**Grievance Management System**

A Minor Project Report

Submitted in partial fulfillment of requirement of the Degree of

## BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING

BY

Under the Guidance of

****

### Department of Computer Science & Engineering Faculty of Engineering

**MEDICAPS UNIVERSITY, INDORE- 453331**

**Grievance Management System**

A Minor Project Report

Submitted in partial fulfillment of requirement of the Degree of

## BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING

BY

Under the Guidance of

****

**Department of Computer Science & Engineering Faculty of Engineering**

**MEDICAPS UNIVERSITY, INDORE- 453331**

## Report Approval

The project work **“Grievance Management System”** is hereby approved as a creditable study of an engineering/computer application subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite for the Degree for which it has been submitted.

It is to be understood that by this approval the undersigned do not endorse or approve any statement made, opinion expressed, or conclusion drawn therein; but approve the “Project Report” only for the purpose for which it has been submitted.

Internal Examiner Name: Designation Affiliation

External Examiner Name: Designation Affiliation

## Declaration

I hereby declare that the project entitled **“Grievance Management System”** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in ‘Computer Science & Engineering’ completed under the supervision of **Prof. Roshni Verma,** Faculty of Engineering, Medi-Caps University Indore is an authentic work.

Further, I declare that the content of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma.

**Signature and name of the student(s) with date**

## Certificate

I, **Prof. Roshni Verma** certify that the project entitled **“Medgel Pvt. Ltd. Web Content Management System”** submitted in partial fulfillment for the award of the degree of Bachelor of Technology by **Anurag Verma** is the record carried out by him under my guidance and that the work has not formed the basis of award of any other degree elsewhere.

Prof. Roshni Verma

Computer Science & Engineering

Medi-Caps University, Indore

<Name of External Guide (If any)>

<Name of the Department>

Name of the Organization

Dr. Ratnesh Litoriya Head of the Department

Computer Science & Engineering Medi-Caps University, Indore

## Acknowledgements

I would like to express my deepest gratitude to the Honorable Chancellor, **Shri R C Mittal,** who has provided me with every facility to successfully carry out this project, and my profound indebtedness to **Prof. (Dr.) D. K. Patnaik,** Vice Chancellor, Medi-Caps University, whose unfailing support and enthusiasm has always boosted up my morale. I also thank **Prof. (Dr.) Pramod S. Nair,** Dean, Faculty of Engineering, Medi-Caps University, for giving me a chance to work on this project. I would also like to thank my Head of the Department **Dr. Ratnesh Litoriya** for his continuous encouragement for the betterment of the project.

It is their help and support, due to which we became able to complete the design and technical report.

Without their support this report would not have been possible.

***Students may write as per their experience.***

#### Anurag Verma

B.Tech. III Year (C)

Department of Computer Science & Engineering Faculty of Engineering

Medi-Caps University, Indore

## Abstract

The efficient management of student grievances is critical to maintaining transparency, trust, and satisfaction within any academic institution. At **Medi-Caps University**, the traditional grievance-handling methods often involve manual processing, limited accessibility, and lack of systematic tracking. These limitations result in delays, miscommunication, and reduced accountability. This project addresses these challenges by conceiving, designing, and implementing a **dynamic, role-based Grievance Management System (GMS)**, tailored specifically to the institutional needs of Medi-Caps University.

The GMS provides a web-based platform that allows students and faculty to lodge grievances online, monitor their resolution status, and receive timely updates. Simultaneously, administrative users are empowered to manage, categorize, respond to, and track complaints via a secure and intuitive dashboard. The system facilitates efficient grievance resolution while ensuring transparency and traceability of actions taken on each complaint.

Developed using a modern backend stack of **Node.js**, **Express.js**, and **MongoDB**, with **EJS** for server-side rendering, the system follows a modular and scalable architecture. Role-based access control is implemented to distinguish between students (complainants) and admin authorities (resolvers). The database schema is designed to store complaint metadata, user information, status updates, and resolution history in a normalized, query-optimized format.

Security and usability are integral to the system design. Authentication is managed securely using session-based or token-based methods, ensuring only authorized users can access or modify grievance records. The interface prioritizes user experience with responsive design elements, ensuring accessibility across various devices including mobile phones and desktops.

The development lifecycle encompassed a comprehensive analysis of user roles, functional requirements, and system constraints. The design phase included flow diagrams, UI wireframes, and entity-relationship modeling. Implementation involved integration of backend APIs with the frontend, and database connectivity, followed by thorough testing—unit, integration, and user acceptance testing—to validate system reliability and performance. Deployment and end-user guidelines were also documented to ensure smooth onboarding and maintenance.

The successful deployment of this system empowers Medi-Caps University to enhance institutional transparency, reduce grievance turnaround time, and maintain a digital grievance record for future policy and process improvements. Moreover, it sets the foundation for expanding the system to support departmental categorization, escalation policies, and analytical dashboards in future iterations.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
|  |  | **Page No.** |
|  | Report Approval | i |
|  | Declaration | ii |
|  | Certificate | iii |
|  | Acknowledgement | iv |
|  | Abstract | v |
|  | Table of Contents | vi |
|  | List of figures | viii |
|  | Abbreviations | ix |
| Chapter 1 | Introduction | 1 |
|  | 1.1 Introduction |  |
|  | 1.2 Literature Review |  |
|  | 1.3 Objectives |  |
|  | 1.4 Significance |  |
|  | 1.5 Research Design |  |
|  | 1.6 Source of Data |  |
|  | 1.7 Chapter Scheme |  |
| Chapter 2 | REQUIREMENTS SPECIFICATION | 5 |
|  | 2.1 User Characteris |  |
|  | 2.2 Functional Requirements |  |
|  | 2.3 Dependencies |  |
|  | 2.4 Performance Requirements |  |
|  | 2.5 Hardware Requirements |  |
|  | 2.6 Constraints & Assumptions |  |
| Chapter 3 | DESIGN ( Whichever is applicable) | 8 |
|  | 3.1 **System Design** |  |
|  | 3.1.1 Data Flow Diagrams (Level 0,Level1) |  |
|  | 3.1.2 Activity Diagram |  |
|  | 3.1.3 Flow Chart |  |
|  | 3.1.4 Class Diagram |  |
|  | 3.1.5 ER Diagram |  |
|  | 3.1.6 Sequence diagram |  |
|  | 3.2 **Database Design** |  |
|  | 3.2.1 Logical Database Design |  |

|  |  |  |
| --- | --- | --- |
|  | 3.2.2 Physical Database Design |  |
| Chapter 4 | Implementation, Testing, and Maintenance | 19 |
|  | 4.1 Introduction to Languages, IDE’s, Tools and Technologies used  for Implementation |  |
|  | 4.2 Testing Techniques and Test Plans (According to project) |  |
|  | 4.3 Installation Instructions |  |
|  | 4.4 End User Instructions |  |
| Chapter 5 | Results and Discussions | 25 |
|  | 5.1 User Interface Representation (of Respective Project) |  |
|  | 5.2 Brief Description of Various Modules of the system |  |
|  | 5.3 Snapshots of system with brief detail of each |  |
|  | 5.4 Back Ends Representation (Database to be used ) |  |
|  | 5.5 Snapshots of Database Tables with brief description |  |
| Chapter 6 | Summary and Conclusions | 39 |
| Chapter 7 | Future scope | 40 |
|  | Bibliography | 41 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Figure No.** | **Figure Name** |
| **1** | **3.1.1 (a)** | Level 0 Data Flow Diagram |
| **2** | **3.1.1 (b)** | Level 1 Data Flow Diagram |
| **3** | **3.1.2** | Activity Diagram |
| **4** | **3.1.3** | Class Diagram |
| **5** | **3.2.4** | ER Diagram |
| **6** | **3.1.5** | Sequence Diagram |
| **7** | **3.2.1** | Logical Database Design |

**Abbreviations**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Abbreviation** | **Full Form** |
| 1. | CMS | Content Management System |
| 2. | UI | User Interface |
| 3. | UX | User Experience |
| 4. | SEO | Search Engine Optimization |
| 5. | API | Application Programming Interface |
| 6. | JWT | JSON Web Token |
| 7. | HTML | HyperText Markup Language |
| 8. | CSS | Cascading Style Sheets |
| 9. | IDE | Integrated Development Environment |

# Chapter 1 Introduction

### Introduction

In any large organization, especially educational institutions like Medi-Caps University, grievances and complaints are a common part of day-to-day operations. Addressing these issues in a timely, transparent, and systematic manner is crucial for maintaining trust and ensuring smooth functioning.

The **Grievance Management System (GMS)** is a web-based platform developed specifically for Medi-Caps University to streamline the process of filing, managing, and resolving student and faculty grievances. It replaces traditional paper-based or email-based systems with a digital solution that offers better tracking, faster response times, and improved communication between users and administrators.

This system allows students and faculty to submit complaints online, view the status of their issues, and receive updates in real-time. On the other hand, administrators can efficiently manage all complaints from a centralized dashboard, update statuses, and communicate resolutions. By implementing this system, the university can ensure better accountability, reduce processing time, and maintain a complete record of all grievance-related activity. The platform is secure, responsive, and designed with simplicity in mind so that all users—technical or non-technical—can operate it with ease.

### Literature Review

The development of the Grievance Management System (GMS) for Medi-Caps University draws upon existing practices in web-based grievance handling, user-centered application design, and modern software development frameworks.

Grievance redressal mechanisms have long existed in institutions through manual systems or basic email-based workflows. However, these methods are prone to delays, lack of transparency, and inefficient tracking. To improve this, universities and government bodies have moved toward centralized portals, such as the **UGC Online Grievance Redressal Portal**, where students can register complaints and track their status. While effective, such systems often lack customization for individual institutions, and their interfaces are not always user-friendly.

From a technology standpoint, research into modern web frameworks helped guide the architecture of the GMS. **Node.js** and **Express.js** were chosen for their non-blocking I/O model and efficient handling of multiple requests. These technologies offer modularity and performance advantages over older server-side platforms like PHP or ASP.NET. For the frontend, **EJS (Embedded JavaScript Templates)** was selected for its simplicity and ability to generate dynamic pages from server-rendered data, making it ideal for form-driven systems like a grievance portal.

The use of **MongoDB** as a NoSQL database was informed by its schema flexibility and efficient storage of complaint records. Unlike traditional relational databases, MongoDB handles dynamic and hierarchical data models effectively—beneficial for evolving datasets like grievances, user profiles, and status logs.

In addition to core technologies, attention was given to **UX/UI design standards**. Research from contemporary design practices emphasized the need for minimal, accessible interfaces and responsive layouts. This ensures that students and administrators can interact with the system smoothly across devices.

Security considerations also played a key role in system design. Research into **session-based authentication**, **role-based access control**, and **secure routing mechanisms** was incorporated to protect user data and ensure that only authorized users can perform certain actions (e.g., complaint resolution or status updates).

Overall, this project integrates insights from established grievance systems, modern web technologies, and academic best practices to create a reliable, responsive, and institution-specific solution tailored to Medi-Caps University.

### Problem Definition

In large academic institutions like **Medi-Caps University**, students and faculty members often face issues related to academics, infrastructure, administration, and campus services. However, the absence of a structured, digital grievance redressal mechanism results in inefficiencies, delays, and lack of accountability. Existing methods, such as handwritten complaint forms, emails, or verbal communication, often go unrecorded, unresolved, or are difficult to trace, leading to dissatisfaction among students and staff.

These traditional approaches have several key limitations:

* No centralized platform to register and track complaints.
* Lack of transparency in grievance resolution processes.
* No proper communication between the complainant and the concerned authority.
* Manual handling of complaints makes tracking and follow-up difficult.
* No digital record or analytics to identify recurring issues or areas of improvement.

Therefore, there is a strong need for a **dedicated, role-based web application** that allows users to submit grievances online and enables the administration to manage, respond to, and resolve those complaints in a timely manner. The proposed **Grievance Management System** aims to overcome these challenges by providing a transparent, secure, and user-friendly platform tailored specifically for the grievance redressal process at Medi-Caps University.

.

### Objectives

The primary goal of this project is to design and implement a web-based **Grievance Management System (GMS)** for **Medi-Caps University** that streamlines the process of submitting, managing, and resolving grievances in a transparent and efficient manner.

**🔹 Main Objectives:**

1. **To develop a centralized web platform** where students and faculty can lodge grievances related to academics, infrastructure, services, or administration.
2. **To enable real-time tracking** of submitted grievances and their resolution status by the users.
3. **To implement role-based access control** for different users (students, administrators) to ensure secure and appropriate use of features.
4. **To provide an intuitive and responsive user interface** that works across devices such as desktops, tablets, and smartphones.
5. **To ensure secure handling of grievance data** through session-based authentication and proper authorization checks.
6. **To reduce response time and improve accountability** in grievance redressal by providing timely notifications and updates.
7. **To maintain a digital record** of all grievances for future reference, analysis, and decision-making.
8. **To design a scalable and modular system** that can be enhanced in the future with features like analytics, notifications, and departmental workflows.

### Significance

The **Grievance Management System (GMS)** plays a crucial role in addressing one of the core administrative challenges faced by educational institutions—transparent and timely grievance redressal. In the context of **Medi-Caps University**, the implementation of a digital grievance handling system brings significant advantages for both students and the administration.

**🔹 Key Significance of the Project:**

* **Improved Transparency:** By providing real-time status updates and a clear tracking system, students can see how their complaints are being addressed, which helps build trust in the system.
* **Faster Resolution:** Administrators can quickly view, categorize, and resolve complaints from a centralized dashboard, significantly reducing response time.
* **Accessibility & Convenience:** Students and staff can submit grievances from anywhere, anytime—eliminating the need for physical visits or paperwork.
* **Better Record-Keeping:** The system maintains a complete digital history of all submitted grievances and resolutions, which can be used for analysis, audits, or future reference.
* **Enhanced Accountability:** Every complaint is assigned to a specific authority and tracked through its resolution cycle, encouraging timely action and ownership.
* **Data-Driven Improvements:** The grievance database can be analyzed periodically to identify recurring issues or departments with frequent complaints, supporting continuous institutional improvement.
* **Scalability:** The platform is built to accommodate future enhancements such as notification systems, escalation workflows, and reporting dashboards without requiring major restructuring.

### Technologies Used

**🔹 Hardware Platform**

* **Local Development Server:** Development and testing were performed on a local machine using Node.js runtime.
* **Cloud Hosting (Optional for Deployment):**  
  Platforms like **Render**, **Railway**, or **Vercel** (for static assets and frontend) can be used for deploying the system.  
  Alternatively, **Amazon Web Services (AWS)** or **Google Cloud Platform (GCP)** can be considered for production-level deployment.

**🔹 Software Platform**

**Frontend:**

* **HTML5:** Markup language for webpage structure.
* **CSS3:** Styling and layout.
* **JavaScript (ES6):** Adds interactivity and dynamic behavior.
* **EJS (Embedded JavaScript):** Server-side templating engine used to render HTML with dynamic data.

**Backend:**

* **Node.js:** JavaScript runtime for server-side development.
* **Express.js:** Lightweight web framework for handling routes, middleware, and APIs.

**🔹 Database**

* **MongoDB:** A NoSQL database used for storing grievance records, user profiles, and complaint statuses.
* **Mongoose:** ODM library used for schema definition and database interaction.

**🔹 Authentication & Security**

* **Express-Session:** Used for managing user login sessions securely.
* **bcrypt.js (optional):** For encrypting user passwords before storing them in the database.
* *(Note: JWT was researched but not used in this version. Can be implemented in future versions.)*

**🔹 Tools**

* **Visual Studio Code (VS Code):** Code editor used for writing and managing the project.
* **Git:** Version control tool used to manage code versions.
* **GitHub:** Online repository hosting service for collaboration and backup.
* **Postman:** API testing tool used during backend development.

**🔹 Libraries & Packages**

* **Body-parser:** Middleware for parsing incoming request bodies.
* **Connect-mongo:** Session store integration for MongoDB.
* **Multer (optional):** Can be used for file uploads (if grievance system includes attachments).
* **dotenv:** To manage environment variables securely.

### Chapter Scheme

This report is organized into the following chapters:

* + 1. Chapter 1: Introduces the background of the project, problem definition, objectives, significance, technologies used, and an overview of the report structure.
    2. Chapter 2: Requirements Specification Defines user characteristics, functional and non-functional requirements, dependencies, hardware/software needs, performance expectations, and system assumptions.
    3. Chapter 3: Design: Covers the system architecture and design diagrams such as Data Flow Diagrams (Level 0 & Level 1), Activity Diagram, Flowchart, Class Diagram, ER Diagram, and Sequence Diagram. Also includes logical and physical database designs.
    4. Chapter 4 Describes the tools and technologies used, coding environment, testing techniques, installation steps, and instructions for end users.
    5. Chapter 5: Presents the output of the project including user interface screenshots, brief descriptions of system modules, backend structure, and database snapshots with explanations.
    6. Chapter 6 Summarizes the project work, outcomes achieved, challenges faced, and how the system solves the defined problem.
    7. Chapter 7 Discusses possible improvements and future enhancements to extend the capabilities of the system.
    8. Appendix: Contains any detailed information, lengthy derivations, or raw experimental observations (if applicable).
    9. Bibliography: Lists all the references cited in the report.

# Chapter 2 REQUIREMENTS SPECIFICATION

This chapter outlines the specific requirements for the Grievance Management System, covering user characteristics, functional and non-functional aspects, and various constraints.

### 2.1 User Characteristics

This system is designed for two main user types:

**1. Students / Faculty (Complainants):**

* Basic understanding of how to use web applications.
* Can log in, file grievances, view status updates, and receive notifications.
* No technical knowledge required to use the platform.

**2. Admin / Grievance Cell Authorities:**

* Responsible for managing grievances, updating statuses, and providing resolutions.
* Access to the admin dashboard for viewing and responding to complaints.
* Requires basic administrative training to operate the system effectively.

### 2.2 Functional Requirements

The Grievance Management System is designed to fulfill the following core functions:

1. **User Login & Authentication**
   * Only registered users (students/faculty) and admins can log in using secure credentials.
2. **Grievance Submission**
   * Users can file a grievance by filling out a form that includes a title, category, and detailed description. Optional file attachment support is also provided.
3. **Grievance Tracking**
   * Users can view the status of submitted grievances (e.g., Pending, In Progress, Resolved) along with timestamps and any admin remarks.
4. **Admin Dashboard**
   * Admins have access to a dashboard to view, search, filter, and manage all complaints. They can sort grievances based on date, type, or status.
5. **Status Update and Resolution**
   * Admins can update the status of each grievance and add remarks for better communication with the complainant.
6. **Complaint History**
   * Both users and admins can view a history of all submitted and handled grievances, useful for tracking past issues.
7. **Logout Functionality**
   * All users can securely log out of the system to end their session.

### 2.3 Dependencies

The Grievance Management System website will have the following dependencies:

* + **Runtime:** Node.js ≥ 18 LTS
* **Server Framework:** Express.js 5
* **Database:** MongoDB 6 .x (with Mongoose ODM)
* **Templating:** EJS 3.x
* **Middleware & Utilities:** body‑parser, express‑session, connect‑mongo, multer (attachments), dotenv.
* **Browser Support:** Latest Chrome, Firefox, Edge; mobile browsers.
* **Hosting Options:** Render / Railway (development) or AWS EC2 / GCP VM / on‑prem VM (production).

### 2.4 Performance Requirements

The system should meet the following performance requirements:

 Concurrent users supported (baseline): ≥ 200 without noticeable delay.

 Average page‑load time (dashboard & forms): ≤ 3 s on 10 Mbps link.

 Database CRUD latency: ≤ 200 ms for 95 % of requests.

 File‑upload size limit: 5 MB per grievance; reject larger files gracefully.

 Backup: daily automatic backup of MongoDB; seven‑day retention minimum.

### Hardware Requirements

| **Layer** | **Minimum** | **Recommended** |
| --- | --- | --- |
| **Client** | Dual‑core CPU, 2 GB RAM | Quad‑core CPU, 4 GB RAM |
| **Server** | 1 vCPU, 1 GB RAM, 20 GB SSD | 2 vCPU, 4 GB RAM, 40 GB SSD |
| **Network** | 10 Mbps uplink | 100 Mbps or higher |

### 2.5 Constraints & Assumptions

**Constraints:**

1. System requires continuous internet connectivity.
2. Attachments limited to approved file types (PDF/JPG/PNG).
3. Only authenticated campus email addresses may register.

**Assumptions:**

1. Users provide accurate grievance details.
2. Admins act on complaints within the university’s SLA (e.g., 72 hours).
3. Future features (SMS/e‑mail alerts, analytics) can be integrated without major redesign due to modular architecture.

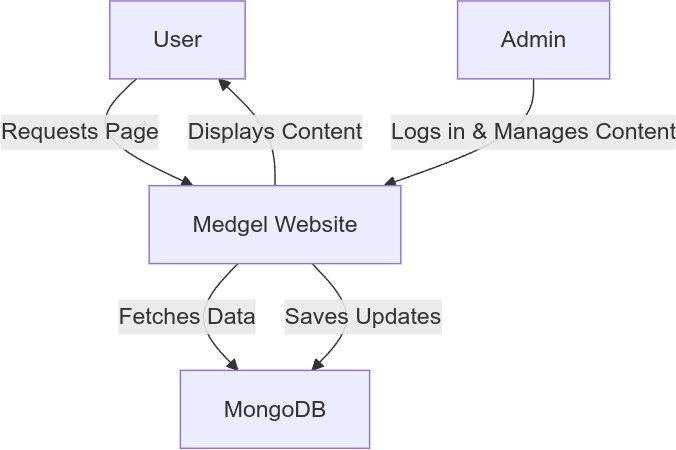
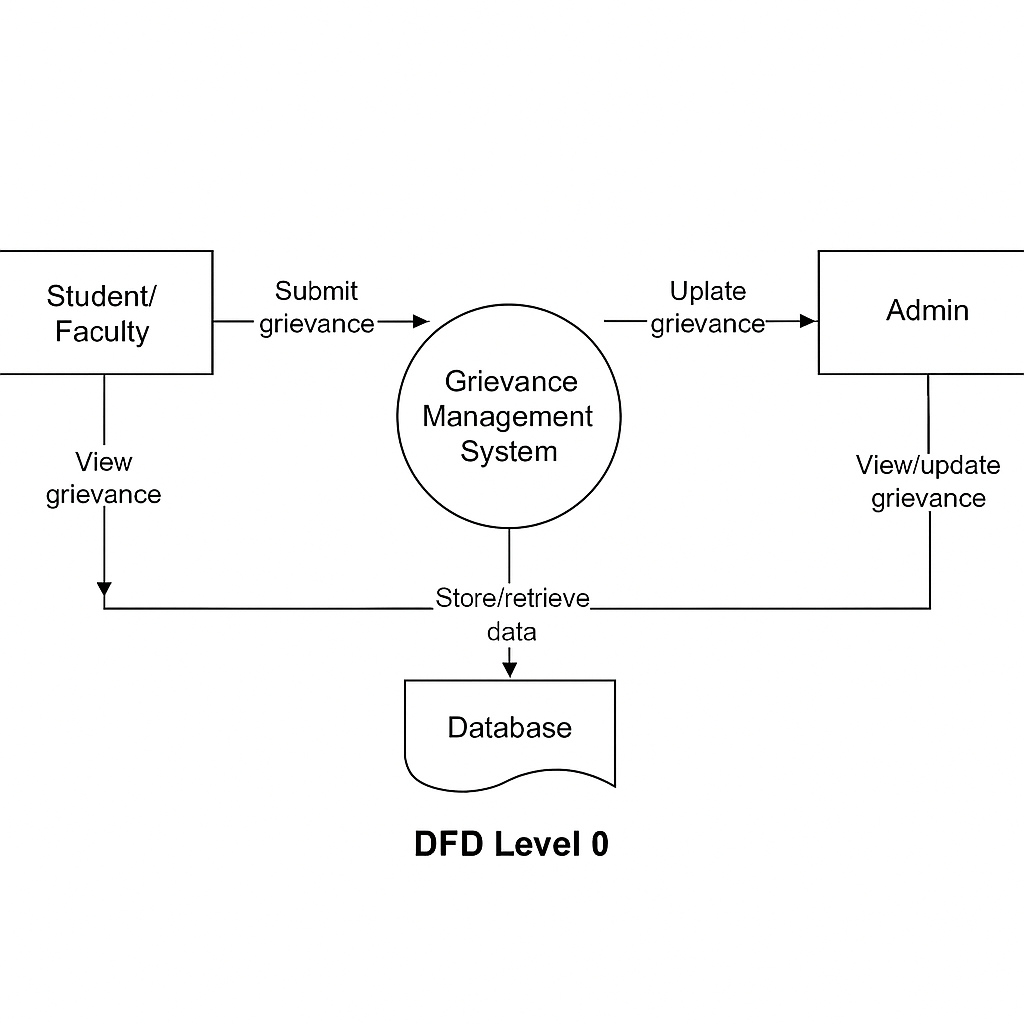
# Chapter 3 DESIGN

This chapter outlines the design of the Medgel Website with CMS, including the overall system architecture, user interface design principles, and the database schema.

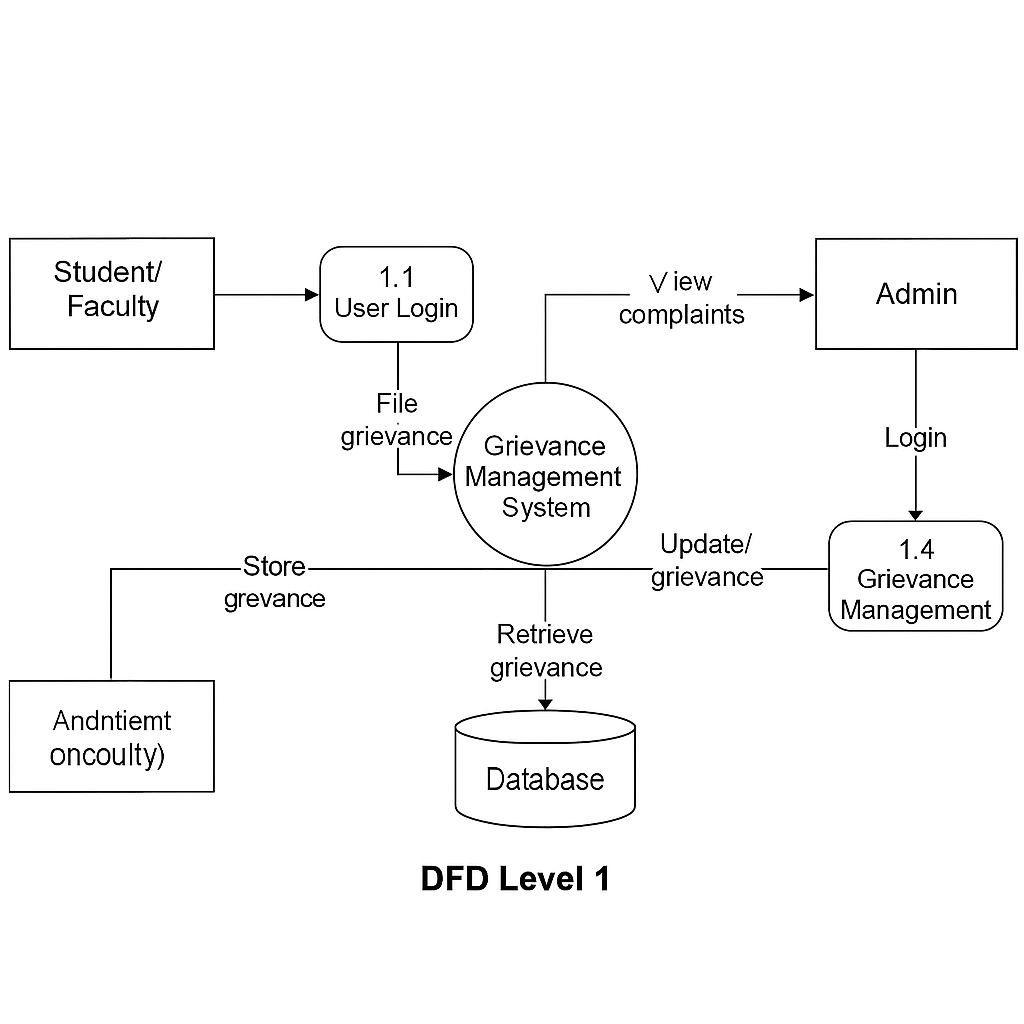
# System Design

### Data Flow Diagram

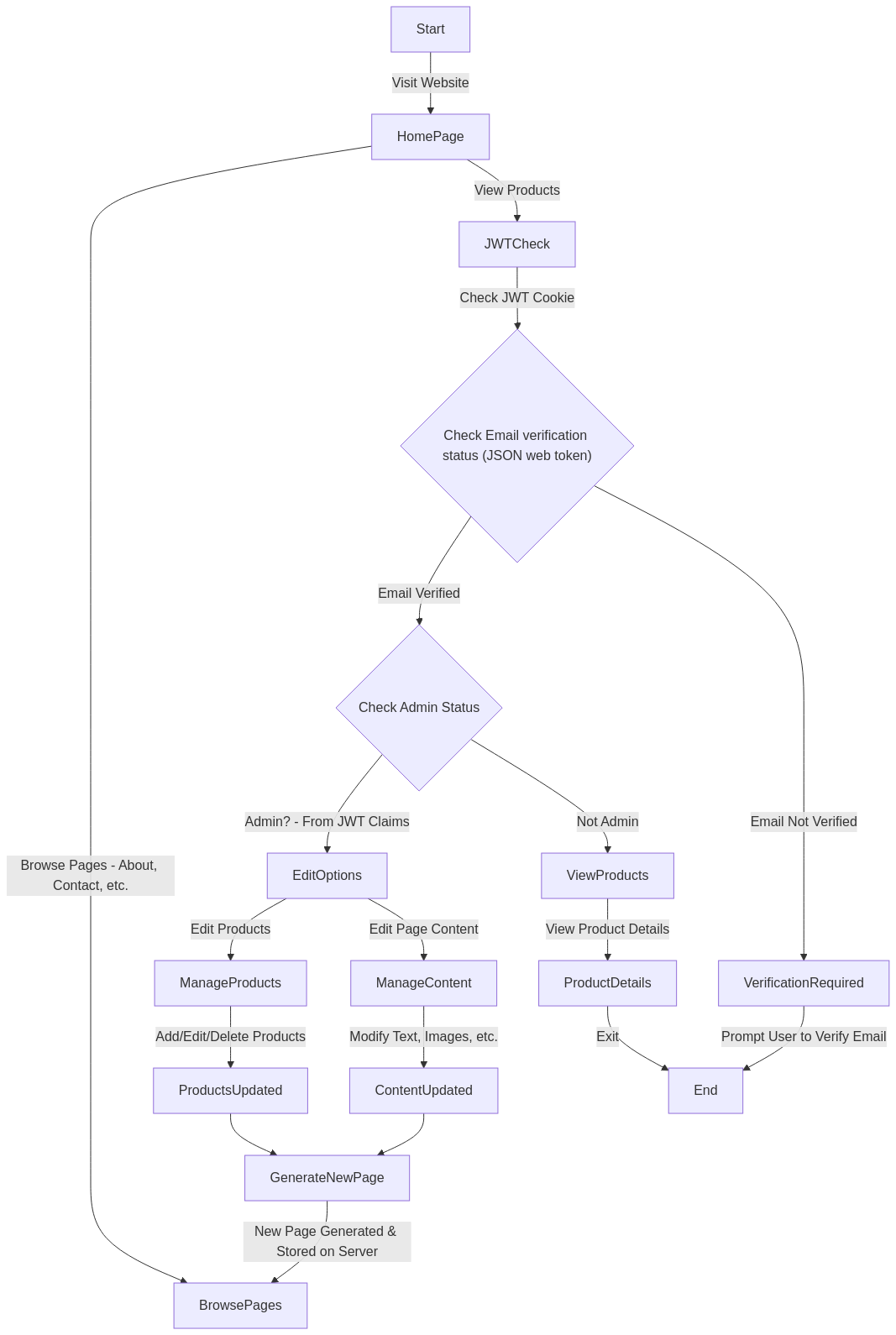
**3.1.1. (a) Level 0 Data Flow Diagram**

****

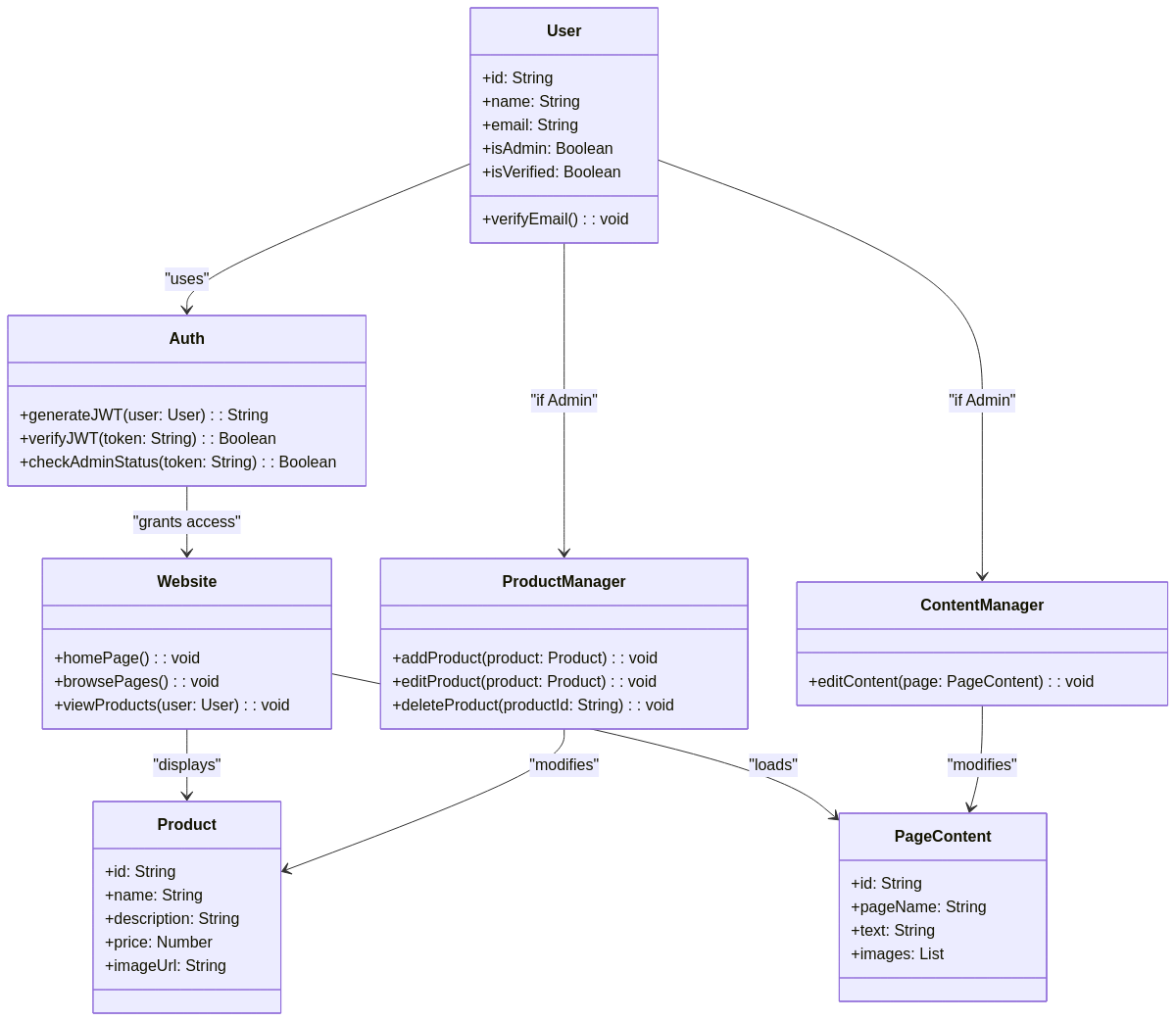
* + 1. **(b) Level 1 Data Flow Diagram**

****

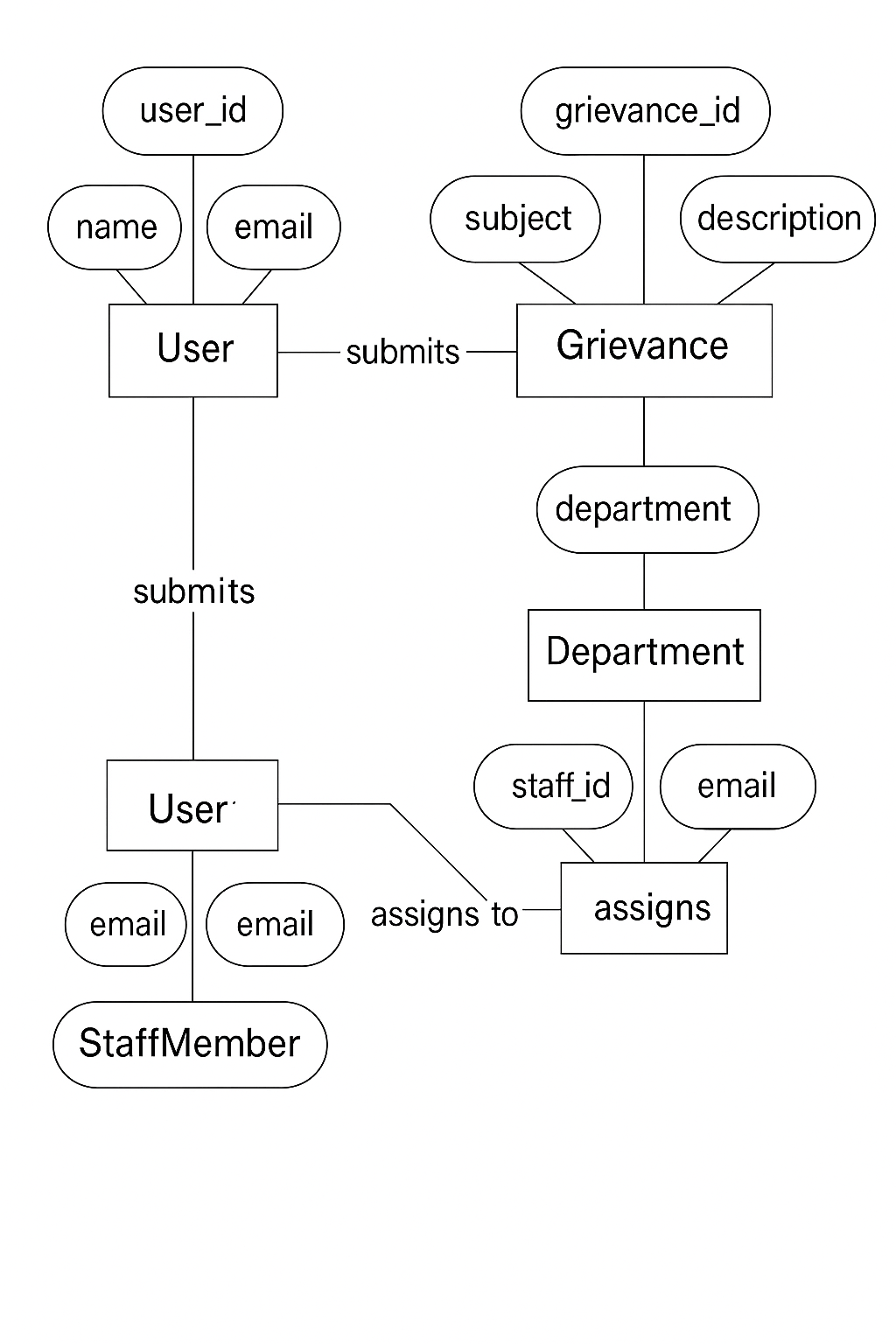
* + 1. **Activity Diagram**

****

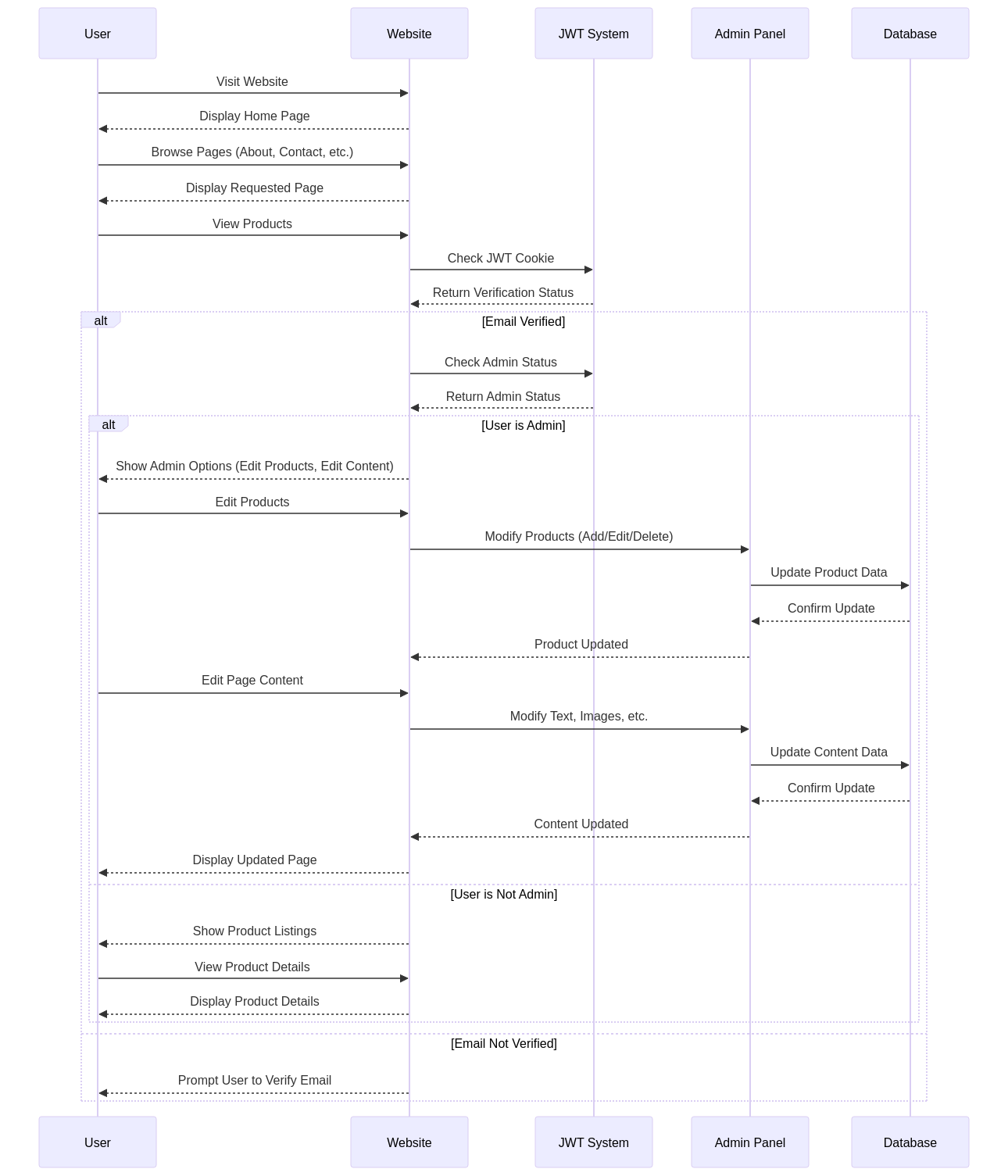
* + 1. **Class Diagram**

****

* + 1. **ER Diagram**

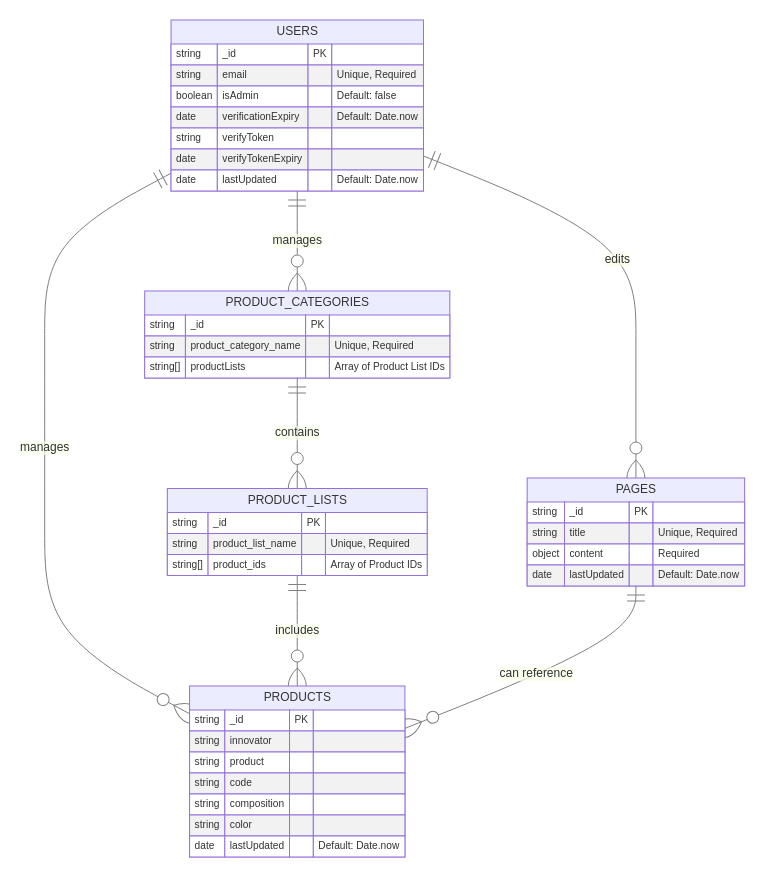


* + 1. **Sequence Diagram**

****

# Database Design

### Logical Database Design

****

* + 1. **Physical Database Design**

// page.js

const pagesSchema = new mongoose.Schema({ title: {

type: String, required: true, unique: true,

},

content: {

type: Object, required: true,

},

lastUpdated: { type: Date,

default: Date.now,

},

});

export default mongoose.models.Pages || mongoose.model("Pages", pagesSchema);

// productCategory.js

const product\_category = new mongoose.Schema({ product\_category\_name: {

type: String,

required: [true, "Please provide a category name"], unique: true,

},

productLists: [String],

});

const ProductCategory = mongoose.models.product\_category || mongoose.model("product\_category", product\_category);

export default ProductCategory;

// productList.js

const product\_list = new mongoose.Schema({ product\_list\_name: {

type: String,

required: [true, "Please provide a product list name"], unique: true,

},

product\_ids: [String],

});

const ProductList =

mongoose.models.product\_list || mongoose.model("product\_list", product\_list);

export default ProductList;

// products.js

const product = new mongoose.Schema({ innovator: String,

product: String, code: String, composition: String, color: String, lastUpdated: {

type: Date,

default: Date.now,

},

});

const Product = mongoose.models.product || mongoose.model("product", product);

export default Product;

// user.js

const userSchema = new mongoose.Schema({ email: {

type: String,

required: [true, "Please provide an email"], unique: true,

},

isAdmin: {

type: Boolean, default: false,

},

verificationExpiry: { type: Date, default: Date.now,

},

verifyToken: String, verifyTokenExpiry: Date, lastUpdated: {

type: Date, default: Date.now,

},

});

const User = mongoose.models.users || mongoose.model("users", userSchema);

export default User;

# Chapter 4 Implementation, Testing, and Maintenance

This chapter details the implementation process, testing strategies, and guidelines for installation and end-user operation of the GMS Website.

### Introduction to Languages, IDE’s, Tools and Technologies used for Implementation

 Languages **Used:**

* HTML5 – For structuring web pages.
* CSS3 – For styling and layout.
* JavaScript (ES6) – For adding interactivity.
* EJS – Templating language for rendering server-side views.

 Backend **Technologies:**

* Node.js – JavaScript runtime for server-side logic.
* Express.js – Web framework for building REST APIs.

 Database**:**

* MongoDB – NoSQL database for storing user data and grievances.
* Mongoose – Object Data Modeling (ODM) tool for MongoDB integration.

 IDEs **and Tools:**

* Visual Studio Code – Code editor for development.
* Postman – Used for API testing.
* Git & GitHub – Version control and repository management.

 **Session & Middleware:**

* express-session – For session management.
* connect-mongo – To store sessions in MongoDB.
* body-parser – For handling form data.

### Testing Techniques and Test Plans

A comprehensive testing strategy is crucial to ensure the reliability and functionality of the GMS Website. The following testing techniques will be employed:

* + - Integration Testing: Tested the flow between frontend forms, backend APIs, and the MongoDB database.
    - User Acceptance Testing (UAT): The system was shared with sample users (students and admins) to gather feedback on usability and functionality.
    - Responsiveness Testing: The website was tested on desktop, tablet, and mobile devices to ensure proper layout and functionality.
    - Security Testing: Verified password hashing, session management, and protection against common web vulnerabilities.

**Test Plans:**

* **Login/Logout:** Verified secure access for both users and admins.
* **Grievance Submission:** Checked form validations, file uploads, and database insertion.
* **Status Updates:** Confirmed admin can update complaint statuses with remarks.
* **Complaint History:** Ensured users can view status history accurately.

### Installation Instructions (haven’t changed yet)

To set up, build, and deploy the Medgel Website with CMS, follow these steps:

1. Clone the Repository: Obtain the project codebase from the GitHub repository.
2. Bash

git clone <repository\_url> cd <project\_directory>

1. Install Dependencies: Install the required Node.js packages using npm. npm install
2. Configure Environment Variables: Create a .env.local file in the project root and configure the necessary environment variables, such as the MongoDB connection

string (MONGODB\_URI), JWT secret key (JWT\_SECRET), and other relevant configurations specified in the .env.sample file. (like the following snippet)

MONGODB\_URI=mongodb://<username>:<password>@<host>:<port>/<dat abase\_name>?retryWrites=true&w=majority JWT\_SECRET=<your\_secret\_key>

1. Run the Development Server (Optional): Start the Next.js development server for local development and testing.

npm run dev

This will typically start the application on [http://localhost:3000.](http://localhost:3000/) You can access the website and admin dashboard locally to verify the functionality.

1. Build the Production Application: Once development is complete, build the Next.js application for production deployment. This command optimizes the application for performance.

npm run build

1. This process generates an optimized build of your application in a .next directory.
2. Deploy to the Cloud: To host your application on a cloud platform, follow the specific instructions for your chosen provider (e.g., Vercel, AWS, Google Cloud Platform):
   * Vercel: Vercel often provides seamless integration with Next.js projects hosted on GitHub, GitLab, or Bitbucket. You can typically connect your repository to Vercel through their web interface, and Vercel will automatically build and deploy your application upon code changes. Ensure your environment variables are configured in the Vercel project settings.
   * AWS (Amazon Web Services): Deployment on AWS might involve services like:
     + AWS Amplify: Simplifies the process of building and deploying full-stack web applications, including Next.js.
     + AWS S3 and CloudFront (for static export): If you performed a static export, you can host the out directory contents on an S3 bucket and use CloudFront as a Content Delivery Network (CDN) for improved performance.
     + AWS EC2 or Elastic Beanstalk (for server-rendered applications): If your application uses server-side rendering or API routes, you'll need a server environment like EC2 or Elastic Beanstalk to run the Next.js server. You'll need to configure your build process and environment variables on these services.
   * Google Cloud Platform (GCP): Similar to AWS, GCP offers various services for deployment:
     + Firebase Hosting (for static export): A fast and secure hosting service for static websites.
     + Google Cloud Storage and Cloud CDN (for static export): Host static files on Cloud Storage and use Cloud CDN for global content delivery.
     + Google Cloud Run or App Engine (for server-rendered applications): Managed platforms for running containerized applications (Cloud Run) or web applications (App Engine), suitable for Next.js server deployments. Configure your build and environment variables accordingly.
3. Refer to the official documentation of your chosen cloud provider for detailed deployment instructions specific to Next.js applications.
4. Connect Domain to the Cloud: Once your application is deployed, you'll likely want to connect it to a custom domain name:
   * Configure DNS Records: You'll need to update the DNS records of your domain name (usually managed by your domain registrar) to point to the IP address or hostname provided by your cloud hosting service. This typically involves adding or modifying A records (for IP addresses) or CNAME records (for hostnames/domain names provided by the cloud service).
   * Cloud Provider Domain Configuration: Most cloud hosting platforms provide tools or settings within their console to associate your custom domain with your deployed application. This might involve verifying domain ownership and configuring routing.
   * HTTPS (SSL/TLS): Ensure that HTTPS is enabled for your domain. Cloud providers like Vercel, AWS (with Certificate Manager and CloudFront), and GCP (with Google Cloud Certificates) offer services to provision and manage SSL/TLS certificates for secure connections. Configure this within your cloud provider's settings.

Remember to consult the specific documentation of your cloud hosting provider and domain registrar for detailed instructions on deployment and domain configuration.

### End User Instructions

**Accessing the Website (Visitors):**

* + - Open a web browser (Chrome, Firefox, Safari, etc.).
    - Navigate to the website's URL (provided by Medgel Pvt. Ltd.).
    - Browse the different sections of the website to find information about the company and its products.
    - The website is designed to be responsive, so it should adapt automatically to the screen size of the device being used.

**Accessing the Admin Dashboard (Administrators):**

1. Open a web browser.
2. Navigate to the admin dashboard URL (e.g., yourwebsite.com/admin).
3. Enter your assigned username and password in the login form.
4. Click the "Login" button.
5. Once logged in, you will have access to the various content management tools.

**Using the Admin Dashboard:**

* + Content Management: Navigate to the specific page that you want to edit contents of. Use the provided rich text editor to format content and insert, update or delete images from the media library.
  + Product Management: Go to the "Products" section to add new products, edit existing product details (name, description, images, specifications, price), and manage their status.
  + Media Library: Access the "Media" section to upload new images and other media files. You can organize and select these files when creating or editing content and products.
  + Navigation: Use the "Navigation" or "Menus" section to manage the website's main navigation structure. You can add, remove, and reorder menu items.
  + SEO Settings: When creating or editing pages and products, you will typically find options to set meta titles, meta descriptions, and URL slugs to optimize for search engines.
  + User Management: (Super Administrators only) The "Users" section allows you to create new administrator accounts and manage their roles and permissions.

**Note: this is changed 4.4 for GMS =>**

**For Students/Faculty:**

* Log in using provided credentials.
* Click on “File Grievance” to submit a complaint.
* Track complaint status in “My Complaints”.
* Log out when done.

**For Admin:**

* Log in with admin credentials.
* Access “Manage Complaints” to view and update grievance statuses.
* Use search and filters to manage submissions.
* Log out after handling complaints.

# Chapter 5

**Results and Discussions**

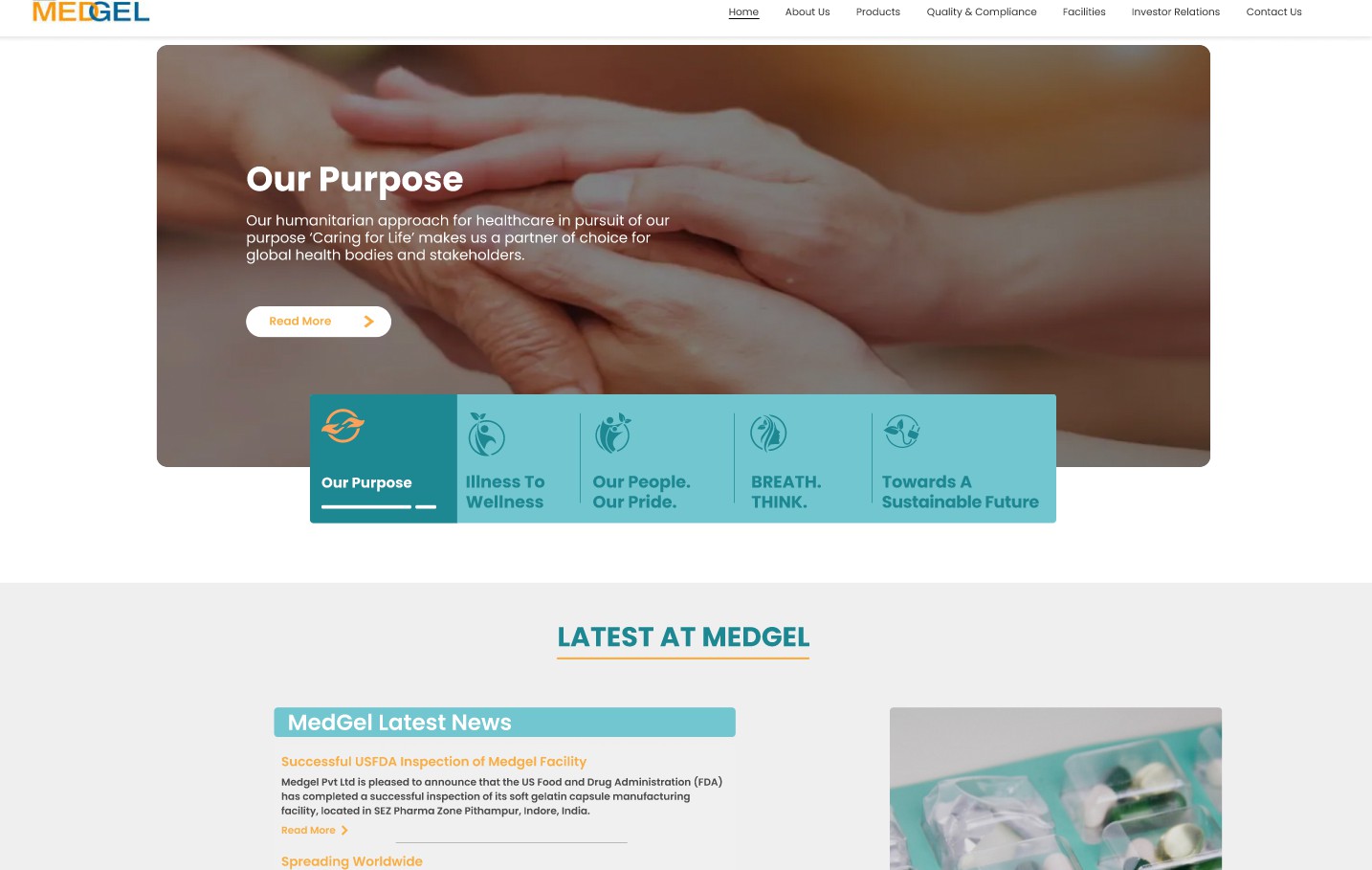
This chapter presents the results of the project, including the user interface representation, a description of the system modules, snapshots of the system and the database, and a discussion of the outcomes.

### User Interface Representation

The user interface of the public-facing website is designed to be clean, modern, and informative, reflecting the professional image of Medgel Pvt. Ltd. The responsive design ensures optimal viewing on various devices. Key elements include clear navigation, and easy access to company details.

**Home Page**

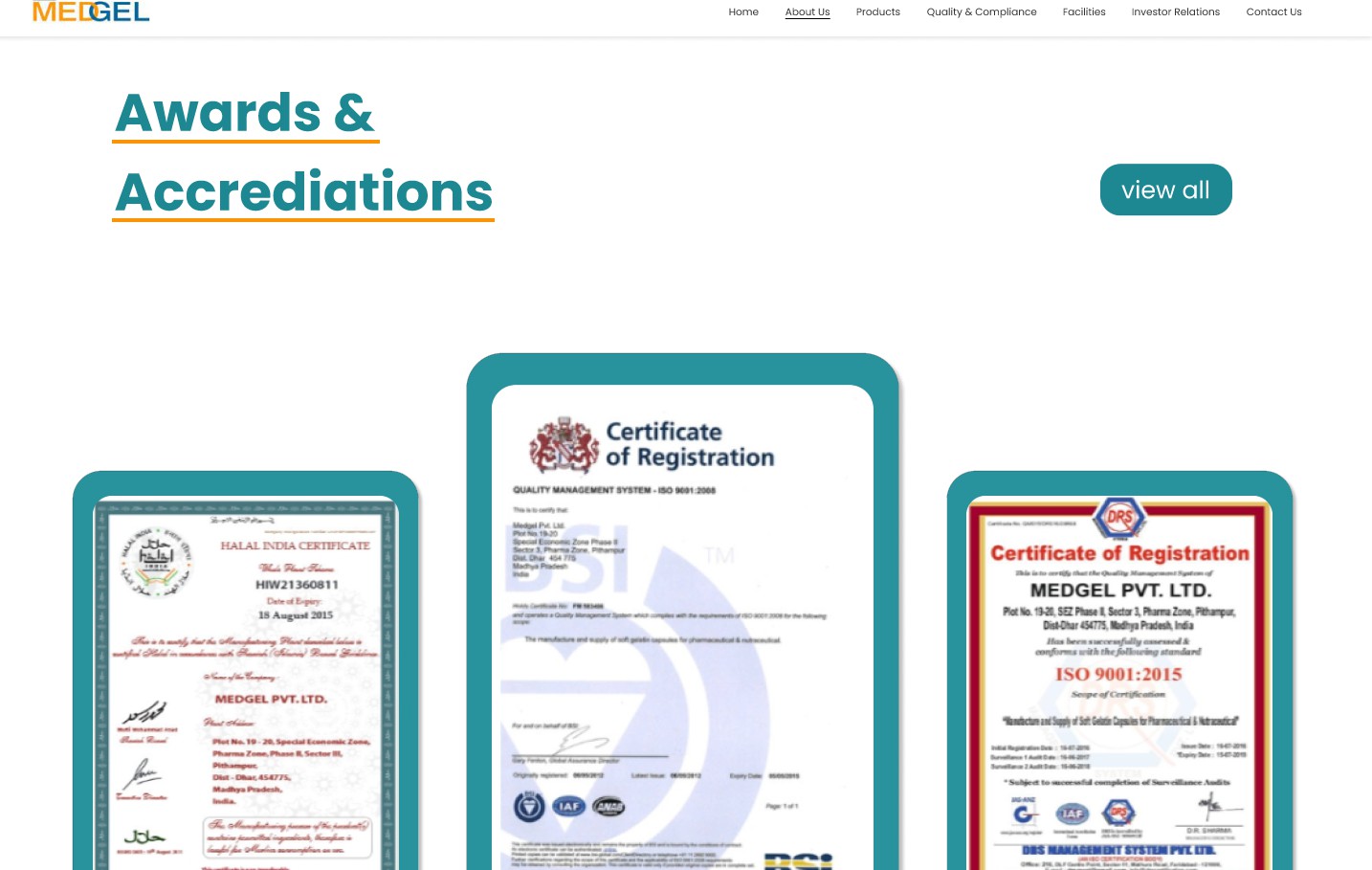
****



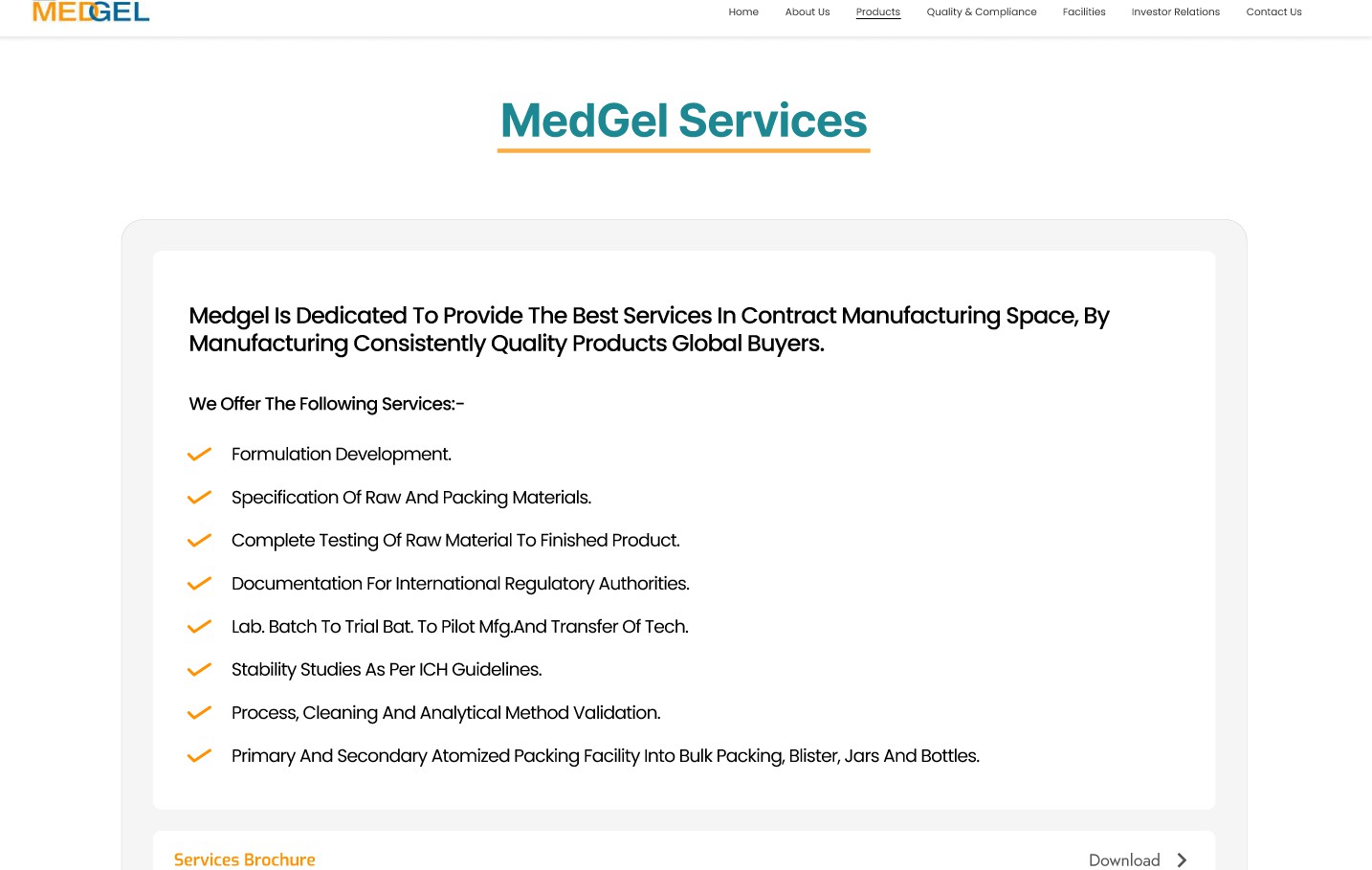
**Medgel Overview**

****

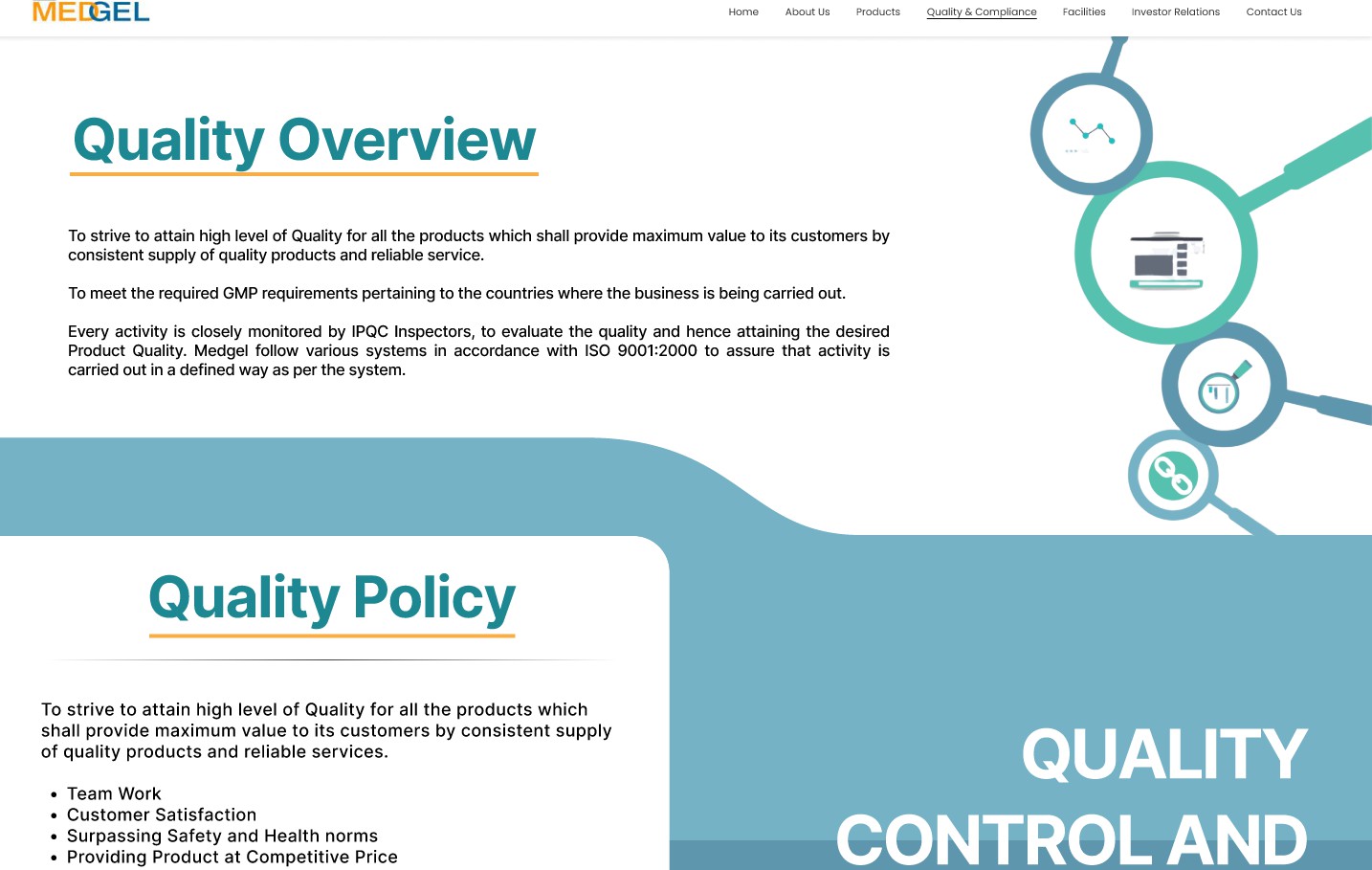
**Awards & Accrediations**

****

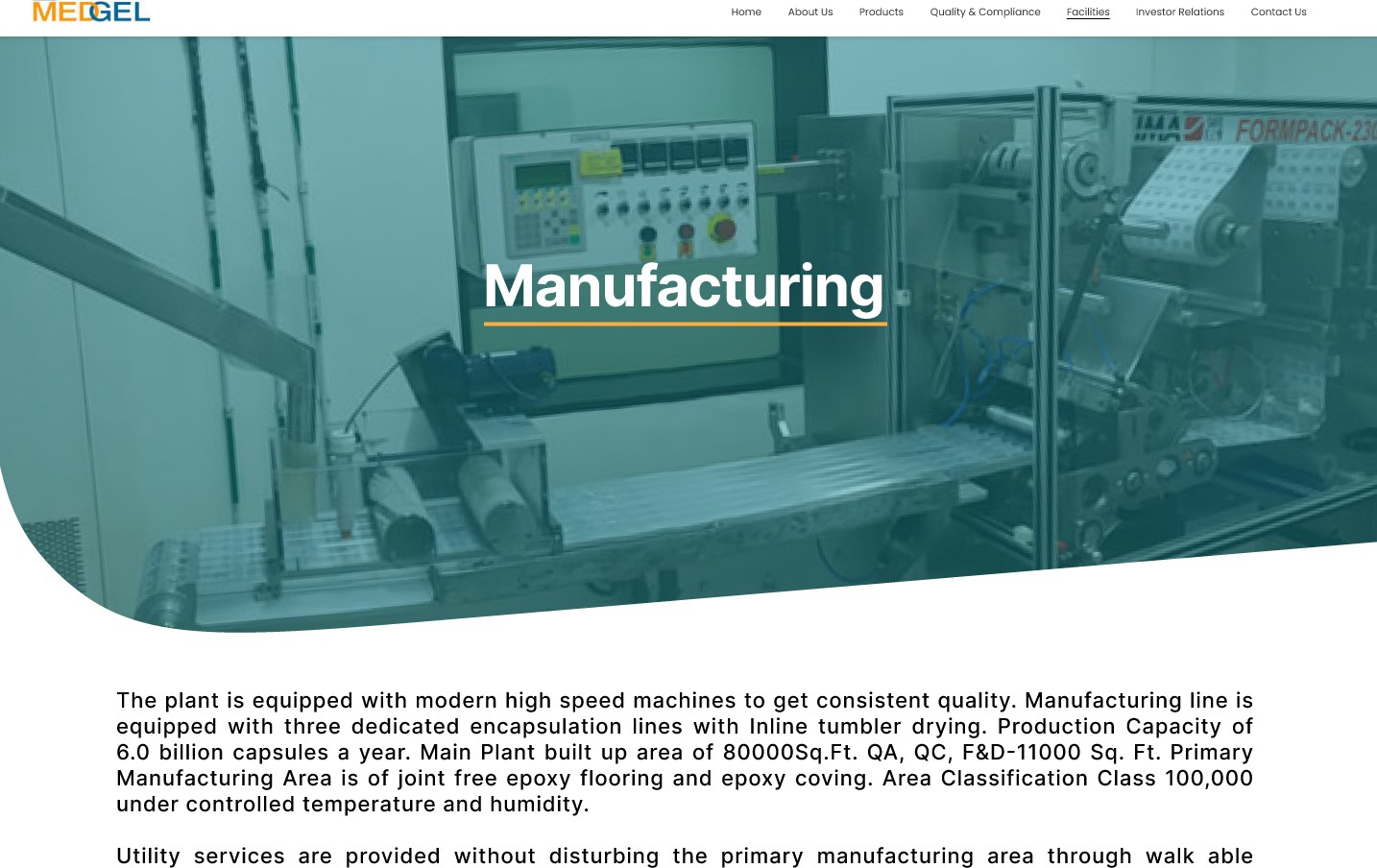
**Medgel Services**

****

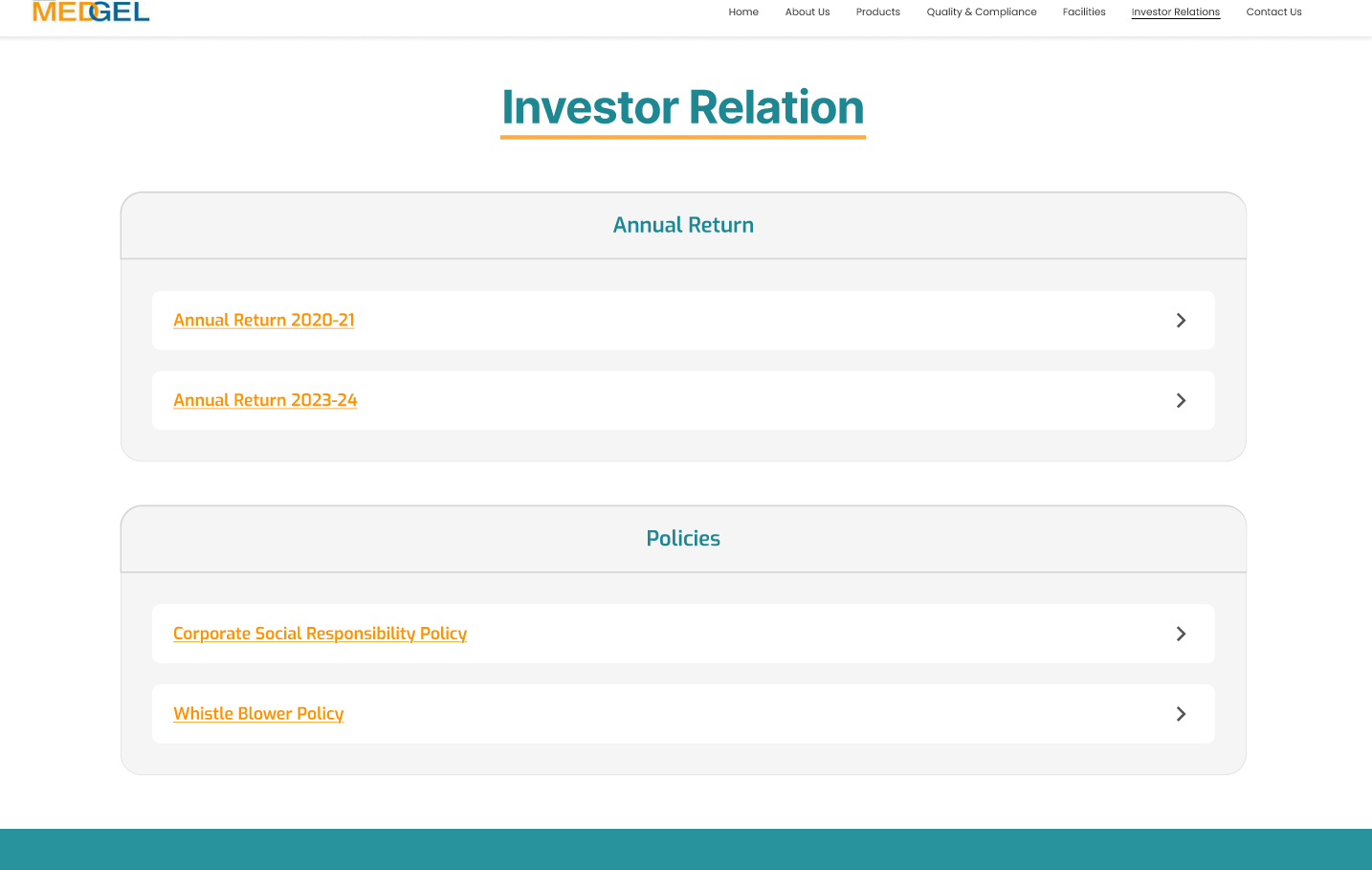
**Quality and Compliance**

****

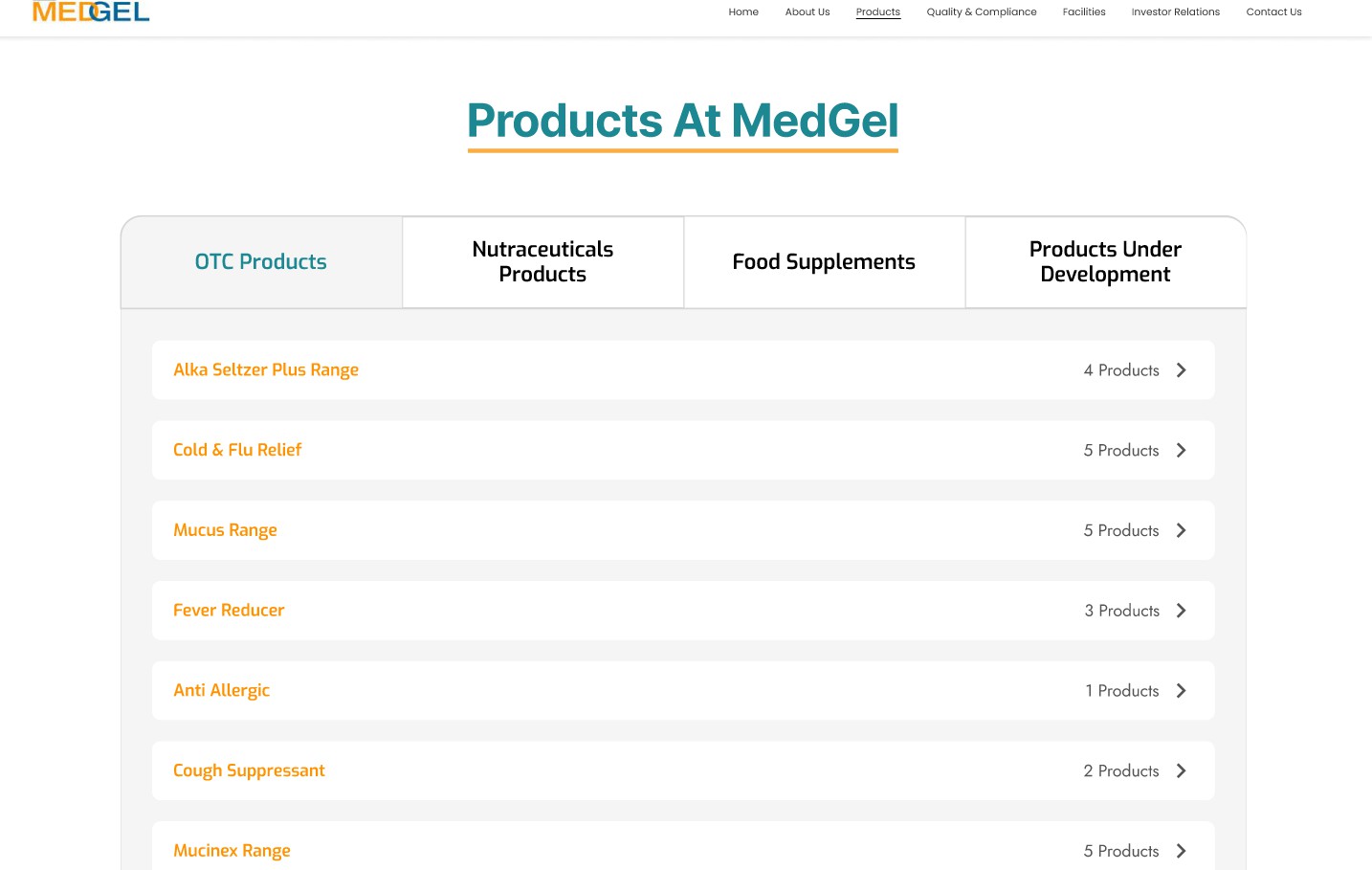
**Facilities**

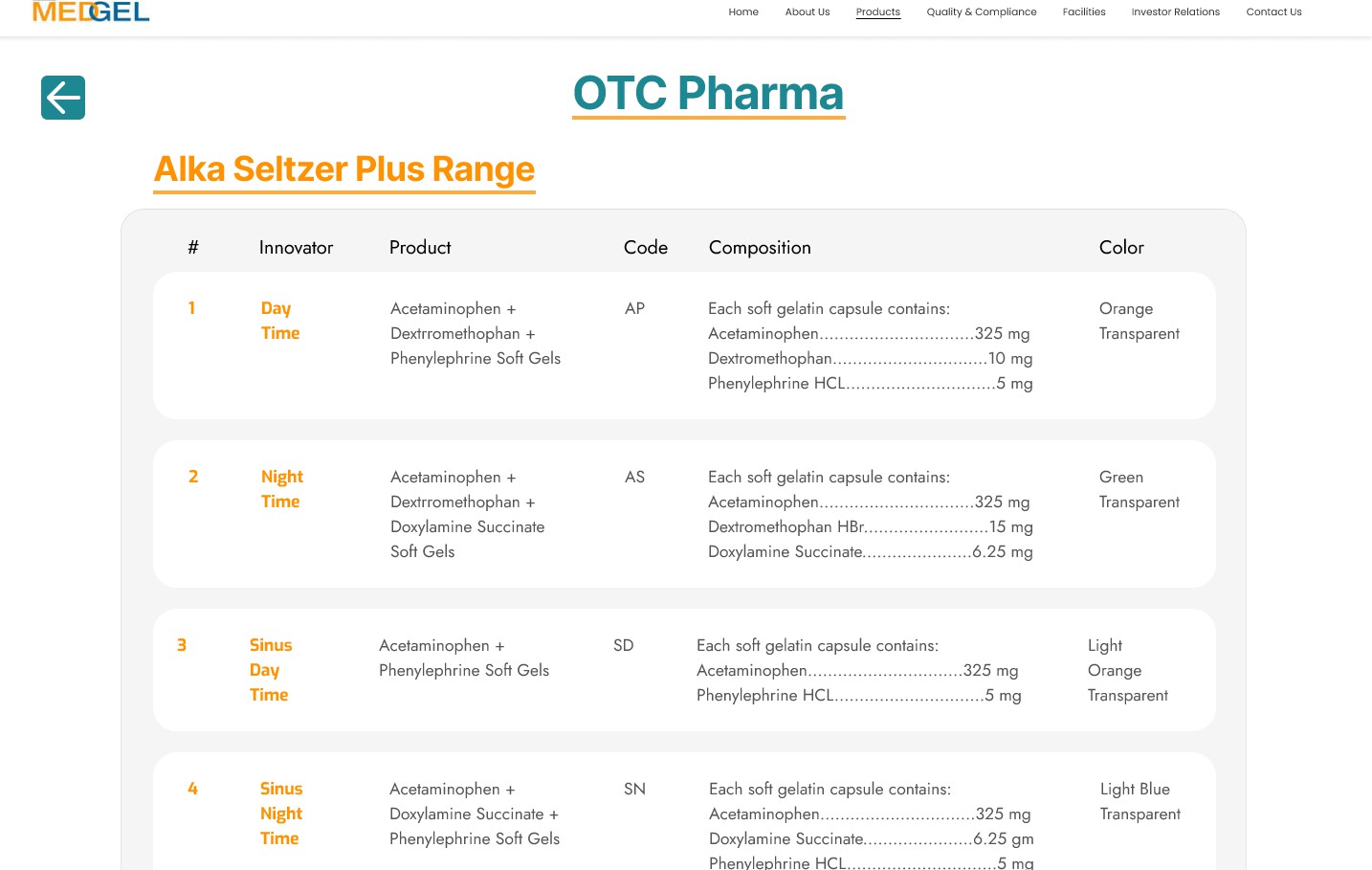
****

**Investor Relations**

****

**Products section**

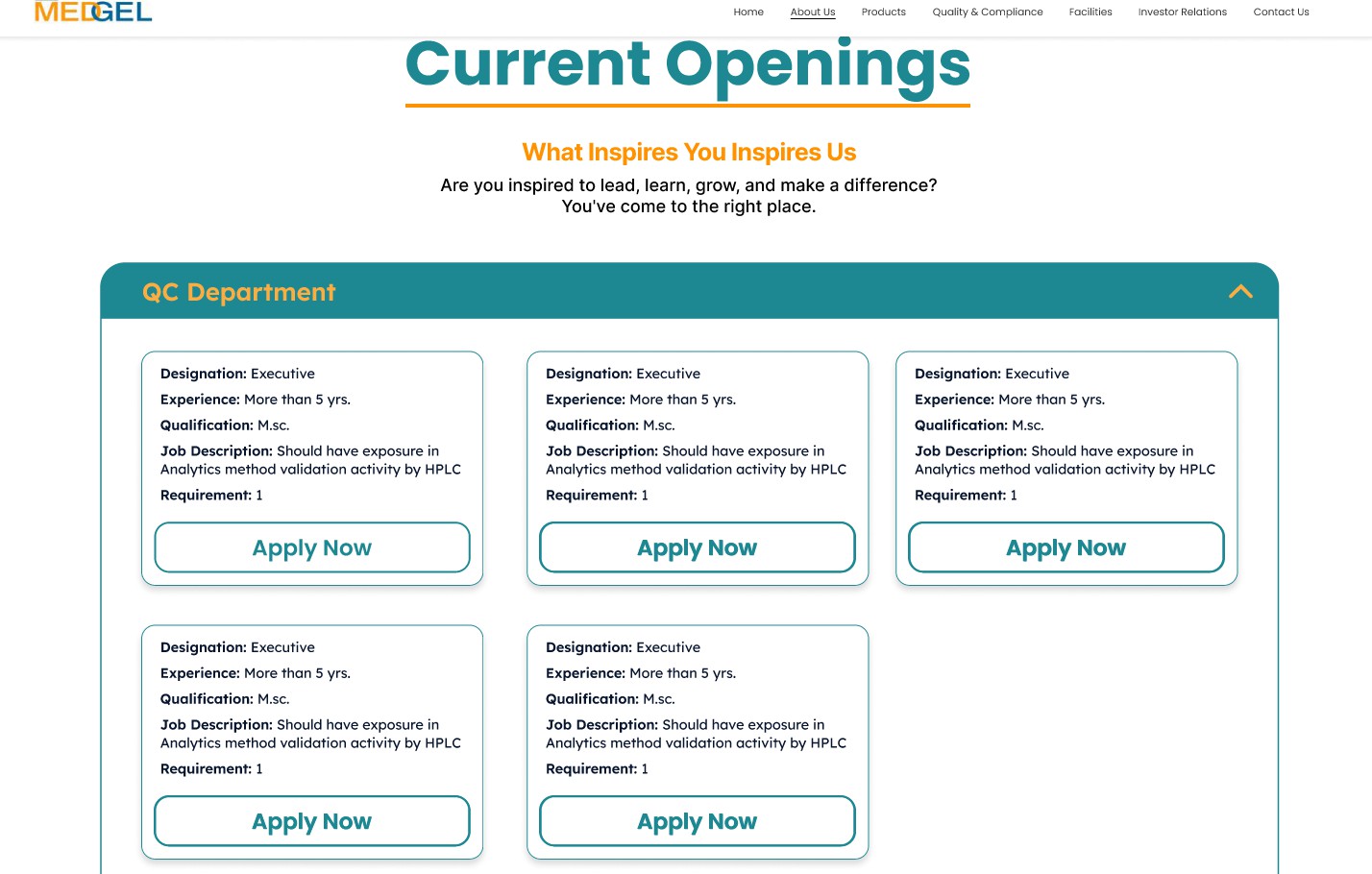
****



**Life at Medgel**

****

**Careers Section**

****

### Brief Description of Various Modules of the system

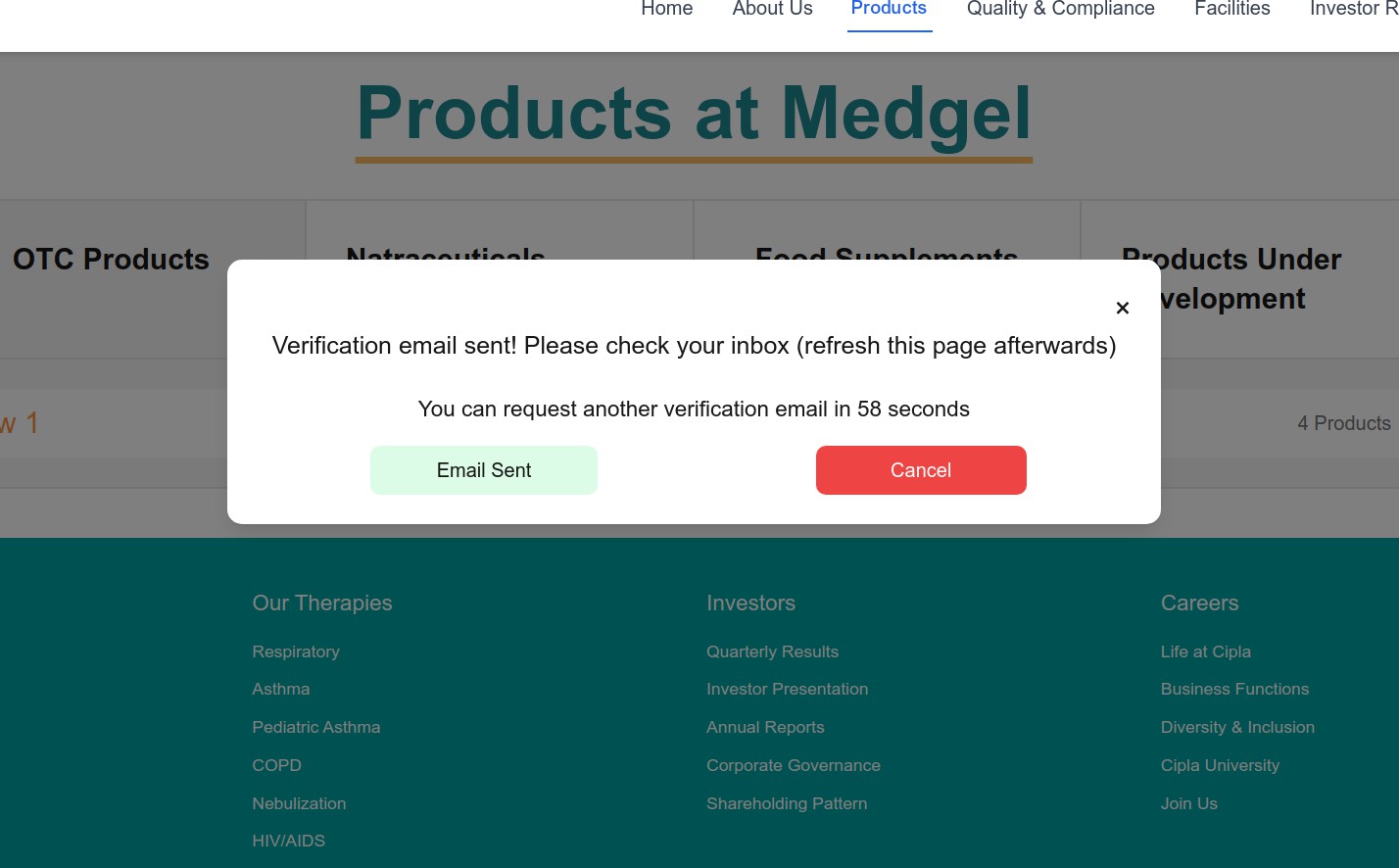
The Medgel Website with CMS comprises the following key modules:

* + - Authentication Module: Handles the login and logout functionality for administrators, ensuring secure access to the CMS backend using JWT for session management.
    - Content Management Module: Allows administrators to create, read, update, and delete (CRUD) various types of website content, including text, images, and rich media, through an intuitive editor.
    - Product Management Module: Enables administrators to manage the product catalog, including adding new products, updating product details (name, description, images, specifications, price), and controlling product visibility.
    - Media Library Module: Provides a centralized repository for uploading, organizing, and managing images and other media files, which can then be easily inserted into website content and product listings.
    - User Management Module (Admin): Allows super administrators to create and manage other administrator accounts, assigning different roles and permissions to control access to CMS features.
    - Front-end Rendering Module: Responsible for fetching data from the backend and database and rendering the dynamic content on the public-facing website in a user-friendly and responsive layout.
    - SEO Management Module: Provides tools within the content and product management sections to configure meta tags, descriptions, keywords, and URL slugs for improved search engine optimization.

### Snapshots of system with brief detail of each

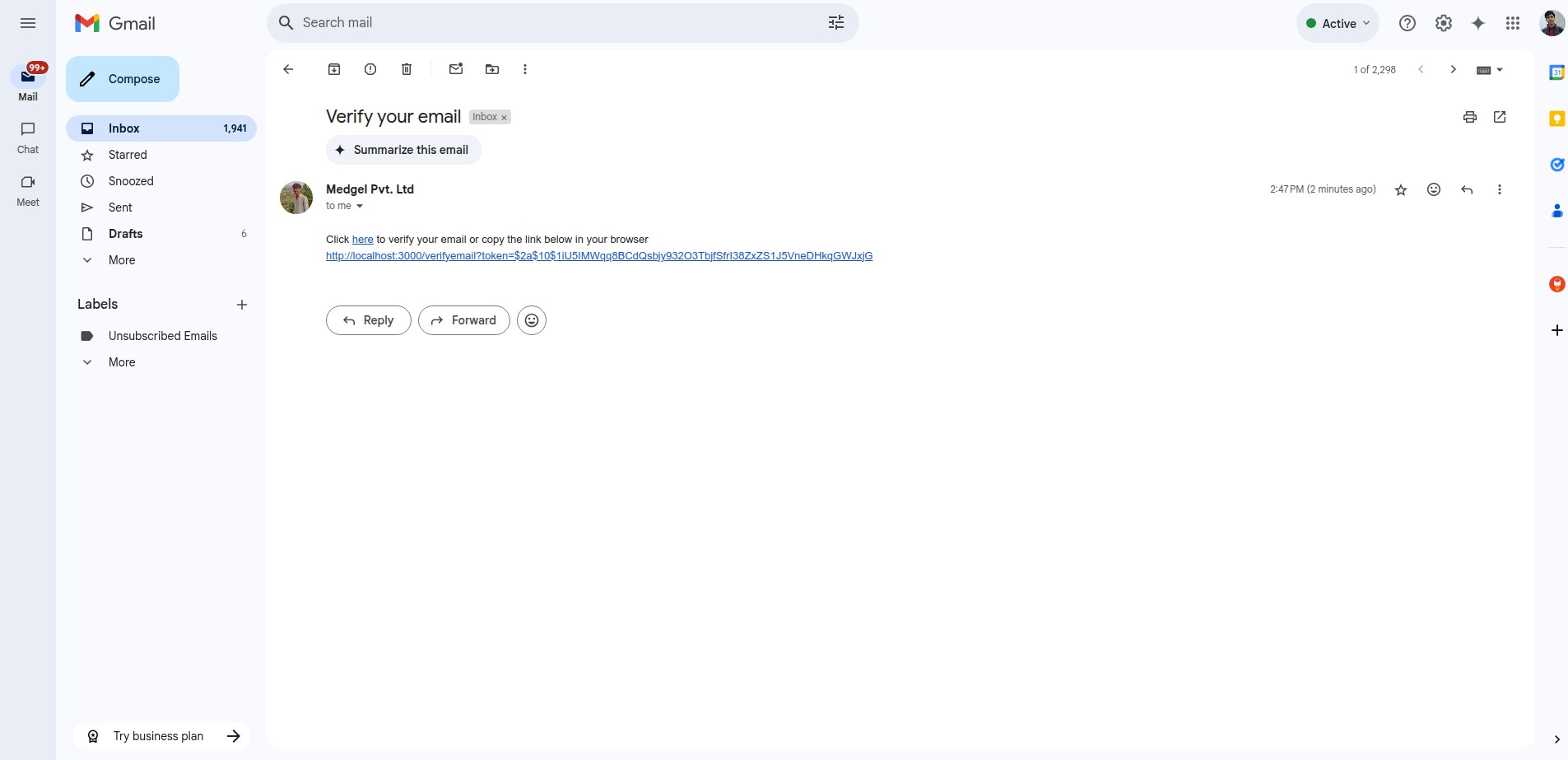
**User email verification dialog**

Allows the users to verify their emails, the website only products to the users who have verified their emails



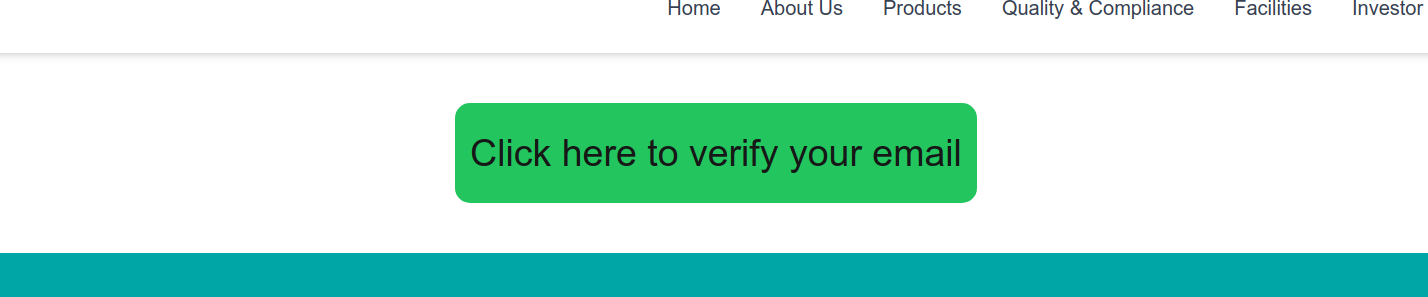
**User verification email**

The system sends a verification email to the user when they request it.



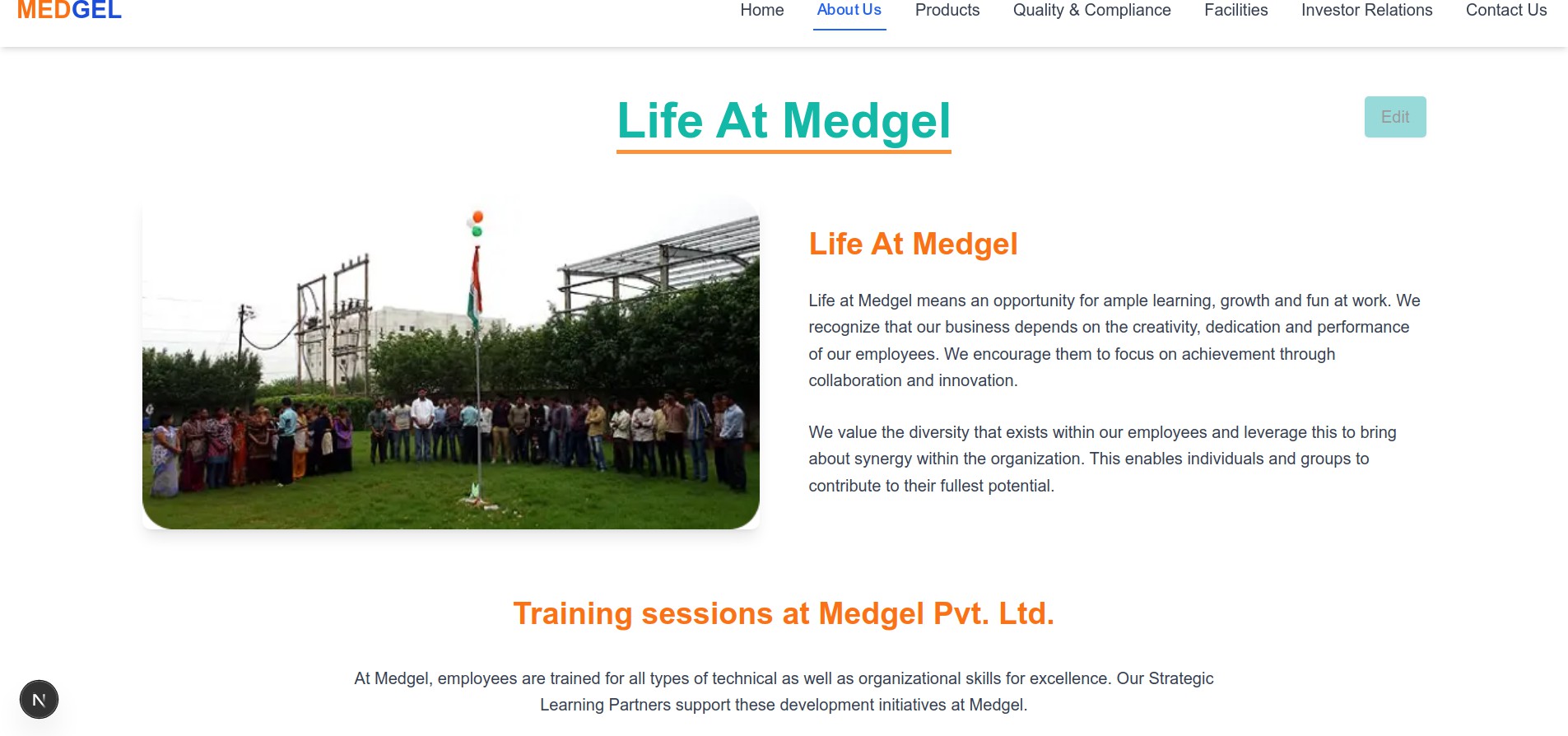
**User email verification page**

Allows the users to verify their email with the link received from in the email.



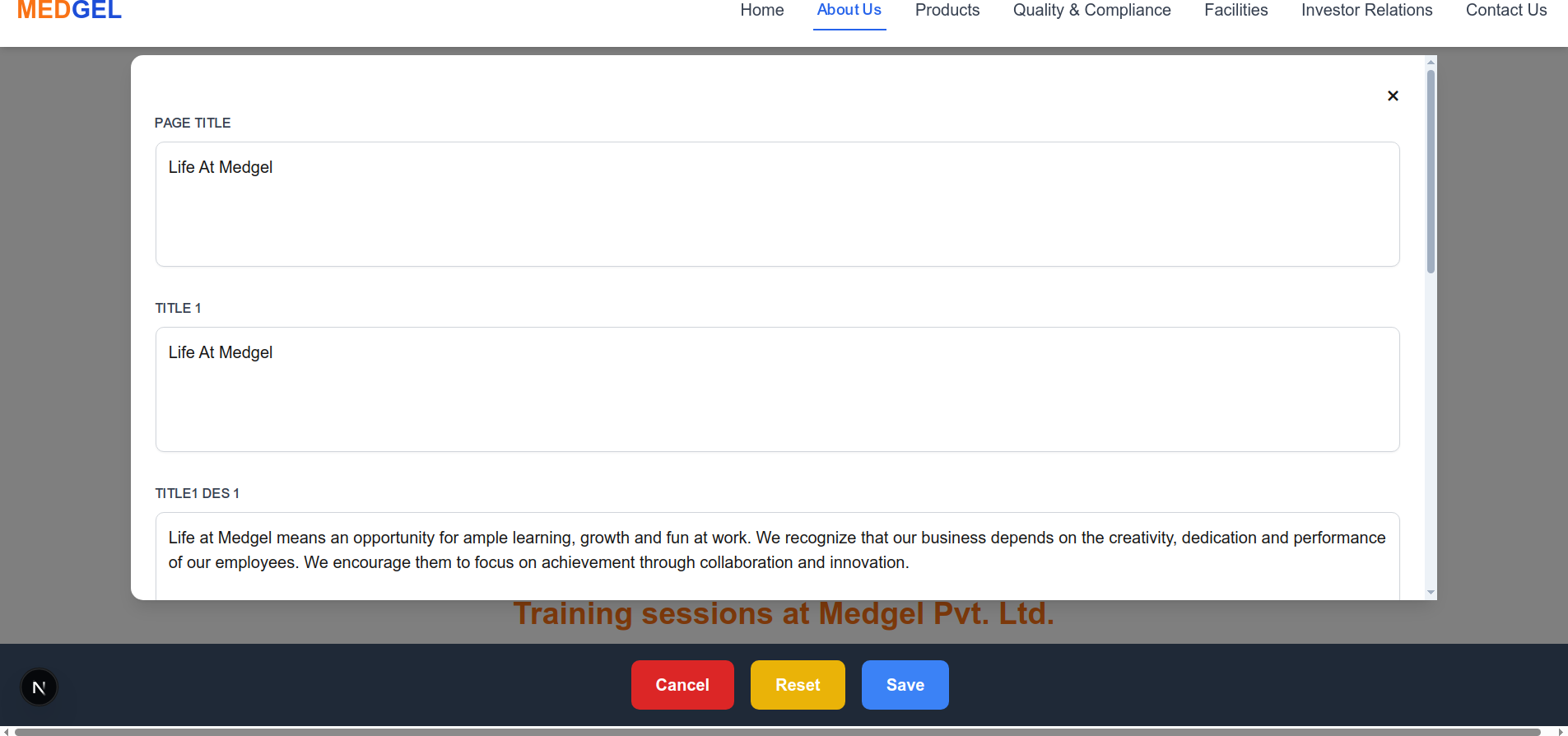
**Life at Medgel page with edit button for administrator**

The edit button makes the editing process smoother for the admin as they don’t have to navigate with a different UI then the end user would.



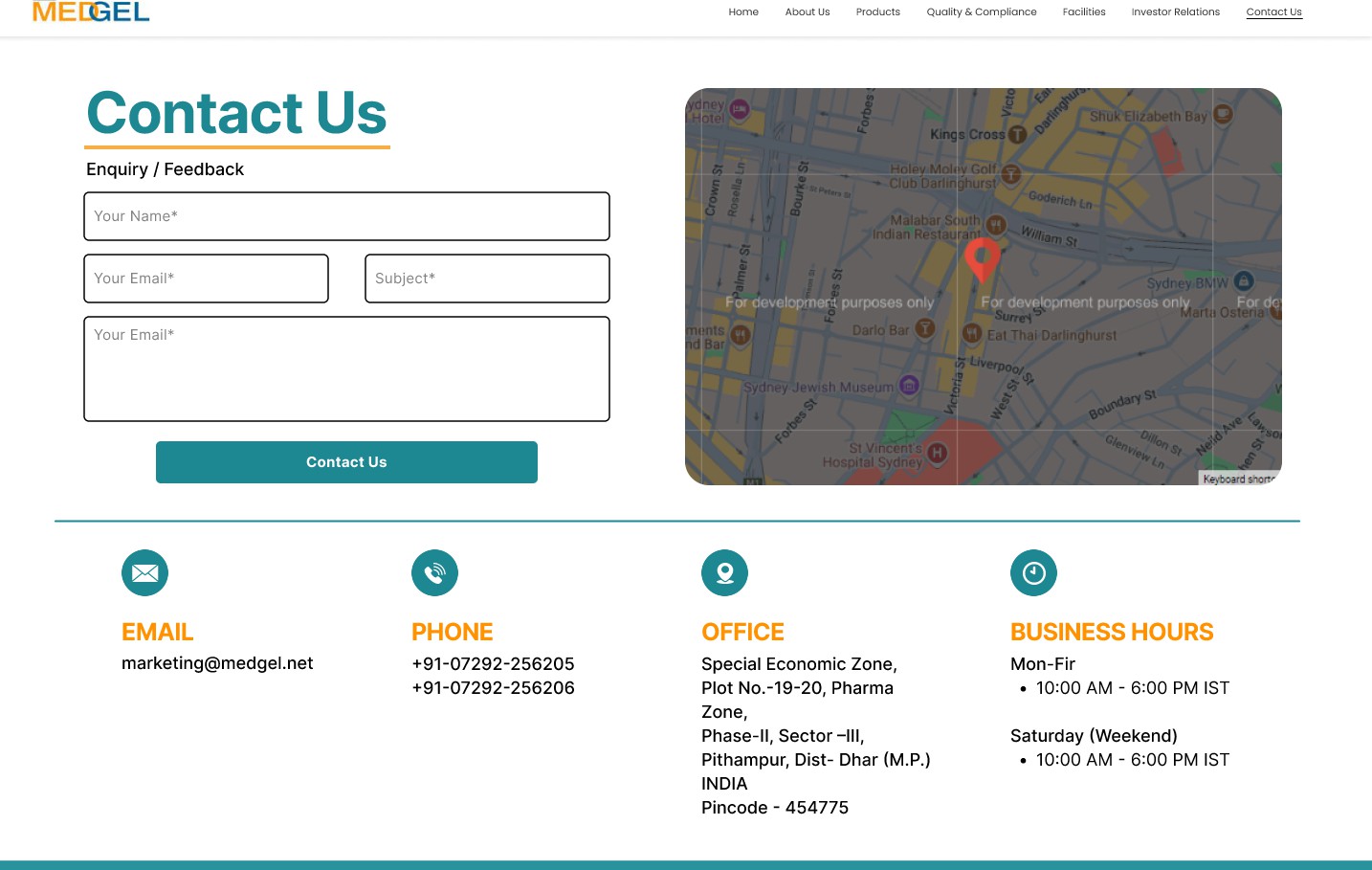
**Editing dialog for admins**

The editing dialog provides an intuitive and organized environment for content management. The navigation is straightforward, allowing administrators to easily access different sections for managing pages and content.



**Contact Page**

Allows the user to contact the company for queries or for any other business communication.



### Back Ends Representation (Database to be used)

The backend of the Medgel Website with CMS utilizes MongoDB as its primary database. Data is stored in collections as JSON-like documents, providing flexibility in schema design. The backend logic, implemented using Next.js API routes, interacts with the MongoDB database to perform CRUD operations on the data based on requests from the front-end.

Mongoose is used as an Object Data Modeling (ODM) library for MongoDB and provides schema definitions to structure the data and handle validation. This ensures data integrity and facilitates interaction with the database in a more organized manner.

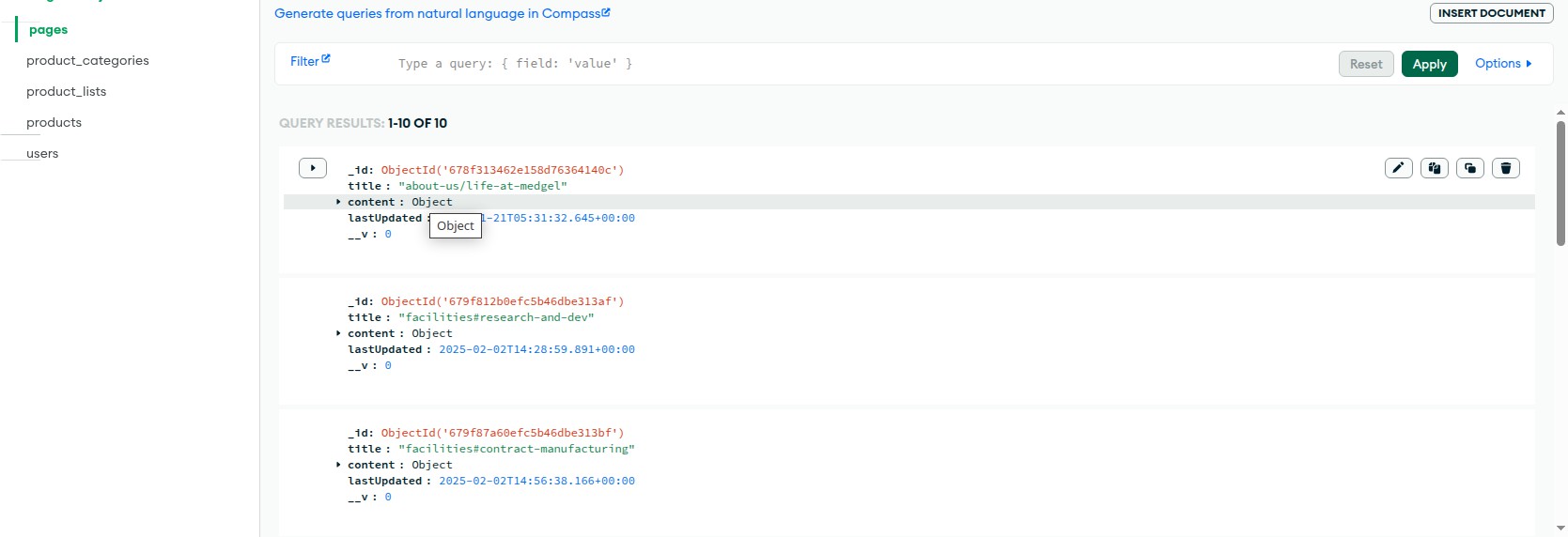
The API routes handle various functionalities, including:

* + - Authentication: Handling user login and generating JWTs upon successful authentication.
    - Content Management: Providing endpoints for fetching, creating, updating, and deleting website pages.
    - Product Management: Providing endpoints for fetching, creating, updating, and deleting product information.
    - Media Management: Handling the upload and retrieval of media files.

### Snapshots of Database Tables with brief description

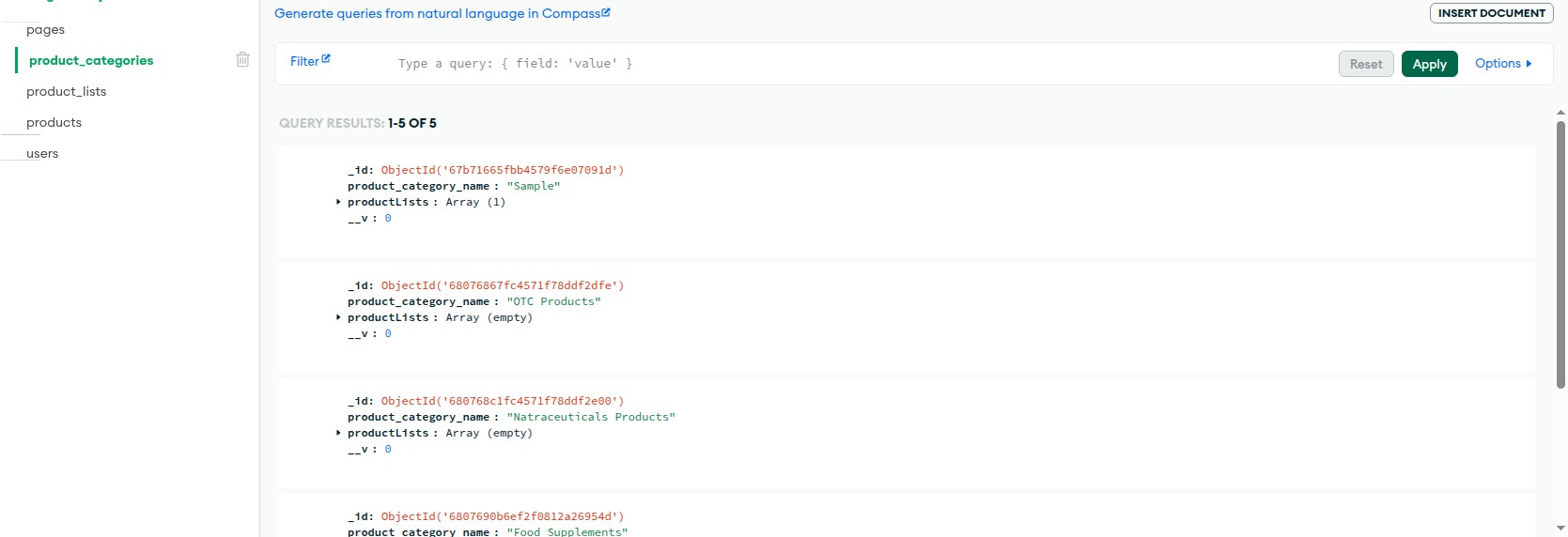
**Pages**

This section allows administrators to manage the website's content pages, with functionalities to edit their contents, images and other details.

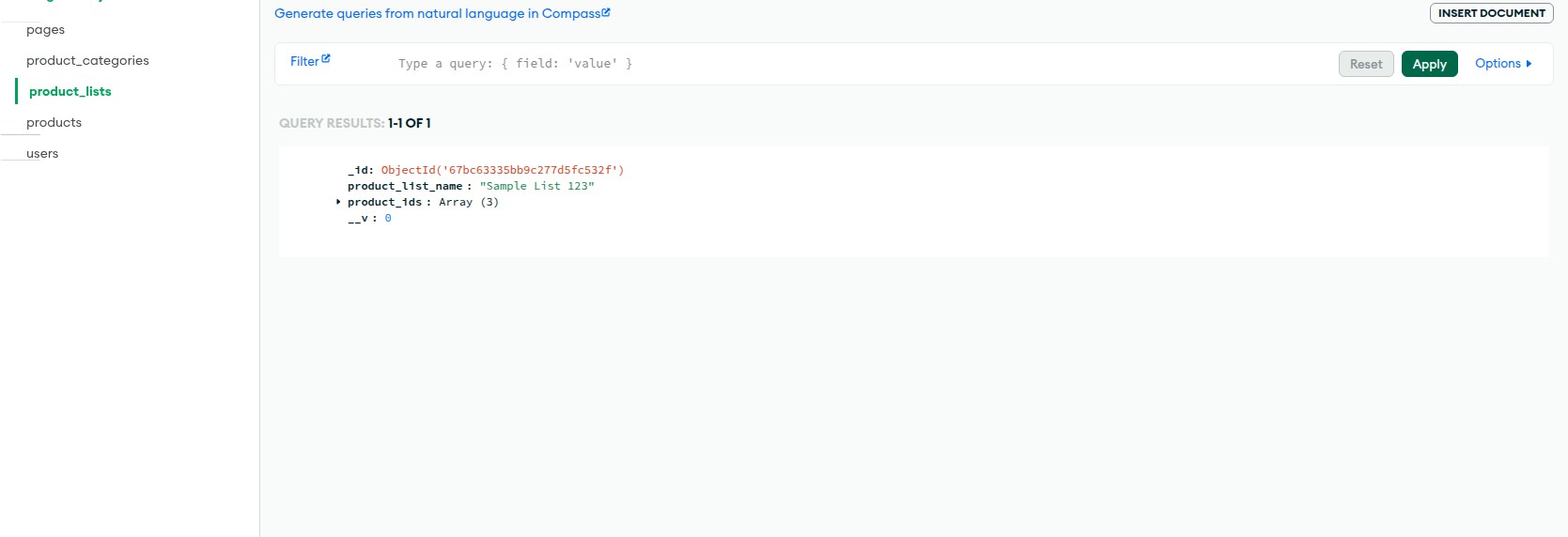


**Categories**

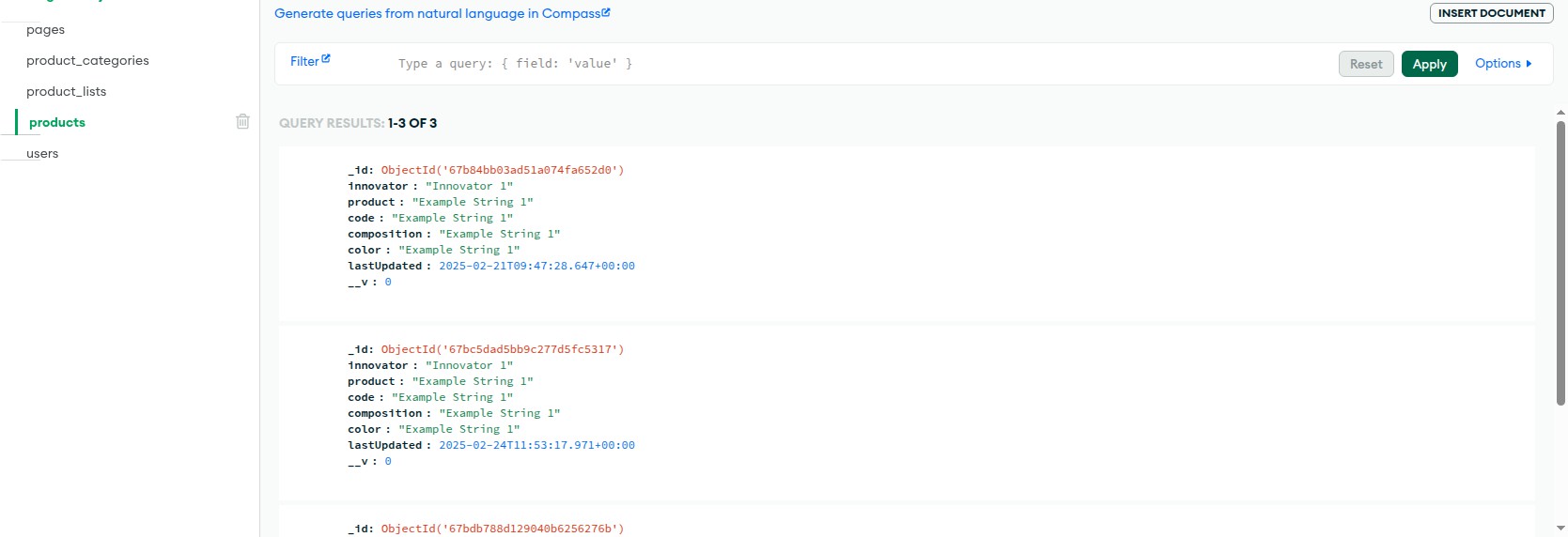
This section enables administrators to organize products or content into different categories for better navigation and organization on the website.



**Product lists**

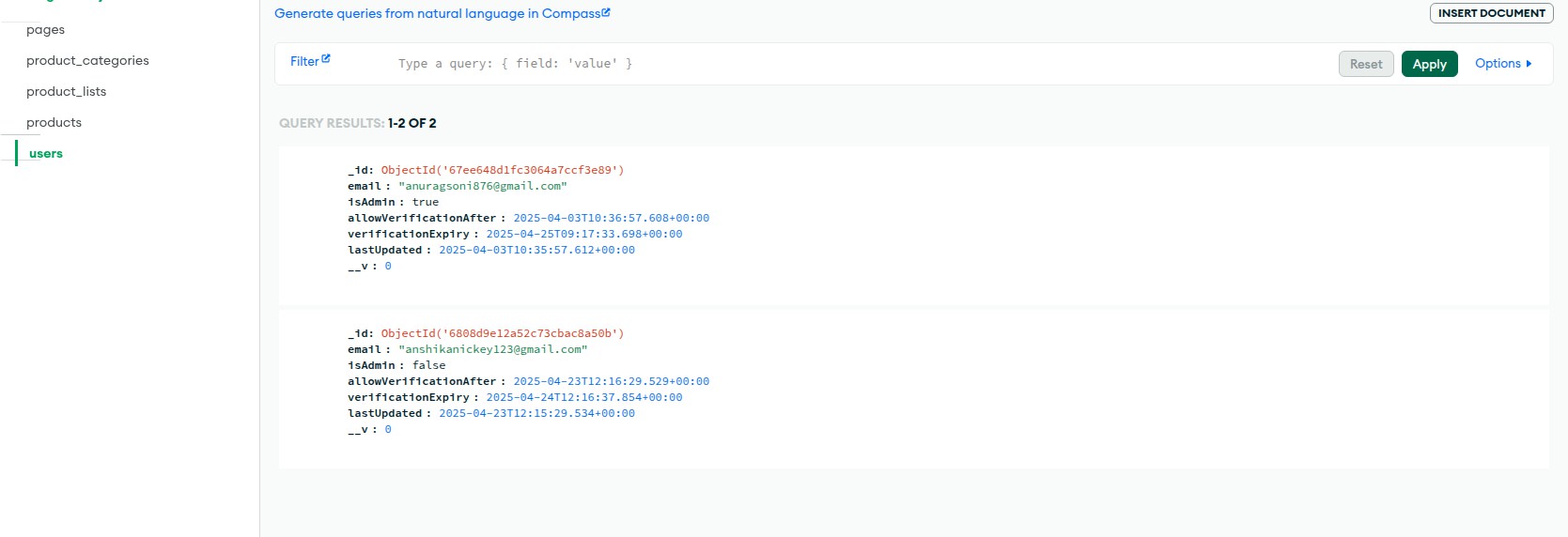
This section provides a way for administrators to manage curated lists or groupings of products, for better content organization.

**Products**

This section serves as the central area for managing individual product details, including descriptions, pricing, images, and other relevant attributes

**Users**

This section allows administrators to manage user accounts for those who have access to the administrative backend of the CMS



# Chapter 6 Summary and Conclusions

The Medgel Pvt. Ltd. Web Content Management System project aimed to address the limitations of static websites by developing a dynamic and user-friendly platform that empowers non-technical administrators to manage their website content effectively. By leveraging modern web technologies such as Next.js, Tailwind CSS, and MongoDB, the project has resulted in a functional CMS with features including content creation and editing, product management, user authentication, media library management, and basic SEO tools.

The implemented system provides a significant improvement over traditional static websites for Medgel Pvt. Ltd. Administrators can now update critical information, add new products, and manage their online presence in real-time without requiring coding expertise or relying on web developers for every change. This leads to increased efficiency, reduced turnaround times for content updates, and improved control over the website's information.

The modern and responsive user interface, built with Tailwind CSS, ensures a consistent and engaging experience for both website visitors and administrators across various devices. The use of Next.js for server-side rendering contributes to faster page load times and improved SEO performance.

The successful implementation and testing of the core modules demonstrate the feasibility and effectiveness of the proposed solution. The user acceptance testing (if conducted) would provide valuable feedback for further refinements.

In conclusion, Medgel Pvt. Ltd. Web Content Management System achieves its primary objectives of providing a dynamic, user-friendly, and efficient platform for managing pharmaceutical web content. This project enhances the company's digital presence, streamlines content update processes, and empowers the team to manage their online information independently.

# Chapter 7 Future scope

While the current implementation provides a solid foundation for Medgel Pvt. Ltd.'s web content management needs, several enhancements and future developments can further improve the platform:

* AI-Powered Content Recommendations: Integrating AI algorithms to suggest relevant content, improve product descriptions, and personalize user experiences.
* Multilingual Support: Expanding the CMS to support multiple languages, catering to a wider audience and potential international markets.
* E-Commerce Integration: Adding e-commerce functionalities to facilitate direct sales or B2B transactions through the website. This could include features like shopping carts, order management, and payment gateway integration.
* Chatbot for Customer Support: Implementing an AI-powered chatbot to provide instant customer support and answer common queries directly on the website.
* Analytics Dashboard: Developing a comprehensive analytics dashboard within the admin panel to provide insights into user engagement, website traffic, and product performance. This data can help inform content strategies and marketing efforts.
* Advanced SEO Tools: Implementing more advanced SEO features, such as automated schema markup generation, sitemap management, and SEO performance monitoring.
* Integration with other Business Systems: Connecting the CMS with other internal systems, such as CRM or inventory management software, to streamline data flow and improve overall business processes.
* Enhanced Security Features: Implementing more robust security measures, including two-factor authentication for administrators and regular security audits.
* Content Scheduling: Allowing administrators to schedule the publication of content for specific dates and times.
* Version Control for Content: Implementing a system to track changes made to content and allow for reverting to previous versions.

These potential future enhancements can further solidify the Medgel Website with CMS as a powerful and versatile tool for the company's digital presence and communication strategies.

# Bibliography

1. B. Boiko and Netlibrary, *Content Management Bible*. Indianapolis, IN: Wiley Pub., 2005.
2. Vercel, “Introduction to Vercel,” *Vercel Documentation*. [Online]. Available: <https://vercel.com/docs>
3. “JSON Web Tokens Introduction,” *jwt.io*, 2024. [Online]. Available: <https://jwt.io/introduction>
4. “Tailwind CSS - Rapidly build modern websites without ever leaving your HTML,”

*tailwindcss.com*, 2023. [Online]. Available: <https://tailwindcss.com/>

1. Meta Open Source, “React,” *react.dev*. [Online]. Available: <https://react.dev/>
2. “npm Documentation,” *docs.npmjs.com*. [Online]. Available: <https://docs.npmjs.com/>
3. “Node.js v20.2.0 Documentation,” *nodejs.org*. [Online]. Available: <https://nodejs.org/docs/latest/api/>
4. “Mongoose,” *mongoosejs.com*. [Online]. Available: <https://mongoosejs.com/docs/guides.html>
5. “Wix Help Center.” [Online]. Available: <https://support.wix.com/en/>
6. “Squarespace Help.” [Online]. Available: <https://support.squarespace.com/>
7. “The CMS Report.” [Online]. Available: <https://www.cmsreport.com/>
8. A. Rockley and C. Cooper, *Managing Enterprise Content: A Unified Content Strategy*. New Riders, 2009.
9. “Vite Documentation.” [Online]. Available: <https://vitejs.dev/guide/>
10. “Create React App Documentation.” [Online]. Available:

<https://create-react-app.dev/docs/getting-started/>

1. “React Router Documentation.” [Online]. Available: <https://reactrouter.com/en/main>
2. “Redux Documentation.” [Online]. Available: <https://redux.js.org/>
3. K. C. Dodds, “Kent C. Dodds' Blog.” [Online]. Available: <https://kentcdodds.com/>
4. A. Wathan, “Adam Wathan's Blog.” [Online]. Available: <https://adamwathan.me/>
5. A. Wathan and S. Schoger, *Refactoring UI*. [n.d.].
6. “MongoDB University.” [Online]. Available: <https://www.mongodb.com/university>
7. K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide*. [n.d.].
8. “Mongoose API Documentation.” [Online]. Available:

<https://mongoosejs.com/docs/api.html>

1. OWASP, “Open Web Application Security Project.” [Online]. Available: <https://owasp.org/>
2. “Auth0 Blog.” [Online]. Available: <https://auth0.com/blog/>
3. “Nielsen Norman Group Website.” [Online]. Available: <https://www.nngroup.com/>
4. J. Yablonski, “Laws of UX.” [Online]. Available: <https://lawsofux.com/>
5. Google, “Material Design Guidelines.” [Online]. Available: <https://material.io/design>
6. “Google Search Central.” [Online]. Available: <https://developers.google.com/search>
7. “Moz Blog.” [Online]. Available: <https://moz.com/blog>