|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User Story  **A. Account Management**  **1. Create Account (Freelancer and Client)** | Component   * **Frontend:** | Task   * 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. | Time  40 min |  |
|  | * **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). | 30 min |  |
|  | * **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) | 10 min |  |
| **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. | 30 min |  |
|  | * **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.(optional)   4. **Store the JWT** in cookies or local storage (depending on your authentication strategy).(optional) | 30 min |  |
|  | * **Database:** | * 1. **Ensure Users collection** is indexed by email.   2. **No new collection needed** for basic login functionality. | 15 minutes |  |
| **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). | 1.5 hr |  |
|  | * **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. | 2 hrs |  |
|  | * **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. | 30 min |  |
| **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. | 1 hr |  |
|  | * **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. | 1 hr |  |
|  | * **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. | 40 min |  |
| **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. | 40 min |  |
|  | * **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). | 1 hr |  |
|  | * **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. | 30 min |  |
| **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. | 2 hrs |  |
|  | * **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. | 1 hr |  |
|  | * **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. | 30 min |  |
| **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. | 1.5 hr |  |
|  | * **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. | 2.5 hrs |  |
|  | * **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. | 2 hrs |  |
|  | * **Database:** | * **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. | 40 min |  |
| **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. | 1.5 hr |  |
|  | * **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. | 1 hr |  |
|  | * **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. | 30 min |  |
| **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. | 2 hr |  |
|  | * **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. | 1.5 hr |  |
|  | * **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. | 30 min |  |
| **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. | 1 hr 30 min |  |
|  | * **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. | 2 hrs |  |
|  | * **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. | 1 hr |  |
| **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). | 1.5 hr |  |
|  | * **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. | 2.5 hrs |  |
|  | * **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. | 40 min |  |
| **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. | 1 hr |  |