**COMPLAINT MANAGEMENT SYSTEM**

Submitted by

### KISHORE P

### 1P21CS011

In partial fulfillment of the requirements for the award of the Degree of

### MASTER OF SCIENCE IN COMPUTER SCIENCE

from Bharathiar University, Coimbatore.

Under the Internal Supervision of

### Dr. P. V. RAVINDRANATH M.Sc., Ph.D

****

**SCHOOL OF COMPUTER STUDIES (PG)**

**RATHNAVEL SUBRAMANIAM COLLEGE OF ARTS AND SCIENCE (AUTONOMOUS)**

**Sulur, Coimbatore – 641 402.**

**April 2023.**

**RATHNAVEL SUBRAMANIAM COLLEGE OF ARTS AND SCIENCE (AUTONOMOUS)**

**Sulur, Coimbatore – 641 402.**

**School of Computer Studies (PG)**

****

**Register Number: 1P21CS011**

Certified bona fide original record work done by **KISHORE P**

**Guide HOD**

Submitted for the project Evaluation and Viva voce held on\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Internal Examiner External Examiner

# CERTIFICATE

This is to certify that the project work entitled **COMPLAINT MANAGEMENT SYSTEM**, submitted to the School of Computer Studies(PG), Rathnavel Subramaniam College of Arts and Science in partial fulfillment of the requirements for the award of the Degree of Master of Computer Science is a record of original project work done by **KISHORE P** during the period January 2023- April 2023 of his study in the **Master of Computer Science, Rathnavel Subramaniam College of Arts and Science**, under my internal supervision and the project work has not formed the basis for the award of any Degree/Diploma/Associateship/Fellowship or other similar title to any candidate of any University.

Internal Supervisor

# DECLARATION

I, **KISHORE** hereby declare that the project entitled **Complaint Management System** submitted to the School of Computer Studies (PG), Rathnavel Subramaniam College of Arts and Science, in partial fulfillment of the requirements for the award of the Degree of Master of Science in Computer Science is a record of original project work done by me during the period Jan 2023 to April 2023 under the internal supervision of **Dr. P. V. RAVINDRANATH M.Sc., Ph.D Assistant Professor, Rathnavel Subramaniam College** **Of Arts and Science (Autonomous)** From Bharathiar University, Coimbatore.

Signature of the Candidate

# ACKNOWLEDGEMENTS

I express my sincere thanks to our Managing Trustee **Dr. K. Senthil Ganesh MBA (USA)., MS (UK)., Ph.D.,** for providing us with adequate faculty and laboratory resources for completing my project successfully.

I take this as a fine opportunity to express my sincere thanks to **Dr. T. Sivakumar M.Sc., M. Phil., Ph.D., Principal,** Rathnavel Subramaniam College of Arts and Science (Autonomous) for giving me the opportunity to undertake this project.

I express my sincere thanks to **Dr. P. Navaneetham M.Sc., M.Phil., Ph.D., Director (Administration), School of Computer Studies** for the help and advice throughout the project.

I express my sincere thanks to **Dr. S. Yamini M.Sc., (CC)., M. Phil., Ph.D., Director (Academic), School of Computer Studies** for her valuable guidance and prompt correspondence throughout the curriculum to complete the project.

I express my sincere thanks to **Dr. D. Maheswari, M.Sc.CS., M.Phil., Ph.D., Head and Research Coordinator, School of Computer Studies(PG)** for her support and advice throughout the project.

I express my gratitude to **Dr. P. V. RAVINDRANATH M.Sc., Ph.D Assistant Professor, School of Computer Studies(PG)** for his valuable guidance, support, encouragement, and motivation rendered by her throughout this project.

Finally, I express my sincere thanks to all other staff members and my dear friends, dear and near for helping me to complete this project.

### KISHORE P

# ABSTRACT

The College Complaint Management System is a software application designed to facilitate the efficient management of student complaints in a college or university. The system provides a platform for students to report grievances, which are then automatically routed to the relevant college department or personnel for resolution.

The system allows students to submit complaints online, and also provides a dashboard for college staff to manage complaints, assign them to the appropriate department or personnel, track the status of complaints, and generate reports. The system ensures that complaints are resolved in a timely and transparent manner, and that students are kept informed throughout the process.

The College Complaint Management System improves the college's complaint management process by providing a centralized platform for the management of complaints, reducing the workload of college staff, and improving the quality of feedback to students. The system also provides valuable data to college administrators, allowing them to identify trends and areas for improvement.

The system is designed to be user-friendly, with a simple interface that is easy to navigate for both students and college staff.

Students can submit complaints anonymously if they prefer, and the system ensures that their identity is protected throughout the process.

The system allows for the categorization of complaints by type, allowing college administrators to identify common issues and prioritize areas for improvement.

The system can be customized to meet the specific needs of individual colleges, with the ability to add or remove features as necessary.

The system provides a record of all complaints and their resolution, which can be used for future reference and analysis.

In summary, the College Complaint Management System is a software application designed to streamline the management of student complaints in a college or university. It provides a platform for students to report grievances, and for college staff to manage and resolve complaints in a timely and transparent manner.

# CONTENTS

Certificate 3

Declaration 4

Acknowledgements 5

Abstract 6

**CHAPTER 1**

1. Introduction 9

1.1 An overview of project 9

1.2 Mission of project 10

1.3 Background study 11

1.3.1 A study of the existing system 11

**CHAPTER 2**

2. System Analysis 12

2.1 A study of the proposed system 12

2.2 User requirement specification 12

2.2.1 Major modules 12

2.2.2 Submodules 13

2.3 Software requirement specification 13

2.4 System specification 13

2.4.1 React.js 13

2.4.2 Node.js 14

2.4.3 Hardware configuration 14

2.4.4 Software configuration 14

**CHAPTER 3**

3. System design and development 15

3.1 Fundamentals of design concepts 15

3.1.1 Abstraction 15

3.1.2 Refinement 15

3.1.3 Modularity 15

3.2 Design Notations 15

3.2.1 System structure chart 16

3.2.2 System flow diagram 16

3.2.3 Data flow diagram / UML 17

3.3 Design process 19

3.3.1 Software Architecture 19

3.3.2 Control Hierarchy 19

3.3.3 Structural Partitioning 19

3.3.4 Data structure 19

3.3.5 Software procedure 19

3.4 Table Design 20

3.5 Input design 21

3.6 Output design 21

3.7 Development approach 22

**CHAPTER 4**

4. Testing and implementation 24

4.1 Testing 24

4.1.1 System testing 50

4.1.2 White box testing 50

4.1.3 Unit testing 51

4.2 System implementation 51

4.2.1 System Maintenance 52

**CHAPTER 5**

5. Conclusion

5.1 Directions for future enhancements references 25

BIBLIOGRAPHY 26

**ANNEXURE**

ANNEXURE - A - Output design 27

ANNEXURE - B - Source code 30

**CHAPTER 1**

**1.Introduction**

**1.1 An overview of project:**

A student complaint management system is a software application designed to manage and streamline the process of handling complaints and feedback from students.

This system can be used by educational institutions, universities, colleges, and other organizations that deal with student grievances.

The system typically includes features such as online complaint submission, automated tracking and processing of complaints, escalation and resolution workflows, and reporting and analytics capabilities. The system can also facilitate communication between students, faculty, staff, and administrators, ensuring transparency and accountability in the complaint handling process.

The complaint management system can be accessed through a web interface or a mobile application, allowing students to submit complaints from anywhere and at any time.

The system can also provide automated notifications and updates to students on the status of their complaints, reducing the need for manual follow-up and improving student satisfaction.

Overall, a student complaint management system can help educational institutions improve their responsiveness and effectiveness in handling student complaints and feedback, ultimately leading to better student outcomes and satisfaction.

2. System Analysis **1.2 Mission of project:**

The mission of the College Complaint Management System project is to provide a platform for effective management of complaints filed by students, faculty, and staff in a college or university setting.

The system aims to streamline the complaint handling process by providing a user-friendly interface for submitting and tracking complaints, assigning complaints to the appropriate department or individual for review and resolution, and providing notifications and updates to users on the status of their complaints.

The project's mission also includes providing administrators with reporting and analytics features to help them identify trends and areas for improvement in the college's operations. By analyzing data on the types of complaints received, the departments involved, and the time it takes to resolve complaints, administrators can make data-driven decisions to improve the college's processes and ensure a better experience for all stakeholders.

* To create a user-friendly interface that allows students to file complaints easily and efficiently, without the need for extensive technical knowledge or support.
* To establish a transparent and accountable process for addressing student complaints, where the progress and status of complaints can be tracked and monitored by all relevant parties.
* To promote fairness and impartiality in the complaint management process, where all complaints are addressed based on merit and without bias.
* To ensure that all complaints are addressed in a timely and efficient manner, to minimize the impact of complaints on student learning and well-being.
* To encourage feedback from students and other stakeholders to continually improve the complaint management system, making it more effective and responsive to the needs of the college community.

Overall, the mission of the College Complaint Management System project is to promote transparency, efficiency, and accountability in the management of complaints in a college or university setting, ultimately improving the overall experience for students, faculty, and staff.

**1.3 Background study:**

**1.3.1 A study of the existing system:**

The existing student complaint management systems vary in their scope, complexity, and effectiveness. Some institutions use manual processes, such as paper-based forms or email communication, which can be time-consuming and error-prone.

Other institutions have implemented digital systems that provide online complaint submission and tracking, but these systems may lack features such as automated workflows, reporting and analytics, and integration with other systems. Additionally, some existing systems may not effectively address the needs and concerns of students, leading to dissatisfaction and low participation rates. Therefore, there is a need for a comprehensive and user-friendly student complaint management system that can improve the complaint handling process and enhance student satisfaction.

**2. System Analysis**

**2.1 A Study of the Proposed System:**

* Overview of the current complaint management process in the college/university
* Identification of pain points and inefficiencies in the current process
* Analysis of potential solutions and technologies to address the identified issues
* Evaluation of the feasibility and impact of the proposed system on college operations

**2.2 User Requirement Specification:**

* Identification of stakeholders (students, staff, administrators) and their respective roles in the system
* Development of use cases and user stories to capture user requirements and workflows
* Definition of functional requirements, such as the ability to file complaints, track complaints, and communicate with staff
* Definition of non-functional requirements, such as security, usability, and scalability

**2.2.1 Major Modules:**

Student Complaint Filing Module: allows students to file complaints online, including information such as the nature of the complaint, date, time, location, and any relevant evidence.

Complaint Tracking Module: enables college staff to track the progress of complaints and manage the resolution process, including assigning complaints to staff members, updating complaint status, and communicating with students.

Complaint Analytics Module: provides reports and data analysis to help administrators identify trends and improve the complaint management process.

**2.2.2 Submodules:**

User Authentication Module: ensures secure access to the system and differentiates between student, staff, and administrator users.

Notification Module: sends email or SMS notifications to students and staff regarding the status of complaints and any updates to the complaint resolution process.

Search and Filter Module: allows users to search and filter complaints based on various parameters such as date, type of complaint, location, and status.

Feedback Module: enables students to provide feedback on the complaint management process and their satisfaction with the outcome of their complaints.

**2.3 Software Requirement Specification:**

This section would outline the software requirements for the system, including the operating system, database management system, web server, and programming languages or frameworks. It would also specify any third-party software or libraries that are required for the system to function properly.

**2.4 System Specification:**

This section would provide detailed specifications for the hardware and software components of the system.

**2.4.1 React.js:**

Single-Page Applications: ReactJS is often used to build single-page applications where the user interacts with a single page without the need to refresh the page. ReactJS provides a smooth and fast user experience by dynamically updating the content of the page without requiring a full page reload.

User Interfaces: ReactJS can be used to build complex user interfaces with reusable components. ReactJS provides a declarative approach to building user interfaces, allowing developers to easily manage the state of the application and update the user interface as needed.

Mobile Applications: React Native is a framework based on ReactJS that allows developers to build mobile applications for iOS and Android platforms using a single codebase. React Native allows developers to build high-performance mobile applications with a native look and feel.

Real-Time Applications: ReactJS can be used to build real-time applications such as chat applications or collaborative applications. ReactJS provides a simple way to manage the state of the application and update the UI in real-time.

**2.4.2 Node.js:**

Web Application Development: Node.js can be used to build scalable and high-performance web applications, including e-commerce sites, social media platforms, and content management systems.

API Development: Node.js can be used to build RESTful APIs for web and mobile applications. Node.js provides a lightweight and scalable server-side environment that can handle a large number of requests simultaneously, making it efficient for building APIs.

Command-Line Tools: Node.js can be used to build command-line tools such as build tools, testing frameworks, and deployment scripts. Node.js provides a lightweight and efficient runtime environment, making it ideal for building command-line tools.

**2.4.3 Hardware configuration**

* Processor : Lenovp idea pad gaming 3 gen 11
* RAM : 8GB
* Hard Disk Drive : 512gb
* Operating System : Windows 11 Home

**2.4.4 Software configuration**

* Operating System : Windows 11 Home
* Front End : React Js, Handlebars
* Backend : NodeJS
* Database : MongoDB (version 6.0)

**CHAPTER 3**

**3. SYSTEM DESIGN AND DEVELOPMENT**

**3.1 Fundamentals of design concepts**

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved.

**3.1.1 Abstraction**

Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.

**3.1.2 Refinement**

It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.

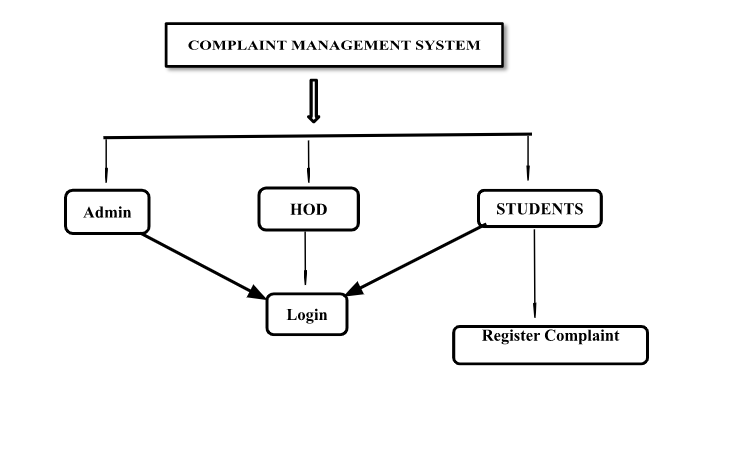
**3.1.3 Modularity**

Software architecture is divided into components called modules.

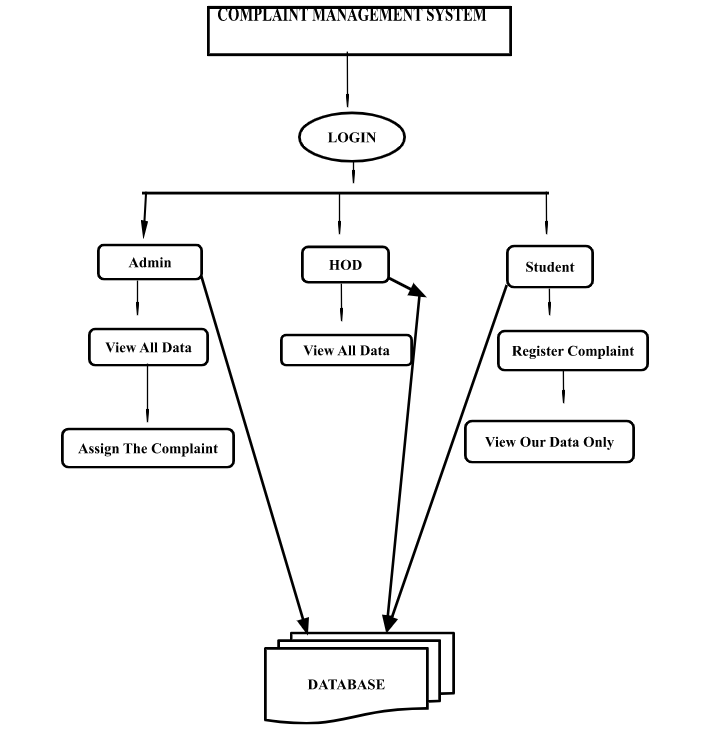
**3.2 Design Notations**

Design notations are used when planning and should be able to communicate the purpose of a program without the need for formal code.

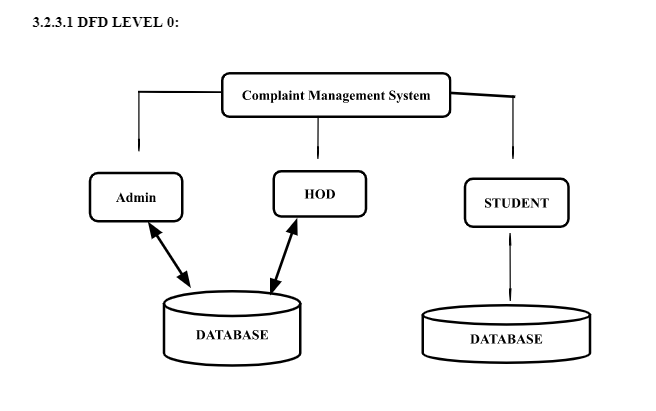
* + 1. **System structure chart:**

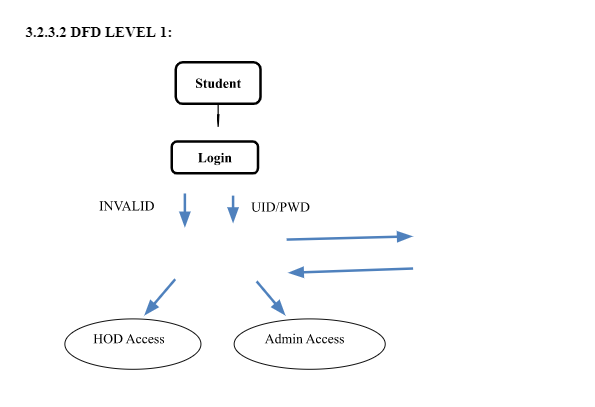
****

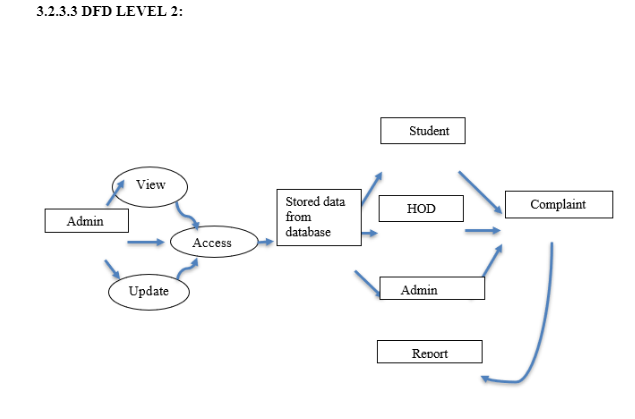
* + 1. **System flow diagram:**



**3.2.3 Data flow diagram:**

****

****

****

**3.3 Design Process:**

**3.3.1 Software Architecture:**

Software architecture refers to the overall structure of the system, including the various components and how they interact with each other. In a complaint management system, the software architecture would include components such as user interface, complaint submission and management, database management, and reporting.

The architecture must be designed to ensure that the system is scalable, reliable, and maintainable.

**3.3.2 Control Hierarchy:**

Control hierarchy refers to the hierarchical arrangement of modules and their dependencies in the system. In a complaint management system, the control hierarchy would include the different levels of access and permissions for users, the flow of complaints from submission to resolution, and the escalation process for unresolved complaints.

**3.3.3 Structural Partitioning:**

Structural partitioning refers to the process of breaking down the system into smaller, manageable parts. In a complaint management system, structural partitioning could involve breaking down the system into modules such as complaint submission, complaint management, reporting, and analytics. Each module would have its own functionality and would be responsible for specific tasks within the system.

**3.3.4 Data Structure:**

Data structure refers to the organization of data within the system. In a complaint management system, the data structure would involve the storage and management of various data elements such as user information, complaint details, and resolution data.

The data structure should be designed to ensure efficient data retrieval, manipulation, and storage.

**3.3.5 Software Procedure:**

Software procedure refers to the process of defining the various steps and actions that are required to complete a specific task within the system. In a complaint management system, software procedures would include the steps involved in submitting a complaint, the actions required for resolving a complaint, and the procedures for generating reports and analytics.

Each software procedure should be designed to ensure that the system operates efficiently and effectively.

**3.4** **Table Design:**

**Login Table:**

The login records are created with the Email and password. There are three type of users. they are Admin, SubAdmin And Users.

|  |  |  |
| --- | --- | --- |
| FIELDS | DATATYPES | DESCRIPTION |
| Table Name | String | Stored the number of tables from the login |

**Student Table:**

To create student table where it also needs Name, Email, Password Type of the user which they belong to and can be created only by admin.

|  |  |  |
| --- | --- | --- |
| FIELDS | DATATYPE | DESCRIPTION |
| Name | String | Valid Student Name |
| Email | String | Valid Student Email |
| Password | String of Characters | Valid Student Password |

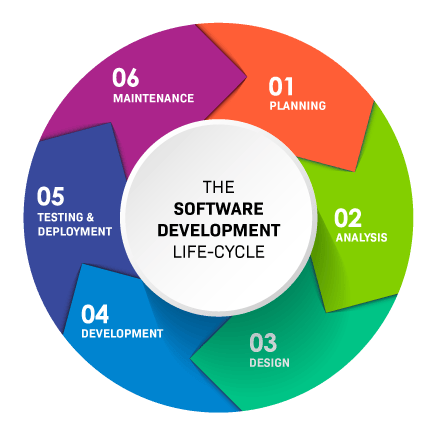
**3.7 Development approach:**

## 3.7.1 Methodology:

## The agile Development Model was used as the framework for the development of the HDMS. However, the structure of this thesis is based on Software Development Life Cycle (SDLC) to provide an overview of the project.

**Software Development Life Cycle (SDLC):**

Software Development Life Cycle, or SDLC, is widely used in the software industry as a framework to define tasks performed in each step of the development process. The goal of SDLC is to provide the best product with the resources that give.



**A typical Software Development Life Cycle consists of the following stages:**

1. Planning and requirement analysis: Based on the objective of the product, a project plan will be conducted. The product feasibility will be assessed according to economic resources and technical requirements. The quality assurance requirements for the product will then be set. Risk assessment is also done in this stage to ensure the project proceeds with minimum risks
2. Defining requirements: Product requirements are to be defined and documented clearly in this stage. This is done through a Software Requirement Specification (SRS) document.
3. Designing the product architecture: Based on the SRS document, a Design Document Specification (DDS) document is formed. It contains design proposals for the product architecture including the flow of data with external, and third-party modules if such modules are used. The document will then be reviewed by important stakeholders to decide on the best approach.
4. Developing the product: This is the stage where the product is built. The programming code is generated according to the DDS document. The programming language is chosen depending on which type of software is being developed. Coding guidelines are defined by the developer’s organization.
5. Testing the product: In this stage, the software is tested for defects. Product defects are reported, tracked, and fixed. The testing phase is repeated until the product reaches the standards defined in the SRS

**bogus negative FN**

1. Deployment and Maintenance: After the product is carefully tested, it is ready to be deployed. The product is deployed according to the strategy of the organization. Maintenance is done after the product is released when new errors are discovered or when enhancements are needed.

**CHAPTER 4**

**4. TESTING AND IMPLEMENTATION**

**4.1 Testing**

Testing is the process of evaluating a software application or system to identify defects, errors, or bugs in its functionality or behavior. The objective of testing is to ensure that the software meets its specified requirements, is reliable, performs as expected, and is free from defects that could impact its usability, performance, or security.

**4.1.1 Input Validation Testing**

Input validation testing is an essential part of ensuring the reliability and security of any software system, including a help desk management system. Input validation testing involves verifying that the system can properly handle various types of input, including invalid or unexpected inputs, without crashing or behaving unpredictably.

* Test for mandatory inputs: Test the application to ensure that it displays appropriate error messages when mandatory input fields are not filled. For example, if the regno, department, name, email, password field is mandatory, but left blank, the application should display an error message indicating that the field is required.
* Data Type: Validate the data type of each field, such as integer, string. Ensure that invalid data types are rejected and the system displays an appropriate error message.
* Format: Validate the format of fields that require specific formatting, such as name or email address. Ensure that the data is entered in the correct format, and the system displays an appropriate error message if the format is incorrect.
* Special Characters: Validate the input for special characters, such as apostrophes or quotation marks. Ensure that the system handles these characters appropriately to prevent any potential security vulnerabilities.
* Error Messages: Ensure that appropriate error messages are displayed for each type of input validation failure to guide the user in correcting the errors.

**CHAPTER 5**

**5. CONCLUSION**

The complaint management system (CMS) is a valuable tool for students to submit and track their complaints efficiently and effectively. The project has been developed using React.js and Node.js technologies and follows a modular and structured design process. The system's features and functionalities, such as complaint submission, tracking, and feedback mechanism, make it user-friendly and efficient. Although there may be limitations and room for improvements, the system has significant potential for future enhancements, including the addition of more features, machine learning algorithms, and robust security measures.

Overall, the complaint management system project has achieved its objective of providing a simple and efficient way for students to submit and track their complaints.

The key features of an CMS include a user-friendly interface, students tracking and management, escalation and prioritization of complaints, reporting and analytics, and integration with other systems. By implementing an CMS, colleges can ensure that all complaints are addressed promptly, and that students receive the necessary support and assistance to achieve their academic goals.

**5.1 Directions for future enhancements references:**

In the Future following Modules will be associated with Help Desk Management System:

* Feedback
* Users rating
* Social media integration

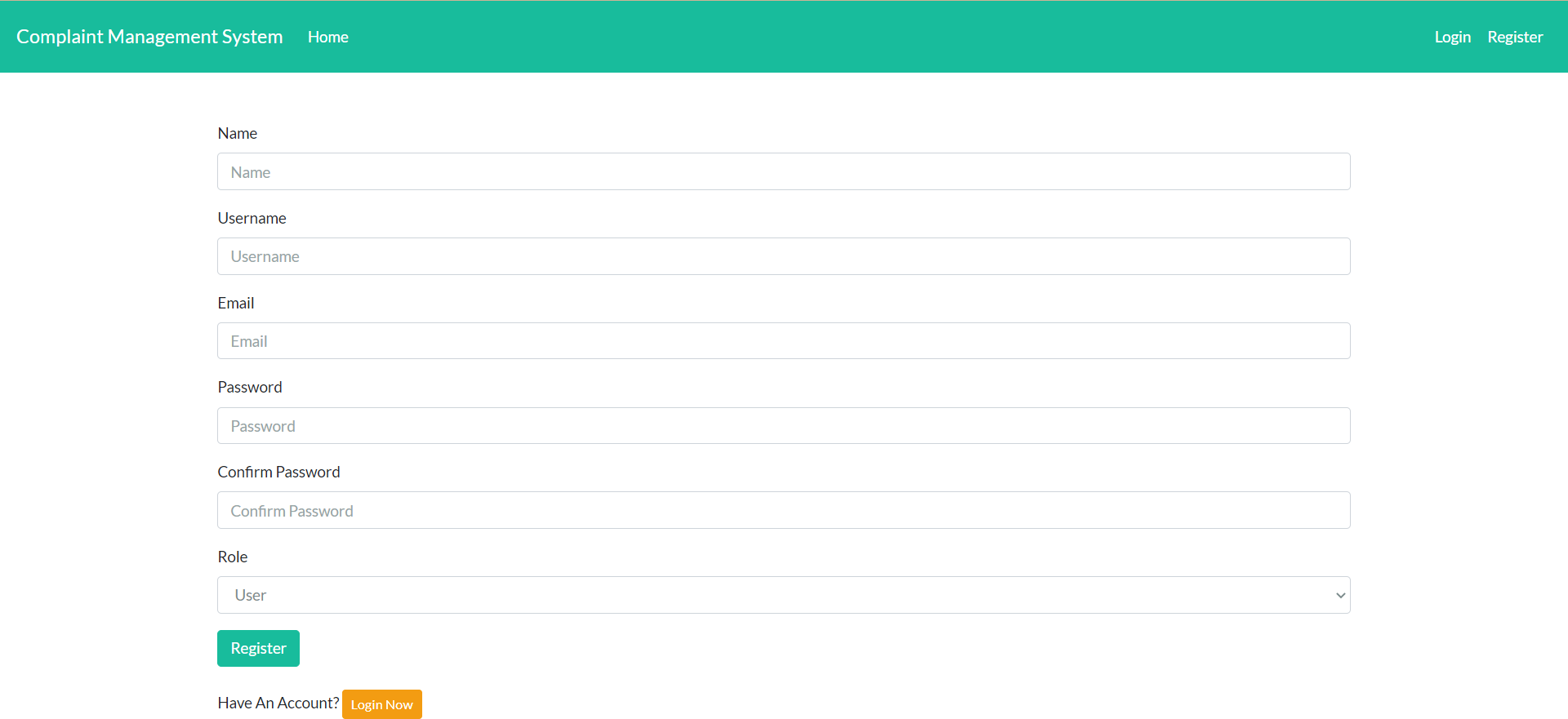
**BIBLIOGRAPHY:**

* <https://developer.mozilla.org/en-US/>
* <https://react.dev/>
* <https://nodejs.org/en/docs>
* <https://www.mongodb.com/>
* <https://www.w3schools.com/>

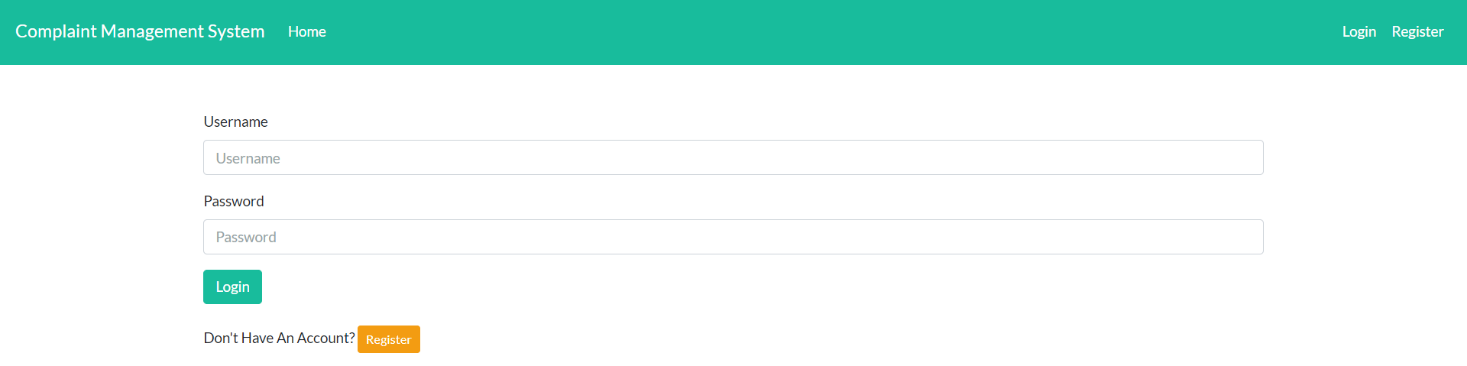
**ANNEXURE**

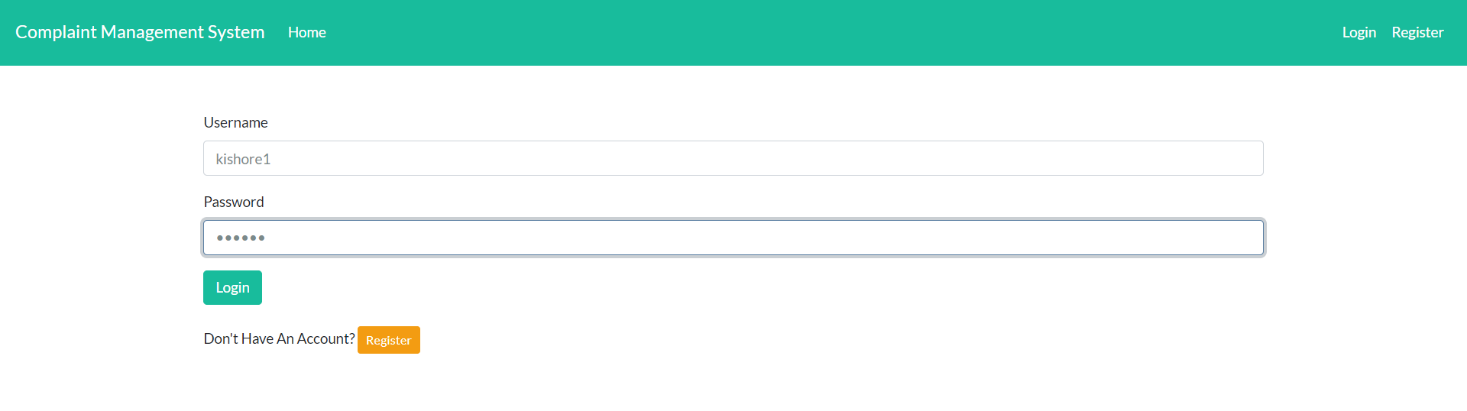
**ANNEXURE - A - Output design**

**Admin & User Register page:**

****

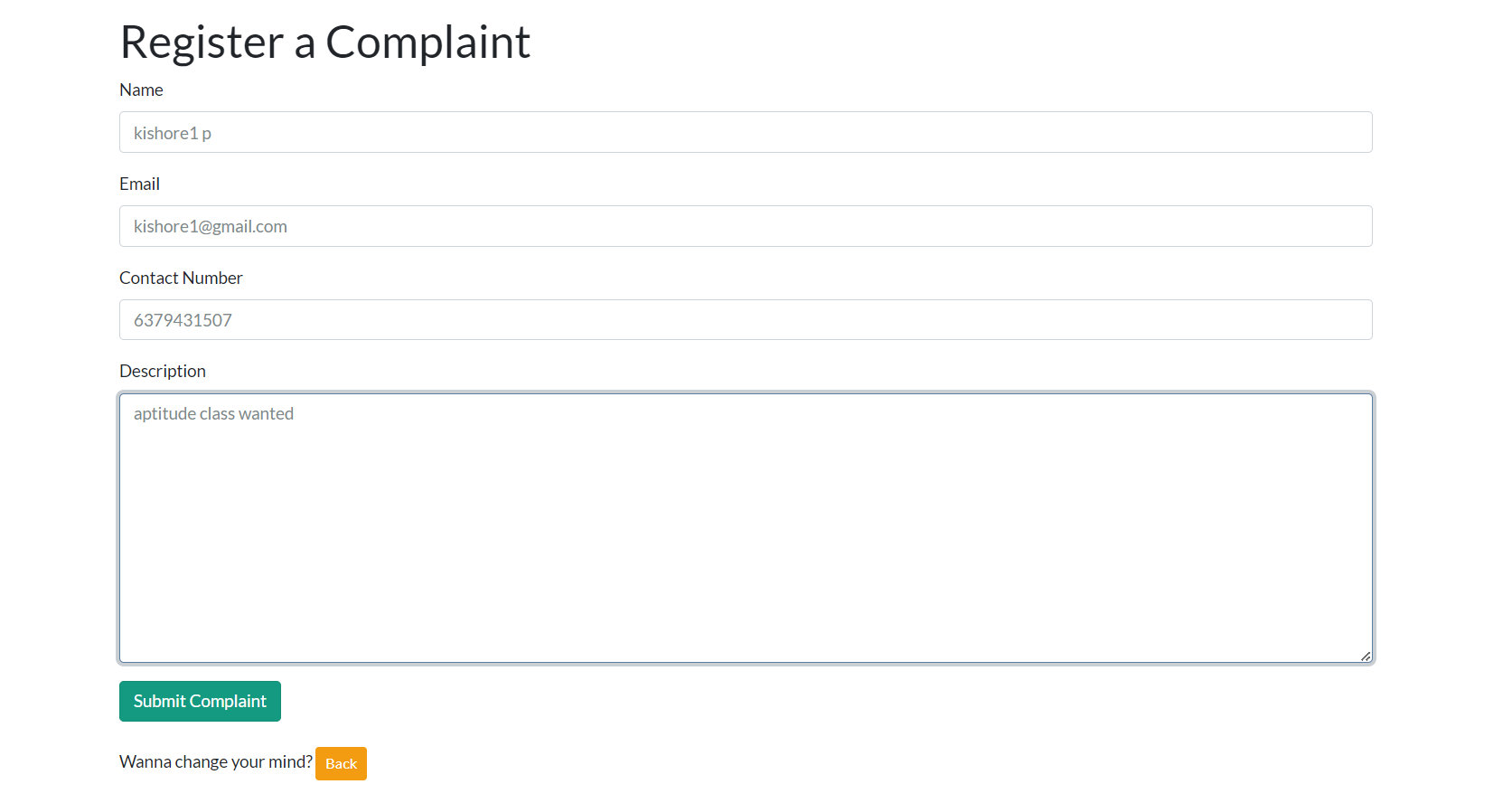
**Admin & User Login Page:**

****

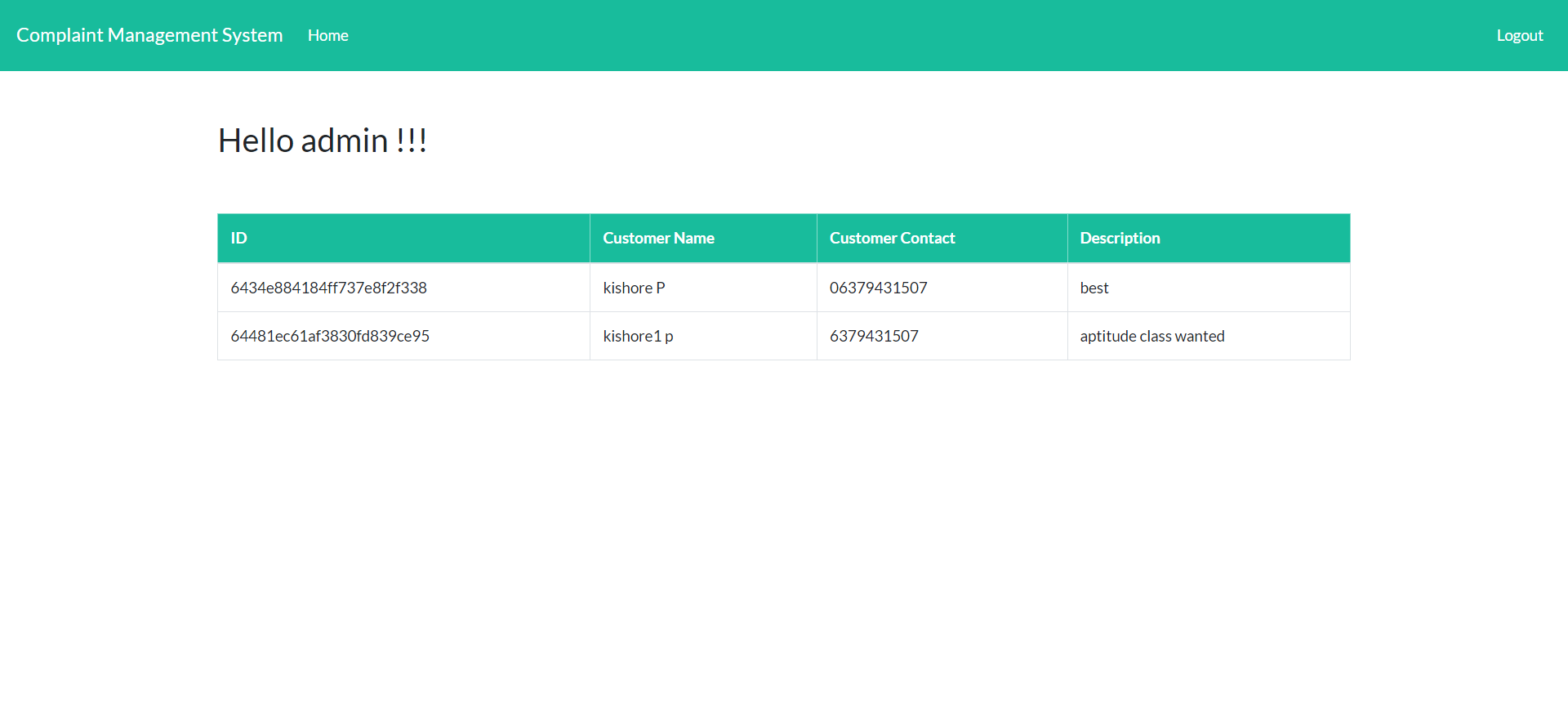
****

**User Complaint Form:**

****

****

**Admin Dashboard:**

****

**ANNEXURE - B - Source code**

**Client Side:**

**Admin handlebars:**

**<h3>Hello {{user.name}} !!!</h3>**

**<br>**

**<br>**

**<table class="table table-bordered">**

**<thead class="table-success">**

**<tr>**

**<th scope="col">ID</th>**

**<th scope="col">Customer Name</th>**

**<th scope="col">Customer Contact</th>**

**<th scope="col">Description</th>**

**</tr>**

**</thead>**

**<tbody>**

**{{#each complaints}}**

**<tr>**

**<td>{{\_id}}</td>**

**<td>{{name}}</td>**

**<td>{{contact}}</td>**

**<td>{{desc}}</td>**

**</tr>**

**{{/each}}**

**</tbody>**

**</table>**

**Main handlebars:**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>Complaint System</title>**

**<link rel="stylesheet" href="/css/style.css">**

**<link rel="stylesheet" href="css/bootstrap.min.css">**

**</head>**

**<body>**

**<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-5">**

**<a class="navbar-brand" href="#">Complaint Management System</a> PC helpedaccuracy, review and f1-score**

**<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor02" aria-controls="navbarColor02" aria-expanded="false" aria-label="Toggle navigation">**

**<span class="navbar-toggler-icon"></span>**

**</button>**

**<div class="collapse navbar-collapse" id="navbarColor02">**

**<ul class="navbar-nav ">**

**<li class="nav-item ">**

**<a class="nav-link" href="/">Home <span class="sr-only">(current)</span></a>**

**</li>**

**</ul>**

**<ul class="navbar-nav ml-auto">**

**{{#if user}}**

**<li class="nav-item "><a class="nav-link" href="/logout">Logout</a></li>**

**{{else}}**

**<li class="nav-item "><a class="nav-link" href="/login">Login</a></li>**

**<li class="nav-item "><a class="nav-link" href="/register">Register</a></li>**

**{{/if}}**

**</ul>**

**PC helped**

**</div>**

**</nav>**

**<div class="container">**

**{{#if success\_msg}}**

**<div class="alert alert-success">{{success\_msg}}</div>**

**{{/if}}**

**{{#if error\_msg}}**

**<div class="alert alert-danger">{{error\_msg}}</div>**

**{{/if}}**

**{{#if error}}**

**<div class="alert alert-danger">{{error}}</div>**

**{{/if}}**

**{{#if errors}}**

**{{#each errors}}**

**<div class="alert alert-danger">{{msg}}</div>**

**{{/each}}**

**{{/if}}**

**{{{body}}}**

**</div>**

**</body>**

**</html>**

**Login handlebars:**

**<form method="post" action="/login">**

**<div class="form-group">**

**<label>Username</label>**

**<input class="form-control" type="text" name="username" placeholder="Username">**

**</div>**

**<div class="form-group">**

**<label>Password</label>**

**<input class="form-control" type="password" name="password" placeholder="Password">**

**</div>**

**<button class="btn btn-success" type="submit">Login</button>**

**<br>**

**<br>**

**Don't Have An Account? <a class="btn btn-warning btn-sm" href="/register">Register</a>**

**</form>**

**Register handlebars:**

**<form method="post" action="/register">**

**<div class="form-group">**

**<label for="name">Name</label>**

**<input class="form-control" type="text" name="name" id="name" placeholder="Name">**

**</div>**

**<div class="form-group">**

**<label for="username">Username</label>**

**<input class="form-control" type="text" name="username" id="username" placeholder="Username">**

**</div>**

**<div class="form-group">**

**<label for="email">Email</label>**

**<input class="form-control" type="email" name="email" id="email" placeholder="Email">**

**</div>**

**<div class="form-group">**

**<label for="password">Password</label>**

**<input class="form-control" type="text" name="password" id="password" placeholder="Password">**

**</div>**

**<div class="form-group">**

**<label for="password2">Confirm Password</label>**

**<input class="form-control" type="text" name="password2" id="password2" placeholder="Confirm Password">**

**</div>**

**<div class="form-group">**

**<label for="role">Role</label>**

**<select class="form-control" name="role" id="role">**

**<option value="user">User</option>**

**<option value="admin">Admin</option>**

**</select>**

**</div>**

**<button class="btn btn-success" type="submit">Register</button>**

**<br>**

**<br>**

**Have An Account? <a class="btn btn-warning btn-sm" href="/login">Login Now</a>**

**</form>**

**Server Side:**

**Index.js:**

**const express = require('express');**

**const router = express.Router();**

**const passport = require('passport');**

**const LocalStrategy = require('passport-local').Strategy;**

**let User = require('../models/user');**

**let Complaint = require('../models/complaint');**

**let ComplaintMapping = require('../models/complaint-mapping');**

**// Home Page - Dashboard**

**router.get('/', ensureAuthenticated, (req, res, next) => {**

**res.render('index');**

**});**

**// Login Form**

**router.get('/login', (req, res, next) => {**

**res.render('login');**

**});**

**// Register Form**

**router.get('/register', (req, res, next) => {**

**res.render('register');**

**});**

**// Logout approval dataset**

**router.get('/logout', ensureAuthenticated,(req, res, next) => {**

**req.logout();**

**req.flash('success\_msg', 'You are logged out');**

**res.redirect('/login');**

**});**

**// Admin**

**router.get('/admin', ensureAuthenticated, (req,res,next) => {**

**Complaint.getAllComplaints((err, complaints) => {**

**if (err) throw err;**

**User.getEngineer((err, engineer) => {**

**if (err) throw err;**

**res.render('admin/admin', {**

**complaints : complaints,**

**engineer : engineer,**

**});**

**});**

**});**

**});**

**// Assign the Complaint to Engineer**

**router.post('/assign', (req,res,next) => {**

**const complaintID = req.body.complaintID;**

**const engineerName = req.body.engineerName;**

**req.checkBody('complaintID', 'Contact field is required').notEmpty();**

**req.checkBody('engineerName', 'Description field is required').notEmpty();**

**let errors = req.validationErrors();**

**if (errors) {**

**res.render('admin/admin', {**

**errors: errors**

**});**

**} else {**

**const newComplaintMapping = new ComplaintMapping({**

**complaintID: complaintID,**

**engineerName: engineerName,**

**});**

**ComplaintMapping.registerMapping(newComplaintMapping, (err, complaint) => {**

**if (err) throw err;**

**req.flash('success\_msg', 'You have successfully assigned a complaint to Engineer');**

**res.redirect('/admin');**

**});**

**}**

**});**

**// Junior Eng**

**router.get('/jeng', ensureAuthenticated, (req,res,next) => {**

**res.render('junior/junior');**

**});**

**//Complaint**

**router.get('/complaint', ensureAuthenticated, (req, res, next) => {**

**//console.log(req.session.passport.username);**

**//console.log(user.name);**

**res.render('complaint', {**

**username: req.session.user,**

**});**

**});**

**//Register a Complaint**

**router.post('/registerComplaint', (req, res, next) => {**

**const name = req.body.name;**

**const email = req.body.email;**

**const contact = req.body.contact;**

**const desc = req.body.desc;**

**const postBody = req.body;**

**console.log(postBody);**

**req.checkBody('contact', 'Contact field is required').notEmpty();**

**req.checkBody('desc', 'Description field is required').notEmpty();**

**let errors = req.validationErrors();**

**if (errors) {**

**res.render('complaint', {**

**errors: errors**

**});**

**} else {**

**const newComplaint = new Complaint({**

**name: name,**

**email: email,**

**contact: contact,**

**desc: desc,**

**});**

**Complaint.registerComplaint(newComplaint, (err, complaint) => {**

**if (err) throw err;**

**req.flash('success\_msg', 'You have successfully launched a complaint');**

**res.redirect('/');**

**});**

**}**

**});**

**// Process Register**

**router.post('/register', (req, res, next) => {**

**const name = req.body.name;**

**const username = req.body.username;**

**const email = req.body.email;**

**const password = req.body.password;**

**const password2 = req.body.password2;**

**const role = req.body.role;**

**req.checkBody('name', 'Name field is required').notEmpty();**

**req.checkBody('email', 'Email field is required').notEmpty();**

**req.checkBody('email', 'Email must be a valid email address').isEmail();**

**req.checkBody('username', 'Username field is required').notEmpty();**

**req.checkBody('password', 'Password field is required').notEmpty();**

**req.checkBody('password2', 'Passwords do not match').equals(req.body.password);**

**req.checkBody('role', 'Role option is required').notEmpty();**

**let errors = req.validationErrors();**

**if (errors) {**

**res.render('register', {**

**errors: errors**

**});**

**} else {**

**const newUser = new User({**

**name: name,**

**username: username,**

**email: email,**

**password: password,**

**role: role**

**}); concealed state**

**User.registerUser(newUser, (err, user) => {**

**if (err) throw err;**

**req.flash('success\_msg', 'You are Successfully Registered and can Log in');**

**res.redirect('/login');**

**});**

**}**

**});**

**// Local Strategy**

**passport.use(new LocalStrategy((username, password, done) => {**

**User.getUserByUsername(username, (err, user) => {**

**if (err) throw err;**

**if (!user) {**

**return done(null, false, {**

**message: 'No user found'**

**});**

**}**

**User.comparePassword(password, user.password, (err, isMatch) => {**

**if (err) throw err;**

**if (isMatch) {**

**return done(null, user);**

**} else {**

**return done(null, false, {**

**message: 'Wrong Password'**

**});**

**}**

**});**

**});**

**}));**

**passport.serializeUser((user, done) => {**

**var sessionUser = {**

**\_id: user.\_id,**

**name: user.name,**

**username: user.username,**

**email: user.email,**

**role: user.role,**

**}**

**done(null, sessionUser);**

**});**

**passport.deserializeUser((id, done) => {**

**User.getUserById(id, (err, sessionUser) => {**

**done(err, sessionUser);**

**});**

**});**

**// Login Processing**

**router.post('/login', passport.authenticate('local',**

**{**

**failureRedirect: '/login',**

**failureFlash: true**

**}), (req, res, next) => {**

**req.session.save((err) => {**

**if (err) {**

**return next(err);**

**}**

**if(req.user.role==='admin'){**

**res.redirect('/admin');**

**}**

**else if(req.user.role==='jeng'){**

**res.redirect('/jeng');**

**}**

**else{**

**res.redirect('/');**

**}**

**});**

**});**

**// Access Control**

**function ensureAuthenticated(req, res, next) {**

**if (req.isAuthenticated()) {**

**return next();**

**} else {**

**req.flash('error\_msg', 'You are not Authorized to view this page');**

**res.redirect('/login');**

**}**

**}**

**module.exports = router;**

**app.js:**

**const express = require('express');**

**const path = require('path');**

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

**const exphbs = require('express-handlebars');**

**const expressValidator = require('express-validator');**

**const flash = require('connect-flash');**

**const session = require('express-session');**

**const passport = require('passport');**

**const mongoose = require('mongoose');**

**const app = express();**

**const port = process.env.PORT || 3000;**

**const index = require('./routes/index');**

**// View Engine**

**app.engine('handlebars', exphbs({defaultLayout:'main'}));**

**app.set('view engine', 'handlebars');**

**// Static Folder**

**app.use(express.static(path.join(\_\_dirname, 'public')));**

**// Body Parser Middleware**

**app.use(bodyParser.json());**

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

**// Express Session**

**app.use(session({**

**secret: 'secret',**

**saveUninitialized: true,**

**resave: true,**

**maxAge: null,**

**cookie : { httpOnly: true, maxAge: 2419200000 } // configure when sessions expires**

**}));**

**// Init passport**

**app.use(passport.initialize());**

**app.use(passport.session());**

**// Express messages**

**app.use(flash());**

**app.use((req, res, next) => {**

**res.locals.success\_msg = req.flash('success\_msg');**

**res.locals.error\_msg = req.flash('error\_msg');**

**res.locals.error = req.flash('error');**

**res.locals.user = req.user || null;**

**next();**

**});**

**// Express Validator**

**app.use(expressValidator({**

**errorFormatter: (param, msg, value) => {**

**let namespace = param.split('.')**

**, root = namespace.shift()**

**, formParam = root;**

**while(namespace.length) {**

**formParam += '[' + namespace.shift() + ']';**

**}**

**return {**

**param : formParam,**

**msg : msg,**

**value : value**

**};**

**}**

**}));**

**app.use('/', index);**

**// Start Server**

**app.listen(port, () => {**

**console.log('Server started on port '+port);**

**});**

**Complaint-mapping.js:**

**const mongoose = require('mongoose')**

**const dbconnect = require('../db')**

**//Call the db to connect the mongo db**

**dbconnect()**

**// Complaint Schema**

**const ComplaintMappingSchema = mongoose.Schema({**

**complaintID: {**

**type: String,**

**required: true**

**},**

**engineerName: {**

**type: String,**

**required: true**

**},**

**});**

**const ComplaintMapping = module.exports = mongoose.model('ComplaintMapping', ComplaintMappingSchema);**

**module.exports.registerMapping = function (newComplaintMapping, callback) {**

**newComplaintMapping.save(callback);**

**}**

**Complaint.js:**

**const mongoose = require('mongoose')**

**const dbconnect = require('../db')**

**//Call the db to connect the mongo db**

**dbconnect()**

**// Complaint Schema**

**const ComplaintSchema = mongoose.Schema({**

**name: {**

**type: String**

**},**

**email: {**

**type: String**

**},**

**contact: {**

**type: String**

**},**

**desc: {**

**type: String**

**}**

**});**

**const Complaint = module.exports = mongoose.model('Complaint', ComplaintSchema);**

**module.exports.registerComplaint = function (newComplaint, callback) {**

**newComplaint.save(callback);**

**}**

**module.exports.getAllComplaints = function(callback){**

**Complaint.find(callback);**

**}**

**user.js:**

**const mongoose = require('mongoose')**

**const dbconnect = require('../db')**

**const bcrypt = require('bcryptjs');**

**//Call the db to connect the mongo db**

**dbconnect()**

**// User Schema**

**const UserSchema = mongoose.Schema({**

**name: {**

**type: String**

**},**

**username: {**

**type: String,**

**unique: true,**

**required: true**

**},**

**email: {**

**type: String**

**},**

**password: {**

**type: String,**

**required: true**

**},**

**role: {**

**type: String**

**}**

**});**

**const User = module.exports = mongoose.model('User', UserSchema);**

**module.exports.registerUser = function (newUser, callback) {**

**bcrypt.genSalt(10, (err, salt) => {**

**bcrypt.hash(newUser.password, salt, (err, hash) => {**

**if (err) {**

**console.log(err);**

**}**

**newUser.password = hash;**

**newUser.save(callback);**

**});**

**});**

**}**

**module.exports.getUserByUsername = function(username, callback){**

**const query = {username: username}**

**User.findOne(query, callback);**

**}**

**module.exports.getUserById = function(id, callback){**

**User.findById(id, callback);**

**}**

**module.exports.comparePassword = function(candidatePassword, hash, callback){**

**bcrypt.compare(candidatePassword, hash, (err, isMatch) => {**

**if(err) throw err;**

**callback(null, isMatch);**

**});**

**}**

**module.exports.getEngineer = function(callback){**

**const query = {role: "jeng"}**

**User.find(query, callback);**

**}**