A Unified Creator Portfolio Platform for Cinematographers and Musicians

Mini Project Report

Submitted by

Noble Joseph

Reg. No.: AJC24MCA-2054

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS (MCA)

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Approved by AICTE, Accredited by NAAC with A+, Accredited by NBA.
Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2026

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report titled "**A Unified Creator Portfolio Platform for Cinematographers and Musicians"** is the bona fide work of **Noble Joseph (Regno: AJC24MCA-2054)** carried out in partial fulfillment of the requirements for the award of the **Degree of Master of Computer Applications** at **Amal Jyothi College of Engineering Autonomous, Kanjirappally.** The project was undertaken during the period from **July 00, 2025** to **October 00, 2025**

Mr. Jinson Devis
Internal Guide

Coordinator
Coordinator

Dr. Bijimol TK
Head of the Department

DECLARATION

I hereby declare that the project report **"A Unified Creator Portfolio Platform for Cinematographers and Musicians"** is a bona fide work done at **Amal Jyothi College of Engineering Autonomous, Kanjirappally**, towards the partial fulfilment of the requirements for the award of the **Master of Computer Applications (MCA)** during the period from **July 00, 2025** to **October 00, 2025**.

Date: **NOBLE JOSEPH**
KANJIRAPPALLY **Reg: AJC24MCA-2054**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Dr. Bijimol T K.** for helping us. I extend my whole hearted thanks to the project coordinator **Coordinator Name** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr. Jinson Devis** for his inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

Jojo Manuel P

ABSTRACT

The rapid evolution of the creative industry has made digital presence essential for artists and professionals seeking visibility, collaboration, and growth. However, most existing platforms are either generalized social media spaces or fragmented solutions that do not adequately serve the diverse needs of creators at different career stages. This project proposes the development of a comprehensive, cloud-based portfolio platform tailored specifically for cinematographers and musicians, offering a unified space for showcasing talent, managing personal branding, and enabling meaningful professional discovery.

Creators across all experience levels—from beginners to professionals—can build dynamic, multimedia-rich public profiles. These profiles support images, videos, audio tracks, and documents, along with detailed information such as biographies, achievements, and social media links. Users categorize themselves by profession and style (e.g., fashion cinematography, classical music, documentary filmmaking, contemporary music) and specify their experience level so the platform adapts their presentation accordingly.

A unique feature is the domain-specific UI rendering, ensuring each creator type is shown through a layout optimized for their content. Musicians see an audio-focused interface, while cinematographers get visually immersive grids suited for video reels and photo sets. These public profiles are accessible without login, allowing clients, fans, or collaborators to explore portfolios with ease. Viewers can submit private suggestions or appreciation messages, fostering professional feedback without exposing creators to public comments or spam.

Built using the MERN stack—MongoDB, Express.js, Node.js, and React.js—the platform offers responsive, component-driven performance. Media assets are managed via cloud-based storage like Cloudinary or Firebase. Admin users access a secure dashboard to moderate content, manage roles, review feedback, and maintain system health.

This initiative not only fulfills the need for structured and inclusive portfolio management but also sets the stage for future expansion. Planned features include AI-driven collaboration and gig matching, analytics dashboards for tracking growth, and project-based virtual workspaces. By merging technology and creator-centric design, the platform aims to bridge the gap between talent and opportunity—serving as both a stage for expression and a tool for development.

CONTENT

List of Abbreviations

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The rapid evolution of the creative industry has made a strong digital presence essential for artists and professionals like cinematographers and musicians seeking visibility, collaboration, and career growth. However, most existing platforms are either generalized social media spaces or fragmented solutions that do not adequately serve the diverse needs of creators at different career stages.
This project proposes the development of a comprehensive, cloud-based portfolio platform tailored specifically for cinematographers and musicians, offering a unified space for showcasing talent, managing personal branding, and enabling meaningful professional discovery.

The project is structured in phases:

Mini Project Phase (Current Focus)

This phase focuses on developing the core portfolio builder and discovery engine.

- **Core Functionality:** The system allows creators (cinematographers, musicians) across all experience levels to build dynamic, multimedia-rich public profiles. These profiles support images, videos, audio tracks, and documents, along with biographies, achievements, and social media links.
- **Key Features:** A unique feature is the **domain-specific UI rendering**, ensuring each creator type is shown through a layout optimized for their content (e.g., audio-focused for musicians, visual grids for cinematographers). Public profiles are accessible without login, and viewers can submit private appreciation messages, fostering professional feedback without spam.

Main Project Phase (Future Scope)

This phase will extend the platform into a full-scale professional networking and collaboration hub.

- **Extended Scope:** This includes analytics dashboards for creators to track their profile growth and project-based virtual workspaces for collaboration.
- **Advanced Features:** Advanced features will include an AI-driven engine for collaboration and gig matching, connecting creators with clients and other artists based on skills and style.

Technology Stack

The system is built using the **MERN Stack** (MongoDB, Express.js, React.js, and Node.js) with cloud-based storage (like Cloudinary or Firebase) for media management.

1.2 PROJECT SPECIFICATION

The **Unified Creator Portfolio Platform** is a web-based Mini Project developed for the Master of Computer Applications (MCA) degree. The system is built using the **MERN Stack** (MongoDB, Express.js, React.js, and Node.js) and cloud storage (Cloudinary/Firebase).

Scope and Goal

The primary goal of the Mini Project is to establish a systematic and **structured portfolio management system**. This is intended to address the current fragmented and generalized nature of existing platforms. Ultimately, the system aims to bridge the gap between talent and opportunity by making creative work discoverable and easy to explore.

Core Functionalities

The system divides responsibilities among three main user types. The **Creator** (Musician/Cinematographer) can sign up, build a rich multimedia profile, upload media, and categorize their work. The **Public Viewer** (Client, Fan, Collaborator) can browse, search, and filter profiles by skill or style without a login and can send private feedback. The **Admin** module oversees the system by moderating content, managing user roles, and reviewing feedback to maintain system health.

Professional Environment and Safety

The system incorporates features to ensure a professional and secure environment. The workflow is streamlined, starting from a creator building their profile to a client discovering it.

- **Private Feedback:** A key feature is the private feedback system, which allows viewers to send appreciation or suggestions directly to the creator, preventing public spam and fostering professional dialogue.
- **Content Moderation:** The Admin dashboard provides tools to review and manage user-uploaded content and profiles, ensuring a high-quality, professional environment for all users.

Future Scope

This project is designed to be the foundation for a full-scale creative collaboration hub. Future phases will incorporate:

- **AI-driven** collaboration and gig matching.
- **Analytics dashboards** for creators to track profile growth.
- **Project-based** virtual workspaces.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

This project is a targeted technological response to the significant systemic challenges inherent in the digital creative industry. The development of this system is rooted in the recognition that while creative talent is abundant, the platforms for showcasing that talent are fragmented and often unsuitable. The current, non-unified system is characterized as generalized, restrictive, or requiring high technical skill. The core problem addressed is the inefficiency in professional discovery and the failure to manage a creator's personal brand in a structured manner that respects their specific medium (video vs. audio).

2.2 LITERATURE REVIEW

A review of existing platforms reveals a clear gap in the market for a unified solution. The current landscape is divided:

1. **Generalized Social Platforms (e.g., Instagram, LinkedIn):** These platforms are built for mass-market networking, not high-fidelity creative portfolios. They force content into restrictive formats (e.g., short clips, square images) and heavily compress video and audio, degrading the quality of the work shown. They also mix professional work with personal social content.
2. **Specialized Audio Platforms (e.g., SoundCloud, Bandcamp):** These are excellent for musicians and audio professionals but offer no features or support for visual artists like cinematographers.
3. **Specialized Visual Platforms (e.g., Vimeo, Behance):** These are strong for filmmakers, designers, and photographers but lack integrated audio hosting, music-specific discovery features, or layouts appropriate for musicians.
4. **Website Builders (e.g., Squarespace, Wix):** These provide full customization but come at a high cost in both money and time. They require technical and design skills that many creators lack. Most importantly, they are "isolated islands"—they do not exist within a centralized, searchable discovery engine where clients and collaborators can find new talent.

This review confirms that no single, dominant platform exists that effectively unifies high-quality video and audio hosting, domain-specific presentation, and professional discovery tools for both cinematographers and musicians.

The current fragmented ecosystem forces creators to face several critical disadvantages:

- **Fragmented Professional Identity:** A creator who is both a musician and a filmmaker must maintain separate profiles on SoundCloud, Vimeo, and perhaps a personal website. This fragments their brand and makes it difficult for collaborators to see their full range of skills.
- **Poor Discovery & Collaboration:** A film director looking for a composer has no single talent pool to search. They must manually check multiple sites, and musicians have no direct way to be discovered by them.
- **Generic, Unprofessional Presentation:** Using a one-size-fits-all platform (like Instagram) fails to present work in its best light. A cinematic reel is not a 60-second vertical video, and an audio track is not a static image post.

- **Lack of Professional Environment:** Generalized social media platforms mix professional work with public comments, spam, and personal content, which can detract from a professional image and discourage constructive feedback.

2.3 PROPOSED SYSTEM

The proposed system is a comprehensive, cloud-based portfolio platform built on the **MERN Stack** (MongoDB, Express.js, React.js, and Node.js). It provides a single, unified space for cinematographers and musicians to build, manage, and showcase their professional brand.

The system is designed with a component-driven architecture using **React.js** for a responsive frontend. The backend, powered by **Node.js** and **Express.js**, provides a secure RESTful API for managing user data, media, and feedback. **MongoDB** is used as the database, offering a flexible NoSQL structure ideal for storing diverse user profiles and media metadata. All heavy media assets (videos, audio) are handled by a dedicated cloud storage service like **Cloudinary** or **Firebase Storage** to ensure high-speed delivery and scalability.

The system's core feature is the **domain-specific UI rendering**: when a user signs up as a "Musician," their profile automatically renders with an audio-focused interface. A "Cinematographer" profile renders with a visually immersive grid suited for video reels. This ensures all work is presented in the most professional and effective manner possible without any technical effort from the creator.

2.4 ADVANTAGES OF PROPOSED SYSTEM

The implementation of this unified platform offers significant advantages over the existing fragmented system:

- **Unified Professional Identity:** Creators can finally showcase all their talents (e.g., cinematography, music composition, sound design) in one professional, high-quality portfolio.

- **Tailored & Professional Presentation:** The domain-specific UI rendering ensures that video and audio work are presented in the best possible light, enhancing viewer engagement and professional appeal.
- **Enhanced Discovery & Collaboration:** A centralized, searchable platform allows clients, directors, and producers to easily find and contact talent based on specific skills (e.g., "classical composer," "fashion cinematographer").
- **Secure & Professional Environment:** The private feedback system protects creators from public spam and harassment, fostering constructive, professional dialogue with potential clients and collaborators.
- **Scalable & Modern Technology:** Building on the MERN stack provides a fast, responsive user experience and a flexible foundation ready for future expansion.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

A Feasibility Study was conducted for the Unified Creator Portfolio Platform project to rigorously assess its viability and practicality before committing significant resources to development. The study focused on three critical areas: Economical, Technical, and Behavioral feasibility, ensuring the proposed system is both justified and achievable in the current market.

3.1.1 Economical Feasibility

The Unified Creator Portfolio Platform is economically viable. It is built using cost-effective, open-source technologies, primarily the MERN stack (MongoDB, Express.js, React.js, Node.js), which eliminates all software licensing fees and reduces initial development expenses. Hosting and media storage costs (e.g., Cloudinary, Firebase) are cloud-based and operate on a pay-as-you-go model, allowing costs to scale directly with user adoption and media volume. By addressing a clear and persistent gap in the market for a unified creative portfolio, the platform has a high potential for rapid adoption by its target audience. The high demand for such a specialized tool suggests a favorable cost-benefit ratio, with potential to recover investment through future premium features (as outlined in the future scope).

3.1.2 Technical Feasibility

The project is technically feasible. The MERN stack is a mature, reliable, and widely supported open-source technology ecosystem. React.js is ideal for building the required responsive, component-driven user interface, especially for the complex domain-specific UI rendering. Node.js and Express.js provide a robust, high-performance backend capable of handling API requests and managing user data. Critically, cloud-based media management services (like Cloudinary or Firebase) are designed specifically for this use case and provide powerful, scalable APIs for video/audio uploading, transcoding, and high-speed delivery via CDN. Skilled MERN stack developers are readily available, and the extensive documentation for these technologies minimizes technical risk.

3.1.3 Behavioral Feasibility

Behavioral (or Operational) feasibility is exceptionally high, as the system is designed to directly solve the primary frustrations of its target users (creators). High user acceptance is anticipated

because the system eliminates the disorganization and fragmented identity creators currently face. By offering a smooth, professional, and unified platform that presents their work in the best possible light (through domain-specific UIs) and protects them from spam (via private feedback), the system is expected to be readily adopted by cinematographers and musicians. The system's alignment with real-world needs (professional discovery) was validated through informal interviews and analysis of creator complaints on existing social platforms.

3.1.4 Feasibility Study Questionnaire

1. What platforms do you currently use to showcase your portfolio (e.g., Instagram, Vimeo, SoundCloud, personal website)?
2. What are the biggest challenges or frustrations you face with your current portfolio solution?
3. If you work in multiple creative fields (e.g., film and music), how do you manage showcasing both?
4. How important is the quality (e.g., video resolution, audio bitrate) of your media when presented online?
5. How do you currently find collaborators or new clients? How do they find you?
6. What are your concerns about receiving feedback or comments on your work on public platforms?
7. What features are "must-haves" for a platform designed for your specific creative field (cinematography or music)?
8. How much time or money are you willing to spend to maintain a professional portfolio?
9. What features would make a portfolio platform easier and more efficient for you to use?
10. What are the challenges you face from initial profile setup to getting discovered by a client?

3.1.1 Geotagged Photograph

3.2 SYSTEM SPECIFICATION

3.2.1 HARDWARE SPECIFICATION (CLIENT)

- **Device:** Any modern computer, tablet, or smartphone.
- **Browser:** Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge (latest versions).

3.2.1 HARDWARE SPECIFICATION (SERVER)

- **Platform:** Cloud-based hosting (e.g., Vercel, Heroku, AWS, Azure).
- **Environment:** Node.js runtime environment.
- **Storage:** Cloud-based object storage (e.g., Cloudinary, Firebase Storage, AWS S3).

3.2.1 Software Specification

Front End - **React.js** (Framework)

Back End -**Node.js** and **Express.js** (Runtime and Framework)

Database - **MongoDB** (NoSQL Database)

Client on PC - Windows 7 and above.

Technologies used - JS, HTML5, AJAX, J Query, PHP, CSS, List APIs,

3.3 SOFTWARE DESCRIPTION

3.3.1 MERN Stack

- **MongoDB (Database):** A NoSQL, document-based database. Its flexible schema is perfect for storing diverse user profiles, which can include varying fields for achievements, social links, and multimedia content.
- **Express.js (Backend Framework):** A minimal and flexible Node.js web application framework used to build the RESTful APIs. It handles all business logic, routing, and communication between the frontend and the database.
- **React.js (Frontend Library):** A JavaScript library for building user interfaces. It is used to create a fast, responsive, and component-driven single-page application (SPA), enabling features like the domain-specific UI rendering.

- **Node.js (Backend Runtime):** A JavaScript runtime built on Chrome's V8 engine. It allows us to run JavaScript on the server, enabling a full-stack JavaScript application that is efficient and scalable.

3.3.2 Cloud Storage (Cloudinary / Firebase)

A dedicated cloud-based media management platform. It is used to handle all media uploads, storage, transformations (e.g., video transcoding, audio optimization), and high-speed content delivery (CDN) for all user-uploaded assets. This separates media from the application logic, improving performance and scalability.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

The **Cloud-Based Portfolio Platform** project is designed as a web-based MERN Stack application that connects creators (cinematographers and musicians), clients, and platform administrators through a unified platform. The system focuses on multimedia portfolio management, professional discovery, domain-specific UI rendering, and private feedback mechanisms, ensuring efficient professional presentation and secure communication.
System design is the process of defining the architecture, modules, interfaces, and data flow of the entire system to meet functional and non-functional requirements. It serves as a blueprint for developers to build a scalable, secure, and maintainable application.
In this project, the system design phase includes:

- **Architectural Design:** Defines how various components (frontend, backend, cloud storage services, and database) interact.
- **Database Design:** Structures the data models (User, Profile, Project, MediaAsset, Feedback, etc.) using MongoDB.
- **Component Design:** Breaks down the backend services such as Authentication, Profile Management, Media Upload (to Cloudinary/Firebase), and the Feedback Service.
- **Interface Design:** Describes how creators, viewers, and admins interact with the system through the frontend (React-based web interface).

The overall design ensures that:

- The system is modular, with separate services for each function.
- Data is secured through JWT-based authentication and role-based access control.
- The application remains responsive and reliable, supporting concurrent user operations.
- Communication between modules occurs over RESTful APIs, with user data stored in MongoDB and large media assets managed via cloud storage.

4.2 UML DIAGRAM

Unified Modeling Language (UML) diagrams are a standardized way to visualize the design and architecture of a software system. They help developers, designers, and stakeholders understand the structure, behavior, and interactions within a system before actual implementation. For the Cloud-Based Portfolio Platform project, UML diagrams provide a comprehensive view of how creators, viewers, administrators, and backend services interact to manage multimedia portfolios, professional discovery, and private feedback in an efficient and secure manner. By using UML, the project ensures clarity in requirements, reduces ambiguity, and serves as a blueprint for development.
The **Use Case Diagram** captures the functional requirements of the system from the perspective of different actors. In this project, the primary actors are the **Creator** (Musician/Cinematographer), who interacts with the system to register, build a profile, upload media, and view feedback, and the **Admin**, who manages users, moderates content, and reviews feedback. A third actor, the **Viewer** (or Client), can browse public profiles and send private appreciation messages.
The **Class Diagram** represents the static structure of the system, defining the data entities, their attributes, methods, and relationships. For this project, core classes include **User**, **Profile**, **Project**, **MediaAsset**, and **Feedback**. Each class reflects a corresponding MongoDB collection and encapsulates the operations that can be performed on the data. Relationships between classes, such as a Profile having multiple Projects, are also clearly defined. This diagram helps in

designing the database schema, backend models, and understanding how data flows between different parts of the system.

The **Sequence Diagram** illustrates the dynamic behavior of the system by showing how objects interact over time. It depicts step-by-step message exchanges between the User, Frontend (React), Backend (Node.js + Express), external services like Cloudinary, and MongoDB. For example, during the media upload process, the creator's request travels from the frontend to the backend to get a secure signature, the file is then uploaded to Cloudinary, and the final URL is saved in the database. Sequence diagrams are valuable for understanding the exact order of operations.

The **Collaboration Diagram**, also called a Communication Diagram, complements the sequence diagram by focusing on object interactions and their structural relationships. It shows how services such as the backend API, Profile Service, Media Service, and Feedback Service collaborate to perform user requests. By numbering the messages, the diagram highlights the sequence of operations while also showing which components interact directly.

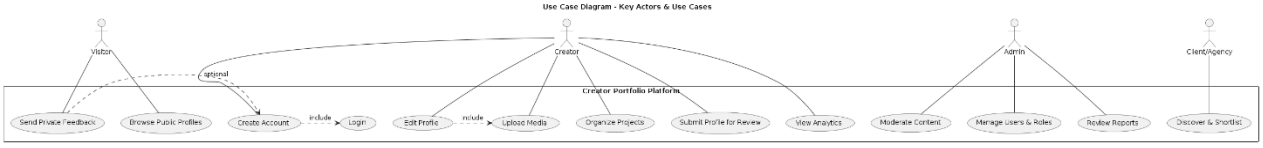
The **Deployment Diagram** models the physical architecture of the system. It visualizes where software components are deployed, such as the React frontend on a static hosting service (like Render Static Site), the Node.js backend on an application server (like Render Web Service), the MongoDB on a cloud database service, and the media assets on a **Cloud Storage** service (like Cloudinary). It also depicts the communication links, typically via HTTPS or API calls.

Finally, the **Activity Diagram** represents the workflow of operations within the system. It shows the sequence of actions that a creator can perform, including registration, filling out their profile, uploading a new project, and managing received feedback, along with decision points and conditional flows.

In summary, UML diagrams collectively provide a comprehensive blueprint of the Cloud-Based Portfolio Platform. They cover the functional, structural, and behavioral aspects of the system. Use case and activity diagrams focus on user interactions and workflows, class diagrams define the data structure, sequence and collaboration diagrams illustrate component interactions, and deployment diagrams ensure proper physical architecture.

4.2.1 USE CASE DIAGRAM

This Use Case Diagram illustrates the functional boundaries of the MERN Stack application and the interactions between the primary actors: **Creator**, **Admin**, and the public **Viewer/Client**. The **Creator** is the most central actor, capable of performing core activities such as Register Account, Manage Profile (which *includes* Upload Media and Categorize Profile), and View Private Feedback. The **Viewer/Client** is a public actor who can Search Creators, View Public Profiles (which are rendered based on creator type), and Send Private Feedback/Appreciation. The **Admin** actor is responsible for system governance, including Login, Manage Users, Moderate Content, and Review All Feedback. The relationships show that Manage Profile is a complex use case extended by creators, while View Public Profiles is the primary function available to viewers.



Diagram

Fig 4.2.1. USE CASE DIAGRAM

4.2.2 SEQUENCE DIAGRAM

1. **User Login and Authorization**
   - The sequence starts when a **User** (Creator or Admin) opens the Web App.
   - The **Web Browser (React)** sends a POST request with login credentials to the **Backend API (Node.js)**.

- The Backend uses the **Authentication Service** to validateUserCredentials.
- The service queries the **MongoDB Database** for user details, returns the details, and the Backend issues a 200 OK response with a JWT token and user info, allowing the User to view their dashboard.

2. **Media Upload Process (to Cloudinary)**
   - The **Creator** selects a file (image, video, audio) to upload via the **React Frontend**.
   - The Frontend sends a request to the **Backend API** to get a secure upload signature.
   - The Backend, using the **Media Service**, requests a signature from the external **Cloudinary API**.
   - Cloudinary returns a unique signature and timestamp to the Backend.
   - The Backend sends the signature to the React Frontend.
   - The Frontend performs a direct POST upload to **Cloudinary**, including the file and the signature.
   - Cloudinary saves the file and returns the secure URL and public_id to the Frontend.
   - The Frontend sends this URL and public_id to the **Backend API**.
   - The Backend (Media Service) saves this information as a new MediaAsset document in the **MongoDB Database**, linking it to the Creator's profile/project.
   - The Backend returns a 201 Created response to the Frontend.
3. **Profile Update**
   - The **Creator** submits changes to their profile (e.g., new bio, updated style category) via the **React Frontend**.
   - The Frontend sends a PUT request with the new data to the **Backend API**.
   - The Backend utilizes the **Profile Management Service** to updateProfileData.
   - The service validates the data and updates the corresponding Profile document in the **MongoDB Database**.
   - A 200 OK response is returned to the Frontend with the updated profile data.
4. **Submit Private Feedback**
   - A **Viewer** fills out the feedback form on a Creator's public profile page (**React Frontend**).
   - The Frontend sends a POST request with the message and profileId to the **Backend API**.
   - The Backend uses the **Feedback Service** to createNewFeedback.
   - The service saves the new Feedback document to **MongoDB**.
   - The User receives a 200 OK response with "Feedback Sent Successfully."

Diagram

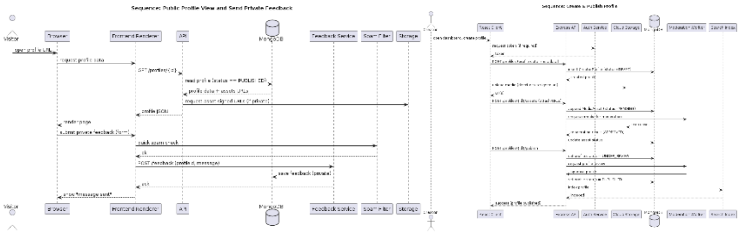


Fig 4.2.2. SEQUENCE DIAGRAM

4.2.3 STATE CHART DIAGRAM

This diagram illustrates the lifecycle of a **Creator Profile** within the platform, from initial registration to becoming a verified member of the community.

The flow begins when a new user registers, placing the profile in the **Profile Incomplete** state. When the creator adds their essential details (bio, profession, first project), the state transitions to **Profile Active**. From here, the creator can continuously Update Profile, which loops back to the Active state.
Separately, an **Admin** can review the profile. If the Admin Verifies the profile, it moves to the **Verified** state, (perhaps unlocking a "verified" badge). If the Admin finds issues, the profile moves to the **Flagged** state. A flagged profile must be Edited by Creator, which moves it to the **Pending Re-review** state. The Admin then re-evaluates; if Approved, it returns to **Profile Active**, but if Rejected Again, it moves to the terminal **Suspended** state. A creator can also Delete Account, which moves the profile to the terminal **Deleted** state.
Diagram

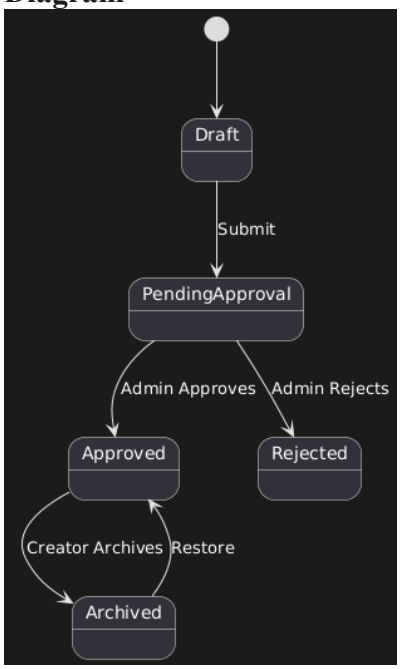


Fig 4.2.3. STATE CHART DIAGRAM

4.2.4 ACTIVITY DIAGRAM

1. **Registration and Login**
   - The process starts with **Creator Registration**.
   - After providing details, the creator can **Login to System**.
2. **Profile Creation and Management**
   - Once logged in, the creator lands on their **Dashboard**.
   - A conditional check determines if **Profile Complete?**.
   - If no, the creator must **Fill Profile Details** (bio, profession, style, etc.).
   - If yes (or after completion), the creator can proceed to **Manage Portfolio**.
3. **Media and Project Management**
   - From Manage Portfolio, the creator can **Add New Project**.
   - For that project, they **Upload Media** (image, video, audio).
   - The system (in a parallel sub-process) Requests Upload Signature from the backend, Uploads File to Cloud Service, and Saves Asset URL to DB.
   - The media is then **Displayed on Profile**. The creator can loop back to upload more media or manage other projects.
4. **Public Viewing and Feedback (Parallel Flow)**
   - A **Public Viewer** visits the site.
   - They **Search/Browse Creators**.

- They **View Creator Profile** (where the system Renders UI based on Profession).
- The viewer can then **Send Private Feedback**.
- The feedback is **Stored in DB** and the flow terminates for the viewer.

5. **Admin Moderation (Parallel Flow)**
   - An **Admin** logs in and accesses the **Admin Dashboard**.
   - They **View Flagged Content / New Profiles**.
   - They **Review Content**.
   - A conditional check asks: **Approve Content?**.
   - If no, the admin will **Flag Content & Notify Creator**, and the flow terminates.
   - If yes, the admin will **Mark as Verified**, and the flow terminates.

Diagram

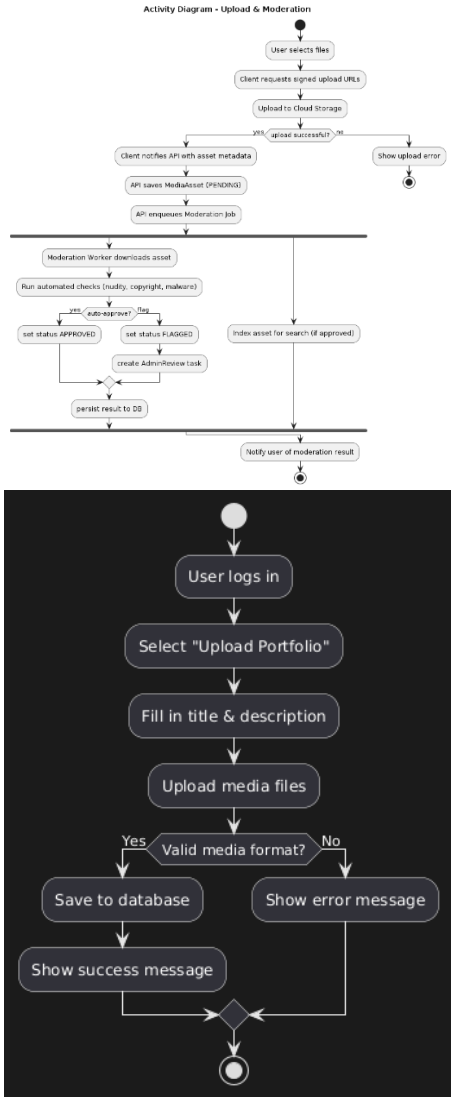


Fig 4.2.4 ACTIVITY DIAGRAM

4.2.5 CLASS DIAGRAM

The image presented is the Class Diagram for the Cloud-Based Portfolio Platform. This diagram visually represents the static structure of the system, defining the core entities (classes), their attributes (data they hold), and their operations (functions they perform), as well as the relationships between them.

The system's architecture is built around the **User** class, which holds basic authentication data like email, password, and role. The **Admin** class *inherits* from the User, gaining operations like manageUser() and moderateContent().
The **Profile** class is central to the application and has a one-to-one relationship with a User (where role is "creator"). It holds all branding information, such as bio, profession (Musician/Cinematographer), style, and experienceLevel. A Profile *has one-to-many* **Project**s. The **Project** class holds a title and description. In turn, a Project *has one-to-many* **MediaAsset**s. The **MediaAsset** class stores the type (image, video, audio), the url (from Cloudinary), and its public_id. Finally, a Profile *has one-to-many* **Feedback** objects, which store the message and senderEmail (optional).

Diagram

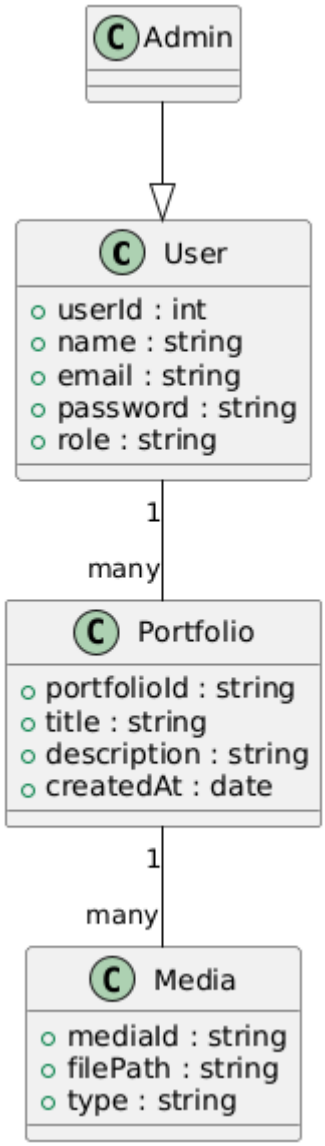


Fig 4.2.5 CLASS DIAGRAM

4.2.6 OBJECT DIAGRAM

The image provided is the Portfolio Platform - Object Diagram, which shows a specific snapshot of the system's data at runtime, illustrating the interactions between instantiated objects.

The Admin1:Admin object (role="admin") is shown to manage the Profile1 object. This Profile1:Profile object (profession="Musician") is owned by the Creator1:User object (email="music@test.com"). Profile1 contains an object Project1:Project (title="My First Album"). This project, in turn, contains two media assets: Media1:MediaAsset (type="audio", url=".../song1.mp3") and Media2:MediaAsset (type="image", url=".../album_cover.jpg"). Separately, an anonymous Viewer object submits a Feedback1:Feedback object (message="Amazing work!"), which is linked to Profile1.

Diagram

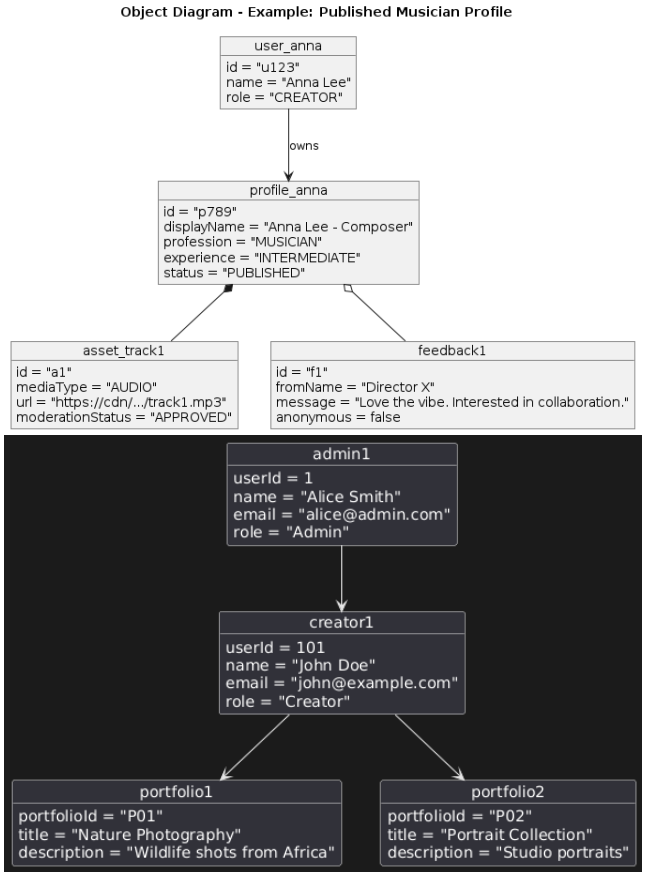


Fig 4.2.6 OBJECT DIAGRAM

4.2.7 COMPONENT DIAGRAM

The diagram maps out the structural architecture and relationships between the project's major software components. The entire system is built around the **Backend API (Node.js + Express)**, which functions as the central hub for all business logic. This Backend API directly interfaces with the **Web Frontend (React.js)**, which serves as the user-facing application for creators, viewers, and admins.

The Backend API coordinates and manages communication with several distinct services and data stores. The core data (user info, profiles, project text) is held within the **Database (MongoDB)**. Dedicated microservices (or logical modules) handle specific logic: the **Authentication Service** manages all user login and verification; the **Profile Management Service** handles CRUD operations for profiles and projects; and the **Feedback Management Service** processes incoming messages.

Crucially, the Backend API (via a **Media Service** component) and the Web Frontend *both* interface with an external third-party component, the **Cloud Storage Service (Cloudinary/Firebase)**, which is responsible for storing and delivering all heavy media assets (images, videos, audio).

Diagram

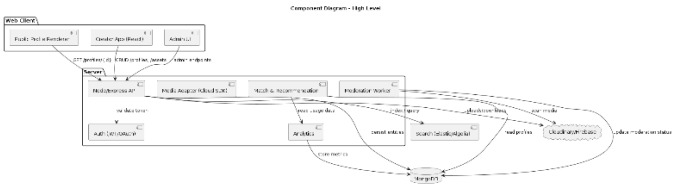


Fig 4.2.7 COMPONENT DIAGRAM

4.2.8 DEPLOYMENT DIAGRAM

The image provided illustrates the Deployment Diagram for the Cloud-Based Portfolio Platform, which maps the physical arrangement and communication pathways between the software and hardware components. The architecture is divided into four distinct nodes.

The **Client Device** hosts the **Web Browser**, which runs the **React.js Frontend** and communicates via HTTPS. This connection targets the **Application Server** node (e.g., a Render Web Service). This server hosts the **Backend API (Node.js + Express)**, which contains the core business logic (Auth Service, Profile Service, Feedback Service). The Backend API maintains a dedicated Database Connection to the **Database Server** node (e.g., MongoDB Atlas), which hosts the **MongoDB Database**.
Finally, a fourth node, the **Cloud Media Service** (e.g., Cloudinary), is essential. This node, hosted by a third-party vendor, stores all **Media Assets**. It communicates directly with the Client Device (for file uploads) and the Application Server (for API-based management).
Diagram

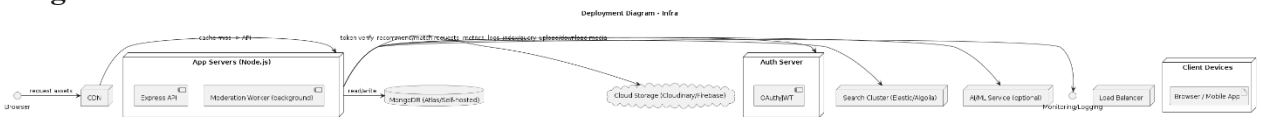


Fig 4.2.8 DEPLOYMENT DIAGRAM

4.2.9 COLLABORATION DIAGRAM

This diagram illustrates the object interactions required for a Creator to add a new project to their profile, emphasizing the relationships between components.

The flow begins when the :Creator actor interacts with the :ProjectDashboard (React) UI.

1. The :ProjectDashboard sends a createNewProject(data) message to the :BackendAPI (Node.js).
2. The :BackendAPI routes this to the :ProjectController.
3. The :ProjectController first sends a verifyToken(token) message to the :AuthService to ensure the user is authenticated.
4. Once verified, the :ProjectController sends a create(data, userId) message to the :ProjectService.
5. The :ProjectService then sends an insert(document) message to the :ProjectModel (MongoDB).
6. The :ProjectModel interacts with the :Database and returns the new project object.
7. A success message is passed all the way back to the :Creator.

This diagram shows the structural collaboration between the frontend, backend controllers, services, and data models to fulfill a single use case.

Diagram

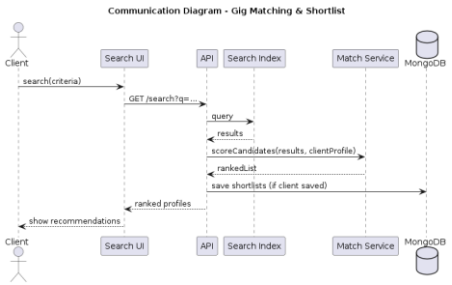
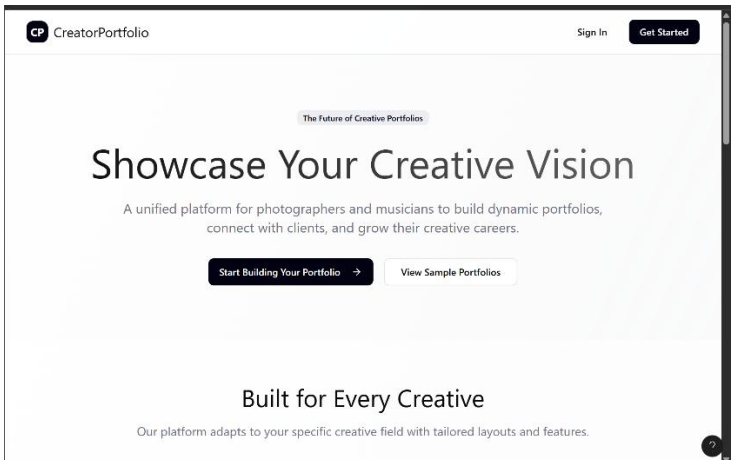


Fig 4.2.9 COLLABORATION DIAGRAM

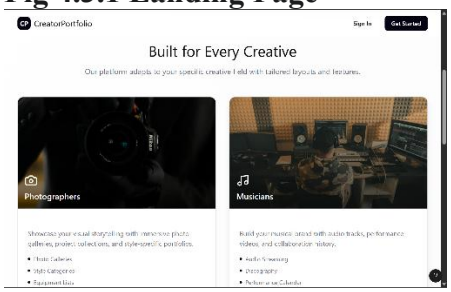
4.3 USER INTERFACE DESIGN USING FIGMA

4.3.1 Form Name: Landing Page



Diagram

Fig 4.3.1 Landing Page



4.3.3 Form Name: Creator Dashboard

Diagram

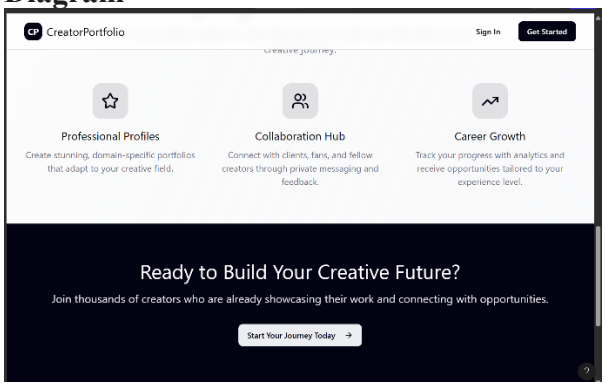


Fig 4.3.3 Creator Dashboard

4.4 DATABASE DESIGN

4.4.1 Relational Database Management System (RDBMS)

Although this platform utilizes the NoSQL database MongoDB, the design incorporates logical data relationships similar to an RDBMS to ensure data consistency and integrity. Key logical relationships managed include linking a **User**'s account (userId) to their specific **Profile** document, linking a **Profile** (profileId) to their various **Project**s, and linking each **Project** (projectId) to its corresponding **MediaAsset**s. These logical connections (using ObjectId references) ensure that data is not isolated and that the system can accurately build a creator's full portfolio from separate, normalized collections.

4.4.2 Normalization

Normalization principles are applied logically within the document-based MongoDB to reduce redundancy and improve data integrity. This is achieved by **Minimizing Duplication**; for instance, creator details (bio, profession) are stored once in the **Profile** collection. Their media assets (images, videos) are stored in a separate **MediaAsset** collection, which references the projectId or profileId rather than duplicating the creator's info for every file. This dedication to **Data Consistency** ensures that separate collections exist for core entities (User, Profile, Project, Feedback), so that information like a creator's bio is updated in only one place.

4.4.3 Sanitization

Data Sanitization is critical to security and involves cleaning and validating user input before it is processed or stored in the database. This includes rigorous **Input Validation** for fields like profile bio, project descriptions, and feedback messages to ensure they adhere to required formats and to strip any potentially malicious HTML. **Security Protection** is paramount, as sanitization prevents common web vulnerabilities like **Cross-Site Scripting (XSS)** by cleaning input strings and mitigates **NoSQL injection** by ensuring inputs are treated strictly as data, not executable commands. Furthermore, **Credential Security** mandates that passwords must be hashed (using bcrypt) and stored securely in the User collection.

4.4.4 Indexing

Indexing is vital for increasing the efficiency of data retrieval, which is essential for the core functionality of the platform's search and discovery feature. To achieve **Optimizing Search Queries**, indexes are created on frequently queried fields. For this project, essential indexes include: the **Profile** collection indexed by profession, style, and experienceLevel to facilitate the main creator search feature. The **User** collection is indexed by username (email) to ensure fast, unique logins. Proper application of indexes minimizes the time required for the system to retrieve matching creator profiles, ensuring rapid response times for clients and viewers.

4.5 TABLE DESIGN (MONGODB COLLECTIONS)

1. User Collection (users)

Primary Key: _id

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	name	String	Required	Full name of the user
3	username	String	Required, Unique, Email format	User login (email

.

No.	Field Name	Data Type	Key Constraints	Description of the field
				address)
4	password	String	Required	Hashed password
5	role	String	Required, Enum: 'creator', 'admin'	User role
6	provider	String	Default: 'local', Enum: 'local', 'google'	Auth provider
7	isBlocked	Boolean	Default: false	Account blocked status
8	createdAt	Date	Auto	Created timestamp

2. Profile Collection (profiles)

Primary Key: _id

Foreign Key: userId references User (_id)

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	userId	ObjectId	FK (User), Required, Unique	Links to the User document
3	bio	String	Max: 1000 chars	Creator's biography
4	profession	String	Required, Enum: 'Musician', 'Cinematographer'	Creator's main profession
5	style	String	Required	E.g., "Classical", "Fashion", "Documentary"
6	experienceLevel	String	Required, Enum: 'Beginner', 'Professional'	Creator's experience level
7	profilePictureUrl	String	Nullable, URL	URL from Cloudinary
8	socialLinks	Object	Nullable	Embedded object for social media links
9	isVerified	Boolean	Default: false	Admin-verified status

No.	Field Name	Data Type	Key Constraints	Description of the field
10	createdAt	Date	Auto	Created timestamp

3. Project Collection (projects)

Primary Key: _id

Foreign Key: profileId references Profile (_id)

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	profileId	ObjectId	FK (Profile), Required, Indexed	Links to the creator's Profile
3	title	String	Required, Min: 3 chars	Title of the project
4	description	String	Max: 2000 chars	Description of the project
5	orderIndex	Number	Default: 0	Used for sorting projects on the profile
6	createdAt	Date	Auto	Created timestamp

4. MediaAsset Collection (mediaassets)

Primary Key: _id

Foreign Key: projectId references Project (_id)

Foreign Key: profileId references Profile (_id)

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	profileId	ObjectId	FK (Profile), Required, Indexed	Links to the creator's Profile
3	projectId	ObjectId	FK (Project), Indexed	Links to the specific Project
4	type	String	Required, Enum: 'image', 'video', 'audio'	Type of media
5	url	String	Required, URL	Secure URL from Cloudinary

No.	Field Name	Data Type	Key Constraints	Description of the field
6	public_id	String	Required	Cloudinary ID for management (e.g., delete)
7	title	String	Nullable	Title for the media (e.g., song name)
8	createdAt	Date	Auto	Created timestamp

5. Feedback Collection (feedbacks)

Primary Key: _id

Foreign Key: profileId references Profile (_id)

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	profileId	ObjectId	FK (Profile), Required, Indexed	The profile receiving the feedback
3	message	String	Required, Max: 1000 chars	The feedback/appreciation message
4	senderName	String	Nullable	Name of the person sending feedback
5	senderEmail	String	Nullable, Email format	Email of the sender
6	isRead	Boolean	Default: false	If the creator has read the message
7	createdAt	Date	Auto	Created timestamp

6. Activity Collection (activities)

Primary Key: _id

Foreign Key: userId references User (_id)

No.	Field Name	Data Type	Key Constraints	Description of the field
1	_id	ObjectId	PK, Auto, Unique	Primary Key
2	userId	ObjectId	FK (User), Required, Indexed	The user who performed the action

.

No.	Field Name	Data Type	Key Constraints	Description of the field
3	role	String	Required	Role of the user at the time of action
4	action	String	Required	E.g., "USER_LOGIN", "PROFILE_UPDATE", "MEDIA_DELETED"
5	details	Mixed	Default: {}	Additional data about the action
6	createdAt	Date	Auto	Created timestamp

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

System Testing is the phase in the project where the integrated **Unified Creator Portfolio Platform** is evaluated against its original requirements to ensure it functions correctly as a whole. The primary goal is to verify that all components of the MERN stack (Frontend, Backend, and Database) and cloud storage services work together seamlessly to achieve the project's objectives, particularly those defined in the Mini Project phase, such as creator registration, media uploading, and domain-specific UI rendering.

This phase is critical for finding errors or vulnerabilities in the completed system before deployment. It verifies that the software meets user expectations and needs, and does not malfunction in an unfavorable way.

Key areas of focus for System Testing in the **Unified Creator Portfolio Platform** project include:

- **End-to-End Workflow Validation:** Testing the entire flow from a new Creator registering, selecting a profession, uploading media, to a Public Viewer discovering and viewing that profile.
- **Security Testing:** Ensuring the private feedback system is secure, media moderation works, and that user authentication (JWT) properly protects profile management routes.
- **Performance Testing:** Verifying the system's ability to handle media uploads and streaming without significant latency.
- **Requirements Verification:** Confirming that all core functionalities listed in the Project Specification (e.g., domain-specific UI rendering, private feedback, search by style) operate as designed.

Successful system testing provides confidence that the application is reliable and ready for deployment to the target audience, which includes Creators, Public Viewers (Clients, Fans), and Administrators.

5.2 TEST PLAN

This Test Plan outlines the strategy for verifying the **Unified Creator Portfolio Platform** Mini Project. The goal is to ensure the integrated system meets its core requirements for portfolio management, media uploading, and professional discovery.

Testing will cover the full integration of the MERN stack components and cloud storage, including the user interfaces, backend logic, and database persistence. The strategy involves various levels of testing. **Unit Testing** will verify individual functions, such as the logic for selecting a UI based on profession. **Integration Testing** will focus on verifying communication and data flow between the React frontend, the Express/Node.js backend, the MongoDB database, and the Cloudinary/Firebase storage. **Validation (System) Testing** will execute the complete core business cycles, such as the full media upload and moderation workflow. Finally, **User Acceptance Testing (UAT)** will validate the system's usability and overall functionality against the requirements of the end-users (Creators, Viewers, and Admins).

A minimum of four test cases will be executed to validate the primary system functions, which include one login case and three core functionalities. The first case is **Creator Login Validation**, ensuring a registered Creator can successfully authenticate. The second is **Media Upload Functionality**, verifying that a creator can upload a media file (e.g., MP3 or MP4), which is

processed by the cloud service. The third, **Domain-Specific UI Rendering**, validates that a user with the 'musician' profession sees the audio-player layout, while a 'cinematographer' sees the video-grid layout. The final critical case is the **Private Feedback Flow**, which verifies that a public viewer can send a message and the intended creator receives it.

The test environment will utilize the MERN Stack development environment. Backend API testing will be conducted using **Postman**, while end-to-end testing will be conducted manually.

5.2.1 Unit Testing

Unit Testing is the initial and most granular phase of testing in the project's development cycle. This testing technique involves validating the smallest testable parts of the application, known as 'units,' to ensure each operates correctly in isolation.

The primary purpose of Unit Testing is to verify the internal logical structure and functions of individual components. For the MERN stack-based Creator Portfolio platform, this includes:

- **Backend Logic:** Testing individual API controllers (in Express.js) and utility functions (in Node.js) that perform specific tasks. This includes verifying the logic for categorizing work by style or handling media metadata.
- **Database Models:** Validating the schema rules (using Mongoose) to ensure data is properly formatted and adheres to constraints, such as checking that a user's email is unique or that a `Media` document is correctly linked to a `Profile`.
- **Frontend Components:** Testing individual React components (using tools like Jest and React Testing Library) to ensure they render correctly and handle user input validation (e.g., verifying the `UploadMedia` modal form).

By completing Unit Testing successfully, the team ensures that the fundamental building blocks of the system are sound, which significantly reduces the probability of errors.

5.2.2 Integration Testing

Integration Testing is the phase that follows Unit Testing. Its purpose is to verify that different modules, services, or components of the system interact and communicate correctly when put together, ensuring the system works as a cohesive whole.

In the context of the MERN stack architecture, Integration Testing specifically focuses on verifying the flow and integrity of data and control between the separate layers:

- **Frontend-Backend Interface:** This involves testing that the React.js frontend correctly sends data (e.g., profile bio updates or new media details) to the Express.js/Node.js Backend API and accurately handles the API's responses.
- **Backend-Database Interface:** This verifies the data mapping and retrieval between the Backend API and the MongoDB database. A key test is confirming that when the system registers a new creator, the `User` and `Profile` documents are correctly created and linked.
- **Backend-Cloud Storage Interface:** This is a critical test, verifying the communication between the Backend API and the Cloudinary/Firebase service. This ensures that a request to upload a file successfully generates a secure URL, and that the URL is correctly received and saved to the `Media` collection in MongoDB.

By performing Integration Testing, the project ensures that the distinct pieces of the application function correctly when working together.

5.2.3 Validation Testing or System Testing

Validation Testing, often referred to as System Testing, is a crucial phase that evaluates the **Unified Creator Portfolio Platform** as a whole, fully integrated entity. This process occurs after all individual units and modules have been integrated and tested.

The primary objective of System Testing is to verify that the final, integrated software meets all the specified requirements and functions correctly in its intended operational environment.

For the Creator Portfolio Mini Project, System Testing validates the following end-to-end business workflows:

- **Full Cycle Workflow (Creator):** Testing the complete process from a user registering as a 'Musician', uploading an MP3, adding a title and style, and seeing it appear correctly on their public profile with an audio player.
- **Full Cycle Workflow (Viewer):** Testing the public-facing discovery. A Public Viewer searches for 'classical music', finds the creator from the previous test, views their profile, and successfully submits a private feedback message.
- **Requirements Verification:** Ensuring that core functionalities, such as the search-by-style and the private feedback system, are accurate and secure.
- **Domain-Specific UI Logic:** Crucially, validating that the system correctly renders the 'musician' UI for one user and the 'cinematographer' UI for another, based on the `profession` field in their profile.
- **Non-Functional Testing:** Assessing the system's performance, especially the load time for media-heavy profiles and the responsiveness of video/audio streaming.

5.2.4 Output Testing or User Acceptance Testing

Output Testing, more commonly and comprehensively referred to as User Acceptance Testing (UAT), is the final phase of system validation. It involves testing the platform against the original business requirements and user needs to confirm that it is ready for deployment.

The core purpose of UAT is to gain confirmation that the system performs acceptably in a real-world context and that the end-users (Creators) are satisfied with its functionality and usability.

For the Creator Portfolio Mini Project, UAT involves:

- **Usability and Interface Review:** Users (simulating creators) test the React-based frontend to ensure the interfaces are intuitive, and the process of building a portfolio is simple and not frustrating.
- **Business Flow Validation:** Users execute specific business scenarios (personas).
    - **Musician Persona:** "Can I easily upload my album and have it display in the correct order?"
    - **Cinematographer Persona:** "Can I easily upload a 4K video reel and have it display in a high-quality grid?"
    - **Client Persona:** "Can I easily find a list of creators tagged with 'documentary' and send them a message?"

- **Requirements Fulfillment:** All functionalities specified in the project scope, from sign-up to the domain-specific UI rendering, must be explicitly tested and approved by the intended user.

Successful UAT signifies that the system is not only technically sound but also operationally acceptable for use in a live environment.

5.2.5 API Testing (Postman)

API Testing with Postman was a specific type of Integration and Validation Testing utilized for the **Unified Creator Portfolio Platform** project. Postman is an API platform used to design, build, and test APIs. It was used to test the backend API (Node.js/Express.js) directly, *before* and *during* the integration with the React.js frontend.

Application to Creator Portfolio Platform:

- **Endpoint and Logic Validation:** Postman was used to send HTTP requests (POST, GET, PUT, DELETE) to every backend API endpoint. This allowed for rapid testing of the business logic for `auth`, `profile`, `media`, and `feedback` routes. For example, creating a new user with a `POST` request to `/api/auth/register` and immediately checking the response.
- **Authentication Testing:** A critical use was testing protected routes. After logging in via the `/api/auth/login` endpoint, the returned JWT token was saved in Postman. This token was then attached to the `Authorization` header for all protected requests (like `PUT /api/profile/me`) to ensure the auth middleware was correctly protecting routes.
- **Data Integrity:** By sending requests via Postman, the MongoDB database could be checked directly to verify that the data was created, updated, or deleted as expected. For instance, after sending a `PUT` request to update a profile `bio`, the database was queried to confirm the document was updated.

By employing Postman, the backend API was confirmed to be robust, secure, and functional, which significantly sped up the frontend integration process.

Example: Postman Test Case

Test Case 1: Creator Registration (API Level)

- **Tool:** Postman
- **Request:** `POST` to `http://localhost:5000/api/auth/register`
- **Body (raw, JSON):**
- ```
  {
    "email": "test_musician@example.com",
    "password": "password123",
    "displayName": "Test Musician",
    "profession": "musician"
  }
  ```
- **Expected Result:**
  - Status Code: `201 Created`
  - Response Body: A JSON object containing the new user's ID and a JWT token.
- **Verification:**
  ```

Creator Portfolio Platform

1. Check the `users` collection in MongoDB to confirm a new document was created with the correct email and a hashed password.
2. Check the `profiles` collection to confirm a corresponding profile document was created with `displayName: "Test Musician"` and `profession: "musician"`.

(Placeholder for Postman Screenshot)

Test Report

Test Case 1 Unified Creator Portfolio Platform: Creator Login Test Case

Test Case ID:	**Test_1**	**Test Designed By:**	
Test Priority:	High	**Test Designed Date:**	
Module Name:	Authentication	**Test Executed By:**	
Test Title:	`POST /api/auth/login` (Valid Credentials)	**Test Execution Date:**	
Description:	Verifies that a registered creator can successfully log in and receive a JWT token.		
Pre-Condition:	A user must exist in the database with email `test_musician@example.com` and password `password123`.		
Step	**Test Step**	**Test Data (JSON Body)**	**Expected Result**
1	Send `POST` request to `/api/auth/login`	`{"email": "test_musician@example.com", "password": "password123"}`	Status Code `200 OK`. Response body contains a valid JWT `token`.
2	Send `POST` request to `/api/auth/login`	`{"email": "test_musician@example.com", "password": "wrongpassword"}`	Status Code `400 Bad Request`. Response body contains an error message.
3	Send `POST` request to `/api/auth/login`	`{"email": "nouser@example.com", "password": "password123"}`	Status Code `400 Bad Request`. Response body contains an

error message.

Post-Condition:	For Step 1, a valid token is returned. For Steps 2 & 3, no token is returned.

(Additional Test Cases for Media Upload, Domain-Specific UI, and Private Feedback would follow)

Eg.Test Report

Test Case 1

Creator Portfolio Platform					
Login Test Case					
Test Case ID: TC-01			**Test Designed By:**		
Test Priority:High			**Test Designed Date:**		
Module Name: Authentication			**Test Executed By :**		
Test Title : POST /api/auth/login			**Test Execution Date:**		
Description: Verifies that a registered creator can successfully log in and receive a JWT token, and that invalid attempts are rejected.					
Pre-Condition : A user must exist in the database with email test_musician@example.com and (hashed) password password123.					
Step	**Test Step**	**Test Data (JSON Body)**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Send `POST` to `/api/auth/login` (Valid)	`{"email": "test_musician@example.com", "password":`	Status Code `200 OK`. Response body contains a valid	Status `200 OK`. Token received.	**Pass**

		"password123"}	JWT token.		
2	Send POST to /api/auth/login (Invalid Pass)	{"email": "test_musician@example.com", "password": "wrongpassword"}	Status Code 400 Bad Request. Response body contains an error message.	Status 400 Bad Request. Error: "Invalid credentials".	**Pass**
3	Send POST to /api/auth/login (No User)	{"email": "nouser@example.com", "password": "password123"}	Status Code 400 Bad Request. Response body contains an error message.	Status 400 Bad Request. Error: "Invalid credentials".	**Pass**
				Actual Result	
Post-Condition: For Step 1, a valid token is returned. For Steps 2 & 3, no token is returned.					

Test Case 2:

Test Case 1

Hope Web- Blood Donor Finding and Donation Automation:	
Login Test Case	
Test Case ID: TC-01	**Test Designed By:**
Test Priority:High	**Test Designed Date:**
Module Name: Authentication	**Test Executed By :**
Test Title : POST /api/auth/login	**Test Execution Date:**
Description: Verifies that a registered creator can successfully log in and receive a JWT token, and that invalid attempts are rejected.	
Pre-Condition : A user must exist in the database with email test_musician@example.com and (hashed) password password123.	

Creator Portfolio Platform

.

Step	Test Step	Test Data (JSON Body)	Expected Result	Actual Result	Status(Pass/Fail)
1	Send `POST` to `/api/media` (Valid)	`{"title": "My New Song", "type": "audio", "mediaUrl": "http://cloud.url/song.mp3"}` (with Auth Header)	Status Code `201 Created`. Response body contains the new media object.	Status `201 Created`. New object returned.	**Pass**
2	Send `POST` to `/api/media` (No Auth)	`{"title": "No Auth Song", "type": "audio", "mediaUrl": "http://cloud.url/song2.mp3"}` (No Auth Header)	Status Code `401 Unauthorized`. Status Code `400 Bad Request`. Response body contains validation error.	Status `401 Unauthorized`. Error: "No token". Status `400 Bad Request`. Error: "mediaUrl is required". **Actual Result**	**Pass** **Pass** **Status(Pass/Fail)**
3	Send `POST` to `/api/media` (Invalid Data)	`{"title": "Missing URL", "type": "audio"}` (with Auth Header)			
Post-Condition: For Step 1, a new document is created in the media collection linked to the user's profile.					

Test Case 1

Test Case ID:	
Test Priority:	
Module Name:	Profile Management
Test Title:	`GET /api/profile/me` (Fetch Own Profile)
Description:	Verifies a logged-in creator can fetch their own profile data, specifically the `profession` field needed for domain-specific UI rendering.

Creator Portfolio Platform

.

Pre-Condition:			Creator (`test_musician@example.com`) is logged in and has a valid JWT (e.g., `Bearer token123...`). Their profile has `profession: "musician"`.		
Test Case ID:			**TC-03**		
Test Priority: High					
Step	**Test Step**	**Test Data (JSON Body)**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Send `GET` to `/api/profile/me` (Valid)	N/A (with Auth Header)	Status Code `200 OK`. Response body is a JSON object containing the user's profile.	Status `200 OK`. Profile object received.	**Pass**
2	Verify Response Data	(Parse JSON response from Step 1)	Response object contains `"profession": "musician"`.	Response object contains `"profession": "musician"`.	**Pass**
3	Send `POST` to `/api/media` (Invalid Data)	`{"title": "Missing URL", "type": "audio"}` (with Auth Header)			
Post-Condition: For Step 1 & 2, the frontend would receive the profession and know to render the "musician" UI.					

Test Case 1

Test Case ID:	
Test Priority:	
Module Name:	Feedback
Test Title:	`POST /api/feedback/:profileId` (Submit Private Feedback)
Description:	Verifies that a public (unauthenticated) viewer can successfully submit a private feedback message to a creator's profile.
Pre-Condition:	A valid `profileId` exists (e.g., `60d5f...a1b2c3`). This is a public route and requires no authentication.

Creator Portfolio Platform

Test Case ID:			TC-04		
Test Priority: High					
Step	**Test Step**	**Test Data (JSON Body)**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**
1	Send `POST` to `/api/feedback/60d5f...a1b2c3` (Valid)	`{"message": "Great portfolio!"}`	Status Code `201 Created`. Response body contains the new feedback object.	Status `201 Created`. New object returned.	**Pass**
2	Send `POST` to `/api/feedback/60d5f...a1b2c3` (Invalid Data)	`{"msg": "Wrong field"}`	Status Code `400 Bad Request`. Response body contains validation error.	Status `400 Bad Request`. Error: "message is required".	**Pass**
3	Send `POST` to `/api/media` (Invalid Data)	`{"title": "Missing URL", "type": "audio"}` (with Auth Header)			
Post-Condition: For Step 1, a new document is created in the feedback collection linked to the correct profile ID..					

Code

Screenshot

Test report

Minimum 4 test cases (1 login 3 functionalities)

CHAPTER 6
IMPLEMENTATION

6. 1 INTRODUCTION

The **Implementation phase** of the Cloud-Based Portfolio Platform for Cinematographers and Musicians project involves the actual construction and deployment of the planned system modules, transitioning the design specifications into a functional application. This phase focuses on building the software components defined in the project scope: the **Admin**, **Creator (Musician/Cinematographer)**, and **Public Viewer** modules.

The implementation is executed utilizing the chosen technology stack, the **MERN Stack** (MongoDB, Express.js, React.js, and Node.js), supplemented with cloud-based storage like **Cloudinary** or **Firebase** for media asset management. This process involves coding the logic for core functionalities, such as building dynamic multimedia profiles, the **domain-specific UI rendering** (audio-focused vs. visual grids), and the private feedback and appreciation mechanism.

Successful implementation culminates in a fully integrated system that is ready for comprehensive testing (System Testing) and, finally, deployment. This phase is the realization of the project's goal: to **bridge the gap between talent and opportunity**, serving as both a stage for expression and a tool for development.

6.2 IMPLEMENTATION PROCEDURES

The implementation of the Cloud-Based Portfolio Platform project involves transitioning the design into a fully functional application using a structured approach. This procedure covers the systematic building, integration, and final preparation of the MERN stack components and cloud services for deployment.

6.2.1 User Training

User Training is essential to ensure that all stakeholders can effectively use the new system.

- **Training on Application Software:** This involves providing targeted training to the different user groups on how to operate their respective module interfaces and utilize the new portfolio features. For **Creators** (Musicians/Cinematographers), training focuses on building their multimedia profiles, categorizing their work, and managing private feedback. For **Admin Staff**, training covers content moderation, role management, and system health monitoring.
- **Training Focus:** Training emphasizes how the system provides a unified and professional alternative to generalized social media, highlighting the efficiency of the domain-specific UI rendering and the secure, private feedback channels.

6.2.2 Training on the Application Software

Training on the Cloud-Based Portfolio Platform application software is a crucial part of the implementation phase, designed to ensure all users can effectively leverage the system's portfolio management features. Training must be customized for each target audience to maximize efficiency and adoption.

Training for Admin Staff

Training for Admin personnel is vital, as they manage the system's integrity and quality:

- **Content Moderation:** Staff must be trained on reviewing and moderating new profile content (images, videos, audio) to ensure it meets community guidelines and quality standards.
- **User and Role Management:** Training focuses heavily on managing user roles, verifying professional accounts, and handling user-submitted feedback, reports, or suggestions.
- **System Health:** Staff learn to use the admin dashboard to monitor system-wide activity, review feedback logs, and maintain overall system health and performance.

Training for Creators (Musicians & Cinematographers)

Training for creators emphasizes the simplicity and professional presentation the system offers:

- **Profile Creation and Management:** Training covers how to register, build a dynamic profile, and efficiently upload various media types (audio tracks, video reels, photo sets) to the integrated cloud storage.
- **Categorization and Branding:** Users learn to efficiently categorize their profession (musician, cinematographer), style (e.g., contemporary music, documentary filmmaking), and experience level to ensure their profile is rendered with the correct, optimized UI.
- **Feedback and Interaction:** Creators are trained on the straightforward process of viewing and managing the private suggestions and appreciation messages submitted by viewers, fostering professional feedback without public spam.

Overall training emphasizes how the new system resolves the fragmentation and lack of specialization on other platforms, highlighting the ease of creating a professional, multimedia-rich portfolio and the value of a dedicated, secure space for professional discovery.

6.2.3 System Maintenance

System Maintenance is a planned and ongoing process essential for ensuring the long-term reliability, security, and sustained high performance of the deployed Cloud-Based Portfolio Platform application. This phase is continuous and aims to prevent system degradation and adapt the application to changing operational needs and user expectations.

6.2.4 Maintenance Procedures

The maintenance strategy for this MERN stack-based project includes:

- **Monitoring and Optimization:** Continuously monitoring the performance of the **Node.js** backend, the **MongoDB** database, and media delivery from cloud storage (**Cloudinary**/**Firebase**) to identify bottlenecks, resource issues, or slow query times, ensuring fast page loads for media-rich portfolios.
- **Security and Updates:** Regularly applying security patches and updating the application's dependencies, including the core MERN libraries, cloud SDKs, and any third-party packages, to mitigate vulnerabilities and maintain system stability.
- **Corrective Maintenance:** Promptly addressing and fixing any bugs, errors, or defects discovered by users (e.g., a UI rendering issue on a specific browser) or through routine monitoring after the system is live.

- **Adaptive Maintenance:** Modifying the system to adapt to changes in the operating environment, such as new browser versions, mobile device resolutions, or updates to external APIs (e.g., changes in the Cloudinary API).
- **Perfective Maintenance:** Refining and enhancing the system's performance, usability, and maintainability based on creator feedback and operational data. This could include optimizing the logic for the **domain-specific UI rendering**, improving the speed of media uploads, or implementing planned features like **AI-driven collaboration matching** and **analytics dashboards**.

Effective maintenance ensures that the system continues to function correctly, providing a smooth, professional, and reliable experience for Creators and Viewers over time.

6.1.1 Hosting Explanation

Eg. Render (for Frontend and Backend)

Explanation

For a MERN stack application, **Render** can be used to host all components of the project: the **React.js frontend**, the **Node.js/Express.js backend**, and the **MongoDB database** (or it can connect to an external one like MongoDB Atlas).

Render simplifies deployment by handling both static sites and dynamic web services from a single platform. The React frontend is deployed as a **"Static Site,"** which Render optimizes for fast global delivery. The Node.js backend is deployed as a **"Web Service,"** which runs the server code. This setup allows for continuous deployment (CI/CD) by connecting directly to a Git repository (like GitHub).

Procedure for hosting a MERN website on Render:

- **Step 1: Database Setup**
  - Create a "Database" service for MongoDB on Render (if available) or create a free cluster on **MongoDB Atlas** (a common choice).
  - Get the database connection URI (the connection string).
- **Step 2: Backend Deployment (Web Service)**
  - Sign up for Render and create a new **"Web Service"**.
  - Connect your GitHub repository for the **backend (Node.js/Express.js)**.
  - Set the build command (e.g., `npm install`) and the start command (e.g., `node server.js`).
  - In the **Environment Variables** section, add your `DATABASE_URL` (from Step 1), `JWT_SECRET`, and `CLOUDINARY_URL` API keys.
  - Deploy the service and copy its live URL (e.g., `https://creator-portfolio-api.onrender.com`).
- **Step 3: Frontend Deployment (Static Site)**
  - Create a new **"Static Site"** on Render.
  - Connect your GitHub repository for the **frontend (React.js)**.
  - Set the build command (e.g., `npm run build`) and the publish directory (e.g., `build`).

Creator Portfolio Platform

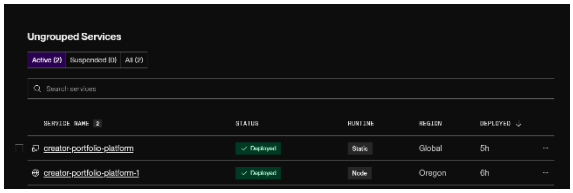
- Go to the **Environment Variables** section for this static site. Add a variable named `REACT_APP_API_URL` (or `VITE_API_URL` if using Vite) and set its value to your live backend URL from Step 2.
- Deploy the static site.

- **Step 4: Final Configuration (CORS)**
  - Go back to your backend **Web Service** settings on Render.
  - Ensure your backend's CORS (Cross-Origin Resource Sharing) configuration is set to allow requests from your new frontend URL (e.g., `https://creator-portfolio.onrender.com`).

Hosted Website:

- **Hosted Link (Frontend):** `https://creator-portfolio.onrender.com`
- **Hosted Link (Backend API):** `https://creator-portfolio-api.onrender.com`

Hosted Link QR Code



Screenshot

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The **Cloud-Based Portfolio Platform for Cinematographers and Musicians** project successfully completed its initial development phase, demonstrating the technical and operational viability of its core objective: to provide a unified, professional, and structured space for creative talent to manage their brand and connect with opportunities. By adhering to the architectural plan utilizing the robust **MERN Stack** (MongoDB, Express.js, React.js, and Node.js) with integrated cloud storage, the system has established a stable and integrated foundation that effectively resolves the identified fragmentation and lack of specialization in existing platforms. The successful execution of System Testing and Validation confirms that all system components function cohesively to meet the initial requirements.

The implemented core functionalities, which include the creation of dynamic, **multimedia-rich public profiles** (supporting images, video, and audio), the unique **domain-specific UI rendering** to optimize layouts for each creator type, and the secure **private feedback mechanism**, confirm the project's capability to bridge the gap between talent and opportunity. Crucially, the system incorporates creator-centric design and secure communication channels designed to foster a professional community: it allows creators to showcase their work without exposure to public spam and provides a clear, structured presentation for clients and collaborators. These features are key to ensuring the platform not only showcases talent but also ensures the sustainability of its creative user base.

In conclusion, the platform has achieved its initial objective by delivering a functional, efficient, and creator-centric application that directly addresses the disorganization inherent in generalized social media spaces. The successful development and validation of this foundation make the system ready to support the advanced features planned for its future expansion. The project is now poised to move into its next phase, where it will integrate AI-driven matching and advanced collaborative tools to evolve into a full-scale creative ecosystem.

7.2 FUTURE SCOPE

The Cloud-Based Portfolio Platform project has successfully delivered a stable MERN stack foundation and validated core portfolio-building workflows, but its full potential is realized in the planned expansion to a full-scale, enterprise-grade creative ecosystem. The future scope is heavily defined by integrating advanced technology and specialized professional tools to enhance both discovery and collaboration. A critical planned addition is **AI-driven gig and collaboration matching**, which will enable the system to analyze creator profiles (style, experience, location) and client needs to provide intelligent, automated recommendations, connecting the right talent with the right opportunity.

The Main Project Phase will introduce multiple specialized modules to digitize and streamline the entire creative-to-client pipeline. New additions will include a **Client & Recruiter Module**, allowing industry professionals to post job listings, search for talent using advanced filters, and manage hiring workflows. This is complemented by **Project-Based Virtual Workspaces**, a feature enabling creators (e.g., a musician and a cinematographer) to collaborate in a private, shared space with file sharing, milestone tracking, and integrated communication tools.

Finally, the scope includes comprehensive analytics and e-commerce capabilities. This involves implementing an **Analytics Dashboard** for creators to track profile views, engagement metrics, and follower growth, providing tangible data on their professional development. Technologically, the platform is prepared to integrate **E-commerce Functionalities**, allowing creators to sell services, digital downloads (like audio tracks or video presets), or prints directly from their profiles. A **Real-time Chat Module** will also be introduced to enable instant messaging between collaborators within a workspace or between clients and creators, ensuring rapid communication for professional engagements.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- Book
- ..
- ..
- ..
- ...

WEBSITES:

- ..
- ..
- ..
- ..

CHAPTER 9
APPENDIX

9.1 Sample Code

Index.js

```
import express from 'express';
import authRoutes from "./authRoutes.js";
import messageRoutes from "./messageRoutes.js";
import creatorRoutes from "./creatorRoutes.js";
import adminRoutes from "./adminRoutes.js";

const router = express.Router();

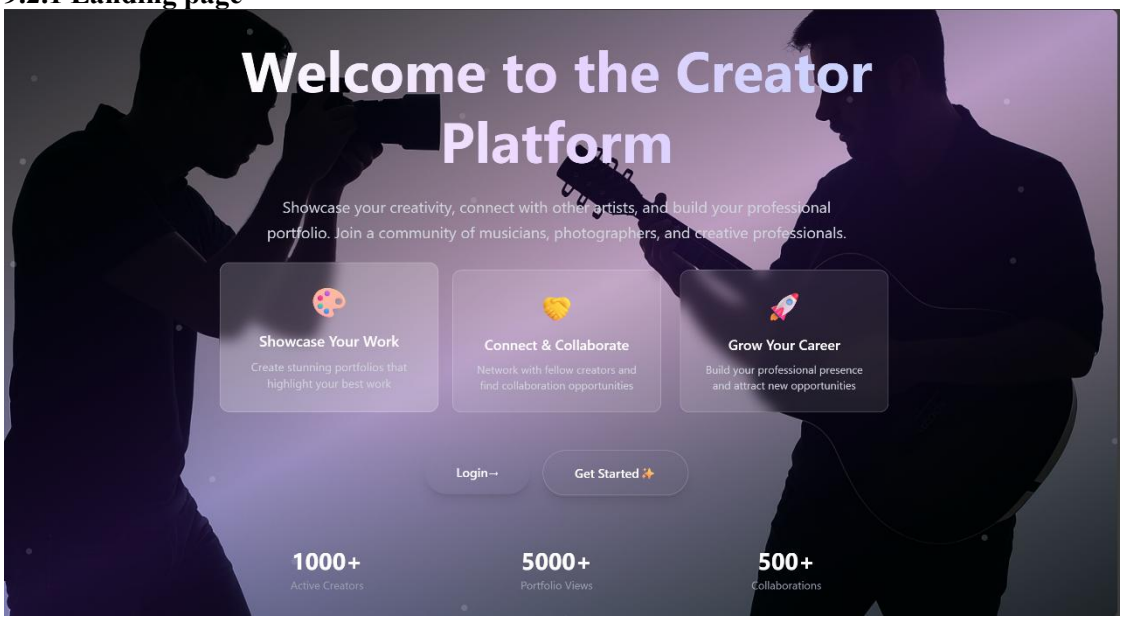
router.use('/auth', authRoutes);
router.use('/messages', messageRoutes);
router.use('/creators', creatorRoutes);
// Alias to support clients calling singular path
router.use('/creator', creatorRoutes);
router.use('/admin', adminRoutes);

router.get('/', (req, res) => {
  res.send('API is working!');
});

export default router;
```

9.2 Screenshots

9.2.1 Landing page



9.3 Git Log

Commits on Oct 23, 2025

- Update client API URLs to use environment variables for deployment

noble-joseph
committed4 days ago

- Update CORS configuration to allow Render deployment origins

noble-joseph
committed4 days ago

- Fix API base URL for production deployment

noble-joseph
committed4 days ago

- Fix build scripts and copy client dist to server public for deployment

noble-joseph
committed4 days ago
Commits on Oct 22, 2025

- Fix build script to run server build for Render deployment

noble-joseph
committedlast week
Commits on Oct 21, 2025

- Add build script to root package.json for client build

noble-joseph
committedlast week

- Fix deployment issues: install server deps in build, fix missing imports, downgrade Express to v4

- for compatibility

noble-joseph
committedlast week

- fix: add react-icons to client dependencies

noble-joseph
committedlast week
Commits on Oct 20, 2025

- Update CORS to use FRONTEND_URL env var for production

noble-joseph
committedlast week

- Fix syntax error in PublicProfile.jsx

noble-joseph
committedlast week

- Update CORS to use env var for production

noble-joseph
committedlast week

- Configure for Render deployment: add build script, serve static client, update API base

noble-joseph
committedlast week

- before hosting

noble-joseph
committedlast week
Commits on Sep 25, 2025

- after landing page ui

noble-joseph
committedon Sep 25
Commits on Sep 18, 2025

- cloudanry setup

noble-joseph
committedon Sep 18

- connections

noble-joseph
committedon Sep 18
Commits on Sep 17, 2025

- Initial commit: Creator Portfolio Platform

noble-joseph
committedon Sep 17
Commits on Sep 15, 2025

- Resolve merge conflicts and complete merge from origin/main aafter connections

noble-joseph
committedon Sep 15

- after implementing connections

noble-joseph
committedon Sep 15
Commits on Sep 9, 2025

- commit before changing profile photo alignment

noble-joseph
committedon Sep 9

- 2nd commit after adding portfolio

noble-joseph
committedon Sep 9

- 2nd commit after adding portfolio

noble-joseph

Creator Portfolio Platform

committedon Sep 9
Commits on Sep 7, 2025
- Initial commit with full project structure

noble-joseph
committedon Sep 7
Commits on Aug 11, 2025
- Add files via upload

noble-joseph
authoredon Aug 11
Commits on Aug 6, 2025
- Initial commit

noble-joseph
authoredon Aug 6