

A
REPORT
On
WEB DEVELOPMENT

Submitted
In partial fulfilment
For the award of the Degree of

Submitted By:
T.Koushik Raj
(221710301058)
CSE

Submitted To:

Dhyanahitha
School of
Professional
Studies

Department of Computer Science and
Engineering

Candidate's Declaration

I, hereby declare that the project report "KAPS Broadband Services" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the certificate of internship. The work has not been submitted to any other college or University for the award of any degree or diploma.

(Name and Signature of Candidate)

**T. Koushik Raj
(221710301058)
CSE**

Counter Signed by

ACKNOWLEDGEMENT

My project would not have been successful without the help of several people. I would like to thank the personalities who were part of my project in numerous ways, those who gave me outstanding support from the birth of the project.

I am extremely thankful to our honorable Pro-Vice-Chancellor, Prof. N. Siva Prasad for providing necessary infrastructure and resources for the accomplishment of my project.

I am highly indebted to Prof. N. Seetha Ramaiah, Principal, School of Technology, for his support during the tenure of the project.

I am very much obliged to our beloved Prof. S. Phani Kumar, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this project and encouragement in the completion of this project.

I hereby wish to express my deep sense of gratitude to Mr. Anil Kumar, for the esteemed guidance, moral support and invaluable advice provided by him for the success of the project.

I am also thankful to all the staff and mentors of the Dhyanahitha School of Professional Studies who have cooperated in making my project a success. I would like to thank my parents and friends who extended their help, encouragement and moral support either directly or indirectly in my project work.

Sincerely,
T.Koushik Raj
(221710301058)
CSE

ABSTRACT

As we all are in the pandemic situation from last 5 months , so every employee who are working from their home are likely to choose a fastest and unlimited internet services. Generally now a days internet is very cheaper and each and every citizen has a smart mobile with the internet access but this mobile internet is not sufficient to do office work,so we are introducing a fastest and quality network to all the employees who are working from their home.

Table of Content

	Title	Page No.
	List of Tables.....	
	List of Figures	
1.	Introduction.....	8-9
1.1	Objectives	
2.	Web Development.....	9-11
2.1	Web Site	9-10
2.2	Web Page	10
2.3	Web Browser	11
2.4	Mobile Web Browser	11
3.	The Steps to Create a Web site.....	12-22
3.1	UI development	13-18
	3.1.1 HTML	13-14
	3.1.2 CSS	14-17
	3.1.3 Bootstrap	17-18
3.2	Scripting	19-21
	3.2.1 Server-side scripting	19-20
	3.2.1.1 Server-side scripting languages	19-20
	3.2.2 Client-side scripting	20-21
3.3	Database	21
3.4	SQL	21
3.5	Queries	22
4.	Scripting Languages.....	23-27
4.1	PHP	23-24
4.2	Installing PHP	24
4.3	Java Script	25
4.4	jQuery	25-26
4.5	AJAX	26
4.6	JSON	26
4.7	XAMPP	27
4.8	Features	27
4.9	Usage	27
5.	Software Requirement Specification (SRS).....	28
5.1	Hardware Requirement	28
5.2	Software Requirement	28
6.	Data Flow Diagram.....	29-33

	6.1 DFD-1	29
	6.2 DFD-2	30-31
	6.3 DFD-3(UML Diagram)	32
	6.4 ER-Diagram	33
7.	Project.....	34
	7.1 Technologies Used	34
	7.2 Technical details	34
8.	Screenshots.....	35-61
	8.1 Sign in	35-38
	8.2 Sign up	38-43
	8.3 Forgot Password	44-46
	8.4 Renewal	47-49
	8.5 Registration	49-51
	8.6 Feedback	52-54
	8.7 Debit/Credit	54-57
	8.8 Net banking	58-61
9.	Maintenance.....	62
10.	Future scope and future enhancement.....	63
11.	Conclusion.....	63
	Bibliography	63-64

List of Figures

Figure 3.1	Steps to create a website.....	12
Figure 3.2	Login form without CSS ...	16
Figure 3.3	Login form without CSS.	17
Figure 3.2.1	Programming language popularity.	20
Figure 3.2.2	Scripting	21
Figure 4.1	PHP & XAMPP... ..	24
Figure 6.1	DFD.	29-33
Figure 6.1.1	DFD-1.....	29
Figure 6.2.1	DFD-2.....	30-31
Figure 6.2.2	DFD-3	32
Figure 6.3.1	ER-Diagram	33
Figure 8.1	Sign in.....	38
Figure 8.2	Sign up.....	43
Figure 8.3	Forgot Password.....	46
Figure 8.4	Renewal.....	49
Figure 8.5	Registration	51
Figure 8.6	Feedback.....	54
Figure 8.7	Debit/Credit Card.....	57
Figure 8.8	Net banking.....	61

CHAPTER-1

INTRODUCTION

The industry definition of a Full Stack Developer is an engineer who can work on different levels of an application stack. The term stack refers to the combination of components and tools that make up the application. The components could be in the front-end or the back-end of the system.

The main objective of a full stack engineer is to keep every part of the system running smoothly. A Full Stack Developer can perform tasks ranging from resizing an image or text in a webpage to patching the kernel.

Full stack development: It refers to the development of both front end (client side) and back end(server side) portions of web application.

Full stack web Developers: Full stack web developers have the ability to design complete web applications and websites. They work on the frontend, backend, database and debugging of web applications or websites.

2.4 FRONT-END

Front-end web development, also known as client-side development is the practice of producing HTML, CSS and JavaScript for a website or Web Application so that a user can see and interact with them directly. The challenge associated with front end development is that the tools and techniques used to create the front end of a website change constantly and so the developer needs to constantly be aware of how the field is developing.

The objective of designing a site is to ensure that when the users open up the site they see the information in a format that is easy to read and relevant. This is further complicated by the fact that users now use a large variety of devices with varying screen sizes and resolutions thus forcing the designer to take into consideration these aspects when designing the site. They need to ensure that their site comes up correctly in different browsers (cross-browser), different operating systems (cross-platform) and different devices (cross-device), which requires careful planning on the side of the developer.

Front end development manages everything that users visually see first in their browser or application. Front end developers are responsible for the look and feel of a site. It is the visible part of a website or web application which is responsible for user experience. The user directly interacts with the front-end portion of the web application or website.

1.2 BACK-END

Back end development refers to the server side of an application and everything that communicates between the database and the browser. It is responsible for managing the database through queries and APIs by client-side commands. Back end development refers to the server side of development where you are primarily focused on how the site works. Making updates and changes in addition to monitoring functionality of the site will be your primary responsibility. This type of web development usually consists of three parts: a server, an application, and a database. Code written by back end developers is what communicates the database information to the browser. Anything you can't see easily with the eye such as databases and servers is the work of a back-end developer. Back end developer positions are often called programmers or web developers.

CHAPTER-2 WEB-DEVELOPMENT

Web development is a broad term for the work involved in developing a website for the Internet (World Wide Web) or an intranet (a private network). Web development can range from developing the simplest static single page of plain text to the most complex web-based internet applications, electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers, may include web engineering, web design, web content development, client liaison, client-side/site scripting, web server and network security configuration, and e-commerce development. Among web professionals, "web development" usually refers to the main non-design aspects of building web sites: writing markup and coding. Most recently Web development has come to mean the creation of content management systems or CMS. These CMS can be made from scratch, proprietary or open source. In broad terms the CMS acts as middleware between the database and the user through the browser. A principle benefit of a CMS is that it allows non-technical people to make changes to their website without having technical knowledge.

For larger organizations and businesses, web development teams can consist of hundreds of people (web developers) and follow standard methods like Agile methodologies while developing websites. Smaller organizations may only require a single permanent or contracting developer, or secondary assignment to related job positions such as a graphic designer or information systems technician. Web development may be a collaborative effort between departments rather than the domain of a designated department. There are three kinds of web developer specialization: front-end developer, back-end developer, and full-stack developer.

2.1 WEB-SITE

A **website** is a collection of related web pages, including multimedia content, typically identified with a common domain name, and published on at least one web server. A website may be accessible via a public Internet Protocol (IP) network, such as the Internet, or a private local area network (LAN), by referencing a uniform resource locator (URL) that identifies the site.

Websites have many functions and can be used in various fashions; a website can be a personal website, a commercial website for a company, a government website or a non-profit organization website. Websites are typically dedicated to a particular topic or purpose, ranging from entertainment and social networking to providing news and education. All publicly accessible websites collectively constitute the World Wide Web, while private websites, such as a company's website for its employees, and are typically a part of an intranet.

Web pages, which are the building blocks of websites, are documents, typically composed in plain text interspersed with formatting instructions of Hypertext Markup Language (HTML, XHTML). They may incorporate elements from other websites with suitable markup anchors. Web pages are accessed and transported with the Hypertext Transfer Protocol (HTTP), which may optionally employ encryption (HTTP Secure, HTTPS) to provide security and privacy for the user. The user's application, often a web browser, renders the page content according to its HTML markup instructions onto a display terminal.

Hyperlinking between web pages conveys to the reader the site structure and guides the navigation of the site, which often starts with a home page containing a directory of the site web content. Some websites require user registration or subscription to access content. Examples of subscription websites include many business sites, news websites, academic journal websites, gaming websites, file-sharing websites, message boards, web-based email, social networking websites, websites providing real-time stock market data, as well as sites providing various other services. As of 2016 end users can access websites on a range of devices, including desktop and laptop computers, tablet computers, smartphones and smart TVs. A web site consists of web pages which are interconnected to each other and contain various data and functionalities.

2.2 WEB-PAGE

A **web page**, or **webpage**, is a document that is suitable for the World Wide Web and web browsers. A web browser displays a web page on a monitor or mobile device. The web page is what displays, but the term also refers to a computer file, usually written in HTML or comparable markup language. Web browsers coordinate the various web resource elements for the written web page, such as style sheets, scripts, and images, to present the web page.

Typical web pages provide hypertext that includes a navigation bar or a sidebar menu to other web pages via hyperlinks, often referred to as links.

On a network, a web browser can retrieve a web page from a remote web server. On a higher level, the web server may restrict access to only a private network such as a corporate intranet or it provides access to the World Wide Web. On a lower level, the web browser uses

the Hypertext Transfer Protocol (HTTP) to make such requests.

A static web page is delivered exactly as stored, as web content in the web server's file system, while a dynamic web page is generated by a web application that is driven by server-side software or client-side scripting. Dynamic website pages help the browser (the client) to enhance the web page through user input to the server.

2.3 WEB BROWSER

A **web browser** (commonly referred to as a **browser**) is a software application for accessing information on the World Wide Web. When a user requests a web page from a particular website, the web browser retrieves the necessary content from a web server and then displays the page on the user's device.

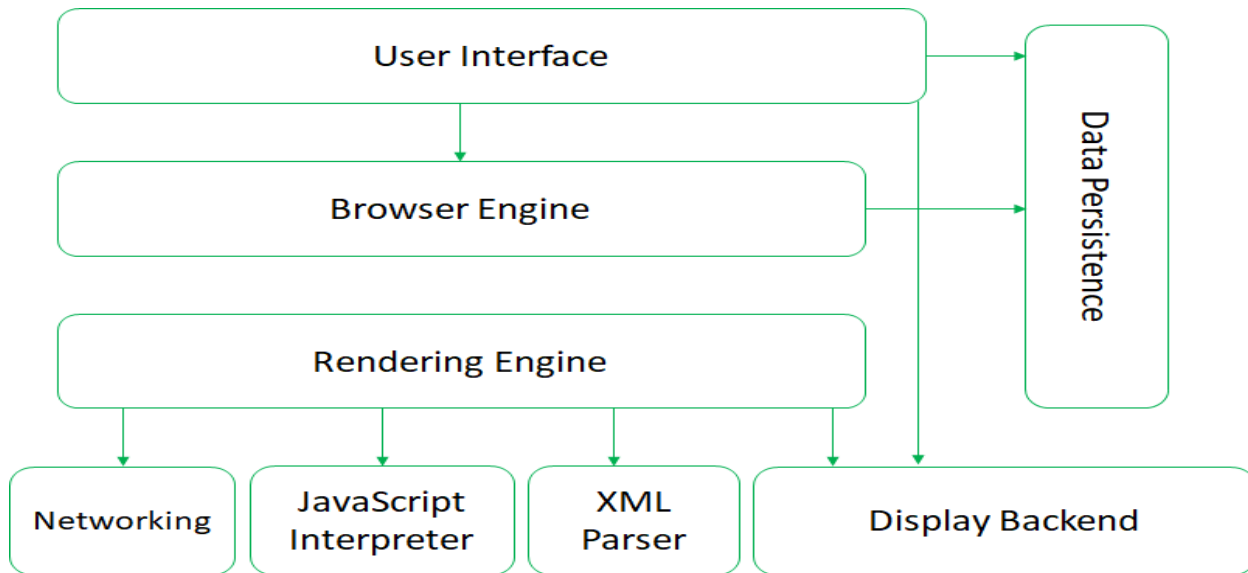
A web browser is not the same thing as a search engine, though the two are often confused. For a user, a search engine is just a website that stores searchable data about other websites. However, to connect to a website's server and display its web pages, a user must have a web browser installed.

Web browsers are used on a range of devices, including desktops, laptops, tablets, and smartphones. In 2019, an estimated 4.3 billion people used a browser. The most used browser is Google Chrome, with a 64% global market share on all devices, followed by Safari with 18%.

2.4 MOBILE WEB BROWSER

A **mobile browser** is a web browser designed for use on a mobile device such as a mobile phone or PDA. Mobile browsers are optimized so as to display Web content most effectively for small screens on portable devices. Mobile browser software must be small and efficient to accommodate the low memory capacity and low-bandwidth of wireless handheld devices. Typically, they were stripped-down web browsers, however, some recent mobile browsers can handle latest technologies also such as CSS 3, JavaScript, and Ajax.

Websites designed so that they may be accessed from these browsers are referred to as *wireless portals* or collectively as the Mobile Web. They may automatically create "mobile" version of each page, for example the Wikipedia website.



CHAPTER-3

THE STEPS TO CREATE A WEBSITE

Creating a web site requires multiple steps which includes the following:

- Creating a UI (User interface)
- Scripting (Both at server end and client end)
- Creating a backend or the database

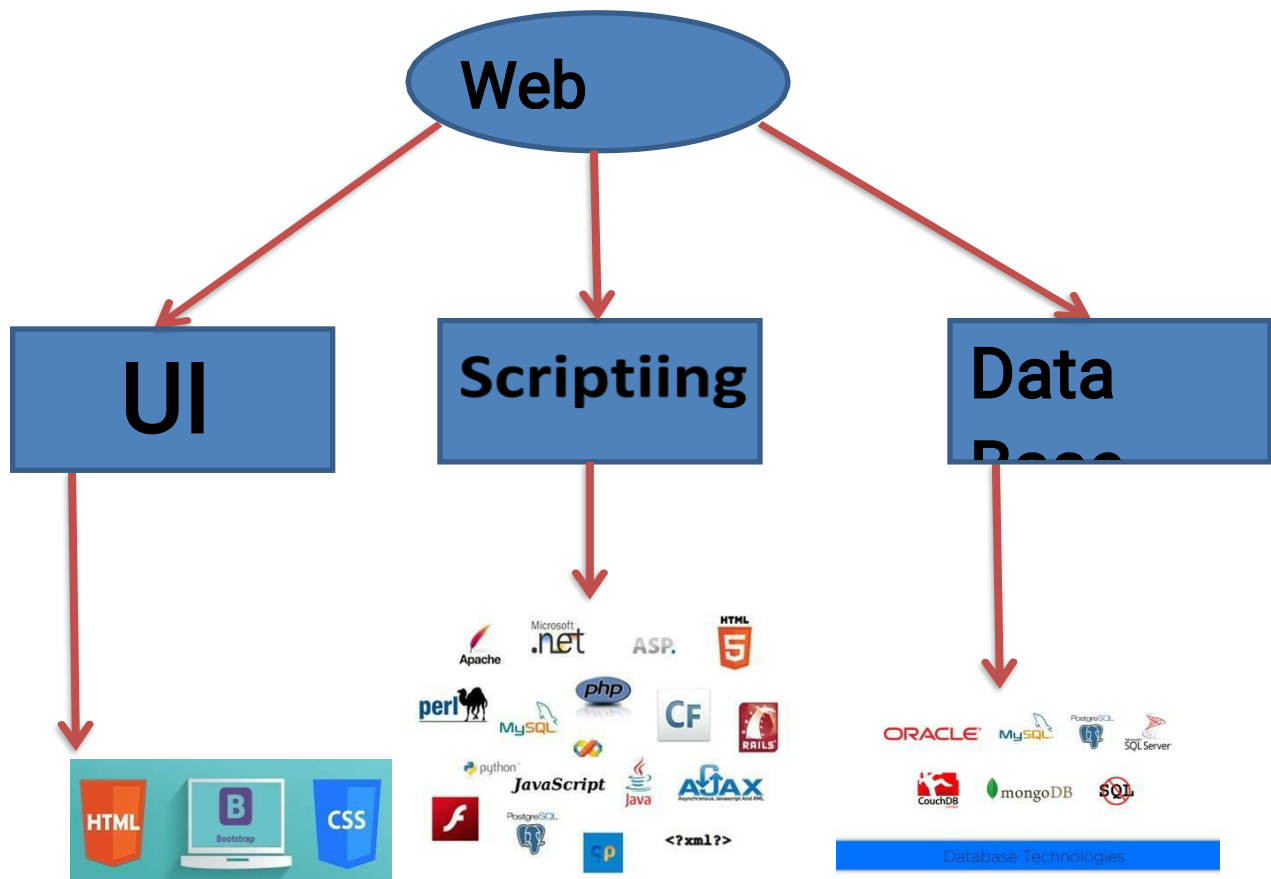


Fig 3.1

3.1 UI DEVELOPMENT

Technologies that are mostly used to develop a User Interface are:

- HTML
- CSS
- Bootstrap

3.1.1 HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript it forms a triad of cornerstone technologies for the World Wide Web. Web browsers receive HTML documents from a web server or from local storage and render them into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms, may be embedded into the rendered page. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as `` and `<input />` introduce content into the page directly. Others such as `<p>...</p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript which affect the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), maintainer of both the HTML and the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

HTML markup consists of several key components, including those called tags (and their attributes), character-based data types, character references and entity references. HTML tags most commonly come in pairs like `<h1>` and `</h1>`, although some represent empty elements and so are unpaired, for example ``. The first tag in such a pair is the start tag, and the second is the end tag (they are also called opening tags and closing tags). Another important component is the HTML document type declaration, which triggers standards mode rendering.

The following is an example of the classic Hello world program, a common test employed for comparing programming languages, scripting languages and markup languages. This example is made using 9 lines of code:

General Syntax of HTML

```
<!DOCTYPE html>
<html>
<head>
<title>This is a title</title>
</head>
<body>
<p>Hello world! </p>
</body>
</html>
```

(The text between <html> and </html> describes the web page, and the text between <body> and </body> is the visible page content. The markup text "<title>This is a title</title>" defines the browser page title.)

The Document Type Declaration <!DOCTYPE html> is for HTML5. If a declaration is not included, various browsers will revert to "quirks mode" for rendering.

3.1.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging web pages, user interfaces for web applications, and user interfaces for many mobile applications.

CSS is designed primarily to enable the separation of presentation and content, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. It can also display

the web page differently depending on the screen size or viewing device. Readers can also specify a different style sheet, such as a CSS file stored on their own computer, to override the one the author specified.

Changes to the graphic design of a document (or hundreds of documents) can be applied quickly and easily, by editing a few lines in the CSS file they use, rather than by changing markup in the documents.

The CSS specification describes a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities (or weights) are calculated and assigned to rules, so that the results are predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/CSS is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

Types of CSS:

- **Inline CSS:** In this CSS is applied in between the tags

E.g.: `<tag style=" styling">Hello World</tag>`

- **Internal CSS:**

In this code is defined inside the style tag in the head section of the HTML page.

General Syntax:

```
<html>

  <head>

    <style>

      <!-- CSS STYLING -->

    </style>

  </head>

</html>
```


- **External CSS:**

In this the CSS code is written on another page and is linked to the HTML page. It is advantageous to use this type of styling as we can use the same file to style various HTML pages.

External CSS uses the extension .CSS and is applied using the following syntax:

```
<html>

  <head>

    <link relation=" stylesheet" type="CSS" href="URL to the page">

  </head>

</html>
```

All the CSS style types are important but can be used in different situations.

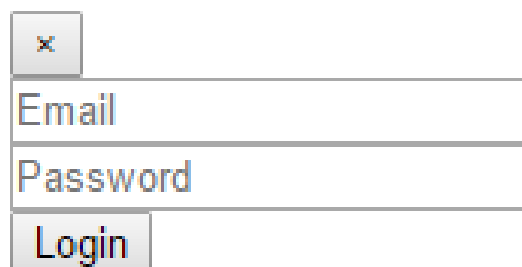
- Inline CSS is used when only small changes are to be done to the HTML tag and the changes are to be reflected only to that specific tag
- Internal CSS is used when the individual HTML pages have to be designed differently. This also slows the page load system if the internal styling is long.
- External CSS files are maintained to design multiple pages and use common styles over various pages. It is useful as it helps in managing the resources in an easy manner.

Both HTML and CSS are used to create a UI but CSS behaves like a makeup on the face of an actress which makes her look even more beautiful than she is in reality.

And here is the difference:

Before using CSS in HTML page:

Enter your account details to login!



A simple login form with a close button (X) in the top left corner. Below it are two input fields: one labeled 'Email' and one labeled 'Password'. At the bottom is a 'Login' button.

Fig 3.2

After using CSS in HTML Page:



Fig 3.3

3.1.3 BOOTSTRAP

Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.

Bootstrap is the second most-starred project on GitHub, with more than 107,000 stars and 48,000 forks.

Bootstrap, originally named Twitter Blueprint, was developed by Mark Otto and Jacob Thornton at Twitter as a framework to encourage consistency across internal tools. Before Bootstrap, various libraries were used for interface development, which led to inconsistencies and a high maintenance burden. According to twitter developer Mark Otto:

“A super small group of developers and I got together to design and build a new internal tool and saw an opportunity to do something more. Through that process, we saw ourselves build something much more substantial than another internal tool. Months later, we ended up with an early version of Bootstrap as a way to document and share common design patterns and assets within the company.”

After a few months of development by a small group, many developers at Twitter began to contribute to the project as a part of Hack Week, a hackathon-style week for the Twitter development team. It was renamed from Twitter Blueprint to Bootstrap, and released as an

open source project on August 19, 2011. It has continued to be maintained by Mark Otto, Jacob Thornton, and a small group of core developers, as well as a large community of contributors.

On January 31, 2012, Bootstrap 2 was released, which added a twelve-column responsive grid layout system, inbuilt support for Glyph icons, several new components, as well as changes to many of the existing components.

On August 19, 2013, Bootstrap 3 was released, which redesigned components to use flat design, and a mobile first approach.

On October 29, 2014, Mark Otto announced that Bootstrap 4 was in development. The first alpha version of Bootstrap 4 was released on August 19, 2015.

Bootstrap 3 supports the latest versions of Google Chrome, Firefox, Internet Explorer, Opera, and Safari (except on Windows). It additionally supports back to IE8 and the latest Firefox Extended Support Release (ESR).

Since 2.0, Bootstrap supports responsive web design. This means the layout of web pages adjusts dynamically, taking into account the characteristics of the device used (desktop, tablet, mobile phone).

Starting with version 3.0, Bootstrap adopted a mobile-first design philosophy, emphasizing responsive design by default.

The version 4.0 alpha release added Sass and flexbox support.

Installing and linking bootstrap to the HTML page:

Install bootstrap from <https://getbootstrap.com/>

- Copy the bootstrap.min.css file to your CSS folder and link it to the HTML page in the similar manner to how any other CSS file is linked.
- Link the bootstrap.min.js file which is present in the JS folder of the bootstrap. It can be linked using a script tag.

E.g.: `<script src ="URL to bootstrap.min.js"></script>`

- Now use bootstrap classes to reduce the work of designing which was earlier done through CSS.

3.2 SCRIPTING

There are two scripting methodologies.

1. Server-side scripting: This scripting is done at the server end
2. Client-side scripting: This scripting is done at the client end or the browser

3.2.1 SERVER-SIDE SCRIPTING

Server-side scripting is a technique used in web development which involves employing scripts on a web server which produce a response customized for each user's (client's) request to the website. The alternative is for the web server itself to deliver a static web page. Scripts can be written in any of a number of server-side scripting languages that are available (see below). Server-side scripting is distinguished from client-side scripting where embedded scripts, such as JavaScript, are run client-side in a web browser, but both techniques are often used together.

Server-side scripting is often used to provide a customized interface for the user. These scripts may assemble client characteristics for use in customizing the response based on those characteristics, the user's requirements, access rights, etc. Server-side scripting also enables the website owner to hide the source code that generates the interface, whereas with client-side scripting, the user has access to all the code received by the client. A down-side to the use of server-side scripting is that the client needs to make further requests over the network to the server in order to show new information to the user via the web browser. These requests can slow down the experience for the user, place more load on the server, and prevent use of the application when the user is disconnected from the server.

When the server serves data in a commonly used manner, for example according to the HTTP or FTP protocols, users may have their choice of a number of client programs (most modern web browsers can request and receive data using both of those protocols). In the case of more specialized applications, programmers may write their own server, client, and communications protocol that can only be used with one another.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be considered client-side operations.

3.2.1.1 Server-Side scripting Languages

There are several languages that can be used for server-side programming:

- PHP
- ASP.NET (C# OR Visual Basic)
- C++
- Java and JSP
- Python
- Ruby on Rails and so on.

Programming Language Popularity By Github Projects

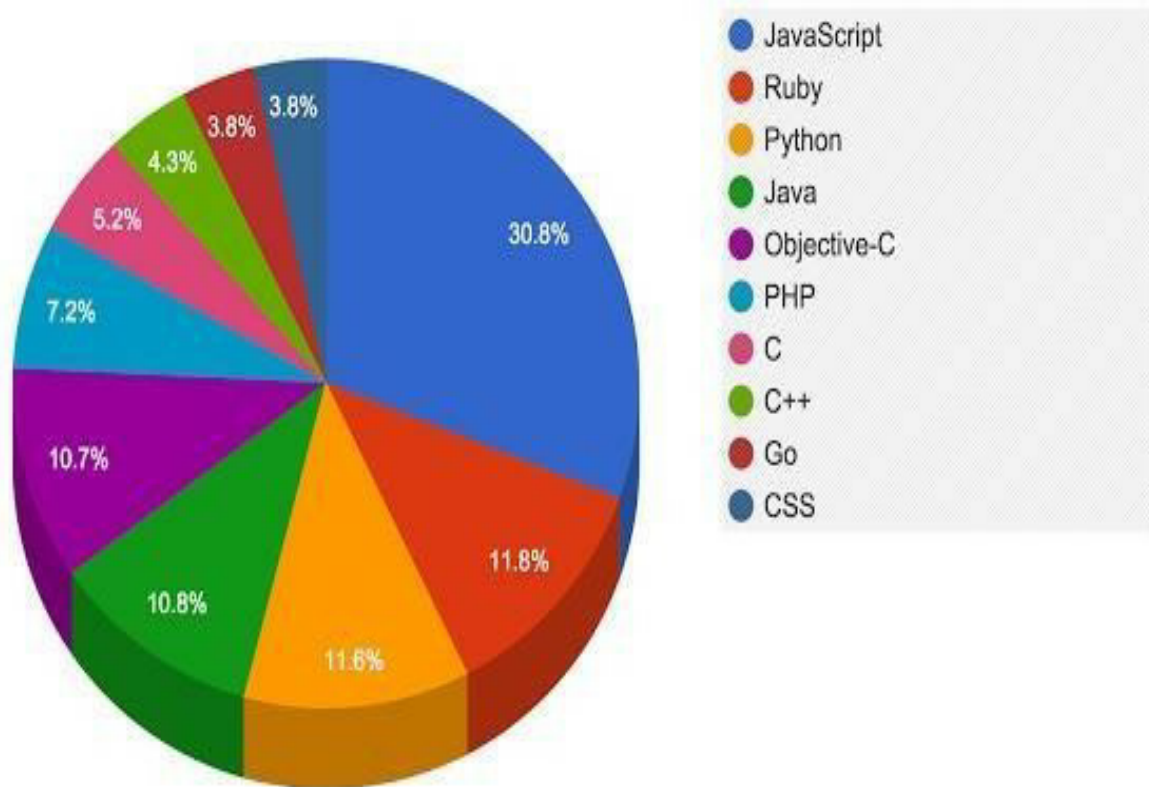


Fig 3.4

3.2.2 CLIENT-SIDE SCRIPTING

Client-side scripting is changing interface behaviors within a specific web page in response to mouse or keyboard actions, or at specified timing events. In this case, the dynamic behavior occurs within the presentation. The client-side content is generated on the user's local computer system.

Such web pages use presentation technology called rich interfaced pages. Client-side scripting languages like JavaScript or ActionScript, used for Dynamic HTML (DHTML) and Flash technologies respectively, are frequently used to orchestrate media types (sound, animations, changing text, etc.) of the presentation. Client-side scripting also allows the use of remote scripting, a technique by which the DHTML page requests additional information from a server, using a hidden frame, XML Http Requests, or a Web service.

The first widespread use of JavaScript was in 1997, when the language was standardized as ECMAScript and implemented in Netscape 3.

Example:

The client-side content is generated on the client's computer. The web browser

retrieves a page from the server, then processes the code embedded in the page (typically written in JavaScript) and displays the retrieved page's content to the user.

The most popularly used client-side scripting languages is **JavaScript**. Flow of request from browser to server:

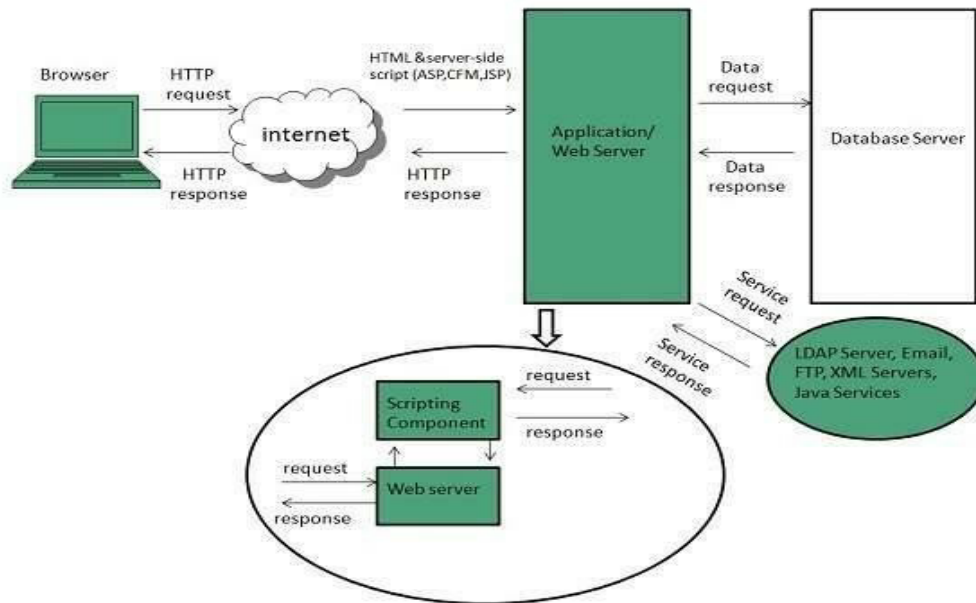


Fig 3.5

3.3 DATABASE

A **database** is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

A **database management system (DBMS)** is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, MongoDB, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, MySQL and IBM DB2. A database is not generally portable across different DBMSs, but different DBMS can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one DBMS. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language. Sometimes a DBMS is loosely referred to as a "database".

3.4 SQL

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

SQL was one of the first commercial languages for Codd's relational, as described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks." Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

3.5 QUERIES

The most common operation in SQL, the query, makes use of the declarative SELECT statement. SELECT retrieves data from one or more tables, or expressions. Standard SELECT statements have no persistent effects on the database. Some non-standard implementations of SELECT can have persistent effects, such as the SELECT INTO syntax provided in some databases.

Queries allow the user to describe desired data, leaving the database management system (DBMS) to carry out planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to include in the final result, normally immediately following the SELECT keyword. An asterisk ("*") can be used to specify that the query should return all columns of the queried tables. SELECT is the most complex statement in SQL, with optional keywords and clauses that include:

The FROM clause, which indicates the table(s) to retrieve data from. The FROM clause can include optional JOIN subclauses to specify the rules for joining tables.

The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set where the comparison predicate does not evaluate to True.

The GROUP BY clause projects rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.

The HAVING clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.

The ORDER BY clause identifies which column[s] to use to sort the resulting data, and in which direction to sort them (ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

The DISTINCT keyword eliminates duplicate data.

CHAPTER-4

SCRIPTING LANGUAGES

4.1 PHP

Paradigm	Imperative, functional, object-oriented, procedural, reflective
Designed by	RasmusLerdorf
Developer	The PHP Development Team, Zend Technologies
First appeared	June 8, 1995; 21 years ago ^[1]
Stable release	7.1.5 / May 11, 2017; 16 days ago,
Typing discipline	Dynamic, weak, gradual (as of PHP 7.0.0)
Implementation language	C (primarily; some components C++)
OS	Unix-like, Windows
License	PHP License (most of Zend Engine under Zend Engine License& The TSRM License)
Filename extensions	.php, .phtml, .php3, .php4, .php5, .php7,. phps
Website	php.net
Major implementations	
Zend Engine, HHVM, Phalanger, Quercus, Project Zero, Parrot	
Influenced by	
C, C++, Java, Perl, Tcl ^[1]	
Influenced	
Falcon, Hack	

PHP is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language. Originally created by RasmusLerdorf in 1994, the PHP reference implementation is now produced by ThePHP Development Team. PHP originally stood for *Personal Home Page*, but it now stands for the recursive acronym *PHP: Hypertext Preprocessor*.

PHP code may be embedded into HTML or HTML5 markup, or it can be used

in combination with various web template systems, web content management systems and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable. The web server software combines the results of the interpreted and executed PHP code, which may be any type of data, including images, with the generated web page. PHP code may also be executed with a command-line interface (CLI) and can be used to implement standalone graphical.

The standard PHP interpreter, powered by the Zend Engine, is free software released under the PHP License. PHP has been widely ported and can be deployed on most web servers on almost every operating system and platform, free of charge.

The PHP language evolved without a written formal specification or standard until 2014, leaving the canonical PHP interpreter as a *de facto* standard. Since 2014 work has gone on to create a formal PHP specification.

4.2 Installing PHP

- I. Step 1: download the files. Download the latest PHP 5 ZIP package from www.php.net/downloads.php. ...
- II. Step 2: extract the files. ...
- III. Step 3: configure php.ini. ...
- IV. Step 4: add C: php to the path environment variable. ...
- V. Step 5: configure PHP as an Apache module. ...
- VI. Step 6: test a PHP file.
- VII. Or we can install **Xampp** which have inbuilt php, mysql, apache

server We have used xampp to run the php files.

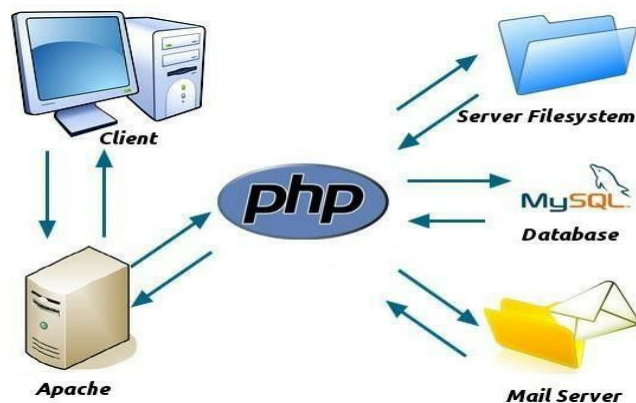


Fig 4.1

4.3 JAVA SCRIPT

JavaScript, often abbreviated as "JS", is a high-level, dynamic, untyped, and interpreted run-time language. It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production; the majority of websites employ it, and all modern Web browsers support it without the need for plug-ins. JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying on these upon the host environment in which it is embedded.

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two are distinct languages and differ greatly in their design. JavaScript was influenced by programming languages such as self and Scheme.

JavaScript is also used in environments that are not Web-based, such as PDF documents, site-specific browsers, and desktop widgets. Newer and faster JavaScript virtual machines (VMs) and platforms built upon them have also increased the popularity of JavaScript for server-side Web applications. On the client side, developers have traditionally implemented JavaScript as an interpreted language, but more recent browsers perform just-in-time compilation. Programmers also use JavaScript in video-game development, in crafting desktop and mobile applications, and in server-side network programming with run-time environments such as Node.js.

4.4 JQUERY

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. It is free, open-source software using the permissive MIT license. Web analysis indicates that it is the most widely deployed JavaScript library by a large margin.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and Web applications.

The set of jQuery core features—DOM element selections, traversal and manipulation—enabled by its selector engine (named "Sizzle" from v1.3), created a new "programming style", fusing algorithms and DOM data structures. This style influenced the architecture of other JavaScript frameworks like YUI v3 and Dojo, later stimulating the creation of the standard Selectors API.

Microsoft and Nokia bundle jQuery on their platforms. Microsoft includes it with Visual Studio for use within Microsoft's ASP.NET AJAX and ASP.NET MVC frameworks while Nokia has integrated it into the Web Run-Time widget development PLATFORM

4.5 AJAX

Ajax (also **AJAX** short for "asynchronous JavaScript and XML") is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications. With Ajax, Web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows for Web pages, and by extension Web applications, to change content dynamically without the need to reload the entire page. In practice, modern implementations commonly substitute JSON for XML due to the advantages of being native to JavaScript.

Ajax is not a single technology, but rather a group of technologies. HTML and CSS can be used in combination to mark up and style information. The DOM is accessed with JavaScript to dynamically display – and allow the user to interact with – the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

4.6 JSON

In computing, **JavaScript Object Notation** or **JSON** is an open- standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format used for asynchronous browser/server communication, including as a replacement for XML in some AJAX-style systems.

JSON is a language-independent data format. It was derived from JavaScript, but as of 2017 many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON file names use the extension .json.

Douglas Crockford originally specified the JSON format in the early 2000s; two competing standards, RFC 7159 and ECMA-404, defined it in 2013. The ECMA standard describes only the allowed syntax, whereas the RFC covers some security and interoperability considerations.^[3]

A restricted profile of JSON, known as **I-JSON** (short for "Internet JSON"), seeks to overcome some of the interoperability problems with JSON. It is defined in RFC 7493.

4.7 XAMPP



Fig 4.2

Xampp is a free and open source cross platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages. XAMPP stands for Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P). It is a simple, lightweight Apache distribution that makes it extremely easy for developers to create a local web server for testing and deployment purposes. Everything needed to set up a web server – server application (Apache), database (MariaDB), and scripting language (PHP) – is included in an extractable file. XAMPP is also cross-platform, which means it works equally well on Linux, Mac and Windows. Since most actual web server deployments use the same components as XAMPP, it makes transitioning from a local test server to a live server extremely easy as well.

4.8 FEATURES

XAMPP is regularly updated to the latest releases of Apache, MariaDB, PHP and Perl. It also comes with a number of other modules including OpenSSL, phpMyAdmin, Media Wiki, Joomla, WordPress and more. Self-contained, multiple instances of XAMPP can exist on a single computer, and any given instance can be copied from one computer to another. XAMPP is offered in both a full and a standard version (Smaller version).

4.9 USAGE

Officially, XAMPP's designers intended it for use only as a development tool, to allow website designers and programmers to test their work on their own computers without any access to the Internet. To make this as easy as possible, many important security features are disabled by default. XAMPP has the ability to serve web pages on the World Wide Web. A special tool is provided to password-protect the most important parts of the package.

XAMPP also provides support for creating and manipulating databases in MariaDB and SQLite among others. Once XAMPP is installed, it is possible to treat a localhost like a remote host by connecting using an FTP client. Using a program like FileZilla has many advantages when installing a content management system (CMS) like Joomla or WordPress. It is also possible to connect to localhost via FTP with an HTML editor.

CHAPTER-5

SOFTWARE REQUIREMENT SPECIFICATION

5.1 Hardware Requirements

The selection of hardware is very important in the existence and proper working of any software. When selecting hardware, the size and requirements are also important.

Processor	Intel CORE i5
RAM	4.0 GB
Hard Disk Drive	500 GB

5.2 Software Requirements

Number	Description
1	Windows 10
2	HTML/CSS/Ajax/JavaScript/ Bootstrap.
3	Apache server/ XAMP SERVER
4	PHP 5.5.38
4	MySQL
5	Compiler: MSVC11 (Visual C++ 2012)
6	Apache version: Apache/2.4.23 (Win32) OpenSSL/1.0.2h PHP/5.5.38

CHAPTER-6

DATA FLOW DIAGRAMS

Data Flow Diagrams show the flow of data from external entities into the system, and from one process to another within the system. There are four symbols for drawing a DFD:

- i. Rectangles representing external entities, which are sources or destinations of data. Ellipses representing processes, which take data as input, validate and process it and output it.
- ii. Arrows representing the data flows, which can either, be electronic data or physical items.
- iii. Open-ended rectangles or a Disk symbol representing data stores, including electronic stores such as databases or XML files and physical stores such as filing cabinets or stacks of paper. Figures below are the Data Flow Diagrams for the current system. Each process within the system is first shown as a Context Level DFD and later as a Detailed DFD.
- iv. The Context Level DFD provides a conceptual view of the process and its surrounding input, output and data stores. The Detailed DFD provides a more detailed and comprehensive view of the interaction among the sub-processes within the system.

Figures below are the Data Flow Diagrams for the current system. Each process within the system is first shown as a Context Level DFD and later as a Detailed DFD. The Context Level DFD provides a conceptual view of the process and its surrounding input, output and data stores. The Detailed DFD provides a more detailed and comprehensive view of the interaction among the sub-processes within the system.

Context Level Diagram

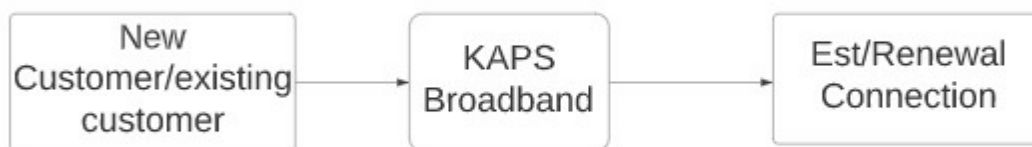


Fig 6.1

LEVEL-I DFD:

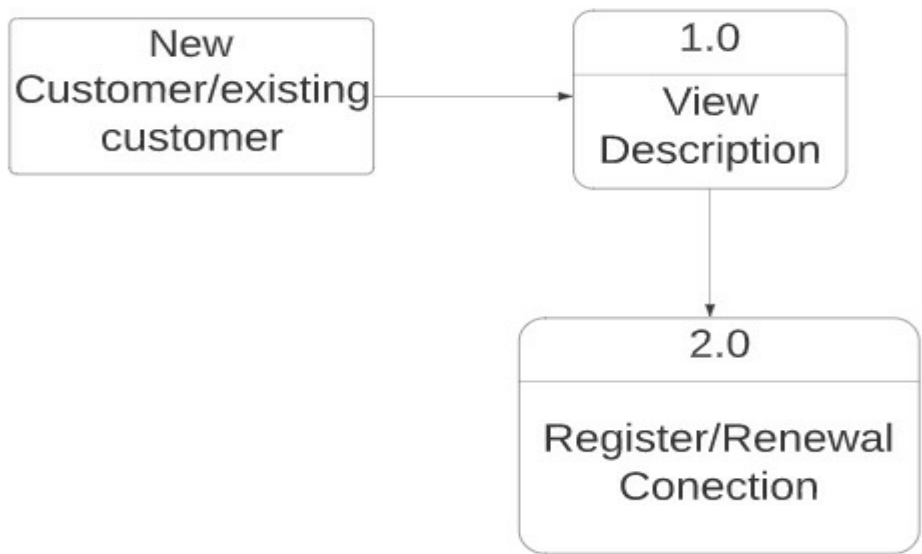


Fig 6.2

LEVEL-II DFD:

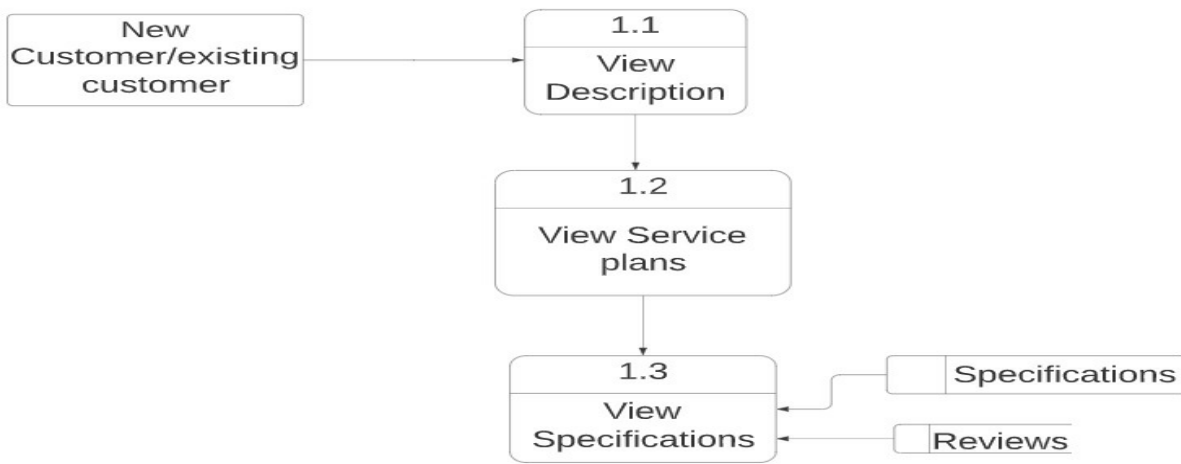


Fig 6.3

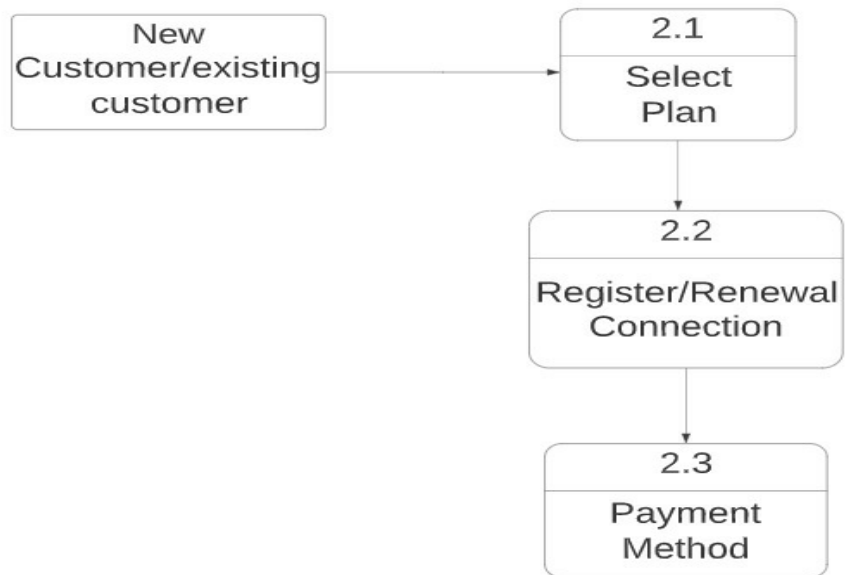


Fig 6.3

UML-USE CASE DIAGRAM

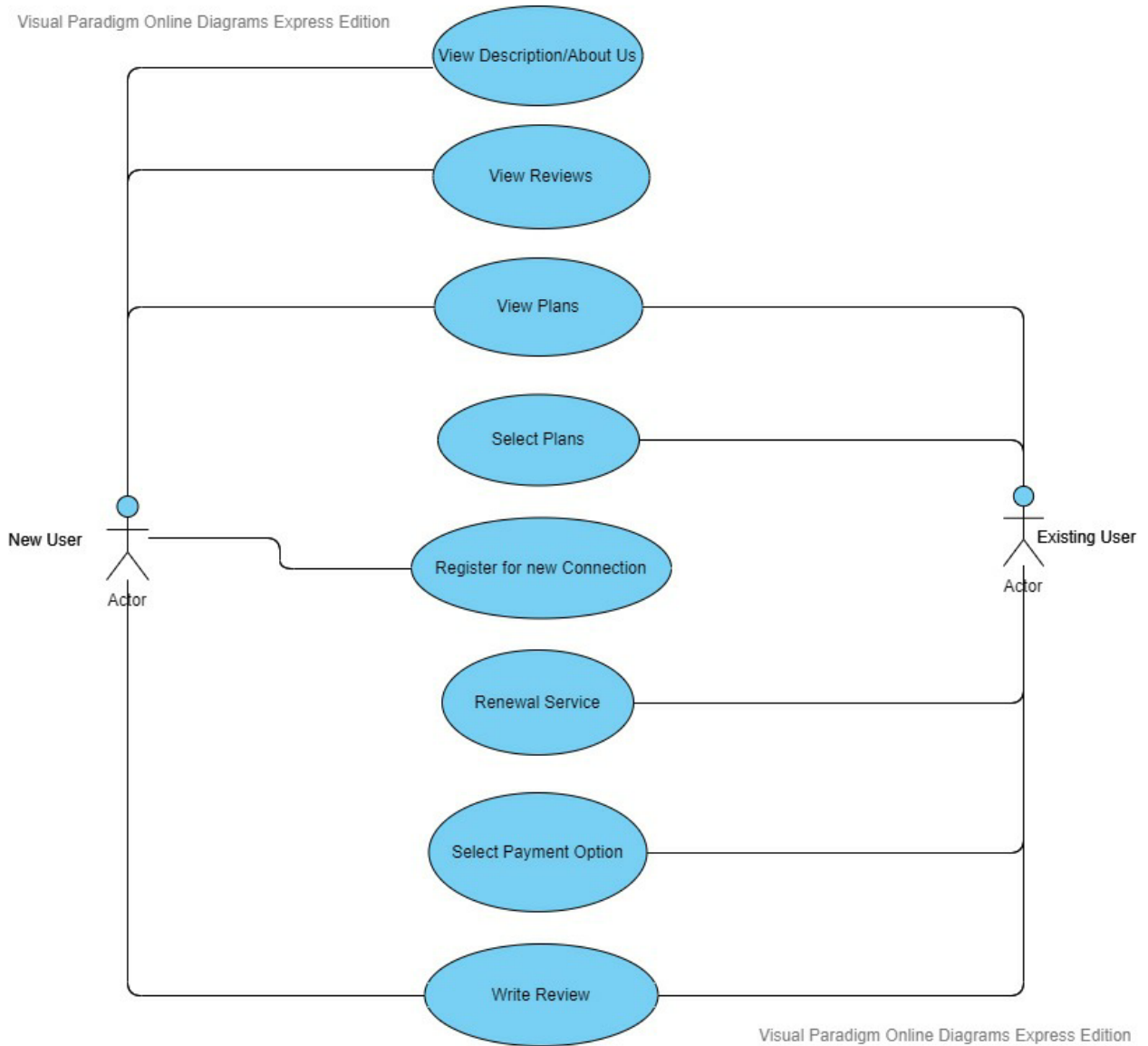


Fig 6.4

ENTITY-RELATIONSHIP Diagram:

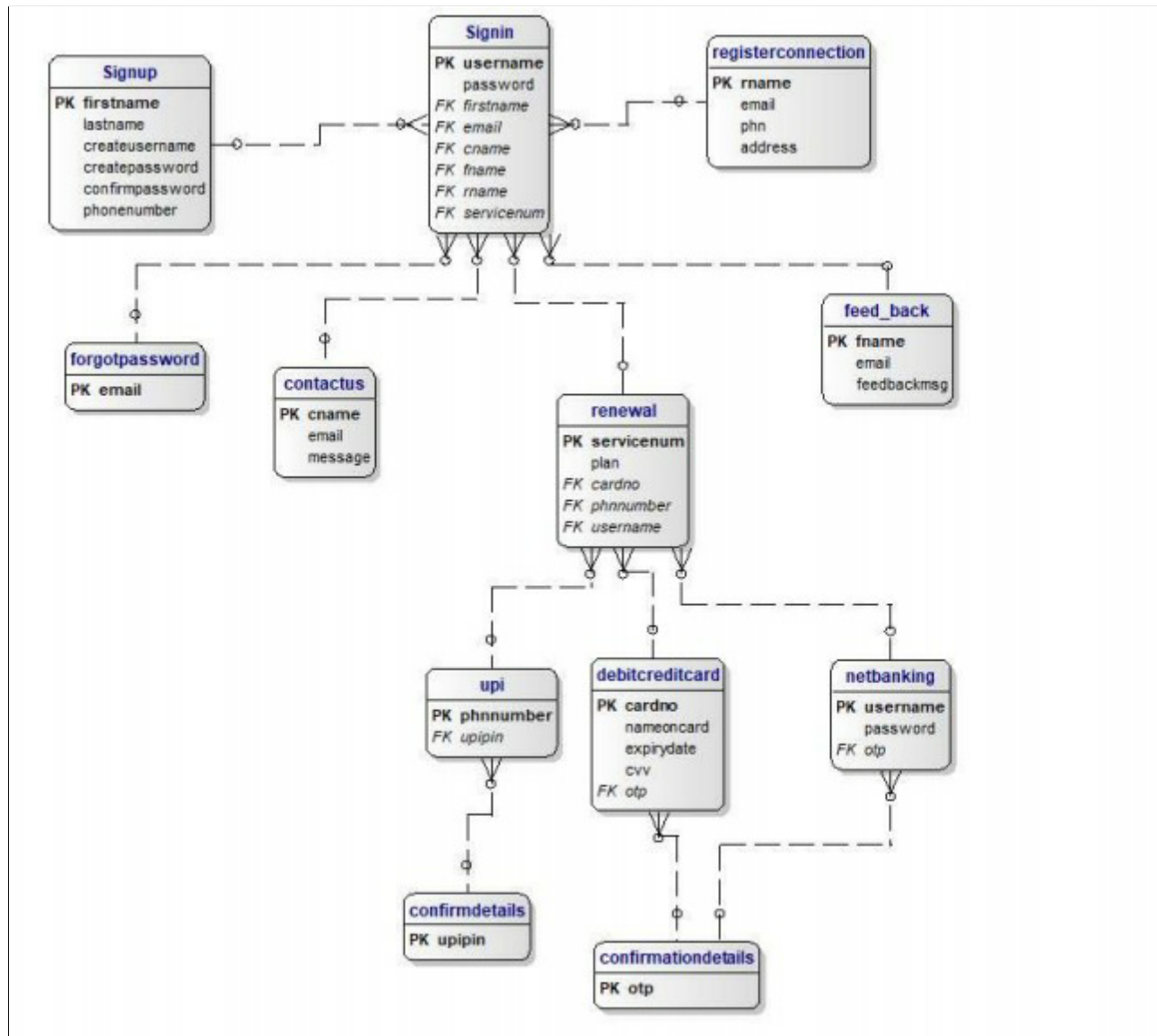


Fig 6.5

CHAPTER-7

PROJECT

Name: ONLINE BROADBAND CONNECTION SERVICES

7.1 Technologies Used:

HTML

CSS

Bootstrap

SQL

Java Script

JSON

jQuery

Node.JS

AJAX

Server: Local Host

Database: MySQL

Operating System: Windows7/8/8.1/10

Wireframing tool: Paint

Team Size: 4

7.2 TECHNICAL DETAILS

Front end is designed using HTML, CSS and Bootstrap. Ajax used to perform behind the screen requests and JavaScript used to perform client-side scripting

Backend is based on MySQL based RDB (Relational Database) model.

The SQL queries are run using the CI SQL library functions

Backend online host includes a centralized database resident on the server, the script which is built in PHP used to SQL query the database on user's request for transaction of data

The forms are made using the HTML, Bootstrap for designing and SQL for back-end JavaScript, AJAX and jQuery used for client-side scripting and PHP for the server-side development

CHAPTER-8

SCREENSHOTS

1. Configuration database(config.db.js)

```
app > config > JS db.config.js > [e] <unknown>
1  module.exports={
2  HOST : "localhost",
3  User : "root",
4  password : "koushik@123",
5  db : "project"
6  };
```

Connection of MY SQL DATABASE

```
app > models > JS db.js > ...
1  const mysql = require('mysql');
2  const dbconfig = require('../config/db.config.js');
3  const connection = mysql.createConnection({
4    host:dbconfig.HOST,
5    user:dbconfig.User,
6    password:dbconfig.password,
7    database:dbconfig.db
8  });
9
10 connection.connect(error =>{
11   if(error){
12     return console.error(error.message);
13   }
14   console.log('Successfully connected to MYSQL DATABASE');
15 });
16 module.exports = connection;
```

Creating Database in the MYSQL Workbench

```
1 • create database project;
2 • ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'koushik@123';
3
4 • flush privileges;
5 • use project;
```

API Code, Validations & Outputs of Data in the Database

8.1.Signin API Code-Model.js

```
app > models > JS signin.model.js > ...
1  const sql = require('./db.js');
2
3  const Signin = function(signin){
4    this.username = signin.txtUsername;
5    this.password = signin.txtPassword;
6
7  };
8
9
10 Signin.create = (newsigin,result)=>{
11   sql.query('Insert into signin set ?',newsigin,(err,res)=>{
12     if(err){
13       console.log(err);
14       result(err,null);
15       return;
16     }
17     console.log("Create Signin :",{id:res.insertedId,...newsigin});
18     return(null,{id:res.insertedId,...newsigin});
19   })
20 };
21
22 module.exports = Signin;
```

Controller.js

```
app > controllers > JS signin.controller.js > create > exports.create > [0] signin > txtPassword
1  const Signin = require("../models/signin.model.js");
2
3  exports.create = (req,res) =>{
4    if(!req.body){
5      res.status(400).send({
6        message : "Content can not be empty!"
7      });
8    }
9    const signin = new Signin({
10     txtUsername: req.body.username,
11     txtPassword: req.body.password
12   });
13   Signin.create(signin,(err,data)=>{
14     if(err)
15       res.status(500).send({
16         message:
17           err.message || "Some error occurred while creating the signin."
18       });
19     else res.send(data);
20   });
21 };
22
```

Routes.js


```

app > routes > JS signin.routes.js > ...
1 module.exports = app =>{
2   const Signin = require('../controllers/signin.controller.js');
3   app.post("/signin", Signin.create);
4 };

```

server.js

```

const express = require('express');
const bodyparser = require('body-parser');

const app = express();

app.use(bodyparser.json());
const bodyparser = typeof bodyParser
app.use(bodyparser.urlencoded({extended:true}));

app.get("/",(req,res)=>{
  res.json({message:'Welcome to World'});
});

require('./app/routes/signin.routes.js')(app);
require('./app/routes/registration.routes.js')(app);
require('./app/routes/signup.routes.js')(app);
require('./app/routes/servicenumber.routes.js')(app);
require('./app/routes/feedback.routes.js')(app);
require('./app/routes/debitcreditcard.routes.js')(app);
require('./app/routes/netbank.routes.js')(app);
require('./app/routes/fgpass.routes.js')(app);

app.listen(3000,()=>{
  console.log('Server is running on port 3000');
});

```

Validations and Ajax for signin

```

<script type="text/javascript">
    $(document).ready(function(){
        $("#btn").click(function(){
            if($('#txtUsername').val() == "")
            {
                swal({
                    text:"Enter Username",
                    icon:"error"
                });
                return false;
            }
            else if ($('#txtPassword').val() == "")
            {
                swal({
                    text:"Enter Password",
                    icon:"error"
                });
                return false;
            }
            else
            {
                window.location.href = "home.html";
            }
            var api_url = "http://localhost:3000/signin";
            var data = {
                username: $('#txtUsername').val(),
                password: $('#txtPassword').val(),
            }
            $.ajax({
                url:api_url,
                type:"POST",
                dataType:"Json",
                data:data,
            })
        })
    })

```

Creating the table of signin

```

10 • create table signin(
11     username varchar(20) not null,
12     password varchar(20) not null
13 )
14 );

```

Output of data in Database

```

14 • );
15 • select * from signin;
16

```

username	password
pratheeka	pratheeka@123
Anilkumar	Anil@123
sairamreddy	sairam@123
varshithreddy	varshith@123
koushikraj	koushik@1600

Fig 8.1

8.2.Signup API Code-model.js

```
app > models > JS signup.model.js > ...
1  const sql = require('./db.js');
2
3  const Signup = function(signup){
4    this.first_name= signup.fname;
5    this.last_name = signup.lname;
6    this.create_user = signup.username;
7    this.create_pass = signup.password;
8    this.confirm_pass = signup.retype_password;
9    this.phn_number = signup.number;
10 };
11
12 (method) Signup.create(newsignup: any, result: any): void
13 Signup.create = (newsignup,result)=>{
14   sql.query('Insert into signup set ?',newsignup,(err,res)=>{
15     if(err){
16       console.log(err);
17       result(err,null);
18       return;
19     }
20     console.log("Create Signup :",{id:res.insertedId,...newsignup});
21     return(null,{id:res.insertedId,...newsignup});
22   })
23 };
24
25 module.exports = Signup;
```

Controller.js

```
1  const Signup = require("../models/signup.model.js");
2
3  exports.create = (req,res) =>{
4    if(!req.body){
5      res.status(400).send({
6        message : "Content can not be empty!"
7      });
8    }
9    const signup = new Signup({
10     fname: req.body.first_name,
11     lname: req.body.last_name,
12     username: req.body.create_user,
13     password: req.body.create_pass,
14     retype_password: req.body.confirm_pass,
15     number: req.body.phn_number
16   });
17   Signup.create(signup,(err,data)=>{
18     if(err)
19       res.status(500).send({
20         message:
21           err.message || "Some error occurred while creating the signup."
22       });
23     else res.send(data);
24   });
25 };
```

routes.js

```
app > routes > JS signup.routes.js > ...
1  module.exports = app =>{
2    const Signup = require('../controllers/signup.controller.js');
3    app.post("/signup",Signup.create);
4  };
```

server.js

```
const express = require('express');
const bodyparser = require('body-parser');

const app = express();

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended:true}));

app.get("/",(req,res)=>{
  res.json({message:'Welcome to World'});
});

require('./app/routes/signin.routes.js')(app);
require('./app/routes/registration.routes.js')(app);
require('./app/routes/signup.routes.js')(app);
require('./app/routes/servicenumber.routes.js')(app);
require('./app/routes/feedback.routes.js')(app);
require('./app/routes/debitcreditcard.routes.js')(app);
require('./app/routes/netbank.routes.js')(app);
require('./app/routes/fgpass.routes.js')(app);

app.listen(3000,()=>{
  console.log('Server is running on port 3000');
});
```

Signup Validations

```
JS validations.js > valid > text
1  function valid(){
2      var fn=document.getElementById('fname').value;
3      var ln=document.getElementById('lname').value;
4      var use=document.getElementById('username').value;
5      var num=document.getElementById('number').value;
6      var pass=document.getElementById('password').value;
7      var repass=document.getElementById('retype_password').value;
8      var nu=/^[6-9][0-9]{9}$/;
9      var pw=/(?=[a-z])(?=[A-Z])(?=[0-9])(?=.*[!@#$]).{8,20}$/;
10     var rpw=/(?=[a-z])(?=[A-Z])(?=[0-9])(?=.*[!@#$]).{8,20}$/;
11     var nf=/^[a-zA-Z.]{6,40}$/;
12     var nl=/^[a-zA-Z.]{2,40}$/;
13     var us=/^[a-zA-Z-_.]{6,40}$/;
14
15
16     if(use=='' || num=='' || pass=='' || repass=='' || fn=='' || ln=='')
17     {
18         swal({
19             text:"Please enter all the fields",
20             icon:"error"
21         });
22         return false;
23     }
24     else{
25         if(nu.test(num) && rpw.test(repass) && pw.test(pass) && nf.test(fn) && nl.test(ln) && us.test(use)){
26             swal({
27                 text: "Your Account has been created successfully and please wait while we redirect you to Login",
28                 icon: "success",
29                 button: false
30             });
31             setTimeout(function(){
32                 window.location.href="internet.html";
33             },5000);
34         }
35     }
36 }
```

```

35     else{
36         if(!fn.test(fn)){
37             swal({
38                 text: "Invalid First Name",
39                 icon: "error",
40             });
41             return false;
42         }
43         else if(!ln.test(ln)){
44             swal({
45                 text: "Invalid last Name",
46                 icon: "error",
47             });
48             return false;
49         }
50         else if(!us.test(use)){
51             swal({
52                 text: "Invalid Username",
53                 icon: "error",
54             });
55             return false;
56         }
57
58         else if(!pw.test(pass)){
59             swal({
60                 text: "Invalid Password",
61                 icon: "error",
62             });
63             return false;
64         }
65         else if(!rpw.test(repass)){
66             swal({
67                 text: "Invalid Confirm Password",
68                 icon: "error",

```

```

67                 text: "Invalid Confirm Password",
68                 icon: "error",
69             });
70             return false;
71         }
72         else{
73             swal({
74                 text: "Invalid Phone Number",
75                 icon: "error",
76             });
77             return false;
78         }
79     }
80 }
81

```

Signup Ajax

```
12     <script type="text/javascript">
13         $(document).ready(function(){
14             $("#btn").click(function(){
15                 var api_url = "http://localhost:3000/signup";
16                 var data = {
17                     first_name: $('#fname').val(),
18                     last_name: $('#lname').val(),
19                     create_user: $('#username').val(),
20                     create_pass: $('#password').val(),
21                     confirm_pass: $('#retype_password').val(),
22                     phn_number: $('#number').val()
23                 }
24                 $.ajax({
25                     url:api_url,
26                     type:"POST",
27                     dataType:"json",
28                     data:data,
29                 })
30             });
31         });
32     </script>
33
```

Creating table of signup

```
create table signup(
    first_name varchar(20) not null,
    last_name varchar(20) not null,
    create_user varchar(20) not null,
    create_pass varchar(20) not null,
    confirm_pass varchar(20) not null,
    phn_number bigint not null
);
```

Output of data in Database

25 • `select * from signup;`

first_name	last_name	create_user	create_pass	confirm_pass	phn_number
Koushik	Raj	koushikraj	Koushik@123	Koushik@123	8978991600
koushik	raj	koushikraj	Koushik@123	Koushik@123	8978991600
ashwanth	reddy	ashwanthreddy	Ashwanth@123	Ashwanth@123	7487398987
koushik	raj	koushikraj	Koushik@123	Koushik@123	8978991600
koushik	raj	koushikraj	Koushik@123	Koushik@123	8978991600
Nagaraj	kumar	nagkumar	Nagkumar@123	Nagkumar@123	9298092017

Fig 8.2

8.3. Forgot password API-model.js

```
1 const sql = require("../db.js");
2
3 const Fgpass = function(fgpass){
4   this.email = fgpass.email;
5 }
6 };
7
8
9 Fgpass.create = (newfgpass,result)=>{
10   sql.query('Insert into fgpass set ?', newfgpass, (err,res)=>{
11     if(err){ (parameter) err: MysqlError
12       console.log(err);
13       result(err,null);
14       return;
15     }
16     console.log("Create Fgpass :",{id:res.insertedId,...newfgpass});
17     return(null,{id:res.insertedId,...newfgpass});
18   })
19 };
20
21 module.exports = Fgpass;
```

Controller.js

```
1 const Fgpass = require("../models/fgpass.model.js");
2
3 exports.create = (req,res) =>{
4   if(!req.body){
5     res.status(400).send({
6       message : "Content can not be empty!"
7     });
8   }
9   const fgpass = new Fgpass({
10     email: req.body.email
11   });
12 };
13 Fgpass.create(fgpass,(err,data)=>{
14   if(err)
15     res.status(500).send({
16       message:
17         err.message || "Some error occurred while creating the fgpass."
18     });
19   else res.send(data);
20 });
21
22
```

routes.js

```
1 module.exports = app =>{
2   const Fgpass = require("../controllers/fgpass.controller.js");
3   app.post("/fgpass",Fgpass.create);
4 };
```

server.js

```
const express = require('express');
const bodyparser = require('body-parser');

const app = express();

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended:true}));

app.get("/",(req,res)=>{
  res.json({message:'Welcome to World'});
});

require('./app/routes/signin.routes.js')(app);
require('./app/routes/registration.routes.js')(app);
require('./app/routes/signup.routes.js')(app);
require('./app/routes/servicenumber.routes.js')(app);
require('./app/routes/feedback.routes.js')(app);
require('./app/routes/debitcreditcard.routes.js')(app);
require('./app/routes/netbank.routes.js')(app);
require('./app/routes/fgpass.routes.js')(app);

app.listen(3000,()=>{
  console.log('Server is running on port 3000');
});
```

Validations and Ajax

```
1 function validate(){
2     var em=document.getElementById('email').value;
3     var e=/^[a-zA-Z0-9.-_]{6,20}[@][a-z]{3,15}[.][a-z]{2,6}$/;
4     if(em=='')
5     {
6         swal({
7             text:"Please enter the Email",
8             icon:"error"
9         });
10        return false;
11    }
12    else{
13        if(e.test(em)){
14            swal({
15                text:"Reset Password Link has been sent to Your Email!",
16                icon:"success"
17            });
18            var api_url = "http://localhost:3000/fgpass";
19            var data = {
20                email: $('#email').val()
21            }
22            $.ajax({
23                url:api_url,
24                type:"POST",
25                dataType:"json",
26                data:data,
27            })
28        }
29    }
30    else{
31        swal({
32            text:"Please enter the valid Email....",
33            icon:"error"
34        });
35    }
```

Creating table of forgot password

```
create table fgpass(
    email varchar(40) not null
);
```

Output of data in Database

```
9 • select * from fgpass;
```

email
koushikrajtodupunuri@gmail.com
koushikrajtodupunuri@gmail.com

Fig 8.3

8.4.Renewal API Code-model.js

```
1  const sql = require('./db.js');
2
3  const Servicenumber = function(servicenumber){
4      this.service_number = servicenumber.number;
5
6
7  };
8
9
10 Servicenumber.create = (newservicenumber,result)=>{
11     sql.query('Insert into servicenumber set ?',newservicenumber,(err,res)=>{
12         if(err){
13             console.log(err);
14             result(err,null);
15             return;
16         }
17         console.log("Create Servicenumber :",{id:res.insertedId,...newservicenumber});
18         return(null,{id:res.insertedId,...newservicenumber});
19     })
20 };
21
22 module.exports = Servicenumber;
```

Controller.js

```
1  const Servicenumber = require("../models/servicenumber.model.js");
2
3  exports.create = (req,res) =>{
4      if(!req.body){
5          res.status(400).send({
6              message : "Content can not be empty!"
7          });
8      }
9      const servicenumber = new Servicenumber({
10
11          number: req.body.service_number,
12
13      });
14
15      Servicenumber.create(servicenumber,(err,data)=>{
16          if(err)
17              res.status(500).send({
18                  message:
19                      err.message || "Some error occurred while creating the Servicenumber."
20              });
21          else res.send(data);
22      });
23  };
24
```

routes.js

```

1 module.exports = app =>{
2   const Servicenumber = require('../controllers/servicenumber.controller.js');
3   app.post("/servicenumber",Servicenumber.create);
4 };

```

server.js

```

1  const express = require('express');
2  const bodyparser = require('body-parser');
3
4  const app = express();
5
6  app.use(bodyparser.json());
7
8  app.use(bodyparser.urlencoded({extended:true}));
9
10 app.get("/",(req,res)=>{
11   res.json({message:'Welcome to World'});
12 });
13
14 require('./app/routes/signin.routes.js')(app);
15 require('./app/routes/registration.routes.js')(app);
16 require('./app/routes/signup.routes.js')(app);
17 require('./app/routes/servicenumber.routes.js')(app);
18 require('./app/routes/feedback.routes.js')(app);
19 require('./app/routes/debitcreditcard.routes.js')(app);
20 require('./app/routes/netbank.routes.js')(app);
21 require('./app/routes/fgpass.routes.js')(app);
22
23
24 app.listen(3000,()=>{
25   console.log('Server is running on port 3000');
26 });

```

Validations and Ajax

```

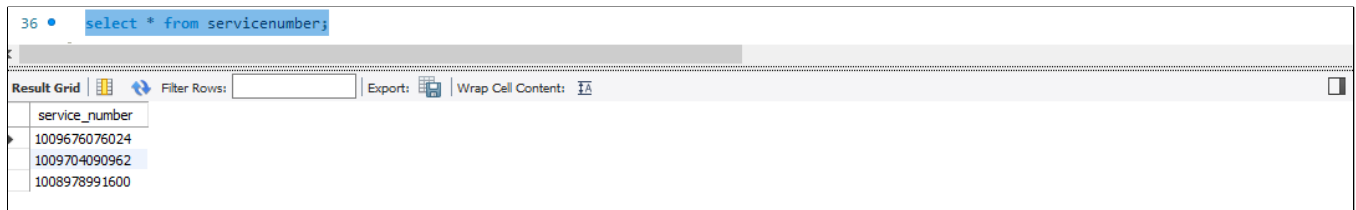
<script type="text/javascript">
  $(document).ready(function(){
    $("#btn").click(function(){
      if($('#number').val() == ""){
        swal({
          text:"Enter Service Number",
          icon:"error"
        });
        return false;
      }
      else{
        var api_url = "http://localhost:3000/servicenumber";
        var data = {
          service_number: $('#number').val(),
        }
        $.ajax({
          url:api_url,
          type:"POST",
          dataType:"Json",
          data:data,
        })
        window.location.href = "payment.html";
      }
    });
  });
</script>

```

Creating table for renewal

```
create table servicenumber(  
  service_number bigint not null  
);
```

Output of data in Database



The screenshot shows a database query result for the query `select * from servicenumber;`. The result is displayed in a table with one column, `service_number`, and three rows of data.

service_number
1009676076024
1009704090962
1008978991600

Fig 8.4

8.5.API for Registration(model.js)

```
1  const sql = require('./db.js');  
2  
3  const Registration = function(registration){  
4    this.name= registration.name;  
5    this.email = registration.email;  
6    this.phn_number = registration.number;  
7    this.address = registration.address;  
8  };  
9  
10  
11  
12  Registration.create = (newregistration,result)=>{  
13    sql.query('Insert into registration set ?',newregistration,(err,res)=>{  
14      if(err){  
15        console.log(err);  
16        result(err,null);  
17        return;  
18      }  
19      console.log("Create registration :",{id:res.insertedId,...newregistration});  
20      return(null,{id:res.insertedId,...newregistration});  
21    })  
22  };  
23  
24  module.exports = Registration;
```

controller.js

```
1  const Registration = require("../models/registration.model.js");
2
3  exports.create = (req,res) =>{
4      if(!req.body){
5          res.status(400).send({
6              message : "Content can not be empty!"
7          });
8      }
9      const registration = new Registration({
10         name: req.body.name,
11         email: req.body.email,
12         number: req.body.phn_number,
13         address: req.body.address
14     });
15     Registration.create(registration,(err,data)=>{
16         if(err)
17             res.status(500).send({
18                 message:
19                     err.message || "Some error occurred while creating the registration."
20             });
21         else res.send(data);
22     });
23 };
24
25
```

routes.js

```
1  module.exports = app =>{
2      const Registration = require('../controllers/registration.controller.js');
3      app.post("/registration",Registration.create);
4  };

```

server.js

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const app = express();
5
6  app.use(bodyParser.json());
7
8  app.use(bodyParser.urlencoded({extended:true}));
9
10 app.get("/",(req,res)=>{
11   res.json({message:'Welcome to World'}});
12 });
13
14 require('./app/routes/signin.routes.js')(app);
15 require('./app/routes/registration.routes.js')(app);
16 require('./app/routes/signup.routes.js')(app);
17 require('./app/routes/servicenumber.routes.js')(app);
18 require('./app/routes/feedback.routes.js')(app);
19 require('./app/routes/debitcreditcard.routes.js')(app);
20 require('./app/routes/netbank.routes.js')(app);
21 require('./app/routes/fgpass.routes.js')(app);
22
23
24 app.listen(3000,()=>{
25   console.log('Server is running on port 3000');
26 });
```

Validations and Ajax


```

1  function valid(){
2      var name=document.getElementById('name').value;
3      var email=document.getElementById('email').value;
4      var num=document.getElementById('number').value;
5      var add=document.getElementById('address').value;
6
7      var e=/^[a-zA-Z0-9.-_]{6,20}@[a-z]{3,15}[.][a-z]{2,6}$/;
8      var nu=/^[6-9][0-9]{9}$/;
9      var n=/^[a-zA-Z\s]{6,40}$/;
10     var a=/^[#.0-9a-zA-Z\s,-]+$/;
11
12
13     if(email==' ' || name==' ' || num==' ' || add==' ')
14     {
15         swal('Please enter all the Fields');
16         return false;
17     }
18     else{
19         if(e.test(email) && nu.test(num) && n.test(name) && a.test(add)){
20             swal('Thanks for Registration...!Our Agent will approach to your doorstep.');

```

Creating table for registration

```

• create table registration(
    name varchar(20) not null,
    email varchar(30) not null,
    phn_number bigint not null,
    address varchar(100) not null

```

Output of data in Database

```
32 • select * from registration;
```

name	email	phn_number	address
koushik raj	koushikrajtodupunuri@gmail.com	8978991600	qwertyuiop asdfghjkl
koushik	koushikrajtodupunuri@gmail.com	8978991600	qwertyui

Fig 8.5

8.6.API for Feedback(model.js)

```

1  const sql = require('./db.js');
2
3  const Feedback = function(feedback){
4      this.name= feedback.name;
5      this.email = feedback.email;
6      this.feedback = feedback.feed_back;
7
8
9
10
11  module Feedback
12  const Feedback: typeof Feedback
13  Feedback.create = (newfeedback,result)=>{
14      sql.query('Insert into feedback set ?',newfeedback,(err,res)=>{
15          if(err){
16              console.log(err);
17              result(err,null);
18              return;
19          }
20          console.log("Create feedback :",{id:res.insertedId,...newfeedback});
21          return(null,{id:res.insertedId,...newfeedback});
22      })
23  };
24  module.exports = Feedback;

```

controller.js

```

const Feedback = require("../models/feedback.model.js");

exports.create = (req,res) =>{
    if(!req.body){
        res.status(400).send({
            message : "Content can not be empty!"
        });
    }
    const feedback = new Feedback({
        name: req.body.name,
        email: req.body.email,
        feed_back: req.body.feedback
    });
    Feedback.create(feedback,(err,data)=>{
        if(err)
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the feedback."
            });
        else res.send(data);
    });
};

```

routes.js

```

1 module.exports = app =>{
2   const Feedback = require('../controllers/feedback.controller.js');
3   app.post("/feedback",Feedback.create);
4 };

```

server.js

```

const express = require('express');
const bodyParser = require('body-parser');

const app = express

app.use(bodyParser.urlencoded({extended:true}));

app.get("/",(req,res)=>{
  res.json({message:'Welcome to World'});
});

require('./app/routes/signin.routes.js')(app);
require('./app/routes/registration.routes.js')(app);
require('./app/routes/signup.routes.js')(app);
require('./app/routes/servicenumber.routes.js')(app);
require('./app/routes/feedback.routes.js')(app);
require('./app/routes/debitcreditcard.routes.js')(app);
require('./app/routes/netbank.routes.js')(app);
require('./app/routes/fgpass.routes.js')(app);

app.listen(3000,()=>{
  console.log('Server is running on port 3000');
});

```

Validation and Ajax

```

1 function valid(){
2     var name = document.getElementById('name').value;
3     var email = document.getElementById('email').value;
4     var feed = document.getElementById('feed_back').value;
5
6     var e=/^[a-zA-Z0-9._-]{6,20}[@][a-z]{3,15}[.][a-z]{2,6}$/;
7     var n=/^[a-zA-Z.\s]{6,40}$/;
8     var f=/^[a-zA-Z0-9.,\s]{10,100}$/;
9
10    if(email==' ' || name==' ' || feed==' ')
11    {
12        swal({
13            text:"Please enter all the Fields",
14            icon:"error"
15        });
16        return false;
17    }
18    else{
19        if(e.test(email) && n.test(name) && f.test(feed)){
20            swal({
21                text: 'Thanks for your Feedback....!',
22                icon: "success"
23            });
24            var api_url = "http://localhost:3000/feedback";
25            var data = {
26                name: $('#name').val(),
27                email: $('#email').val(),
28                feedback: $('#feed_back').val()
29            }
30
31            $.ajax({
32                url:api_url,
33                type:"POST",

```

```

34            $.ajax({
35                url:api_url,
36                type:"POST",
37                dataType:"Json",
38                data:data,
39            })
40        }
41        else{
42            swal({
43                text:"Please enter the valid details....",
44                icon:"error"
45            });
46            return false;
47        }
48    }

```

Creating table for feedback

```

create table feedback(
    name varchar(20) not null,
    email varchar(30) not null,
    feedback varchar(200) not null
);

```

Output of data Database

48 • `select * from feedback;`

name	email	feedback
koushik raj	koushikrajtodunuri@gmail.com	qwertyuiop asdfghjk

Fig 8.6

8.7.API for Debit/credit card(model.js)

```
1 const sql = require('./db.js');
2
3 const Debitcreditcard = function(debitcreditcard){
4   this.card_no= debitcreditcard.number;
5   this.name_on_card = debitcreditcard.name;
6   this.expiry_date = debitcreditcard.date;
7   this.cvv = debitcreditcard.cvv;
8
9 };
10
11 (parameter) result: any
12 Debitcreditcard.create = (newdebitcreditcard,result)=>{
13   sql.query('Insert into debitcreditcard set ?',newdebitcreditcard,(err,res)=>{
14     if(err){
15       console.log(err);
16       result(err,null);
17       return;
18     }
19     console.log("Create Debitcreditcard :",{id:res.insertedId,...newdebitcreditcard});
20     return(null,{id:res.insertedId,...newdebitcreditcard});
21   })
22 };
23
24 module.exports = Debitcreditcard;
```

controller.js

```
1 const Debitcreditcard = require("../models/debitcreditcard.model.js");
2
3 exports.create = (req,res) =>{
4   if(!req.body){
5     res.status(400).send({
6       message : "Content can not be empty!"
7     });
8   }
9   const debitcreditcard = new Debitcreditcard({
10     number: req.body.card_no,
11     name: req.body.name_on_card,
12     date: req.body.expiry_date,
13     cvv: req.body.cvv
14   });
15
16   Debitcreditcard.create(debitcreditcard,(err,data)=>{
17     if(err)
18       res.status(500).send({
19         message:
20           err.message || "Some error occurred while creating the debitcreditcard."
21       });
22     else res.send(data);
23   });
24 };
25
```

routes.js

```
1 module.exports = app =>{
2   const Debitcreditcard = require('../controllers/debitcreditcard.controller.js');
3   app.post("/debitcreditcard",Debitcreditcard.create);
4 };
```

server.js

```
1 const express = require('express');
2 const bodyparser = require('body-parser');
3
4 const app = express();
5
6 app.use(bodyparser.json());
7
8 app.use(bodyparser.urlencoded({extended:true}));
9
10 app.get("/",(req,res)=>{
11   res.json({message:'Welcome to World'});
12 });
13
14 require('./app/routes/signin.routes.js')(app);
15 require('./app/routes/registration.routes.js')(app);
16 require('./app/routes/signup.routes.js')(app);
17 require('./app/routes/servicenumber.routes.js')(app);
18 require('./app/routes/feedback.routes.js')(app);
19 require('./app/routes/debitcreditcard.routes.js')(app);
20 require('./app/routes/netbank.routes.js')(app);
21 require('./app/routes/fgpass.routes.js')(app);
22
23
24 app.listen(3000,()=>{
25   console.log('Server is running on port 3000');
26 });
```

Validations and Ajax

```
function valid(){
    var num=document.getElementById('number').value;
    var nam=document.getElementById('name').value;
    var dat=document.getElementById('date').value;
    var cv=document.getElementById('cvv').value;

    var nu= /^[0-9\s]{16}$/;
    var na= /^[A-Z\s]{5,30}$/;
    var da= /^(0?[1-9]|1[012])[\/\-\]\d{2}$/;
    var c= /^[0-9]{3}$/;

    if(num==' ' || nam==' ' || dat==' ' || cv==' '){
        {
            swal({
                text:"Please enter all the Fields",
                icon:"error"
            });
            return false;
        }
    }
    else{
        if(nu.test(num) && na.test(nam) && da.test(dat) && c.test(cv)){
            swal({
                text: "Enter OTP",
                content: "input",
                button: {
                    text: "Submit",
                }
            }).then((value)=>{
                swal({
                    title:'Please Wait',
                    text:'Your transaction is in progress',
                    icon:"info",

```

```
35         button:false
36     });
37     setTimeout(function(){
38         swal({
39             text:"Your transaction has completed successfully",
40             icon:"success"
41         });
42     },2000);
43     });
44     var api_url = "http://localhost:3000/debitcreditcard";
45     var data = {
46         card_no: $('#number').val(),
47         name_on_card: $('#name').val(),
48         expiry_date: $('#date').val(),
49         cvv: $('#cvv').val()
50     }
51     $.ajax({
52         url:api_url,
53         type:"POST",
54         dataType:"Json",
55         data:data,
56     })
57     }
58     }
59     }
60     else{
61         swal({
62             text:"Please enter the valid details....",
63             icon:"error"
64         });
65         return false;
66     }
67     }
68 }
```

Creating table for debit/credit card

```
create table debitcreditcard(
    card_no bigint not null,
    name_on_card varchar(40) not null,
    expiry_date varchar(20) not null,
    cvv int not null
);
```

Output of data Database

55 • `select * from debitcreditcard;`

card_no	name_on_card	expiry_date	cvv
1234567890123456	MARAM ASHWANTH REDDY	12/24	123
1234567890112345	KOUSHIK RAJ	12/20	123
1123456789012345	KOUSHIK	12/20	123

Fig 8.

8.8.API for Net banking (model.js)

```
const sql = require('./db.js');

const Netbank = function(netbank){
  this.username= netbank.txtUsername;
  this.password = netbank.txtPassword;
};

Netbank.create = (newnetbank,result)=>{
  sql.query('Insert into netbank set ?',newnetbank,(err,res)=>{
    if(err){
      console.log(err);
      result(err,null);
      return;
    }
    console.log("Create Netbank :",{id:res.insertedId,...newnetbank});
    return(null,{id:res.insertedId,...newnetbank});
  })
};

module.exports = Netbank;
```

controller.js

```
const Netbank = require("../models/netbank.model.js");

exports.create = (req,res) =>{
  if(!req.body){
    res.status(400).send({
      message : "Content can not be empty!"
    });
  }
  const netbank = new Netbank({
    txtUsername: req.body.username,
    txtPassword: req.body.password
  });

  Netbank.create(netbank,(err,data)=>{
    if(err)
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the netbank."
      });
    else res.send(data);
  });
};
```


routes.js

```
module.exports = app =>{
  const Netbank = require('../controllers/netbank.controller.js');
  app.post("/netbank",Netbank.create);
};
```

server.js

```
const express = require('express');
const bodyparser = require('body-parser');

const app = express();

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended:true}));

app.get("/",(req,res)=>{
  res.json({message:'Welcome to World'});
});

require('./app/routes/signin.routes.js')(app);
require('./app/routes/registration.routes.js')(app);
require('./app/routes/signup.routes.js')(app);
require('./app/routes/servicenumber.routes.js')(app);
require('./app/routes/feedback.routes.js')(app);
require('./app/routes/debitcreditcard.routes.js')(app);
require('./app/routes/netbank.routes.js')(app);
require('./app/routes/fgpass.routes.js')(app);

app.listen(3000,()=>{
  console.log('Server is running on port 3000');
});
```

Validations and Ajax

```
1 function valid(){
2     var use = document.getElementById('txtUsername').value;
3     var pas = document.getElementById('txtPassword').value;
4
5     var u=/^[a-zA-Z0-9.-_]{3,20}$/;
6     var p=/^[a-zA-Z0-9!@#*]{3,20}$/;
7
8     if(use==' ' || pas==' '){
9         swal({
10             text:"Please enter all the fields",
11             icon:"error"
12         });
13     }
14     else {
15         if(u.test(use) && p.test(pas)){
16             swal({
17                 text: "Enter OTP",
18                 content: "input",
19                 button: {
20                     text: "Submit",
21                 }
22             }).then((value)=>{
23
24                 swal({
25                     title:'Please Wait',
26                     text:'Your transaction is in progress',
27                     icon:"info",
28                     button:false
29                 });
30                 setTimeout(function(){
31                     swal({
32                         text:"Your transaction has completed successfully",
33                         icon:"success"
34                     });
```

```

        icon:"success"
    });
    },2000);
});
var api_url = "http://localhost:3000/netbank";
var data = {
    username: $('#txtUsername').val(),
    password: $('#txtPassword').val(),
}
$.ajax({
    url:api_url,
    type:"POST",
    dataType:"Json",
    data:data,

})
}
else{
    swal({
        text:"Please enter the valid details....",
        icon:"error"
    });
    return false;
}
}
}

```

Creating table for Netbanking

```

create table netbank(
    username varchar(20) not null,
    password varchar(20) not null
);

```

Output of data in Database

select * from netbank;

username	password
ashwanthreddy	ashwanth@2315
varshithreddy	varshith@123

Fig 8.8

CHAPTER-9

MAINTENANCE

The maintenance phase involves making changes to hardware, software, and documentation to support its operational effectiveness. It includes making changes to improve a system's performance, correct problems, enhance security, or address user requirements. To ensure modifications do not disrupt operations or degrade a system's performance or security, organizations should establish appropriate change management standards and procedures.

Routine changes are not as complex as major modifications and can usually be implemented in the normal course of business. Routine change controls should include procedures for requesting, evaluating, approving, testing, installing, and documenting website modifications. Maintaining accurate, up-to-date hardware and software inventories is a critical part of all change management processes. Management should carefully document all modifications to ensure accurate system inventories. Management should coordinate all technology related changes through an oversight committee and assign an appropriate party responsibility for administering software patch management programs. Quality assurance, security, audit, regulatory compliance, network, and end-user personnel should be appropriately included in change management processes. Risk and security review should be done whenever a system modification is implemented to ensure controls remain in place.

For maintenance of the website:

1. The database has to be updated regularly according to new available information.
2. Redundant and false information must be removed from the database.
3. Newer versions of PHP and MYSQL can be used for up gradation of website and to improve the overall performance of the system.

CHAPTER-10

FUTURE SCOPE AND FUTURE ENHANCEMENT

PROJECT NAME: ONLINE BROADBAND INTERNET SERVICES

In India, freedom to choose is hindered by regulatory and tax complexity. The only way forward for India in this digital world is facilitating broadband over the already ubiquitous cable television wires. According to TRAI, there is a little over one broadband line for every 100 people in India. Cable broadband delivers 1.5Gbps speed to each home in many countries along with HDTV channels. The standard can deliver 10.0Gbps downlink and uplink speeds.

CONCLUSION

CHAPTER-11

We have successfully implemented the site 'KAPS BROADBAND SERVICES'. With the help of various links and tools, we have been able to provide a site which will be live soon and running on the web. We have been successful in our attempt to take care of the needs of both the user as well as the administrator. Finally, we hope that this will go a long way in popularizing.

BIBLIOGRAPHY

1. www.javatutpoint.com

2. www.w3schools.com
3. www.getbootstrap.com
4. www.codeigniter.com
5. www.stackoverflow.com
6. www.fontawesome.io
7. www.sweetalert.js.org
8. www.codepen.io
9. Learn HTML and CSS faster (Mark Myers)
10. Wikipedia

