

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI, 590-018



A MINI PROJECT REPORT ON

“STACK OVERFLOW”

Submitted on partial fulfillment of the requirements for the fifth semester

DATABASE MANAGEMENT SYSTEM LABORATORY

for the course of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING

Submitted by

SHREEVATSA G HEGDE

(1ST17CS151)

RAKSHTIH GOWDA G

(1ST17CS126)

Under the Guidance of

Mrs. BILWASHREE H

Asst. Prof, Dept. of CSE

Mrs. DIVYASHREE N

Asst. Prof, Dept. of CSE



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAMBHRAM INSTITUTE OF TECHNOLOGY, BENGALURU 560-097

SAMBRAM INSTITUTE OF TECHNOLOGY

BENGALURU - 560097

(Affiliated to Visvesvaraya Technological University, Belgaum)



DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **“Stack Overflow”** has been carried out by **RAKSHITH GOWDA G (USN 1ST17CS126)** and **SHREEVATSA G HEGDE (USN 1ST17CS151)** in the partial fulfilment for the requirement of DBMS lab of Bachelor of Engineering in Computer Science & Engineering of **Visvesvaraya Technological University, Belgaum** during the year 2019-20. It is certified that all correction/suggestion indicated for Internal Assessment have been incorporated in this report. The project report has been approved as it satisfies the academic requirements with respect of project work prescribed for the said Degree.

Mrs. Bilwashree H

Asst.Prof, Dept. of CSE

SaIT, Bengaluru

Dr. T John Peter

HOD, Dept. of CSE

SaIT, Bengaluru

Examiner

Signature with Date

1. _____

2. _____

ABSTRACT

Stack Overflow is a popular on-line programming question and answer community providing its participants with rapid access to knowledge and expertise of their peers, especially benefitting coders. Despite the popularity of Stack Overflow, its role in the work cycle of open-source developers is yet to be understood: on the one hand, participation in it has the potential to increase the knowledge of individual developers thus improving and speeding up the development process. On the other hand, participation in Stack Overflow may interrupt the regular working rhythm of the developer, hence also possibly slow down the development process. In this paper we investigate the interplay between Stack Overflow activities and the development process, reflected by code changes committed to the largest social coding repository, GitHub. Our study shows that active GitHub committers ask fewer questions and provide more answers than others. Moreover, we observe that active Stack Overflow askers distribute their work in a less uniform way than developers that do not ask questions. Finally, we show that despite the interruptions incurred, the Stack Overflow activity rate correlates with the code changing activity in GitHub.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any project would be incomplete without mentioning the people who made it possible, whose constant guidance and encouragements crowned my efforts with success. We take this opportunity to express the deepest gratitude and appreciation to all those who held me directly or indirectly towards the successful completion of the project.

We would like to thank **Dr. H.G. Chandrakanth**, Principal, Sambhram Institute of Technology, Bengaluru-97, for all facilities provided.

We would like to thank **Dr. T John Peter, HOD**, Dept. of CSE, for his support and encouragement that went a long way in successful completion of this project work.

This would not have been completed without the guidance of our lecturer **Mrs. Bilwashree H & Ms. Divyashree**, Assistant Professors, Dept. of CSE who thought the concept underlying the project and helped us rectify errors at every stage. And also, being our internal guide, for his integral and incessant support offered to us throughout the course of this project and for the constant source of inspiration throughout the project.

Finally, we are thankful to our family and friends who have constantly motivated and supported us and provided us with the best of opportunities in life to achieve our dreams.

RAKSHITH GOWDA G
SHREEVATSA G HEGDE

TABLE OF CONTENTS

CHAPTER NAME	DESCRIPTION	PAGE NO
Chapter 1	Introduction to databases	1-5
	1.1 Database System environment	2
	1.2 Advantages of using dbms approach	4
	1.3 Architecture of database	5
Chapter 2	System requirements and Specification	7-11
	2.1 Functional requirements	7
	2.1.1 Valid assumptions and dependencies	7
	2.1.2 Data requirement	8
	2.1.3 External interface requirement	8
	2.1.4 Operational requirements	9
	2.2 Non-Functional requirements	10
	2.2.1 Performance requirement	10
	2.2.2 Product usability requirements	10
	2.2.3 Requirements attributes	10
	2.3 Software requirements specification	11
	2.4 Hardware requirements specification	12
Chapter 3	Design and implementation	13-16
	3.1 ER Model	13
	3.2 Schema diagram	14
	3.3 Description of relations(Tables)	15
	3.3.1 User Schema	15
	3.3.2 Answer Schema	15
	3.3.3 Tag Schema	15
	3.3.4 Blog Schema	15
	3.3.5 Question Schema	16
Chapter 4	Implementation	17-22
	4.1 Code Snippets for front end	17
	4.1.1 Stack overflow main page HTML	17
	4.1.2 User registration HTML	18
	4.1.3 Ask question HTML	19

	4.1.4 Question and Answer HTML	19
	4.2 Code Snippets for back end	21
	4.2.1 Database Connection	21
	4.2.2 User Registration	21
	4.2.3 Inserting a Question	22
Chapter 5	Discussions and Screenshots	24-29
	5.1 Home page	24
	5.2 Register page	25
	5.3 Login page	25
	5.4 Main page	26
	5.5 Ask Question	26
	5.6 All Users	27
	5.7 New Blog	28
	5.8 All Blogs	28
	5.9 Question Answer	29
	5.10 Logout	29
	5.11 Result	29
	Conclusion	30
	Bibliography	31

LIST OF FIGURES

Fig No.	Description	Page No.
1.1	The database system environment	2
1.2	A physical, centralized and basic client server architecture	5
3.1	ER diagram of Stack Overflow website	13
3.2	Schema diagram of Stack Overflow website	14
5.1	Stack Overflow Home Page	24
5.2	SignUp Page	25
5.3	Login Page	25
5.4	Main Page	26
5.5	Question Composition	26
5.6	All Users	27
5.7	Compose New Blog	28
5.8	All Blogs	28
5.9	Question Answer	29

LIST OF TABLES

Table No.	Description	Page No.
3.1	User Schema	15
3.2	Answer Schema	15
3.3	Tags Schema	15
3.4	Blogs Schema	15
3.5	Question Schema	16

STACK OVERFLOW



CHAPTER 1

INTRODUCTION TO DATABASES

A **database** is a collection of related data.¹ By **data**, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. Nowadays, this data is typically stored in mobile phones, which have their own simple database software. This data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term *database* is usually more restricted. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested

A database may be generated and maintained manually or it may be computerized. For example, a library card catalog is a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system. Of course, we are only concerned with computerized databases in this text.

A **database management system (DBMS)** is a computerized system that enables users to create and maintain a database. The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating, and sharing* databases among various

users and applications. **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**. **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS. **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the data- base to reflect changes in the miniworld , and generating reports from the data. **Sharing** a database allows multiple users and programs to access the database simultaneously.

1.1 Database system environment

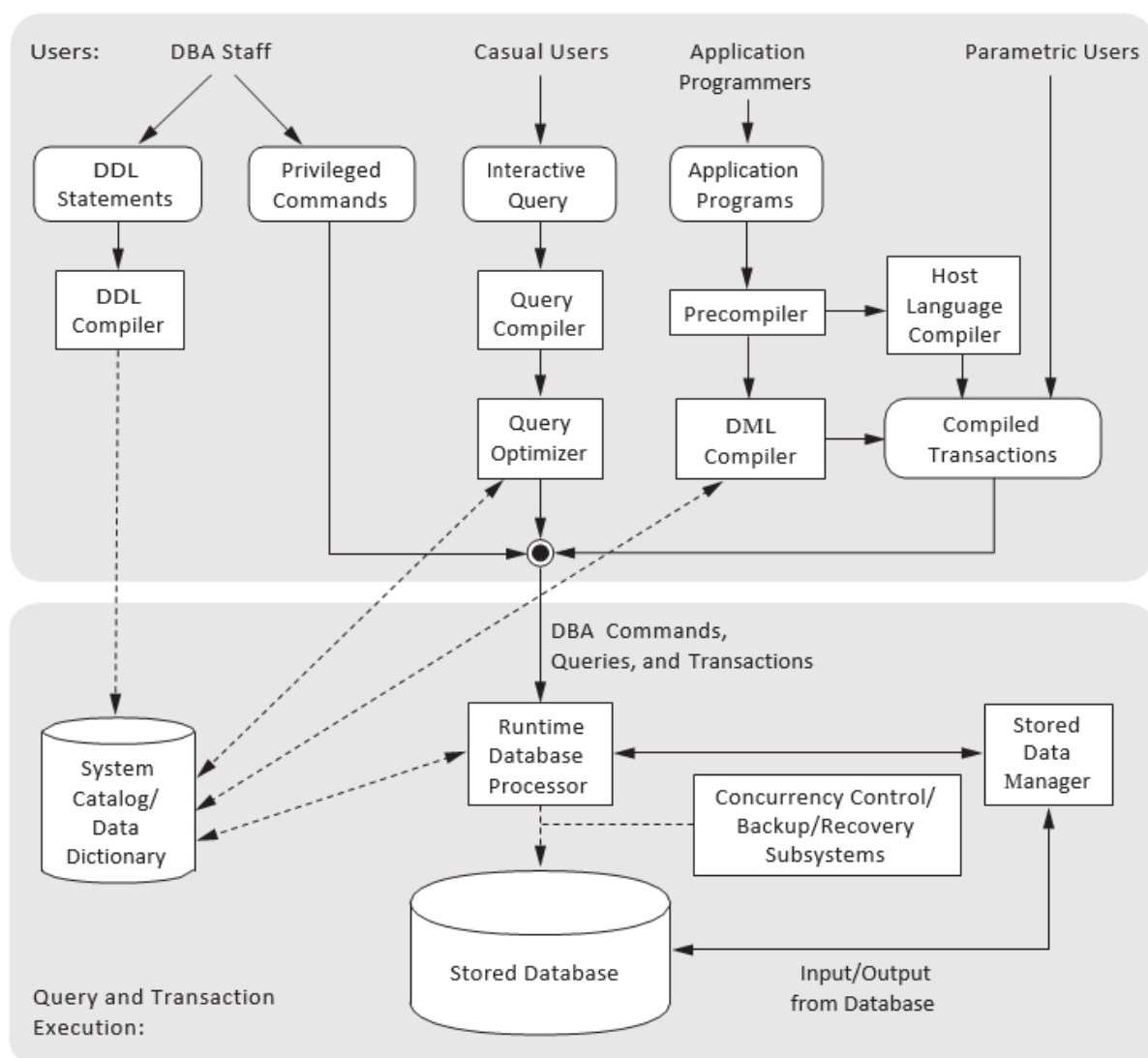


Fig 1.1: The database system environment

The figure is divided into two parts. The top part of the figure refers to the various users of the



database environment and their interfaces. The lower part shows the internal modules of the DBMS responsible for storage of data and processing of transactions.

The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the **operating system (OS)**, which schedules disk read/write. Many DBMSs have their own **buffer management** module to schedule disk read/write, because management of buffer storage has a considerable effect on performance. Reducing disk read/write improves performance considerably. A higher-level **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog.

Let us consider the top part of Figure 1 first. It shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries, application programmers who create programs using some host programming languages, and parametric users who do data entry work by supplying parameters to predefined transactions. The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.

In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Casual users and persons with occasional need for information from the database interact using the **interactive query** interface in Figure 2.3. We have *not explicitly shown* any menu-based or form-based or mobile interactions that are typically used to generate the interactive query automatically or to access canned transactions. These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a **query compiler** that compiles

them into an internal form. This internal query is subjected to query optimization (discussed in Chapters 18 and 19). Among other things, the **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Application programmers write programs in host languages such as Java, C, or C++ that



are submitted to a precompiler. The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. It is also becoming increasingly common to use scripting languages such as PHP and Python to write database programs. Canned transactions are executed repeatedly by parametric users via PCs or mobile apps; these users simply supply the parameters to the transactions. Each execution is considered to be a separate transaction. An example is a bank payment transaction where the account number, payee, and amount may be supplied as parameters.

In the lower part of Figure 1, the **runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the **system catalog** and may update it with statistics. It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory. Some DBMSs have their own buffer management module whereas others depend on the OS for buffer management. We have shown **concurrency control** and **backup and recovery systems** separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.

It is common to have the **client program** that accesses the DBMS running on a separate computer or device from the computer on which the database resides. The former is called the **client computer** running DBMS client software and the latter is called the **database server**. In many cases, the client accesses a middle computer, called the **application server**, which in turn accesses the database server.

1.2 Advantages of using DBMS approach

- ✓ Controlling Redundancy
- ✓ Restricting Unauthorized Access
- ✓ Providing Persistent Storage for Program Objects
- ✓ Providing Storage Structures and Search Techniques for Efficient Query Processing

- ✓ Providing Backup and Recovery
- ✓ Providing Multiple User Interfaces
- ✓ Representing Complex Relationships among Data
- ✓ Enforcing Integrity Constraints
- ✓ Permitting Inferencing and Actions Using Rules and Triggers
- ✓ Additional Implications of Using the Database Approach

1.3 Architecture of database

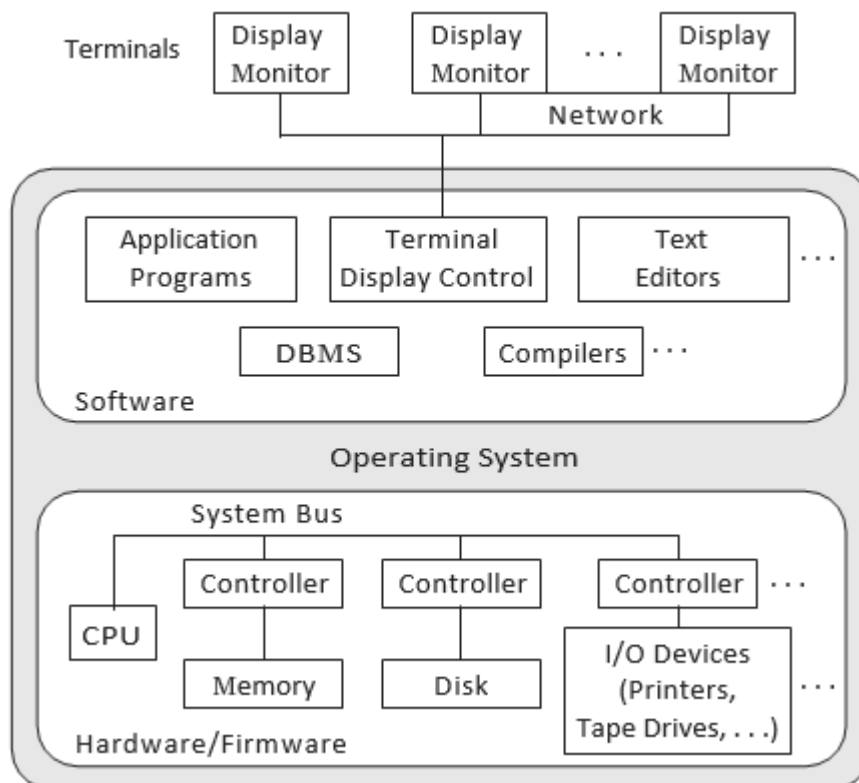


Fig 1.2: A physical, centralized and a basic client server architecture

Architectures for DBMSs have followed trends similar to those for general computer system architectures. Older architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system housing the DBMS, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.



As prices of hardware declined, most users replaced their terminals with PCs and workstations, and more recently with mobile devices. At first, database systems used these computers similarly to how they had used display terminals, so that the DBMS itself was still a **centralized** DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine. Figure 2 illustrates the physical components in a centralized architecture. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.

CHAPTER 2

SYSTEM REQUIREMENTS AND SPECIFICATIONS

2.1 Functional requirements

Functional requirements are statements of service the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

2.1.1 Valid assumptions and dependencies

The assumptions are:

- The source code pertaining to each module developed should be error free.
- The system should be user friendly
- The information, modification and updates to the database should be available to all the administrators of the library.
- The system should have a feasible storage capacity and should provide faster access to the data.
- The system should provide retrieval facilities and also facilities for quick and understandable transactions.
- Users should register themselves in order to procure administrative membership.
- Valid credentials shall only be keyed-in without which, authentication becomes unsuccessful.

The dependencies are:

- The specific hardware and software through which the product operates
- Thus, on the basis of the list of the requirements specifications the application would be developed to a full functional one.
- The end-users have to possess proper knowledge and understanding in order to work with it.
- The system should have a general report generator.
- All updates pertaining to all the modules are to be recorded and changes should happen in the referenced modules concurrently



2.1.2 Data requirement

The input consists of query to the database and the output consists of solutions for the query. The output also includes the user receiving the details of their accounts.

In this project, the inputs will be the queries as fired by the users like creating an account, adding information on the basis of various criteria, updating the information pertaining to individual records, deleting individual records, searching and retrieving records etc.

Now, the solutions for the queries as the respective outputs will be visible when the users request the server through the GUI

2.1.3 External interface requirement

GUI

The application that is developed provides a good graphical user interface for the user or the administrator who operates the system, performs the required tasks such as inset, update, delete and search on the basis of various criteria.

General features of the GUI

- Provides stock verification search facility based on different criteria.
- It allows the user to generate quick reports and export data to different formats.
- All the modules designed are collectively integrated to form the Menu Driven Interface (MDI).
- The design of the interface is simple and all the modules within it, follow a standard template.
- The user interface must be dedicated to the login/logout module.

Login interface or the User Authentication Window

In case the user is not yet registered, the registration can be done through this window, Once the registration is successful, authenticity can be established through the user authentication window which on successful authentication redirects the user to the menu driven interface.

➤ Search

The user of the site can enter the name of the resources he/she looking for based on the criteria mentioned in the module.

➤ Menu-oriented views



The menu may further show the categories based on which the whole system is supposed to be administered.

➤ **Control Panel**

This panel will allow the user to update/delete and refresh the contact of records based on particular criterion.

2.1.4 Operational requirements

The product will operate on windows 10 environment. The only requirements to use this product would be a web browser such as google chrome, Microsoft edge, apple safari, mozilla Firefox etc.

The basic hardware configuration includes:

	Chrome	Opera	Safari
Processor Windows	Pentium 4	Pentium II	500-MHz Pentium-class
Process Mac	Intel	Intel	
Min RAM	128 MB	128 MB	256 MB
Recommended RAM		256 MB	
Min Disk Space	100 MB	20 MB	unknown
Rec. Disk Space		100 MB	
Rec. Disk Space 64-bit			
Windows	Windows XP SP2	Windows 2000	Windows XP SP2
OS X	OS X 10.5.6	Mac OS X 10.5	OS X 10.5.8
Linux	Ubuntu 10.04	Any recent	Not available
	Debian 6		
	OpenSuse 11.3		
	Fedora Linux 14		

Fig 2.1: Minimum browser requirements



2.2 Non-Functional Requirements

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on development process, and constraints imposed by standards. Non functional requirements often apply to the system as a whole, rather than individual features or services.

2.2.1 Performance requirement

The proposed system that we are going to develop may be used as a chief performance system within the different shops, through secured interaction with the customer.

Therefore, it is expected that the database would perform functionally, all the requirements that are specified by the university.

- The performance of the system should be fast and accurate.
- The website shall handle expected and non-expected errors in ways that prevent loss in information and long downtime period. Thus, it should have inbuilt error testing to identifying and handle exceptions.
- The system should be able to handle large amount of data.

2.2.2 Product usability requirements

- **Availability:** The system is available 100% for the user and can be used on a 24x7 basis.
- **Accuracy:** The system shall accurately provide real time information taking into consideration various concurrency issues.
- **Reliability:** The system has to be 100% reliable due to the importance of data and the damages that can be caused by incorrect or incomplete data.
- **Maintainability:** Changes must be verified once per day at least.
- **Portability:** The system should also be portable.

2.2.3 Requirements attributes

- There may be administrators creating the project, so , all of them would have the rights to create changes to the system.
- The project should be open source.
- The quality of the database is maintained in such a way so that it can be very friendly to all the users of the database.
- The users should be able to easily install the software on the system.



2.3 Software requirements specification

The application is developed using HTML, CSS and JavaScript as the front end which is supported by Node.js and MongoDB as the backend for accessing and connecting the front end to the database.

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server- and client-side scripts.

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL).

3.3.1 Specifications of the software used for application development

The following software are required to develop the application:

- **Front end: Atom** is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in Node.js, and embedded Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js. It is written in CoffeeScript and Less[3].
- **Back end : Node.js** is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. Node.js[4] lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server- and client-side scripts[2][1].



- **Database :** MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL)[5].
- **Operating system: Windows 10** is a series of personal computer operating systems produced by Microsoft as part of its Windows NT family of operating systems. It is the successor to Windows 8.1, and was released to manufacturing on July 15, 2015, and broadly released for retail sale on July 29, 2015. Windows 10 receives new builds on an ongoing basis, which are available at no additional cost to users, in addition to additional test builds of Windows 10 which are available to Windows Insiders. The latest stable build of Windows 10 is Version 1909 (November 2019 Update). Devices in enterprise environments can receive these updates at a slower pace, or use long-term support milestones that only receive critical updates, such as security patches, over their ten-year lifespan of extended support.

2.4 Hardware requirements specification

The following requirements are needed to develop the application:

- Computer that has 500MHz or faster processor.
- Minimum 128 Mb RAM (256 MB recommended).
- Minimum 100 Mb Disk space.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 ER Model

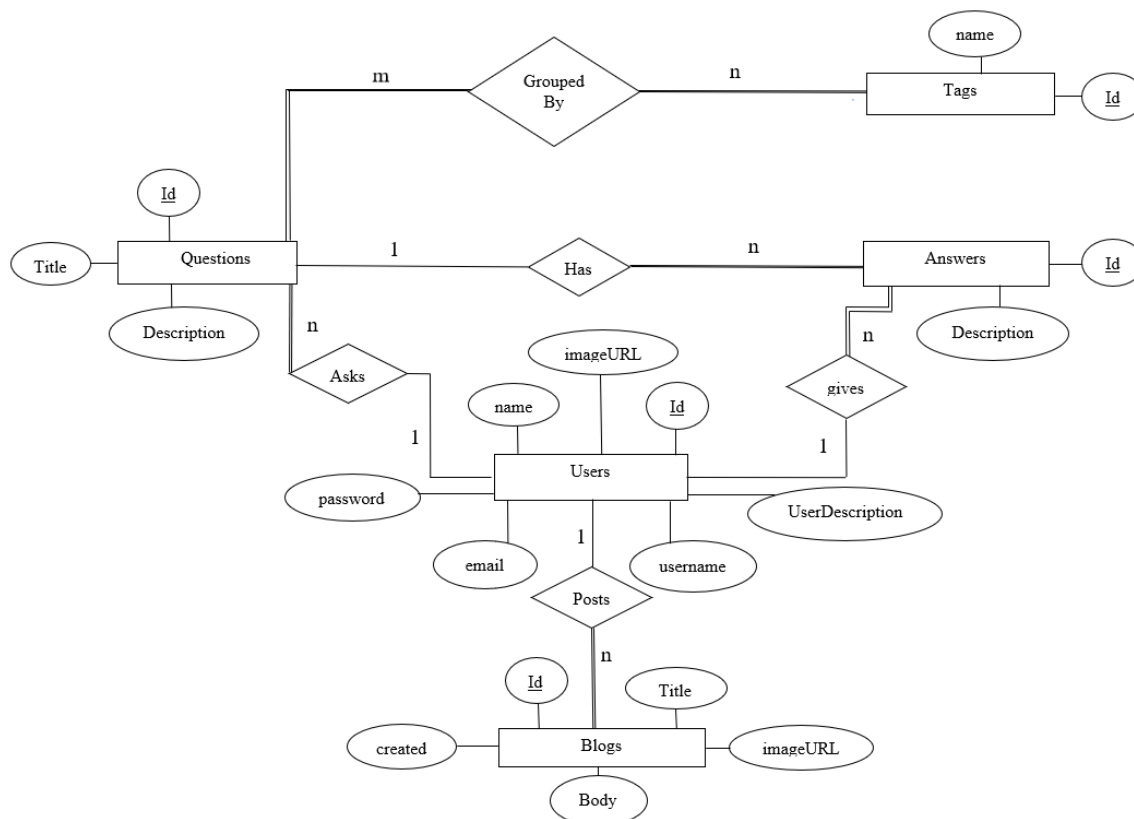


Fig 3.1: ER diagram of stack overflow website

The ER model describes data as entities, relationships and attributes. The basic object that the ER model represents is an entity, which is a thing in the real world with an independent existence[6].

An entity may be an object with a physical existence or it may be an object with a conceptual existence. Here each entity has one or more attributes. Fig 3 shows the entities with either respective attributes, relationships, with cardinality ratios which further mean something about the quality of participation that may be offered by the entities for their relationships between others



3.2 Schema diagram

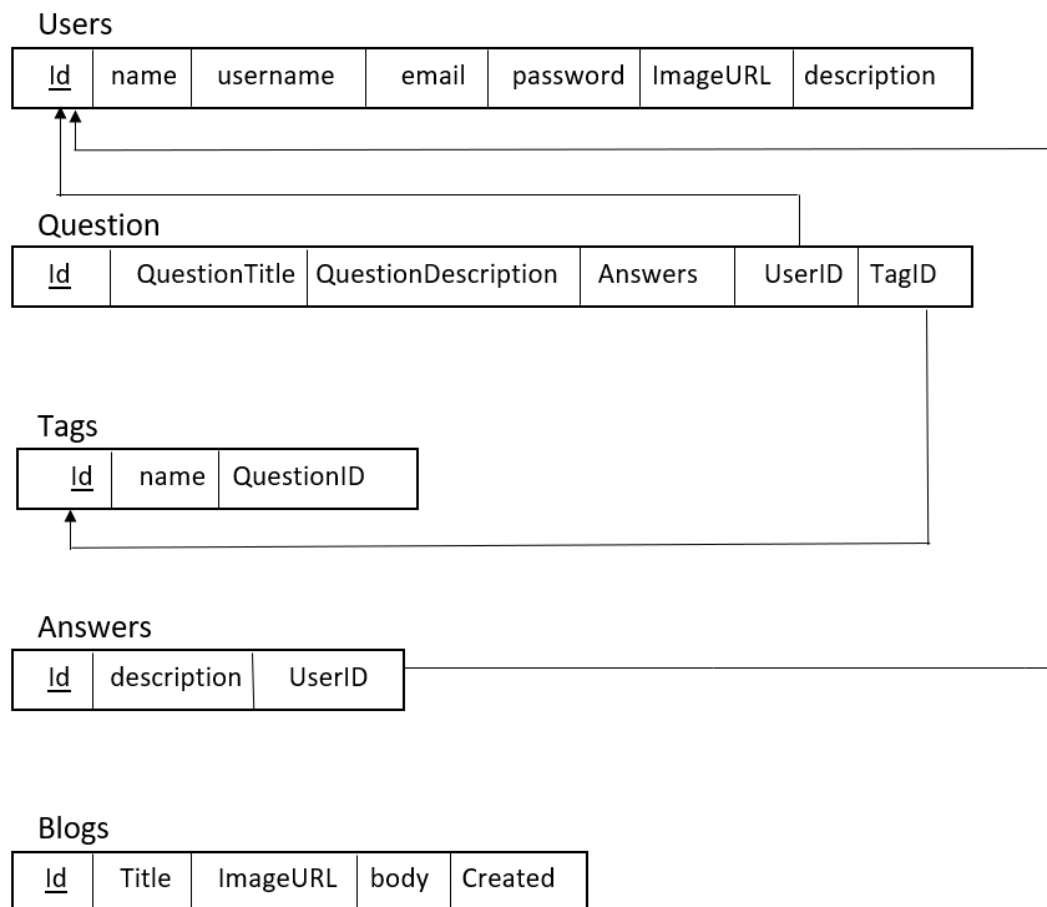


Fig 3.2: Schema Diagram of Stack Overflow website

Logical schema diagram displaying the referential integrity constraints pertaining to a simple question and answer website[7][8].



3.3 Description of relations (Tables)

3.3.1 User Schema

Name	Type	Description
Id	ObjectID	Unique Id belonging to a user
Name	String	A user's personal name
Username	String	A user's username in the website
Email	String	User's email
Password	String	User's password
Description	String	A brief description about the user

Table 3.1 : User Schema

3.3.2 Answer Schema

Name	Type	Description
Id	ObjectID	Unique id for that answer
Answer	String	The complete answer description
User	ObjectID	Refers to the user who posted that particular answer

Table 3.2: Answer Schema

3.3.2 Tags Schema

Name	Type	Description
Id	ObjectID	Unique id for that tag
Name	String	The name of the tag
Question	Array	Refers to the question that have the tag name.

Table 3.3: Tags Schema

3.3.4 Blogs Schema

Name	Type	Description
Title	String	Title of blog
Image	String	An image URL
Body	String	The description the blog
Created at	Date	Time at which a blog was posted

Table 3.4: Blogs Schema



3.3.5 Question Scheme

Name	Type	Description
Id	ObjectID	Unique ID for that question
QuestionTitle	String	Title of that question
QuestionDescription	String	Description of that question
QuestionTags	Array	Tags belonging to that question
UpdatedAt	Date	Date
Answers	Array	Answers belonging to that question
userId	ObjectID	User who asked that question

Table 3.5: Question Schema

CHAPTER 4

IMPLEMENTATION

4.1 Code snippets for front end

4.1.1 Stack overflow main page HTML

```
<%- include("partials/stackHeader"); -%>

<% questions.forEach(function(element){ %>

<div class="card bg-light">
  <div class="card-body">
    <h4 class="card-
title"><a href="/questionAnswer/<%= element._id %>" class="questionTitle"><%=
element.questionTitle %></a></h4>
    <p class="card-text mb-0">
      <pre class="mb-
0"><%= element.questionDescription.substring(0,130)+" ..." %></pre>
    </p>
    <% element.questionTags.forEach(function(item){ %>
    <a href="/tags/<%= item._id %>" class="btn btn-dark btn-
sm"><%= item.name %></a>
    <% }) %>
    <p class="card-text ml-auto mt-2 mb-0 mr-1">
      <strong>Asked by :</strong>
      <%= element.userId.username %>
      <strong>, on : </strong>
      <%= element.updatedAt.toString().substring(0,25) %>
    </p>
  </div>
</div>
<% }) %>

<%- include("partials/stackFooter"); -%>
```



4.1.2 User registration HTML

```
<form action="/register" class="form-signin" method="POST">
  <div class="text-center mb-4">
    
    <i class="fab fa-stack-overflow fa-6x"></i>
    <h1 class="h3 mb-3 font-weight-normal">Stack OverFlow</h1>
  </div>
  <div class="form-row">
    <div class="form-label-group col-md-6">
      <input type="text" class="form-control" name="firstName" placeholder="First Name" autofocus>
      <label> First Name:</label>
    </div>
    <div class="form-label-group col-md-6">
      <input type="text" class="form-control" name="lastName" placeholder="Last Name">
      <label>Last Name:</label>
    </div>
  </div>
  <div class="form-label-group">
    <input type="text" class="form-control" name="profileName" placeholder="First Name" autofocus>
    <label>UserName:</label>
  </div>
  <div class="form-label-group">
    <input type="email" id="inputEmail" name="email" class="form-control" placeholder="Email address" autocomplete="off" required >
    <label for="inputEmail">Email</label>
  </div>
  <div class="form-label-group">
    <input type="password" id="inputPassword" name="password" class="form-control" placeholder="Password" autocomplete="off" required>
    <label for="inputPassword">Password</label>
  </div>
  <div class="form-label-group">
    <textarea id="description" class="form-control" name="description" rows="2" cols="80" placeholder="describe yourself"></textarea>
  </div>
  <button class="btn btn-lg btn-dark btn-block" type="submit">Sign up</button>
  <a href="/login" class="btn btn-lg btn-secondary btn-block">Login?</a>
  <p class="mt-5 mb-3 text-muted text-center">&copy; Stack overflow</p>
</Form>
```



4.1.3 Ask question HTML

```
<h1>Ask a public question</h1>
<form action="/compose" method="post">
  <div class="form-group">
    <label for="title"><strong>Title:</strong></label>
    <input type="text" name="titleText" class="form-
control" placeholder="What's your programming question? Be specific">
    <small id="emailHelp" class="form-text text-
muted">Be specific and imagine you're asking a question to another developer</
small>
  </div>
  <div class="form-group">
    <label for="post"><strong>Question Body:</strong></label>
    <textarea class="form-
control" name="postText" rows="8" cols="80"></textarea>
    <small id="emailHelp" class="form-text text-
muted">Include all the information someone would need to answer your question</
small>
  </div>
  <div class="form-group">
    <label for="title"><strong>Tags</strong></label>
    <input type="text" class="form-
control" placeholder="eg:(android html c#)" name="tagsText">
    <small id="emailHelp" class="form-text text-
muted">Add up to 5 tags to describe what your question is about</small>
  </div>

  <button type="submit" name="button" class="btn btn-
primary">Post your question</button>
</form>
```

4.1.4 Question and Answer HTML

```
<h1 class="display-4"><%= title %></h1>
<p><strong>Asked by : </strong><%=user.username%></p>
<hr style="border-top: solid #F0EFEF 1px">
<div class="m1-5">
  <p style="white-space: pre-line"><%= body %></p>

</div>
<!-- <hr style="border-top: solid #F0EFEF 1px"> -->
<br>
<% if(answers.length != 0){ %>
```



```

    <h3 class="inlineHeading"><%= numberOfAnswers %></h3>
    <h3 class="inlineHeading"> Answers</h3>
    <hr style="border-top: solid #F0EFEF 1px">
<% } %>

<% answers.forEach(function(answer){ %>
    <div class="content">
        
        <p style="white-space: pre-line"><%= answer.answerDescription %></p>
    </div>
    <hr style="border-top: solid #F0EFEF 1px">
<% }) %>

<% if (cuser.email !== user.email) { %>
    <form action="/questionAnswer" method="post">
        <div class="form-group">
            <label for="post"><strong>Your Answer:</strong></label>
            <input type="text" style="visibility : hidden" name="questionId" value="<
%= questionId %>">
            <textarea class="form-
control" name="answerText" rows="8" cols="80" placeholder="your answer goes he
re"></textarea>
            <small id="emailHelp" class="form-text text-
muted">Be as descriptive as possible while answering.</small>
        </div>
        <button type="submit" name="button" class="btn btn-
dark">Post your answer</button>
    </form>
<% } %>

<% if(user.email == cuser.email) {%>
<a class="btn btn-
dark" href="/questionAnswer/<%=questionId %>/update">Update</a>
<a class="btn btn-
dark" href="/questionAnswer/<%=questionId %>/delete">Delete</a>

<%}%>

```



4.2 Code snippets for back end

4.2.1 Database connection

```
mongoose.connect("mongodb://localhost:27017/stackUsers", {
  useNewUrlParser: true,
  useFindAndModify: false,
  useCreateIndex: true,
  useUnifiedTopology: true
});
```

4.2.2 User registration

```
app.post("/register", function(req, res) {
  const userData = new User({
    _id: new mongoose.Types.ObjectId(),
    name: req.body.firstName + " " + req.body.lastName,
    username: req.body.profileName,
    email: req.body.email,
    password: md5(req.body.password),
    description : req.body.description
  });
  user=userData;
  userData.save(function(err){
    if(err){
      console.log(err);
    }else{
      passport.authenticate("local")(req ,res ,function(){
        res.redirect("/stackoverflow");
      });
    }
  });
});
```



4.2.3 Insert Questions

```
app.post("/compose", function(req, res) {
  const questionTags = req.body.tagsText.split(" ");
  const question = new Question({
    _id: new mongoose.Types.ObjectId(),
    questionTitle: req.body.titleText,
    questionDescription: req.body.postText,
    userId : req.user._id
  });
  question.save(function(err) {
    if (!err) {
      res.redirect("/stackoverflow");
    } else {
      console.log(err);
    }
  });
  questionTags.forEach(function(tag) {
    Tag.findOne({
      name: tag
    }, function(err, results) {
      if (err) {
        console.log(err);
      }
      if (!results) {
        var newTag = new Tag({
          _id: new mongoose.Types.ObjectId(),
          name: tag
        });
        newTag.question.push(question._id);
        newTag.save(function(err) {
          if (err) {
            console.log(err);
          }
        });
      }
      Question.findOneAndUpdate({
        _id: question._id
      }, {
        "$push": {
          questionTags: newTag._id
        }
      }, function(err, success) {
        if (err) {
          console.log(err);
        }
      });
    } else {
      //tag is found
      Tag.findOneAndUpdate({
```



```
        name: tag
      }, {
        "$push": {
          question: question._id
        }
      }, function(err, success) {
        if (err) {
          console.log(err);
        }
      });
    Tag.findOne({
      name: tag
    }).populate('question').exec(function(err, foundTag) {
      Question.update({
        _id: question._id
      }, {
        "$push": {
          questionTags: foundTag._id
        }
      }, function(err, rawResponse) {
        if (err) {
          console.log(err);
        }
      });
    });
  }
});
});
```

CHAPTER 5

DISCUSSION AND SCREENSHOTS

5.1 Home page

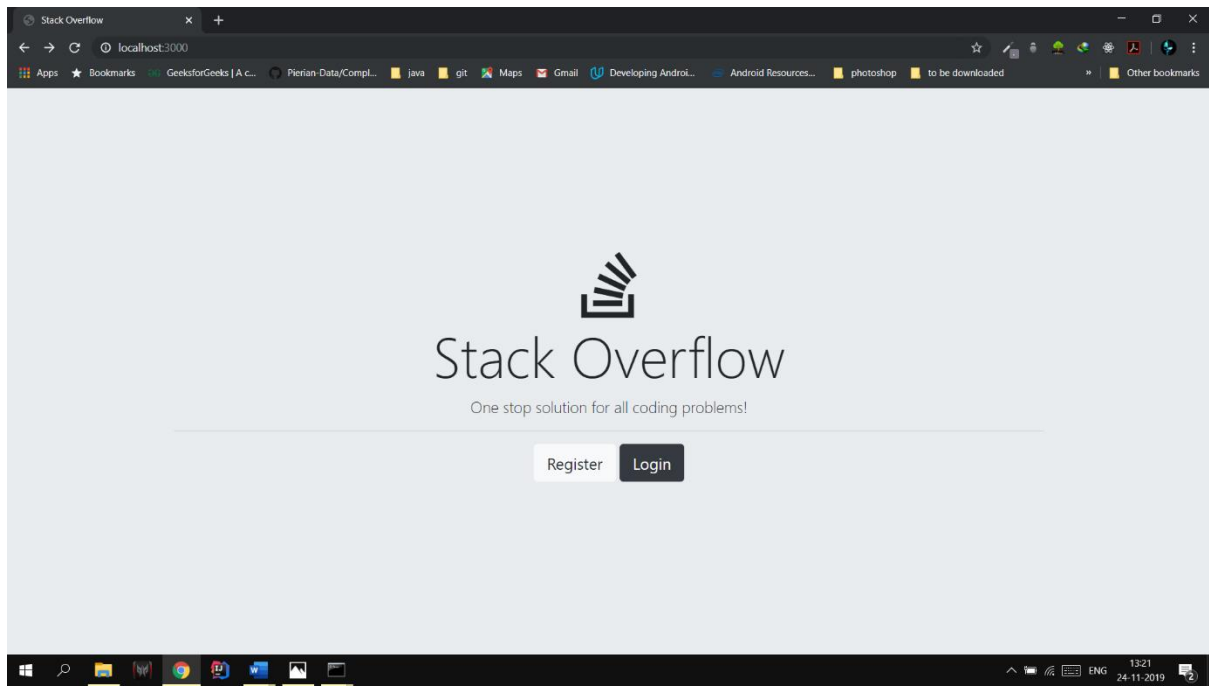
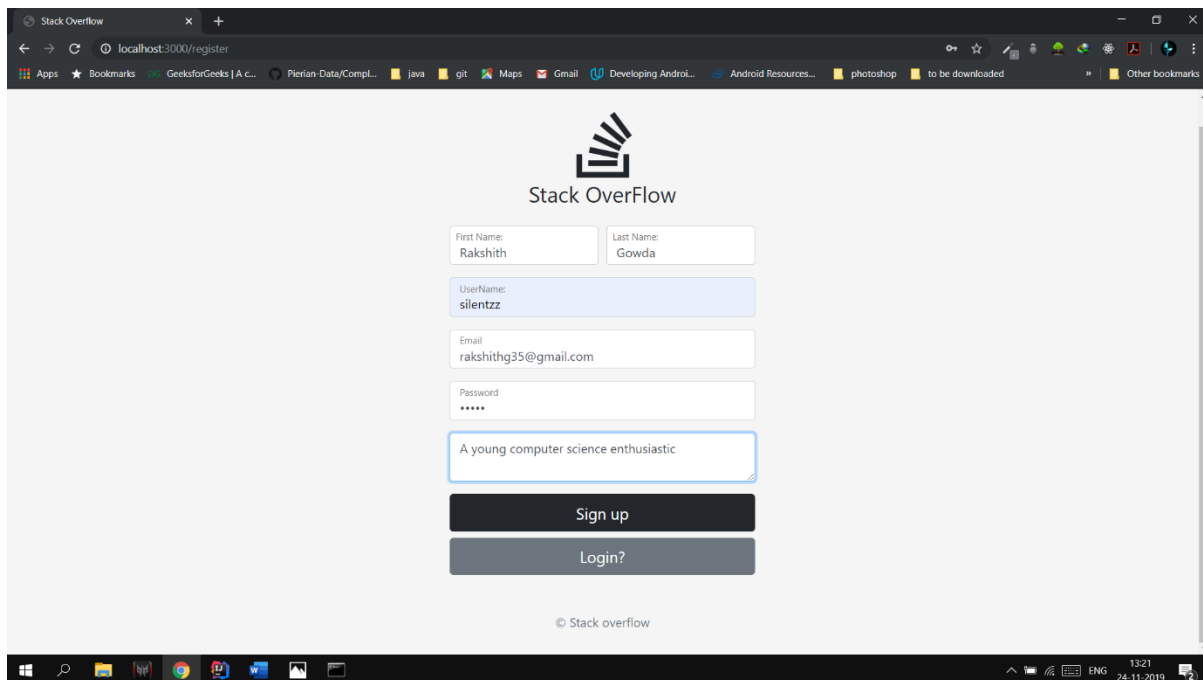


Fig 5.1: Stack Overflow home page

Home page is the page that a user accesses the stack overflow website. The user is asked to register if they are visiting this website for the first time else, they can login with their credentials to access stack overflow main page.

5.2 Register page

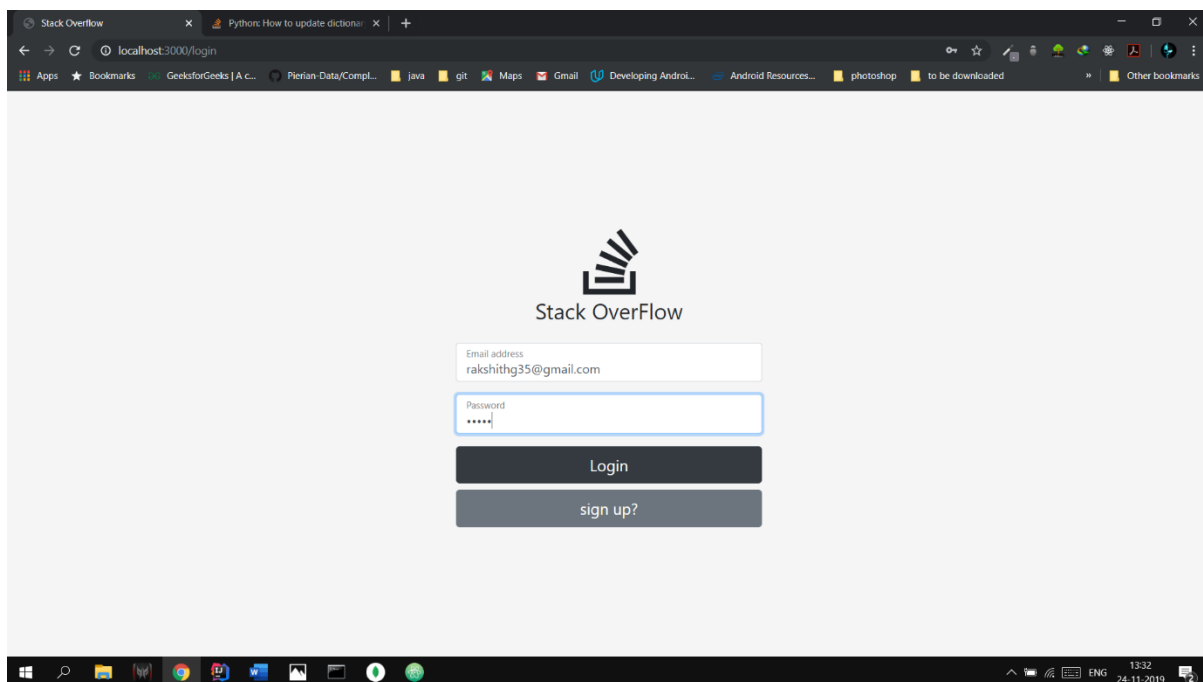


The screenshot shows a web browser window with the URL `localhost:3000/register`. The page features the Stack Overflow logo at the top. Below the logo, there are several input fields for registration: "First Name" (containing "Rakshith"), "Last Name" (containing "Gowda"), "UserName" (containing "silentzz"), "Email" (containing "rakshithg35@gmail.com"), "Password" (masked with "*****"), and a text area for a bio (containing "A young computer science enthusiastic"). At the bottom of the form are two buttons: "Sign up" and "Login?". The footer of the page says "© Stack overflow".

Fig 5.2: Sign up page

A new user should enter all the above details to in order successfully sign up for stack overflow page. If a user doesn't fill any one the details then the page is redirected to the same page.

5.3 Login page



The screenshot shows a web browser window with the URL `localhost:3000/login`. The page features the Stack Overflow logo at the top. Below the logo, there are two input fields for login: "Email address" (containing "rakshithg35@gmail.com") and "Password" (masked with "*****"). At the bottom of the form are two buttons: "Login" and "sign up?".

Fig 5.3: Login page

A user can login using his email and password provided during signing up to the page.



5.4 Main page

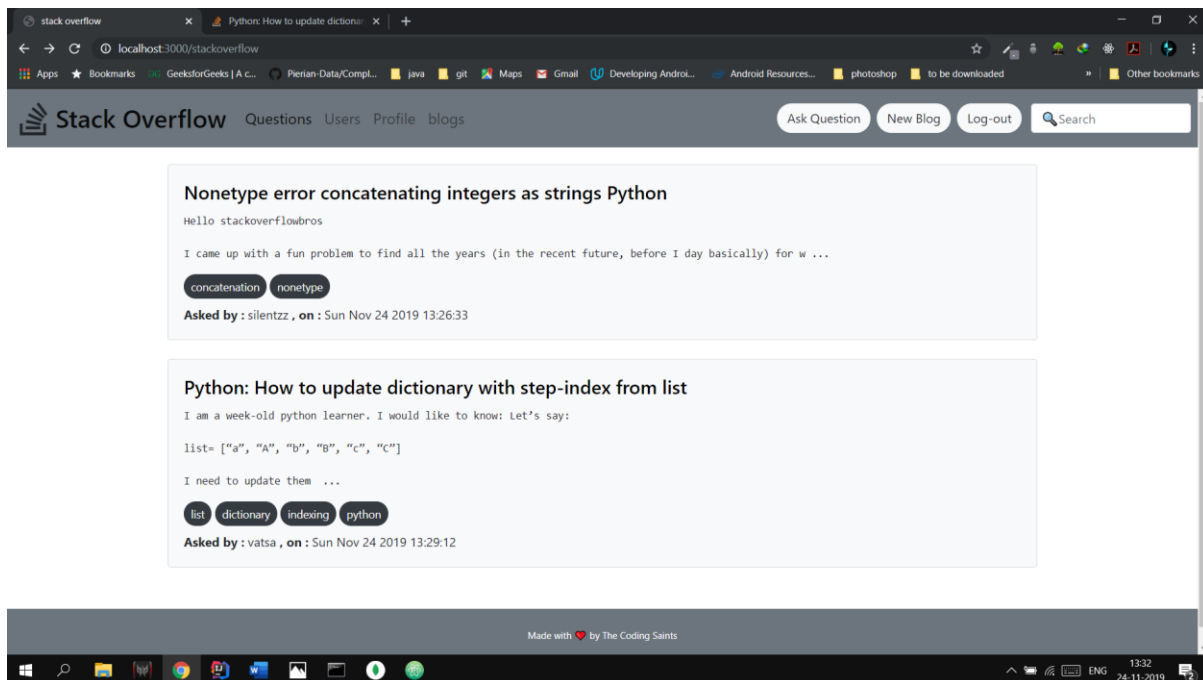


Fig 5.4: Main page

This page consists of all questions asked by all the users of stack overflow website. It also shows details like which user asked the question and when the question was asked. A user can also search for the question and related questions to his query popup on the main page.

5.5 Ask question

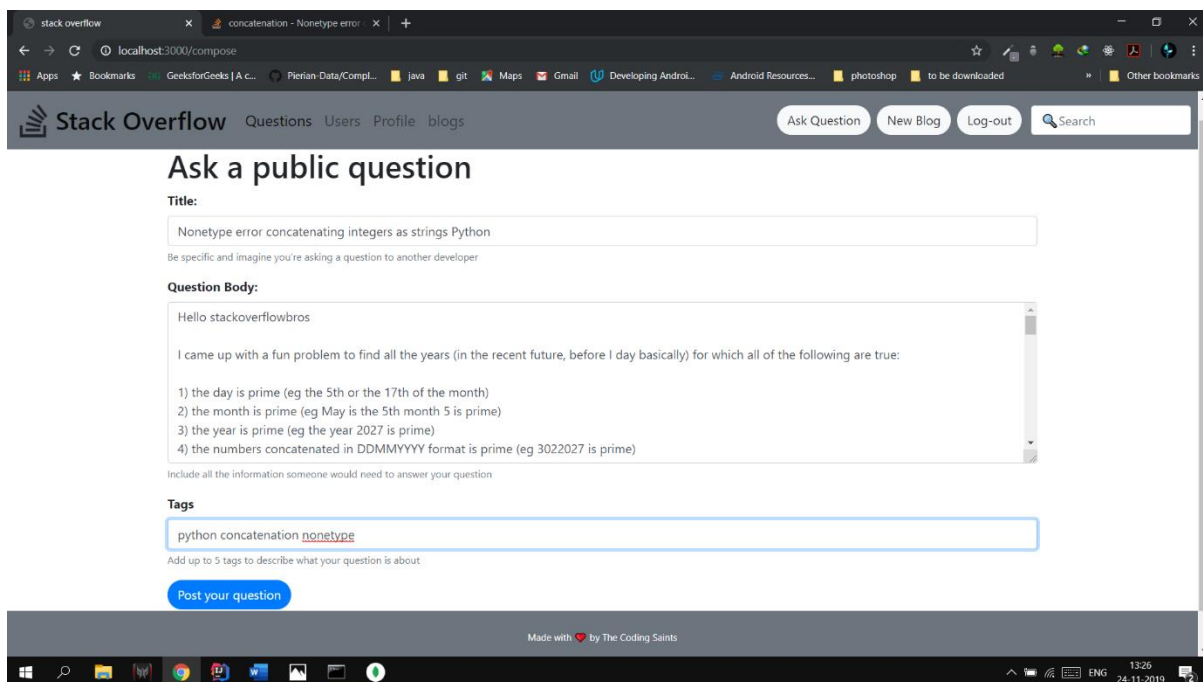


Fig 5.5: Question composition



A logged in user can ask a question by clicking on ask question button on the navbar and is redirected to this page. A user should provide a suitable title for his question and a complete description about the question and also the topics related to this question in the tags question.

5.6 All users

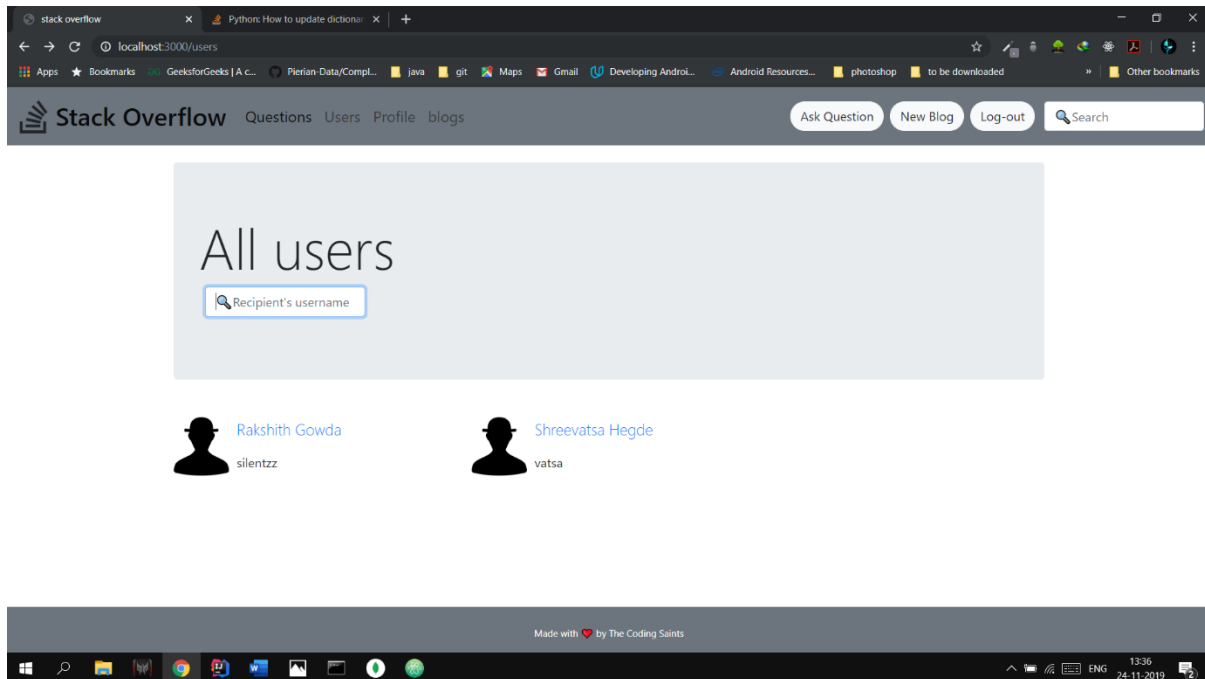


Fig 5.6: All users

This page displays all the users who have been successfully signed up to the stack overflow website. A user can search for another user in the search bar provided in the page by using the username of the user whom he wants to find.



5.7 New blog

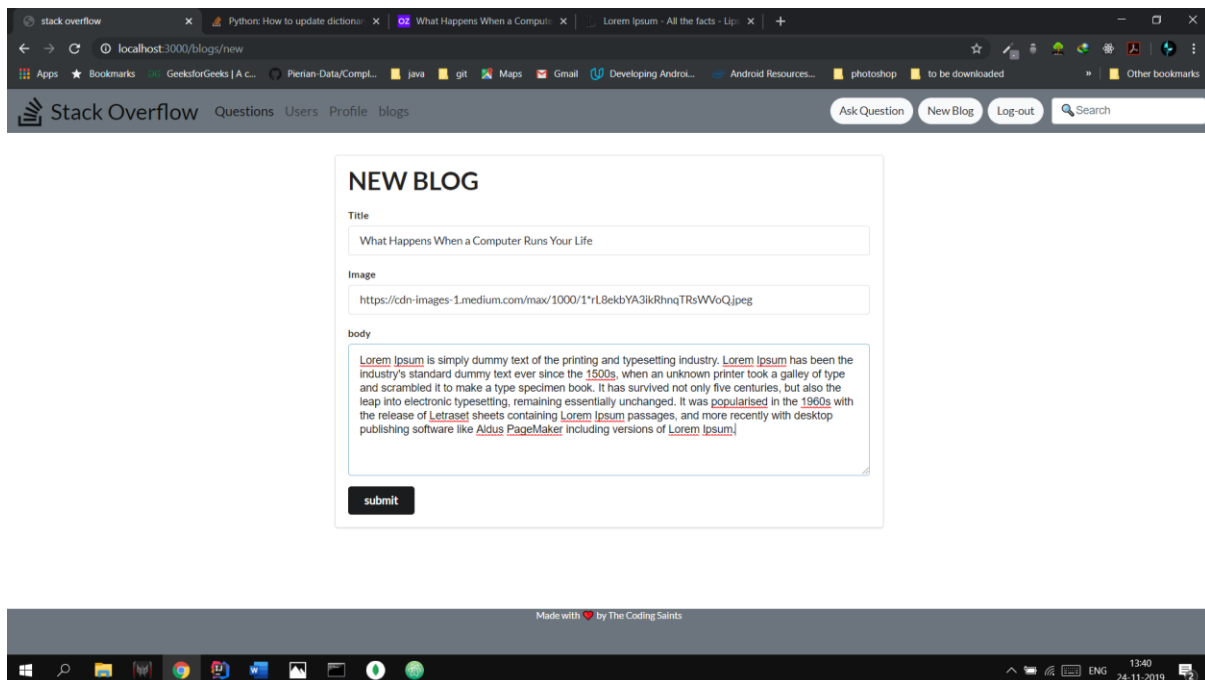


Fig 5.7: Compose new blog

A user can post a new blog by clicking the new blog button the nav bar which redirects the user to new blog page. A user can enter the title of his blog a image url for his blog and also the description of his blog.

5.8 All blogs

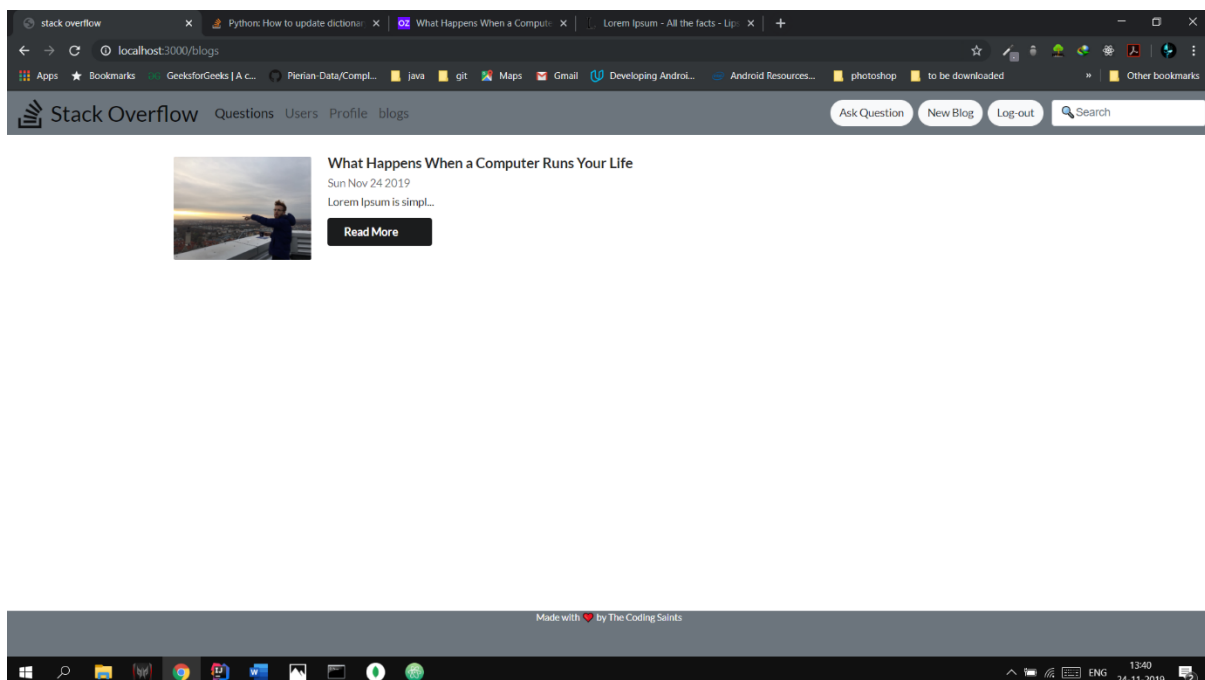


Fig 5.8: All blogs

All the blogs posted by users are shown in the all blogs page. A user can click on read more button on a particular blog to read the details of that blog.

5.9 Question Answer

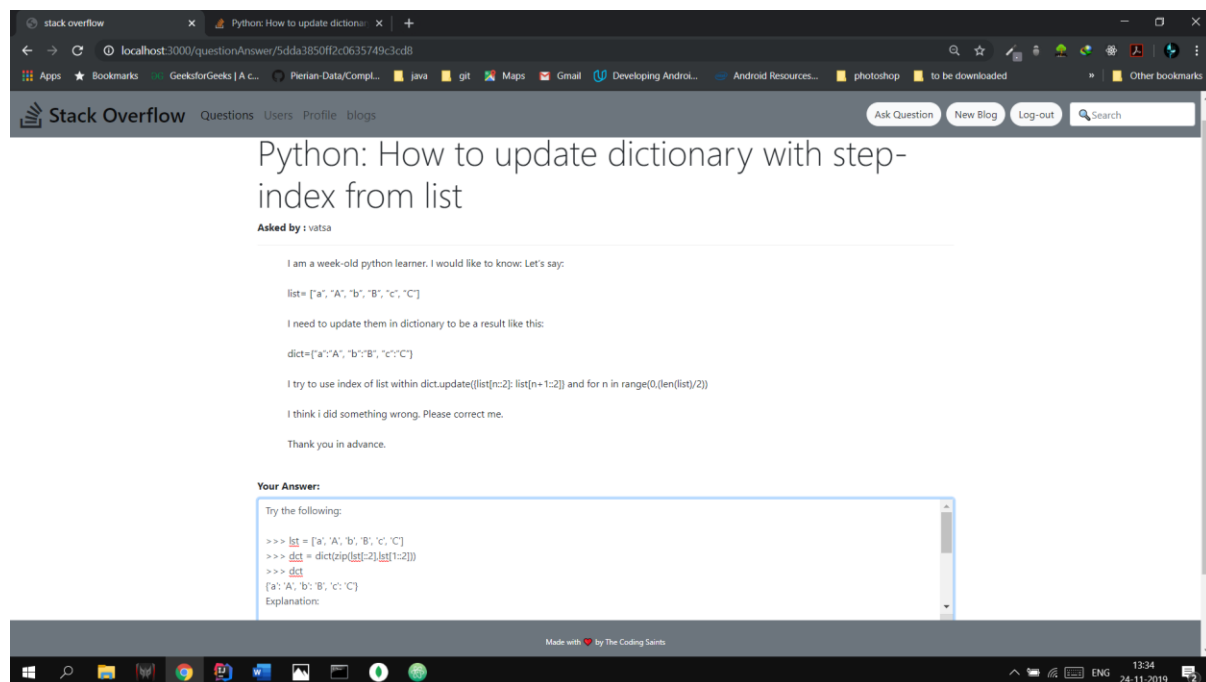


Fig 5.9: Question answer

A user can click on any question in the main page to be redirected to the question and answer page of that question. A user can also answer to a question posted by some other user. If a question was posted by the user then a user has the option to update or delete the question.

5.10 Logout

If a user wishes to logout a particular session then he/she may do so by clicking on the Log-out button on the navbar. The user is redirected to the login page once he logs out.

5.11 Result

The project or the application was thus developed, by considering the valid assumptions and dependencies, and by carrying out a careful analysis on the requirements that were specified.



CONCLUSION

The “Stack overflow” question and answer site has been successfully computed and was tested successfully by taking into consideration the “test cases”. It is user-friendly, and has required options, which can be utilized by the user to perform the desired operations.

The software or the so-called website was developed using HTML, CSS and JavaScript as the front end and Node.js and MongoDB as backend in WINDOWS environment. It, is also hereby ensured that the goals are met by the website. The goals are:

- Optimum utilization of resources
- Efficient management of records
- Simplification of operations
- Less processing time and quick retrieval of records.
- Characteristic of being portable and flexible for further futuristic enhancement.

Stack overflow allows the users to ask questions regarding programming and computer science in general. This was an effort to develop a simple question and answer website to store questions, answers, blogs and details of users. I hope you will like it.



BIBLIOGRAPHY

- [1] The Complete 2020 Web Development Bootcamp - Angela YU
- [2] The Web Developer Bootcamp – Colt Steel
- [3] Front End Web Developer – Udacity
- [4] Full Stack Web Developer - Udacity
- [5] MongoDB - The Complete Developer's Guide - Maximilian Schwarzmüller
- [6] <https://stackoverflow.com/>
- [7] <https://www.udemy.com/>
- [8] <https://udacity.com/>