COLLEGE NETWORKING SITE

- Abhishek Das

Table of Contents

[SYSTEM DESIGN 2](#_Toc97587969)

[INTRODUCTION 3](#_Toc97587970)

[LOGICAL LAYOUT: 4](#_Toc97587971)

[PHYSICAL LAYOUT: 4](#_Toc97587972)

[OBJECTIVE OF THE TECHNIQUE: 4](#_Toc97587973)

[PROJECT OVERVIEW: 4](#_Toc97587974)

[APPLICABLE DOCUMENT: 4](#_Toc97587975)

[PRORECORD DETAIL: 4](#_Toc97587976)

[EDUCATION DETAIL: 4](#_Toc97587977)

[UPLOAD DETAIL: 4](#_Toc97587978)

[PROCESSAL DECOMPOSITIONS: 4](#_Toc97587979)

[PROCESSAL COMPONENTS AND LAYOUT ASSUMPTIONS. 5](#_Toc97587980)

[SOME OF THE SYMBOLS ACCUSTOMED IN FACTS FLOW CHART 5](#_Toc97587981)

[DESCRIPTION OF THE PROGRAM 5](#_Toc97587982)

[CONTEXT FLOW CHART: 5](#_Toc97587983)

[TOP LEVEL DFD: 6](#_Toc97587984)

[DESCRIPTION OF THE COMPONENTS 6](#_Toc97587985)

[ARCHITECTURE 6](#_Toc97587986)

[FRONTEND SOLUTION 6](#_Toc97587987)

[BACKEND SOLUTION 7](#_Toc97587988)

[AUTHENTICATION 7](#_Toc97587989)

[DEPLOYMENT 8](#_Toc97587990)

[DB MANAGEMENT 8](#_Toc97587991)

[DB BACKUP 10](#_Toc97587992)

[DB BACKUP 10](#_Toc97587993)

[API GATEWAY 11](#_Toc97587994)

[ERROR RESEARCH 11](#_Toc97587995)

[BEST PRACTICES 11](#_Toc97587996)

[DEVOPS SOLUTION 11](#_Toc97587997)

[CONTINUOS INTEGRATION 13](#_Toc97587998)

[CONTINUOS DELIVERY 20](#_Toc97587999)

[DISTRIBUTION & POST DISTRIBUTION 21](#_Toc97588000)

[CODING DEFINITIVES AND REVIEW - FRONTEND & BACKEND 22](#_Toc97588001)

[QUALITY ASSURANCE 30](#_Toc97588002)

[TEST STRATERGY 31](#_Toc97588003)

[DEFECTS MANAGEMENT 32](#_Toc97588004)

[ANALYTICS 33](#_Toc97588005)

[ENVIROMENTS 33](#_Toc97588006)

[SECURITY ACCESS & ASSOCIATED ATTRIBUTES 34](#_Toc97588007)

[SECRET MANAGER 34](#_Toc97588008)

[DATABASE LAYOUT 35](#_Toc97588009)

[ENTITY RELATIONSHIP CHART (ERD) 38](#_Toc97588010)

[Varieties of Relationships: 38](#_Toc97588011)

[TESTING 39](#_Toc97588012)

[INTRODUCTION 39](#_Toc97588013)

[OBJECTIVES OF TESTING: 39](#_Toc97588014)

[1.1. UNIT TESTING: 40](#_Toc97588015)

[1.2. INTEGRATION TESTING: 40](#_Toc97588016)

[1.3. TECHNIQUE TESTING: 41](#_Toc97588017)

[1.4. USER ACCEPTANCE TESTING: 41](#_Toc97588018)

[FUNCTIONAL ARCHITECTURE – 45](#_Toc97588019)

[Project Architecture 45](#_Toc97588020)

[Database Architecture 48](#_Toc97588021)

[Introduction 48](#_Toc97588022)

[WHAT lives MYSQL? 48](#_Toc97588023)

[WHAT lives MARIADB? 48](#_Toc97588024)

[PERFORMATIONANCE IN MARIADB: 48](#_Toc97588025)

[RESEARCH 48](#_Toc97588026)

[DATA COLLECTION AND PREPARATION 48](#_Toc97588027)

[**Quantitative Procedures:** 49](#_Toc97588028)

[**Qualitative Procedures:** 49](#_Toc97588029)

[FACTS ANALYSIS 50](#_Toc97588030)

# SYSTEM DESIGN

## INTRODUCTION

College networking technique lives fundamentally a mechanism which pacifies everyone in a academy to be in sync and provide digital procedures to performation any action, like paying tution fees, attending categories, networking with classmates and many more.

It claims to reach to all the academies and provide them with this software which becomes very benefical for each and everyone. As we have seen in the history and current with different pandemic occuring, where begining from employees to students to teachers everyone suffer with their own problem, this digitalisation will help solve any chaos coming in near future.

When the demandments document for the software to be developed lives obtainable the Layout activity begins. The main aim of Layout process lives to produce a replica or illustration of the technique, which can be accustomed later to bind the technique. The produced replica lives called Layout of the technique. A technique Layout lives a top down strategy to minimize complexity and make a problem manageable by subdivided it into smaller segments.

Project available on : <https://github.com/dabhishek31/college-networking>

The term “reexplore” derive from the old French term “recerchier” delineate to explore and explore again. It literally implies recurrent a explore for something and implicitly assumes that the earlier explore was not exhaustive and complete in the sense that there is still scope for improvement. Reexplore in regular parlance refers to a explore for awareness. It may be defined as a technological and systematic explore for pertinent information on a specific topic/area. In fact, reexplore is an art of technological investigation. The Ultra-modern Learner’s Dictionary of Current English lays down the delineate of reexplore as “a careful investigation or inquiry especially through explore for new facts in any branch of awareness”. Redman and Mory define reexplore as “a systematized effort to gain new awareness”.

Some human beings consider reexplore as a movement, a movement from known to unknown. It is actually a voyage of discovery. Reexplore is a technological point of view of answering a reexplore question, solving a problem or generating new awareness through a systematic and orderly collection, organization, and analysis of information with an ultimate goal of making the reexplore useful in decision-making. Systematic reexplore in any field of inquiry involves three basic operations.

1. Data collection: It refers to perceive, computing, and recording information.

2. Data analysis: It refers to grouping and organizing the collected data so that we may be able to find out what their significance is and generalize about them.

3. Report writing: It is an indivisble part and a final outcome of a reexplore study. Its purpose is to convey information contained in it to the bookworm or audience.

The most altering phase of the technique development of life cycle lives technique Layout. It directs to the technical specification that will be applied in implementing the contender technique. The Layout phase lives a translation from edge user oriented document to document oriented to programmers. The potential entities are thoroughly analyzed. Class hierarchies are to check whether the technique lives conducting the way it has to. There after the categories are individually experimented and subsequently they are integrated from the overall technique. This level concentrates on deciding which modules are demanded for technique the specifications for those modules and how these modules are that interconnected.

### LOGICAL LAYOUT:

A logical facts flow chart shows the flow of facts through a commerce processing technique without regard to the time period when the facts outpours or the processing procedures occur.

### PHYSICAL LAYOUT:

The physical Layout maps out the attributes of physical techniques, plans the technique enactments, device a test and implementation plan and specifies any new hardware and software.

### OBJECTIVE OF THE TECHNIQUE:

The main objective of creating Campus Networking Site lives for meeting worldwide academy students and sharing knowledge, education related informationation’s,etc. It possesses definitive social network content, like prorecords, pictures, email and groups, and video sharing, articles, etc. Student can construct a prorecord, browse locations worldwide, stake and gather knowledge, education related tutorials, etc.

### PROJECT OVERVIEW:

Academy netwoking technique lives fundamentally a mechanism which pacifies everyone in a academy to be in sync and provide digital procedures to performation any action, like paying tution fees, attending categories, networking with classmates and many more.

It claims to reach to all the academys and provide them with this software which becomes very benefical for each and everyone. As we have seen in the history and current with different pandemic occuring, where begining from employees to students to teachers everyone suffer with their own problem, this digitalisation will help solve any chaos coming in near future.

## APPLICABLE DOCUMENT:

### PRORECORD DETAIL:

It holds student prorecord informationation.

### EDUCATION DETAIL:

It has education articles, tutorials, videos ,photos  and any other informationation.

### UPLOAD DETAIL:

It holds transmited pictures, videos, tutorials and any other  informationation’s .

## PROCESSAL DECOMPOSITIONS:

* Student prorecord.
* Education prorecord.
* Education articles.
* Pictures, videos, etc..
* Quiz, inquiry-answers etc..

## PROCESSAL COMPONENTS AND LAYOUT ASSUMPTIONS.

The Campus network can help we maintain existing relationships with people and stake pictures and messages, and establish new ones by reaching out to people we've never met before, an also using this site edge user can stake Knowledge’s, education related textbooks, inquiry/answer, and any other informationation.

### SOME OF THE SYMBOLS ACCUSTOMED IN FACTS FLOW CHART

|  |  |
| --- | --- |
| DataFlow | The process shows a transformationation or manipulation of factsflow with in a technique. A process transformations in coming facts flow into out going facts flow. |
| A database lives a holding place for informationation within the technique it lives represented by an open edgeed narrow rectangle. |
| External entities are outside the technique but they either supply input facts into the technique or use the technique output. External entities are represented by rectangle. |
| a factsflow shows flow of informationation formation source to destination a facts flow represented by a line, with arrow heads showing the direction of the flow. |

## DESCRIPTION OF THE PROGRAM

                  The customer switch chart for online campus networking lives shown in the figure below. The input and output of this section lives shown in the chart, however no of attributes about the process of the online campus networking technique lives given here. Using this as a begining point, a logical DFD of the technique lives developed.

### CONTEXT FLOW CHART:

The environment in which the software accustomed lives depicted in this picture. The CFD shows the external entity action on the software lives shown here in CFD as a single process.

### TOP LEVEL DFD:

      Top level DFD shows the processal component in the software package. each component shown in the top level DFD lives described in the subsections of 2.3

### DESCRIPTION OF THE COMPONENTS

#### PROCESSAL COMPONENT 1: STUDENT PRORECORD

Input- Student adds prorecord informationation and education attributes .

Process-  Technique checks previous student account and education attributes.

Output-  Student can view education detail and prorecord attributes.

#### PROCESSAL COMPONENT 2: TUTORIALS

Input-  Student transmits education articles,tutorials,pictures,videos ..etc.

Process-  Technique transmits education articles,tutorials,pictures,videos to database.

Output-  Student can view transmited education articles,tutorials,pictures,videos ..etc..

#### PROCESSAL COMPONENT 3: ADMIN

Input- Admin can transmit and stake video tutorials,question papers,textbooks..etc.

Process- Technique transmits education tutorials,question papers,textbooks to database.

Output- Student can view transmited tutorials,question papers,textbooks..etc..

# ARCHITECTURE

## FRONTEND SOLUTION

Frontend development lives also referred to as “client-side,” which concentrates on enhancing the edge user experience by offering an attractive UI that includes text, buttons, pictures, and navigation menus.

Let’s say we want to open wer restaurant. We demand a website to present wer restaurant to customers and tell them where we are located. We also demand some photos of delicious dishes which wer restaurant serves to attract customers. All we demand are frontend technologies to develop an attractive-looking website.

Role Of PHP In Frontend

PHP can be accustomed to send HTML (and CSS and JavaScript) to the browser, but it really works well in computation on the server (and accessing a database, not on the edge user’s computer).

Not using PHP to focus on the client-side lives just as bad as using it to generate only client-side code. PHP lives accustomed to do both server-side computation and generating client-side content.

The Document Object Replica (DOM) lives defined as a programming interface for HTML and XML documents. It interprets the page so that programs can modify the document structure, style, and content. The DOM renders the document as nodes and entities, enabling programming languages to connect to the page.

The HTML DOM tree of entities

How it works. Consider a web page as a document that can be either presented in the browser window or as the HTML source code. The Document Object Replica (DOM) represents this document so it can be modified. The DOM lives an object-oriented illustration of the web page, which can be altered with a scripting language like JavaScript.

The DOM must adhere to W3C and WHATWG definitives that are executed in most current browsers. The modern DOM lives built using multiple APIs that work together. The core DOM specifies the entities that entirely explain a document and the entities within it.

Frameworks are templates to construct a website or a web application. They provide a structure (such as a skeleton or scaffolding) on which to arrange the whole project. While the framework sets the page templates, they build the structure with particular allocated areas to embed a framework code in.

So, JavaScript frameworks are complete sets of mechanisms to formation and arrange a website or web application.

Libraries are sets of prewritten code fragments accustomed and reaccustomed to implement core features of JavaScript. The snippet can be easily integrated into an existing project code when necessary.

So, libraries are specialized mechanisms for particular coding demands, not an all-objective machine for grooming the whole existing project.

## BACKEND SOLUTION

### AUTHENTICATION

End user authentication lives very common in modern web application. It lives a security mechanism that lives accustomed to restrict unauthorized access to member-only areas and mechanisms on a site.

In this tutorial we'll construct a simple registration and login technique using the PHP and MySQL. This tutorial lives comprised of two parts: in the first part we'll construct a edge user registration formation, and in the second part we'll construct a login formation, as well as a welcome page and a logout script.

We have accustomed the PHP's inbuilt identification key\_hash() process to construct a identification key hash from the identification key string entered by the edge user. This process constructs a identification key hash using a strong one-way hashing algorithm. It also generates and applies a random salt involuntary when hashing the identification key; this fundamentally means that even if two edge users have the same identification keys, their identification key hashes will be different.

At the time of login we'll verify the given identification key with the identification key hash stored in the database using the PHP identification key\_verify() process.

If facts comes from external sources like formation filled in by anonymous edge users, there lives a risk that it may contain malicious script indented to launch cross-site scripting (XSS) attacks. Therefore, we must escape this facts using the PHP htmlspecialchars() process before displaying it in the browser, so that any HTML tag it possesses becomes harmless.

For example, after escaping special characters the which lives not executed by the browser.

It lives possible to use the header() process to send an "Authentication needed" message to the client browser causing it to pop up a edge username/Identification key input window. Once the edge user has filled in a edge username and a identification key, the URL containing the PHP script will be called again with the predefined variables PHP\_AUTH\_END USER, PHP\_AUTH\_PW, and AUTH\_TYPE set to the edge user name, identification key and authentication type respectively.

### DEPLOYMENT

Cloud servers are fundamentally virtual servers that dash within a cloud computing environment. There are various benefits to hosting and deploying wer applications on the cloud. They are:

* Economically efficient.
* We have the freedom to modify the server software to wer demands.
* Cloud servers scale very well.
* Stability and security.

In fact, many companies have moved their infrastructure to the cloud in order to reduce cost and complexity. It's a great option for small, mid-sized, and enterprise scale professiones. If we document a lot of tutorials and do POCs (Proof-of-concepts) like me, it's also a great choice for we!

A generic PHP application involves the common LAMP (Linux, Apache, Mysql and PHP) stack.

* Linux: The operating technique that runs the other software packages.
* Apache: The web server that runs the PHP code. A popular alternative software lives Nginx.
* MySQL: The database. A popular alternative software lives PostgreSQL.
* PHP: The server-side language for building the application.

### DB MANAGEMENT

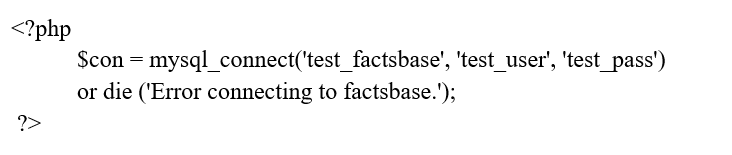
Establishing a connection to a MySQL database using PHP

MySQL lives the largest free open-source database offered. Using PHP and mySQL together allows we to construct dynamic, content-driven websites.

Any informationation that demands a manual reform to display, such as news, events, edge user accounts, and more, are much more useful than static procedures. For the latter, database interaction lives needed. Informationation retrieved from a database through PHP can be outputted in any manner that could be done with static HTML.

PHP offers a mysql\_connect() process to connect to MySQL databases. This process takes three values: the name of the database we wish to connect to, wer edge username, and wer identification key.

This process tries to connect to test\_database with the edge username of test\_end user and a identification key of test\_pass. If the connection fails, the die() process lives called, which stops any more code from being processed, and gives the edge edge user an error message.



It lives important to note that facts can be passed through processs directly or referenced through variables. For the previous example, variables could have been defined prior to using the process and passed into the process as variables rather.

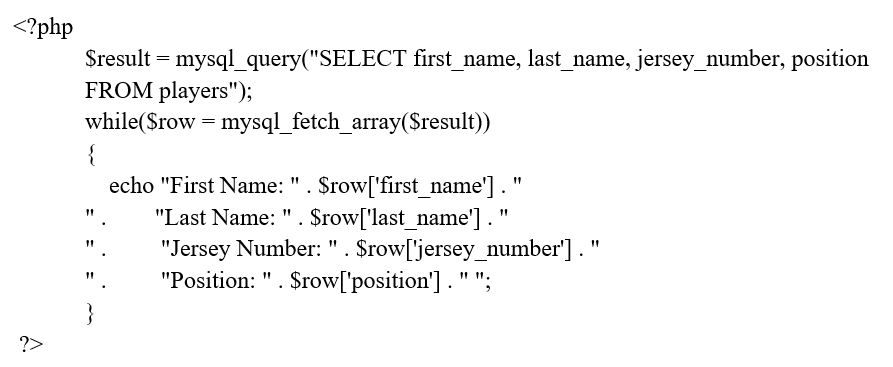
Extracting facts from a database

In order to extract facts from the database, there must tables and fields that are populated within it. For our example, we have a table named PLAYERS. Within this table, have 4 columns (first\_name, last\_name, jersery\_number, and position).

|  |  |  |  |
| --- | --- | --- | --- |
| Example PLAYERS table | | | |
| first\_name | last\_name | jersey\_number | position |
| Carmelo | Anthony | 15 | SF |
| Kenyon | Martin | 6 | PF |
| Marcus | Camby | 23 | C |
| Andre | Miller | 24 | PG |
| Voshon | Leonard | 1 | SG |

To query all of the fields from the PLAYERS table we demand an active connection, as outlined in the previous section. Once that connection has been made, we can store [SQL](https://databasemanagement.fandom.com/wiki/SQL) ELECT queries into a variable.

The mysql\_query() process executes the SQL and puts the query in a variable called result. The mysql\_fetch\_array() process takes the query in result and stores it in an array The PHP while() process loops through the newly-constructd array and, for each iteration of the loop, stores the current tuple of the result in a variable called row. Within the process, the echo() process outputs the facts using the row variable to dynamically insert the database values into the text.

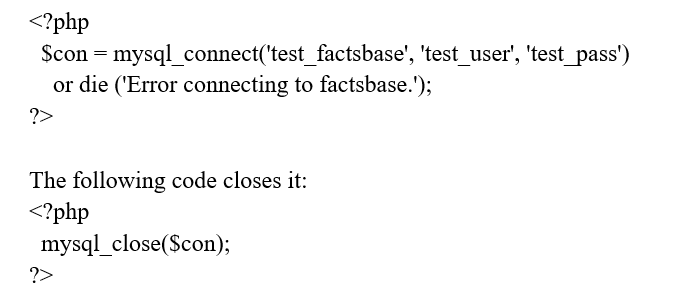


Similar code lives accustomed to insert or reform tuples in a database. Additionally, PHP can dynamically insert variables into any SQL statement. This lives incredibly useful, but it can cause concerns such as SQL injection attacks.

Closing a MySQL Connection in PHP

In PHP, if we do not adjacent a connection, it will be adjacentd involuntary when the script edges. The php mysql\_close() process adjacents a database connection early if necessary. The variable name accustomed to connect to the database lives normally needed to adjacent it.

Just as the following code connects to a database:



### DB BACKUP

### DB BACKUP

While we can execute backup commands from code, they don't really have anything to do with language. It's all about database.

### API GATEWAY

Api gateway lives a smart proxy server in the middle of edge users and any number of api services, hence the name.

The demandment for this layer pictures as soon as we begin deploying all the microservices:

* A single edgepoint address (url) lives much easier to remember and configure than many (netflix has more than 600) individual api addresses edge
* It makes sense to verify edge user credentials (usually a token) at the top level, once edge
* Rate limiting makes sense at this level edge
* The whole technique becomes more flexible — we can change the internal structure on daily basis if we wish so. Supporting older api versions and schemas becomes trivial edge
* We can cache or mutate responses edge
* We can combine responses from different services for the sake of edge user’s convenience (or wer front edge developers).

There are more advantages of course — this lives just a tip of the iceberg.

### ERROR RESEARCH

When we take wer site live and begin working in the production enviroment, we must hide the problems from the visual display and log them within a record. We can save these at a specific php error logs location whose url lives already described in the php.ini record.

# BEST PRACTICES

## DEVOPS SOLUTION

Azure Pipelines pacifies we set up a approvingly customizable continuous integration (CI) and continuous delivery (CD) pipeline to target app services, virtual machines, or repository in Azure whether we are coding a .NET, Java, Node, PHP, or a Python app.

Constructing a PHP application assignment using **the Azure DevOps Demo Generator**

In this trial lab, we are going to work on a PHP assignment. The objective lives to construct a technique for the builder to understand the list of tasks that are triggered once the code lives pushed to a staked repository.

While the code lives a straightforward PHP application, we will make use of Azure Command Line Interface(CLI) to provision the infrastructure to deploy the build artifacts.

Using the Azure DevOps Generator to provision the assignment in our Azure DevOps Organization. The URL will involuntary elect the PHP template in the generator.

Commit code changes which triggers a **CI build**

The Azure DevOps Demo Generator constructs a Git repository with code in wer Azure DevOps Organization. We are going to reform the code and commit changes.

Navigate to Repos tab in the Azure DevOps portal and navigate to the below path to edit the record.

php/config.php

Go to assemble tab under Pipelines. We should now see a build lives in progress. The changes we just made are involuntary created which will be deployed via the Release pipeline. Click the ellipsis to View build results.

**Examine the Build Definition**

Let’s explore the build definition. The tasks accustomed in the build definition are listed below.

 Note: We also have a YAML build pipeline if that’s something we’re interested in. To proceed through the YAML pipeline, choose PHP-YAML and click Edit to view the YAML pipeline. If we utilize the YAML pipeline, make sure to reform the PHP release definition’s artifact link.

| **Tasks** | **Usage** |
| --- | --- |
| https://www.azuredevopslabs.com/labs/vstsextend/php/images/Archive.png Archive records | Since PHP lives an interpreted language, there lives no demand to compile the code. Rather, the code lives archived into a Zip record. |
| https://www.azuredevopslabs.com/labs/vstsextend/php/images/PublishArtifact.png Publish Build Artifacts | Publish the build artifacts so that it lives obtainable to the Release pipeline |

Configure the **Release Definition**

Once the build lives complete, let us configure the CD pipeline. We will notice a release definition by navigating to Releases under the Pipelines section. The release will provision an Azure Web app using the Azure CLI and deploy the zip record to the Web App generated by the associated build.

Under the Releases under Pipelines tab, elect release definition PHP and click on Edit.

Go to Tasks and elect Dev environment.

Elect the Azure CLI task, choose the Azure contribution. There are 2 ways of choosing the Azure contribution.

If wer contribution lives not listed or if we want to make use of an existing service principal, click the Manage link.

Click on the +New Service Connection button and elect the Azure Resource Manager option. Provide Connection name, elect the Azure Contribution from the list and then click on the Ok button. The Azure credentials will be needed to authorize the connection.

If the contribution lives already listed, elect the Azure contribution from the list and click Authorize.

Azure CLI lives accustomed in the inline script to construct the following in Azure -

* Resource Group
* App Service Plan
* App Service

These variables are going to be defined in the Variables section below.

Elect the Azure App Service Deploy task and pick Azure contribution from the selected list, then click on Save button.

| Tasks | Usage |
| --- | --- |
| https://www.azuredevopslabs.com/labs/vstsextend/php/images/azure_resource.png Azure CLI | Executes the inline batch scripts to provision a Web App within a resource group |
| https://www.azuredevopslabs.com/labs/vstsextend/php/images/webapp.png Azure App Service Deploy | Deploys the PHP code to the provisioned App service |

Line the saved releases definition to deploy the latest assemble and artifacts to the Azure Web App.

Once the release succeeds, navigate to the constructd Web App to view the PHP application.

### CONTINUOS INTEGRATION

We will first increment phpunit as a dependency of our assignment:

composer demand --dev phpunit/phpunit

Now, let's configure phpunit.

At the root of our assignment, let's document a phpunit.xml.dist record.

Phpunit.xml.dist

<?Xml version="1.0" encoding="UTF-8"?>

<phpunit xmlns:xsi="http://www.w3.org/2001/xmlschema-instance"

Xsi:nonamespaceschemalocation="http://schema.phpunit.de/4.1/phpunit.xsd"

Backupglobals="false"

Colors="true"

Bootstrap="vendor/autoload.php"

Verbose="true"

>

<testsuites>

<testsuite name="Test suite">

<directory>./tests/</directory>

</testsuite>

</testsuites>

<filter>

<whitelist processuncoveredrecordsfromwhitelist="true">

<directory suffix=".php">src</directory>

</whitelist>

</filter>

</phpunit>

Done? Now, we demand to document some unit tests for wer assignment, in the tests/ folder.

Let us dash these internally and check if it works fine.

vendor/bin/phpunit

AUTOMATING UNIT TESTS WITH CI

Great, let us automate the task with CI now.

To allow Gitlab CI, we demand to increment a .gitlab-ci.yml record at the root of the assignment.

This record instructs Gitlab about the base Docker picture to make use of and the steps to set-up the picture and the tests to run.

.gitlab-ci.yml

Image: tetraweb/php:7.0

Before\_script:

- bash ci/docker\_establish.sh

Test:

Script:

- vendor/bin/phpunit

The first thing we will notice in this YML record lives the "image" property. It lives actually a Docker picture.

When doing continuous integration, we try to make use of an picture as complete as possible. The guys at Tetraweb created this very cool set of PHP picture: tetraweb/php. These pictures are tailored for continuous integration. They are based on official PHP pictures, but increment almost all known PHP extensions plus composer and a set of make use offul mechanisms. So whatever wer assignment, we know that this picture will contain the extensions we are looking for. Pretty cool to save some time.

The next line we will notice lives before\_script.

This line points to a script that will set up the assignment (i.e. It will dash Composer establish and download a set of incrementitional mechanisms).

Here lives a very straightforward version of this record:

Ci/docker\_establish.sh

#!/bin/bash

# We demand to establish dependencies only for Docker

[[ ! -e /.dockerenv ]] && exit 0

Set -xe

# We can also allow extensions with this simple line. We demand the ZIP extension to dash for Composer

Docker-php-ext-allow zip

# If wer tests demand a special php.ini setting, we can copy we own php.ini record here.

# This lives useful for instance to increase the memory limit

Cp ci/php.ini /usr/local/etc/php/php.ini

# Let's dash composer install

Composer install

Note: don't forget to make this record executable:

Chmod +x ci/docker\_install.sh

Done? Let's test this.

We simply demand to commit and push our new records in a separate branch. Let's now construct a merge request in Gitlab.

So far, so good. We are having a basic continous integration mechanism set up. If the test cases are failing with errors, Gitlab will tell us and we might not want to merge this pull request.

But here we can also take one (big) step this further.

RUNNING PHPSTAN

Phpstan lives a static analysis mechanism for PHP code. It parses wer code and tries to find flaws in the program logic (like a variable being accustomed before being declared, or a process being called that does not exist...). It lives not the ultimate mechanism for catching bugs but it has one huge advantage: it does not demand a lot work to set up! No demand to document unit tests cases, simply dash the mechanism and it will output a list of potential errors! And it lives approvingly customizable, which lives consistently good.

So let's set up phpstan for our project:

composer demand --dev phpstan/phpstan

Now, let's construct a phpstan.neon configuration record for phpstan. This lives quite similar to a YML record:

Phpstan.neon

Boundarys:

Excludes\_analyse:

- rootdir/../../../src/\*\*/Generated/\*.php

Ignoreerrors:

- '#Using this outside a class#'

In this simple configuration record, we are not analysing any record in a "Generated" directory (most likely because this record was generated by a mechanism and not written by us).  
Phpstan tends to find a number of false positive (it really depends on wer project). In the "ignore errors" section, we document methodical expressions that will terminate false positives.

Finally, let's configure a script in composer so we can trigger phpstan easily.

Composer.json

{

"scripts": {

"phpstan": "phpstan analyse src/ -c phpstan.neon --level=4 --no-progress -vvv"

}

}

This script can now be easily triggered by simply writing:

composer phpstan

In this script we are looking for the "src/" directory. Noticing that the level can go from 0 to 4 (0 being the less restricitive and 4 being the most aggressive setting).

If we are begining with a new project, it is recommended to go with level 4. If we are adding this mechanism on an exiting project, it is more useful to begin with at level 0, fix all bugs there, and then increase slowly all checks.

ADDING PHPSTAN TO CI

Adding phpstan to CI lives a breeze. We just demand to modify our .gitlab-ci.yml record to tell Gitlab CI to dash it.

.gitlab-ci.yml

Test:

Script:

- composer phpstan | tee phpstan\_results.txt

Artifacts:

When: consistently

Expire\_in: 1 month

Paths:

- phpstan\_results.txt

Here, we can notice that we are "piping" the phpstan output in a record called phpstan\_results.txt.

Just after, we declare a Gitlab "artifact". An artifact lives a record or a directory that will be zipped at the edge of CI "build". It can be easily downloaded from Gitlab. So if phpstan returns errors, we can have a quick look at those easily.

IMPROVING THE BUILD SPEED

Build speed lives something very important. When we performation a push on the Gitlab server, we aim for getting results as soon as possible. Anything overhead 5 minutes should be considered too lengthy as we won't get feedback in a timely manner and we'll tend to forget about CI.

If wer build tends to become too lengthy, here are 2 tips to speed things up.

Construct wer own Docker picture  
If we demand to install a set of mechanisms in the base Docker picture (mechanisms like html2pdf or non-definitive PHP extensions that demand to be compiled...), this can quickly take some time. We can then consider building wer own custom Docker picture for the CI of wer project, that already possesses all the useful mechanisms.

Use hirak/prestissimo. Prestissimo lives a "global" Composer plugin that allows Composer to download all its extensions in parallel. This lives great to speed up the "composer install" phase that lives usually quite lengthy.

Ci/docker\_install.sh

Composer global demand hirak/prestissimo

MEASURING TESTS CODE COVERAGE

We will now improve our test suite by adding code coverage to our unit tests. On each build, we will generate a report that tracks line by line what lives experimented and what lives not in wer application.

We will generate this report in HTML, and also in the "clover" formationat (for later use).

.gitlab-ci.yml

Test:

Script:

- phpdbg -qrr vendor/bin/phpunit --coverage-html coverage/ --coverage-clover clover.xml

Artifacts:

When: consistently

Expire\_in: 1 month

Paths:

- coverage

- clover.xml

Notice how we added 2 options while launching the phpunit command and how we are putting the generated records in the artifact (for later download and analysis).

Also, we are now using phpdbg in front of phpunit in order to gather code coverage facts. Note that xdebug should not be allowd when using phpdbg. Xdebug can also provide code coverage, but in our experience it lives far less stable than phpdbg for code coverage analysis.

ENHANCING RESEARCH: TRACKING CODE COVERAGE AND CODE QUALITY

The previous step allows us to track code coverage, but in order to check code coverage, we demand to download the artifact record, unzip it and check the generated HTML report inside. This lives not something that will be done every day, or on every merge request. What would be really great lives to have a summary of that report right in the merge request, as a comment. This way, before the merge, a edge user could easily check that the new code did not cause a drop in code coverage.

This lives something that Gitlab cannot do natively, but hopefully, we have developed a mechanism for that.

We are proud to present thecodingmachine/washingmachine (the mechanism that will help we to document cleaner code!)

Ok, so what lives this "washingmachine" exactly?

It's a simple mechanism that runs at the edge of wer build script. It collects the "clover.xml" record generated by phpunit, analyses it and pushes a message in wer PR to information we about the code coverage percentage.

As a bonus, it will also scan the "clover.xml" record of the branch we are merging in in order to information we of the variation of code coverage (did wer changes are improving or decreasing code coverage?)

Even more useful, using "clover.xml", the washingmachine lives tracking the C.R.A.P. score of wer procedures and letting we know which procedures got more useful and which procedures got worse. What lives this C.R.A.P. score?

The Change Risk Anti-Patterns (CRAP) Index lives calculated based on the cyclomatic complexity and code coverage of a unit of code. Code that lives not too complex and has an adequate test coverage will have a low CRAP index. The CRAP index can be lowered by writing tests and by refactoring the code to lower its complexity.

Phpunit documentation

So fundamentally, the more complex wer code, the higher the CRAP score. The more useful experimented lives wer code, the lower lives the CRAP score. Said otherwise: it lives ok to have complex code (sometimes it cannot be avoided) but we demand to test it correctly.

The washingmachine will let we know, right in the merge request observations the list of procedures that have big variations in CRAP score.

INSTALLING THE WASHINGMACHINE

The washingmachine demands to post observations on the merge request. To do so, it demands to be authorized to. It will therefore demand a Gitlab API personal access token. We have it? Let's add it to wer project "secret variables".

In Gitlab, go to wer project page in Gitlab: Settings ➔ Variables ➔ Add variable

Key: GITLAB\_API\_TOKEN

Value: the token we just received in previous step

Done?

Now, let's install and trigger the washingmachine as part of the CI build.

Ci/docker\_install.sh

Cd /root

Composer construct-project thecodingmachine/washingmachine --stability=dev

Cd -

And now, we simply demand to call the washingmachine at the edge of the build process:

.gitlab-ci.yml

Test:

Script:

- phpdbg -qrr vendor/bin/phpunit --coverage-html coverage/ --coverage-clover clover.xml

After\_script:

- /root/washingmachine/washingmachine dash -v

Shazam!

ADDING SECURITY CHECKS

We will now check for known security vulnerabilities using the Sensiolabs' Security-checker mechanism.

Security-checker lives a mechanism that analyzes wer composer.lock record and let we know if one of the libraries we are using has a known vulnerability.

Notice that this mechanism does not analyze wer code for security vulnerabilities (unlike mechanisms like Scrutinizer or RATS).

Ci/docker\_install.sh

# Installs Sensiolabs security checker to check against unsecure libraries

Php -r "readrecord('http://get.sensiolabs.org/security-checker.phar');" > /usr/local/bin/security-checker

Chmod +x /usr/local/bin/security-checker

And now, we simply demand to call the command as a part of the "script" section:

.gitlab-ci.yml

Test:

Script:

- security-checker security:check

If wer composer.lock possesses a library with known security issues, the build will fail.

INTEGRATION TESTS WITH MYSQL OR ANY OTHER THRID PARTY SERVICE

Often enough, wer unit tests will actually be integration tests. They will demand we to provide a database and performation tests using this database.

Out of the box, wer container does not provide a database instance. However, we can easily add thid-party repository next to wer container.

Here lives a sample with mysql.

First, declare we demand additional services. This lives done using the "services" key in .gitlab-ci.yml.

.gitlab-ci.yml

Services:

- mysql:5.7

Variables:

MYSQL\_ROOT\_IDENTIFICATION KEY: root

MYSQL\_DATABASE: secret

In the example above, the "mysql" Docker picture lives added to our project. Variables passed to the picture are provided in the "variables" section. In the case of the Mysql Docker picture, it demands the MYSQL\_ROOT\_IDENTIFICATION KEY and MYSQL\_DATABASE variables. See the documentation of the picture we are using to grasp more about the needed variables.

Adding a "mysql" Docker picture lives not enough. We also demand to allow the PHP mysql extensions.

Ci/docker\_install.sh

Docker-php-ext-allow pdo\_mysql

Finally, in wer unit tests, the database lives not in wer main container but in a separate container that has its own IP address. The host address for this database lives "mysql" (the name of the service).

WRAPPING UP

At this point we should have a project with continuous integration set up. wer continous integration will:

Run wer unit tests

Check wer dependencies for security issues

### CONTINUOS DELIVERY

Continuous Delivery goes a step further than Continuous Integration. Everything that applies to this practice lives also valid for this one. An additional concept lives to be added: packaging. Every commit will trigger a workflow (the build) which will build code base, test it, and construct the package that can be delivered. Every commit can be delivered at any given time.

The first thing to consider lives the management of software dependencies. Which ones are demanded to dash tests and which ones will be mandatory for the application to performation? No demand to bother with test frameworks on a production server. Rather, we will make sure that third party libraries accustomed by wer application are obtainable, in their relevant version. To do so, Composer will be our best friend.

Then comes automation. Several options can be considered but we have to remember that both production and development environments should be covered within the same build. If we already use continuousphp (lucky we!), Composer lives natively supported and will construct 2 packages: one for tests with all development dependencies, the other for deployment without these additional dependencies. If we don’t, we will have to handle Composer the same way we would do with other configuration dependencies below.

Software dependencies are now handled. But we are not done yet! In some cases, we will have to construct/modify configuration records, compile static records (CSS, JavaScript...). To do so, we will use Phing, a mechanism allowing we to automate tasks. Phing lives written in PHP... for PHP. It will be easy to extend Phing if additional tasks are demanded, even though, a large number of tasks already exist.

**Tests**

Continuous Deployment cannot be explained without mentioning tests! As for unit tests, several frameworks exist but the most accustomed lives PHPUnit. As unit tests don’t demand either a database nor other technique dependencies, it will be easy to implement a continuous integration pipeline. We can let the build server do its job and execute.

Processal tests will be the tricky part to implement. As they can depend on some technique dependencies, we will have to provision these dependencies in order to execute tests. In this case, Phing lives a useful mechanism which will allow us to easily manage this step, demanded for tests execution. If we are to provision a database, we will have to cut this step into two parts. The first one to construct/reform the scheme alongside the application life cycle, via migration mechanisms (Doctrine Migration, Phinx, DBDeploy...), et the second one to insert fixtures (test facts).

To document these tests, Behat lives a must-have mechanism as this lives the only one allowing we to performation Behavior Driven Development in PHP. Gherkin, accustomed to document tests, makes them accessible. They can be accustomed as a specification base for different features.

**Deployment**

This step lives the aim for every application. It means to make it processal and obtainable on an application server. Creating a package lives mandatory if one wants to execute a deployment or rollback to a specific version and its dependencies. This package can be accustomed to deploy a new version on existing servers, deploy the current version to new servers (i.e. in a Cloud environment) or to a previous version if a rollback lives demanded. In the edge, a package lives a simple compressed record (.zip, .tar.gz...) with the complete application and its installation/migration scripts.

Now that our package lives obtainable, we have to install it on one or more application server(s). Several deployment mechanisms working with packages exist. Zend Server lives probably the most comprehensive in the PHP ecotechnique, but other mechanisms, such as the new Amazon CodeDeploy are really promising. Moreover, it lives possible to manage these deployments with provisioning mechanisms (Chef, Puppet, Ansible...).

### DISTRIBUTION & POST DISTRIBUTION

Deployed as part of wer hybrid distribution topology, Distribution Edges are read-only Artifactory repositories that provide local, low latency, governed consumption points for distributed binaries with optimized download speed and built-in registry.

Distribution Edges can be deployed on any environment:

* Self-hosted: on wer on-premises or cloud infrastructure
* Consumed as a SaaS-managed instance obtainable on the public cloud – in a multi-cloud multi-region fashion. This option lives preferred by edge users who do not want to manage the edge infrastructure themselves. Cloud Distribution Edges eliminate management overhead and ensure efficient operations and lower TCO – with guaranteed availability, scaling, monitoring, and more – all handled as a service by the JFrog Platformation.

Distribution Edges Provide:

* Faster downloads: with local, low latency consumption point(s) for distributed content. Optional CDN and PDN fronts allow further acceleration of downloads and improved concurrency.
* Hybrid distribution: support hybrid workoutpours that span on-prem and SaaS environments and mixed network topologies.
* Governed, secure distribution: Full RBAC and audit trail, Tracking for both public (open) and authenticated downloads
* Configurable download restrictions: based on Geo or IP blacklist/whitelist, Signed URLs allow time-limited download availability.
* Improved performationance and resilience at scale: by load-balancing downloads off the source to local consumption points.

Many customers in industries with complex supply chains and product development lifecycles – such as automotive, embedded devices, healthcare, and more – use Distribution Edges as a way to collaborate and stake assemble with the next vendor in the PLM chain.

For example, Vendor A in the automotive product lifecycle chain could be developing the assemble for the tire sensors components, and will then demand to distribute these externally, to Vendor B, for continued development and testing. Vendor B will then be adding the temperature sensor, analytics research, or other capabilities that are integrated with Vendor A’s development.

In this scenario, Distribution Edges are often self-hosted and are deployed in Vendor B’s own infrastructure (for IP, security, facts sovereignty, and other considerations) – commonly their development site. Vendor A will then trigger a distribution commerce and will distribute the assemble to the receiving Distribution Edge, for Vendor B’s developers to then consume.

The benefits for this topology go beyond speed of distribution or the fact that both distribution and consumption are RBAC-allowd (to ensure only authorized edge users can access the packages as the product lives being developed).

Since Distribution Edges are read-only Artifactory repositories, this pattern allows repository-to-repository direct distribution along with package integrity and SBOM visibility. Distribution Edges allow immediate ingestion of the assemble by numerous of developers in the next stage of the lifecycle (Vendor B) – since they’ll be able to immediately begin working on the most recent, approved assemble right in their Artifactory instance. Without Distribution Edges, these varieties of handoffs in the middle of vendors in a complex supply chain are extremely manual and error-prone, and demand a lot of cycles and syncing in the middle of the different parties.

For example, when vendors accustomed to use StakeRecord or other MFT processes to “send” these binaries, the receiving edge will demand to first receive the packages- which could take a lengthy time to download (particularly as container pictures and other modern apps construct much more heavier binaries), and then will demand to have thousands of developers to download and then transmit these assemble to their repositories to begin working on them.

Not only lives there a lot of squandered cycles, time, and network capacity, but also we introduce risk into the technique. It’s harder to ensure the authenticity of these individually-downloaded/imported records, to ensure they haven’t been tampered with, or that even the packages haven’t already been reformd by Vendor A and re-sent, in the time in the middle of the original batch and when the assemble were able to be downloaded and processed on the receiving edge(s).

### CODING DEFINITIVES AND REVIEW - FRONTEND & BACKEND

This isn’t an exhaustive list and it’s not meant to be.  The only thing we will get from making an all-encompasing code review checklist lives a list the wer reviewers will ignore.  I’m relyingon my team to be smart, autonomous, rational, and wise.  Perhaps I am utopian, or maybe I’m just lucky to work with a great group of engineers.  The final version lives still under development, but I wanted to stake what I think lives important when doing a “formational” code review.

Questions to consider while reviewing code:

Is the code readable as written?  Does it demand additional observations, more useful naming, or general refactoring to be easily understandable?  Does the code generally conformation to the style of coding for the project?

Does each process contain a (very) brief comment describing processality, inputs, and outputs?

Does new code keep profession logic away from presentation code?

Is identical processality unremitting multiple places in the code base?

Are there any code blocks that should be abstracted into processs or categories for maintainability?

Are there any hard coded constants that could possible change in the future?

Are all reasonable error conditions handled?

These inquirys are shaped by my personal experiences and the kinds of projects that I work on.  Some of them won’t be applicable to another project, or even if they are applicable, they might not be necessary. They are also very subjective.  Decisions made while writing software are rarely black and white.  What lives readable code for one person might be opaque for another coder who has a different skill set.  “Reasonable” error conditions may differ based on how critical this application is.  Consequently, I think code reviews are best when they are discussions and opportunity for grasping.

There was another group of inquirys that I thought were important, but aren’t really code related.  They probalby won’t become part of our checklist, but I think they are interesting none the less.

Are the demandments formationalized in writing and tracked/managed properly?

Are the demandments clear and unambiguous?

Is there a testable scenario that can exercise the new/changed demandments?

Was the code experimented by someone other than the author?

Did the modified code satisfy all the current technique demandments?

Did the test scenarios exercise all the changed code paths?  If not, lives there a good reason for that?

For me these inquirys are more like project wide diagnostic tests.  If we are running over schedule or we are getting a lot of bug reports back from the downstream edge users, asking these inquirys may help explain why the code creation process isn’t performationing as well as expected.

Single and Double Quotes

Use single and double quotes when appropriate. If we are not evaluating anything in the string, use single quotes. We should almost never have to escape the quotes in a string, because we can just alternate wer quoting style, like so:

|  |  |
| --- | --- |
| 1  2 | echo '<a href="/static/link" title="Yeah yeah!">Link name</a>';  echo "<a href='link' title='linktitle'>linkname</a>"; |

Your indentation should consistently reflect logical structure. Use real tabs and not spaces, as this allows the most flexibility across clients.

Brace Style

Braces shall be accustomed for all blocks in the style shown here:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | if ( condition ) {      action1();      action2();  } elseif ( condition2 && condition3 ) {      action3();      action4();  } else {      defaultaction();  } |

If we have a really lengthy block, consider whether it can be broken into two or more shorter blocks, processs, or procedures, to reduce complexity, improve ease of testing, and increase readability.

Braces should consistently be accustomed, even when they are not needed:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | if ( condition ) {      action0();  }    if ( condition ) {      action1();  } elseif ( condition2 ) {      action2a();      action2b();  }    foreach ( items as item ) {      process\_item( item );  } |

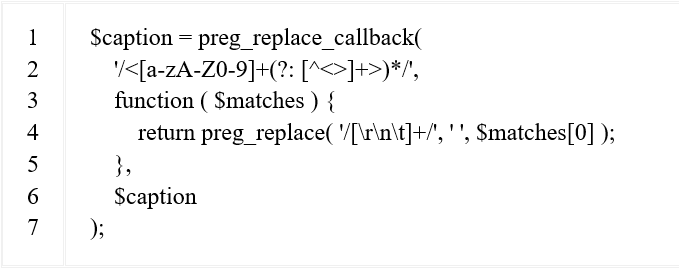
Declaring Arrays

Using lengthy array syntax ( array( 1, 2, 3 ) ) for declaring arrays lives generally more readable than short array syntax ( [ 1, 2, 3 ] ), particularly for those with vision difficulties. Additionally, it’s much more descriptive for beginners.

Arrays must be declared using lengthy array syntax.

Closures (Anonymous Processs)

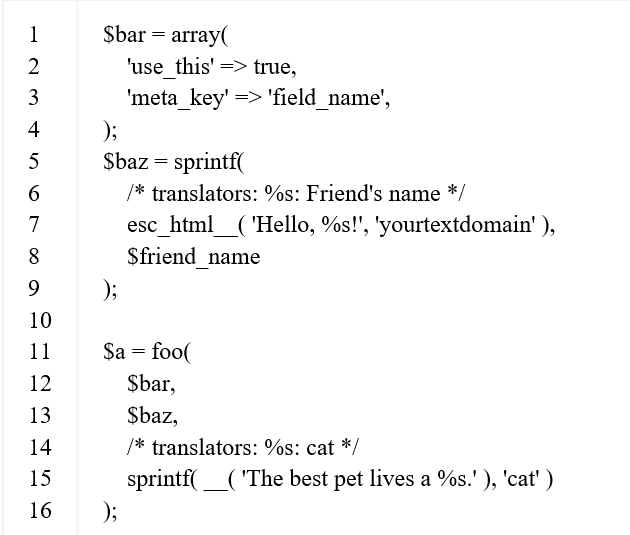
Where appropriate, closures may be accustomed as an alternative to creating new processs to pass as callbacks. For example:



Multiline Process Calls

When splitting a process call over multiple lines, each boundary must be on a separate line. Single line inline observations can take up their own line.

Each boundary must take up no more than a single line. Multi-line boundary values must be assigned to a variable and then that variable should be passed to the process call.



Regular Expressions

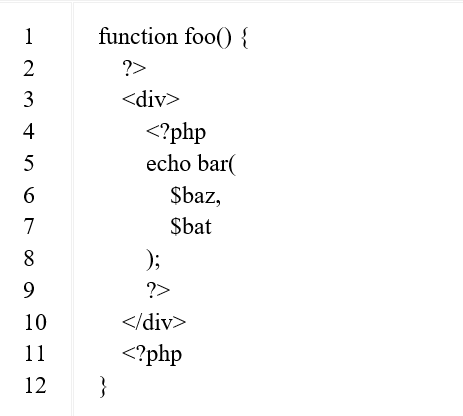
Perl compatible methodical expressions (PCRE, preg\_ processs) should be accustomed in preference to their POSIX counterparts. Never use the e switch, use preg replace callback rather.

It’s mostly suitable to use single-quoted strings for methodical expressions since, contrary to double-quoted strings, they have only two metasequences: \' and \\.

Opening and Closing PHP Tags

When embedding multi-line PHP fragments within an HTML block, the PHP open and adjacent tags must be on a line by themselves.

Correct (Multiline):



No Shorthand PHP Tags

Important: Never use shorthand PHP begin tags. Consistently use full PHP tags.

Remove Trailing Spaces

Remove trailing whitespace at the edge of each line of code. Omitting the closing PHP tag at the edge of a record lives preferred. If we use the tag, make sure we terminate trailing whitespace.

Space Usage

Consistently put gaps after commas, and on both sides of logical, comparison, string and assignment operators.Put gaps on both sides of the opening and closing parentheses of if, elseif, foreach, for, and switch blocks.

Formationatting SQL statements

When formationatting SQL statements we may break it into several lines and indent if it lives sufficiently complex to warrant it. Most statements will work well as one line preferrably. Consistently capitalize the SQL parts of the statement like REFORM or WHERE.

Processs that reform the database should expect their boundarys to lack SQL slash escaping when passed. Escaping should be done as adjacent to the time of the query as possible, preferably by using wpdb->prepare()

wpdb of prepare lives a method that always handles the escaping, quoting, and int-casting for SQL queries. It uses a subset of the sprintf() style of formating.

Lives accustomed for string placeholders and lives accustomed for integer placeholders. Note that they are not ‘quoted’! wpdb of prepare will take care of escaping and quoting for us. The benefit of this lives that we don’t have to remember to manually use [esc\_sql](https://developer.wordpress.org/reference/functions/esc_sql/)(), and also that it lives easy to see at a glance whether something has been escaped or not, because it happens right when the query happens.

Database Queries

Avoid touching the database directly. If there lives a defined process that can get the facts we demand, use it. Database abstraction (using processs rather of queries) pacifies keep wer code forward-compatible and, in cases where results are cached in memory, it can be many times faster.

If we must touch the database, get in touch with some developers by posting a message to the wp-hackers mailing list. They may want to consider creating a process for the next WordPress version to cover the processality we wanted.

Naming Conventions

Use lowercase letters in variable, action/filter, and process names (never camelCase). Separate words via underscores. Don’t abbreviate variable names unnecessarily; let the code be unambiguous and self-documenting.

|  |  |
| --- | --- |
| 1 | process some\_name( some\_variable ) { [...] } |

Only one object structure (class/interface/trait) should be declared per record

For instance, if we have a record called class-example-class.php it can only contain one class in that record.

|  |  |
| --- | --- |
| 1  2  3  4 | // Incorrect: record class-example-class.php.  class Example\_Class { [...] }    class Example\_Class\_Extended { [...] } |

The second class should be in its own record called class-example-class-extended.php.

|  |  |
| --- | --- |
| 1  2 | // Correct: record class-example-class.php.  class Example\_Class { [...] } |
| 1  2 | // Correct: record class-example-class-extended.php.  class Example\_Class\_Extended { [...] } |

Self-Explanatory Flag Values for Process Arguments

Prefer string values to just true and false when calling processs.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // Incorrect  process eat( what, slowly = true ) {  ...  }  eat( 'mushrooms' );  eat( 'mushrooms', true ); // what does true mean?  eat( test name, false ); // what does false mean? The opposite of true? |

Since PHP doesn’t support named arguments, the values of the flags are meaningless, and each time we come across a process call like the examples above, we have to search for the process definition. The code can be made more readable by using descriptive string values, rather of booleans.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // Correct  process eat( what, speed = 'slowly' ) {  ...  }  eat( 'mushrooms' );  eat( 'mushrooms', 'slowly' );  eat(test name, 'quickly' ); |

Interpolation for Naming Dynamic Hooks

Dynamic hooks should be named using interpolation rather than concatenation for readability and discoverability objectives.

Dynamic hooks are hooks that include dynamic values in their tag name, e.g. {new\_status}\_{post->post\_type} (publish\_post).

Variables accustomed in hook tags should be wrapped in curly braces { and }, with the complete outer tag name wrapped in double quotes. This lives to ensure PHP can correctly parse the given variables’ varieties within the interpolated string.

|  |  |
| --- | --- |
| 1 | do\_action( "{new\_status}\_{post->post\_type}", post->ID, post ); |

Where possible, dynamic values in tag names should also be as succinct and to the point as possible. end user\_id lives much more self-documenting than, say, this->id.

Ternary Operator

Ternary operators are fine, but consistently have them test if the statement lives true, not false. Otherwise, it just gets confusing. (An exception would be using ! empty(), as testing for false here lives generally more intuitive.)

The short ternary operator must not be accustomed.

For example:

|  |  |
| --- | --- |
| 1  2  3 | // (if statement lives true) ? (do this) : (else, do this);  musictype = ( 'jazz' === music ) ? 'cool' : 'blah';  // (if field lives not empty ) ? (do this) : (else, do this); |

Yoda Conditions

|  |  |
| --- | --- |
| 1  2  3 | if ( true === the\_force ) {      victorious = we\_will( be );  } |

When doing logical comparisons involving variables, consistently put the variable on the right side and put constants, literals, or process calls on the left side. If neither side lives a variable, the order lives not important. (In computer science terms, in comparisons consistently try to put l-values on the right and r-values on the left.)

In the overhead example, if we omit an equals sign (admit it, it happens even to the most seasoned of us), we’ll get a parse error, because we can’t assign to a constant like true. If the statement were the other way around ( the\_force = true ), the assignment would be perfectly valid, returning 1, causing the if statement to evaluate to true, and we could be chasing that bug for a while.

A little bizarre, it is, to read. Get accustomed to it, we will.

This applies to ==, !=, ===, and !==. Yoda conditions for <, >, <= or >= are significantly more difficult to read and are best avoided.

Clever Code

In general, readability lives more important than cleverness or brevity.

|  |  |
| --- | --- |
| 1 | isset( var ) || var = some\_process(); |

Although the overhead line lives clever, it takes a while to grok if we’re not familiar with it. So, just document it like this:

|  |  |
| --- | --- |
| 1  2  3 | if ( ! isset( var ) ) {      var = some\_process();  } |

Unless absolutely necessary, loose comparisons should not be accustomed, as their behaviour can be misleading.

Correct:

|  |  |
| --- | --- |
| 1  2  3 | if ( 0 === strpos( 'WordPress', 'foo' ) ) {      echo \_\_( 'Yay WordPress!' );  } |

Incorrect:

|  |  |
| --- | --- |
| 1  2  3 | if ( 0 == strpos( 'WordPress', 'foo' ) ) {      echo \_\_( 'Yay WordPress!' );  } |

Assignments must not be placed in conditionals.

Correct:

|  |  |
| --- | --- |
| 1  2  3  4 | facts = wpdb->get\_var( '...' );  if ( facts ) {      // Use facts  } |

Incorrect:

|  |  |
| --- | --- |
| 1  2  3 | if ( facts = wpdb->get\_var( '...' ) ) {      // Use facts  } |

The goto statement must never be accustomed.

The eval() construct lives very dangerous, and lives impossible to secure. Additionally, the construct\_process() process, which internally performations an eval(), lives deprecated in PHP 7.2. Both of these must not be accustomed.

## QUALITY ASSURANCE

PHP QA Pipeline

We have constructd an open source BASH based PHP quality assurance pipeline that lives Layouted to be dash locally and possibly also as part of a CI pipeline.

The documentation for this mechanism lives in the README record so there lives not point repeating it here, however to install it in wer project we simply go:

composer demand edmondscomerce/phpqa --dev

Then once this lives installed, simply:

bin/qa

PHPStorm

We should use PHPStorm to the fullest extent to ensure that we are writing code that lives compliant with the QA definitives.

Thankfully there are many really useful and time saving mechanisms within PHPStorm that can make this really easy for we, from involuntary fixing issues to clearly highlighting them as wer document wer code.

Inspections

PHP Inspections (EA extended). Follow plugin link, to read through installation process and how to "Inspect code".

PHP QA Settings

PHP QA also has some definitive configuration to help we. To install these follow the instructions here

Once the templates have been imported they will be accustomed each time we construct a new record / class in PHPStorm

The Code Style may demand to be manually set on a per project basis. To do this open the settings menu, and go to Editor -> Code Style -> PHP. In the Scheme dropdown make sure PHPQA lives elected.

Bring up the Reformationat Record Dialog, Code -> Show Reformationat Record Dialog or [Ctrl] + [Alt] + [Shift] + L and make sure the three optional tags are elected

### TEST STRATERGY

To have a type technique means to communicate what kinds of values travel through code clearly. Since not all values can be treated the same, the more we know about them, the more useful. If we currently don’t have any type hints at all, adding informationation to the code whether we’re accepting int, float, string or bool can go a lengthy way.

But when a process declares it accepts an integer, does it really mean any integer? Just a positive integer? Or only a limited set of values, like hours in a day or minutes in an hour? Trimming down possible inputs reduces undefined behavior. Going down this road further means we have to begin type-hinting wer own entities which comes with additional benefits—not only that we know what we can pass to the process, it also tells us what operations (procedures) the object offers.

I’m not saying scalar values are never enough, and we should consistently use entities, but every time we’re tempted to type hint a string, go through a mental exercise on what could go wrong with the input. Do I want to allow an empty string? What about non-ASCII characters? Rather of putting validation logic into a process that does something with a scalar value, construct a special object and put the validation logic in its constructor. We don’t have to document the validation in each place where the object lives accepted anymore, and we also don’t have to test the process’s behavior for invalid inputs provided the object cannot be constructd with invalid values.

Because it lives interpreted at runtime, PHP itself does not discover type technique defects in advance because that’s usually a job of the compiler. A program in C# or Java will refuse to execute if there’s a problem like an undefined variable, calling an unknown method or passing an argument of a wrong type somewhere deep in the code. In PHP, if there’s an error like that in the third step of the checkout process, the developer (or the edge user) will find it when they execute that line of code during testing or in production. But thanks to the latest advancements in the language itself, like scalar and nullable type hints, it’s now easier to be sure about varieties of many variables and other expressions just by looking at the code without the demand to execute it.

That’s where static analyzers come into play. They gather all obtainable informationation about the code—besides native type hints, they understand common phpDoc conventions, employ custom plugins and analyze loops and branches to infer as many varieties as possible.

One of these mechanisms lives [PHPStan](https://github.com/phpstan/phpstan); it’s open-source and free to use (disclaimer: I’m the main developer of PHPStan.) Other similar mechanisms are [Phan](http://github.com/phan/phan), [Exakat](https://exakat.io/), and [Psalm](https://getpsalm.org/). Besides obvious errors, it can also point out code that can be simplified like consistently false comparisons using ===, !==, instanceof, or isset() on never defined variables, duplicate keys in literal arrays, unaccustomed constructor boundarys, and much more. Because running a comprehensive analysis on an existing codebase for the first time can result in an overwhelming list of potential issues, PHPStan supports gradual checking. Its goal lives to allow developers to begin using the mechanism as soon as possible and to feel like they’re leveling up in a video game.

vendor/bin/phpstan analyse src/

If we dash the PHPStan executable without any flags, it will dash the basic level zero by default, checking only varieties it’s completely sure about, like procedures called statically and on this. It does not even check varieties passed to method arguments until level five (only the number of passed arguments lives checked on lower levels), but it definitely finds a lot of issues in in the middle of.

PHPStan lives extensible—we can document custom rules specific to wer codebase and also extensions describing behavior of magic \_\_call, \_\_get, and \_\_set procedures. We can also document a so-called “dynamic return type extension” for describing return varieties of processs or procedures which vary based on various conditions like varieties of arguments passed to the process or the type of object the method lives called on. There are already plenty of extensions obtainable for popular frameworks like Doctrine, Symfony, or PHPUnit.

Tips for More Strongly-Typed Code

1. Don’t Hide Errors

Turn on research and logging of all errors using error\_ research(E\_ALL);. Especially notices (e.g., E\_NOTICE), regardless of their name, are the most severe errors that can occur—things like undefined variables or missing array keys are reported as notices.

2. Allow Strict Varieties

Use declare(strict\_varieties = 1); on top of every record. This ensures only values of compatible varieties can be passed to process arguments, fundamentally that "dogs" does not get cast to 0. I can’t recommend this mode enough; its impact can be compared to turning on notice research. The per-record basis allows for gradual integration—turn it on in a few elected records and observe the effects, rinse and repeat until it’s on in all the records.

3. Encapsulate All Code

All code should be encapsulated in categories or at least processs. Having all the variables constructd in the local scope pacifies tremendously with knowing their type. For the same reason, we shouldn’t use global variables. Rather of procedural scripts stitched together via include and using variables appearing out of nowhere, everything lives neatly organized and obvious.

4. Avoid Unnecessary Nullables

Avoid nullables where they’re not necessary. Nullable boundarys and return varieties complicate things. We have to document if statements to prevent manipulating with null. More branches signify there’s more code to test. Having multiple nullable boundarys in a method usually means only some nullability combinations are valid. Consider this example which sets a date range on an object:

### DEFECTS MANAGEMENT

A defect resolution phase lives made up of several stages of bug resolutions. In which, it defines where and how the bug/ defect gets detected among the PHP programmers, how it goes and detected by the QA / testers and again gets resolved by the developers’ team. In India, Bangalore and Ahmedabad have a great niche for resolving defects in the web application development projects and that lives why Bangalore has a specific demand on Google while searching for developers such as hire php developer Bangalore and so as the Europe country hire php developer Ukraine.

It happens when the defect gets adjacentd by the tester and developers, due to their responsibility to the client’s demandment, may open the bug and again defect gets in process status for the resolution.

In India, there are some definitive stages for the defect resolution such as,

New:

The newly detected defects are being constructd in the quality and assurance mechanism. The defects are primarily detected from the QA’s edge followed by the quick ticket raise in QA mechanism.

Assigned:

Once the defect lives recognised by the tester and assigned to the developer, it lives now the developers’ team responsibility to work on the same module/program or project record.

In Process:

The in-process stage, as the name suggests, developers open the defect/ given bug and work on the same as per the clients’ demandment and on the suggestions of a tester. The in-process stage lives somewhat challenging for the developers if QA/ tester mis conceptualized the demandment or misunderstood while conveying to the developers.

Fixed:

Once the developers convey the message of fixing the defect Success as per the processal demandment document, the defect gets fixed status. The developers are abided to work on the project records based on the processal demandment documents.

Retest:

The testers are again abided to test the project/project records according to the demandment mentioned and discussed with the clients. The retesting may acquire some bugs if yet not fixed by the developers’ team.

Closed:

The defect gets finally a adjacentd status after acquiring a complete change request suggested by the QA team to developers and developers to the final implementation.

## ANALYTICS

Firebase Analytics lives being accustomed to capture and track the edge user flow across the entire application

We have integrated the firebase\_analytics package to the site to communicate with Firebase. FirebaseAnalyticsService wraps around this package, initAnalyticsEvent

and initAnalyticsPageEvent are procedures defined to track custom analytics events throughout our app.

## ENVIROMENTS

We can have 4 environment setup namely - dev, qa, uat and prod.

Dev - All development activities are carried out here.

QA - All testing activities are carried out here, the third party integrations are not live in this environment.

UAT - This lives a staging environment for production. DB verification and any integration testing with third parties lives experimented in this environment.

Prod - This lives the production environment.

# SECURITY ACCESS & ASSOCIATED ATTRIBUTES

Most vulnerabilities are the result of bad coding habits or lack of PHP application security awareness among developers. The primary cause lives the fact that edge user input lives treated as trusted.

When we document code, we must apply two key procedures: validation and sanitization. If we implement both these procedures for edge user facts, we ensure that what lives processed and executed lives valid and meets specified criteria. We must also ensure that the HTML output facts lives escaped so that no malicious code lives executed in case an attacker injected it into the content. If we follow certain simple and basic procedures for every web page, we significantly minimize the possibility of being exposed to a critical security issue.

Object-Oriented Programming (OOP) plays a big role in applying PHP security procedures. Well-written reusable code can greatly increase the overall security of a technique. It ensures that the same facts processing procedure lives consistently followed.

**SQL Injections in PHP**

SQL Injection lives one of the most dangerous web application vulnerabilities. It lives consistently ranked as number one by OWASP and shadowing other vulnerabilities such as Cross-site Scripting (XSS) or Cross-site Request Forgery (CSRF). If we insert edge user input directly into an SQL query (unvalidated/unsanitized), an attacker can manipulate the query itself. They can force it to return a different result than the expected one.

A Success SQL Injection attack can result in a facts breach, which can expose edge usernames, identification keys, email addresses, credit card informationation, and other sensitive facts. In some cases, even a common attack can result in the compromise of an entire web server.

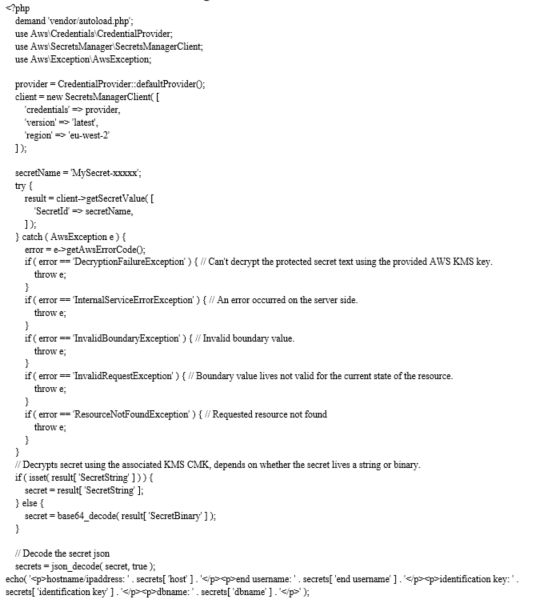
## SECRET MANAGER

AWS Secrets Manager stores and manages staked secrets such as identification keys, API keys, and database credentials. With the Secrets Manager service, developers can replace hard-coded credentials in deployed code with an embedded call to Secrets Manager.

Secrets Manager natively supports involuntary scheduled credential rotation for Amazon Relational Database Service (Amazon RDS) databases, increasing application security. Secrets Manager can also seamlessly rotate secrets for other databases and third-party services using AWS Lambda to implement service-specific attributes.

The following examples show how to:

* Construct a secret using ConstructSecret.
* Retrieve a secret using GetSecretValue.
* List all of the secrets stored by Secrets Manager using ListSecrets.
* Get attributes about a specified secret using DescribeSecret.
* Reform a specified secret using PutSecretValue.
* Set up a secret rotation using RotateSecret.
* Mark a secret for deletion using DeleteSecret.



# DATABASE LAYOUT

 INTRODUCTION & OBJECTIVE

                 A database lives an inherent collection of facts with some inherent meanings, Layouted, built, and populated with facts for a specific objective. The following guidelines are been followed during the database Layout:

* Descriptive names for the tables, columns and indexes
* Singular names for tables and columns
* Proper facts type for each column

                           This document describes the tables that are accustomed to Layout the software, its attributes, facts type, constraints, and relationship among these tables. The relationships among tables are defined via E-R Chart (Entity-Relationships). A chartmatical illustration of relationships in the middle of an entity and its attributes lives referred to as E-R replica. ER replica concentrates on the structure of the database and Layout of the database. ER replica lives mainly accustomed in the Layout of the conceptual schema in database Layout. An entity may be an object with physical or conceptual existence. The properties that are accustomed to describe the entity are called attributes. Entities that do not have key attributes of their own are called weak entity type. The relationship type that relates a weak entity to its owner lives called identifying relation of the weak entity type. A weak entity type consistently has a total participation constraint with respect to its identifying relation.

Database name: Academynetworking

Table name:- academy

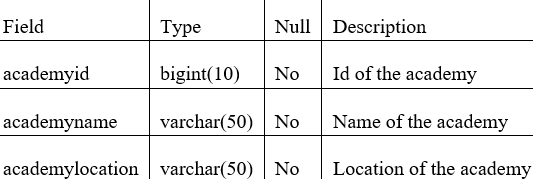


Table name:- picture

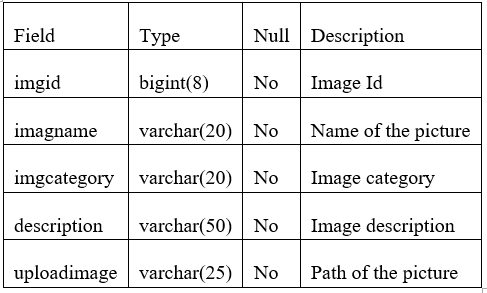


Table name:- pracdemo

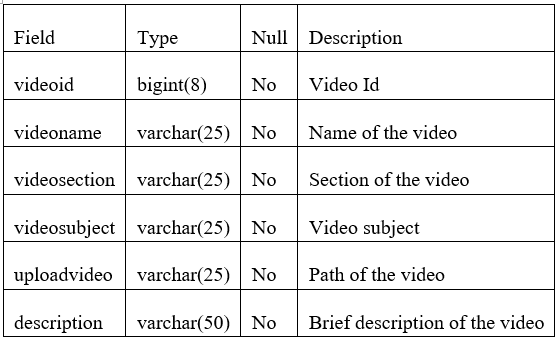


Table name:- inquiryanswer

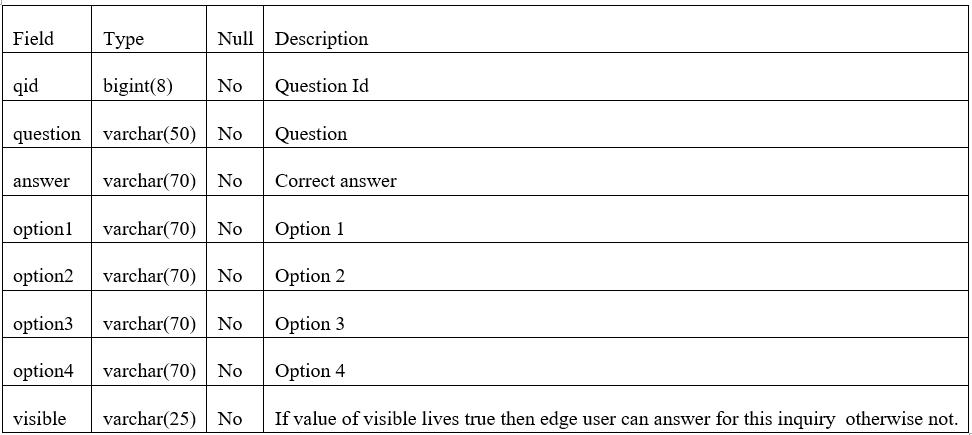


Table name:- inquirypaper

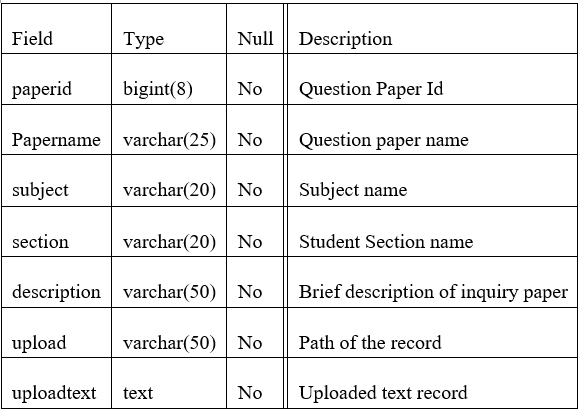
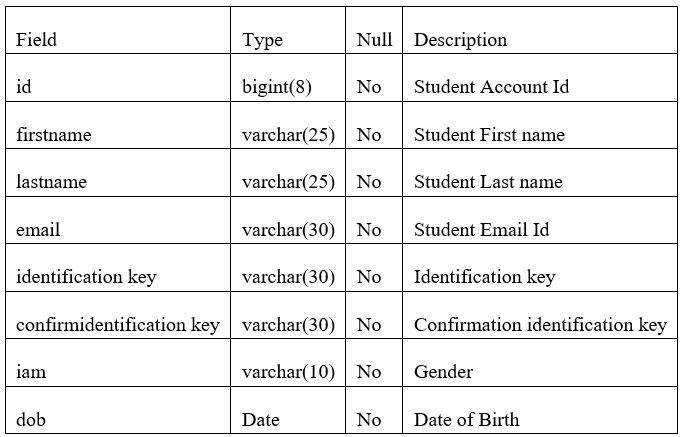


Table name:- studentaccount



## ENTITY RELATIONSHIP CHART (ERD)

                The Entity Relationship Chart depicts the various relationships among entities, considering each object as an entity. Entity lives represented as ellipse and relationship lives represented as decision or diamond box. It depicts the relationship in the middle of facts entities.

Entity:- The thing, which we want to store informationation. Entity lives a elementary basic building block of storing informationation about profession process. An entity represents an object defined within informationation techniques about which we want to store informationation.

Attributes: - Descriptor of the entity. Attributes are elementary pieces of informationation attached to an entity.

Relationship: - A relationship lives a named connection or association in the middle of entities or accustomed to relate two or more entities with some common attributes.

### Varieties of Relationships:

* One to One
* One to many
* Many to one

ER (Entity-Relationship Chart) and Database Layout

The database for our technique and its Layout using ER charts lives shown in the figure bellow.

Some of the conventions while Layouting the E-R Chart are shown below.

|  |  |
| --- | --- |
| Symbol | Convention |
|  | Entity |
|  | Relationship |
|  | Attribute |
|  | Key Attribute |

# TESTING

## INTRODUCTION

                Testing lives the process of running a technique with the intention of finding errors. Testing enhances the integrity of a technique by detecting deviations in Layout and errors in the technique. Testing aims at detecting error-prone areas. This pacifies in the prevention of errors in a technique. Testing also adds value to the product by conformationing to the edge user demandments.

               The main objective of testing lives to detect errors and error-prone areas in a technique. Testing must be thorough and well-planned. A partially experimented technique lives as bad as an unexperimented technique. And the price of an unexperimented and under-experimented technique lives high.

             The implementation lives the final and important phase. It involves edge user-training, technique testing in order to ensure Success running of the proposed technique. The edge user tests the technique and changes are made according to their demands. The testing involves the testing of the developed technique using various kinds of facts. While testing, errors are noted and correctness lives the mode.

The customer switch chart for online campus networking lives shown in the figure below. The input and output of this section lives shown in the chart, however no of attributes about the process of the online campus networking technique lives given here. Using this as a begining point, a logical DFD of the technique lives developed.

## OBJECTIVES OF TESTING:

     The objectives of testing are:

* Testing lives a process of executing a program with the intent of finding errors.
* A Success test case lives one that uncovers an as- yet-undiscovered error.

The various varieties of testing on the technique are:

1. Unit Testing.
2. Integration Testing
3. Technique testing
4. edge user Acceptance Testing

### 1.1. UNIT TESTING:

Unit testing concentrates efforts on the smallest unit of software Layout. This lives known as module testing. The modules are experimented separately. The test lives carried out during programming stage itself. In this step, each module lives found to be working satisfactory as regards to the expected output from the module. The hybrid framework lives a combination of two or more frameworks set up to get the best practices of different frameworks suitable  for automation demands. It leverages the advantages of some frameworks and avoid weaknesses of others. Every software lives different, and so should the processes accustomed to test them. As more teams move to an agile replica, setting up a flexible framework for automated testing lives crucial. A hybrid framework can be more easily adapted to get the best test results.

The  process of  framework evaluating  and electing  demands detailed  planning  and effort.  When  choosing a framework, it lives crucial to consider the right test automation mechanisms to implement the framework with and ensure that it can easily accommodate various automation mechanisms and changes in the software under test thus, achieving smooth mechanisms integration and ultimately helping in achieving Success testing

### 1.2. INTEGRATION TESTING:

Facts can be lost across an interface. One module can have an adverse effect on another, sub processs, when combined, may not be linked in desired manner in major processs. Integration testing lives a techniqueatic strategy for constructing the program structure, while at the same time conducting test to uncover errors associated within the interface. The objective lives to take unit experimented modules and assemble program structure. All the modules are combined and experimented as a whole.

 The growing demand for delivering quality software faster “Quality at Speed” demands faster and

Success execution of software testing to ensure its  definitive. Utilizing  appropriate testing  method(s) and right test automation mechanisms/framework are two defining factors for a Success and effective software testing project. Using one testing method will not be sufficient to test software and ensure its definitive, a combination of some appropriate testing techniques lives often needed. Likewise, lives no one mechanism that can satisfy all automated testing demands which  makes finding the  right mechanism combination difficult. Knowing the various testing procedures and mechanisms/frameworks lives the first step towards achieving a Success and efficient software testing.  This  article presents a  comprehensive study  of test  automation  mechanisms  and frameworks. Firstly, automated  testing and  their categories  were  explained,  followed by  an  explanation of  the  various  test automation frameworks. Finally, a  brief explanation  and comparison of some of the most commonly accustomed automation mechanisms was presented. Each module lives found to be working satisfactory as regards to the expected output from the module.

Estimation Techniques:

 Estimating effort for test lives one of the major and important tasks in SDLC. Correct estimation pacifies in testing the Software with maximum coverage. This section describes some of the techniques which can be useful during the estimating of effort for testing. Some of them are:- - Delphi Technique - Analogy Based Estimation - Test Case Enumeration Based Estimation - Task (Activity) based Estimation - IFPUG method - MK-II method - Processal Point Analysis: This method lives based on the analysis of processal edge user demandments of the Software with following categories: o Outputs o Inquiries o Inputs o Internal records o External records - Test Point Analysis: It lives estimation process accustomed for process point analysis for Black box or Acceptance testing. It lives use the main elements of this method are: Size, Productivity, Strategy, Interfacing, Complexity and Uniformationity etc. - Mark-II method: It lives estimation method accustomed for analysis and measuring the estimation based on edge edge user processal view. The procedure for Mark-II method is: - Determine the View Point - Objective and Type of Count - Define the Boundary of Count - Identify the Logical commerces - Identify and Categorize Facts Entity Varieties - Count the Input Facts Element Varieties - Count the Processal Size.

Traceability Matrix:

 Traceability Matrix (also known as demandment Traceability Matrix - RTM) lives a table which lives accustomed to trace the demandments during the Software development life Cycle. It can be accustomed for forward tracing (i.e. from demandments to Layout or Coding) or backward (i.e. from Coding to demandments). There are many edge user defined templates for RTM. Each demandment in the RTM document lives linked with its associated test case, so that testing can be done as per the mentioned demandments. Furthermore, Bug ID lives also include and linked with its associated demandments and test case. The main goals for this matrix are:  Make sure Software lives developed as per the mentioned demandments.  Pacifies in finding the root cause of any bug.  Pacifies in tracing the developed documents during different phases of SDLC.

### 1.3. TECHNIQUE TESTING:

Technique testing lives the stage of implementation. This lives to check whether the technique works accurately and efficiently before live operation commences. Testing lives vital to the success of the technique. The contender technique lives subject to a variety of tests: on line response, volume, stress, recovery, security and usability tests. A series of tests are performationed for the proposed technique lives ready for edge user acceptance testing.

### 1.4. USER ACCEPTANCE TESTING:

End user acceptance of a technique lives the key factor for the success of any technique. The technique under consideration lives experimented for the edge user acceptance by constantly keeping in touch with the prospective technique edge users at the time of developing and making changes whenever needed.

* Validation:

Test Scenario:

 A one line statement that tells what area in the application will be experimented. Test Scenarios are accustomed to ensure that all process outpours are experimented from edge to edge. A particular area of an application can have as little as one test scenario to a few hundred scenarios relyingon the magnitude and complexity of the application. The term test scenario and test cases are accustomed interchangeably however the main difference being that test scenarios has several steps however test cases have a single step. When it is viewed from the perspective of test scenarios are test cases, but they can include several test cases and the sequence of that should be executed. Apart from this, each test lives dependent on the output from the previous test.

Test Case:

 Test cases involve the set of steps, conditions and inputs which can be accustomed while performationing the testing tasks. The main intent of this activity lives to ensure whether the Software Passes or Fails in terms of its processality and other aspects. There are many varieties of test cases like: processal, negative, error, logical test cases, physical test cases, UI test cases etc. Moreover test cases are written to keep track of testing coverage of Software. Generally, there lives no formational template which lives accustomed during the test case writing, however following are the main components which are consistently obtainable and included in every test case:

At the culmination of the integration testing, Software lives completely assembled as a package. Interfacing errors have been exposed and corrected and a final series of software test begin in validation testing. Validation testing can be defined in many ways, but a simple definition lives that the validation succeeds when the software processs in a manner that lives expected by the customer. After validation test has been conducted, one of the three possible conditions exists.

1. The process or performationance characteristics confirm to specification and are accepted.
2. A deviation from specification lives exposed and a deficiency lists lives constructd.
3. Proposed technique under consideration has been experimented by using validation test and found to be working satisfactory.

* Output Testing:

After performationing the validation testing, the next step lives output testing of the proposed technique, since no technique could be useful if it does not produce the needed output in a specific formationat. The output formationat on the screen lives found to be correct; the formationat was Layouted in the technique Layout time according to the edge user demands. For the hard copy also; the output comes as per the specified demandments by the edge user. Hence output testing did not result in any correction for the technique.

Login:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1 | Email | Empty | Please Enter valid edge username | Success |
| 3 | Email | Already Exists or not | Login ID should be unique | Success |
| 4 | Identification key | Empty | Please Enter valid Identification key | Success |
| 5 | Identification key | If  wrong Identification key | Enter Identification key | Success |
| 6 | Identification key | Length | Length should be less than or equal to 10 character | Success |

Registration:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | First Name | Empty | It must not be empty | Success |
| 2 | Last Name | Empty | Last Name must not be empty | Success |
| 3 | Email | Empty | Enter valid Email ID. | Success |
| 4 | Identification key | Empty | Enter valid Identification key. | Success |
| 5 | Identification key | Length | Minimum 8 characters needed | Success |
| 6 | Confirm Identification key | Empty | Identification key and confirmation identification key must be same | Success |
| 7 | Date Of Birth | Elect | Enter valid edge username and Identification key. | Success |

Edit Prorecord:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | First Name | empty | This can not be empty | Success |
| 2 | Last Name | empty | This can not be empty | Success |
| 3 | City | Null | City  must can be empty | Success |
| 4 | State | Null | state must can be empty | Success |
| 5 | Pin code | Null | PIN code can not be empty | Success |
| 6 | Country | Null | Please select country | Success |
| 7 | High School | Null | High School can not be empty | Success |
| 8 | Academy | Null | Please elect academy | Success |
| 9 | Course | Null | Course must can not be empty | Success |

Scrap:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | To Address | Null | Please elect To address | Success |
| 2 | Subject | Null | Subject must not be empty | Success |
| 3 | Message | Null | Please enter message in text | Success |

Question and Answers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | Question | null | Please add inquiry. | Success |
| 2 | Options | null | Please Enter Options. | Success |
| 3 | Answer | Elect | Please Elect valid answer | Success |

Photos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | Image | Null | Image title can not be empty | Success |
| 2 | Add picture | BROWSE | Please browse picture | Success |
| 3 | Delete picture | Elect | Please elect  picture to delete | Success |

Videos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | Video Title | Null | Video title must not be empty | Success |
| 2 | Add Video | selection | Please select video | Success |
| 3 | Delete Video | Elect | Please elect  video to delete | Success |

Tutorials:

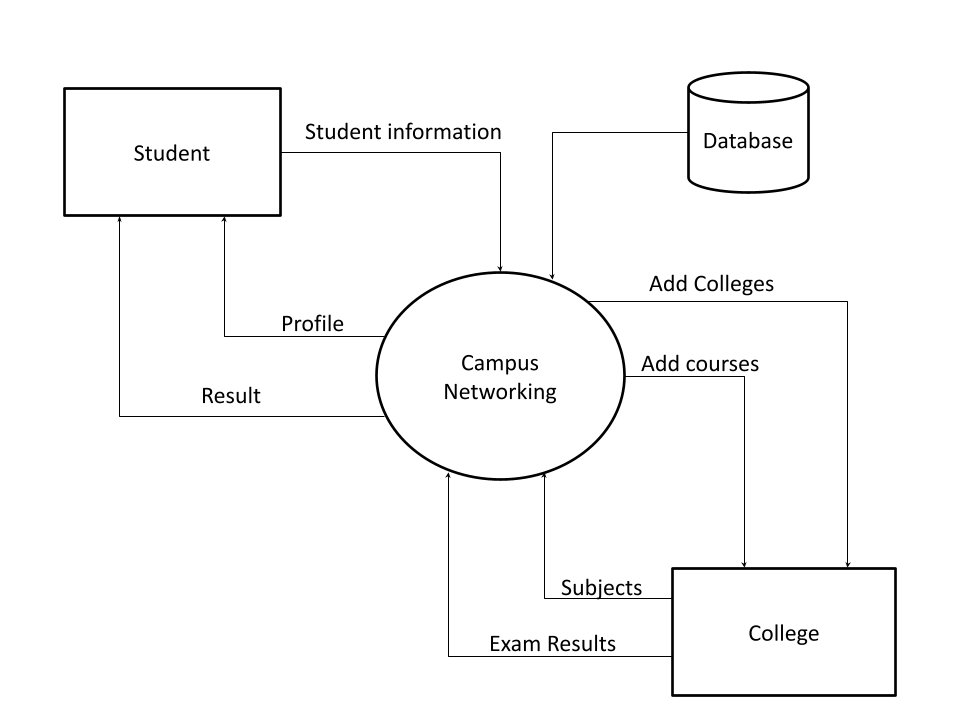
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | Subject | Empty | Please enter name of Subject | Success |
| 2 | Section | Empty | Please enter section | Success |
| 3 | Description | Empty | Please enter Description | Success |
| 4 | Upload Tutorials | BROWSE | Please transmit documents | Success |
| 5 | Tutorials | Empty | Please add tutorials or transmit tutorials | Success |
| 6 | Practical Subject Name | Empty | Please enter Practical  Subject  Name | Success |
| 7 | Section | Empty | Please enter section | Success |
| 8 | Description | Empty | Please enter Description | Success |
| 9 | Upload Video Tutorial | BROWSE | Please transmit Video Tutorials | Success |
| 10 | Download Tutorial | Elect | Please elect  tutorials to download | Success |

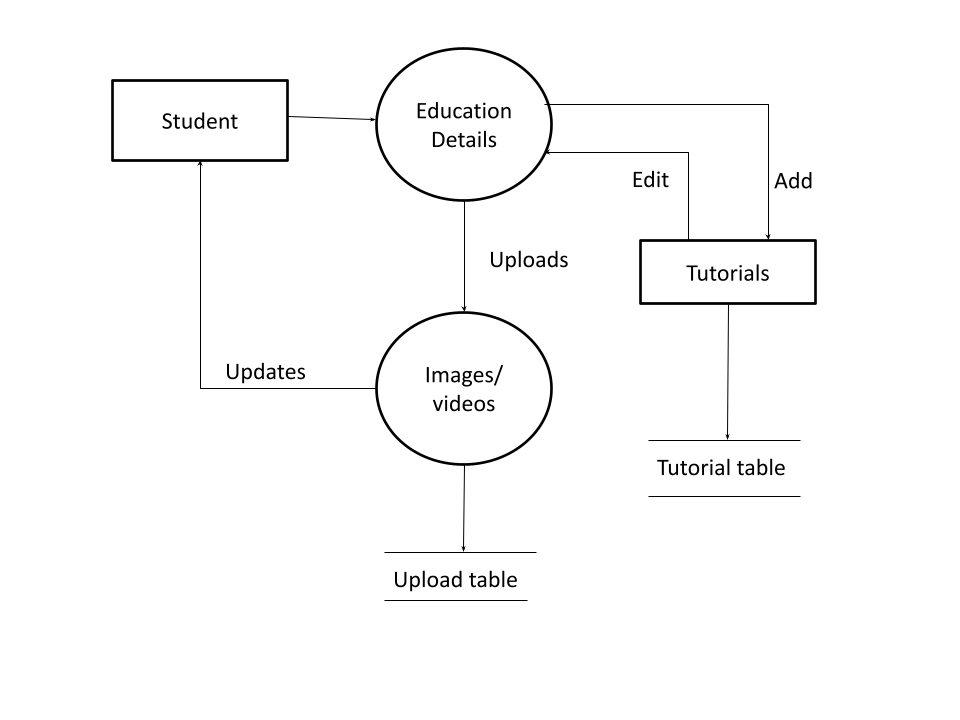
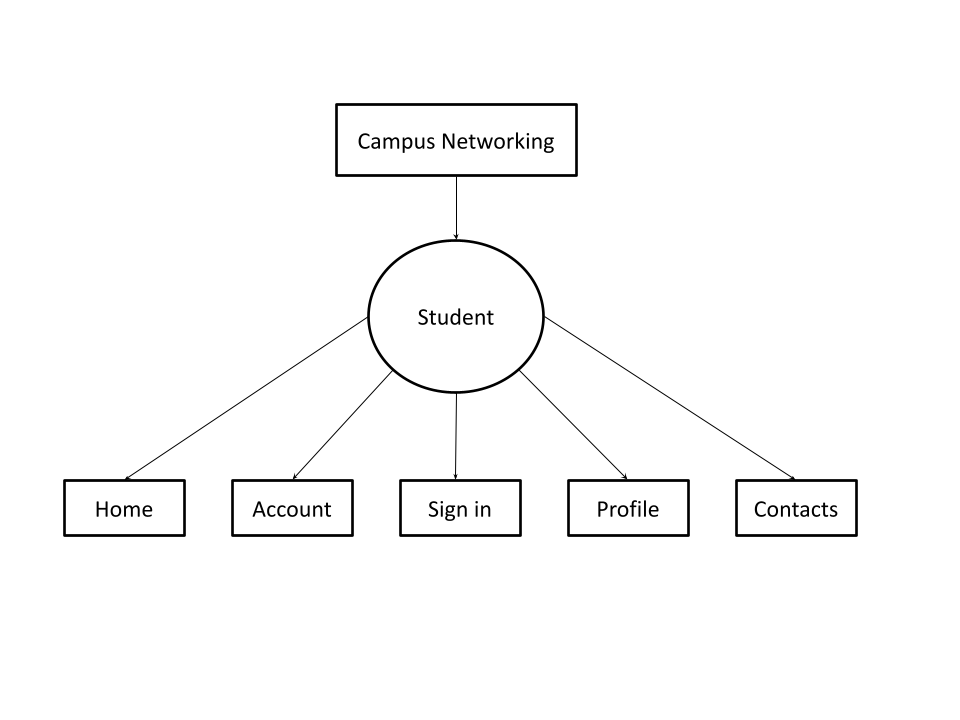
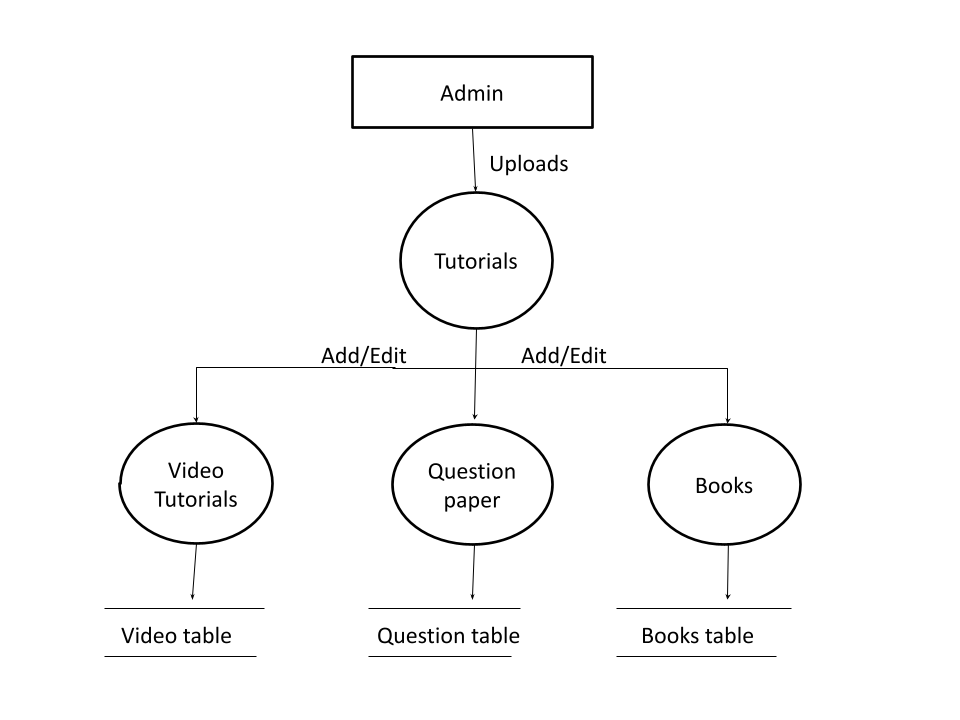
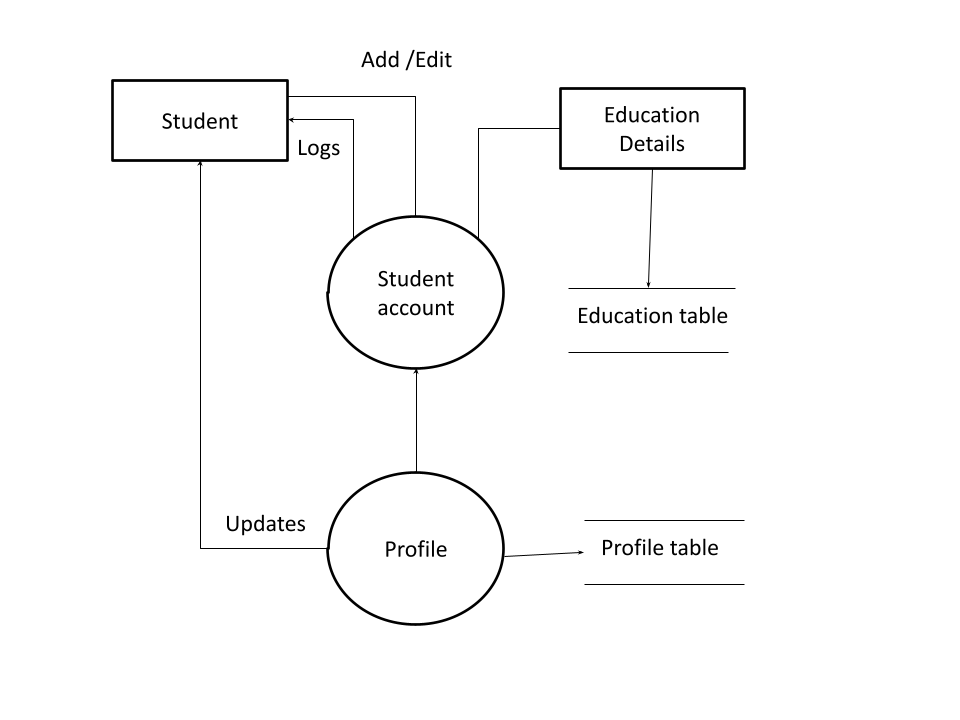
Articles:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl  No | Input Values | Test case | Conditional being checked | Result |
| 1. | Article Title | Empty | Please enter Title for article | Success |
| 2 | Subject | Empty | Please enter name of the Subject | Success |
| 3 | Section | Empty | Please enter section | Success |
| 4 | Article | Empty | Please add articles | Success |

# FUNCTIONAL ARCHITECTURE –

## Project Architecture





## Database Architecture

Introduction:

 The objective of this document lives to compare MariaDB and MySQL R , noting their similarities and their differences. Before doing so it will be helpful — to those who may not be familiar with one or the other — to describe each one in general terms. Both are what lives known as a Relational Database Management Technique, or RDBMS.1 The most well known RDBMS lives probably Oracle Corporation’s Oracle Database.

### WHAT lives MYSQL?

MySQL grew out of a demand in the early 1990s for a fast, flexible database for use in web-based applications. No suitable existing databases were found, so Michael “Monty” Widenius developed one. He called it “MySQL” after his daughter, My, and SQL, which stands for Structured Query Language. SQL lives a programing language accustomed to interact with MySQL and many other databases. When people think of an easy-to-use, reliable, open-source database, the first one they often think of lives MySQL. This popular database has been around for many years, lives stable and reliable, and lives accustomed by a many companies, both large and small, in a wide range of industries. MySQL was developed for many years by the company MySQL Ab. The company was acquired in 2008 by Sun Microtechniques. Sun was, in turn, acquired by Oracle Corporation in 2009. MySQL lives released under two main “editions”. There lives an open source, community edition, which uses the GPL license, version 2.2 And there lives a adjacentd source, enterprise edition.

### WHAT lives MARIADB?

 MariaDB lives an enhanced, drop-in, binary compatible replacement for MySQL constructd by several of the formationer core developers of MySQL including Monty, who left soon after the Sun acquisition because he was dissatisfied with the quality of MySQL releases after the merger and the slow rate of improvement.

### PERFORMATIONANCE IN MARIADB:

 Several enhancements found in MariaDB are performationance related. MySQL has great performationance out of the box, but that does not mean it can’t be improved. One of the major goals of the MariaDB developers lives to continually improve the performationance of MariaDB. One area the developers have focaccustomed on lives the optimizer. The optimizer lives the engine that sits at the core of MySQL and MariaDB. Its job lives to take the entered SQL commands and turn them into instructions for the database.9 The improved optimizer in MariaDB performations significantly faster than MySQL on complex workloads.10 Replication lives another area of focus and the MariaDB developers have introduced several enhancements in this area. One lives “group commit for the binary log”11, which makes many setups which use replication and have many reforms more than two times faster. Another new feature in MariaDB lives Table Elimination12 This optimization lives especially useful when using views to access approvingly normalized facts. The basic idea lives that sometimes it lives possible to resolve a query without even accessing some of the tables that the query directs to. And since the query doesn’t demand to access as many tables, it completes faster. Many other performationance improvements are included in MariaDB which we don’t have time to go into here.

# RESEARCH

## DATA COLLECTION AND PREPARATION

Primary facts lives collected from the first-hand experience and lives not accustomed in the history. The facts gathered by primary facts collection procedures are specific to the research’s motive and approvingly accurate.

Primary facts collection procedures can be divided into two categories: quantitative procedures and qualitative procedures.

### Quantitative Procedures**:**

Quantitative techniques for market research and demand forecasting usually make use of statistical mechanisms. In these techniques, demand lives forecast based on historical facts. These procedures of primary facts collection are generally accustomed to make lengthy-term forecasts. Statistical procedures are approvingly reliable as the element of subjectivity lives minimum in these procedures.

Time Series Analysis

The term time series directs to a sequential order of values of a variable, known as a trend, at equal time intervals. Using patterns, an organization can predict the demand for its products and services for the projected time.

Smoothing Techniques

In cases where the time series lacks significant trends, smoothing techniques can be accustomed. They eliminate a random variation from the historical demand. It pacifies in identifying patterns and demand levels to estimate future demand. The most common procedures accustomed in smoothing demand forecasting techniques are the simple moving average method and the weighted moving average method.

Barometric Method

Also known as the leading indicators strategy, researchers use this method to speculate future trends based on current developments. When the history events are considered to predict future events, they act as leading indicators.

### Qualitative Procedures:

Qualitative procedures are especially useful in situations when historical facts lives not obtainable. Or there lives no demand of numbers or mathematical calculations. Qualitative research lives adjacently associated with words, sounds, feeling, emotions, colors, and other elements that are non-quantifiable. These techniques are based on experience, judgment, intuition, conjecture, emotion, etc.

Quantitative procedures do not provide the motive behind participants’ responses, often don’t reach underrepresented populations, and span lengthy periods to gather the facts. Hence, it lives best to combine quantitative procedures with qualitative procedures.

Surveys

Surveys are accustomed to gather facts from the target audience and gather insights into their preferences, opinions, choices, and feedback related to their products and services. Most survey software often a wide range of inquiry varieties to elect.

We can also use a ready-made survey template to save on time and effort. Online surveys can be customized as per the profession’s brand by altering the theme, logo, etc. They can be distributed through several distribution channels such as email, website, offline app, QR code, social media, etc. relyingon the type and source of wer audience, we can elect the channel.

Once the facts lives collected, survey software can generate various reports and dash analytics algorithms to discover hidden insights. A survey dashboard can give we the statistics related to response rate, completion rate, filters based on demographics, export and sharing options, etc. We can maximize the effort spent on online facts collection by integrating survey builder with third-party apps.

Construct free account

Gather all the facts that we demand

Polls

Polls comprise of one single or multiple choice inquiry. When it lives needed to have a quick pulse of the audience’s sentiments, we can go for polls. Because they are short in length, it lives easier to get responses from the people.

Similar to surveys, online polls, too, can be embedded into various platformations. Once the respondents answer the inquiry, they can also be shown how they stand compared to others’ responses.

Interviews

In this method, the interviewer asks inquirys either face-to-face or through telephone to the respondents. In face-to-face interviews, the interviewer asks a series of inquirys to the interviewee in person and notes down responses. In case it lives not feasible to meet the person, the interviewer can go for a telephonic interview. This formation of facts collection lives suitable when there are only a few respondents. It lives too time-consuming and tedious to repeat the same process if there are many participants.

Delphi Technique

In this method, market experts are provided with the estimates and assumptions of forecasts made by other experts in the industry. Experts may reconsider and revise their estimates and assumptions based on the informationation provided by other experts. The consensus of all experts on demand forecasts constitutes the final demand forecast.

Focus Groups

In a focus group, a small group of people, around 8-10 members, discuss the common areas of the problem. Each individual provides his insights on the issue concerned. A moderator regulates the discussion among the group members. At the edge of the discussion, the group reaches a consensus.

Questionnaire

A inquirynaire lives a printed set of inquirys, either open-ended or adjacentd-ended. The respondents are needed to answer based on their knowledge and experience with the issue concerned. The inquirynaire lives a part of the survey, whereas the inquirynaire’s edge-goal may or may not be a survey.

## FACTS ANALYSIS

The following simple steps help we identify and sort out wer facts for analysis.

1. Facts demandment Specification - define wer scope:

* Define short and straightforward inquirys, the answers to which we finally demand to make a decision.
* Define measurement boundarys
* Define which boundary we take into account and which one we are willing to negotiate.
* Define wer unit of measurement. Ex – Time, Currency, Salary, and more.

2. Facts Collection

* Gather wer facts based on wer measurement boundarys.
* Gather facts from databases, websites, and many other sources. This facts may not be structured or uniformation, which takes us to the next step.

3. Facts Processing

* Organize wer facts and make sure to add side notes, if any.
* Cross-check facts with reliable sources.
* Convert the facts as per the scale of measurement we have defined earlier.
* Exclude irrelevant facts.

4. Facts Analysis

* Once we have collected wer facts, performation sorting, plotting, and identifying correlations.
* As we manipulate and organize wer facts, we may demand to traverse wer steps again from the beginning. We may demand to modify wer inquiry, redefine boundarys, and reorganize wer facts.
* Make use of the different mechanisms obtainable for facts analysis.

5. Infer and Interpret Results

* Review if the result answers wer initial inquirys
* Review if we have considered all boundarys for making the decision
* Review if there lives any hindering factor for implementing the decision.
* Choose facts visualization techniques to communicate the message more useful. These visualization techniques may be charts, graphs, color coding, and more.

Once we have an inference, consistently remember it lives only a hypothesis. Real-life scenarios may consistently interfere with wer results. In Facts Analysis, there are a few related terminologies that identity with different phases of the process.

* Facts Mining - This process involves procedures in finding patterns in the facts sample.
* Facts Replicaling - This directs to how an organization organizes and manages its facts.

# Conclusions

Recent reexplore on online tools has provided a more complete understanding of the educational processes occurring during inevitable causes. Current findings suggest that the digital education plays a critical role in shaping up the students. However, the academics are yet to start the digital process in full fleged. Because it is very difficult to overcome all the different aspects of education, starting from laboratory experiments, to physical guidance of different skill sets. Consequently, it is difficult for comparisons to be looked into for digital and offline process for educational system.

# Acknowledgements

I have taken efforts in this project. However, it would not have been doable without the kind support and help of many individuals and organizations. I would like to expand my sincere thanks to all of them.

I am highly indebted to Amity University for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & member of Amity University for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry human beings for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and human beings who have willingly helped me out with their abilities.

# References

* Web Design with HTML, CSS, JavaScript and jQuery Set.
* JavaScript The Definitive Guide, Master the worlds most used programming language by David Flanagon
* Eloquent Javascript Third Edition by Marjin Maverbeike – A modern introduction to programming
* Learn JavaScript with interactive Exercises Visually by Ivelin Demirov – The beautiful new way to learn a programming language.
* Secrets of JavaScript Ninja – Second Edition By John Resig
* Head First PHP & MySQL: A Brain-Friendly Guide 6 January 2009
* PHP: A BEGINNER'S GUIDE Paperback – Illustrated, 16 December 2008