );**

**}**

**$class->implements = $implements;**

**if ( $class->extends === null )**

**{**

**continue;**

**}**

**$class->extends = array\_key\_exists( $class->extends, $this->classes )**

**? $this->classes[$class->extends]**

**: ( $this->classes[$class->extends] = new ClassInformation( $class->extends ) );**

**}**

**foreach( $this->interfaces as $interface )**

**{**

**if ( $interface->extends === null )**

**{**

**continue;**

**}**

**$interface->extends = array\_key\_exists( $interface->extends, $this->interfaces )**

**? $this->interfaces[$interface->extends]**

**: ( $this->interfaces[$interface->extends] = new InterfaceInformation( $interface->extends ) );**

**}**

**}**

**}**

**?>**