Requirements Document

Freelancing Application

Freelancing Tool User Stories

Must Have Features.

a. **Account Management**

1 As a freelancer, I want to create an account so that I can access the platform and find work.

2 As a freelancer, I want to log in to my account so that I can manage my profile and job applications.

b. **Profile Management**

3 As a freelancer, I want to create and update my profile so that clients can see my skills, experience, and portfolio.

4 As a freelancer, I want to upload my resume and portfolio pieces so that clients can evaluate my work.

c. **Job Search and Application**

5 As a freelancer, I want to search for jobs by category, skills, and location so that I can find suitable opportunities.

6 As a freelancer, I want to view detailed job descriptions so that I understand the requirements and expectations.

7 As a freelancer, I want to apply for jobs directly through the platform so that I can efficiently seek work.

d.**Communication and Collaboration**

8 As a freelancer, I want to send and receive messages from clients so that I can discuss project details and requirements.

9 As a freelancer, I want to share files and documents with clients so that we can collaborate effectively.

e. **Project Management**

10 As a freelancer, I want to view and manage my ongoing projects so that I can keep track of my work.

11 As a freelancer, I want to update project statuses and milestones so that clients are informed of progress.

f. **Reviews and Ratings**

12 As a freelancer, I want to receive feedback and ratings from clients so that I can improve my services and build my reputation.

13 As a freelancer, I want to view my overall rating and client reviews so that I can understand my performance.

Client User Stories

a. **Account Management**

14 As a client, I want to create an account so that I can hire freelancers for my projects.

15 As a client, I want to log in to my account so that I can manage my job postings and projects.

b. **Job Posting**

16 As a client, I want to create and post job listings so that I can find freelancers for my projects.

17 As a client, I want to provide detailed job descriptions, requirements, and budgets so that freelancers understand what I need.

c. **Freelancer Search and Hiring**

18 As a client, I want to search for freelancers by skills, experience, and location so that I can find the best candidates for my projects.

19 As a client, I want to view freelancer profiles and portfolios so that I can evaluate their qualifications.

d. **Communication and Collaboration**

20 As a client, I want to send and receive messages from freelancers so that we can discuss project details and requirements.

Nice To Have Features

21 As a client, I want to share files and documents with freelancers so that we can collaborate effectively.

e. **Payment and Invoicing**

22 As a client, I want to receive invoices from freelancers so that I can process payments.

23 As a client, I want to make payments to freelancers securely through the platform so that I can ensure timely compensation.

f. **Reviews and Ratings**

24 As a client, I want to provide feedback and ratings for freelancers so that I can help them improve and inform other clients.

25 As a client, I want to view my overall rating and freelancer reviews so that I can understand my hiring reputation.

Additional Advanced Features

a. **Mobile Access**

26 Mobile-responsive design

27 Mobile apps for iOS and Android

28 Push notifications for updates and alerts

b. **Virtual Collaboration Tools**

29 Integrated video conferencing

30 Real-time collaboration tools (document editing, whiteboards)

3. **Networking and Community**

31 Forums and discussion boards

32 Networking events and webinars

33 Freelancer and client matching based on profiles

4. **Advanced Security Features**

34 Two-factor authentication

35 Data encryption

36 Fraud detection and prevention tools

Tasks List Based on CRUD Functionality Including Database

A. Account Management

Freelancer:

- Create Freelancer Account (Create)

Frontend:

- Design and implement account creation form UI.
- Implement form validation for required fields (email, password).

Backend:

- Create API endpoint for user registration.
- Hash passwords before storing them in the database.
- Send email verification.

Database:

- Design and implement Users table with fields (id, name, email, password_hash, role, created_at, verified).
- Login (Read)

Frontend:

- Design and implement login form UI.
- Implement "Forgot Password" flow UI.

Backend:

- Create API endpoint for user authentication.
- Implement session management (JWT or OAuth).
- Handle password reset via email.

Database:

- Query the Users table to verify credentials.
- Update Users table with session tokens or authentication details.
- Client: 14. Create Client Account (Create)

Frontend:

- Design and implement account creation form UI specific to clients.

Backend:

- Create API endpoint for client registration.
- Send email verification.

Database:

- Add client-specific fields to the Users table if needed.
- Client Login (Read)

Frontend:

- Design and implement client-specific login UI.

Backend:

- Implement client authentication and session management.

Database:

- Query the Users table to verify credentials for clients.

B. Profile Management

- Freelancer: 3. Create and Update Profile (Create/Update)

Frontend:

- Design profile creation/editing UI.

Backend:

- Create API endpoint for profile creation and updates.

Database:

- Design and implement Profiles table linked to Users with fields (user_id, skills, experience, bio, etc.).
- Implement UPDATE operations for editing profiles.

- Upload Resume and Portfolio (Create/Read)

Frontend:

Design UI for uploading resume and portfolio files.

Backend:

Create API endpoints for file upload and retrieval.

Database:

Implement file storage solution (e.g., AWS S3, Google Cloud Storage).

Link uploaded files to user profiles in the Profiles table (resume_url, portfolio_urls).

Client: 19. View Freelancer Profiles (Read)

Frontend:

Design UI for searching and viewing freelancer profiles.

Backend:

Create API endpoint to retrieve freelancer profiles based on search criteria.

Database:

Query the Profiles table and return relevant profiles.

C. Job Search and Application

Freelancer: 5. Search Jobs (Read)

Frontend:

Design UI for job search with filters (category, skills, location).

Backend:

Create API endpoints to retrieve job listings based on filters.

Database:

Design and implement Jobs table with fields (id, title, category, skills_required, location, description, client_id).

Perform READ operations to return filtered job results.

View Job Descriptions (Read)

Frontend:

Design UI to display detailed job descriptions.

Backend:

Create API endpoint to retrieve job details.

Database:

Query the Jobs table to retrieve job description data.

Apply for Jobs (Create)

Frontend:

Design UI for submitting job applications.

Backend:

Create API endpoint for job applications.

Database:

Design and implement Applications table with fields (id, job_id, freelancer_id, application_date, status).

Store new application entries.

Client: 16. Post Job Listings (Create)

Frontend:

Design UI for job posting with detailed fields.

Backend:

Create API endpoint for job creation.

Database:

Store job listings in the Jobs table.

Input Job Details (Update)

Frontend:

Design UI for updating job details (requirements, budget).

Backend:

Create API endpoint to update job details.

Database:

Implement UPDATE operations on the Jobs table.

D. Communication and Collaboration

Freelancer & Client: 8 & 20. Messaging (Create/Read)

Frontend:

Design UI for messaging between freelancers and clients.

Backend:

Create API endpoints for sending and retrieving messages.

Database:

Design and implement Messages table with fields (id, sender_id, receiver_id, message_text, timestamp).

Implement CREATE and READ operations for messages.

9 & 21. File Sharing (Create/Read)

Frontend:

Design UI for uploading and sharing files.

Backend:

Create API endpoints for file uploads and retrieval.

Database:

Store file metadata in the FileShares table (id, sender_id, receiver_id, file_url, timestamp).

Implement file storage solution and READ operation to retrieve shared files.

E. Project Management

Freelancer: 10. Track Projects (Read)

Frontend:

Design dashboard UI for project tracking.

Backend:

Create API endpoint to retrieve project data.

Database:

Design and implement Projects table with fields (id, freelancer_id, client_id, title, status, start_date, end_date).

Query the Projects table for ongoing projects.

Update Project Status (Update)

Frontend:

Design UI for updating project statuses and milestones.

Backend:

Create API endpoint for updating project details.

Database:

Implement UPDATE operations on the Projects table.

Client: 22. Manage Invoices (Create/Read)

Frontend:

Design UI for viewing and managing invoices.

Backend:

Create API endpoint to retrieve and store invoice data.

Database:

Design and implement Invoices table with fields (id, project_id, freelancer_id, client_id, amount, status, due_date).

Perform CREATE and READ operations for invoices.

Process Payments (Create)

Frontend:

Design UI for payment processing.

Backend:

Integrate payment gateway API (e.g., Stripe, PayPal).

Database:

Store payment transaction records in the Payments table (id, invoice_id, amount, status, payment_date).

F. Reviews and Ratings

Freelancer: 12. Collect Feedback (Create)

Frontend:

Design UI for clients to submit feedback.

Backend:

Create API endpoint to store feedback data.

Database:

Design and implement Feedback table with fields (id, project_id, freelancer_id, client_id, rating, comments, timestamp).

Store new feedback entries.

View Ratings and Reviews (Read)

Frontend:

Design UI for viewing ratings and reviews.

Backend:

Create API endpoint to retrieve feedback data.

Database:

Query the Feedback table to display freelancer ratings and reviews.

Client: 24. Provide Feedback (Create)

Frontend:

Design UI for submitting freelancer feedback.

Backend:

Create API endpoint to store client feedback.

Database:

Store feedback in the Feedback table.

View Ratings and Reviews (Read)

Frontend:

Design UI for clients to view their ratings and feedback from freelancers.

Backend:

Create API endpoint to retrieve feedback data.

Database:

Query the Feedback table to display client ratings and reviews.

Additional Advanced Features

Mobile Access:

26. Mobile-Responsive Design (Read)

Frontend:

Ensure all UI components are mobile-responsive.

Backend:

No specific tasks.

Database:

No specific tasks.

Develop Mobile Apps (Create/Read)

Frontend:

Develop native mobile apps for iOS and Android.

Implement push notifications.

Backend:

Create API endpoints for mobile app data access.

Database:

Ensure data is accessible through mobile endpoints.

Push Notifications (Create)

Frontend:

Design notification UI for mobile and web.

Backend:

Integrate push notification service (e.g., Firebase Cloud Messaging).

Database:

Store notification data in the Notifications table (id, user_id, message, timestamp).

Virtual Collaboration Tools:

29. Video Conferencing Integration (Create)

Frontend:

Design UI for initiating video calls.

Backend:

Integrate video conferencing API (e.g., Zoom, Google Meet).

Database:

Store video call metadata in the Calls table (id, project_id, participants, timestamp).

Real-Time Collaboration Tools (Create/Update)

Frontend:

Design UI for real-time document editing and whiteboards.

Backend:

Implement real-time collaboration functionality using WebSockets or similar technologies.

Database:

Store collaboration data (e.g., document changes) in the Collaborations table.

Networking and Community: 31. Forums and Discussion Boards (Create/Read/Update)

Frontend:

Design UI for forums and discussion boards.

Breaking down the tasks into smaller steps helps ensure that every part of the development process is clearly defined and manageable. Below is a more granular breakdown for some of the tasks across different features:

A. Account Management

1. **Create Account Form (Freelancer and Client)**
    - **Frontend:**
        1. Design the registration form (name, email, password, confirm password).
        2. Add form validation (e.g., email format, password strength, password match).
        3. Create a "Create Account" button.
        4. Add submit button functionality to handle form submission.
        5. Display error messages for invalid inputs.
        6. Redirect to a confirmation screen or dashboard upon successful registration.
    - **Backend:**
        1. Create API endpoint /register to handle account creation.
        2. Hash the password before saving.
        3. Validate email uniqueness in the database.
        4. Store the new user record in the Users collection.
        5. Send a verification email (optional).
    - **Database:**
        1. Ensure Users collection is indexed on email for fast lookup.
2. **Login Form (Freelancer and Client)**
    - **Frontend:**
        1. Design the login form (email, password).
        2. Add a "Log In" button.
        3. Implement client-side form validation.
        4. Display error messages for incorrect credentials.

5. Redirect to the dashboard upon successful login.

- **Backend:**

  1. Create API endpoint /login to handle authentication.
  2. Verify email and password against stored records.
  3. Create and return a session token on successful login.

- **Database:**

  1. Store session tokens in the Users collection for session management.

B. Profile Management

3. **Create/Update Profile Form (Freelancer)**

   - **Frontend:**

     1. Design the profile form (skills, experience, bio).
     2. Add an "Upload Resume" button for attaching a resume file.
     3. Add fields to input portfolio URLs.
     4. Create a "Save Profile" button.
     5. Implement form validation (e.g., required fields, file format check).
     6. Add a preview option to see how the profile will look to clients.

   - **Backend:**

     1. Create API endpoint /profile to create or update profile data.
     2. Handle file uploads for the resume and portfolio files.
     3. Validate and store profile data in the Profiles collection.

   - **Database:**

     1. Store resume file URLs and portfolio links in the Profiles collection.

C. Job Search and Application

4. **Job Search Page**
    - **Frontend:**
        1. Create a search bar with filters (e.g., category, skills, location).
        2. Design the job listing cards to display brief job information (title, category, budget).
        3. Add pagination or infinite scroll to load more jobs.
        4. Implement sorting options (e.g., by date, budget).
        5. Add "Apply Now" buttons for each job listing.
    - **Backend:**
        1. Create API endpoint /jobs to fetch filtered job listings.
        2. Implement search and filter logic based on the provided criteria.
    - **Database:**
        1. Ensure Jobs collection has indexes on searchable fields (e.g., category, skills_required).
5. **Job Application Form**
    - **Frontend:**
        1. Create a form for the job application (cover letter, attach files if needed).
        2. Add an "Apply" button.
        3. Display confirmation message upon successful application submission.
    - **Backend:**
        1. Create API endpoint /apply to handle job applications.
        2. Validate the application data (e.g., check if the user has already applied).

        3. Store the application in the Applications collection.

   - **Database:**

        1. Ensure Applications collection is properly linked to the Jobs and Users collections.

D. Communication and Collaboration

6. **Messaging System**

   - **Frontend:**

        1. Design a messaging interface with a list of conversations and a chat window.
        2. Add an input box for typing messages.
        3. Implement a "Send" button.
        4. Display conversation history in the chat window.
        5. Add real-time message updates using WebSockets or polling.

   - **Backend:**

        1. Create API endpoints /messages to send and receive messages.
        2. Implement WebSocket support for real-time messaging.
        3. Store messages in the Messages collection.

   - **Database:**

        1. Index the Messages collection by sender_id and receiver_id for quick lookups.

7. **File Sharing**

   - **Frontend:**

        1. Add a "Share File" button to the chat interface.
        2. Allow file selection and display the selected file name.
        3. Implement file upload progress indicator.

4. Display shared files in the chat history.

- **Backend:**
    1. Create API endpoint /share-file to handle file uploads.
    2. Store file metadata in the FileShares collection.
- **Database:**
    1. Ensure the FileShares collection is indexed on sender_id and receiver_id.

E. Project Management

8. **Project Dashboard**
    - **Frontend:**
        1. Design a dashboard to list ongoing projects with statuses.
        2. Add options to filter projects by status (e.g., "In Progress", "Completed").
        3. Display milestones and deadlines for each project.
        4. Add buttons to update project statuses and mark milestones as completed.
    - **Backend:**
        1. Create API endpoint /projects to fetch and update project data.
        2. Implement logic to update project statuses and milestones.
    - **Database:**
        1. Ensure Projects collection stores milestone data and project statuses.
9. **Invoice Management**
    - **Frontend:**
        1. Design a form to create invoices (amount, due date).

2. Add a "Send Invoice" button.
3. Display invoice history with statuses (e.g., "Pending", "Paid").
4. Implement a payment confirmation system for clients.

- **Backend:**
  1. Create API endpoint /invoices to create and manage invoices.
  2. Implement payment processing logic (integrate with Stripe or PayPal).
- **Database:**
  1. Ensure Invoices collection stores payment status and timestamps.

F. Reviews and Ratings

10. **Feedback Form**
    - **Frontend:**
      1. Design a form to provide feedback (rating, comments).
      2. Add a "Submit Feedback" button.
      3. Display feedback history and overall ratings.
    - **Backend:**
      1. Create API endpoint /feedback to handle feedback submission.
      2. Store feedback in the Feedback collection.
    - **Database:**
      1. Index the Feedback collection by freelancer_id and client_id for efficient querying.

Advanced Features

11. **Two-Factor Authentication (2FA)**
    - **Frontend:**
      1. Design a 2FA setup page (e.g., QR code for Google Authenticator).

    2. Add a field to input the 2FA code during login.

  - **Backend:**
    1. Implement 2FA generation and validation using a 2FA provider.
    2. Store 2FA secret in the Users collection.
  - **Database:**
    1. Ensure the Users collection has a field for storing 2FA data.

12. **Real-Time Collaboration Tools**
  - **Frontend:**
    1. Design an interface for real-time document editing and whiteboarding.
    2. Add collaboration tools (e.g., drawing tools, text editing).
  - **Backend:**
    1. Implement real-time data synchronization using WebSockets.
    2. Store collaboration data in the Collaborations collection.
  - **Database:**
    1. Ensure the Collaborations collection can store real-time updates for projects.

Refined Steps for the user stories:

A. Account Management

1. Create Account (Freelancer and Client)

- **Frontend:**
  1. **Design the registration form** in React with fields:
     - Name

        - Email
        - Password
        - Role selection (Freelancer/Client)
    2. **Add form validation** using a library like Formik or React Hook Form.
    3. **Create a Submit button** to trigger form validation and make an API call to the backend.
    4. **Handle success/failure** by showing appropriate messages (e.g., “Account created successfully” or “Email already exists”).
    5. **Implement navigation** to redirect to the login page or email verification page.

- **Backend:**
    1. **Set up an Express route (/api/auth/register)** to handle account creation.
    2. **Validate input**:
        - Check for existing email.
        - Ensure password meets security requirements.
    3. **Hash the password** using bcrypt.
    4. **Save the user data** (e.g., name, email, hashed password, role) to the Users collection in MongoDB.
    5. **Send a verification email** using Nodemailer (optional).

- **Database:**
    1. **Create the Users collection** with fields:
        - name
        - email
        - password_hash
        - role (freelancer or client)

        - created_at
        - verified (optional)
    2. **Index the email field** to ensure uniqueness.(optional)

2. Login (Freelancer and Client)

- **Frontend:**
    1. **Design the login form** with fields:
        - Email
        - Password
    2. **Add form validation** for required fields.
    3. **Create a Submit button** that triggers form validation and sends data to the backend.
    4. **Handle login errors** (e.g., invalid credentials) and display appropriate error messages.
    5. **Redirect users** to the dashboard after successful login.
- **Backend:**
    1. **Set up an Express route (/api/auth/login)** to handle login requests.
    2. **Validate user credentials**:
        - Find the user by email.
        - Compare the hashed password using bcrypt.
    3. **Generate a JWT token** and send it back to the frontend.
    4. **Store the JWT** in cookies or local storage (depending on your authentication strategy).
- **Database:**
    1. **Ensure Users collection** is indexed by email.
    2. **No new collection needed** for basic login functionality.

B. Profile Management (Freelancer)

3. Create/Update Profile

- **Frontend:**
    1. **Design the profile form** in React with fields:
        - Skills
        - Experience
        - Bio
        - Upload Resume/Portfolio
    2. **Handle file uploads** using a file uploader like React Dropzone or FilePond.
    3. **Create a Save Profile button** that validates the form and sends data to the backend.
    4. **Display success/failure notifications** after the profile is updated.
    5. **Implement preview** for uploaded files (e.g., images, PDFs).
- **Backend:**
    1. **Set up an Express route (/api/profile)** to handle POST/PUT requests for creating and updating profiles.
    2. **Handle file uploads**:
        - Integrate with cloud storage (e.g., Cloudinary or AWS S3) for file management.
    3. **Store profile data** (e.g., skills, experience, bio) in the Profiles collection.
    4. **Validate file uploads** to ensure they meet size and format requirements.
- **Database:**
    1. **Create the Profiles collection** with fields:
        - user_id
        - skills

        - experience
        - bio
        - resume_url
        - portfolio_urls

    2. **Ensure the Profiles collection** is indexed by user_id for quick lookup.

C. Job Search and Application

4. Job Search Page

- **Frontend:**
    1. **Design the job search page** with filters:
        - Category
        - Skills
        - Location
        - Budget
    2. **Display job listings** in a grid or list layout using React components.
    3. **Implement pagination or infinite scroll** to load more jobs dynamically.
    4. **Add sorting options** (e.g., by date, relevance).
    5. **Create an Apply Now button** that navigates to the job application page.
- **Backend:**
    1. **Set up an Express route (/api/jobs)** to handle GET requests for job listings.
    2. **Implement filtering and sorting** based on query parameters (e.g., category, skills, location).
    3. **Handle pagination** by returning a limited set of jobs per request.
    4. **Ensure proper security** by allowing only authenticated users to access job listings.

- **Database:**
    1. **Create the Jobs collection** with fields:
        - title
        - category
        - skills_required
        - description
        - client_id
        - budget
    2. **Ensure indexing** on searchable fields like category, skills_required, and location for performance.

5. Job Application Form

- **Frontend:**
    1. **Design the application form** with fields:
        - Cover letter
        - File uploads (if required)
    2. **Add form validation** to ensure required fields are completed.
    3. **Create an Apply button** that submits the form to the backend.
    4. **Show confirmation messages** after the application is successfully submitted.
- **Backend:**
    1. **Set up an Express route (/api/apply)** to handle POST requests for job applications.
    2. **Validate application data**:
        - Ensure the user hasn't applied for the same job multiple times.
    3. **Store application data** in the Applications collection.

4. **Send a confirmation email** or notification to the freelancer and client (optional).

- **Database:**
    1. **Create the Applications collection** with fields:
        - job_id
        - freelancer_id
        - cover_letter
        - application_date
        - status
    2. **Index the job_id and freelancer_id fields** for quick lookup.

D. Communication and Collaboration

6. Messaging System

- **Frontend:**
    1. **Design the chat interface**:
        - Chat window with conversation list and message input box.
    2. **Implement real-time messaging** using WebSockets (Socket.io) for live chat.
    3. **Add a Send button** to send messages to the backend.
    4. **Display conversation history** and update it in real-time when new messages arrive.
- **Backend:**
    1. **Set up an Express route (/api/messages)** to handle message sending and retrieval.
    2. **Implement WebSocket support** with Socket.io to enable real-time messaging.

3. **Store messages** in the Messages collection with fields like sender_id, receiver_id, message_text, timestamp.
4. **Add message read/unread status** to track message state.

- **Database:**
    1. **Create the Messages collection** with fields:
        - sender_id
        - receiver_id
        - message_text
        - timestamp
        - read_status
    2. **Index the sender_id and receiver_id** for efficient query performance.

E. Project Management

7. Project Dashboard

- **Frontend:**
    1. **Design the project dashboard** that lists ongoing projects with statuses and milestones.
    2. **Create React components** for each project card with details like title, status, deadline.
    3. **Add filtering options** to sort by deadline, status, etc.
    4. **Implement a View Details button** to navigate to a detailed project page.
- **Backend:**
    1. **Set up an Express route (/api/projects)** to handle GET requests for fetching projects.
    2. **Implement filtering** by status, client, or freelancer.
    3. **Handle pagination** for projects to avoid large payloads.

- **Database:**
    1. **Create the Projects collection** with fields:
        - title
        - client_id
        - freelancer_id
        - description
        - status
        - milestones

8. Update Project Status and Milestones

- **Frontend:**
    1. **Design a detailed project page** that displays all project information, including milestones, deadlines, and status updates.
    2. **Create an Update Status button** that opens a modal or form for changing the project status (e.g., In Progress, Completed).
    3. **Create a form to update milestones** where freelancers can add, edit, or mark milestones as complete.
    4. **Implement form validation** to ensure correct data input for status and milestone updates.
    5. **Show success/failure notifications** upon successful or failed updates.
- **Backend:**
    1. **Set up an Express route (/api/projects/update)** to handle PUT requests for updating project statuses and milestones.
    2. **Validate milestone updates** to ensure they match the project timeline and prevent invalid transitions (e.g., marking a milestone as complete before its due date).

3. **Update project status and milestone data** in the Projects collection based on user input.
4. **Notify clients** via email or in-app notifications when milestones are updated or project status changes.

- **Database:**
  1. **Update the Projects collection** to include milestones as an array of objects, such as:
     - milestone_name
     - due_date
     - completed (boolean)
  2. **Ensure the collection is indexed** by project_id and freelancer_id to optimize query performance during updates.

F. Reviews and Ratings

9. Leave a Review and Rating

- **Frontend:**
  1. **Design a review form** with input fields for star ratings and a text review.
  2. **Add form validation** to ensure the review has at least the minimum required fields (e.g., star rating and review text).
  3. **Create a Submit Review button** that triggers form validation and sends the data to the backend.
  4. **Show confirmation messages** after successful submission, or handle errors if submission fails.
- **Backend:**
  1. **Set up an Express route (/api/reviews)** to handle POST requests for submitting reviews and ratings.
  2. **Validate review submissions**:

        - Ensure the project is completed before allowing reviews.
        - Prevent multiple reviews for the same project from the same user.
    3. **Store the review data** in the Reviews collection, associating it with the project, freelancer, and client.

- **Database:**
    1. **Create the Reviews collection** with fields:
        - project_id
        - freelancer_id
        - client_id
        - rating
        - review_text
        - created_at
    2. **Index the collection** by freelancer_id and client_id for efficient querying of reviews.

10. View Reviews and Ratings

- **Frontend:**
    1. **Design a review and rating page** where freelancers and clients can view their overall ratings and individual reviews.
    2. **Display the average rating** at the top of the page and list individual reviews below it.
    3. **Implement pagination** or lazy loading to retrieve reviews in batches.
    4. **Create React components** for each review, showing the rating, review text, and timestamp.
- **Backend:**
    1. **Set up an Express route (/api/reviews/:user_id)** to handle GET requests for fetching reviews and ratings for a specific freelancer or client.

2. **Calculate the average rating** for the user based on their reviews and return it along with the list of reviews.

3. **Implement pagination** on the backend to fetch reviews in chunks.

- **Database:**

    1. **Ensure the Reviews collection** is indexed for quick access by freelancer_id and client_id.

    2. **Optionally, store the average rating** as a derived field in the Profiles collection for faster retrieval during profile lookups.

G. Payment and Invoicing

11. Generate Invoice

- **Frontend:**

    1. **Design an invoice creation form** with fields:

        - Amount
        - Description
        - Due date
        - Supporting documents (optional file uploads)

    2. **Implement form validation** to ensure that all required fields are filled in.

    3. **Create a Generate Invoice button** that triggers form validation and sends the invoice data to the backend.

    4. **Display success/failure notifications** upon successful invoice generation or errors.

- **Backend:**

    1. **Set up an Express route (/api/invoices)** to handle POST requests for invoice generation.

    2. **Store the invoice data** in the Invoices collection, linking it to the freelancer, client, and project.

3. **Integrate with a payment gateway** (e.g., Stripe, PayPal) to facilitate payments once the invoice is generated.
4. **Send invoice notifications** to the client via email or in-app notifications.

- **Database:**
  1. **Create the Invoices collection** with fields:
     - invoice_id
     - project_id
     - freelancer_id
     - client_id
     - amount
     - description
     - due_date
     - status (e.g., pending, paid)
  2. **Index the collection** by client_id and freelancer_id for quick access to invoice records.

12. Make Payment

- **Frontend:**
  1. **Design a payment page** that displays invoice details and available payment options (e.g., credit card, PayPal).
  2. **Integrate with a payment gateway API** for processing payments.
  3. **Add a Pay Now button** that triggers the payment process and handles success/failure notifications.
  4. **Update the UI** based on the payment status (e.g., success, pending, failed).
- **Backend:**

1. **Set up an Express route (/api/payments)** to handle POST requests for payment processing.
2. **Integrate with the payment gateway API** (e.g., Stripe, PayPal) to securely process the payment.
3. **Update the invoice status** in the Invoices collection to reflect the payment outcome (e.g., paid, pending, failed).
4. **Send payment confirmation notifications** to both the freelancer and client via email or in-app notifications.

- **Database:**
    1. **Update the Invoices collection** to store the payment status (paid, pending, failed).
    2. **Create a Payments collection** to store payment transaction data:
        - payment_id
        - invoice_id
        - transaction_id
        - amount
        - status
        - created_at
    3. **Index the collection** by invoice_id for quick lookup of payment details.

H. Mobile Access (Advanced Feature)

13. Mobile-Responsive Design

- **Frontend:**
    1. **Implement responsive design** across the app using CSS frameworks like Bootstrap or Tailwind CSS.
    2. **Ensure all components** (e.g., forms, dashboards, modals) are optimized for different screen sizes, including mobile devices.

3. **Test the app on multiple devices** and screen sizes to ensure a consistent user experience.