ACKNOWLEDGEMENT

My sincere thanks to our internal project guide **Prof.Jalpesh Vasa** for giving us valuable inputs and ideas right from the selection of topic for project till its successful completion. Without his constant encouragement we could not have been able to achieve what we have.

I thank **Dr.Parth Shah(HOD IT department)** for his ongoing support and encouragement in every aspect. Last but not the least, entire staff of Department of IT Engineering for guiding me thoughts and vision. The successful completion of my project would not have been possible without the dedicated support from all our mentors, family and friends.

It's my good fortune that we had support and well wishes of many.We are thankful to all and those names which have been forgotten to acknowledge here but contributions have not gone unnoticed.

With Sincere Regards,
Dhvani Raval(16IT106)
Payal Rohit(16IT109)

ABSTRACT

Rails is a web application development framework written in the Ruby programming language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. It allows you to write less code while accomplishing more than many other languages and frameworks. Experienced Rails developers also report that it makes web application development more fun.

The project explains core information of Ruby on Rails It explains how to make simple blog using Rails framework which is MVC architectured. The final project named **"Tech Hub"** which is good example of blog in Ruby on Rails.

The Rails philosophy includes two major guiding principles:

- **Don't Repeat Yourself:** DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is more maintainable, more extensible, and less buggy.
- **Convention Over Configuration:** Rails has opinions about the best way to do many things in a web application, and defaults to this set of conventions, rather than require that you specify minutiae through endless configuration files.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter:1 Introduction

1.1 Project Overview

To easily demonstrate the principles of working with Ruby on Rails We chose to build a basic blog. Each blog post will be able to be created, read, edited, and deleted. There will also be comments associated with each individual blog post. Comments will be able to be created and deleted.

With Ruby on Rails, the possibilities are pretty endless in terms of what you can build. I'm sure new features and improvements to our blog are easy to spot.

Moreover, Ruby on Rails is based on an MVC (Model–View–Controller) design pattern, which supports rapid project development. Secondly, it also amazed us by by the selection of gems – special libraries that allow developers to add any functionality from authorization and authentication to file uploading and payments

1.1.1 Ruby on Rails Installation

Before you install Rails, you should check to make sure that your system has the proper prerequisites installed. These include Ruby and SQLite3.

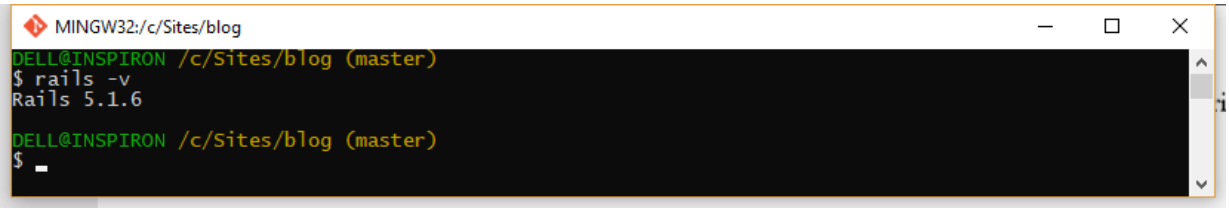
Open up a command line prompt. On macOS open Terminal.app, on Windows choose "Run" from your Start menu and type 'cmd.exe' or Git Bash. Any commands prefaced with a dollar sign $ should be run in the command line. Verify that you have a current version of Ruby installed:

If Ruby is not installed, then download an installation package from rubyinstaller.org. Follow the **download** link, and run the resulting installer. This is an exe file **rubyinstaller-2.2.2.x.exe** and will be installed in a single click. It's a very small package, and you'll get RubyGems as well along with this package.

Install Rails − With Rubygems loaded, you can install all of Rails and its dependencies using the following command through the command line –

```
C:\> gem install rails
```

To check the version of rails following command is used:



Keeping Rails Up to Date:

Assuming you have installed Rails using RubyGems, keeping it up-to-date is relatively easy. We can use the same command in both Windows and Linux platform. Use the following command −

```
tp> gem update rails
```

1.1.2 Create Simple blog Application

Rails comes with a number of scripts called generators that are designed to make your development life easier by creating everything that's necessary to start working on a particular task. One of these is the new application generator, which will provide you with the foundation of a fresh Rails application so that you don't have to write it yourself.

To use this generator, open a terminal, navigate to a directory where you have rights to create files, and type:

```
$ rails new blog
```

The blog directory has a number of auto-generated files and folders that make up the structure of a Rails application. Most of the work in this tutorial will happen in the app folder, but here's a basic rundown on the function of each of the files and folders that Rails created by default:

File/Folder	Purpose
app/	Contains the controllers, models, views, helpers, mailers, channels, jobs and assets for your application. You'll focus on this folder for the remainder of this guide.
bin/	Contains the rails script that starts your app and can contain other scripts you use to setup, update, deploy or run your application.
config/	Configure your application's routes, database, and more.
config.ru	Rack configuration for Rack based servers used to start the application.
db/	Contains your current database schema, as well as the database migrations.
Gemfile Gemfile.lock	These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem.
lib/	Extended modules for your application.
log/	Application log files.
package.json	This file allows you to specify what npm dependencies are needed for your Rails application. This file is used by Yarn.
public/	The only folder seen by the world as-is. Contains static files and compiled assets

Rakefile	This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application.
README.md	This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on.
test/	Unit tests, fixtures, and other test apparatus.
tmp/	Temporary files (like cache and pid files).
vendor/	A place for all third-party code. In a typical Rails application this includes vendored gems.
.gitignore	This file tells git which files (or patterns) it should ignore.
.ruby-version	This file contains the default Ruby version.

1.1.2 Structure of Rails Application

1.2 Scope

Our Ruby Blog is an ongoing commentary by an individual,is a traditional, most common blog. Personal bloggers usually take pride in their blog postseven if their blog is neber read. Blogs often become more than a way to just communicate; they became a way to reflect on life, or works of art.Blogging can have a sentimental quality.

Few personal blogs rise to fame and the main stream but some personal blogs quickly graner an extensive following.One type of personal blog refers to as micro blog,is extremely detailed and seeks to capture a moment in time.

Our blog website is developed based on Ruby on Rails framework which has following features:

- ✓ Benefits of different Categories like: News, Programming Languages ,different frameworks and tutorials etc.
- ✓ User can comment if he/she has any query or wants any extra information for that particular topic.

1.3 Objective

Blogging has become one of the more popular pastimes on the internet. Some people blog for money, others blog about current events, and others blog for humor. The list goes on. Increasingly, bloggers are use weblogs as a personal journal, preferring to keep it out of the spotlight.

- ✓ Our project's aim is to create a blog application.
- ✓ In our blog application, only the admin can post the updates regarding any programming language, frameworks or news and anyone can view that.
- ✓ If user has any query regarding any particular topic then he/she can comment about it and admin can post regarding that.

1.4 Problem and Solution Statement

Building a blog with comments using Ruby on Rails is a foundational exercise We went through to learn more about the framework. Working together, both Ruby and Rails lend us a hand to generate a fairly simple MVC pattern built on top of a CRUD approach when working with dynamic data.

Chapter:2 System Analysis

2.1 User Characteristics

Every system or project can use to the many user. But it can access in different way.

Every user of the system are given some authentication and according to that they can perform the task.

Our blog website can be accessed by any user and they can read the posts and comment on that posts.

2.2 Tools and Technology

2.2.1 Software requirement

1. Operating system : Microsoft windows
2. Front End :Ruby on Rails
3. Back End: Puma Webserver(Rails Server), Postgresql

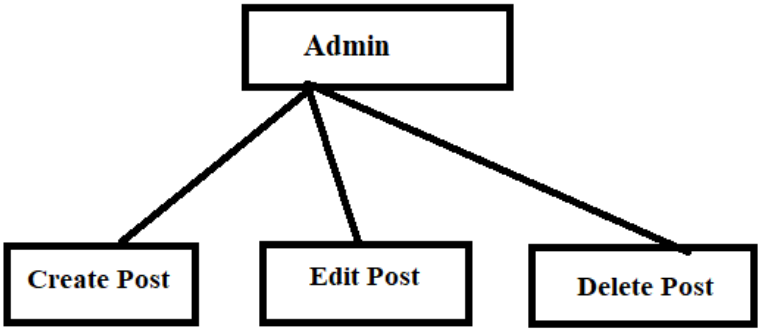
Programming Language

1. Ruby
2. HTML , css, js

Chapter:3 System Design

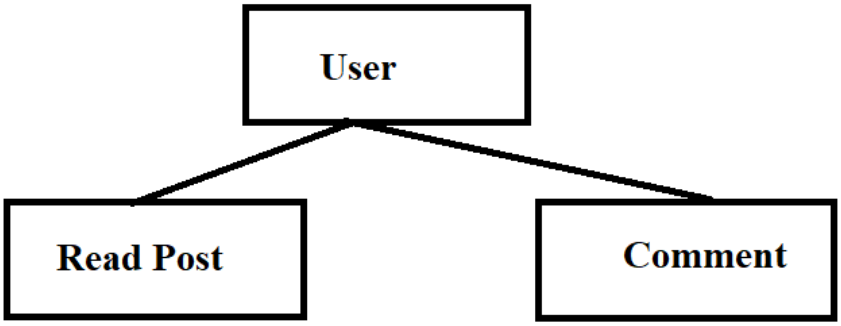
3.1 Data flow diagram:

1)Admin



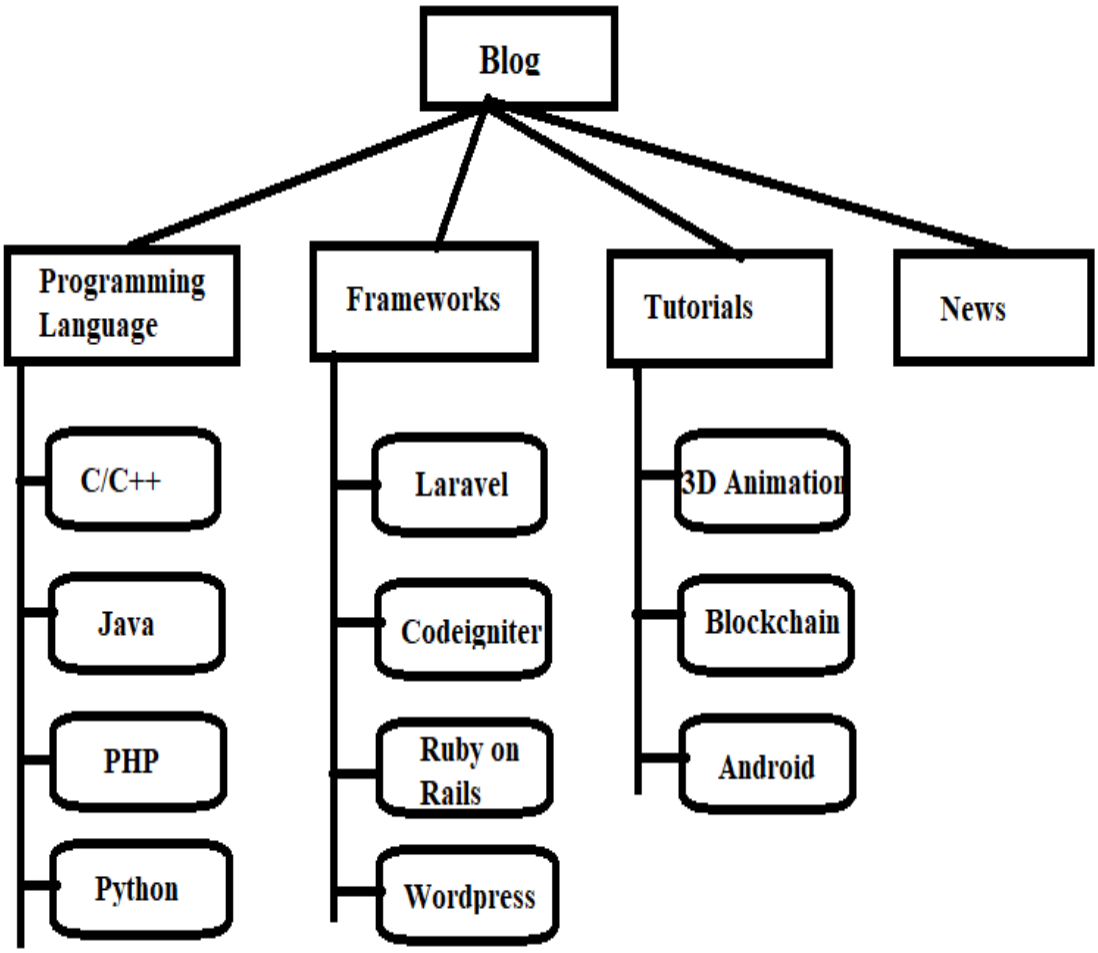
3.1.1 Admin

2)User:



3.1.2 User

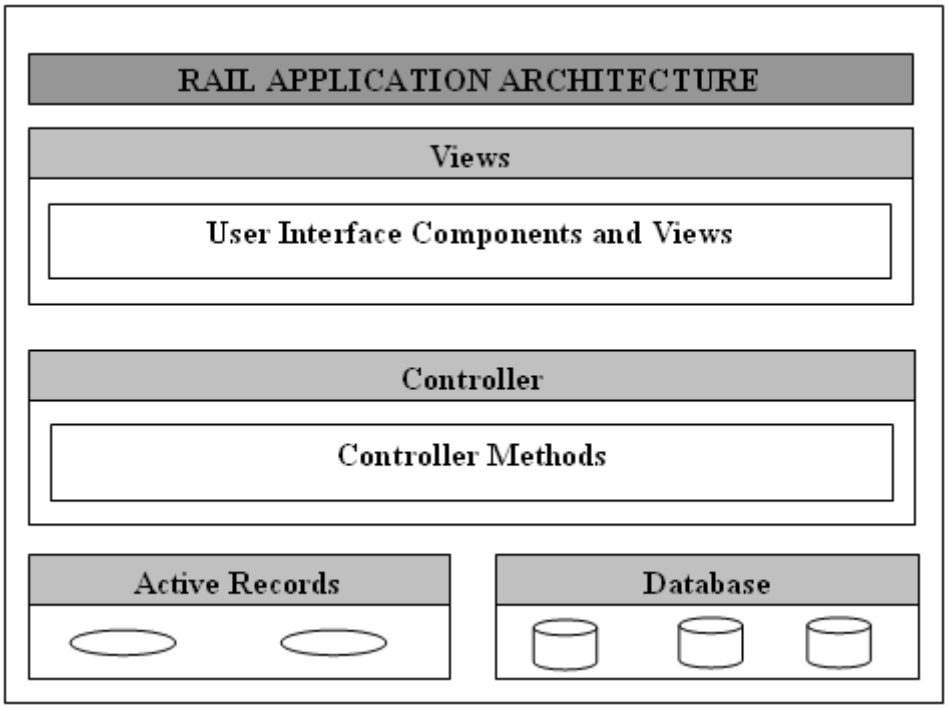
3.2 GUI Forms



3.2.1 GUI

Chapter:4 Implementation

4.1 Implementation Environment



4.1.1 MVC Structure

4.2 Coding Standards

4.2.1 Models

1)Active Admin:

```
class CreateActiveAdminComments < ActiveRecord::Migration::Current

  def self.up

    create_table :active_admin_comments do |t|

      t.string :namespace

      t.text   :body

      t.references :resource, polymorphic: true

      t.references :author, polymorphic: true

      t.timestamps

    end

    add_index :active_admin_comments, [:namespace]

  end

  def self.down

    drop_table :active_admin_comments

  end
end
```

2)Comments:

```
class CreatePostComments < ActiveRecord::Migration[5.1]

  def change

    create_table :post_comments do |t|

      t.string :name
```

```
    t.string :email

    t.text :body
    t.references :post, foreign_key: true

    t.timestamps

  end
        add_index :post_comments, :post_id
 end
end
```

4.2.2 Views

1)Header:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Blog</title>
        <link        href="http://fonts.googleapis.com/css?family=Abel"        rel="stylesheet"
type="text/css" />
  <%= stylesheet_link_tag    "application", :media => "all" %>
 <%= javascript_include_tag "application" %>
 <%= csrf_meta_tags %>
 </head>
<body>
        <div id="wrapper">
        <div id="header-wrapper">
                <div id="header">
                        <div id="logo">
                                <h1><a href="#">My Ruby Blog</a></h1>
                                <p>Learning Ruby on Rails</p>
                        </div>
                        <div id="menu">
                                <ul class="main-nav">
                                        <li><%= link_to "Home", root_path %></li>
                                        <li><%= link_to "About", about_path %></li>
                                        <li><%= link_to "Blog", posts_path %></li>
                                        <li><%= link_to "Contacts", contact_path %></li>
                                        <li><%= link_to "Resources", resources_path %></li>
                                </ul>
                        </div>
```

```
                </div>
        </div>
<!-- end #header -->
```

2)Sidebar:

```
<div id="sidebar">
        <ul>
                <li>
                        <div style="clear: both;"> </div>
                </li>
                <li>
                    <h2>Get The Framework</h2>
                                <p>You can download Ruby and/or Ruby on Rails for free
right now
                <br/>
                <a style="color:#D93544;" href="http://rubyonrails.org/download"
target="blank">GET IT NOW</a>
            </p>
                </li>
                <li>
                        <h2>Categories</h2>
                        <ul>
                                <% all_categories.each do |cat| %>
                                        <li><%= link_to cat.name,category_path(:id => cat.id)
%></li>
                                <% end %>
                        </ul>
                </li>
                <li>
                        <h2>Blogroll</h2>
                                <ul>
                                        <% all_posts.each do |post| %>
                                                <li><%= link_to post.title,post_path(:id =>
post.id) %></li>
                                        <% end %>
                                </ul>
                </li>

        </ul>
</div>
<!-- end #sidebar -->
```

4.2.3 Controllers:
1)Post Controller:

```
class PostsController < ApplicationController
	def index
		@search = Post.search(params[:search])
		@posts=@search.all
		#@posts=@post.all

	end

	def new
		@post = Post.new
		@category = Category.all
	end

	def create
		@post = Post.new(post_params)
		if @post.save
			redirect_to posts_path, :notice => "Your post has been saved"
		else
			render "new"
		end
	end

	def edit
		@post = Post.find(params[:id])
	end

	def update
		@post = Post.find(params[:id])
		if @post.update_attributes(post_params)
			redirect_to post_path, :notice => "Your post has been updated"
		else
			render "edit"
		end

	end

	def show
		@post = Post.find(params[:id])
		@user = AdminUser.all
```

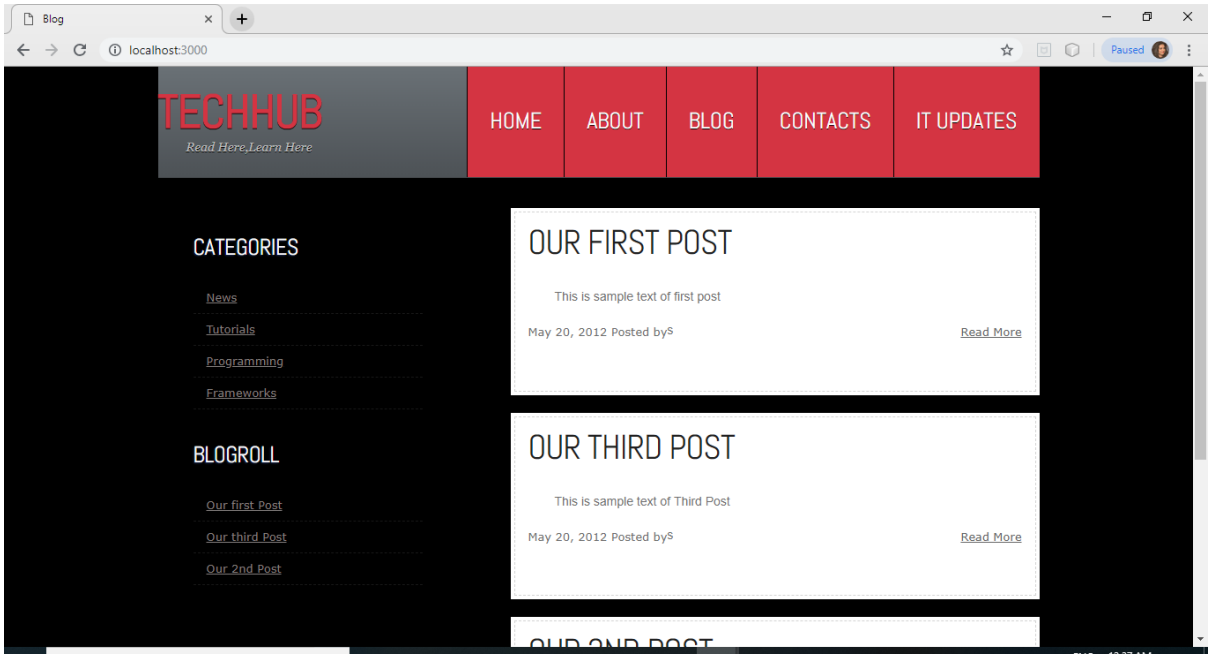
```
                @post_comment = PostComment.new(:post => @post)
        end

        def destroy
                @post = Post.find(params[:id])
                @post.destroy
                redirect_to posts_path, :notice => "Your post has been deleted"
        end

        private

        def post_params
                params.require(:post).permit!
        end

end
```

2)Categories Controller:

```
class CategoriesController < ApplicationController
  before_action :set_category, only: [:show, :update, :destroy]

  def index
    @categories = Category.all
  end

  def show
        @categories = Category.find(params[:id])
        @title = @category.name
        @posts = @category.posts
  end


  private
    # Use callbacks to share common setup or constraints between actions.
    def set_category
      @category = Category.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def category_params
      params.require(:category).permit(:name)
    end
end
```

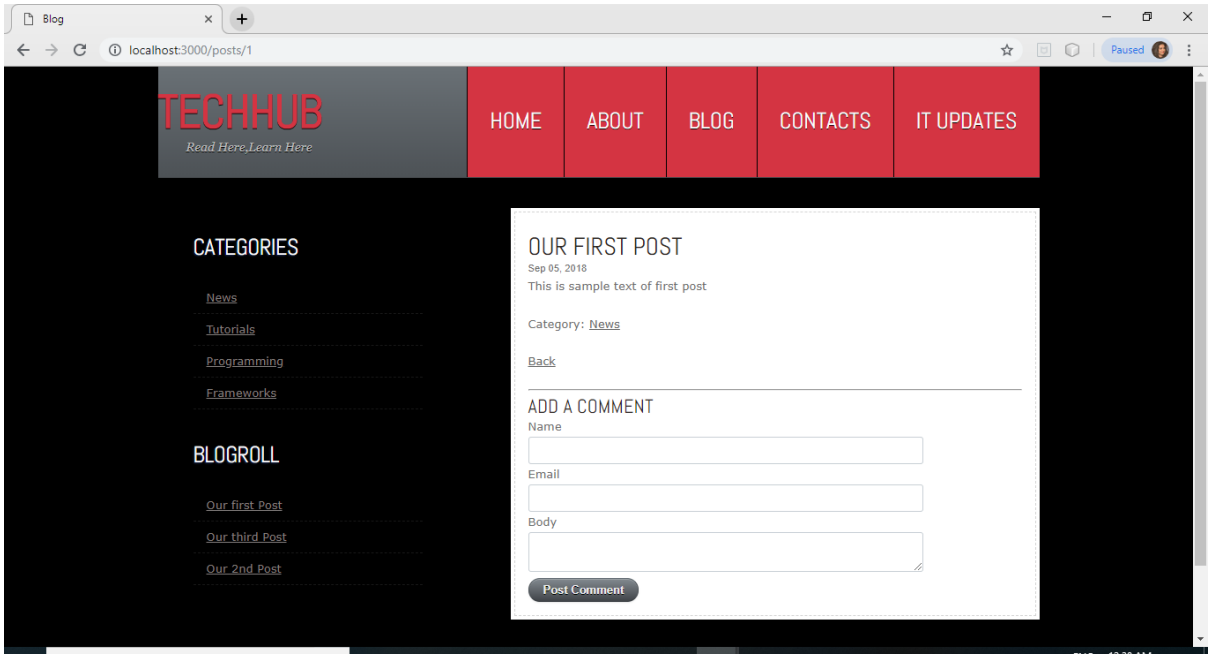
4.3 Snapshots

1)Home Page:



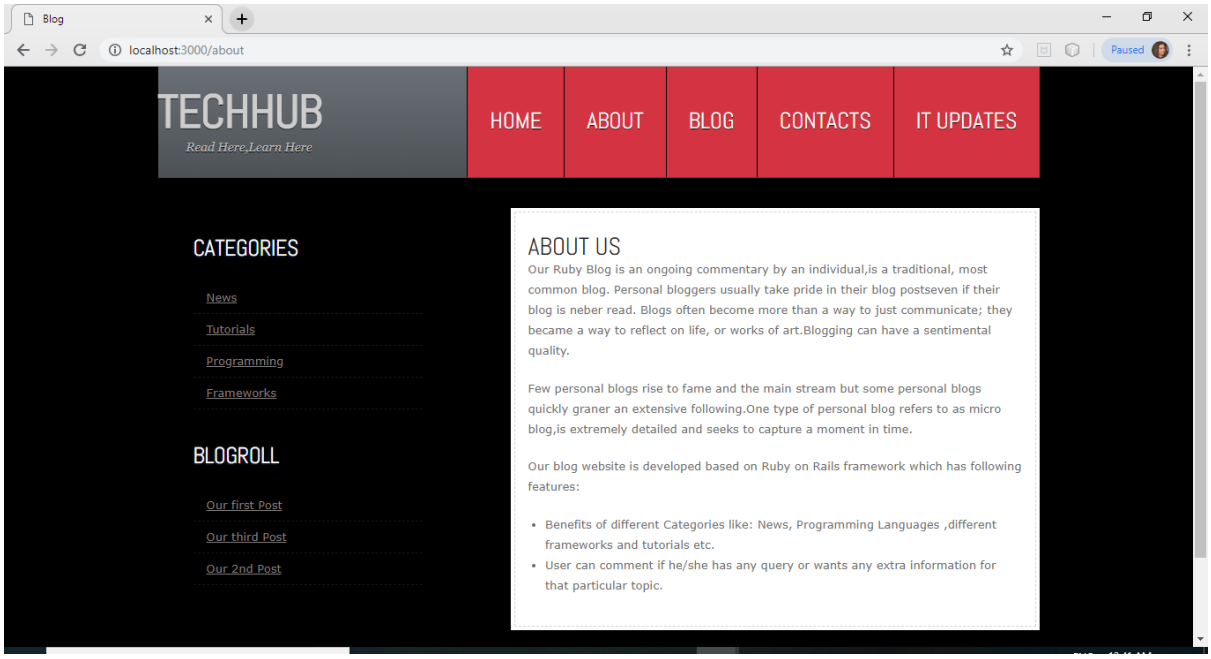
4.3.1 Home Page

2)Post Page:



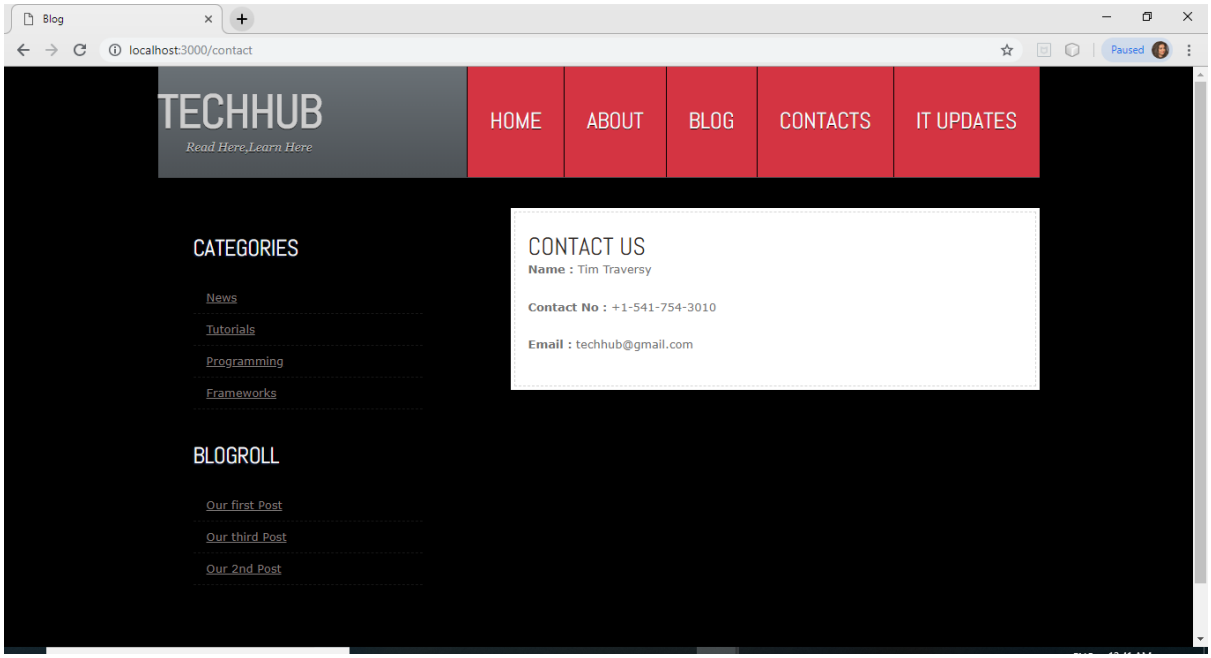
4.3.2 Post Page

3)About Us:



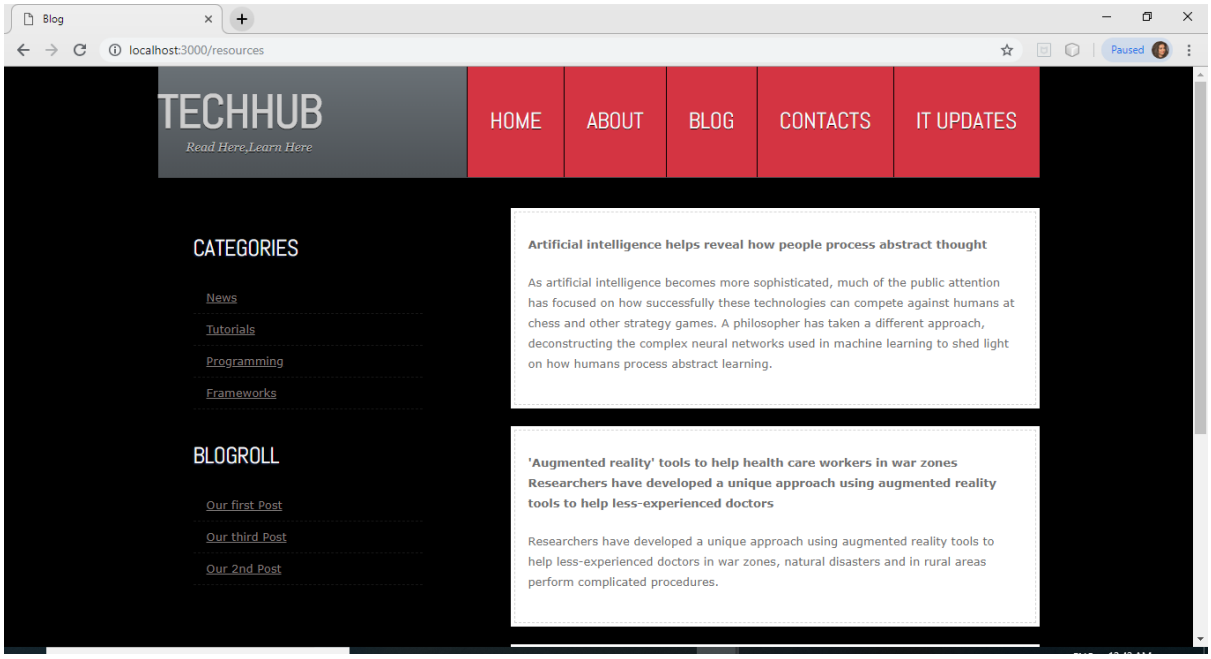
4.3.3 About Us

4)Contact Us:



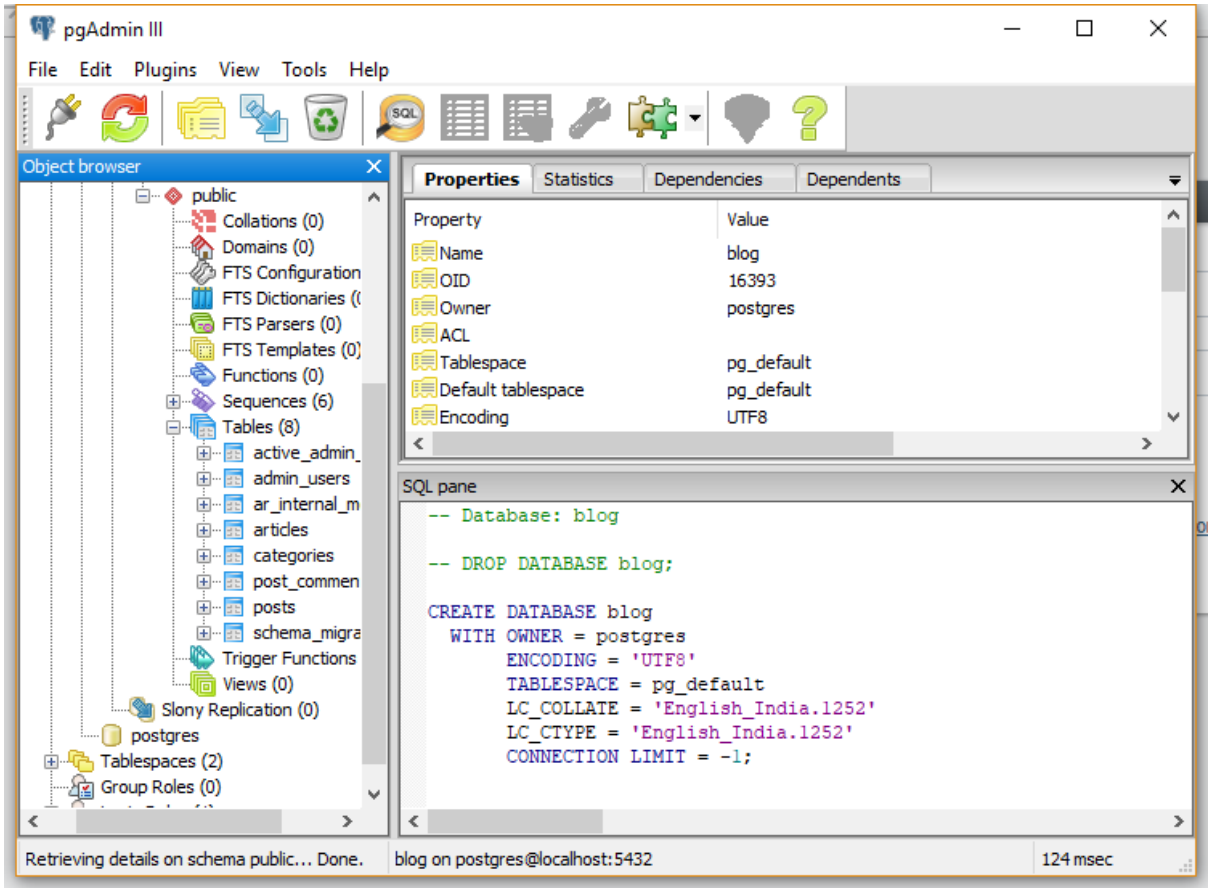
4.3.4 Contact Us

5)IT Updates:



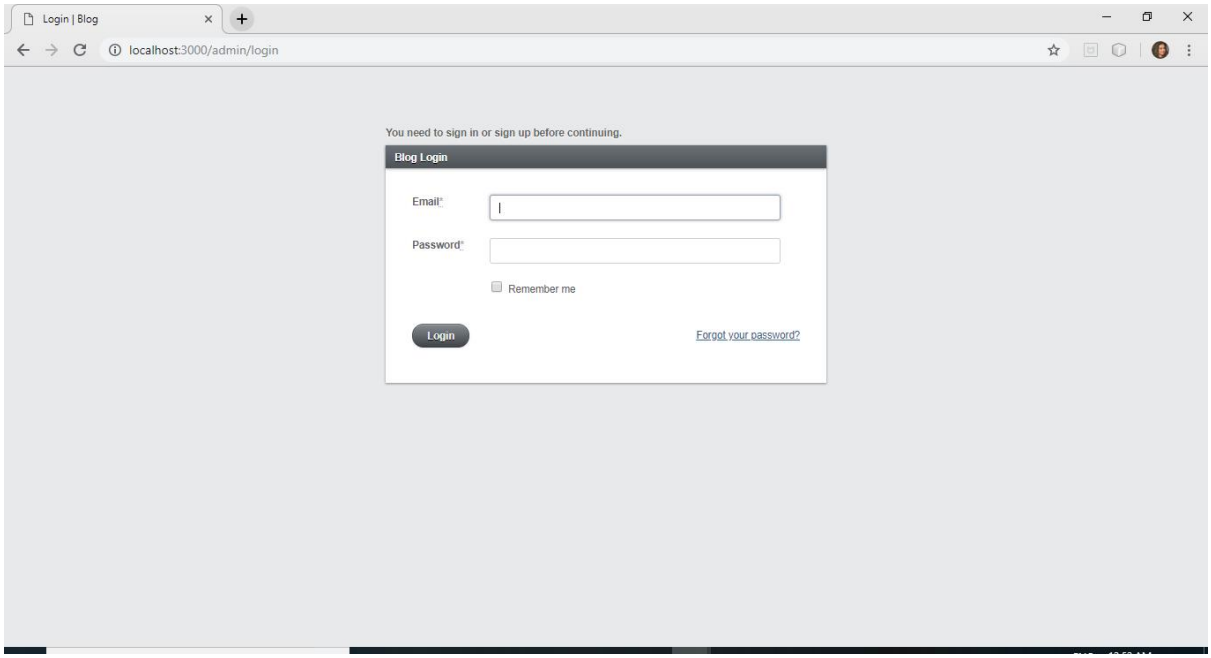
4.3.5 IT UPDATES

6)Database:



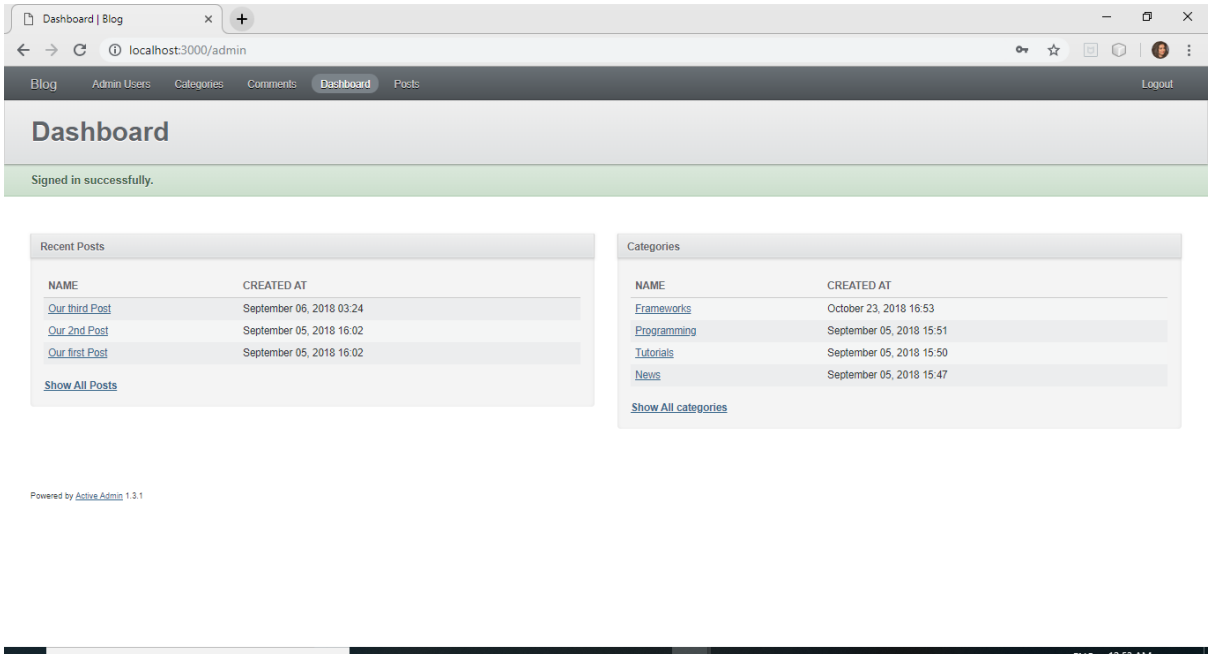
4.3.6 Database

7)Active Admin Login:



4.3.7 Active Admin Login

8) DashBoard:



4.3.8 DashBoard

Chapter:5 Constraints and Future Enhancement

- ✓ Authorization: only logged in user can view posts
- ✓ User can create post
- ✓ Searching facility
- ✓ Feedback of post

Chapter:6 Conclusion

We have learned a new framework “RUBY ON RAILS” and by using it we have create a personal blog application. Our blog application can help anyone to view the updates of any programming language, frameworks, tutorial and also be tuned with the latest news about various new technologies. Also the user can comment his/her query or request for more information about the post.

References:

- https://www.udemy.com/learn-ruby-on-rails-from-scratch/
- https://guides.rubyonrails.org/getting_started.html
- https://github.com/rails/rails
- https://stackoverflow.com/
- https://www.railstutorial.org/book