TASKDO-TASK MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

GOVARTHANA K	**830119104013**
MOHAMED FARHAN S	**830119104023**
SIVASANKARANARAYANAN D	**830119104045**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT COLLEGE OF ENGINEERING SRIRANGAM

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"TASKDO-TASK MANAGEMENT SYSTEM"** is the bonafide work of **GOVARTHANA K (830119104013) , MOHAMED FARHAN S (830119104023) , SIVASANKARANARAYANAN D (830119104045)** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Prof.P.Vanitha Muthu, M.Tech.,	**Prof.H.Sheik Mohideen, M.E.,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Dept of Computer Science Engineering ,	Dept of Computer Science Engineering
Government College of Engineering Srirangam,	Government College of Engineering Srirangam,
Tiruchirapalli-620012	Tiruchirapalli -620012

Submitted for Semester Mini-Project work (CS8611) viva-voice examination held on _________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We, first of all thank GOD ALMIGHTY for giving us a golden opportunity to express our individual and technical skills in the field of Computer Science and Engineering.

We express our gratitude to our beloved Principal **Dr.V.M.SHANTHI M.E.,Ph.D.,**for giving us a chance to complete our higher education in one of the reputed government institution running under her magnificent leadership.

We are extremely thankful to our Head of the Department, **Prof.P.VANITHA MUTHU M.Tech.,** Department of Computer Science and Engineering, for her support, motivation, inspiration and tireless encouragement.

We are immensely pleased to thank our project guide, **Prof. H. SHEIK MOHIDEEN, M.E.,** Assistant Professor, Department of Computer Science and Engineering, for his valuable comments and suggestions.

We also express our thanks to our Project Co-coordinator, **Prof.P.SARANYA, M.E.,** Assistant Professor, Department of Computer Science and Engineering, for ideas given by her and constant inspiration throughout our project period.

It is a great opportunity to express my sincere thanks to my parents, friends and all the people who have contributed to the successful completion of my project work through their support, encouragement and guidance.

ABSTRACT

Many working professionals like headmasters and professors often find lack of time and time management as problems for successful task accomplishment. One of the key reasons for failure in task accomplishment is inefficient planning of the tasks. It’s tough to achieve big goals when they seem to overwhelm you. As a consequence, it’s easy to procrastinate on them. One of the best ways to solve this problem is by breaking down a large goal into smaller parts. This is where a task management apps becomes significant. It lets you make large and overwhelming projects manageable. Also, you get more done by focusing on high-value activities. Once you have a list of things you need to-do, it’s much easier to prioritize the tasks on it. This will ensure you’re always working on the right things. Otherwise, it’s easy to fall into a circle of doing what seems easiest or most urgent. Doing this may result skipping important things that don’t require your immediate attention. There are many task management and to-do-list applications, but most of them do not advise on optimal task management and guidance for optimal performance. This problem has driven us to contribute a task management system which is used by the headmasters and professors to complete their tasks. This system also allows headmasters to rate the tasks of the professors in their respective department.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTML	Hyper Text Markup Language
HOD	Head of the Department
XML	Extensible Markup Language
DOM	Document Object Model
UML	Unified Modeling Language
PHP	Hypertext Pre Processor
IOS	Iphone Operating System
UI	User Interface
OS	Operating System
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator
API	Application Programming Interface
NPM	Node Package Manager

CHAPTER 1

INTRODUCTION

Striving to be productive remains a challenge for headmasters and professors. Various researches and surveys state that 70% of employees work beyond scheduled time and on weekends; more than half cited "self-imposed pressure" as the reason. The survey, conducted by Greenfield Online, found that nearly 54 percent of college professors felt they would be more productive if they "got organized and stayed organized. To recover from these sorts of problems, spending time organizing daily tasks would be helpful. Research says that for every hour of planning, 3 to 4 hours are saved from redundancy, waiting for information, not being prepared and poorly managed tasks. With new age advancements in information technology, we have many applications which focus on time and task management. Task and time management tools such as Todoist and Wunderlist allow users to add and track tasks. Some tools have implemented some visual analytics to help users to understand their productivity, and how it changes over time. Driven by the importance of time and task management, and lack of the tools which suggest a specific task for a specific time, we are inspired by the research question: “Based on the user’s history of completed tasks, can we recommend certain task types to be done at certain days of The week and times of day to increase a user’s productivity?” “Can we make headmasters to assign tasks for their department professors and rate their tasks based on their performance” . To answer this question, we contribute TaskDo, a task management system. The headmaster can rate the professors tasks once the professors have done the tasks and mark as complete.

1.1 Problem statement

Scheduling and organizing day to day workflow without a task management system, may seem hard. Whether you’re looking to achieve more of your goals or controlling your time better, a task management system will help you. You can get a positive boost to your career by becoming the person who is always on top of things and feel good every day. The aim of this project is to create a web application that will help headmasters and professors in the college management to track their tasks and do them very effectively and within the time range anywhere using only a browser in a Mobile Phone, Personal computer or even laptops

1.2 Motivation

Officially, task management is the process of managing a specific task over the course of its life cycle. As an important part of project planning, it includes identifying, planning, monitoring, and following up on the various activities involved in completing a particular job. Task management is the process of managing a specific task over the course of its life cycle. It involves identifying, planning, monitoring, and following up on the activities involved in completing a job.

- Helps you achieve your individual goals (aka personal task management).
- Helps you collaborate and share knowledge to meet group objectives (aka team task management).

- Enhances your decision-making process through map-based task planning and tracking (aka visual task management)

CHAPTER 2

LITERATURE SURVEY

2.1 TITLE: "USER TRENDS MODELING FOR A CONTENT-BASED RECOMMENEDER SYSTEM"

AUTHOR: BAGHER, R. C, HASSANPOUR, H., AND MASHAYEKHI, H.

YEAR : 2018

OVERVIEW:

The concept of trend to capture the interests of user in selecting items among different group of similar items. The trend based user model is constructed by incorporating user profile into a new extension.dd-CRP which is a Bayesian Nonparametric model, provides a framework for constructing an evolutionary user model that captures the dynamics of user interests.

ADVANTAGES

- Constructs an appropriate model to estimate the user interests is the major task of this recommender system.

DISADVANTAGES

- This system is not optimized for task management.

2.2 TITLE:"TIME MANAGEMENT AND PERSONAL DEVELOPMENT"

AUTHOR: JOHN ADAIR AND MELANIE ALLEN

YEAR : 2018

OVERVIEW:

Time Management and Personal Development stands out from other books in an important way. It is a book to use for our own self-development. This can be used in several ways. Some of the possibilities to consider are Individual study, workshops ,seminars and distance or open learning.

ADVANTAGES

- It gives lot of ideas and details about the time management.

DISADVANTAGES

- It only has ideas and information and not the ways for execution.

2.3 TITLE: "MAMAY WEBPUM: A WEB BASED RECOMMENDATION SYSTEM"

AUTHOR: PROF. SANOBER SHAIKH, ADHITHYARAM , ARYAK DESHPANDE

YEAR : 2021

OVERVIEW:

Recommender systems are have become a staple in this era of the internet economy. They help in reducing the overload of information by providing customized information access. Modeling, programming, and deploying these recommender systems have allowed businesses to enhance revenues and retain customers. To be beneficial to the user and clients as users only see relevant content and are not bombarded with unnecessary products and clients get better engagement. Hence it has become imperative for businesses nowadays to build smart recommendation systems and make use of the past behavior of their users.

ADVANTAGES

- This method of recommendation uses movie features to recommend other similar movies to what the user has already liked.

DISADVANTAGES

- It has no rating system and hence a drawback.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware requirements

RAM	: 1 GB Ram and more
Processor	: AMD Processor / Intel Processor
Hard Disk	: 40 GB min

3.2 Software requirements

Operating System	: Platform independent
IDE	: Visual Studio Code
Browsers	: All browsers

CHAPTER 4

ARCHITECTURE DIAGRAM

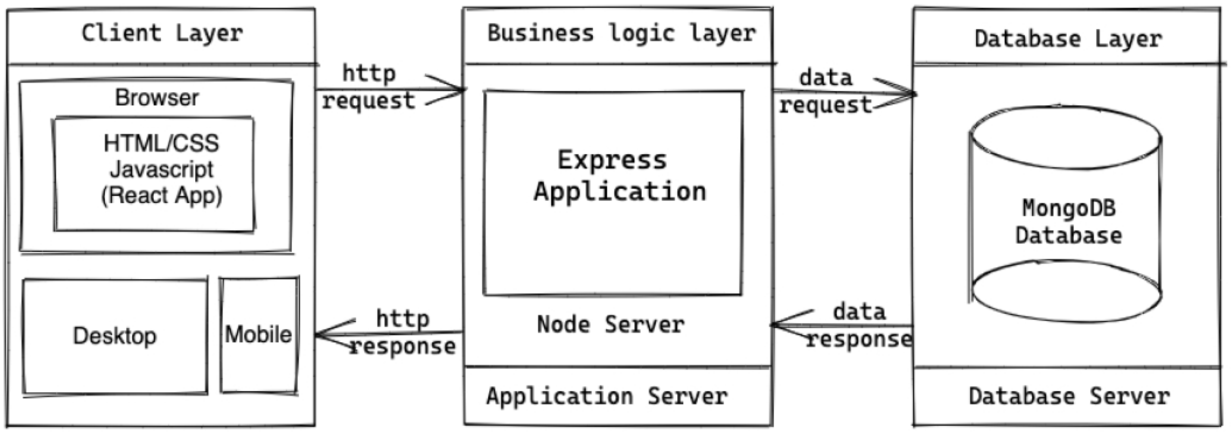


Fig 4.1 Proposed Architecture

CHAPTER 5
MODULES

5.1 USER CREATION

New Users are created in this module. The user who register can either be a Head of the department of a department or professor of some department. But only one user can be a headmaster of a department. Totally there are five departments namely Civil engineering, Mechanical engineering, Computer Science and Engineering, Electrical and Electronics Engineering and Electronics and communication Engineering.

If you try to create a second head of the department for a department which already has a HOD, it will display an error. But there can be any number of professors in a department. If you enter the appropriate credentials you will be redirected to the login for login. Or else you will stay in the same page and the error will be displayed.

5.2 USER LOGIN

In user login module the registered user can login. If the user who had not registered tried to login, it will display an error saying that there is no user with the given credentials. But if you have already registered then you will be logged in. If you have registered as a HOD, you will be logged in as a HOD. If you have registered as a professor you will be logged in as a professor.

The HOD will see the professors who registered in their department in their homepage and the Professors will see their pending and completed tasks.

5.3 TASK ASSIGNMENT

When you logged in as Head of the department of any department, you will be displayed with all the professors in your department with the option to assign task for them individually. For example, if you are a HOD of Computer Science department, you will be displayed with professors in the Computer Science department. You can choose any professor whom you want to assign task and type the task title, date of which the task should be completed and the description of the task. The fields cannot be empty.

Once you, as a HOD entered these details you can click the assign task button. Your task will now be successfully be assigned to the selected professor. Once the professor that you assigned task logged in, they will be displayed with the task you assigned in the pending section.

5.4 TASK COMPLETION

In case if you are a professor who logged in, you will be displayed with a page with two sections . They are

Pending Tasks – In this section, there will be the list of tasks which the HOD of their department assigned for them. The task includes the title of the department, date in which the task should be completed and finally the description of the task. Once they have completed the tasks, they shall click the completed button

Completed tasks - In this section, there will be the list of the completed tasks of the professor which are assigned by the HOD of their department. On seeing this, the professor can make sure that the task is still pending or already completed.

5.5 TASK RATING

When a professor of a department completes a task and clicks the Completed button, the task state will be changed as completed in the backend. Now when the HOD of the respective department logs in they will be displayed with the list of completed tasks of their department professors.

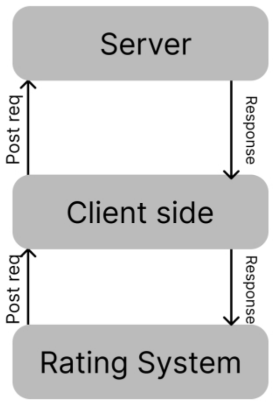


Fig.5.5.1 Task rating

For every completed task, they will be a button called ‘rate now’ which when clicked asks you to rate the task. The maximum number will be out of 5. No string is allowed. Once the rating is provided, they can click the Rate now button. Now the professor can will be able to see rating of the completed task.

CHAPTER 6

LIBRARIES AND PACKAGES

6.1 REACT JS:

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

History

React was created by Jordan Walke, a software engineer at Facebook, who released an early prototype of React called "FaxJS". He was influenced by XHP, an HTML component library for PHP. It was first deployed on Facebook's News Feed in 2011 and later on Instagram in 2012.It was open-sourced at JSConf US in May 2013

React Native, which enables native Android, iOS, and UWP development with React, was announced at Facebook's React Conf in February 2015 and open-sourced in March 2015.

On April 18, 2017, Facebook announced React Fiber, a new set of internal algorithms for rendering, as opposed to React's old rendering algorithm, Stack.React Fiber was to become the foundation of any future improvements and feature development of the React library.[needs update] The actual syntax for programming with React does not change; only the way that the syntax is executed has changed. React's old rendering system, Stack, was developed at a time when the focus of the system on dynamic change was not understood. Stack was slow to draw complex animation, for example, trying to accomplish all of it in one chunk. Fiber breaks down animation into segments that can be spread out over multiple frames. Likewise, the structure of a page can be broken into segments that may be maintained and updated separately. JavaScript functions and virtual DOM objects are called "fibers", and each can be operated and updated separately, allowing for smoother on-screen rendering.

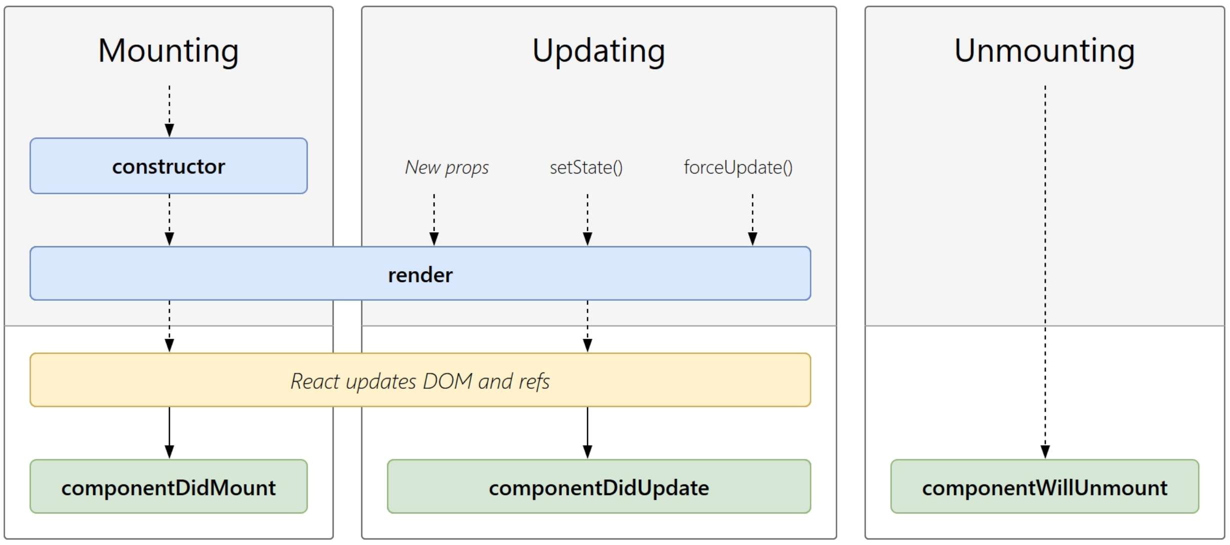


Fig.6.1.1 React lifecycle methods

6.2 EXPRESS

Node js:

Node (or more formally Node.js) is an open-source, cross-platform runtime environment that allows developers to create all kinds of server-side tools and applications in JavaScript. The runtime is intended for use outside of a browser context (i.e. running directly on a computer or server OS). As such, the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs including HTTP and file system libraries.

EXPRESS

Express is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.

- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

While Express itself is fairly minimalist, developers have created compatible middleware packages to address almost any web development problem. There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and many more. You can find a list of middleware packages maintained by the Express team at Express Middleware (along with a list of some popular 3rd party packages).

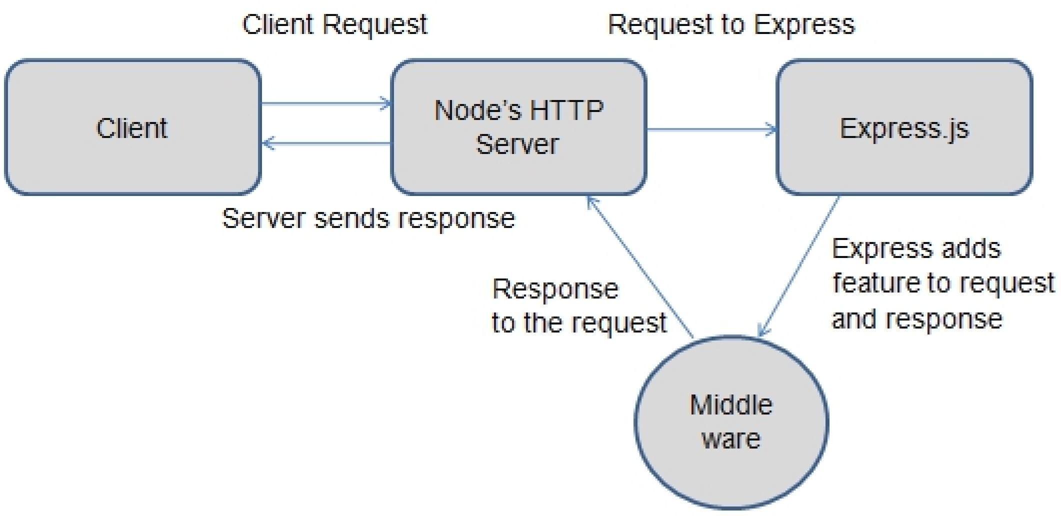


Fig.6.2.1Working of Express js

History

Node was initially released, for Linux only, in 2009. The NPM package manager was released in 2010, and native Windows support was added in 2012. Delve into Wikipedia if you want to know more. Express was initially released in November 2010 and is currently on version 4.17.3 of the API (with 5.0 in "beta"). You can check out the changelog for information about changes in the current release, and GitHub for more detailed historical release notes.

Importing and creating modules

A module is a JavaScript library/file that you can import into other code using Node's require() function. Express itself is a module, as are the middleware and database libraries that we use in our Express applications.

The code below shows how we import a module by name, using the Express framework as an example. First we invoke the require() function, specifying the name of the module as a string ('express'), and calling the returned object to create an Express application. We can then access the properties and functions of the application object.

Using asynchronous APIs

JavaScript code frequently uses asynchronous rather than synchronous APIs for operations that may take some time to complete. A synchronous API is one in which each operation must complete before the next operation can start. By contrast, an asynchronous API is one in which the API will start an operation and immediately return (before the operation is complete). Once the operation finishes, the API will use some mechanism to perform additional operations.

Using non-blocking asynchronous APIs is even more important on Node than in the browser because Node is a single-threaded event-driven execution environment. "Single threaded" means that all requests to the server are run on the same thread (rather than being spawned off into separate processes). This model is extremely efficient in terms of speed and server resources, but it does mean that if any of your functions call synchronous methods that take a long time to complete, they will block not just the current request, but every other request being handled by your web application.

Using middleware

Middleware is used extensively in Express apps, for tasks from serving static files to error handling, to compressing HTTP responses. Whereas route functions end the HTTP request-response cycle by returning some response to the HTTP client, middleware functions typically perform some operation on the request or response and then call the next function in the "stack", which might be more middleware or a route handler. The order in which middleware is called is up to the app developer.

Serving static files

You can use the express.static middleware to serve static files, including your images, CSS and JavaScript (static() is the only middleware function that is actually part of Express).

Using databases

Express apps can use any database mechanism supported by Node (Express itself doesn't define any specific additional behavior/requirements for database management). There are many options, including PostgreSQL, MySQL, Redis, SQLite, MongoDB, etc.

In order to use these you have to first install the database driver using NPM. For example, to install the driver for the popular NoSQL MongoDB you would use the command:

$ npm install mongodb

The database itself can be installed locally or on a cloud server. In your Express code you require the driver, connect to the database, and then perform create, read, update, and delete (CRUD) operations. The example below (from the Express documentation) shows how you can find "mammal" records using MongoDB.

6.3 BRCRYPT

bcrypt is a password-hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher and presented at USENIX in 1999. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.

The bcrypt function is the default password hash algorithm for OpenBSD and was the default for some Linux distributions such as SUSE Linux. There are implementations of bcrypt in C, C++, C#, Embarcadero Delphi, Elixir, Go, Java, JavaScript,Perl, PHP, Python, Ruby, and other languages.

Blowfish is notable among block ciphers for its expensive key setup phase. It starts off with subkeys in a standard state, then uses this state to perform a block encryption using part of the key, and uses the result of that encryption (which is more accurate at hashing) to replace some of the subkeys. Then it uses this modified state to encrypt another part of the key, and uses the result to replace more of the subkeys. It proceeds in this fashion, using a progressively modified state to hash the key and replace bits of state, until all subkeys have been set.

Provos and Mazières took advantage of this, and took it further. They developed a new key setup algorithm for Blowfish, dubbing the resulting cipher "Eksblowfish" ("expensive key schedule Blowfish"). The key setup begins with a modified form of the standard Blowfish key setup, in which both the salt and password are used to set all subkeys.

Description

The input to the bcrypt function is the password string (up to 72 bytes), a numeric cost, and a 16-byte (128-bit) salt value. The salt is typically a random value. The bcrypt function uses these inputs to compute a 24-byte (192-bit) hash. The final output of the bcrypt function is a string of the form:

$2<a/b/x/y>$[cost]$[22 character salt][31 character hash]

For example, with input password abc123xyz, cost 12, and a random salt, the output of bcrypt is the string

```
$2a$12$R9h/cIPz0gi.URNNX3kh2OPST9/PgBkqquzi.Ss7KIUgO2t0jWM
UW
\__/\/ \______________________/\________________________________/
Alg Cost      Salt                      Hash
```

Where:

$2a$: The hash algorithm identifier (bcrypt)

12: Input cost (212 i.e. 4096 rounds)

R9h/cIPz0gi.URNNX3kh2O: A radix-64 encoding of the input salt

PST9/PgBkqquzi.Ss7KIUgO2t0jWMUW: A radix-64 encoding of the first 23 bytes of the computed 24 byte hash

The radix-64 encoding in bcrypt uses the table ./ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,[10] which is different than RFC 4648 Base64 encoding.

Algorithm

The bcrypt algorithm is the result of encrypting the text "OrpheanBeholderScryDoubt" 64 times using Blowfish. In bcrypt the usual Blowfish key setup function is replaced with an expensive key setup (EksBlowfishSetup) function:

```
Function bcrypt
Input:
    cost:     Number (4..31)                log2(Iterations). e.g. 12 ==> 212 = 4,096 iterations
    salt:     array of Bytes (16 bytes)          random salt
    password: array of Bytes (1..72 bytes)       UTF-8 encoded password
Output:
    hash:     array of Bytes (24 bytes)
    //Initialize Blowfish state with expensive key setup algorithm
    //P: array of 18 subkeys (UInt32[18])
    //S: Four substitution boxes (S-boxes), S0...S3. Each S-box is 1,024 bytes (UInt32[256])
    P, S ← EksBlowfishSetup(cost, salt, password)
    //Repeatedly encrypt the text "OrpheanBeholderScryDoubt" 64 times
    ctext ← "OrpheanBeholderScryDoubt"   //24 bytes ==> three 64-bit blocks
    repeat (64)
    ctext ← EncryptECB(P, S, ctext) //encrypt using standard Blowfish in ECB mode
    //24-byte ctext is resulting password hash
    return Concatenate(cost, salt, ctext)
```

6.4 BODY PARSER

NPM Version NPM Downloads Build Status Test Coverage

Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

Installation

```
$ npm install body-parser
```

API

```
var bodyParser = require('body-parser')
```

The bodyParser object exposes various factories to create middlewares. All middlewares will populate the req.body property with the parsed body when the Content-Type request header matches the type option, or an empty object ({}) if there was no body to parse, the Content-Type was not matched, or an error occurred.

bodyParser.json([options])

Returns middleware that only parses json and only looks at requests where the Content-Type header matches the type option. This parser accepts any Unicode encoding of the body and supports automatic inflation of gzip and deflate encodings.

6.5 EMAIL VALIDATOR

Email validator module is a simple module to validate an e-mail address in your backend of the application

Installation via npm:

```
npm install email-validator
```

Usage

javascript

```
var validator = require("email-validator");
validator.validate("test@email.com"); // true
```

6.6 JSON WEB TOKEN

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

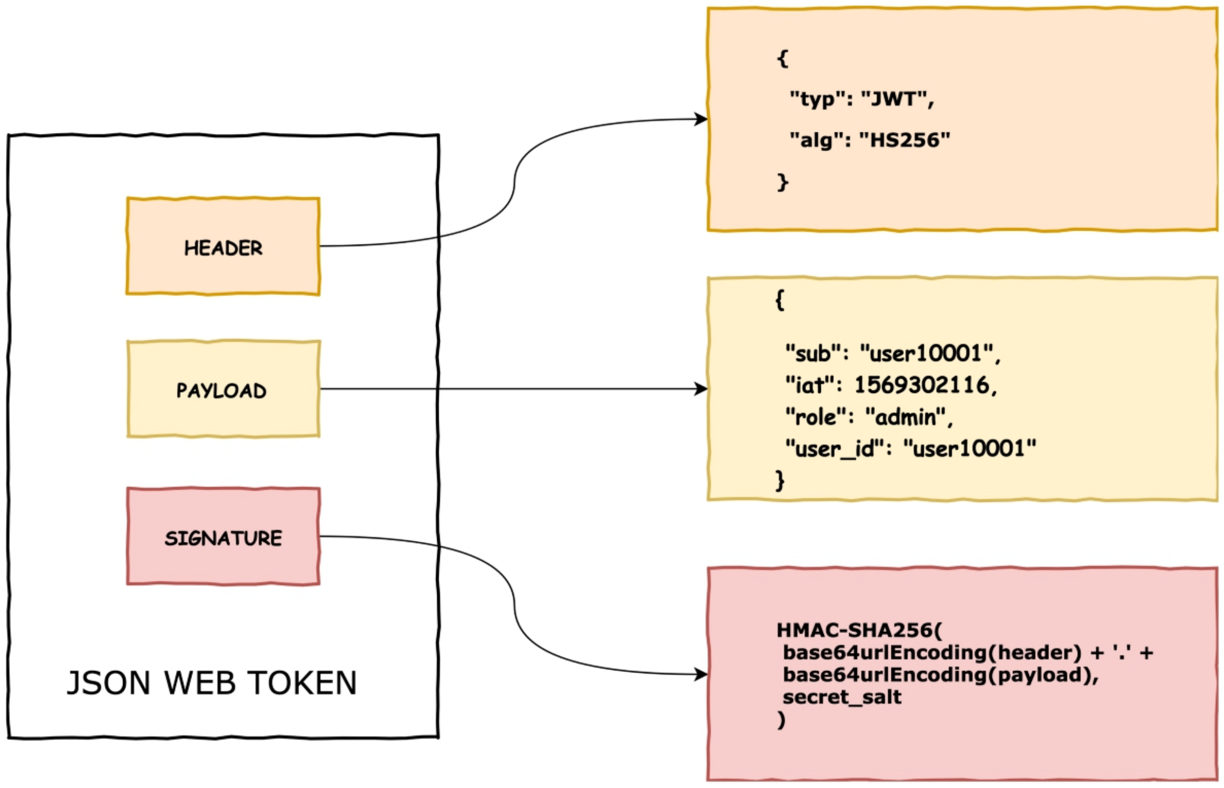

HEADER
PAYLOAD
SIGNATURE
JSON WEB TOKEN
{ "typ": "JWT", "alg": "HS256" }
{ "sub": "user10001", "iat": 1569302116, "role": "admin", "user_id": "user10001" }
HMAC-SHA256(base64urlEncoding(header) + '.' + base64urlEncoding(payload), secret_salt)

Fig.6.5.1 JSON web token structure

When should we use them?

Authorization: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single

Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

JSON Web Token structure

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

Header

Payload

Signature

Therefore, a JWT typically looks like the following.

Xxxxx.yyyyy.zzzzz

How do JSON Web Tokens work?

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required. You also should not store sensitive session data in browser storage due to lack of security.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

Authorization: Bearer <token>

This can be, in certain cases, a stateless authorization mechanism. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case.

Why JSON Web Tokens?

Let's talk about the benefits of JSON Web Tokens (JWT) when compared to Simple Web Tokens (SWT) and Security Assertion Markup Language Tokens (SAML). As JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, SWT can only be symmetrically signed by a shared secret using the HMAC algorithm. However, JWT and SAML tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Regarding usage, JWT is used at Internet scale. This highlights the ease of client-side processing of the JSON Web token on multiple platforms, especially mobile.

6.7 MONGOOSE

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks

Installation

First install Node.js and MongoDB. Then:

```
$ npm install mongoose
```

Importing

```
// Using Node.js `require()`
const mongoose = require('mongoose');

// Using ES6 imports
import mongoose from 'mongoose';
```

Connecting to MongoDB

First, we need to define a connection. If your app uses only one database, you should use mongoose.connect. If you need to create additional connections, use mongoose.createConnection.

Both connect and createConnection take a mongodb:// URI, or the parameters host, database, port, options.

await mongoose.connect('mongodb://localhost/my_database');

Once connected, the open event is fired on the Connection instance

6.8 AXIOS

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

Features

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

Installing using npm

```
$ npm install axios
```

6.9 REACT ROUTER DOM

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

The main Components of React Router are:

BrowserRouter: BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the **URL**. It is the parent component that is used to store all of the other components.

Routes: It’s a new component introduced in the v6 and a upgrade of the component. The main advantages of Routes over Switch are:

Relative s and s

Routes are chosen based on the best match instead of being traversed in order.

Route: Route is the conditionally shown component that renders some UI when its path matches the current URL.

Link: Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.

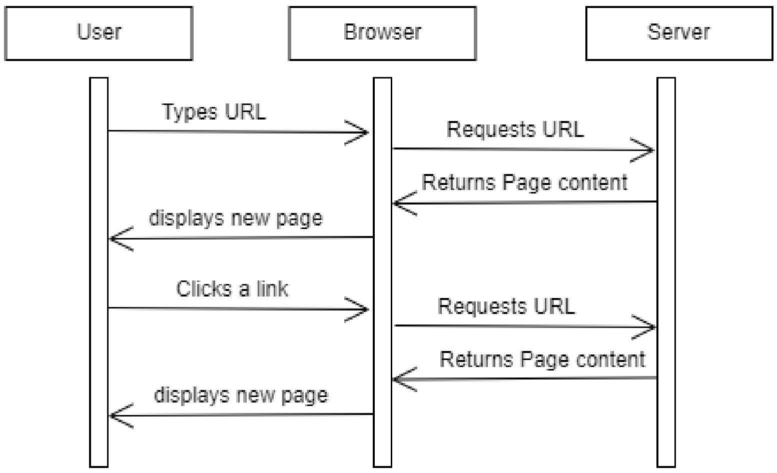


Fig.6.9.1 React Router working

Installation

You can install React Router from the public npm registry with either npm or yarn.

npm install react-router-dom

6.10 TAILWIND CSS

Tailwind CSS is basically a utility-first CSS framework for rapidly building custom user interfaces. It is a highly customizable, low-level CSS framework that gives you all of the building blocks you need to build bespoke designs without any annoying opinionated styles you have to fight to override.

The beauty of this thing called tailwind is it doesn’t impose design specification or how your site should look like, you simply bring tiny components together to construct a user interface that is unique. What Tailwind simply does is take a ‘raw’ CSS file, processes this CSS file over a configuration file, and produces an output.

Why Tailwind CSS?

- Faster UI building process
- It is a utility-first CSS framework which means we can use utility classes to build custom designs without writing CSS as in traditional approach.
- No more silly names for CSS classes and Id’s.
- Minimum lines of Code in CSS file.
- We can customize the designs to make the components.
- Makes the website responsive.
- Makes the changes in the desired manner.

Installation:

Step 1:npm init -y

Step 2:npm install tailwindcss

Step 3:Use the @tailwind directive to inject Tailwind’s base, components, and utilities styles into your CSS file.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Step 4:npx tailwindcss init

This is used to create a config file to customize the designs. It is an optional step.

Step 5:npx tailwindcss build styles.css -o output.css

This command is used to compile style.css is the file which has to be compiled and output.css is the file on which it has to be compiled.If the file output.css is not created earlier then it will automatically created.

CHAPTER 7

DATA FLOW DIAGRAM

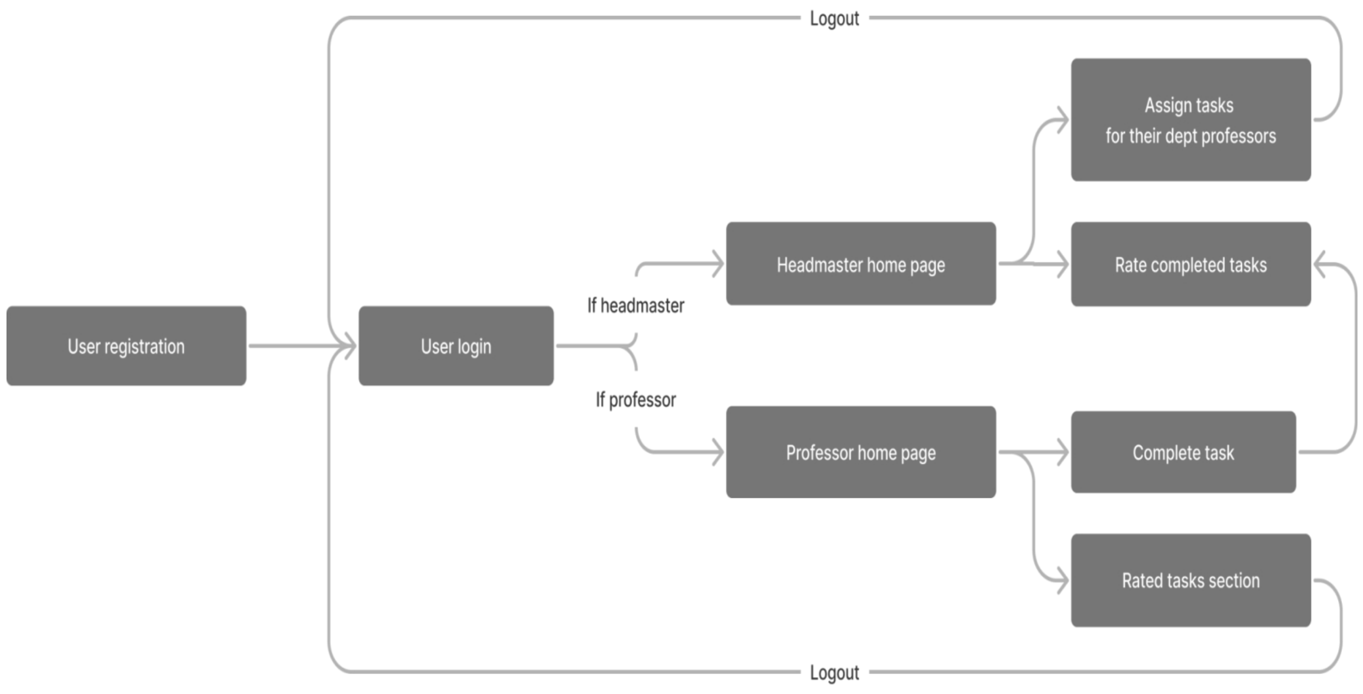


Fig.7.1 Proposed data flow

CHAPTER 8
CONCLUSION

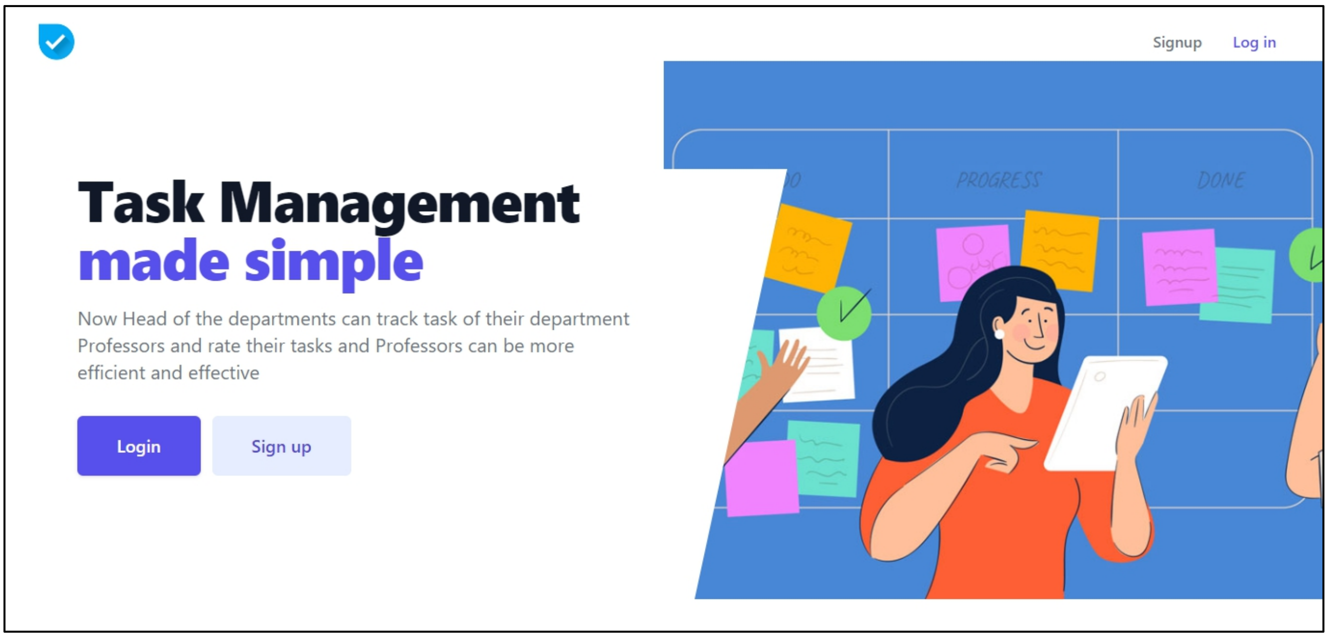
Succeeding in today’s work environment is tough. There are too many projects and tasks to manage. With new things popping up and your personal commitments, things can get overwhelming. Add the huge amount of distractions and you’re in for a difficult climb up the ladder of success.

Having a task management system can make things much easier. Whether you’re looking to achieve more of your goals or controlling your time better, a task management system will help you. You can get a positive boost to your career by becoming the person who is always on top of things and feel good every day.

CHAPTER 9

9.APPENDICES

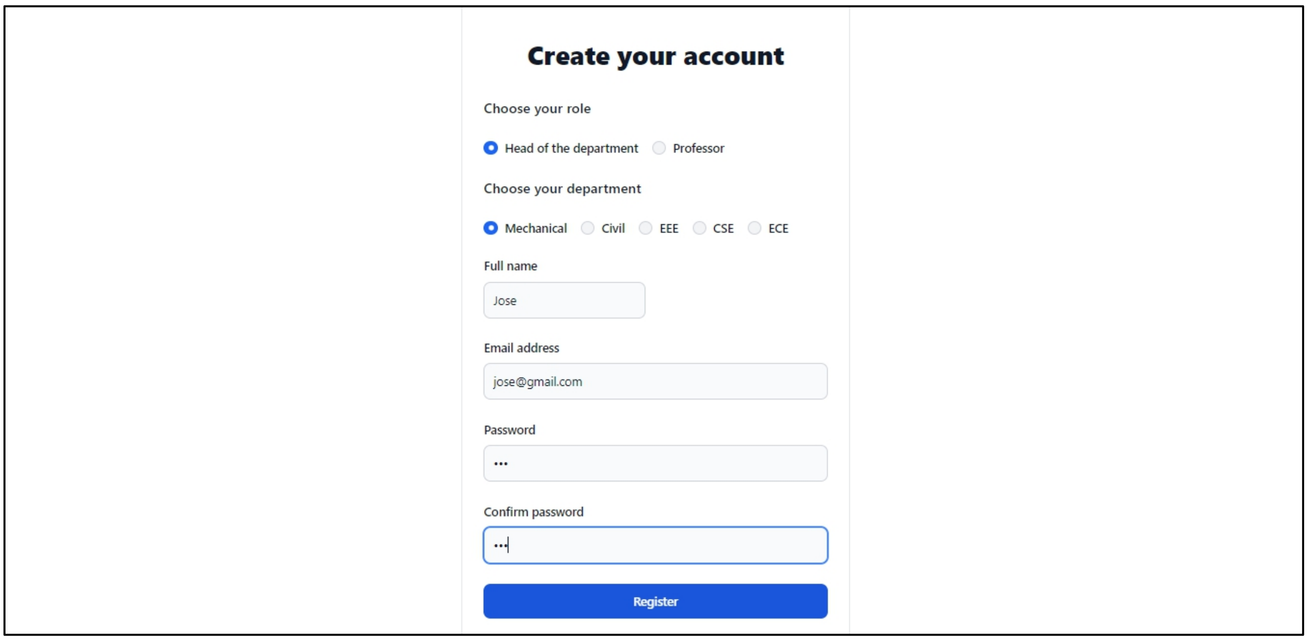
APPENDICES 1: SCREENSHOTS



Task Management made simple

Now Head of the departments can track task of their department Professors and rate their tasks and Professors can be more efficient and effective

Fig.9.1 HOMEPAGE



Create your account

Choose your role

[✓ Head of the department] Professor

Choose your department

[✓ Mechanical] Civil EEE CSE ECE

Full name: Jose

Email address: jose@gmail.com

Password: •••

Confirm password: •••

Fig 9.2 REGISTER PAGE

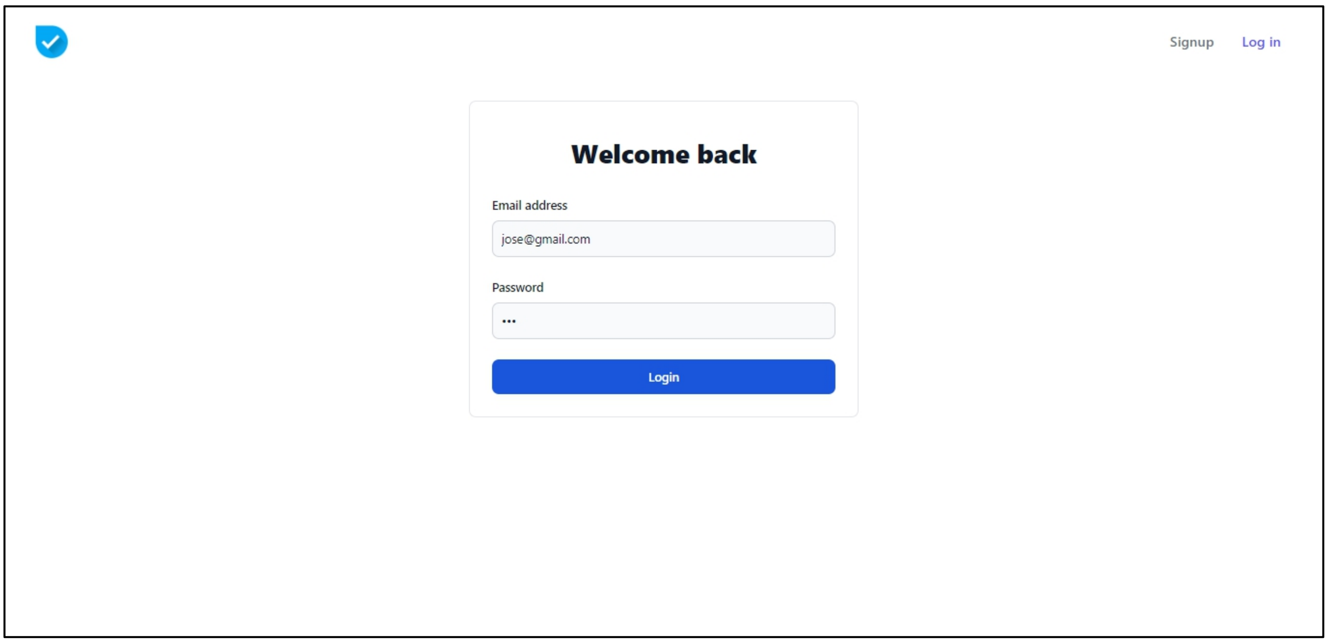
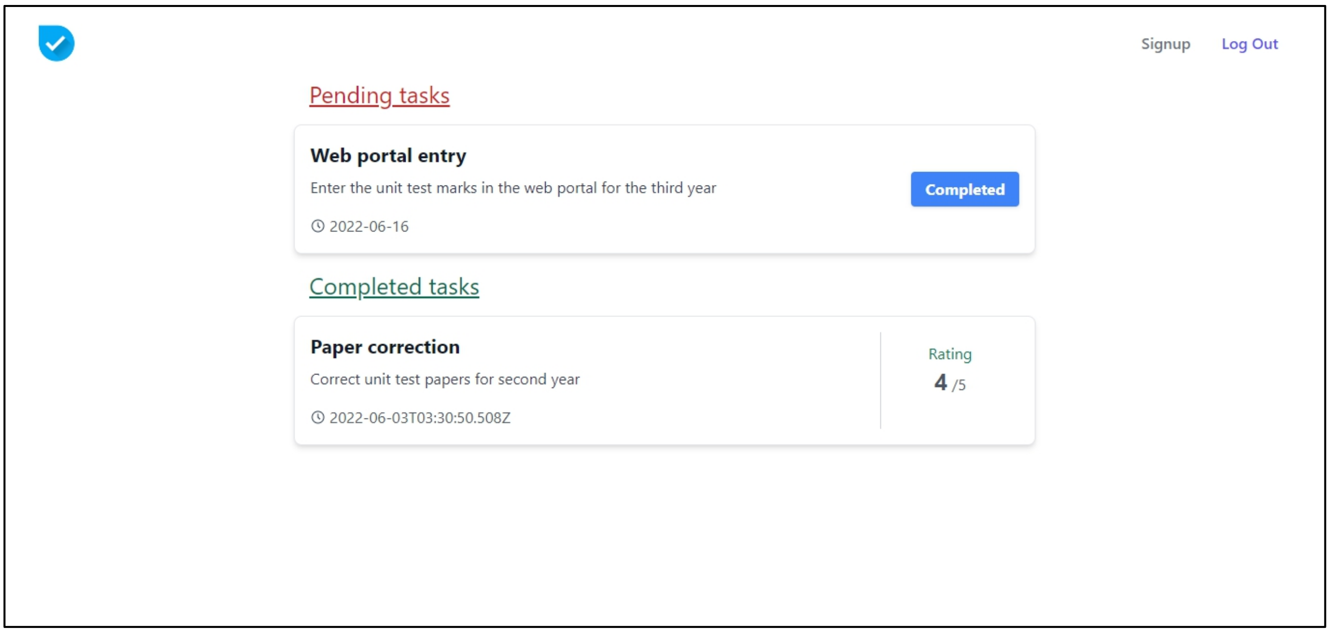


Fig.9.3 LOGIN PAGE



Pending tasks

Web portal entry

Enter the unit test marks in the web portal for the third year

2022-06-16

Completed tasks

Paper correction

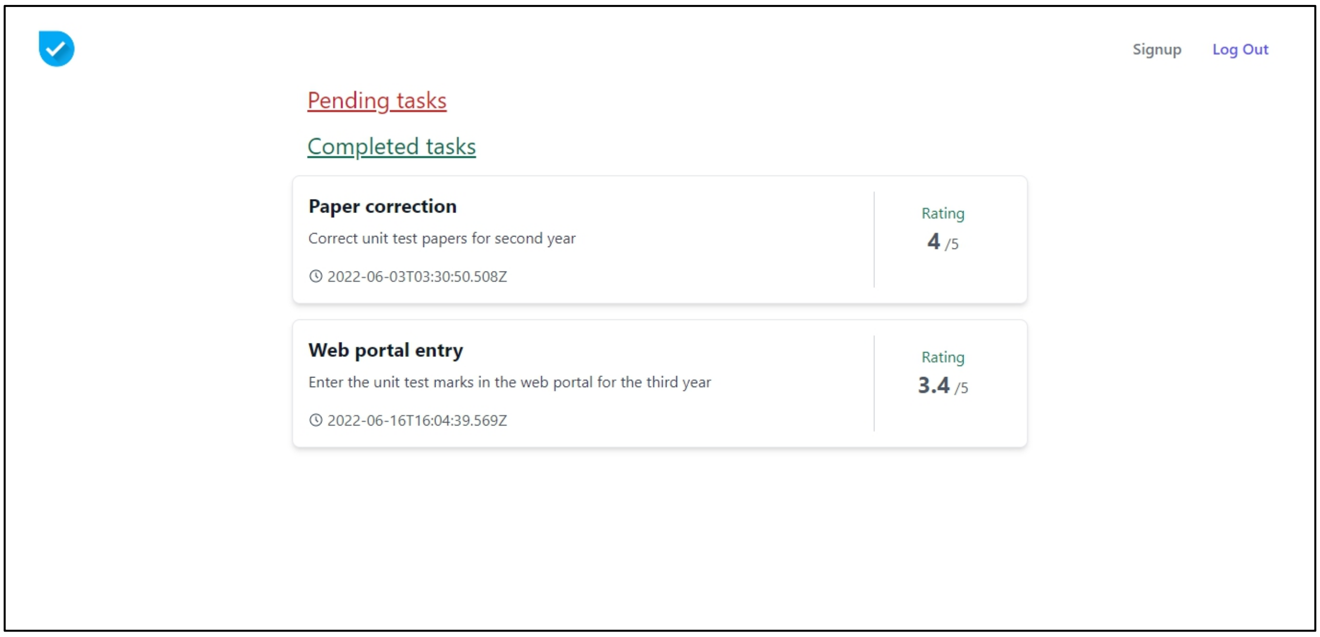
Correct unit test papers for second year

2022-06-03T03:30:50.508Z

Rating

4 /5

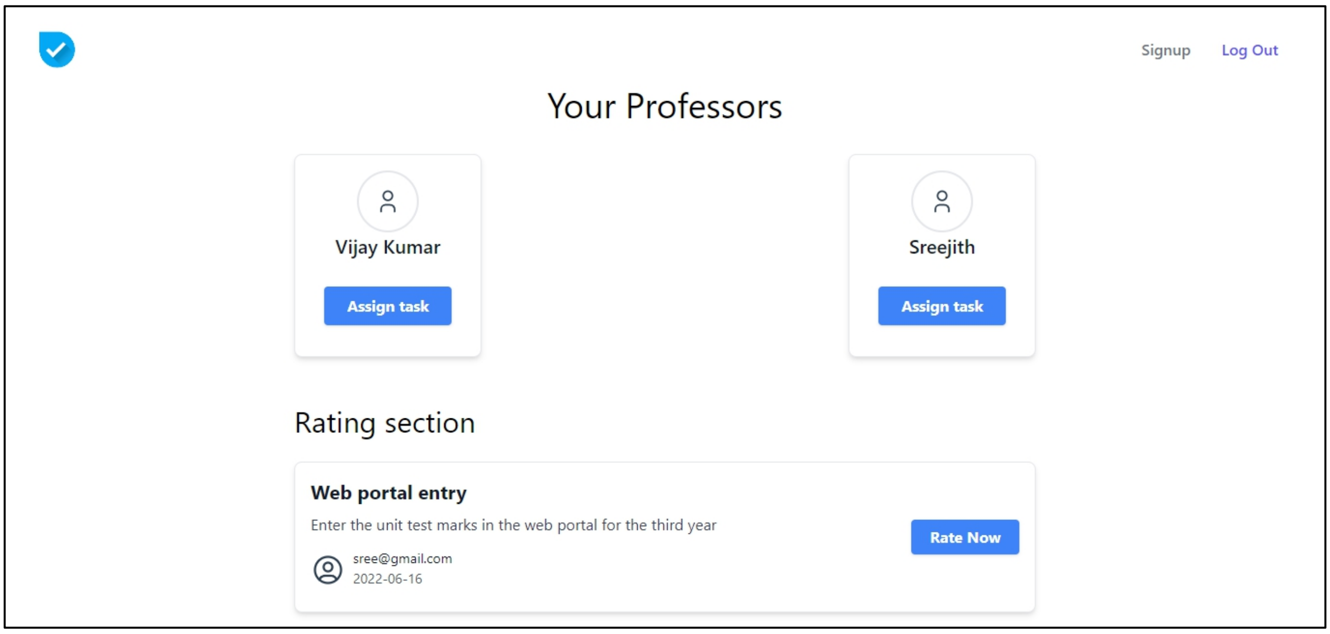
Fig 9.4 PROFESSOR PENDING TASKS PAGE



Paper correction
Correct unit test papers for second year
2022-06-03T03:30:50.508Z
Rating
4 /5

Web portal entry
Enter the unit test marks in the web portal for the third year
2022-06-16T16:04:39.569Z
Rating
3.4 /5

Fig 9.5 PROFESSOR COMPLETED TASKS



Web portal entry
Enter the unit test marks in the web portal for the third year
sree@gmail.com
2022-06-16

Fig 9.6 HOD HOME PAGE

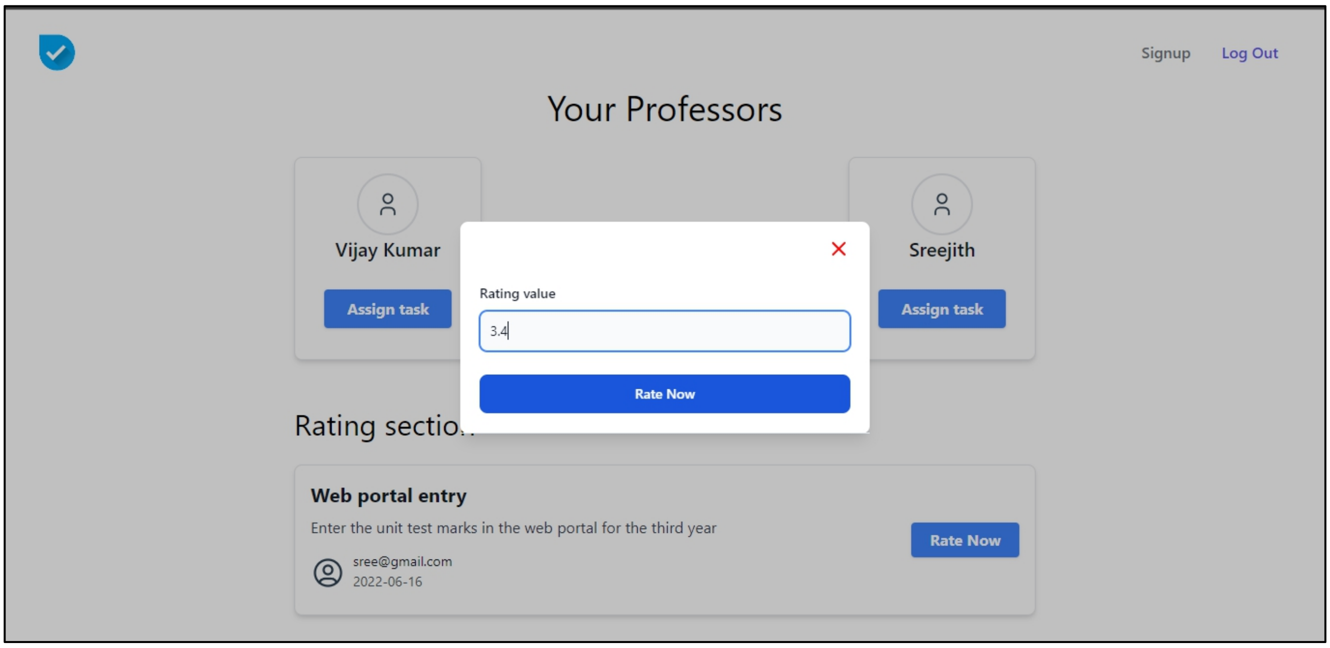

Signup
Log Out
Your Professors
Vijay Kumar
Assign task
Sreejith
Assign task
Rating value
3.4
Rate Now
Rating sectio
Web portal entry
Enter the unit test marks in the web portal for the third year
sree@gmail.com
2022-06-16
Rate Now

Fig 9.7 HOD TASK RATING

APPENDIX 2:SOURCE CODE

#Developed by
#Govarthana K - 830119104013
#Mohamed Farhan S - 830119104023
#Sivasankaranarayanan D - 830119104045

Frontend

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/taskdo-logo2.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>TaskDo</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>
  <script src="https://unpkg.com/flowbite@1.4.6/dist/datepicker.js"></script>
</html>
```

Index.js

```
import React from "react";
import { BrowserRouter } from "react-router-dom";
import "flowbite";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
const root = ReactDOM.createRoot(document.getElementById("root"));
```

```
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
reportWebVitals();
```

App.js

```
import "./App.css";
import Hero from "./components/Hero";
import { Routes, Route} from "react-roter-dom";
import Login from "../src/components/Login";
import Navbar from "./components/Navbar";
import Register from "./components/Register";
import HeadHome from "./components/HeadHome";
import ProfHome from "./components/ProfHome";
function App() {
  return (
    <div>
      <Navbar />
      <Routes>
        {!localStorage.getItem("AccessToken") && (
          <Route path="/" element={<Hero />} />
        )}
        <Route path="/login" element={<Login />} />
        {localStorage.getItem("userType") == "Hod" && (
          <Route path="/" element={<HeadHome />} />
        )}
        <Route path="hod" element={<HeadHome />} />
        <Route path="prof" element={<ProfHome />} />
        {localStorage.getItem("userType") == "Prof" && (
          <Route path="/" element={<ProfHome />} />
        )}
        <Route path="register" element={<Register />} />
      </Routes>
    </div>
  );
}
export default App;
```

Backend

Server.js

```
const express = require("express");
const app = express();
const mongoose = require("mongoose");
const body_parser = require("body-parser");
const cors = require("cors");
app.use(cors());
app.get("/test", (req, res) => {
  res.send("testing");
});
const port = process.env.PORT || 5000;
const SignupRouter = require("./src/signup/signup.route");
const LoginRouter = require("./src/login/login.router");
const { authenticateToken } = require("./src/login/login.service");
const TaskRouter = require("./src/tasks/task.router");
app.use(body_parser.json());
app.use(body_parser.urlencoded({ extended: false }));
mongoose
  .connect(    "mongodb+srv://farhan:KS2S1tm1zIy3Z7Xs@cluster0.7bmbg.mon
godb.net/SafaDB?retryWrites=true&w=majority" )
  .then(() => console.log("Db is connected"))
  .catch((err) => console.log(err, "it has an error"));
app.use("/register", SignupRouter);
app.use("/login", LoginRouter);
app.use("/tasks", authenticateToken, TaskRouter);
app.listen(port, () => [
  console.log(`server starts at http://localhost:${port}`),
]);
```

Login.router.js

```
const express = require("express");
const LoginRouter = express.Router();
const { login } = require("./login.service");
LoginRouter.post("/", login);
module.exports = LoginRouter;
```

Login.service.js

```
const jwt = require("jsonwebtoken");
const User = require("../models/users.models");
const bcrypt = require("bcrypt");
const generateToken = (email, password) => {
  return jwt.sign(
    {                    email,                    password                    },
"LKD394dsjflkasjdfNqnadlfAALKJDFCMAIERULAKDFLAIJFANDFAKNF
EALJA",
    {
      expiresIn: "604800s",
    }
  );
};
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  if (!authHeader) res.send("authentication failed", 404);
  const token = authHeader.split(" ")[1];
  jwt.verify(
    token,
"LKD394dsjflkasjdfNqnadlfAALKJDFCMAIERULAKDFLAIJFANDFAKNF
EALJA",
    (err, user) => {
      if (err) res.send("Invalid user", 404);
      else {
        req.user = user;
        next();
      }
    }
  );
};
const login = async (req, res) => {
  const email = req.body.email;
  const password = req.body.password;
  const user = await User.findOne({ email });
  if (!user) res.send("No user found with this email", 404);
  else if (!(await bcrypt.compare(password, user.password))) {
    res.send("Password does not match with the email", 404);
  } else {
    const accessToken = generateToken(email, password);
    res.json({ AccessToken: accessToken });
  }
};
module.exports = { login, authenticateToken };
```

Signup.route.js

```
const express = require("express");
const signupRouter = express.Router();
const { signup } = require("./signup.service");
```

```
signupRouter.post("/", signup);
module.exports = signupRouter;
```

Signup.service.js

```
const email_validator = require("email-validator");
const bcrypt = require("bcrypt");
const User = require("../models/users.models");
const signup = async (req, res) => {
  const fullname = req.body.fullname;
  const email = req.body.email;
  const department = req.body.department;
  const role = req.body.role;
  const password = req.body.password;
  const newUser = User({
    fullname,
    email: email_validator.validate(email)
      ? email
      : res.send("Invalid Email", 404),
    department,
    role,
    password: await bcrypt.hash(password, 10),
  });
  try {
    if (newUser.role === "Head of the department") {
      User.findOne({ role, department }, function (err, user) {
        if (err) res.send("something went wrong", 404);
        if (user) {
          res.send(
            `HOD position for ${department} has already been occupied`,
            404
          );
        } else {
          newUser
            .save()
            .then((user) => res.send("new hod has been appointed"))
            .catch((err) => {
              if (err.toString().includes("email")) {
                res.send("Email is taken", 404);
              }
            });
        }
      });
    } else {
      await newUser.save();
      res.send("user has been saved successfully");
    }
  } catch (err) {
    if (err.toString().includes("email")) {
      res.send("Email is taken", 404);
    }
  }
};
module.exports = { signup };
```

Task.route.js

```
const express = require("express");
const TaskRouter = express.Router();
const {
  saveTask,
  isHod,
  myTask,
  allProf,
  submitMyTask,
  departmentTask,
  rateMyTask,
} = require("./task.service");
TaskRouter.post("/saveTask", saveTask);
TaskRouter.get("/isHod", isHod);
TaskRouter.get("/myTask", myTask);
TaskRouter.get("/allProf", allProf);
TaskRouter.get("/submitTask/:id", submitMyTask);
TaskRouter.get("/departmentTask", departmentTask);
TaskRouter.put("/rateMyTask/:id", rateMyTask);
module.exports = TaskRouter;
```

Task.service.js

```
const Task = require("../models/tasks.models");
const User = require("../models/users.models");
// checking the login user
const isHod = async (req, res) => {
  const user = await User.findOne({ email: req.user.email });
  console.log(user);
  if (user.role === "Head of the department") {
    res.send("Hod Login");
  } else {
    res.send("Professor Login");
  }
};

const saveTask = async (req, res) => {
  const assignedUser = req.body.assignedUser;
  const title = req.body.title;
  const description = req.body.description;
  const assignedDate = req.body.assignedDate;
  const expectedDate = req.body.expectedDate;
  const submittedDate = req.body.submittedDate;

  const newTask = Task({
    assignedUser,
    title,
    description,
    assignedDate,
    expectedDate,
    submittedDate,
  });

  try {
    await newTask.save();
    res.send("task is saved")
  } catch (err) {
    console.log(err, "this has error");
  }
};

const myTask = async (req, res) => {
  const myTasks = await Task.find({ assignedUser: req.user.email });
  res.send(myTasks);
};
const allProf = async (req, res) => {
  const hod = await User.findOne({
    email: req.user.email,
  });

  console.log(hod.department, "hod");
  const allProf = await User.find({
    department: hod.department,
  });
  console.log(allProf);
  res.send(allProf);

};

const submitMyTask = async (req, res) => {
  const id = req.params.id;
  const completeTask = await Task.findByIdAndUpdate(id, { isCompleted:
true });
```

```
  res.send(completeTask);
};

const rateMyTask = async (req, res) => {
  const id = req.params.id;
  const rate = req.body.rate;
  const rateTask = await Task.findByIdAndUpdate(id, { rating: rate });
  res.send(rateTask);

};

const departmentTask = async (req, res) => {
  const hod = await User.findOne({ email: req.user.email });
  const profTask = await Task.find({ department: hod.department });
  res.send(profTask);

};

const rating = (module.exports = {
  saveTask,
  isHod,
  myTask,
  allProf,
  submitMyTask,
  rateMyTask,
  departmentTask,
});
```

Models

Tasks.models.js

```
const mongoose = require("mongoose");
const TaskSchema = mongoose.Schema({
  assignedUser: {
    type: String,
    required: true,
  },
  title: {
    type: String,
    requried: true,
  },
  description: {
    type: String,
    required: true,
  },
  assignedDate: {
    type: Date,
    default: Date.now,
  },
  expectedDate: {
    type: Date,
    required: true,
  },
  isCompleted: {
    type: Boolean,
    default: false,
  },
  rating: {
    type: Number,
    default: 0,
  },
});

module.exports = Task = mongoose.model("task", TaskSchema);
```

User.models.js

```
const mongoose = require("mongoose");
const UsersSchema = mongoose.Schema({

  fullname: {
    type: String,
    required: true,
  },

  email: {
    type: String,
    unique: true,
    requied: true,
  },

  department:{
    type: String,
    required: true,
  },

  role:{
    type: String,
    required: true,
  },

  password: {
    type: String,
    requied: true,
  },

  date: {
    type: Date,
    default: Date.now,
  },
});

module.exports =  User = mongoose.model("User", UsersSchema);
```

REFERENCES

LIST OF REFERENCES

[1] Adair, J., & Allen, M. (2008). The concise: Time Management and Personal Development. New Delhi: VivaBooks.

[2] Bagher, R. C, Hassanpour, H., and Mashayekhi, H. "User Trends Modeling for a Content-based Recommender System." NeuroImage. June 13, 2017. Accessed December 03, 2018)

[3] Carol E Smith, "Organizing & Time Management Statistics." Simply Productive. April 23,2014. Accessed December 03, 2018.

[4] David Mandell, "To-do List, Reminders, Errands - App of the Year!" Wunderlist. Accessed December 03, 2018.

[5] Etherington, Darrell. "Google Acquires Timeful To Bring Smart Scheduling To Google Apps." Accessed December 03, 2018.

[6] Jo-Ana Chase "Todoist – The Best To Do List App & Task Manager." Todoist. Accessed December 03, 2018

[8] Martinez, Ana Belen Barragans, Jose J. Pazos Arias, Ana Fernandez Vilas, Jorge Garcia Duque, and Martin Lopez Nores. "What's on TV Tonight? An Efficient and Effective Personalized Recommender System of TV Programs." An Introduction to Biometric Recognition - IEEE Journals & Magazine. April 17, 2009. Accessed December 03, 2018.

[9] M. Jalali, N. Mustapha, M. Sulaiman, A. "Mamay
WEBPUM: a web-based recommendation system to predict user future movement ", September 2010.

[10]Robert Topp, "Time Management Statistics." Key Organization Systems. Accessed December 3, 2018.

[11] Wires, RP News. "College Professors Struggle with Organizational Skills." Reliable Plant. November 15, 2006. Accessed December 03, 2018.