# Content Management System

## Table of Contents

# 1 Introduction

## 1.1 Defination

A Content Management System or CMS is software that is used for creating, managing and editing a website even if you do not have any specialized technical skills or knowledge.

With CMS, you can develop and design a website without even having to write the code. In short, a CMS website is a website that is developed by using a content management system.

CMS not only helps in managing the text or images which are being displayed on the website, but they also help in tracking the user sessions, handling the search queries, collecting visitor feedback and

comments, etc.

## 1.2 Features

- Content creation (allows users to easily create and format content)
- Content storage (stores content in one place, in a consistent fashion)
- Content storage (stores content in one place, in a consistent fashion)
- Publishing (organizes and pushes content live)

## 1.3 Purpose

- **Increase efficiency** — Content can be published easily and efficiently as editing and revisions do not require visual design or coding knowledge. This allows for fast and efficient updates, saving your business cost and time.
- **Increase your search engine ranking** — To improve or maintain your search engine ranking your business has to remain relevant, and a good and easy-to-use CMS will help your publishers keep the content fresh. This invites external contribution (e.g.

comments, forum, likes etc…), an integral component in staying relevant and improving your search engine ranking.

- **Maintain control over your content** - Workflow is a core feature of any good CMS. Irrespective of how basic your workflow needs might be, workflows will ensure your business maintains control over content.
- **Help your visitors in their search for information** — With powerful CMS search engines new content is indexed automatically so it can be instantly found. Visitors can also use taxonomy applications, sorting lists, saved searches and more to personalize the search experience.
- **Cross-selling** — Sophisticated Content Management Systems can learn user behavior and preferences, making your cross-selling and up-selling efforts much more cost effective.

## 1.4 Benefits

- One major advantage of a CMS is its collaborative nature. Multiple users can log on and contribute, schedule or edit content to be published. Because the interface is usually browser-based, a

CMS can be accessed from anywhere by any number of users.

- he second major advantage of a CMS is that it allows non-technical people who don't know programming languages to easily create and manage their own web content. The WYSIWYG editors of a typical content management platform allows users to enter text and upload images without needing to know any HTML or CSS.
- When a company uses a CMS to publish its pages, it reduces its reliance on front-end engineers to make changes to the website, making it quicker and easier to publish new content.

## 1.5 Examples

- Episerver
- Drupal
- Wordpress
- Joomla
- Magenta
- ModX

- Squarespace
- Wix
- Weebly

## 1.6 Stakeholders

- Consumers or General Audience
- Individuals
- Educators
- Enterprise
- Publishers
- Freelancers
- Web Developers
- Authors
- Bloggers
- Small Business Owners

## 1.7 Scope

In this digital world, content management systems will go far away from the mark and bring the latest revolution, for sure that is very helpful for the internet. So, if we talk about CMS, it helps create, publish and update website content very effectively.

Nowadays technology is at its top. And it is very important we have to upgrade our website, from time to time.

- Quick Implementation
- User Experience
- Virtual and Augmented Reality
- Ease of Use
- Scalability
- Flexibility
- Ongoing Support

## 1.8 Goals and Objectives

### 1.8.1 Support marketing

Websites have become a key marketing channel for businesses. The CMS should facilitate the delivery of marketing material, as well as supporting current brands and corporate identity.

### 1.8.2 Reduce duplication of information

Duplication of information across business units and platforms increases maintenance costs and error rates. Wherever possible, information should be stored once, and reused multiple times.

### 1.8.3 Improve customer experience

The CMS should enhance the ability to provide a full-featured, rich environment for website visitors. This will include enhancements to the quality of the site, and the ease of use.

### 1.8.4 Support sales

E-commerce sales are growing steadily, and the CMS should provide further sales material to enhance the sales impact of the website. It

should also complement current e-commerce infrastructure.

### 1.8.5 Support website growth

There is a strategic need to substantially increase the amount of information published on corporate websites. The CMS should deliver efficiency and management benefits to support the goal of website growth.

### 1.8.6 Improve business responsiveness

The CMS should support the development of new products and services, as well as other changes in corporate direction. This is achieved by providing a rapid and efficient mechanism to update internal corporate information and resources.

### 1.8.7 Improve publishing process

Ad-hoc publishing processes prevent effective management and tracking. The CMS should improve on this, as well as providing greater transparency and accountability.

### 1.8.8 Capture business knowledge

All information presented on the corporate website exposes the business

to legal liability. This should be reduced by establishing greater control and accountability over the review and publishing processes.

### 1.8.9 Improve staff efficiency

Staff efficiency can be improved by supporting key business processes with sufficient information. The CMS must aim to provide staff with the information they need, when they need it.

This will translate into direct time savings by avoiding fruitless searches for required information.

### 1.8.10 Reduce publishing costs

Many business manuals are still produced in paper form. Direct cost savings would be realised by replacing these with online resources.

### 1.8.11 Reduce customer support costs

Customer support requirements should be reduced, by providing more accurate and comprehensive information to customers.

### 1.8.12 Reduce website maintenance costs

By replacing the current labour-intensive maintenance activities, the CMS should reduce the need for website administration staff, and other associated costs.

### 1.8.13 Increase website audience

The CMS should allow a wide audience to access the corporate website. All customers of the business will then benefit from the site. (This includes non-English speakers.)

### 1.8.14 Improve information accuracy

The overall quality of the information is to be improved, on both the website and intranet. All pages should be:

- Accurate
- Up-to-date
- Comprehensive

### 1.8.15 Business-specific goals

- Beyond the general CMS goals outlined above, there will be a

number of very specific goals that relate directly to your business.

- Identifying these ensures that the CMS matches the unique nature of your business, and that it meets the overall strategic direction.
- Examine the key products, services and processes within the business to identify these business-specific goals.

Without business goals, success is impossible to measure

# 2 Supported Platforms & Architectures

## 2.1 Supported architectures

- Intel x86 (32-bit) and x86_64 (64-bit, aka AMD64)

## 2.2 Well supported platforms

- Linux
- Windows Vista and newer
- FreeBSD 10 and newer
- macOS Snow Leopard (macOS 10.6, 2008) and newer

- Android 4.4 and newer

# 3 Databases

We officially supports the following databases:

## 3.1 Relational Databases

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

## 3.2 Non - Relational Databases

- MongoDB

# 4 Prerequisites Softwares

## 4.1 Well supported Browsers

- Google Chrome
- Firefox
- Safari
- Opera
- Brave

### 4.1.1 Browser Must Support

- HTML
- Javascript
- CGI

# 5 Logical Strcuture Design

## 5.1 Classes

- **User class :** Manages all the operations of user.

- **Role class :** Manages all the roles of user.
- **Permission class :** Manage all the operation of permission assigned to each user.
- **Blog class :** Manages all the operations on Blog.
- **Comment class :** Manages all the operation of comment on each Blog.
- **Content class :** Manages the content of each Blog.
- **Viewer class :** Manages all the operation for each Blog viewed by each User.

## 5.2 Classes and thier Attributes

- **User Attributes :** user_id, user_role_id, user_name, user_email, user_dob, user_ address
- **Role Attributes :** role_id, role_title, role_description
- **Permission Attributes :** permission_id, permission_role, permission_title, permission_module, permission_description
- **Blogs Attributes :** blog_id, blog_user_id, blog_type, blog_content, blog_description

- **Content Attributes :** content_id, content_blog_id, content_title, content_type, content_description
- **Comment Attributes :** comment_id, comment_user_id, comment_type, comment_title, comment_description
- **Viewer Attributes :** viewer_id, viewer_blog_id. viewer_user_id

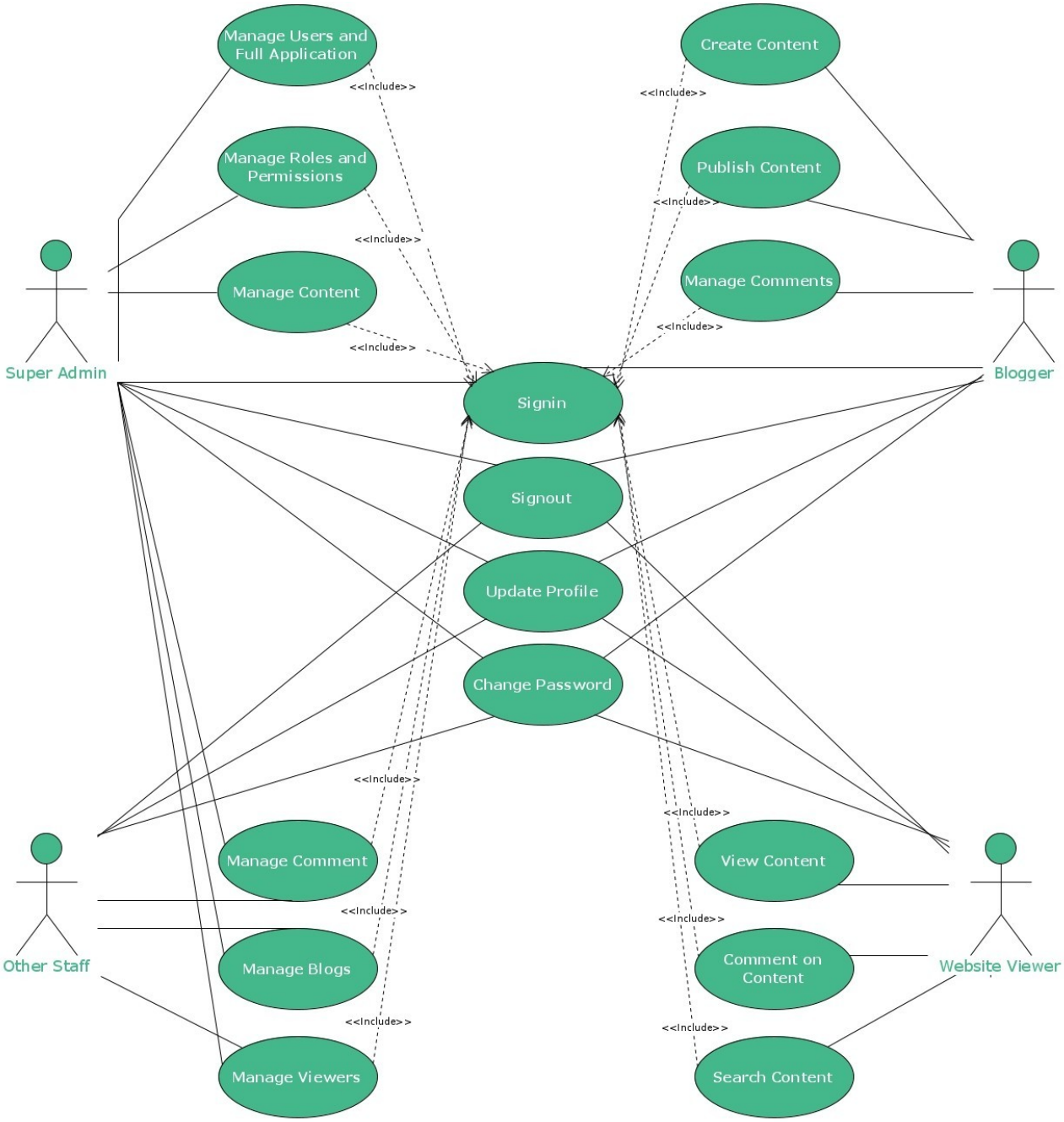
## 5.3 Classes and thier Methods

- **User :** addUser(), editUser(), deleteUser(), searchUser()
- **Role :** addRole(), editRole(), deleteRole(), searchRole()
- **Permission :** addPermission(), editPermission(), deletePermission(), searchPermission()
- **Blogs Methods :** addBlogs(), editBlogs(), deleteBlogs(), searchBlogs()
- **Content Methods :** addContent(), editContent(), deleteContent(), searchContent0
- **Comment Methods :** addComment(), editComment(), deleteComment(), searchComment()
- **Viewer Methods :** addViewer(), editViewer(), deleteViewer(),

searchViewer()

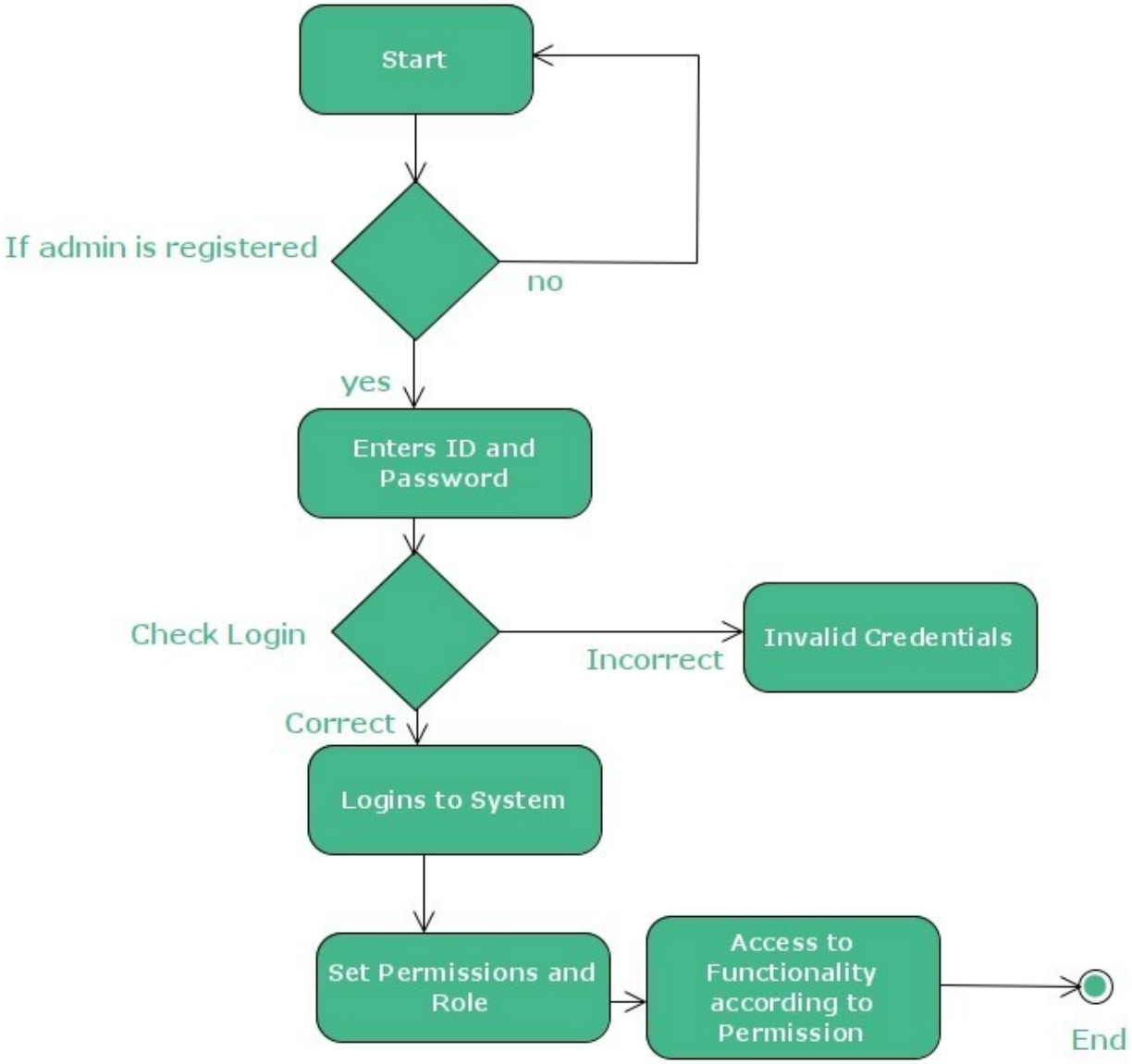
# 6 Diagrams

## 6.1 Use Case Diagram

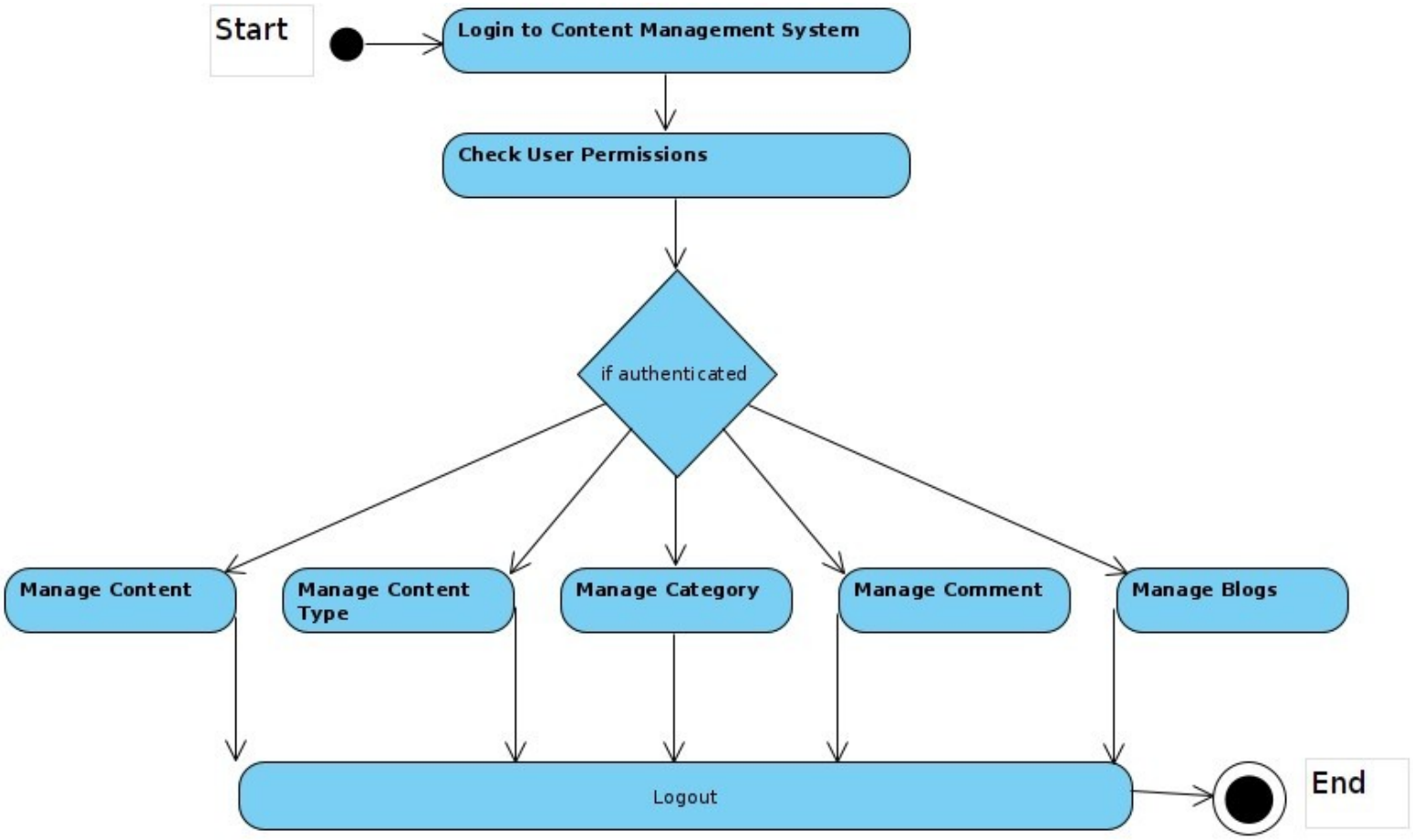


# 6.2 Activity Diagram

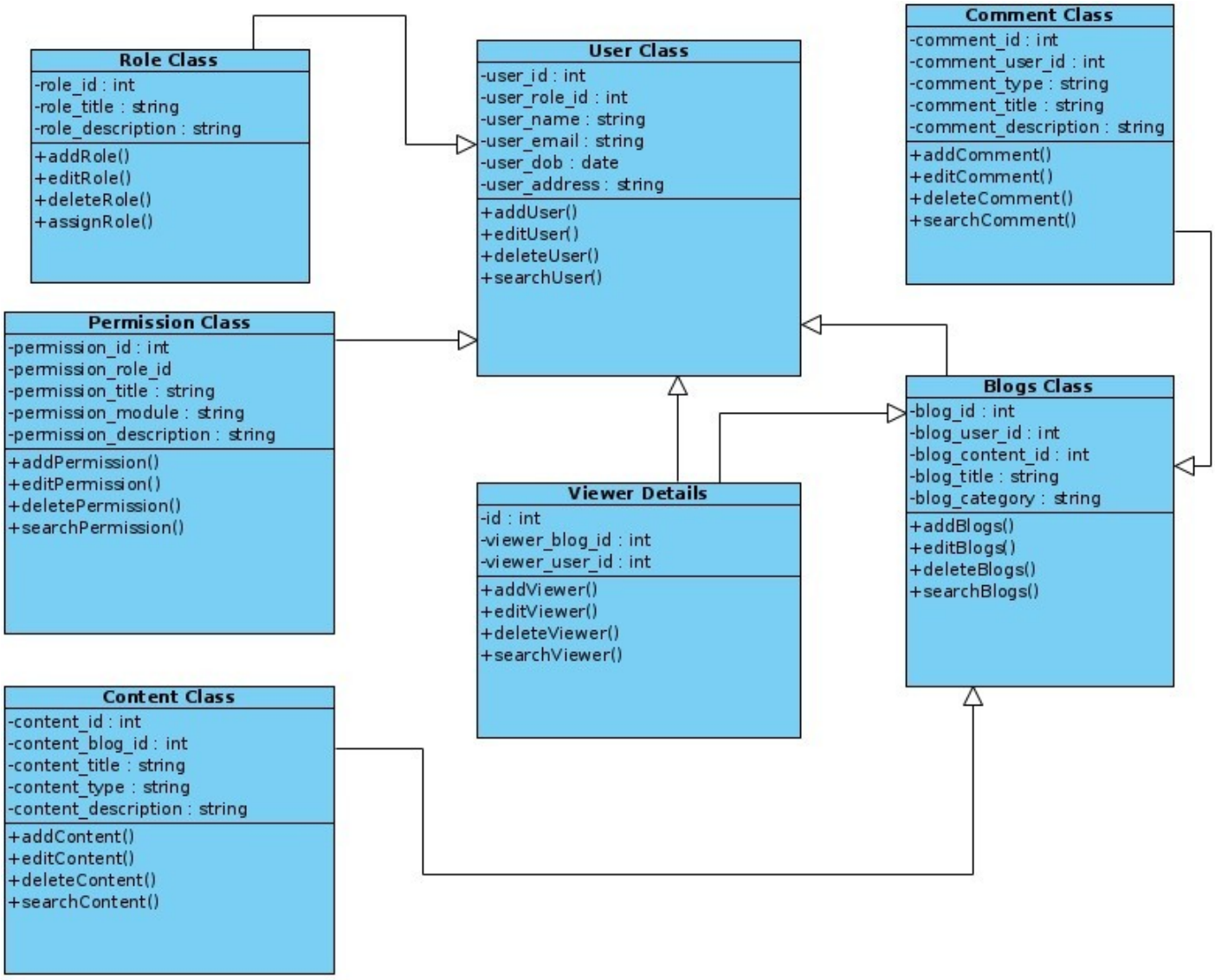
## 6.2.1 Login Activity Diagram



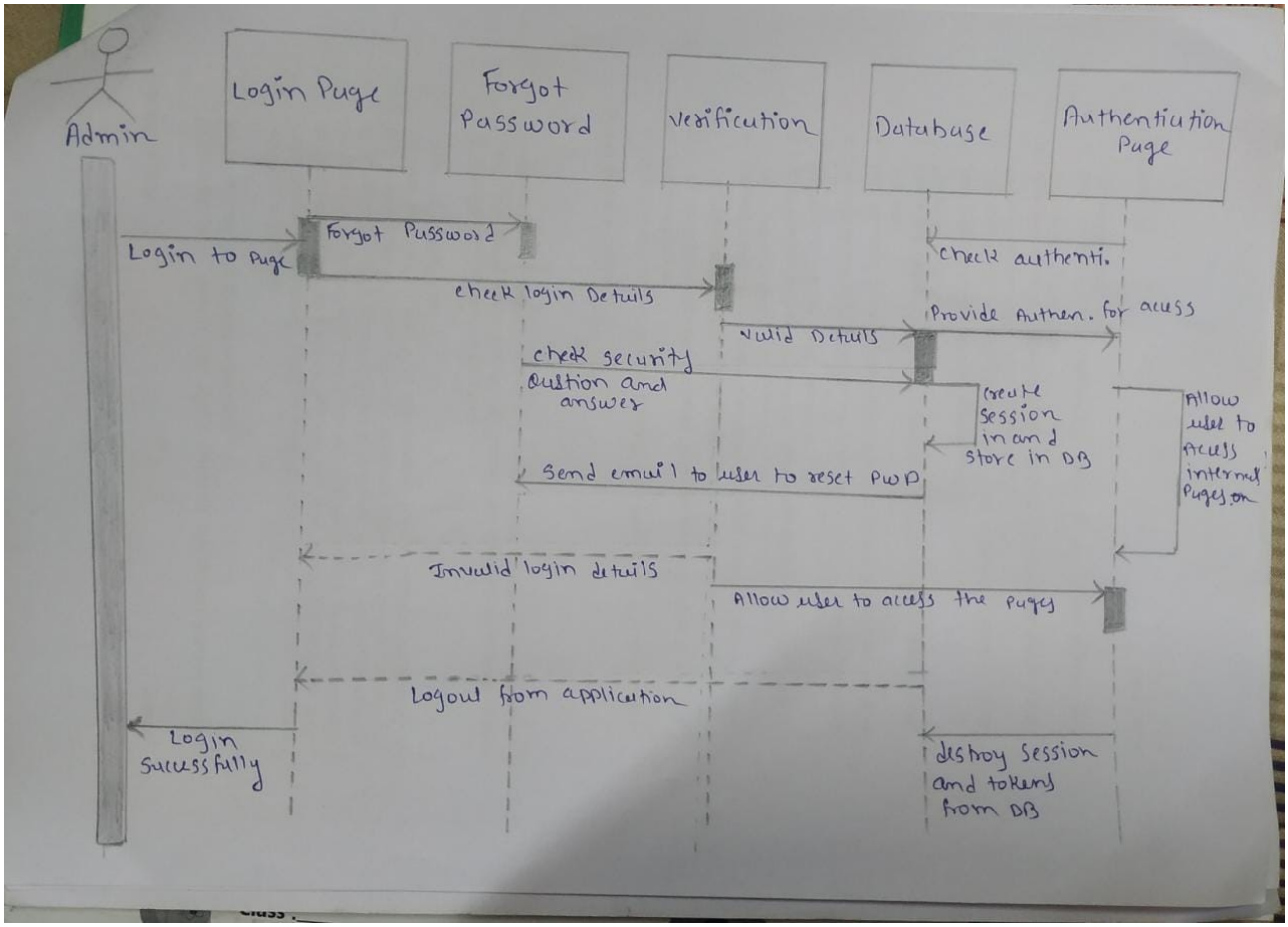
### 6.2.2 Hierarchical Activity Diagram



## 6.3 Class Diagram



## 6.4 Sequence Diagram




# 7 Cohesion and Coupling

Above, we have the “Code” class with a dependency on the options class. The problem is that we only use a single option from it. We don’t use the class for anything else and that’s why it’s a greedy dependency.

Often, we create this type of coupling because we're trying to protect our future selves. We expect that we'll need more than one option in the future so we ask for the options class now. But all we did in practice is create coupling to meet a hypothetical need.

But what about increasing cohesion? What did we do about that?

Well, increasing cohesion is a much more natural process than decreasing coupling. That's why it doesn't get talked about as much. With object-oriented programming, it all comes down to the job of a class.

This is something that comes up over and over in object-oriented design. You always want to ensure that a class has a job. This gives it purpose, but it also ensures that the code in it doesn't have a low amount of cohesion.

But it's also something that you have to keep reviewing. Our code will evolve over time and a class can lose cohesion as a result. When that happens, there's a good chance that it's because our class now does too much.

The solution is to break up our class into separate classes. This should

create classes with higher cohesion than the original class. Not only that, but our new classes should also have less coupling than the original class as well. That's because we spread the dependencies throughout the new classes.

# 8 APIs

Our APIs designed to preserve the syntax and power of SQL as much as possible, but also:

To support multiple database servers easily;

- To allow developers to leverage more complex functionality, such as transactions;
- To provide a structured interface for the dynamic construction of queries;
- To enforce security checks and other good practices;
- To provide modules with a clean interface for intercepting and modifying a site's queries.

## 8.1 Database API

Our database abstraction layer provides a unified database query API

that can query different underlying databases. It is built upon PHP's PDO (PHP Data Objects) database API, and inherits much of its syntax and semantics.

Besides providing a unified API for database queries, the database abstraction layer also provides a structured way to construct complex queries, and it protects the database by using good security practices.

## 8.2 Rest API

Our REST API provides an interface for applications to interact with your site by sending and receiving data as JSON (JavaScript Object Notation) objects. It can likewise enable your theme, plugin or custom application to present new, powerful interfaces for managing and publishing your site content.

## 8.3 Settings API

Our Settings API, allows admin pages containing settings forms to be managed semi-automatically. It lets you define settings pages, sections within those pages and fields within the sections.

New settings pages can be registered along with sections and fields

inside them. Existing settings pages can also be added to by registering new settings sections or fields inside of them.

Organizing registration and validation of fields still requires some effort from developers, but avoids a lot of complex debugging of underlying options management.

### 8.3.1 Benefits of using Settings API

- **Visual Consistency**

Using the API to generate your interface elements guarantees that your settings page will look like the rest of the administrative content. Your interface will follow the same styleguide and look like it belongs, and thanks to the talented team of our designers, it'll look awesome!

- **Robustness**

Since the API is part of CMS Core, any updates will automatically consider your plugin's settings page. If you make your own interface without using Setting API, our Core updates are more likely to break your customizations. There is also a wider audience testing and maintaining that API code, so it will tend to be more stable.

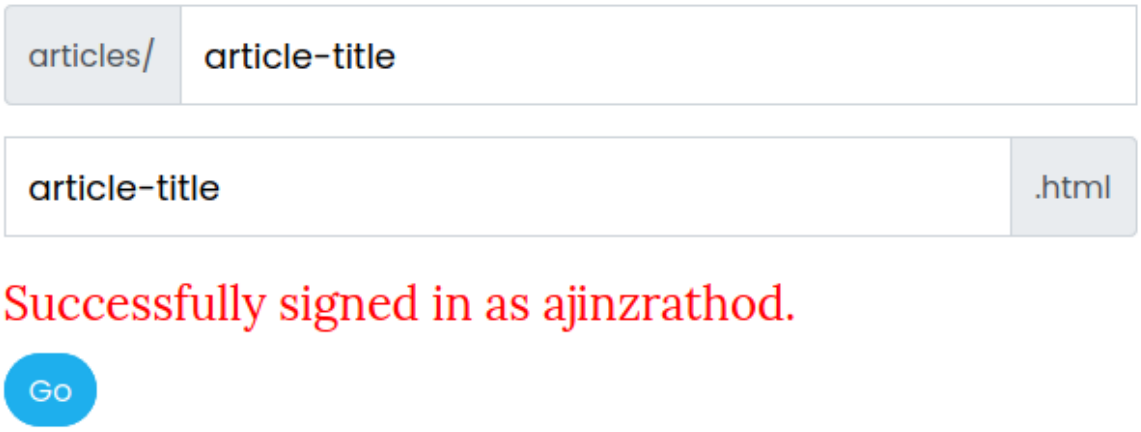
- **Less Work**

Of course the most immediate benefit is that the API does a lot of work for you under the hood. Here are a few examples of things the Settings API does besides applying an awesome-looking, integrated design.

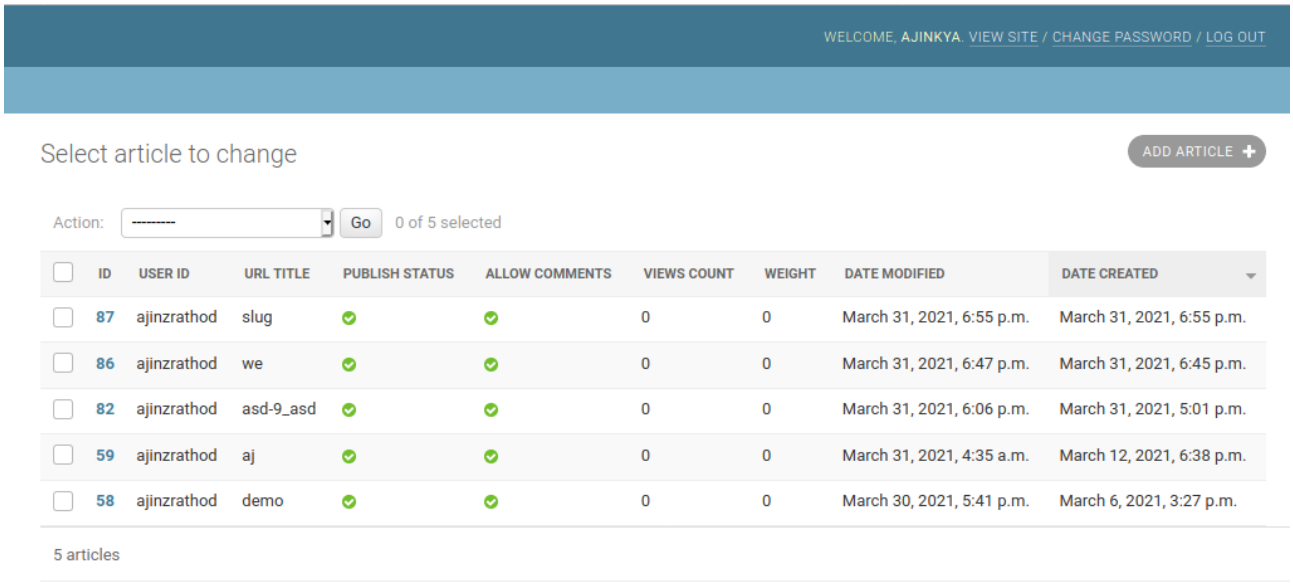
- **Handling Form Submissions:** Let us handle retrieving and storing your POST submissions.
- **Include Security Measures:** You get extra security measures such as nonces, etc. for free.
- **Sanitizing Data:** You get access to the same methods that the rest of our CMS uses for ensuring strings are safe to use.

# 9 Screenshots

## 9.1 Screenshots




*Screenshot 1*




*Screenshot 2*

# 10 Object Oriented Database

If you're thinking of creating a plugin in our system, whether small or large, for your own use or the whole world to benefit from, it's important to start right. The example code on our documentation shows the functionalities in a procedural layout, because that makes it easier

to explain. But best practice is to use modern programming techniques — classes, inheritance, closures, and so on.

### Steps to create plugin for our Object Oriented Database

## 10.1 Step 1: Naming conventions

An important thing to decide before you even put fingers to the keyboard is what to call your plugin. Even if you're the only person who will ever use it, you need to be sure that someone else isn't using the same plugin name so that you don't run into issues later. A quick search in the plugin diretcory will answer your question. And remember that it's about the internal name, not the display name; you'll see what I mean in a moment. If you've ever asked yourself how many Daves you know, you'll understand why unique naming is important.

First create a folder under /content/plugins/ with the name of your plugin, for example /content/plugins/demo-plugin/. The folder name should be unique in the entire plugin ecosystem. In theory I suppose you could use a random string here, but that would just look weird, so

don't do that.

Next, create a PHP file in that folder with the same unique name, in this case "demo-plugin.php". For clarity it's important that the names match, not because our site will get confused but because you might, or another developer who looks at the code.

## 10.2 Step 2: The class

Here's our first bit of code, in demo-plugin.php.

```
<?php

/*
Plugin Name: Demo Plugin
Description: A demo Plugin
Author: Ghanshyam Pandey
Version: 0.0.1
*/

class Demo_Plugin {
  public function __construct() {
    // Write your default code here...
  }
}

$demo_plugin = new Demo_Plugin();
?>
```

The comment block at the top is important, as that's what will be displayed in your sites plugins page to tell everyone what the plugin is.

Technically the plugin name here doesn't need to be unique, but it helps you be sure which plugin you're activating!

Next, we create the class, again using that unique name. We wouldn't want to run into issues where other plugins are using the same class name! We'll create a __construct() function, which we'll put stuff into later. And then, finally, we simply use a variable to create an instance of the class. The variable itself won't be used directly, but it's a way of making sure the class is run.

## 10.3 Step 3: Activate and deactivate

The first thing you do when you install a plugin is activate it. Some plugin don't need to do anything when they activate, but if yours does then you'll need to catch that event and do something with it. In our case, our plugin will need to set up a database table to store the number of things visitors have.

First off, we only want to listen out for activation/deactivation when we're logged in as an administrator, so we'll use is_admin() to

conditionally check. And we'll use register_activation() to listen out for the activation event.

- Construct Function

```
<?php

public function __construct() {
  if (is_admin()) {
    register_activation(__FILE__, array(&$this, 'activate'));
  }
}
public function activate() {
    // Code to run when user activates plugin
}
?>
```

Hopefully that's given you enough to get started writing your own object oriented plugin in our site.

# 11 Our User Interface Design

## 11.1 User Centric Design for all Users

- **Detailed Focus**

We design under the premise of detail. How granular can we go? What

is the smallest thing that will make a difference? Instead of worrying about solving big problems with every change, we focused on solving small things first.

While that sounds vague, the reality is that it is true. By making a lot of small tweaks to their products, we started to change how things worked for the better.

We are detail-oriented which is what helps enable the designers to be user-centric. We focused on every aspect of the experience and know what the customer wanted before they did.

- **Simplicity**

Our products are often regarded as having a minimalist aesthetic. But while they may fit that description visually, our products aren't lacking in functionality.

- **Design**

Our designers never waited for confirmation that they had the developers capable of creating their designs. They focused on designing for the future and designing for simplicity.

We believe the product is a tool helping the users achieve something.

- **Social Auth Plugins**
  - If your website is a dynamic and requires user authentication, you will love this part :) but if you just have a static website, you can just skip this part.
  - All of us know that password in each and every website can be a pain in the nerve. User cannot remember password for each and every website. So, we have embedded *Social Auth Plugins* bt default which will help users to log into your site in just one click. And thus users do not have to remember passwords each time they visit website.
  - Our research shows that user is 78% more likely to visit your site when they are provided *social apps authentication* in any website.
  - You can easily add these social apps in your website by just few steps. We support the following *Social Authentication* by default:

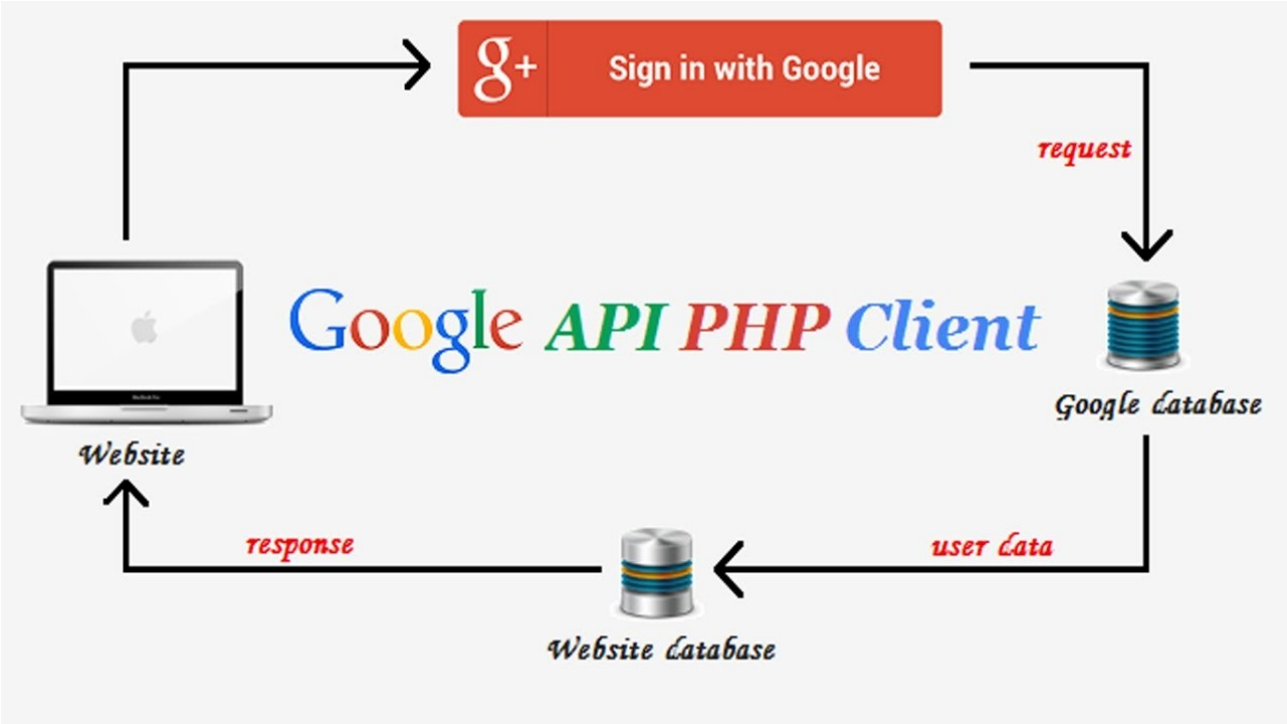
    ***1. Signin with Google***

    ***2. Signin with Facebook***

### *3. Signin with GitHub*

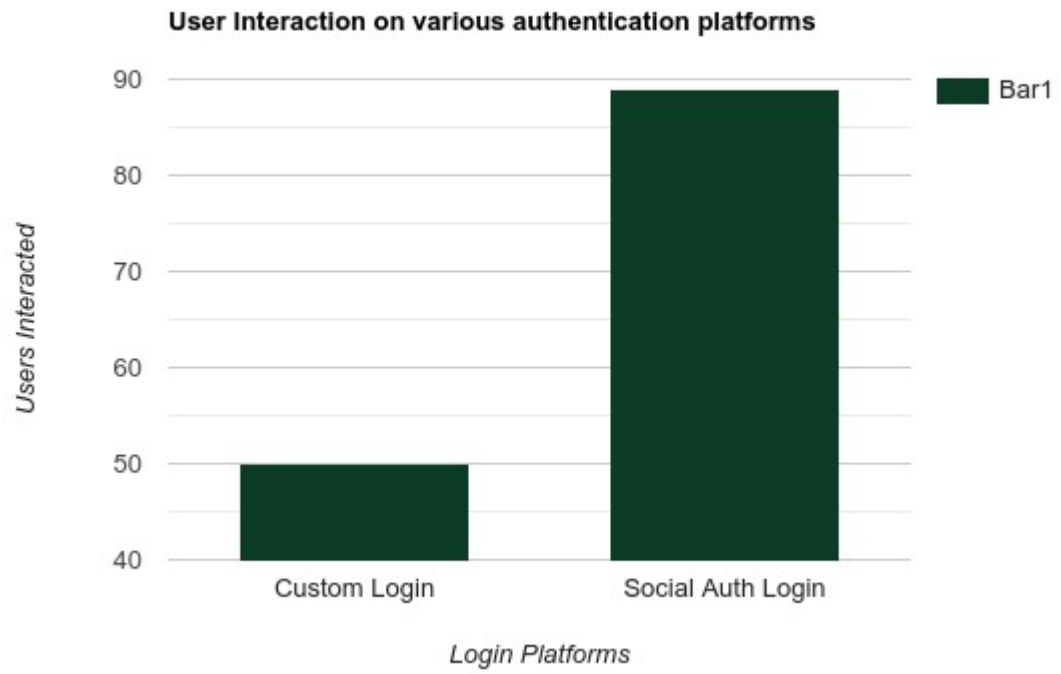
### *4. Signin with Twitter*

- This is a diagram of how we use *Login with Google.*



- 1000 users were given to browse 2 websites with exact same content. In first webiste, authentication was done by Social Apps. In second website, authentication was done by custom Login Page.

- You can see when *Custom Login* was used, only 50 users Logged into webiste. While on other hand, when *Social Auth Login* was

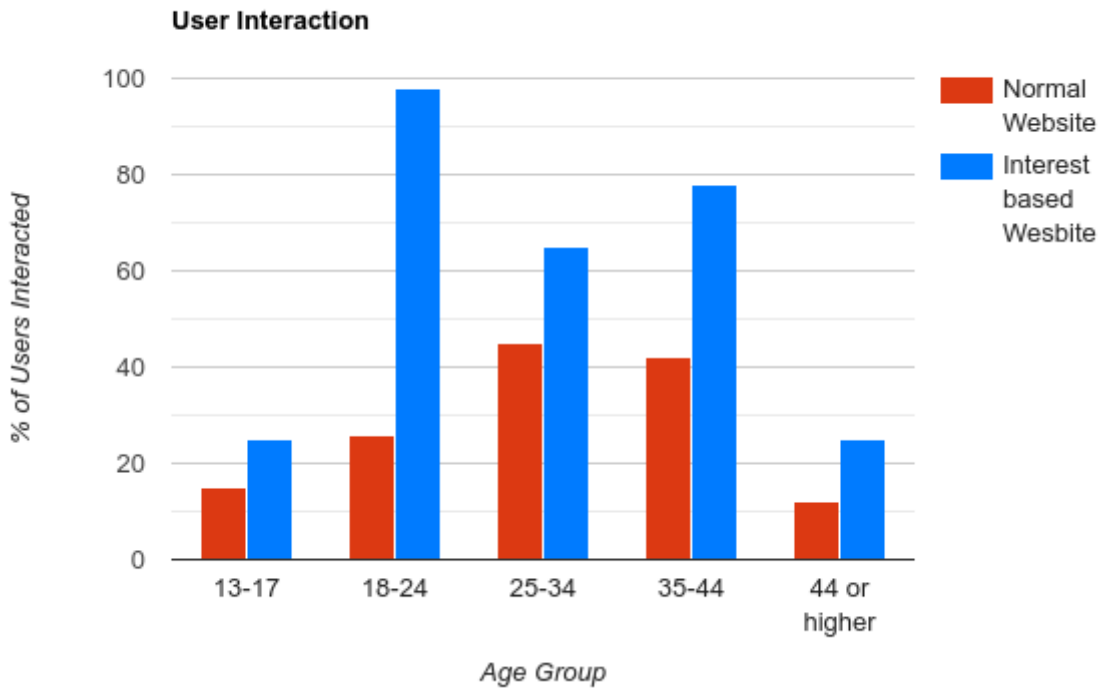



used, 89 users logged into website. WOW!! Its just amazing. Its like 78% more users.

- We also tend to provide most *Trending* Articles at the top of your

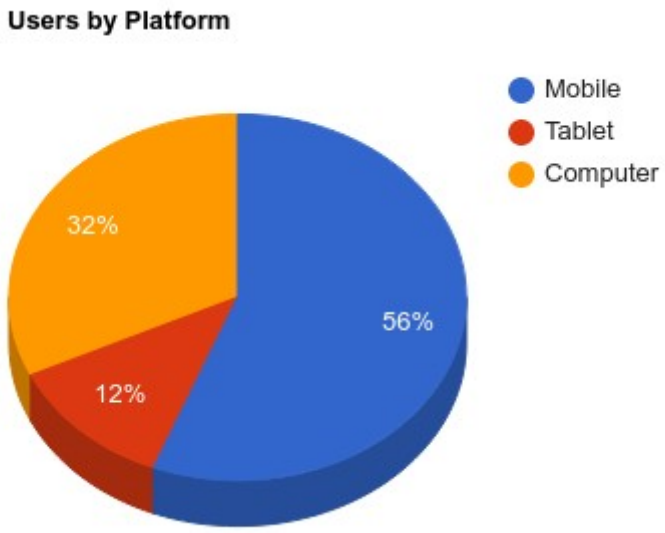
site. And show Artciles to users as per thier interests. Interests are taken into conideration by seeing what the user had previously browsed and how interest interacts user more to your amazing site.

- Users are more likely to stay into your site when it comes to show them Articles based on thier interest. Above is a age-wise diagrammatical look how users get interacted when they are shown everything according to thier likes and interests.




- Websites made in our CMS are completely device friendly. Most of websites made in our CMS are used by each and every users.

Well, you can still have a look at this pie chart to see how much users uses what kind of device to see the web pages.




## 11.2 Design for Programmers

Other than basic Text Editor, we also provide 2 highly used Text Editors called Vim and Emacs.

Emacs tends to be relatively straightforward, similar to commonly used text editors like Notepad. On the other hand, Vim is a power-user's tool, using keyboard shortcuts to speed up tasks.

### 11.2.1 For Vim Users

We have embedded Vrapper inside our custom IDE.

Vrapper acts as a wrapper for Vim-like input scheme for moving around and editing text. Unlike other plugins which embed Vim, Vrapper imitates the behaviour of Vim while still using whatever editor you have opened in the workbench. The goal is to have the comfort and ease which comes with the different modes, complex commands and count/operator/motion combinations which are the key features behind editing with Vim, while preserving the powerful features of the different Eclipse text editors, like code generation and refactoring. Vrapper tries to offer Eclipse users the best of both worlds.

If you are looking for the best customized Vim Configuration file, have a look at this GitHub page.

These are the very basic commands that we support:

x - to delete the unwanted character

- u - to undo the last the command and U to undo the whole line
- CTRL-R to redo
- A - to append text at the end
- :wq - to save and exit
- :q! - to trash all changes
- dw - move the cursor to the beginning of the word to delete that

word

- 2w - to move the cursor two words forward.
- 3e - to move the cursor to the end of the third word forward.
- 0 (zero) to move to the start of the line.
- d2w - which deletes 2 words .. number can be changed for deleting the number of consecutive words like d3w
- dd to delete the line and 2dd to delete to line .number can be changed for deleting the number of consecutive words

Once you activate this plugin, you can look at all commands just by pressing <CTRL + SHIFT + K>

### 11.2.2 For Emacs Users

We would like to share with you the details on Emacs specific features introduced in our site and some new which are upcoming.

It provides an enhanced Emacs-like experience in our text editors. Includes:

Keyboard Macros (including name, bind, save, load and auto-loading

- M-x command execution
- C-u universal-argument

- Repeat command
- C-x b, C-x 4 b, C-x 5 b with completion, search and buffer list
- Emacs style point/mark selection with both global and buffer-local mark rings
- Emacs style search and query/replace with regular expressions
- A kill ring for deleted text
- Digit argument support M-1, M-2 ... C-M-1, C-M-2 ..., etc.
- Balanced expression (s-expression) commands
- Split editor windows (C-x 2, C-x 3, C-x {, C-x }, C-x 4 .)
- Frame handling (C-x 5 2, C-x 5 1, C-x 5 0, C-x 5 o)
- Rectangle Commands
- Register Commands
- Tags-like searching
- Transposition of characters, words, lines, paragraphs and s-expressions
- Line and character commands
- Space and blank line handling
- Case conversion commands

Once you activate this plugin, you can look at all commands just by

pressing <META + SHIFT + K>

# 12 Ultimate Guide for Usage

## 12.1 Dashboard Login

To access the dashboard, you need to:

- Type in the browser the website address + "/admin" = so that it looks like this: website_name/admin.
- Fill in the USERNAME and PASSWORD fields with the user id and password you used when installing the platform for first time.
- Press *Log In*.

## 12.2 Login Successful! What to Do Next?

- This is the main area where you can make modifications and updates to the website or blog.
- ***Customize your site*** — clicking this link, you'll be taken to the Customizer (an interface where you can adjust settings for the website appearance, and preview them in real time)

- ***Change your theme completely*** — clicking this link, you'll go to an interface for visualizing themes, selecting and activating a different theme than the one already installed.
- ***Edit your front page*** — this is an option for making edits to the front page (homepage) of your website. You can add text, customize existing text (using bold, italics, bulleted list, blockquote, alignment variants), you can add links, media (images, videos) or insert a Read More tag.
- ***View your site*** — this link takes you to the live version of the website. You can thus check what it looks like for users, assess its quality and decide on what improvements it needs.

## 12.3 Navigations

### 12.3.1 Post Navigations

- ***All posts*** — by accessing this subsection, you'll see the full list of posts (both published posts and drafts). You'll also see information regarding the authors of the posts, the date when the posts were published or modified, categories where the posts belong to, and what comments correspond to posts.

- ***Add new*** — From here, you can add a new post, save it as draft or publish it immediately, ***f***ill in the title of the post, add content for the post body.
- ***Categories*** — Here, you can create categories you'll assign posts to. You'll have to give a name to the category, specify the URL version of the name (called "slug") and you can also add a description for that category.
- ***Media*** — This is the list of all media files that were uploaded. You'll find them in a table, with the date when they were uploaded, and the page/post they were attached to.
- ***Comments*** — Here, you'll find the list of comments to your blog posts. Hovering your mouse over a comment will let you use buttons for approving/replying to/editing/deleting comments or marking them as spam.

### 12.3.2 Apperance and Menu Navigations

- ***Appearance*** — This is the section you'll use most for customizing the appearance of your website/blog.
- ***Menus*** — This section is destined to adding menus in the website.

### 12.3.3 Plugin Navigation

- ***Plugins*** — There's also a dedicated section for plugins. In that section, you have an overview of the plugins that are installed, plugins that need modifications, and you'll also find there new plugins you need for the site.
    - ***Installed plugins*** — here, you'll see the list of installed plugins, with details about which plugin version your website is using. You can activate/deactivate plugins, or delete some of them, if no longer needed.

      ***Add new*** — browse through the list of available plugins or enter the name of the desired plugin in the search box and pick the plugin you'll want to install. After selection, you need to install and activate the plugin, for it to work on the website.

      ***Editor*** — this section is for editing the code for plugins. It's recommended that you should not make major changes to these lines of code, as they might affect the plugins functionality.

### 12.3.4 Settings Navigation

- ***General Settings*** — From the "General Settings" subsection of

the admin panel, you can:

- ***Add a site title*** — it sums up what the website is about
- ***Add a tagline*** — it serves as a motto for the entire website
- ***Address/URL*** — this is the address you want the website to be found at. It helps defining whether you want the website address to be a domain or a subdomain (version **with** or version **without www.**)
- ***E-mail address*** — enter the email address you want to use for admin purposes
- ***Set the site language*** — form here, you'll set the language for the website. After making changes, you'll have the dashboard and admin area displayed in the language you've selected.
- ***Timezone*** — this setting is important especially for blogs. To show accurate data concerning the date when you publish blog posts, set the timezone that corresponds to your geographic area, and that of most of your readers.
- ***Date format*** — set the format for date display, in the blog posts

- ***Time format*** — set the format for the time display, in the blog posts.

### 12.3.5 User Roles Navigation

- ***Administrator*** — This is the most powerful user role. The administrator can publish, edit or delete posts/pages, and they can also make site-wide changes, such as change the theme or manage plugins.
- ***Editor*** — Editors have the right to manage content (theirs or other users') — edit, publish or delete articles, manage comments and categories. However, they cannot make site-wide changes, like in the case of the administrator.
- ***Author or Contributor*** — Authors have limited access to the admin dashboard. They can only produce content, edit their own content, publish it and upload media files to it.

### 12.3.6 Other Navigation

- ***Permalinks*** — Permalinks are the permanent links (or URLs) of posts and pages in your website.

# 13 Getting Help

## 13.1 FAQs

First, please check if your question is answered on the FAQ. Also, search for answers using your favorite search engine, and in the forum.

If you can't find an answer, please take a few minutes to formulate your question well. Explaining the problems you are facing clearly will help others help you. See the StackOverflow guide on asking good questions.

We have also developed our own support forums at ask.ourwebsite.com

## 13.2 Found a Bug?

Well-written bug reports are *incredibly* helpful. However, there's a certain amount of overhead involved in working with any bug tracking system so your help in keeping our ticket tracker as useful as possible is appreciated.

## 13.3 Requesting Features

We're always trying to make our site better, and your feature requests

are a key part of that. Here are some tips on how to make a request most effectively:

- First request the feature to the developers in our github, not in the ticket tracker. It'll get read more closely if it's on the mailing list. This is even more important for large-scale feature requests. We like to discuss any big changes to our site's core on the mailing list before actually working on them.