**Applied Software Project Report**

By

Saideep Kankarla

**A Master’s Project Report submitted to Scaler Neovarsity - Woolf in partial fulfillment of the requirements for the degree of Master of Science in Computer Science**

May 10th, 2025



**Scaler Mentee Email ID :** saideep.kankarla@gmail.com

**Thesis Supervisor :** Naman Bhalla

**Date of Submission :** 10/05/2025

**Certification**

I confirm that I have overseen / reviewed this applied project and, in my judgment, it adheres to the appropriate standards of academic presentation. I believe it satisfactorily meets the criteria, in terms of both quality and breadth, to serve as an applied project report for the attainment of Master of Science in Computer Science degree. This applied project report has been submitted to Woolf and is deemed sufficient to fulfill the prerequisites for the Master of Science in Computer Science degree.

Naman Bhalla

…………………

Project Guide / Supervisor

**DECLARATION**

I confirm that this project report, submitted to fulfill the requirements for the Master of Science in Computer Science degree, completed by me from 22/04/2025 to 10/05/2025, is the result of my own individual endeavor. The Project has been made on my own under the guidance of my supervisor with proper acknowledgement and without plagiarism. Any contributions from external sources or individuals, including the use of AI tools, are appropriately acknowledged through citation. By making this declaration, I acknowledge that any violation of this statement constitutes academic misconduct. I understand that such misconduct may lead to expulsion from the program and/or disqualification from receiving the degree.

**Saideep Kankarla**

**<Signature of the Candidate> Date: 10th May 2025**

**ACKNOWLEDGMENT**

I am deeply grateful to my family for their unwavering love, support, and belief in me throughout this journey—your encouragement kept me grounded and motivated even during the toughest times. A heartfelt thank you to the incredible instructors at Scaler, whose guidance, patience, and expertise played a pivotal role in my learning and growth. I also want to acknowledge my peers, mentors, and every individual who inspired or motivated me along the way—whether through a kind word, a shared resource, or simply leading by example. This achievement is not mine alone, but a reflection of the collective support and inspiration I was fortunate to receive.

**Table of Contents**

[**List of Tables 6**](#_9nnr2lniv90f)

[**List of Figures 7**](#_ju1gc9w3iuai)

[**Applied Software Project 8**](#_b4cf8683b1wd)

[Abstract 8](#_60wow5ajoyaj)

[Project Description 8](#_1z5fx61h0cc)

[Requirement Gathering 9](#_joagy45av5k0)

[Payments Integration 9](#_a8rmw64l62m)

[Security 9](#_q2dc32x96flh)

[Deployment Flow 10](#_vzi61lbcv0ip)

[Technologies Used 10](#_z8c3joobm3pb)

[Conclusion 10](#_yiba2zhwwxul)

[**References 11**](#_g3fbztq1ko08)

## List of Tables

**(To be written sequentially as they appear in the text)**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page No.** |
| **1** | **Features set** | **10** |
| **2** | **System requirements** | **11** |
| **3** | **Payment gateway tech stack** | **12** |
| **4** | **Payment flow** | **15** |
| **5** | **Security** | **16** |
| **6** | **Brcypt** | **17** |
| **7** | **Security npm packages** | **18** |
| **8** | **Token verification** | **18** |
| **9** | **Deployment process** | **19** |
| **10** | **Deployment setup** | **20** |
| **11** | **Mern stack** | **21** |
| **12** | **Additional technologies** | **22** |

## List of Figures

**(List of Images, Graphs, Charts sequentially as they appear in the text)**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| **1** | **Audiio App – complete usecase diagram** |  |
| **2** |  |  |

## Applied Software Project

### Abstract **Audio Player by Saideep Kankarla** is a full-stack web application that allows users to stream music online. It supports both free and premium user tiers. Free users can access a curated set of free music, whereas premium users gain access to exclusive music content upon payment through Stripe. The application also includes an admin panel for managing users and music content. This project implements modern technologies such as React.js, Node.js, MongoDB, and Stripe for a robust and scalable architecture.

### Introduction

**Problem Statement:**

In the current digital age, access to music has become an integral part of everyday life. However, finding high-quality music for free or through a clean and user-friendly platform remains a challenge. This project aims to address this issue by creating a sleek and functional music player web application with free and premium access options.

**Objective:**

The objective of this project is to design and develop a full-stack web music player that allows users to stream music with a smooth user experience. The application supports user authentication, premium subscriptions via Stripe, and admin capabilities for managing content.

**Scope of the Project:**

This project supports two user types: Free and Premium. Free users can access only free music, while premium users, after payment, can access all available music. Admin users can manage albums, songs, and users, providing a robust backend management system.

### Project Description

The **Audio Player by Saideep Kankarla** is a modern, full-stack web-based music streaming application designed to offer users a smooth and interactive experience while accessing music content. Built using the latest frontend technologies like React.js (v19), Vite, SCSS, and Material UI, it allows users to explore, play, and manage music with ease.

The application supports two types of users: **Free Users** and **Premium Users**. Free users can access a limited library of free songs, whereas premium users, after a secure payment of $10 via **Stripe**, can access both free and premium songs. User authentication is handled securely using **bcryptjs** for password hashing and **JSON Web Tokens (JWT)** for session handling and API protection.

An **Admin Panel** is built into the same app, with restricted access. Admin users can view all users, payment records, manage music content (add/remove albums and songs), and perform administrative actions via an easy-to-use dashboard. All content management operations are backed by **MongoDB** and accessed using **Mongoose ORM**.

The backend is developed using **Node.js** and deployed on the **Render** platform (free-tier). The application emphasizes scalability, modular architecture, and secure API integrations.

**Key Features**:

* Secure registration and login system.
* Role-based content access (Free vs. Premium).
* Stripe payment gateway integration.
* Admin dashboard for content and user management.
* Fully responsive UI with Material UI components.
* JWT-based API authorization and secure password encryption.
* Audio player functionality for high-quality music playback.

This project demonstrates the practical application of full-stack development skills, integration of third-party services like Stripe, and strong focus on user experience and security.

Use the below format as reference when including a picture



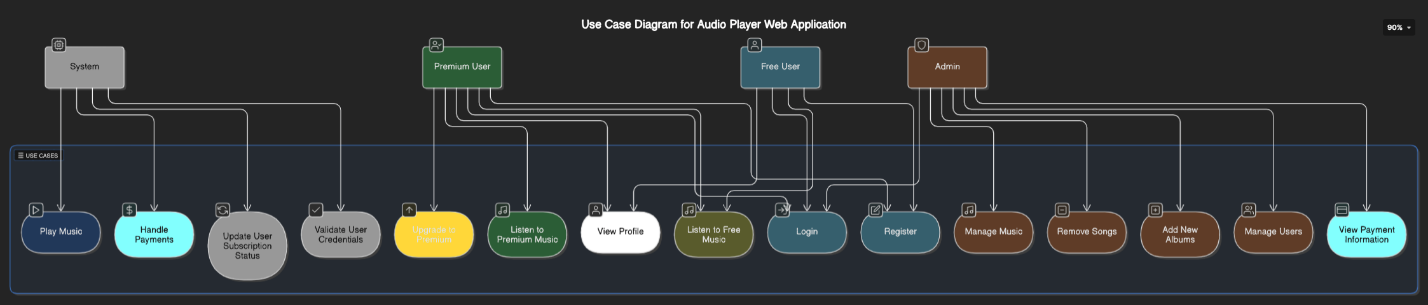
**Figure 1.1**: Project Development Process (Figure captions go below figures.)

### Requirement Gathering

**Functional Requirements**

* Users should be able to register and log in.
* Free users can stream only free music albums.
* Premium users can stream both free and premium albums.
* Users can view there profile details.
* Users should be able to upgrade to premium via Stripe payment.
* Admin can:
  + View all users and payments.
  + Add new albums and songs.
  + Delete users, albums, and songs.
* The audio player should support play, pause, re-play, seek and volume features.

**Non-Functional Requirements**

* Secure password storage using bcrypt with salting and hashing.
* API authorization using JWT tokens.
* Role-based access control (Free, Premium, Admin).
* Responsive design compatible with desktop and mobile devices.
* Smooth and fast user experience using React and Vite.
* Error handling and validation for all forms and API interactions.
* Audiio App Complete Usecase

**Table 1: Audiio App Features set**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Feature** | **User Role** | **Description** |
| 1 | Register | Free, Premium | Allows users to create an account by entering email, password, and other info. |
| 2 | Login | Free, Premium, Admin | Enables users to log in with secure authentication using JWT. |
| 3 | Listen to Free Music | Free, Premium | Users can play music from the free music library. |
| 4 | Listen to Premium Music | Premium | Premium users can access and play premium music in addition to free music. |
| 5 | View Profile | Free, Premium | Displays user information and subscription status. |
| 6 | Upgrade to Premium | Free | Free users can purchase a premium plan using Stripe payment gateway. |
| 7 | Admin Dashboard | Admin | Provides access to admin-specific controls and data. |
| 8 | Manage Users | Admin | Admin can view all users and delete unwanted accounts. |
| 9 | Manage Albums & Songs | Admin | Admin can add, update, or delete albums and songs in the database. |
| 10 | View Payment Information | Admin | Admin can view all user payment records. |
| 11 | Secure Authentication | System | Passwords are hashed with bcryptjs; JWT is used for token-based login. |
| 12 | Payment Processing | System | Stripe handles secure transactions for premium subscriptions. |
| 13 | Role-Based Access Control | System | Access to music and features is determined by user role (Free, Premium, Admin). |
| 14 | Audio Playback Controls | All Users | Built-in audio player with play/pause/Volume/Seek and skip functionality. |

### System Requirements

Table 2 : System Requirements

|  |  |
| --- | --- |
| **Category** | **Requirement** |
| **Hardware Requirements** | A PC or laptop with minimum 4 GB RAM |
|  | Modern web browser |
|  | Internet connection |
| **Software Requirements** | Operating System: Windows / Linux / macOS |
|  | Node.js, MongoDB |
|  | Code Editor (e.g., Visual Studio Code) |
|  | Web Browser (e.g., Google Chrome, Mozilla Firefox) |

### Modules Description

**1. User Module:**

Handles user registration, login, profile management, and distinguishing between free and premium users.

**2. Authentication Module:**

Manages secure user login using bcryptjs for password hashing and JWT for generating and validating tokens.

**3. Payment Module:**

Enables users to purchase a premium subscription using the Stripe payment gateway. Handles checkout flow and redirects.

**4. Admin Module:**

Allows an admin user to manage the platform including viewing users, payments, albums, and deleting or modifying records.

**5. Music Player Module:**

Displays the audio player interface and enables users to browse and play songs. Premium songs are restricted to paid users only.

### Implementation Details

**APIs Overview:**

REST APIs are used to interact with the backend services. API routes are protected using JWT-based middleware.

**JWT Flow:**

On successful login, a JWT is issued and stored in localStorage. For each API call, the token is sent as a Bearer token and verified at the backend.

**Payment Integration:**

The Stripe API is used to create checkout sessions. On successful payment, the user’s premium status is updated in the database.

**MongoDB & Mongoose Schemas:**

Three main schemas are used: User, Album, and Payment. User schema includes encrypted passwords and a flag for premium status.

### Payments Integration

To enable premium access in the **Audio Player by Saideep Kankarla**, a secure and reliable payment gateway is essential. This project uses **Stripe**, a widely adopted platform for handling online payments. Stripe provides APIs and frontend libraries that allow seamless payment integration with minimal security risk.

**Table3 : Payment gateway tech stack**

| **Technology** | **Purpose** |
| --- | --- |
| **Stripe API** | Manages secure transactions and payment sessions |
| **@stripe/stripe-js** | Loads Stripe scripts on the client side |
| **@stripe/react-stripe-js** | Provides React components for UI checkout flow |
| **Axios** | Sends API requests to create Stripe sessions |
| **JWT** | Ensures secure access to protected payment routes |

**Integration Flow**

1. **Upgrade Button on Profile Page**
   * Free users see an **"Upgrade to Premium"** button on their profile page.
   * Clicking the button triggers a client-side request to initiate a Stripe Checkout session.
2. **Backend API Request**
   * A POST request is sent to a protected backend route /api/create-checkout-session using **Axios**.
   * The server verifies the user's JWT and role.
3. **Stripe Checkout Session Creation**
   * The server creates a Stripe session using the **Stripe Secret Key** and sends back the sessionId.
4. **Redirect to Stripe Checkout**
   * The frontend uses useStripe().redirectToCheckout({ sessionId }) to take the user to the Stripe payment page.
5. **Payment Completion**
   * On success/failure, Stripe redirects the user to predefined URLs:
     + /success for successful payments
     + /cancel for failed or canceled payments
6. **Post-Payment Handling**
   * On the /success page, a backend call verifies payment status and **updates the user’s role to 'premium'** in the MongoDB database.

**Security Considerations**

* **JWT-based route protection** ensures that only authenticated users can initiate or complete payments.
* **Stripe handles sensitive information** like card numbers and billing details—none of it is stored or processed by the application.
* All communication is conducted over HTTPS (enforced by Stripe and your frontend host like Render).

**Real-Time Updates (Optional Enhancement)**

To make payment handling more dynamic, you can:

* Use **Stripe Webhooks** to listen for events like payment\_intent.succeeded or checkout.session.completed and automatically update the database.
* This prevents users from tampering with post-payment redirects and ensures integrity.

**Testing Stripe (Development)**

* Stripe provides **test card numbers** for simulating various payment scenarios.
* Test credentials:
  + Card number: 4242 4242 4242 4242
  + Expiry: Any future date
  + CVC: Any 3 digits
  + ZIP: Any 5-digit number

**How Payment Gateways Work (Stripe Focused)**

A **payment gateway** is a service that securely authorizes and processes payments made online via credit or debit cards, UPI, wallets, or net banking. For web applications like your **Audio Player**, a payment gateway handles the sensitive task of communicating between:

* The **user’s browser**
* The **web server**
* The **bank/card provider**
* The **payment processor**

In this project, we used **Stripe** as the payment gateway to collect and process $10 payments for premium subscriptions.

**Payment Flow (Stripe Checkout)**

Here’s the **standard payment flow** using **Stripe Checkout** with React and Node.js:  
Table 4: Payment flow

| **Step** | **Description** |
| --- | --- |
| 1. User Action | User clicks "Upgrade to Premium" on the React frontend. |
| 2. API Request | A request is sent to the backend (/create-checkout-session) via Axios. |
| 3. Session Creation | The backend uses Stripe's SDK to create a checkout session with pricing details. |
| 4. Redirect | The frontend redirects the user to the Stripe-hosted checkout page. |
| 5. Payment | User enters card details, Stripe processes the payment securely. |
| 6. Response | Stripe redirects to a success or cancel URL defined in the backend. |
| 7. Subscription Update | On success, the backend updates the user’s status in the database. |

**Security & PCI DSS Compliance**

Stripe ensures all data handling and processing is secure and compliant with PCI DSS (Payment Card Industry Data Security Standard), a set of security guidelines for managing card data.

**Key Security Features:**Table 5: Security

| **Feature** | **Purpose** |
| --- | --- |
| HTTPS Encryption | All requests are encrypted, including communication between client and Stripe. |
| Tokenization | Card data is never stored on your server. Instead, Stripe generates tokens representing payment methods. |
| PCI DSS Compliance | Stripe is certified as a PCI Level 1 Service Provider (the highest level). |
| Webhook Validation | All events (like payment success) can be validated via signed webhooks. |

**Important:** Since Stripe handles card data on their hosted page (Checkout), your app does not fall under strict PCI DSS scope, making integration easier and safer.

**React-Specific Stripe Integration**<https://docs.stripe.com/sdks/stripejs-react>

### Security

Security is a critical component of any web application, especially when handling sensitive data such as user credentials and payment details. The **Audio Player by Saideep Kankarla** incorporates multiple security layers using trusted Node.js packages and modern best practices to ensure data confidentiality, integrity, and access control.

**a. Password Security with bcrypt**

**What is bcrypt?**

**bcrypt** is a widely-used hashing algorithm designed for secure password storage. Unlike regular encryption (which can be decrypted), **bcrypt performs one-way hashing**, making it practically irreversible. It is ideal for protecting passwords in databases.

**How bcrypt Works**

* When a user registers, the password is:
  1. **Salted** – a unique, random string is added to the password.
  2. **Hashed** – the salted password is then converted into a fixed-length, scrambled string using the bcrypt algorithm.
* During login, the entered password is re-hashed with the stored salt and compared to the stored hash.

**What is Salt?**

A **salt** is a random value added to passwords before hashing to ensure that identical passwords result in different hashes. This prevents:

* Rainbow table attacks
* Detection of reused passwords

**Benefits of Using bcrypt  
table6 : brycpt**

| **Feature** | **Benefit** |
| --- | --- |
| Adaptive hashing | Can increase computation cost as hardware improves |
| Built-in salting | Adds an extra layer of randomness |
| Irreversible | Enhances protection against brute-force and dictionary attacks |

**b. Node Packages for Added Security**

In addition to bcrypt, several Node.js packages have been integrated to harden the backend server:

Table7: security npm packages

| **Package** | **Purpose** | **Usage in App** |
| --- | --- | --- |
| **jsonwebtoken (JWT)** | Authenticates users via secure tokens | Used for login sessions and API protection. Tokens are stored in localStorage and sent in headers for verification. |
| **helmet** | Secures HTTP headers | Hides server details, prevents clickjacking, and enforces HTTPS headers. |
| **cors** | Cross-Origin Resource Sharing | Controls which origins can access the backend API to prevent unauthorized frontend interactions. |
| **dotenv** | Secures environment variables | Keeps sensitive data like API keys and DB credentials out of source code. |
| **multer** | Validates file uploads | Handles safe media uploads by validating file types and size limits. |

**JWT (JSON Web Token) Authentication**

JWT is used for **stateless authentication** between the client and server. Once a user logs in, the server generates a token containing encoded user information and sends it to the client. The client includes this token in the header of subsequent requests.

**Token Verification Flow**

1. Login → Generate JWT
2. Store JWT in localStorage
3. Use JWT as Bearer token in API headers
4. Backend verifies token using secret key

Table8: Token verification table

| **Feature** | **Implementation** |
| --- | --- |
| Password Protection | bcrypt with salt and hash |
| Secure HTTP Headers | helmet middleware |
| Token-based Auth | JWT in headers with expiration |
| File Upload Security | multer for sanitization |
| Environment Management | dotenv for key protection |
| Cross-Origin Policy | cors to restrict domain access |

### Deployment Flow

Deploying a full-stack web application requires proper configuration of the frontend, backend, and server environments to work together seamlessly in a production setting. The **Audio Player by Saideep Kankarla** follows a straightforward yet effective deployment process using GitHub and Render.

**Deployment Process Overview**

Table 9 : deployment process

| **Step** | **Description** |
| --- | --- |
| **1. Version Control (GitHub)** | The complete project source code, including client and server code, is maintained in a **GitHub** repository. This allows version tracking, collaboration, and CI/CD integration. |
| **2. Hosting Platform (Render)** | Both the **React frontend** and **Node.js backend** are deployed on **Render's free-tier platform**, which supports auto-deployment from GitHub repositories. |
| **3. Environment Variables** | Sensitive data like database URIs, Stripe keys, and JWT secrets are configured using Render's built-in **Environment Variables** feature. |
| **4. Build Commands** | - Frontend: npm run build (Vite)  - Backend: node index.js or npm start |
| **5. CORS Configuration** | To allow communication between different frontend and backend domains (especially during development or cross-origin deployment), the **CORS policy** is handled using the cors Node.js middleware. |

**Key Deployment Notes**

* The frontend is set up as a **static site** on Render using the Vite build output (dist/ folder).
* The backend is deployed as a **web service**, exposing RESTful APIs over HTTPS.
* CORS is enabled with the following middleware in server.js:

**Benefits of This Deployment Setup**Table 10 : deployment setup

| **Aspect** | **Benefit** |
| --- | --- |
| **Free Hosting** | Budget-friendly deployment using Render's free tier. |
| **Auto-Deploy** | GitHub integration allows code changes to trigger new deployments automatically. |
| **Environment Management** | Render allows secure handling of environment secrets without exposing them in code. |
| **CORS Handling** | Ensures secure communication between React and Node apps on different subdomains or ports. |

### Technologies Used

The **Audio Player by Saideep Kankarla** is developed using the **MERN Stack**—a powerful combination of technologies for building modern full-stack applications. Alongside the core MERN components, the project integrates several additional tools to enhance functionality, performance, and user experience.

**1. MERN Stack**

The MERN stack is a collection of JavaScript-based technologies used to build dynamic web applications:

Table 11 : Mern stack

| **Technology** | **Purpose** | **Role in This Project** | **Real-Life Applications** |
| --- | --- | --- | --- |
| **MongoDB** | NoSQL Database | Stores user data, music metadata, and payment records in JSON-like documents. | Used by platforms like **Uber**, **eBay**, and **Forbes** for scalability and flexible schemas. |
| **Express.js** | Web Application Framework | Handles routing, API endpoints, and server-side logic. | Powering backend services for apps like **MySpace** and **Accuweather**. |
| **React.js** | Frontend Library | Manages UI rendering, state updates, and component-based design. | Used by **Facebook**, **Instagram**, **Netflix** for dynamic SPAs. |
| **Node.js** | JavaScript Runtime | Runs the server-side code and handles async operations and APIs. | Used in production by **LinkedIn**, **PayPal**, and **Walmart**. |

**2. Additional Technologies & Tools**

Table12 : additional technologies

| **Technology / Tool** | **Purpose** | **Usage in This Project** |
| --- | --- | --- |
| **Vite** | Frontend Build Tool | Speeds up React development with hot module replacement. |
| **SCSS (SASS)** | CSS Preprocessor | Used for styling components with nested and reusable styles. |
| **Material UI (MUI)** | React UI Library | Provides responsive and accessible UI components. |
| **JWT (JSON Web Tokens)** | Authorization | Used for secure login sessions and API protection. |
| **bcryptjs** | Password Hashing | Ensures passwords are safely stored using salt + hash. |
| **Stripe** | Payment Gateway | Enables premium users to subscribe securely for $10. |
| **Render** | Hosting Platform | Deploys the backend API and database on free-tier cloud. |

**🌍 Real-Life Applications of MERN Stack**

1. **E-Commerce Sites**  
   Platforms like **Shopify** and **BigCommerce** use similar tech stacks to support user authentication, product management, and order handling.
2. **Streaming Applications**  
   Just like Spotify or SoundCloud, your audio player demonstrates how premium content access, playback, and subscriptions can be built with React, Node.js, and MongoDB. Your admin panel shows how role-based features can be implemented and scaled.

### Conclusion

The **Audio Player by Saideep Kankarla** project has been an insightful and rewarding experience in full-stack web development. It brought together a wide range of modern technologies and development practices, offering hands-on experience in building a fully functional and interactive web application.

**Key Takeaways:**

* Gained deep practical knowledge of **React.js (v19)** and **Vite** for building fast, modular frontends.
* Understood the importance of **state management**, secure **authentication using JWT**, and **RESTful API** development.
* Learned how to integrate third-party services such as **Stripe** for secure and smooth payment processing.
* Experienced the use of **MongoDB and Mongoose** for designing and handling NoSQL databases effectively.
* Implemented **role-based access control**, showing how different features and permissions can be handled at both frontend and backend levels.

**Practical Applications:**

* The technologies used are widely adopted in modern software development — making this project a practical blueprint for commercial-grade applications like music streaming services, subscription-based platforms, or admin-controlled content portals.
* **React Material UI** and **SCSS** were used to design responsive and user-friendly interfaces, emphasizing user experience.
* **JWT authentication and bcrypt.js** password hashing simulate real-world security practices in web apps.
* **Stripe payment integration** represents how e-commerce and SaaS platforms handle real transactions securely.

**Limitations & Suggestions:**

* The application relies on **free-tier hosting (Render)**, which may lead to slower performance or cold starts; a shift to premium hosting services would enhance scalability and uptime.
* **MongoDB**, while efficient for flexible schema designs, can become complex to manage at scale without proper indexing or sharding.
* A more advanced **audio player interface** with playlist management, equalizer, or download functionality could improve user engagement.
* Implementing **refresh token mechanisms** for authentication and **email verification** would further improve app security.
* Real-time notifications or live chat support (via WebSockets) could enhance the user experience in a production scenario.

In summary, this project serves as a strong foundation in full-stack web application development, demonstrating real-world implementations of secure user management, role-based access, and payment integration — with clear room for future enhancement and scalability.

## References

- https://reactjs.org/  
- https://nodejs.org/  
- https://stripe.com/docs  
- https://mongoosejs.com/  
- https://jwt.io/  
- https://mui.com/

Format Guidelines

1. Detailed and Elaborate report of 40 pages at least is expected
2. Margins - Every page of your document must meet the margin requirements of 1.25 inches on the left and right, and 1 inch on the top and bottom.
3. Font:
   1. Style: Times New Roman,
   2. Font Size:14 (For Headings), 12 (For body text) in black colored text.
   3. All text must be the same justification, like left justified or fully justified.
4. Line Spacing:
   1. Body of the text: 1.5
   2. List of Tables/graphs/charts/bibliography: single line.
5. Alignment:
   1. Title page: Centre
   2. Chapter Heading: Centre
   3. Subheading: Left
   4. Body of the text: Justify
6. Titles: All titles and subtitles should be in bold. All tables/graphs/charts/figures should have appropriate titles.
7. Numbering of the tables, charts, graphs should be in the following fashion: Second table/graph/chart in the second chapter should be numbered as Table/graph/chart no. 2.02; where the first digit stands for chapter no. and digits after (.) stands for number of table/graph/charts in that chapter. Same numbering should be followed for all other chapters.