

CAMP GUIDE

A PROJECT REPORT

Submitted by

ROHIT SHARMA

2100290140113

HARSH

2100290140067

NIKUNJ TYAGI

2100290140097

**Submitted in partial fulfilment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**Under the Supervision of
Mr. Praveen Kumar Gupta
Assistant Professor**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(JUNE 2023)**

CERTIFICATE

Certified that **Rohit Sharma, Harsh** and **Nikunj Tyagi** have carried out the project work having title “**Camp Guide**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Rohit Sharma (2100290140113)

Harsh (2100290140067)

Nikunj Tyagi (2100290140097)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Praveen Kumar Gupta
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner

Signature of External Examiner

Dr. Arun Tripathi
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The Camp Guide project is all about the recreational camping activities done at the various tourist places. It also allows user to post comment, Feedback as well as the related pictures of the respective places. It is basically a camping site through one camping that place and know about the camping place, see the related picture and also sign up to comment and one can also post the cost queries.

Camp Guide will play an essential role in making decisions like choosing a campground. This project is all about the recreational camping activities done at the various tourist places. It also allows user to post comment, Feedback as well as the related pictures of the respective places. It is basically a camping site through one camping that place and know about the camping place. People interested in camping get to camping spots by difficulties and find it being crowded or polluted by other campers. This site uses the feedback of the people already visited a specific camp to make easier decisions for other campers who can choose the camping spots based on this feedback from other people.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my thesis supervisor, **Mr. Praveen Kumar Gupta** Sir for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Tripathi**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to **my all-faculty** members & also my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Rohit Sharma

Harsh

Nikunj Tyagi

TABLE OF CONTENTS

S.No.	Contents	Pg. No.
1.	Certificate	ii
2.	Abstract	iii
3.	Acknowledgements	iv
4.	List of Chapters	v
5.	List of Figures	vii

List of Chapters

Chapter 1 - Introduction	1-2
1.1 Project description	1
1.2 Project Scope	1
1.3 Project Features	2
Chapter 2 Feasibility Study	3-4
2.1 Related Things	3
2.2 Front End Development	3
2.3 Back End Development	4
Chapter 3 Problem Statement	5-6
3.1 Motivation	5
3.2 Problem Statement	5
3.3 Objective	6
Chapter 4 System Analysis and Design	7-10
4.1 Software Requirement Specification	7
4.1.1 Hardware Requirements	7
4.1.2 Software Requirements	7

4.2 System Design	8
4.2.1 Use Case Diagram	8
4.2.2 Data Flow Diagram	9
4.2.3 Entity Relationship Diagram	10
4.3 Database Design	11-12
4.3.1 Database Tables	11-12
Chapter 5 Proposed Work	13-18
5.1 Tools Introduction	13
5.2 HTML	13
5.3 CSS	15
5.4 JavaScript	17
5.5 MongoDB	18
Chapter 6 Project Modules	19-21
6.1 Registration Module	19
6.2 Login Module	20
6.3 Campground Module	20
6.4 Search Module	21
Chapter 7 Coding	22-40
7.1 Module wise code	22
Chapter 8 Testing	41-45
8.1 Introduction of Testing	41
8.2 Principles of Testing	42
8.3 Types of Testing	43
8.4 Test Cases	44
8.5 Test Cases for Camp Guide	45
Chapter 9 Conclusion and Future Scope	46-46
9.1 Conclusion	46
9.2 Future Scope	46
Chapter 10 References	47

LIST OF FIGURES

Figure No.	Name of Figure	Page No.
Fig 2.1	Front-End Development	3
Fig 2.2	Back-End Development	4
Fig 4.1	Use Case Diagram of Camp Guide	8
Fig 4.2	Data Flow Diagram of Camp Guide	9
Fig 4.3	ER Diagram of Camp Guide	11
Fig 4.4	User Table	11
Fig 4.5	Campground Table	12
Fig 5.1	CSS3	15
Fig 5.2	JavaScript	17
Fig 5.3	MongoDB	18
Fig 6.1	Registration Module	19
Fig 6.2	Login Module	20
Fig 6.3	Campground Module	20
Fig 6.4	Search Module	21

CHAPTER 1

INTRODUCTION

1.1 Project Description

The Camp Guide project is all about the recreational camping activities done at the various tourist places. It also allows user to post comment, Feedback as well as the related pictures of the respective places. It is basically a camping site through one camping that place and know about the camping place, see the related picture and also sign up to comment and one can also post the cost queries.

It is an internet application designed to feature, rate and review different campgrounds, different users (read campers) can put in their comments and concerns, in order that it's a well informed and well prepared camping. This app contains API secrets and passwords that have been hidden deliberately, so the app cannot be run with its features on your local machine. In order to review or create a campground, you want to have an account pass real time data to a long distance.

Camp Guide may be a website where users can create and review campgrounds. In order to review or create a campground, you want to have an account. This project was designed using Node.js, Express, MongoDB, and Bootstrap. Passport.js was used to handle authentication. The Login Feature gives the user right to login to the website after creation of the account successfully for the website. The login process is on high priority. During the login process the user needs to put the User id and Password in order to access the website contents. Whenever the user gets logged-in to the website he/she will be directed to the Home page.

1.2 Project Scope

Due to the vast number of camping sites spread worldwide, it is very difficult to get to know the quality of a camping ground and the services provide at the camping sites. Camp guide is one such website that provides all the information regarding a

camping sites. Camp guide provides us with reviews and ratings provided by the users and other services offered at the camping sites.

Camp Guide will play an essential role in making decisions like choosing a campground. This project is all about the recreational camping activities done at the various tourist places. It also allows user to post comment, Feedback as well as the related pictures of the respective places. It is basically a camping site through one camping that place and know about the camping place. People interested in camping get to camping spots by difficulties and find it being over crowded or polluted by other campers. This site uses the feedback of the people already visited a specific camp to make easier decisions for other campers who can choose the camping spots based on these feedback from other people.

1.3 Project Features

- Authentication: User login with username and password o Admin sign-up with admin code
- Authorization: One cannot manage posts and view user profile without being authenticated
- One cannot edit or delete posts and comments created by other users
- Admin can manage all posts and comments Manage campground posts with basic functionalities:
- Create, edit and delete posts and comments
- Upload campground photos
- Display campground location on Google Maps and Search existing campgrounds
- Manage user account with basic functionalities:
- Password reset via email confirmation (disabled)
- Profile page setup with sign-up
- Flash messages responding to users' interaction with the app
- Responsive web design.

CHAPTER 2

FEASIBILITY STUDY

2.1 Related Things

Camping with Campground allows you to see how the locals really live in certain places. Often, people live off the grid and immersed in nature. This is a very in-depth experience where you can see natural beauty every day and learn about simpler living first hand from locals.

When you go camping, you often live in a communal space with locals or other volunteers. This allows you to form new bonds and friendships with your peers. You can trade stories and travel advice. You grow and learn together. You can have a more intimate experience with the people around you and discover more about different cultures in the process.

Camping is the best way to create a deep connection with a foreign land because you are living outside, with few boundaries. If you love nature and want to step outside of your comfort zone, an adventurous, hands-on volunteer camping experience with World packers is the way to go.

2.2 Front-End Development



Fig 2.1 Front End Development

A front-end dev takes care of layout, design and interactivity using HTML, CSS and JavaScript. They take an idea from the drawing board and turn it into reality.

What you see and what you use, such as the visual aspect of the website, the drop down menus and the text, are all brought together by the front-end dev, who writes a series of programs to bind and structure the elements, make them look good and add interactivity. These programs are run through a browser.

2.3 Back-End Development



Fig 2.2 Back End Development

The backend developer engineers what is going on behind the scenes. This is where the data is stored, and without this data, there would be no frontend. The backend of the web consists of the server that hosts the website, an application for running it and a database to contain the data.

The backend dev uses computer programs to ensure that the server, the application and the database run smoothly together. This type of dev needs to analyze what a company's needs are and provide efficient programming solutions. To do all this amazing stuff they use a variety of server-side languages, like PHP, Ruby, Python and Java.

CHAPTER 3

PROBLEM STATEMENT

3.1 Motivation

A great way to immerse yourself in a new country is by camping with World packers. We have lots of camping experiences available where travelers can live in a foreign country while getting up close and personal with the natural world. We have work exchanges that include camping all over the globe, so you can volunteer in a country that intrigues you.

Camping is especially perfect if you are an adventurous soul or a nature lover. Volunteering in cities and working in hostels or hotels is great for those who want a bit of comfort. But for those who want to venture off the beaten path and try a more spontaneous and wild volunteer experience, camping is the way to go.

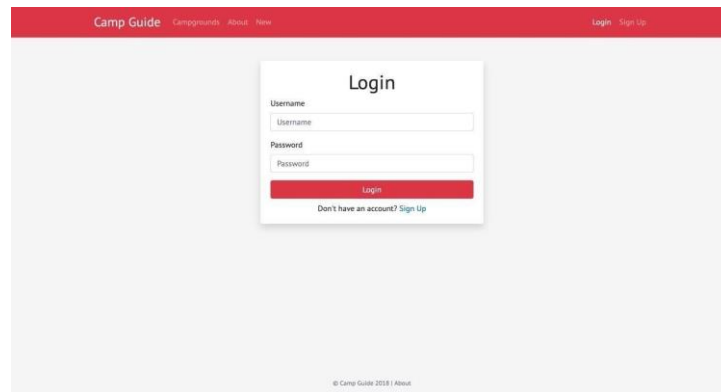
We envisage this product can truly reach the objectives with the rapid popularization of the tablets.

3.2. Problem Statement

Due to the vast number of camping sites spread worldwide, it is very difficult to get to know the quality of a camping ground and the services provided at the camping sites. Camp guide is one such website that provides all the information regarding a camping site. Camp guide provides us with reviews and ratings provided by the users and other services offered at the camping sites.

3.3 Objectives

In this project, we are providing Camping areas available around the world. people are not getting proper information about the camping area i.e its opening time, closing time, Any stay available near the camping area, how much it will cost per person, and working.



The image shows a web application interface for 'Camp Guide'. At the top, there is a red navigation bar with the text 'Camp Guide' and links for 'Campsgrounds', 'About', and 'Home'. On the right side of this bar are links for 'Login' and 'Sign Up'. The main content area is light gray and features a white login form in the center. The form is titled 'Login' and contains two input fields: 'Username' and 'Password'. Below these fields is a red button labeled 'Login'. Underneath the button, there is a link that says 'Don't have an account? Sign Up'. At the bottom of the page, there is a small copyright notice: '© Camp Guide 2018 | About'.

CHAPTER 4

SYSTEM ANALYSIS AND DESIGN

4.1 Software Requirement Specifications (H/W and S/W requirements)

- **Hardware Requirements**

- Processor- Dual core processor @ 1.65 GHz or above
- RAM- 256 MB or above.
- Hard Disk- 40 GB or above.
- Monitor- 14" VGA
- Mouse, Standard 104 enhanced keyboard

- **Software Requirements**

- Operating System- Windows 10
- Language Used: HTML, CSS, JS
- Editor- Visual Studio Code
- Browsers- Google chrome, Mozilla FireFox
- Platforms Used- Node.js, Express.js
- Database Used- MongoDB

4.2 System Design

4.2.1 Use case diagram

The purpose of use case diagram The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams. Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled to present the outside view. In brief, the purposes of use case diagrams can be said to be as follows –

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

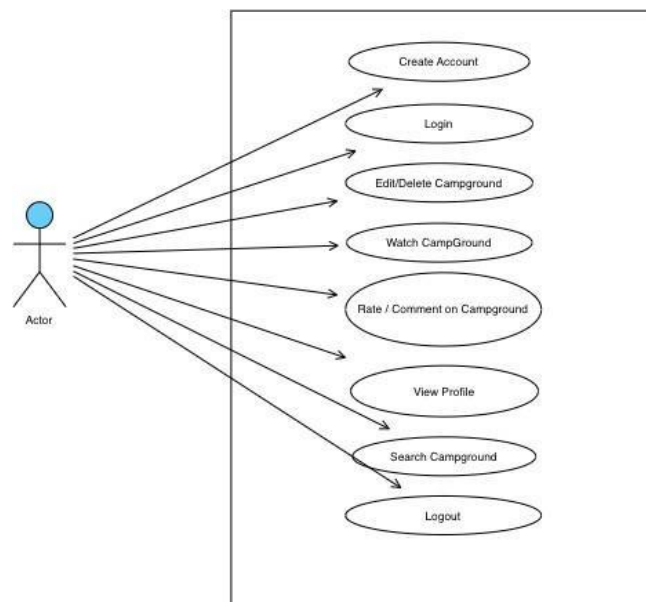


Fig 4.1 Use Case Diagram of Camp Guide

4.2.2 Data flow diagram (DFD)

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as level 0 of the DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower-level

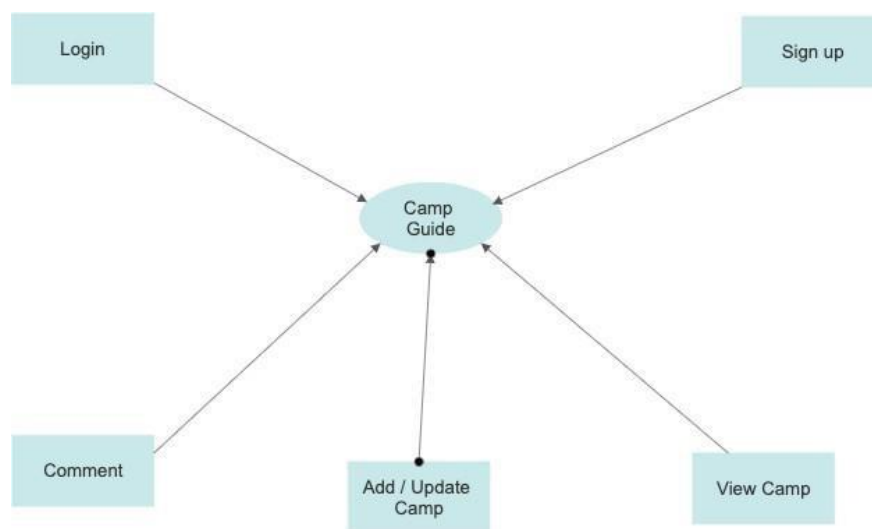


Fig 4.2 1-level Data Flow Diagram of Camp Guide

4.2.3 Entity Relationship Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used to sketch out the design of a database.

- **ER Diagram: Entity**

An Entity can be any object, place, person or class. In ER Diagram, an entity is represented using rectangles. Consider an example of an Organization - Employee, Manager, Department, Product and many more can be taken as entities in an Organization.

- **ER Diagram: Attribute**

An Attribute describes a property or characteristics of an entity. For example, Name, Age etc. can be attributes of a Student. An attribute is represented using ellipse.

- **ER Diagram: Relationship**

A Relationship describes relation between entities. Relationship is represented using diamonds or rhombus.

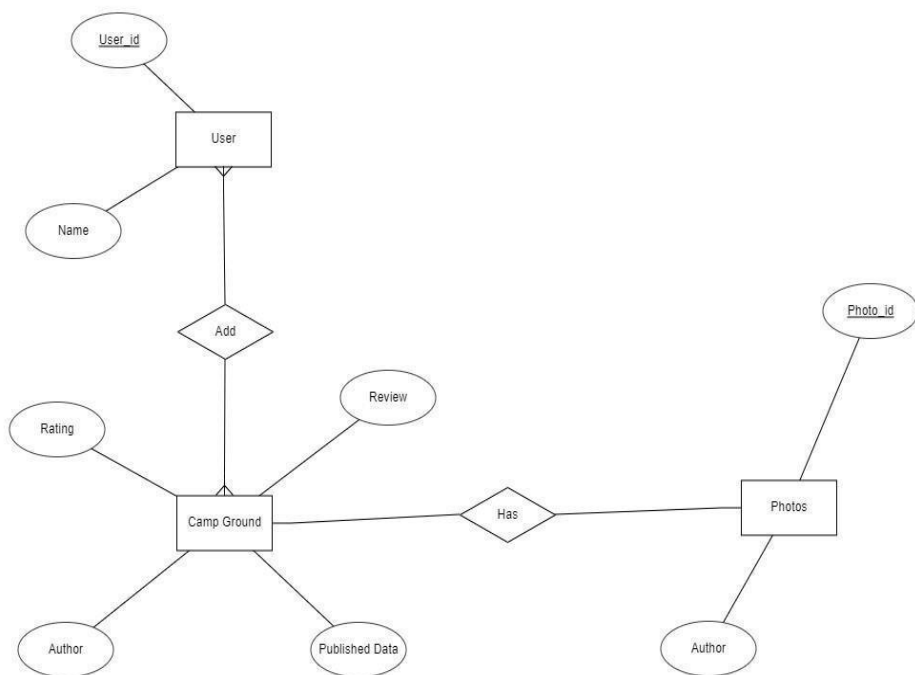


Fig 4.3 ER Diagram Of Camp Guide

4.3 Database Design

4.3.1 Database Tables

- **User Table**

#	Name	Type	Collation	Attributes	Null	Default
1	username	varchar(40)	latin1_swedish_ci		No	None
2	password	varchar(100)	latin1_swedish_ci		No	None
3	email	varchar(40)	latin1_swedish_ci		No	None
4	phone	bigint(10)			No	None
5	fullname	varchar(40)	latin1_swedish_ci		No	None
6	image	varchar(40)	latin1_swedish_ci		No	None
7	imageId	varchar(40)	latin1_swedish_ci		No	None
8	joined	date			No	None

Fig. 4.4 User Table

- Campground Table

#	Name	Type	Collation	Attributes	Null	Default
1	name	varchar(40)	latin1_swedish_ci		No	<i>None</i>
2	price	int(10)			No	<i>None</i>
3	image	varchar(40)	latin1_swedish_ci		No	<i>None</i>
4	imageld	varchar(40)	latin1_swedish_ci		No	<i>None</i>
5	description	text	latin1_swedish_ci		No	<i>None</i>
6	location	varchar(40)	latin1_swedish_ci		No	<i>None</i>
7	lat	int(10)			No	<i>None</i>
8	lng	int(10)			No	<i>None</i>
9	phone	bigint(10)			No	<i>None</i>
10	booking	varchar(40)	latin1_swedish_ci		No	<i>None</i>

Fig. 4.5 Campground Table

CHAPTER 5

PROPOSED WORK

5.1 Tools Introduction

The Translate and Edit application had been planned to consist of two parts- front-end and back-end development. The front-end is the part of the web that you can see and interact with (e.g. Client-side programming). While front-end code interacts with the user in real time, the back-end interacts with a server to return user ready results. The front-end is a combination of HTML, CSS and JavaScript coding. By using JavaScript, modifications of the design of a web page can be made immediately, however only temporary and visible only by the user.

Normally the user would not have rights to modify web content dynamically on the server side. Logically, administrators are the ones who deal with back-end modification of databases for example, as they often contain sensitive data which should not be available to see or modify by the general public. These front-end and back-end tools includes languages like HTML, CSS, JavaScript, PHP, MYSQL etc.. We will discuss all these languages in brief as given.

5.2 HTML

- **Introduction**

HTML (Hyper Text Mark-Up Language) is what is known as a "mark-up language" whose role is to prepare written documents using formatting tags. The tags indicate how the document is presented and how it links to other documents.

The World Wide Web (WWW for short), or simply the Web, is the worldwide network formed by all the documents (called "web pages") which are connected to one another by hyperlinks.

Web pages are usually organized around a main page, which acts as a hub for browsing other pages with hyperlinks. This group of web pages joined by hyperlinks and centered around a main page is called a website.

The Web is a vast living archive composed of a myriad of web sites, giving people access to web pages that may contain formatted text, images, sounds, video, etc.

- **What is the Web?**

The Web is composed of web pages stored on web servers, which are machines that are constantly connected to the Internet and which provide the pages that users request. Every web page, and more generally any online resource, such as images, video, music, and animation, is associated with a unique address called a URL. The key element for viewing web pages is the browser, a software program which sends requests to web servers, then processes the resulting data and displays the information as intended, based on instructions in the HTML page. The most commonly used browsers on the Internet include: ➤

Mozilla Firefox, ➤ Microsoft Internet Explorer, ➤
Netscape Navigator, 6 ➤ Safari, ➤ Opera

- **Versions of HTML:**

HTML was designed by Tim Berners-Lee, at the time a researcher at CERN (European Organization for Nuclear Research), beginning in 1989. He officially announced the creation of the Web on Usenet in August 1991. However, it wasn't until 1993 that HTML was considered advanced enough to call it a language (HTML was then symbolically christened HTML 1.0).

RFC 1866, dated November 1995, represented the first official version of HTML, called HTML2.0. After the brief appearance of HTML 3.0, which was never officially released, HTML 3.2 became the official standard on January 14, 1997. The most significant changes to HTML 3.2 were the standardization of tables, as well as many features relating to the presentation of web pages.

On December 18, 1997, HTML 4.0 was released. Version 4.0 of HTML was notable for standardizing and frames. HTML version 4.01, which came out on December 24, 1999, made several minor modifications to HTML 4.0.

5.3 CSS:



Fig. 5.1 CSS

- **What Is CSS?**

- CSS stands for Cascading Style Sheets.
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files.
- CSS describes how HTML elements should be displayed.
- CSS Saves a Lot of Work! The style definitions are normally saved in external .css files.
- With an external stylesheet file, we can change the look of an entire website by changing just one file!
- CSS can be either external or internal.

- **CSS Syntax:**

A CSS rule-set consists of a selector and a declaration block: CSS selector: The selector points to the HTML element you want to style. The declaration block contains one or more declarations separated by semicolons. Each declaration includes a CSS property name and a value, separated by a colon. A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces. The External CSS can be declared in the required HTML page as: `link rel="stylesheet" href="CSS_file_name".css"`.

The External CSS file is saved by using the '.css' extension , whereas the internal CSS is saved in corresponding HTML file using the <style> tag. Using External CSS is much better than using Internal .Here are a few reasons this is better.

- Easier Maintenance
- Reduced File Size
- Reduced Bandwidth
- Improved Flexibility

5.4 JavaScript



Fig. 5.2 JavaScript

JavaScript is an object-based scripting language that is lightweight and cross-platform. JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code. JavaScript is among the most powerful and flexible programming languages of the web. It powers the dynamic behaviour on most websites, including this one.

It is mainly used for:

- Client-side validation
- Dynamic drop-down menus.
- Displaying data and time.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box).
- Displaying clocks etc.

Example of JavaScript-

```
<h2>Welcome to JavaScript</h2>
```

```
<script>
```

```
</script>
```

Here ,<script> tag is used to initialize the script and document.write() is a function used to write. Like CSS, JavaScript also can be can be placed in:

1. Between the body tag of html

5.5 MongoDB



Fig. 5.3 MongoDB

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need document database with the scalability and flexibility that you want with the querying and indexing that you need.

- **Makes development easy**

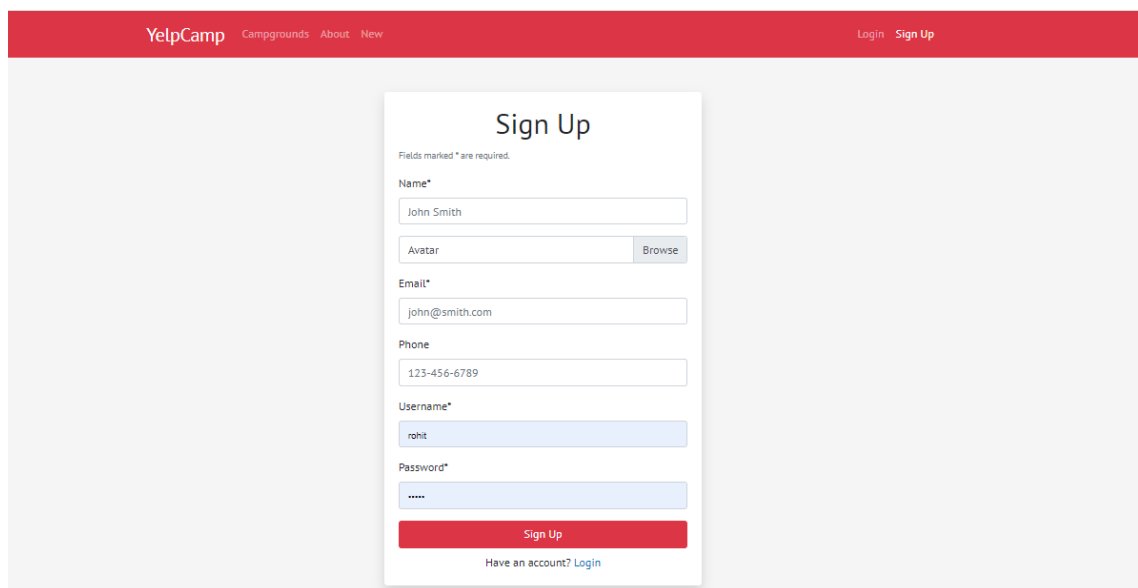
MongoDB's document model is simple for developers to learn and use, while still providing all the capabilities needed to meet the most complex requirements at any scale. We provide drivers for 10+ languages, and the community has built dozens more.

- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time
- The document model maps to the objects in your application code, making data easy to work with
- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyse your data
- MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.

CHAPTER 6

PROJECT MODULES

- **Registration Module**



The screenshot shows a web application interface for 'YelpCamp'. At the top, a red navigation bar contains the 'YelpCamp' logo and links for 'Campgrounds', 'About', and 'New'. On the right side of the bar are links for 'Login' and 'Sign Up'. The main content area is light gray and features a white 'Sign Up' form. The form includes a title 'Sign Up', a note 'Fields marked * are required.', and several input fields: 'Name*' (with 'John Smith' entered), 'Avatar' (with a 'Browse' button), 'Email*' (with 'john@smith.com' entered), 'Phone' (with '123-456-6789' entered), 'Username*' (with 'rohit' entered), and 'Password*' (with masked characters '****' entered). A red 'Sign Up' button is at the bottom of the form, and a link 'Have an account? Login' is below it.

Fig. 6.1 Registration Module

- **Login Module**

The screenshot shows the 'YelpCamp' website header with navigation links: 'Campgrounds', 'About', and 'New'. On the right, there are links for 'Login' and 'Sign Up'. The main content area features a white 'Login' form with a red border. The form has two input fields: 'Username' with the text 'rohit' and 'Password' with masked characters '*****'. Below the fields is a red 'Login' button. At the bottom of the form, it says 'Don't have an account? [Sign Up](#)'.

Fig. 6.2 Login Module

- **Campground Module**

The screenshot displays the 'YelpCamp' website with a red header containing 'YelpCamp', 'Campgrounds', 'About', 'New', and a user profile 'rohit'. The main content area shows a detailed view of 'Camp Mountain Valley'. On the left, there's a sidebar with filters: 'Booking Window' (March - December), 'Amenities' (Boating), and 'Contact' (9897979894). Below this is a Google Maps embed showing the location in Dhanaulti, Uttarakhand, India. The main content area features a large photo of a wooden cabin. Below the photo, the text reads: 'Camp Mountain Valley', '\$3000/Night', '★★★★★ 2 Reviews', 'Camping Site in Dhanaulti Uttarakhand', and 'Submitted by rohit on May 27, 2023'. At the bottom, there are icons for editing and deleting the listing.

Fig. 6.3 Campground Module

- **Search Module**

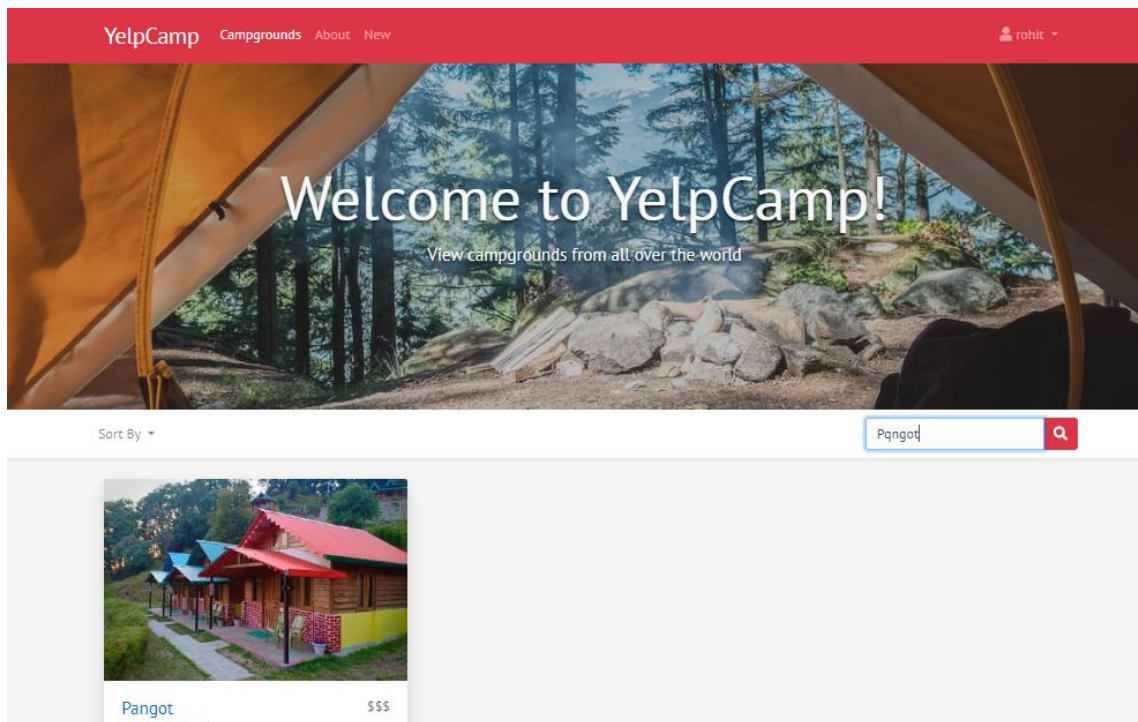


Fig. 6.4 Search Module

CHAPTER 7

CODING

- **App.js**

```
require("dotenv").config();
var express = require("express"),
    app = express(),
    bodyParser = require("body-parser"),
    mongoose = require("mongoose"),
    Campground = require("./models/campground"),
    Comment = require("./models/comment"),
    passport = require("passport"),
    LocalStrategy = require("passport-local"),
    User = require("./models/user"),
    methodOverride = require("method-override"),
    flash = require("connect-flash");

// Requiring routes
var commentRoutes = require("./routes/comments"),
    campgroundRoutes = require("./routes/campgrounds"),
    indexRoutes = require("./routes/index");

mongoose.connect(
  process.env.DATABASEURL,
  { useNewUrlParser: true }
);

app.use(bodyParser.urlencoded({ extended: true }));
app.set("view engine", "ejs");
app.use(express.static(__dirname + "/public"));
app.use(methodOverride("_method"));
app.use(flash());

// PASSPORT CONFIG
app.use(
  require("express-session")({
    secret: "shibas are the best dogs in the world.",
```

```

    resave: false,
    saveUninitialized: false
  })
);
app.locals.moment = require("moment");
app.use(passport.initialize());
app.use(passport.session());
passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

app.use(function(req, res, next) {
  res.locals.currentUser = req.user;
  res.locals.error = req.flash("error");
  res.locals.success = req.flash("success");
  next();
});

app.use("/", indexRoutes);
app.use("/campgrounds", campgroundRoutes);
app.use("/campgrounds/:id/comments", commentRoutes);

app.get("*", function(req, res) {
  res.render("error");
});

app.listen(process.env.PORT || 3000, function() {
  console.log("listening on http://localhost:3000/");
});

```

• Index.js

```

var Campground = require("../models/campground");
var Comment = require("../models/comment");
var User = require("../models/user");
var middlewareObj = {};

middlewareObj.checkCampgroundOwnership = function(req, res, next) {
  Campground.findById(req.params.id, function(err, foundCampground) {
    if (err || !foundCampground) {
      req.flash("error", "Sorry, that campground does not exist!");
      res.redirect("/campgrounds");
    } else if (
      foundCampground.author.id.equals(req.user._id) ||
      req.user.isAdmin
    ) {
      next();
    } else {
      req.flash("error", "You do not have permission to delete that campground.");
      res.redirect("/campgrounds");
    }
  });
};

```

```

    ) {
      req.campground = foundCampground;
      next();
    } else {
      req.flash("error", "You don't have permission to do that!");
      res.redirect("/campgrounds/" + req.params.id);
    }
  });
};

middlewareObj.checkCommentOwnership = function(req, res, next) {
  Comment.findById(req.params.comment_id, function(err, foundComment) {
    if (err || !foundComment) {
      req.flash("error", "Sorry, that comment does not exist!");
      res.redirect("/campgrounds");
    } else if (
      foundComment.author.id.equals(req.user._id) ||
      req.user.isAdmin
    ) {
      req.comment = foundComment;
      next();
    } else {
      req.flash("error", "You don't have permission to do that!");
      res.redirect("/campgrounds/" + req.params.id);
    }
  });
};

middlewareObj.checkProfileOwnership = function(req, res, next) {
  User.findById(req.params.user_id, function(err, foundUser) {
    if (err || !foundUser) {
      req.flash("error", "Sorry, that user doesn't exist!");
      res.redirect("/campgrounds");
    } else if (foundUser._id.equals(req.user._id) || req.user.isAdmin) {
      req.user = foundUser;
      next();
    } else {
      req.flash("error", "You don't have permission to do that!");
      res.redirect("/campgrounds/" + req.params.user_id);
    }
  });
};

middlewareObj.isLoggedIn = function(req, res, next) {
  if (req.isAuthenticated()) {
    return next();
  }
};

```



```

    }
    req.flash("error", "You need to be logged in to do that!");
    res.redirect("/login");
  };

```

```

module.exports = middlewareObj;

```

Routes

- **Campground.js**

```

var express = require("express");

var router = express.Router();

var Campground = require("../models/campground");
var middleware = require("../middleware");
var NodeGeocoder = require("node-geocoder");

var options = {
  provider: "google",
  httpAdapter: "https",
  apiKey: process.env.GEO_CODE_API,
  formatter: null
};

var geocoder = NodeGeocoder(options);

var multer = require("multer");

var storage = multer.diskStorage({
  filename: function(req, file, callback) {
    callback(null, Date.now() + file.originalname);
  }
});

var imageFilter = function(req, file, cb) {
  // accept image files only
  if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/i)) {
    return cb(new Error("Only image files are allowed!"), false);
  }
}

```

```

    }
    cb(null, true);
  };

  var upload = multer({
    storage: storage,
    fileFilter: imageFilter
  });

  var cloudinary = require("cloudinary");
  cloudinary.config({
    cloud_name: "dkxmrlblp",
    api_key: process.env.API_KEY,
    api_secret: process.env.API_SECRET
  });

  var Fuse = require("fuse.js");

  // INDEX - show all campgrounds
  router.get("/", function(req, res) {
    var noMatch = null;
    if (req.query.search) {
      Campground.find({}, function(err, allCampgrounds) {
        if (err) {
          console.log(err);
        } else {
          var options = {
            shouldSort: true,
            threshold: 0.5,
            location: 0,
            distance: 100,

```

```

    maxPatternLength: 32,
    minMatchCharLength: 2,
    keys: ["name", "location"]
  };

  var fuse = new Fuse(allCampgrounds, options);
  var result = fuse.search(req.query.search);
  if (result.length < 1) {
    noMatch = req.query.search;
  }
  res.render("campgrounds/index", {
    campgrounds: result,
    noMatch: noMatch
  });
}

});

} else if (req.query.sortby) {
  if (req.query.sortby === "rateAvg") {
    Campground.find({ })
      .sort({
        rateCount: -1,
        rateAvg: -1
      })
      .exec(function(err, allCampgrounds) {
        if (err) {
          console.log(err);
        } else {
          res.render("campgrounds/index", {
            campgrounds: allCampgrounds,
            currentUser: req.user,
            noMatch: noMatch
          });
        }
      });
    }
  }
}

```

```

    });
  }
});

} else if (req.query.sortby === "rateCount") {
  Campground.find({ })
    .sort({
      rateCount: -1
    })
    .exec(function(err, allCampgrounds) {
      if (err) {
        console.log(err);
      } else {
        res.render("campgrounds/index", {
          campgrounds: allCampgrounds,
          currentUser: req.user,
          noMatch: noMatch
        });
      }
    });
} else if (req.query.sortby === "priceLow") {
  Campground.find({ })
    .sort({
      price: 1,
      rateAvg: -1
    })
    .exec(function(err, allCampgrounds) {
      if (err) {
        console.log(err);
      } else {
        res.render("campgrounds/index", {

```

```

        campgrounds: allCampgrounds,
        currentUser: req.user,
        noMatch: noMatch
    });
}
});
} else {
    Campground.find({ })
        .sort({
            price: -1,
            rateAvg: -1
        })
        .exec(function(err, allCampgrounds) {
            if (err) {
                console.log(err);
            } else {
                res.render("campgrounds/index", {
                    campgrounds: allCampgrounds,
                    currentUser: req.user,
                    noMatch: noMatch
                });
            }
        });
}
} else {
    Campground.find({}, function(err, allCampgrounds) {
        if (err) {
            console.log(err);
        } else {
            res.render("campgrounds/index", {

```

```

        campgrounds: allCampgrounds,
        currentUser: req.user,
        noMatch: noMatch
    });
}
});
}
});

// CREATE - add new campground to db
router.post("/", middleware.isLoggedIn, upload.single("image"), function(
    req,
    res
) {
    cloudinary.v2.uploader.upload(
        req.file.path,
        {
            width: 1500,
            height: 1000,
            crop: "scale"
        },
        function(err, result) {
            if (err) {
                req.flash("error", err.message);
                return res.render("error");
            }
            req.body.campground.image = result.secure_url;
            req.body.campground.imageId = result.public_id;
            req.body.campground.booking = {
                start: req.body.campground.start,

```

```

    end: req.body.campground.end
  };
  req.body.campground.tags = req.body.campground.tags.split(",");
  req.body.campground.author = {
    id: req.user._id,
    username: req.user.username
  };
  geocoder.geocode(req.body.campground.location, function(err, data) {
    if (err || !data.length) {
      console.log(err);
      req.flash("error", "Invalid address");
      return res.redirect("back");
    }
    req.body.campground.lat = data[0].latitude;
    req.body.campground.lng = data[0].longitude;
    req.body.campground.location = data[0].formattedAddress;
    Campground.create(req.body.campground, function(err, campground) {
      if (err) {
        req.flash("error", err.message);
        return res.render("error");
      }
      res.redirect("/campgrounds");
    });
  });
},
{
  moderation: "webpurify"
}
);
});

```

```

// NEW - show form to create new campground
router.get("/new", middleware.isLoggedIn, function(req, res) {
  res.render("campgrounds/new");
});

// SHOW - shows more information about one campground
router.get("/:id", function(req, res) {
  Campground.findById(req.params.id)
    .populate("comments")
    .exec(function(err, foundCampground) {
      if (err || !foundCampground) {
        console.log(err);
        req.flash("error", "Sorry, that campground does not exist!");
        return res.render("error");
      }
      var ratingsArray = [];

      foundCampground.comments.forEach(function(rating) {
        ratingsArray.push(rating.rating);
      });
      if (ratingsArray.length === 0) {
        foundCampground.rateAvg = 0;
      } else {
        var ratings = ratingsArray.reduce(function(total, rating) {
          return total + rating;
        });
        foundCampground.rateAvg = ratings / foundCampground.comments.length;
        foundCampground.rateCount = foundCampground.comments.length;
      }
    });
});

```



```

    foundCampground.save();
    res.render("campgrounds/show", {
      campground: foundCampground
    });
  });
});

// EDIT CAMPGROUND ROUTE
router.get(
  "/:id/edit",
  middleware.isLoggedIn,
  middleware.checkCampgroundOwnership,
  function(req, res) {
    res.render("campgrounds/edit", {
      campground: req.campground
    });
  }
);

// UPDATE CAMPGROUND ROUTE
router.put(
  "/:id",
  upload.single("image"),
  middleware.checkCampgroundOwnership,
  function(req, res) {
    geocoder.geocode(req.body.campground.location, function(err, data) {
      if (err || !data.length) {
        req.flash("error", "Invalid address");
        return res.redirect("back");
      }
    });
  }
);

```

```

req.body.campground.lat = data[0].latitude;
req.body.campground.lng = data[0].longitude;
req.body.campground.location = data[0].formattedAddress;
req.body.campground.booking = {
  start: req.body.campground.start,
  end: req.body.campground.end
};
req.body.campground.tags = req.body.campground.tags.split(",");
Campground.findByIdAndUpdate(
  req.params.id,
  req.body.campground,
  async function(err, campground) {
    if (err) {
      req.flash("error", err.message);
      res.redirect("back");
    } else {
      if (req.file) {
        try {
          await cloudinary.v2.uploader.destroy(campground.imageId);
          var result = await cloudinary.v2.uploader.upload(
            req.file.path,
            {
              width: 1500,
              height: 1000,
              crop: "scale"
            },
            {
              moderation: "webpurify"
            }
          );

```

```

        campground.imageId = result.public_id;
        campground.image = result.secure_url;
    } catch (err) {
        req.flash("error", err.message);
        return res.render("error");
    }
}

campground.save();
req.flash("success", "Successfully updated your campground!");
res.redirect("/campgrounds/" + req.params.id);
}
}
);
});
}
);

router.delete("/:id", middleware.checkCampgroundOwnership, function(req, res) {
    Campground.findById(req.params.id, async function(err, campground) {
        if (err) {
            req.flash("error", err.message);
            return res.redirect("back");
        }
        try {
            await cloudinary.v2.uploader.destroy(campground.imageId);
            campground.remove();
            res.redirect("/campgrounds");
        } catch (err) {
            if (err) {
                req.flash("error", err.message);
                return res.render("error");
            }
        }
    });
});

```

```

    }
  }
});
});

```

```
module.exports = router;
```

- **Comments.js**

```

var express = require("express");
var router = express.Router({ mergeParams: true });
var Campground = require("../models/campground");
var Comment = require("../models/comment");
var middleware = require("../middleware");

// Comments new
router.get("/new", middleware.isLoggedIn, function(req, res) {
  Campground.findById(req.params.id, function(err, campground) {
    if (err) {
      console.log(err);
    } else {
      res.render("comments/new", { campground: campground });
    }
  });
});

```

```

// Comments create
router.post("/", middleware.isLoggedIn, function(req, res) {
  Campground.findById(req.params.id, function(err, found) {
    if (err) {
      console.log(err);
    }
  });
});

```

```

}

var ratedArray = [];

found.hasRated.forEach(function(rated) {
    ratedArray.push(String(rated));
});

if (ratedArray.includes(String(req.user._id))) {
    req.flash(
        "error",
        "You've already reviewed this campground, please edit your review instead."
    );
    res.redirect("/campgrounds/" + req.params.id);
} else {
    Campground.findById(req.params.id, function(err, campground) {
        if (err) {
            console.log(err);
            res.redirect("/campgrounds");
        } else {
            var newComment = req.body.comment;
            Comment.create(newComment, function(err, comment) {
                if (err) {
                    req.flash("error", "Something went wrong.");
                    res.render("error");
                } else {
                    // add username and id to comment
                    comment.author.id = req.user._id;
                    comment.author.username = req.user.username;
                    campground.hasRated.push(req.user._id);
                    campground.rateCount = campground.comments.length;
                    // save comment
                    comment.save();
                }
            });
        }
    });
}

```

```

        campground.comments.push(comment);
        campground.save();
        req.flash("success", "Successfully added review!");
        res.redirect("/campgrounds/" + campground._id);
    }
});
}
});
}
});
});
});

```

// COMMENT EDIT ROUTE

```

router.get(
    "/:comment_id/edit",
    middleware.isLoggedIn,
    middleware.checkCommentOwnership,
    function(req, res) {
        res.render("comments/edit", {
            campground_id: req.params.id,
            comment: req.comment
        });
    }
);

```

// COMMENT UPDATE ROUTE

```

router.put("/:comment_id", middleware.checkCommentOwnership, function(
    req,
    res
) {

```

```

Comment.findByIdAndUpdate(req.params.comment_id, req.body.comment, function(
  err,
  updatedComment
) {
  if (err) {
    res.redirect("back");
  } else {
    req.flash("success", "Review updated!");
    res.redirect("/campgrounds/" + req.params.id);
  }
});
});

```

// DESTROY COMMENT ROUTE

```

router.delete("/:comment_id", middleware.checkCommentOwnership, function(
  req,
  res
) {
  Comment.findByIdAndRemove(req.params.comment_id, function(err) {
    if (err) {
      res.redirect("back");
    } else {
      Campground.findByIdAndUpdate(
        req.params.id,
        { $pull: { comments: { $in: [req.params.comment_id] } } },
        function(err) {
          if (err) {
            console.log(err);
          }
        }
      )
    }
  })
}

```

```

);

Campground.findByIdAndUpdate(
  req.params.id,
  { $pull: { hasRated: { $in: [req.user._id] } } },
  function(err) {
    if (err) {
      console.log(er);
    }
  }
);

req.flash("success", "Review deleted!");
res.redirect("/campgrounds/" + req.params.id);
}
});
});

module.exports = router;

```


CHAPTER 8

TESTING

8.1 Introduction of Testing

Software testing is a critical element of software quality assurance and represents the ultimate reuse of specification. Design and code testing represents interesting anomaly for the software during earlier definition and development phase, it was attempted to build software from an abstract concept to tangible implementation.

Testing plays a critical role in quality assurance for software. Due to the limitation of the verification method for the previous phases, design and requirement faults also appear in the code. Testing is used to detect these errors, in addition to the error introduced during coding phase.

There are two methods of testing: **functional** and **structural**. In functional testing, the internal logic of the system under testing is not considered and the test cases are decided from the specification or the requirements. It is often called "Black Box Testing". Equivalence class partitioning, boundary analysis, and cause effect graphing are examples of methods for selecting test cases for functional testing. In structural testing, the test cases are decided entirely on the internal logic of the program or module being tested.

The goal here is to test the system design. In system testing and acceptance testing, the entire system is tested. The goal here is to test the requirements themselves. Structural testing can be used for unit testing while at higher level mostly functional testing is used.

System testing is a critical phase in systems implementation. Testing of a system hardware device testing and debugging of computer programs and testing information processing procedures. Testing can be done with test data, which attempts to simulate all possible conditions that may arise during processing. The plans for testing are prepared and then implemented.

8.2 Principles of Testing:

- (i) All the test should meet party the customer requirements.
- (ii) To make our software testing should be performed by third party
- (iii) Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk Assessment of the application.
- (iv) All the test to be conducted should be planned before implementing it.
- (v) It follows Pareto Rule (80/20 rule) which states that 80% of errors comes from 20% of components.
- (vi) Start testing with small parts and extend it to large parts.

8.3 Types of Testing:

8.3.1 Unit Testing-

It focuses on smallest unit of software design. In this we test an individual unit or group of inters related units. It is often done by programmer by using sample input and observing its corresponding outputs.

Example:

- a) In a program we are checking if loop, method or function is working fine
- b) Misunderstood or incorrect, arithmetic precedence.
- c) Incorrect initialization.

8.3.2 Integration Testing-

The objective is to take unit tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components are combined to produce output.

8.3.3 Alpha Testing-

This is a type of validation testing. It is a type of acceptance testing which is done before the product is released to customers. It is typically done by QA people. Example: When software testing is performed internally within the organization.

8.3.4 Beta Testing-

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for the limited number of users for testing in real time environment.

Example: When software testing is performed for the limited number of people.

8.3.5 System Testing-

In this software is tested such that it works fine for different operating system .It is covered under the black box testing technique. In this we just focus on required input and output on internal. In this we have security testing, recovery testing stress testing and performance testing without focusing on internal working. In this we have security testing, recovery testing, stress testing and performance testing. Example: This include functional as well as non-functional Testing

8.3.6 Performance Testing-

It is designed to test the run-time performance of software within the context of an integrated system. It is used to test speed and effectiveness of program. Example: Checking number of processor cycle.

8.4 Levels Of Testing-

In order to uncover the errors present in different phases, we have the concept of levels of testing. The basic levels of testing are-

8.4.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software i.e. the module Using the detailed design and the process specifications, testing is done to uncover errors within the boundary of the module. All modules must be successful in the unit test before the start of the integration testing begins.

In this project each service can be thought of a module. There are so many modules like Login and Registration, Admin, Candidate, Exam and Report Module. Each module has been tested by giving different sets of inputs. When developing the module as well as finishing the development, the module works without any error. The inputs are validated when accepting them from the user.

8.4.2 Integration Testing

After unit testing, we have to perform integration testing. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules. This testing activity can be considered as testing the design and hence the emphasis on testing module interactions.

In this project the main system is formed by integrating all the modules. When integrating all the modules I have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before Integration.

8.4.3 System Testing

Here the entire software system is tested. The reference document for this process is the requirements document, and the goal is to see if software meets its requirements.

Here entire 'HRRP has been tested against requirements of project and it is checked whether all requirements of project have been satisfied or not.

8.4.4 Acceptance Testing

Acceptance Testing is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behavior of the system: the internal logic of program is not emphasized.

- Test cases should be selected so that the largest number of attributes of an equivalence class is exercised at once. The testing phase is an important part of software development. It is the process of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied.

8.5 Test Cases

A Test Case is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, pre-condition, post-condition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

A test case is a detailed specification of inputs, execution conditions, expected outcomes, and test procedures designed to validate a specific functionality or aspect of a software website. It serves as a set of instructions or guidelines for testers to follow during the testing process.

A test case typically consists of several components. Firstly, it includes the preconditions, which outline the necessary setup or state required for the test to be executed. Secondly, it specifies the input values or actions that need to be provided to the system under test. These inputs can range from user interactions to data inputs or system configurations.

Next, the test case defines the expected outcomes or results based on the specified inputs. It describes the anticipated behavior or response of the website, such as correct data display, error messages, or successful completion of a task. Additionally, the test case includes the steps or procedures that testers should follow to execute the test accurately.

Test cases should be comprehensive, covering different scenarios and edge cases to ensure maximum coverage. They should be clear, concise, and unambiguous, enabling testers to perform the test consistently and accurately. Test cases play a crucial role in ensuring the thoroughness and effectiveness of the testing process, aiding in the identification of defects and validating the website's functionality.

8.5 Test Cases for Camp Guide

Test Case No.	Input	Expected Behavior	Observed Behavior	Status P=Passed F=Failed
1.	Register as new User	Registration Page should be displayed	-do-	P
2.	Register with empty fields	Error message should warn to fill required details	-do-	P
3.	Login as user with wrong credentials	Error message to for invalid details	-do-	P
4.	Add Campground without required details	Alert message should appear for required details	-do-	P
5.	Add a Campground with required details	Added message should appear	-do-	P
6.	Search non-added Campground	No results message should appear	-do-	P
7.	Add review on a user's campground	Review should appear in the comment section	-do-	P
8.	Add review on other user's campground	Not Allowed warning should appear	-do-	P

CHAPTER 9

CONCLUSION & FUTURE SCOPE

9.1 Conclusion

This project will help people to get feedback about the camp. User will register and login to the system where user can enter the login details. User need to verify their account. After Authentication the user can add reviews about campgrounds, edit it and delete it. The other users can then view the rating and reviews of the particular or all nearby campgrounds. This website will be more convenient for users for searching and validating the camp ground.

9.2 Future Scope

This project can be hosted on the internet for public use. The need to improve the User Interface and make it more interactive and hassle free. Keeping the security till date could be a challenge to maintain. Hence, improving the overall security of the system and all users. Expanding the storage for storing hi-res images and optimizing it to be efficient for both storing and retrieving. Contact with tour guide feature can be added to improve the user experience before planning the tour. Chat bot feature can also be added to increase the reliability of the website.

CHAPTER 9

REFERENCES

- <https://docs.mongodb.com/>
- <https://expressjs.com/>
- <https://www.javatpoint.com/html-tutorial/>
- <https://www.javatpoint.com/javascript-tutorial/>
- <https://www.javatpoint.com/mongodb-tutorial/>
- <https://www.javatpoint.com/nodejs-tutorial/>
- <https://stackoverflow.com/>
- JavaScript and JQuery: Interactive Front-End Web Development Book by Jon Duckett
- HTML & CSS: Design and Build Web Sites Book by Jon Duckett