Freelancing Application - FreelanHub

**REPORT**

**Team Members:**

Afrah Tabassum (311121104003): Full Stack Developer

Arul Prasad SP (311121104010): Full Stack Developer

Jovitha Melcy M (311121104030): Full Stack Developer

Rahul Jayaraj (311121104048): Full Stack Developer

**Project Overview:**

**Purpose:**

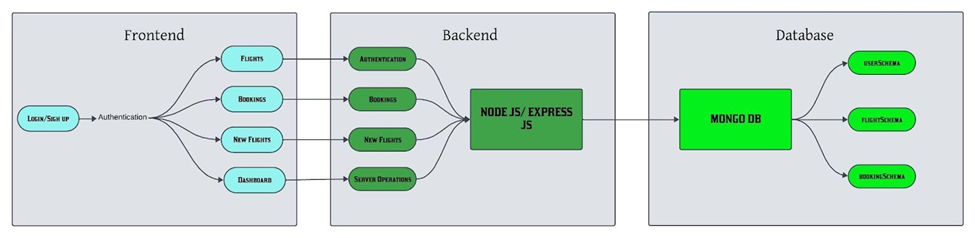
FreelanceHub is a comprehensive freelancing platform designed to bridge the gap between clients and freelancers by creating a seamless, efficient, and transparent digital experience. The platform enables clients to post diverse projects, review freelancer profiles, and select the best fit for their needs. Freelancers, in turn, can bid on projects and submit high-quality deliverables through an intuitive interface. The platform also empowers administrators with tools to ensure data integrity, oversee interactions, and resolve conflicts, ensuring smooth operations.

The ultimate goal is to provide a reliable, user-friendly environment where clients can efficiently manage their projects, freelancers can showcase their skills, and administrators maintain platform integrity, fostering collaboration and trust within the freelancing ecosystem.

**Features:**

* **User Authentication:**Secure registration and login system to ensure only authorized users can access the platform, protecting user data and privacy.
* **Project Posting and Bidding:**Clients can post detailed project descriptions, and freelancers can bid based on their expertise and capabilities, fostering a competitive and transparent selection process.
* **Freelancer Profiles and Reviews:**Clients can assess freelancers by reviewing profiles, past projects, and ratings to make informed decisions.
* **Seamless Communication:**Built-in messaging system allows clients and freelancers to communicate effectively, ask questions, and provide updates on project progress.
* **Project Submission and Feedback:**Freelancers can submit completed work directly through the platform, and clients can review, approve, and provide feedback to maintain a high standard of quality.
* **Admin Dashboard for Oversight:**Administrators can monitor platform activity, enforce policies, resolve disputes, and provide user support to ensure smooth operations.

**Architecture:**

****

FreelanceHub adopts a client-server model, using the MERN stack for efficient development and scalability.

* **Frontend:**
  + Built using **React.js** for dynamic, responsive user interfaces.
  + Styled with **Bootstrap** and **Material UI** for a visually appealing and intuitive user experience.
  + Communication with the backend is facilitated via **Axios** and RESTful APIs.
* **Backend:**
  + Developed with **Express.js**, providing robust routing, middleware, and API development.
  + Integrated with **MongoDB** for secure, scalable, and efficient data storage and retrieval.
* **Database:**
  + **MongoDB** is used to store user profiles, project data, and transaction records in a JSON-like format.
  + Managed using **Mongoose** for efficient schema definitions and data manipulation.
* **Authentication and Security:**
  + User credentials are securely stored using **bcrypt** for password hashing.
  + **CORS** is implemented to handle cross-origin resource sharing securely.

**Setup Instructions:**

**1.** **Prerequisites:**

● Node.js (v14.17.0 or higher)

● MongoDB (local or cloud-based using MongoDB Atlas)

● npm (Node Package Manager)

● React (Frontend framework)

● Two web browsers (recommended: Chrome and Firefox)

**2.** **Installation:**

● Clone the repository:  
 **git clone <repository-url>  
 cd flight-booking-app\**

● Install dependencies for the server:  
 **cd server  
 npm install**

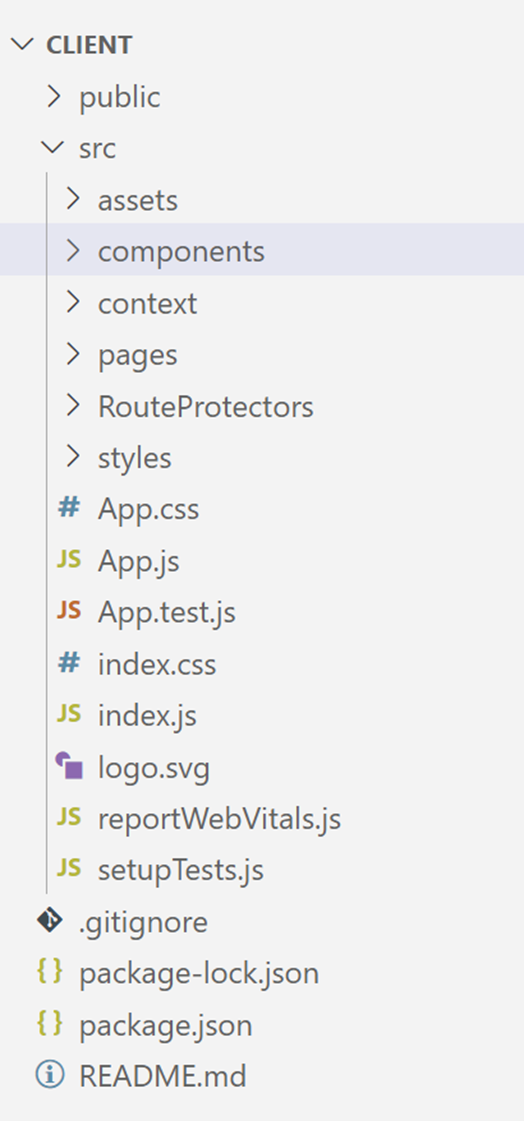
● Install dependencies for the client:  
 **cd ../client  
 npm install**

**Folder Structure:**

**1.** **Server (Node.js Backend, MongoDB Database)**

****

**2.** **Client (React js):**

****

**Running the Application:**

**1.** **Frontend:**

**●** Navigate to the client directory and run:  
 **npm start**

**2. Backend:**

**●** Navigate to the server directory and run:  
 **node index.js**

**API Documentation:**

1. **Register Endpoint:**

* **URL:** /register
* **Method:** POST
* **Description:** Registers a new user by saving their details to the database.
* **Authentication:** None.
* **Request Parameters:**
  + **Body Parameters:**
    - email (required): The email of the user.
    - password (required): The password of the user.
    - username (required): The name of the user.
    - usertype (required): The type of user (freelancer, client).
* **Response Example:**
  + Success Response (201 Created)**:**  
    {

"\_id": "654321",

"usertype": "freelancer",

"username": "Jane Doe",

"email": "[newuser@example.com](mailto:newuser@example.com)"

}

1. **Login Endpoint**

* **URL:** /login
* **Method:** POST
* **Description:** Authenticates the user by verifying their email and password.
* **Authentication:** None.
* **Request Parameters:**
* **Body Parameters:**
  + email (required): The email of the user.
  + password (required): The password of the user.
* **Response Example:**
  + Error Response (401 Unauthorized):

{

"error": "Invalid email or password"

}

1. **Login Endpoint**

* **URL**: /post-job
* **Method**: POST
* **Description**: Allows clients to post a new job listing.
* **Authentication**: Required.
* **Request Parameters:**
  + Body Parameters:
    - clientId (required): The unique ID of the client.
    - title (required): The title of the job.
    - description (required): The job description.
    - budget (required): The proposed budget for the job.
    - skills (required): Skills required for the job.
    - deadline (required): The job completion deadline.
* **Response Example:**
  + Success Response (201 Created)

{

"message": "Job posted successfully", "jobId": "J12345"

}

1. **Fetch Jobs Endpoint**

* **URL:** /fetch-jobs
* **Method:** GET
* **Description:** Retrieves a list of job postings.
* **Authentication:** None.
* **Request Parameters:** None.
* **Response Example:**
  + Success Response (200 OK)

[ { "jobId": "J12345", "title": "Web Development", "budget": 500, "skills": ["HTML", "CSS", "JavaScript"], "deadline": "2024-12-15" } ]

1. **Submit Proposal Endpoint**

* **URL**: /submit-proposal
* **Method**: POST
* **Description**: Allows freelancers to submit a proposal for a job.
* **Authentication**: Required.
* **Request Parameters:**
  + Body Parameters:
    - freelancerId (required): The unique ID of the freelancer.
    - jobId (required): The ID of the job being applied for.
    - proposal (required): The proposal description.
    - bidAmount (required): The freelancer's bid amount.
    - deliveryTime (required): The estimated delivery time for the job.
* **Response Example:**
  + Success Response (200 OK)

{ "message": "Proposal submitted successfully" }

1. **Fetch Proposals Endpoint**

* **URL**: /fetch-proposals/{jobId}
* **Method**: GET
* **Description**: Retrieves all proposals submitted for a specific job.
* **Authentication**: Required.
* **Request Parameters:**
  + URL Parameters:
    - jobId (required): The unique ID of the job.
* **Response Example:**
  + Success Response (200 OK)

[ { "proposalId": "P123", "freelancerId": "F5678", "bidAmount": 400, "proposal": "I can complete this job within 7 days.", "deliveryTime": "7 days" } ]

1. **Approve Proposal Endpoint**

* **URL:** /approve-proposal
* **Method:** POST
* **Description:** Approves a freelancer's proposal for a job.
* **Authentication:** Required.
* **Request Parameters:**
  + **Body Parameters:**
    - proposalId (required): The unique ID of the proposal to be approved.
* **Response Example:**
  + Success Response (200 OK)

{ "message": "Proposal approved successfully" }

1. **Mark Job as Completed Endpoint**

* **URL**: /mark-completed/{jobId}
* **Method**: PUT
* **Description**: Marks a job as completed by the client.
* **Authentication**: Required.
* **Request Parameters:**
  + URL Parameters:
    - jobId (required): The unique ID of the job to be marked as completed.
* **Response Example:**
  + Success Response (200 OK)

{"message": "Job marked as completed" }

1. **Fetch User Details Endpoint**

* **URL:** /fetch-user/{id}
* **Method:** GET
* **Description:** Fetches the details of a specific user based on their ID.
* **Authentication:** Required.
* **Request Parameters:**
  + **URL Parameters:**
    - id (required): The unique ID of the user.
* **Response Example:**
  + Success Response (200 OK)

{ "\_id": "12345", "username": "John Doe", "email": "johndoe@example.com", "usertype": "client" }

1. **Fetch Freelancers by Skills Endpoint**

* **URL**: /fetch-freelancers
* **Method**: POST
* **Description**: Retrieves a list of freelancers based on required skills.
* **Authentication**: None.
* **Request Parameters:**
  + Body Parameters:
    - skills (required): An array of required skills (e.g., ["Python", "Django"]).
* **Response Example:**
  + Success Response (200 OK)

[ { "freelancerId": "F5678", "username": "Jane Doe", "skills": ["Python", "Django"], "hourlyRate": 50 } ]

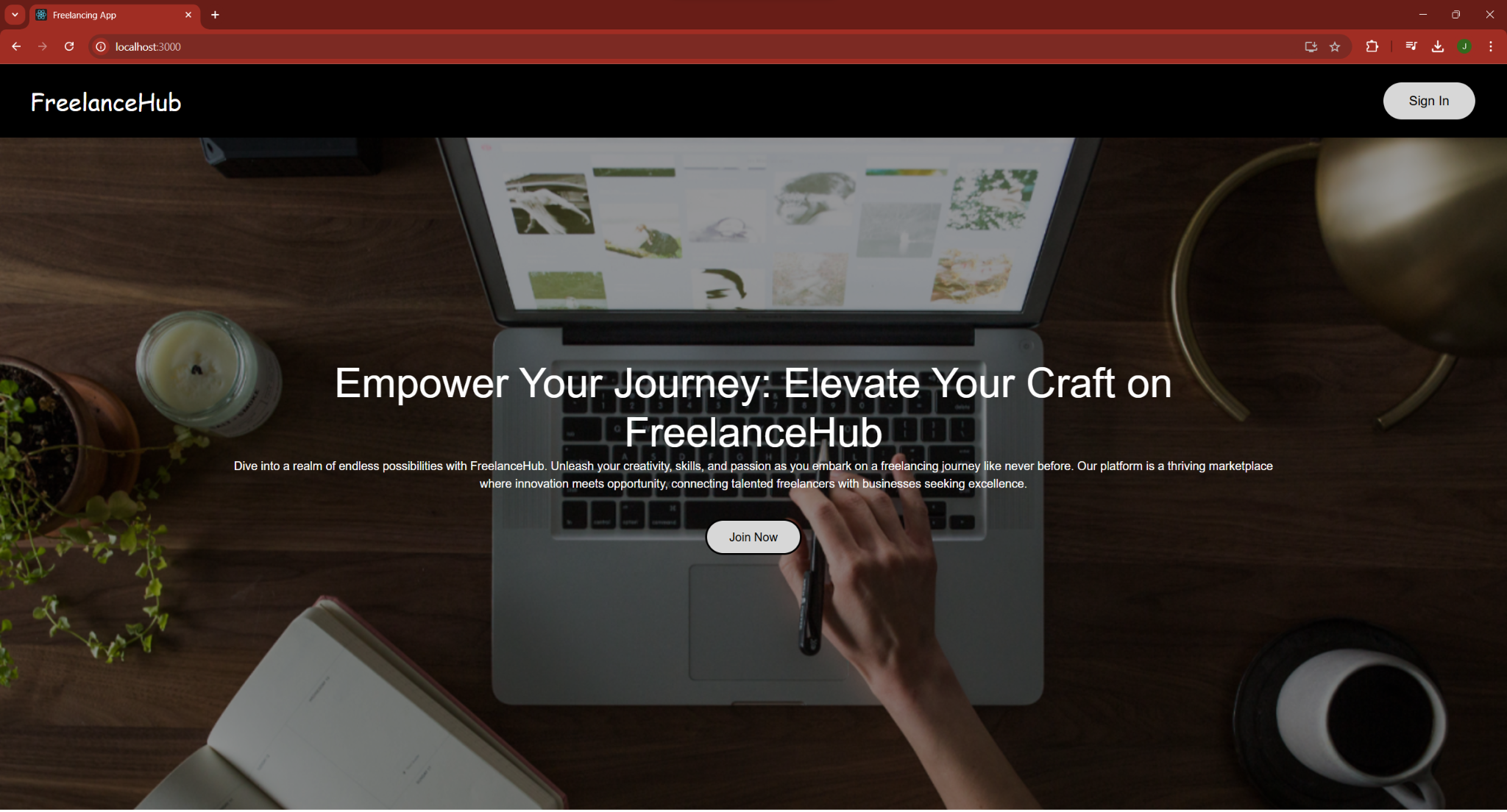
**Authentication:**

Authentication in this project is managed through a simple user login system.

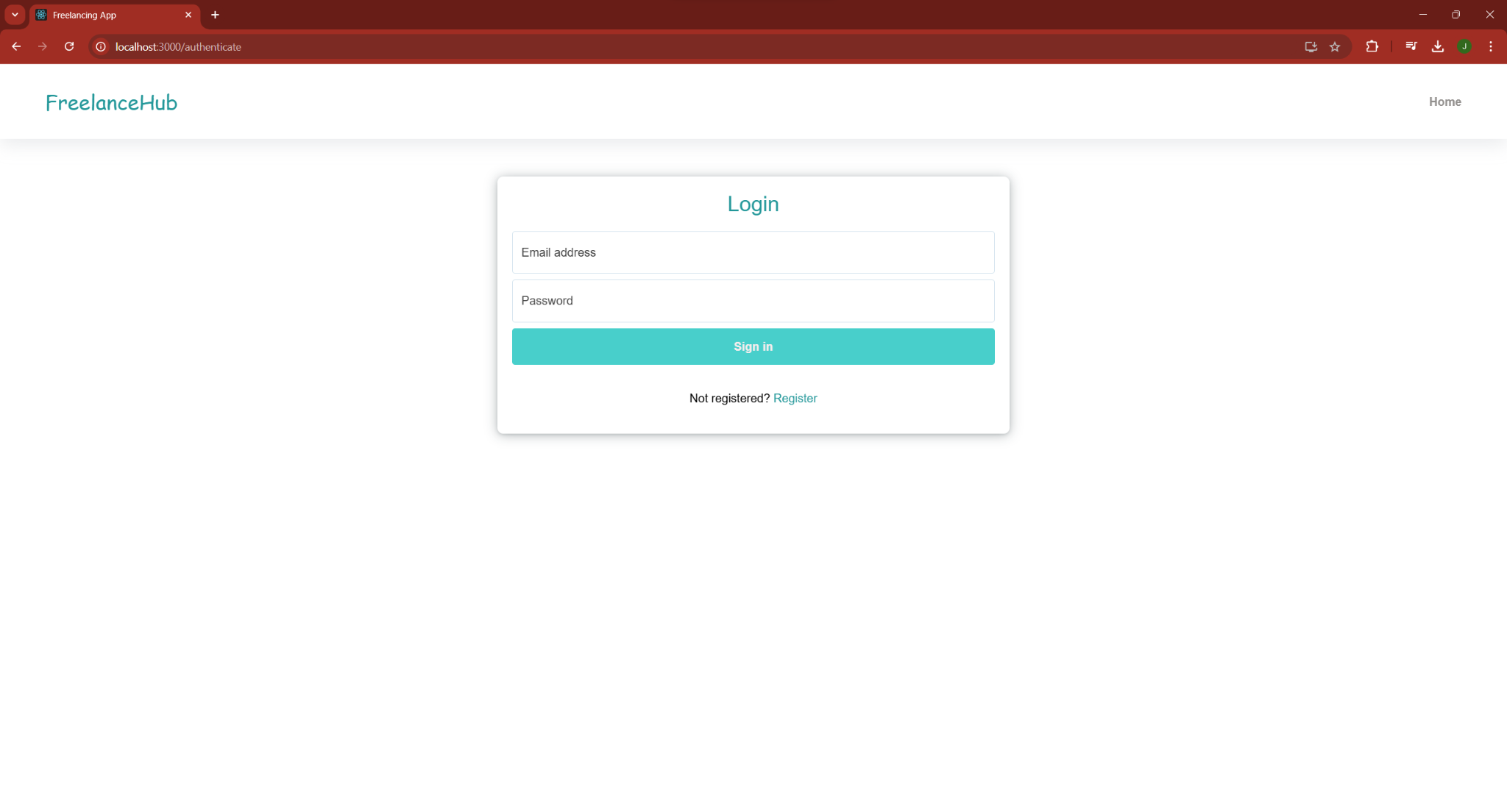
* **User Login and Registration:** Users can create an account with a username, password, and email. The credentials are securely stored in the database.
* **Middleware:** Basic authentication middleware is used to check if a user is logged in before accessing certain protected routes.
* **Role-Based Access:** The system differentiates between regular users and admin users using a usertype field in the user schema, allowing the admin to manage flights and bookings.

**User Interface:**

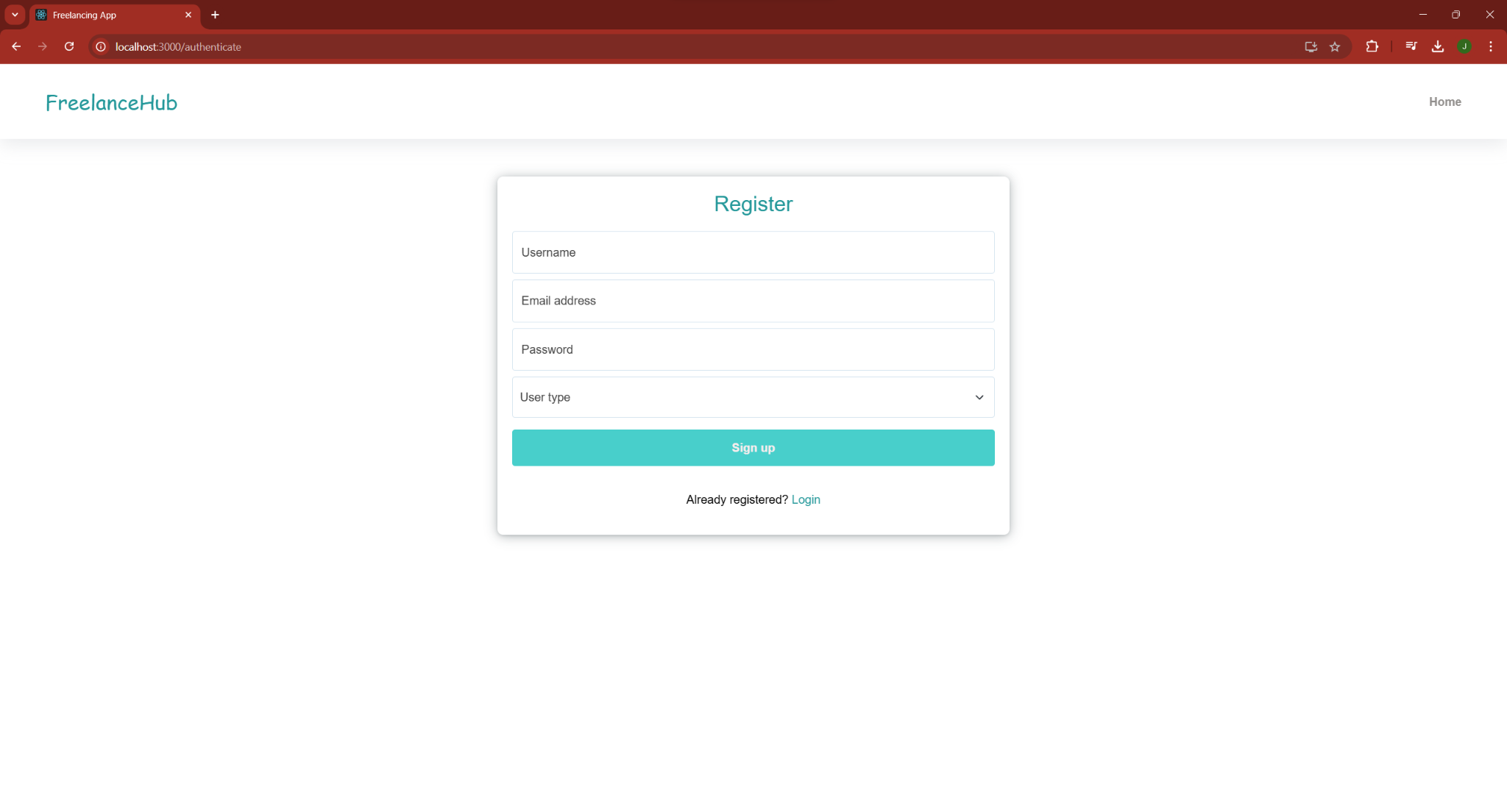
**Landing page:**

****

**Login page:**

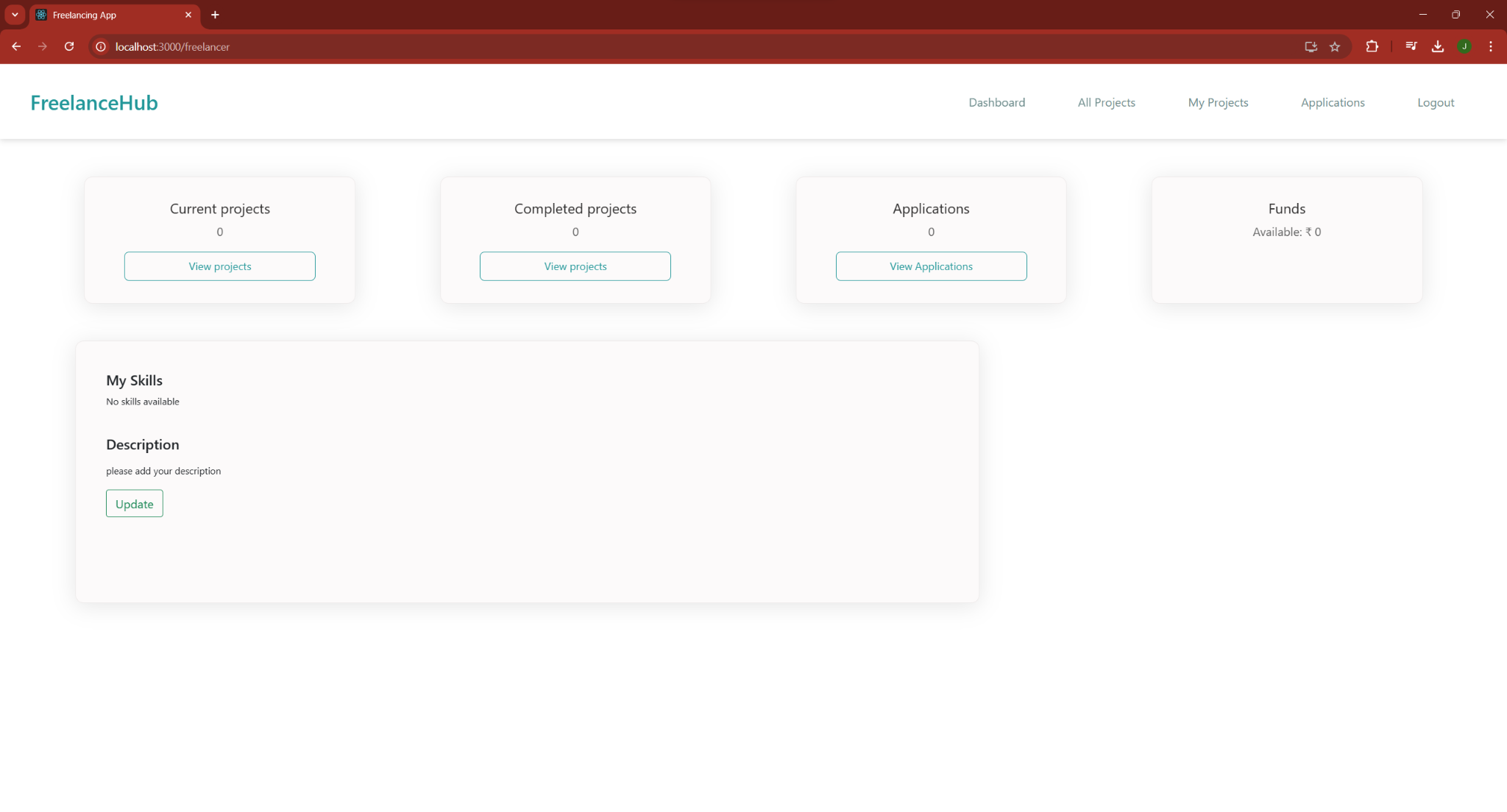
****

**Register page:**

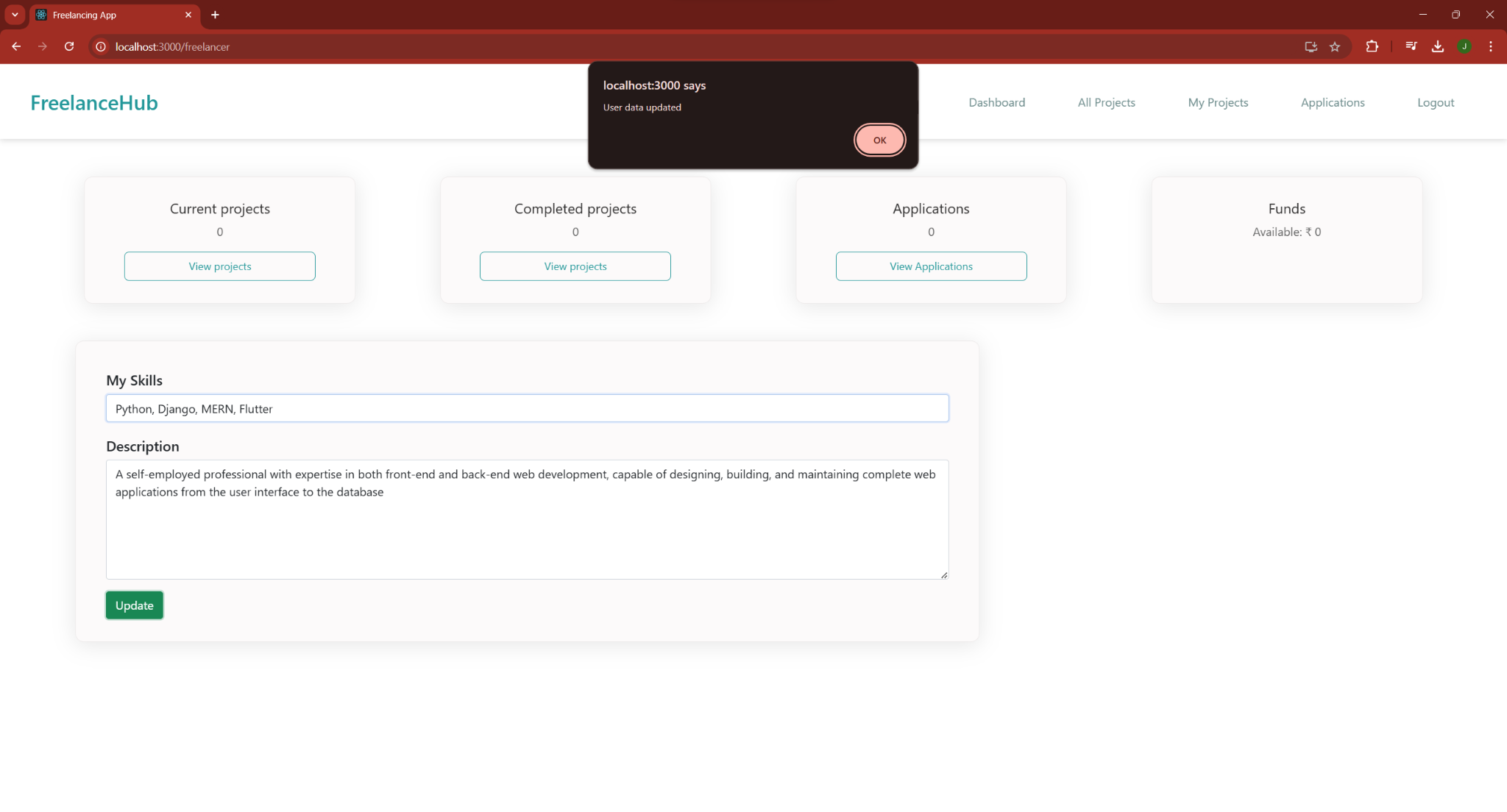
****

**Freelancer:**

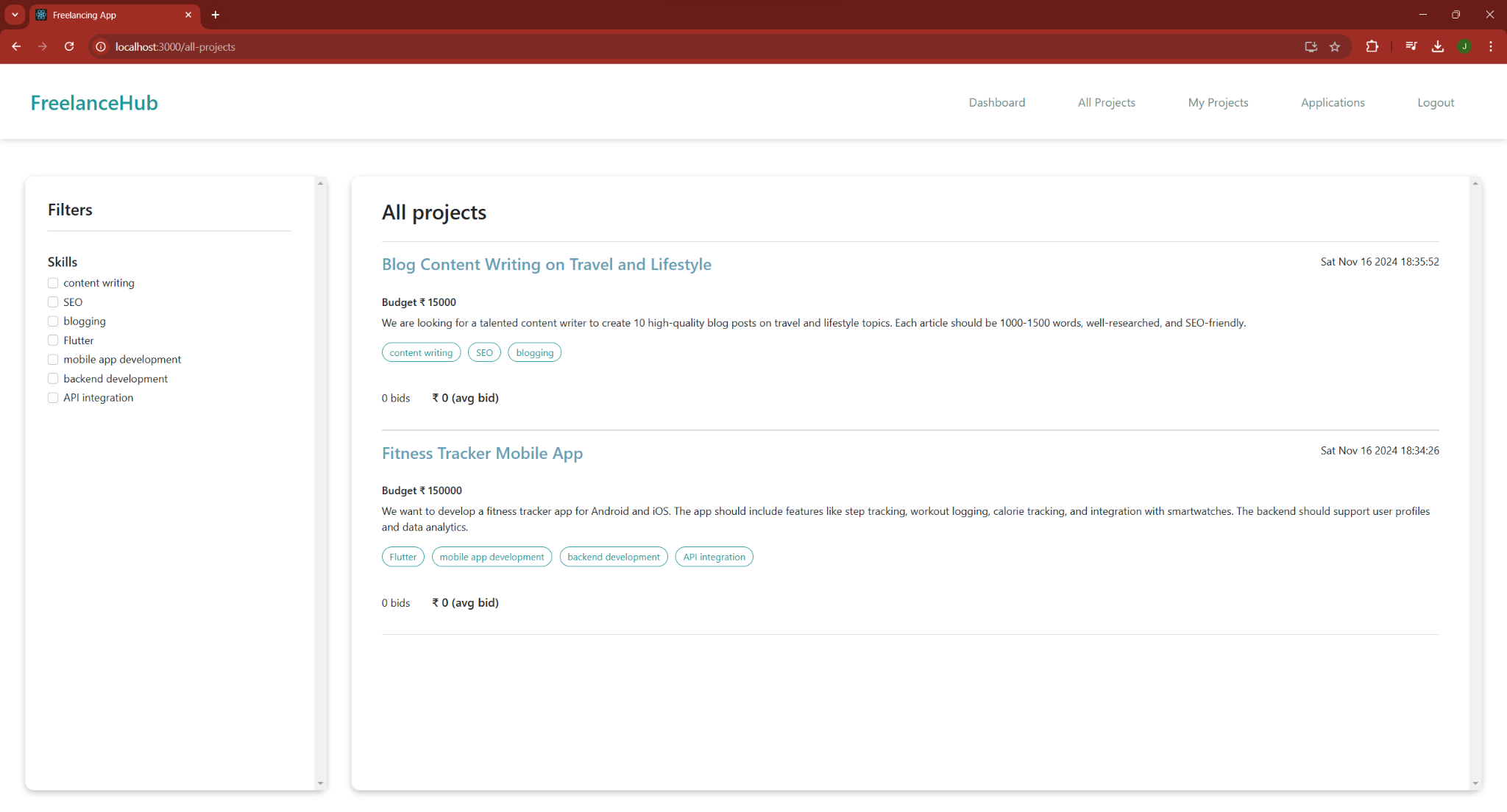
**Dashboard page:**

****

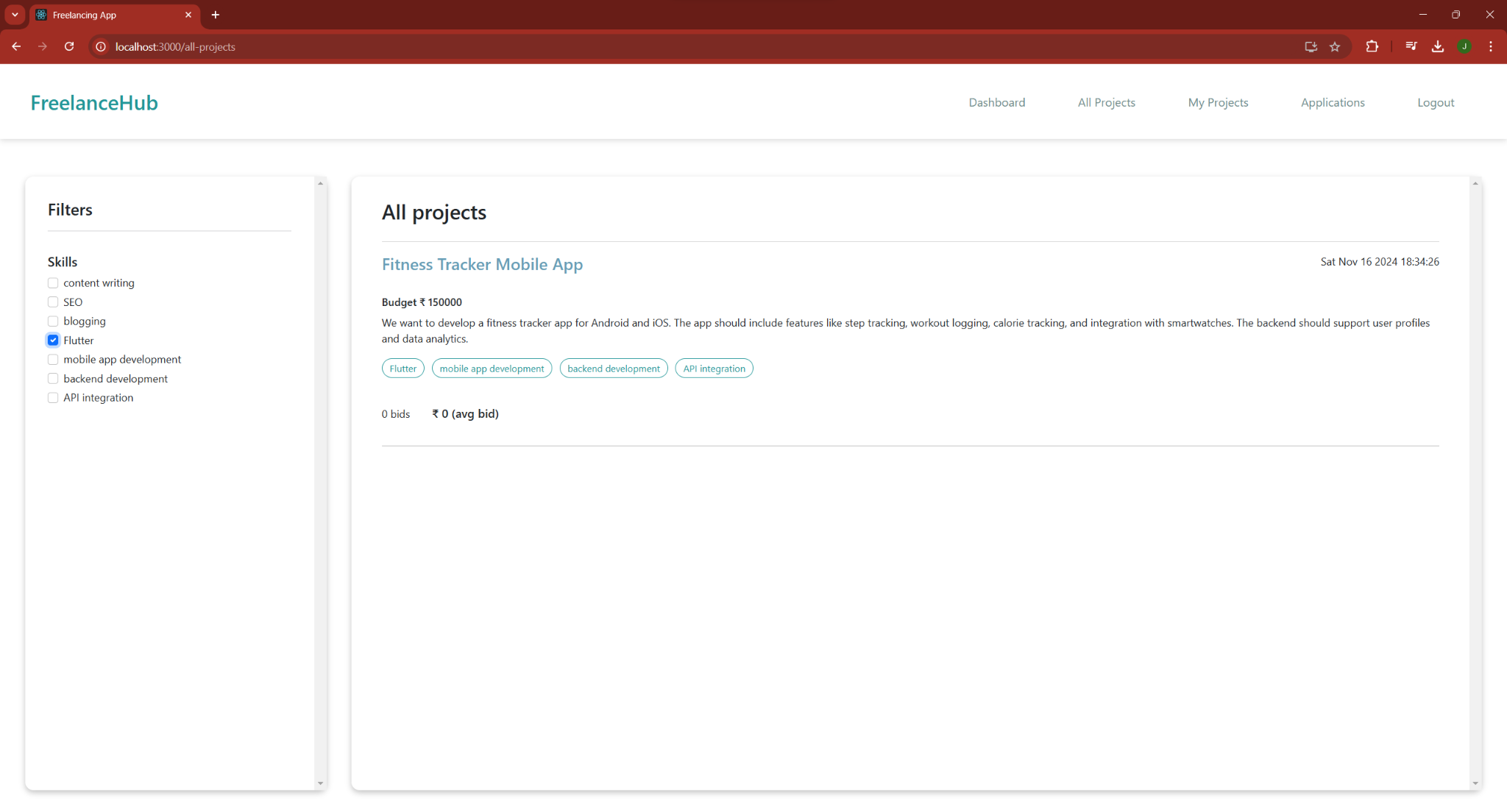
**Update details:**

****

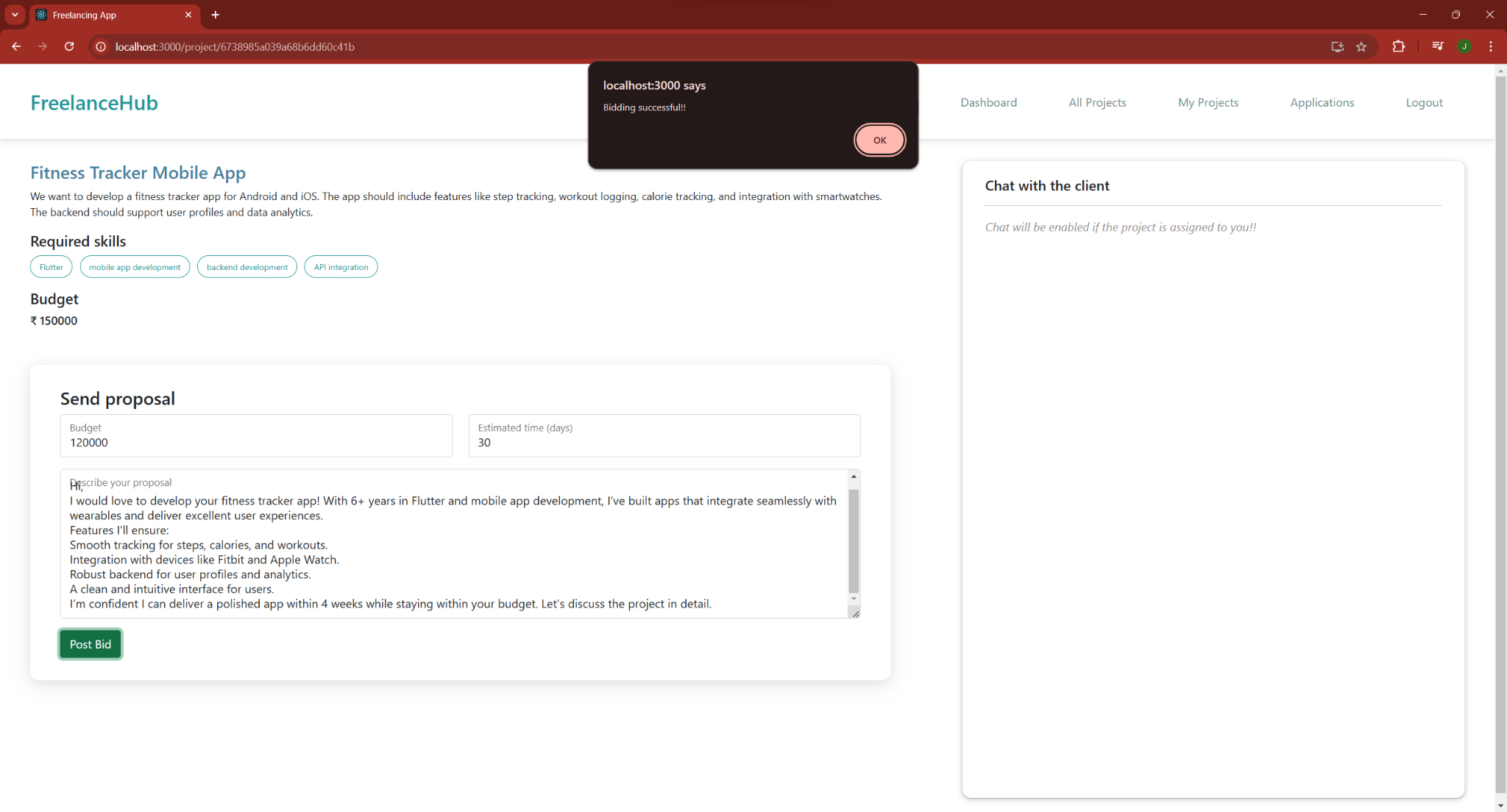
**Projects page:**

****

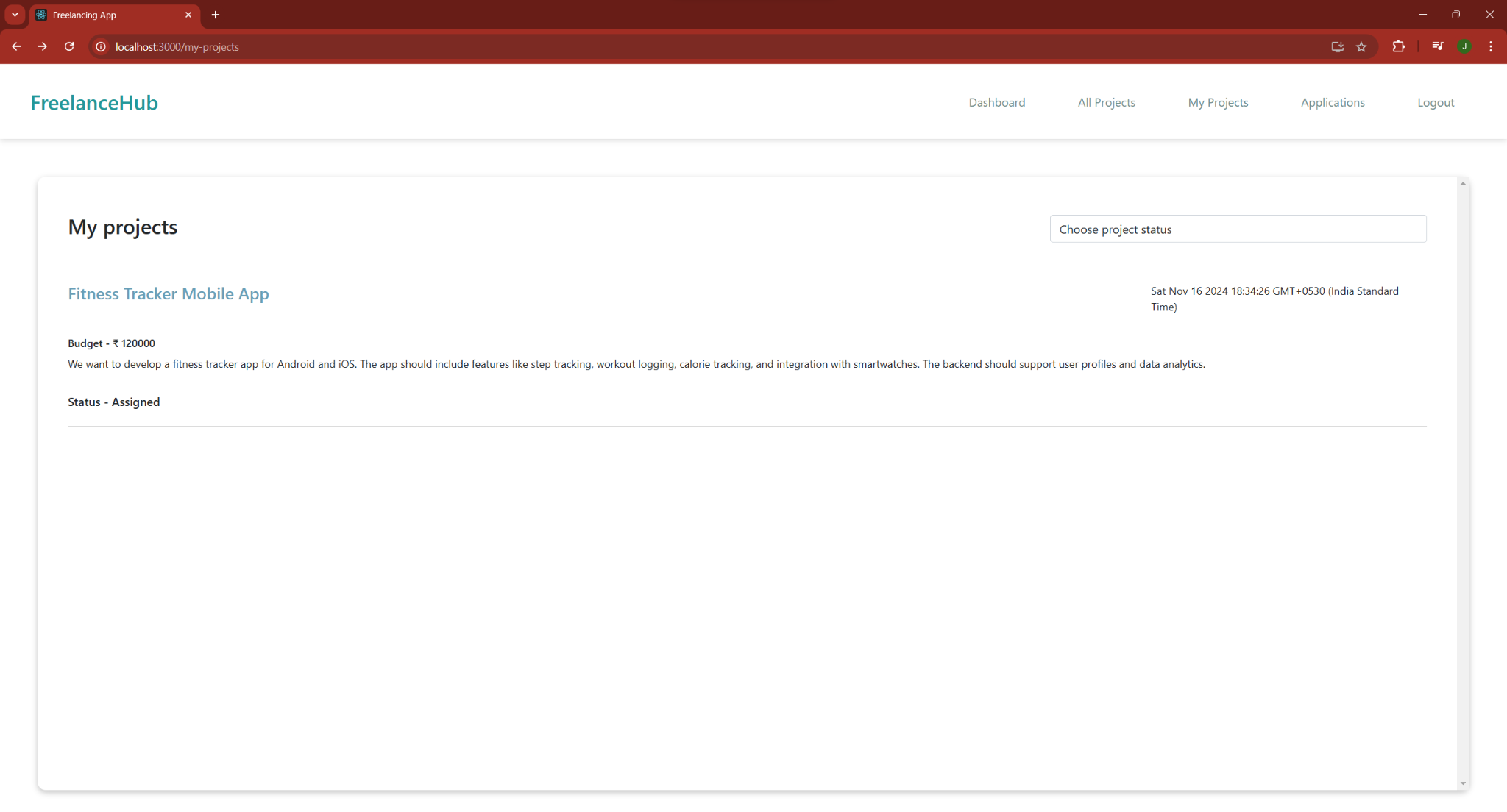
**Filters for projects:**

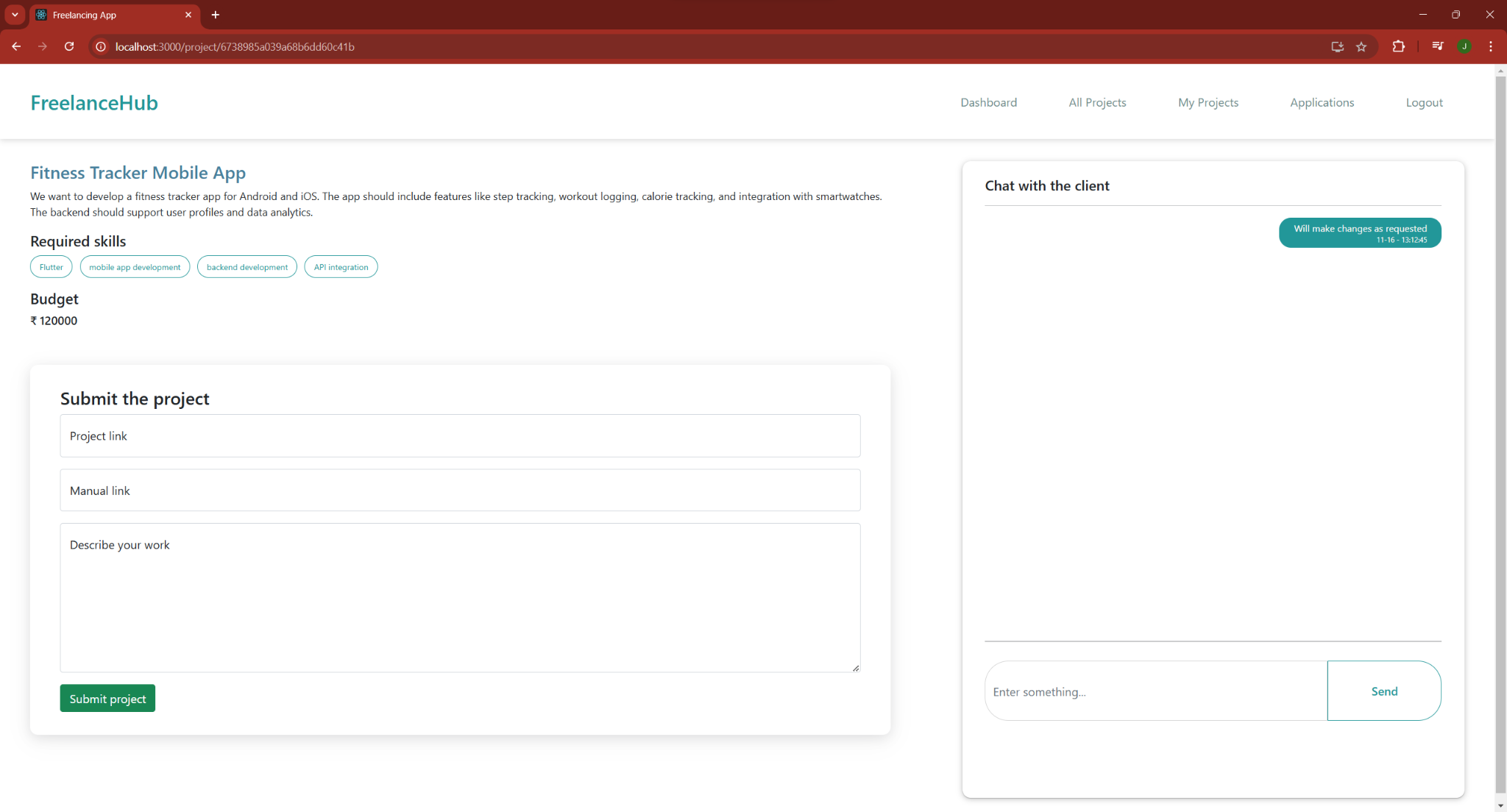
****

**Proposal for project:**

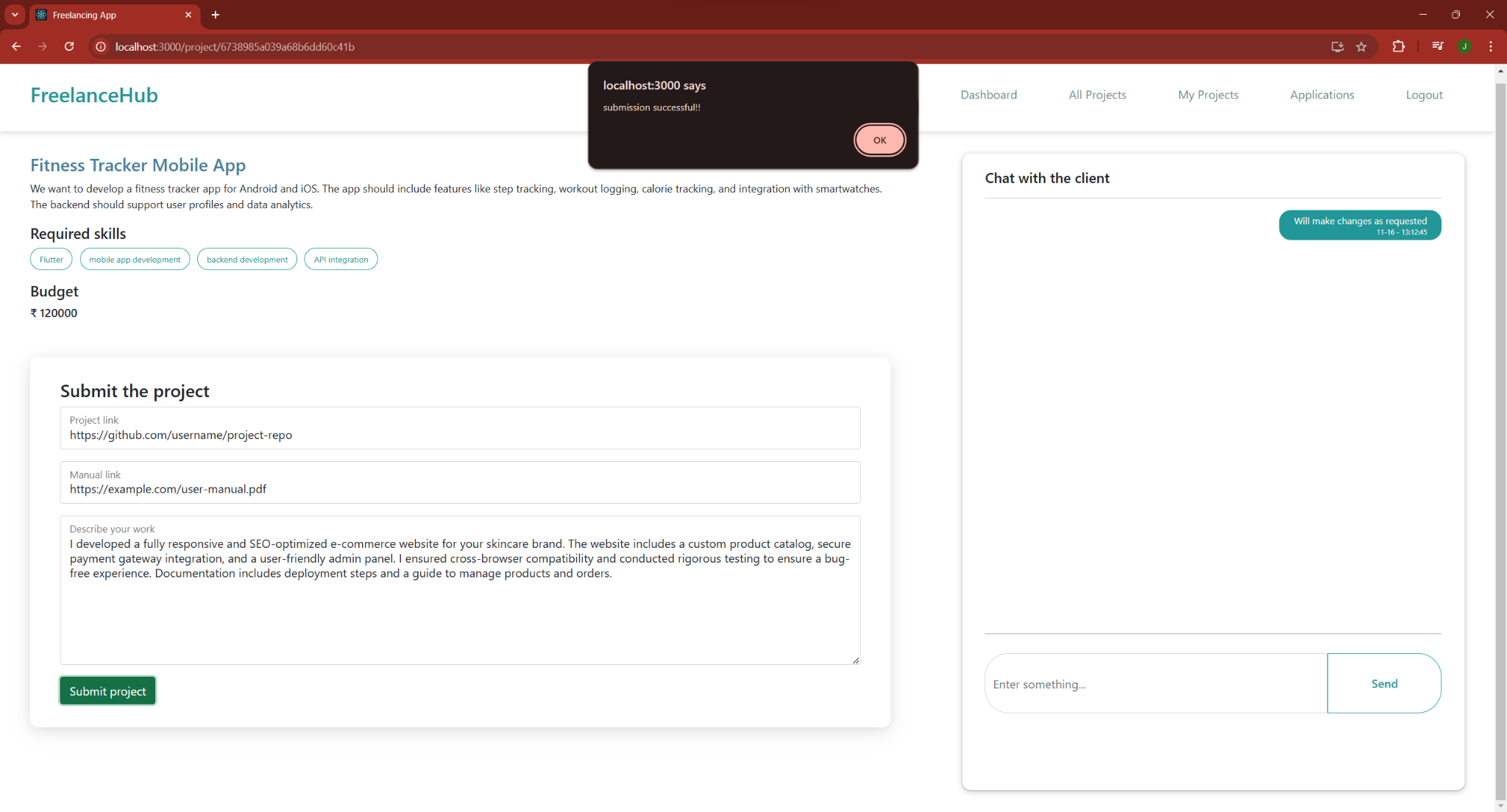
****

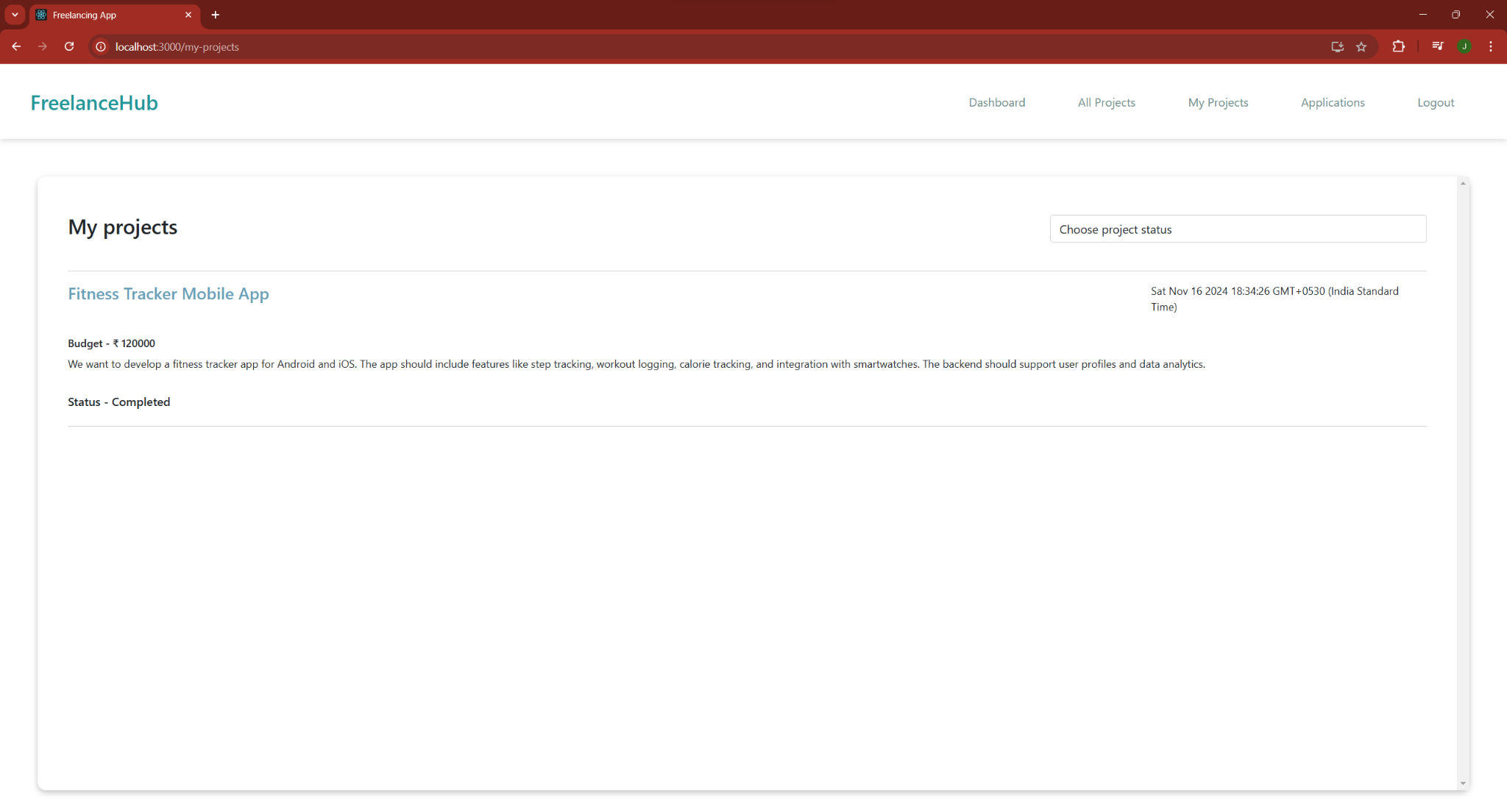
**Assigned projects:**

****

****

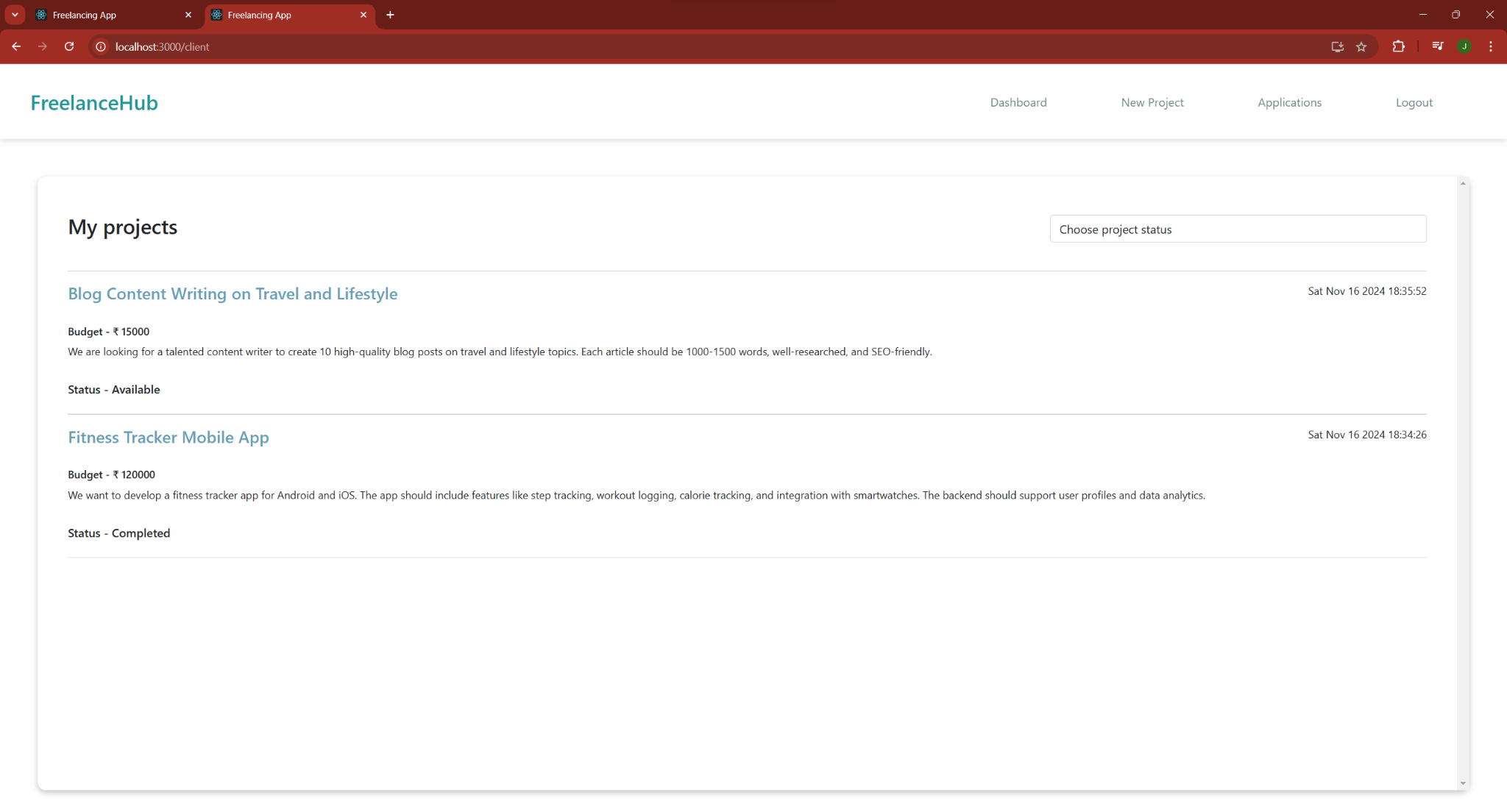
**Submitting the project:**

****

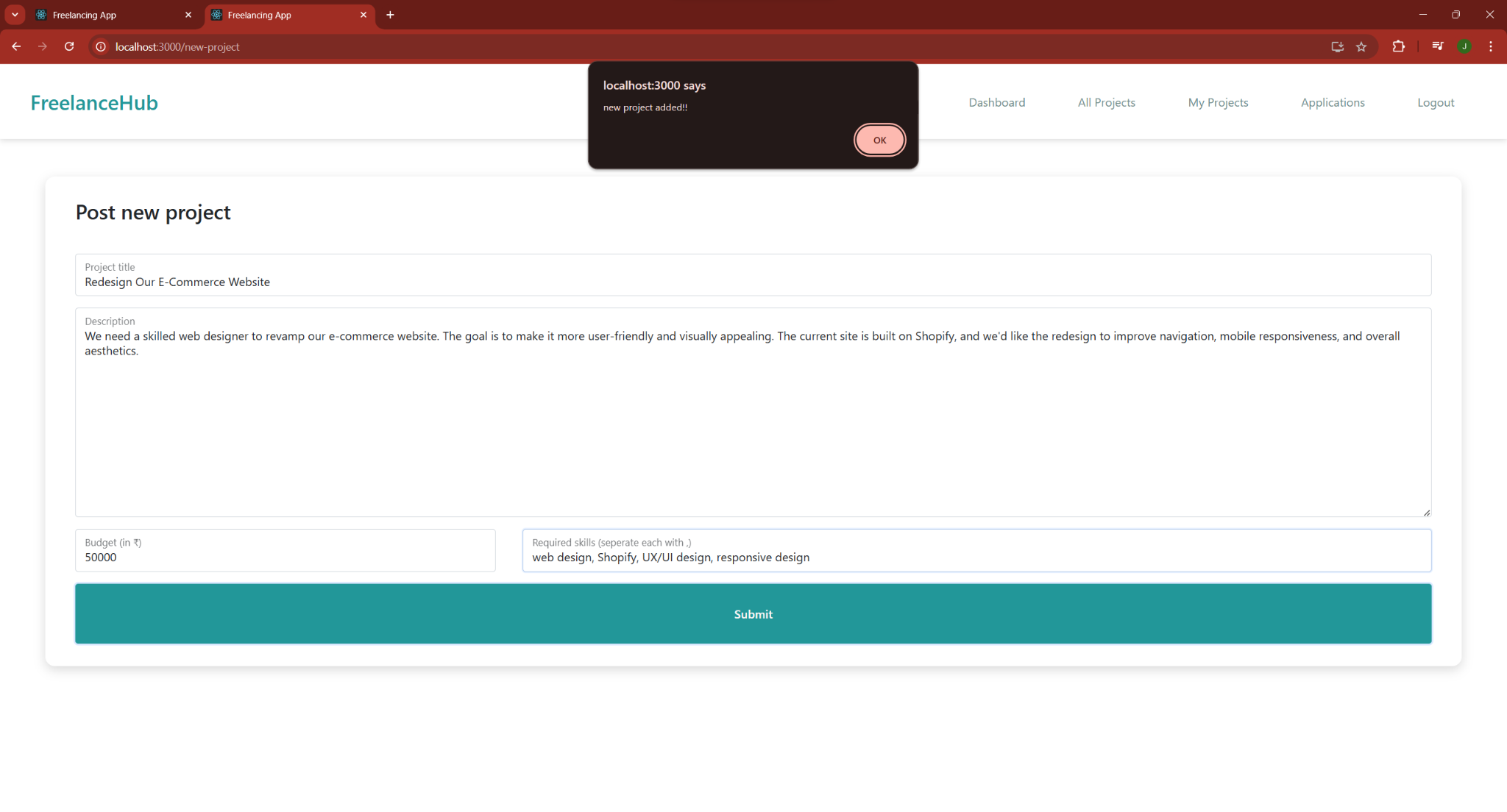


**Client:**

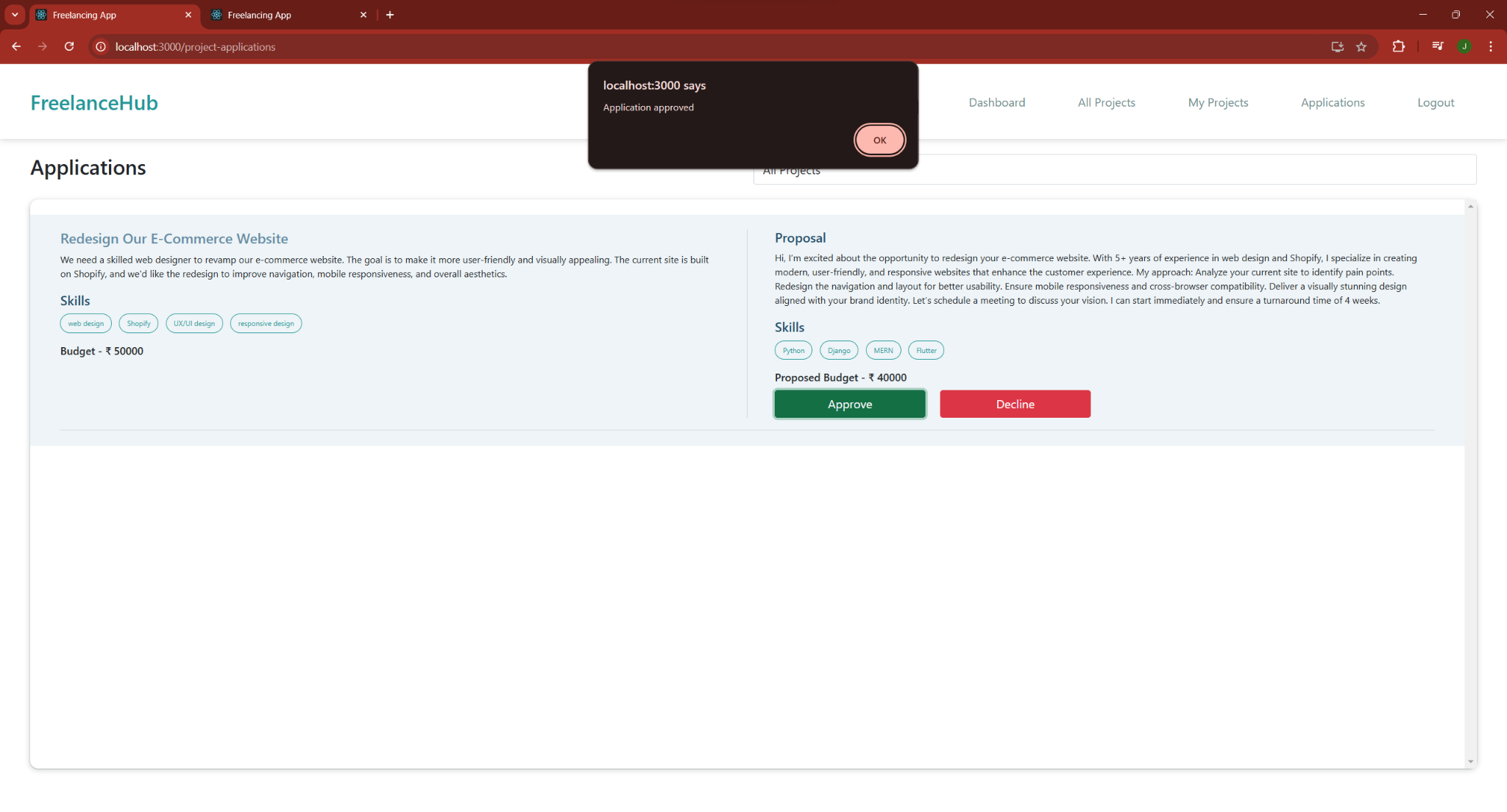
**Dashboard page:**

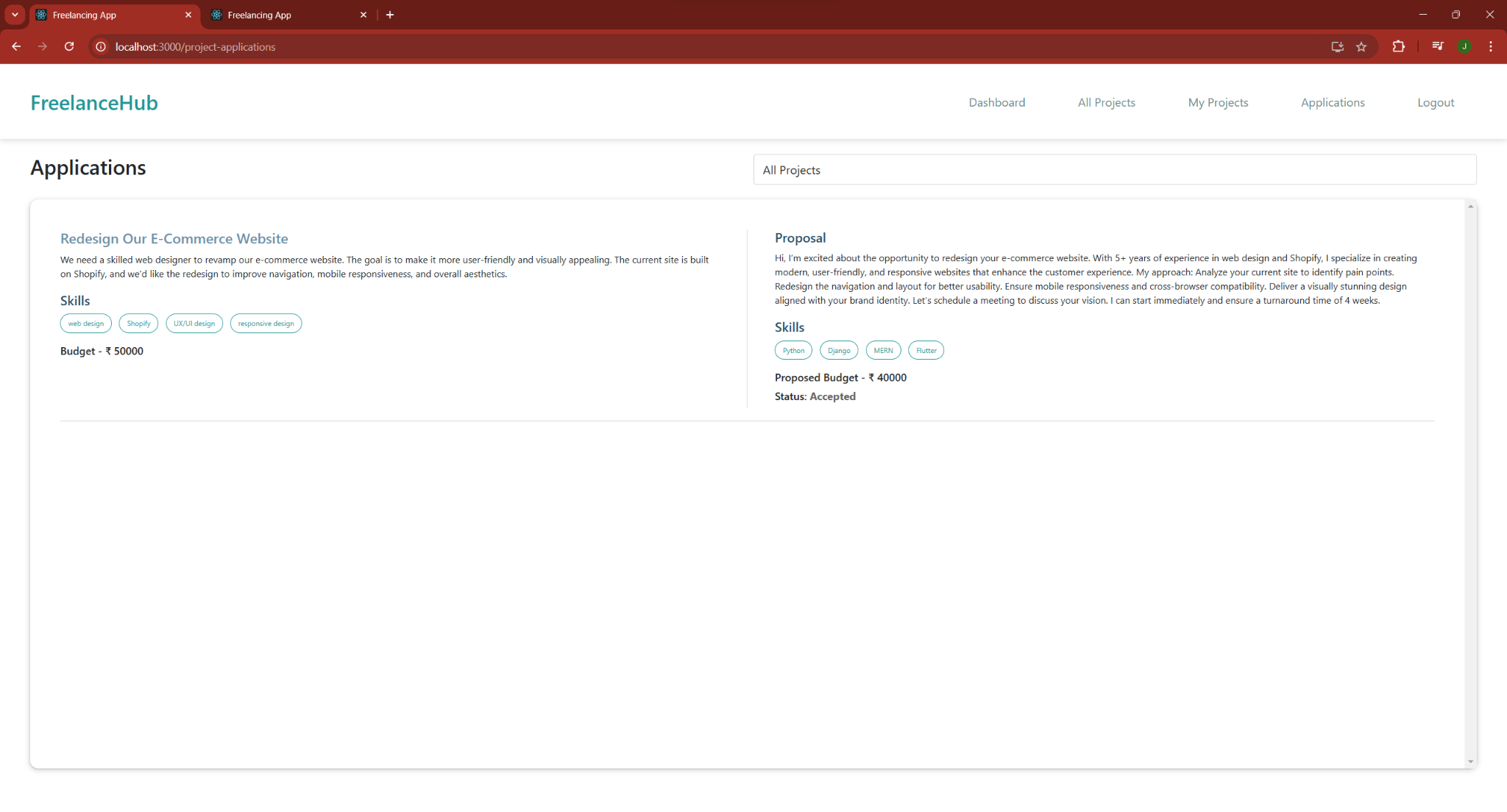
****

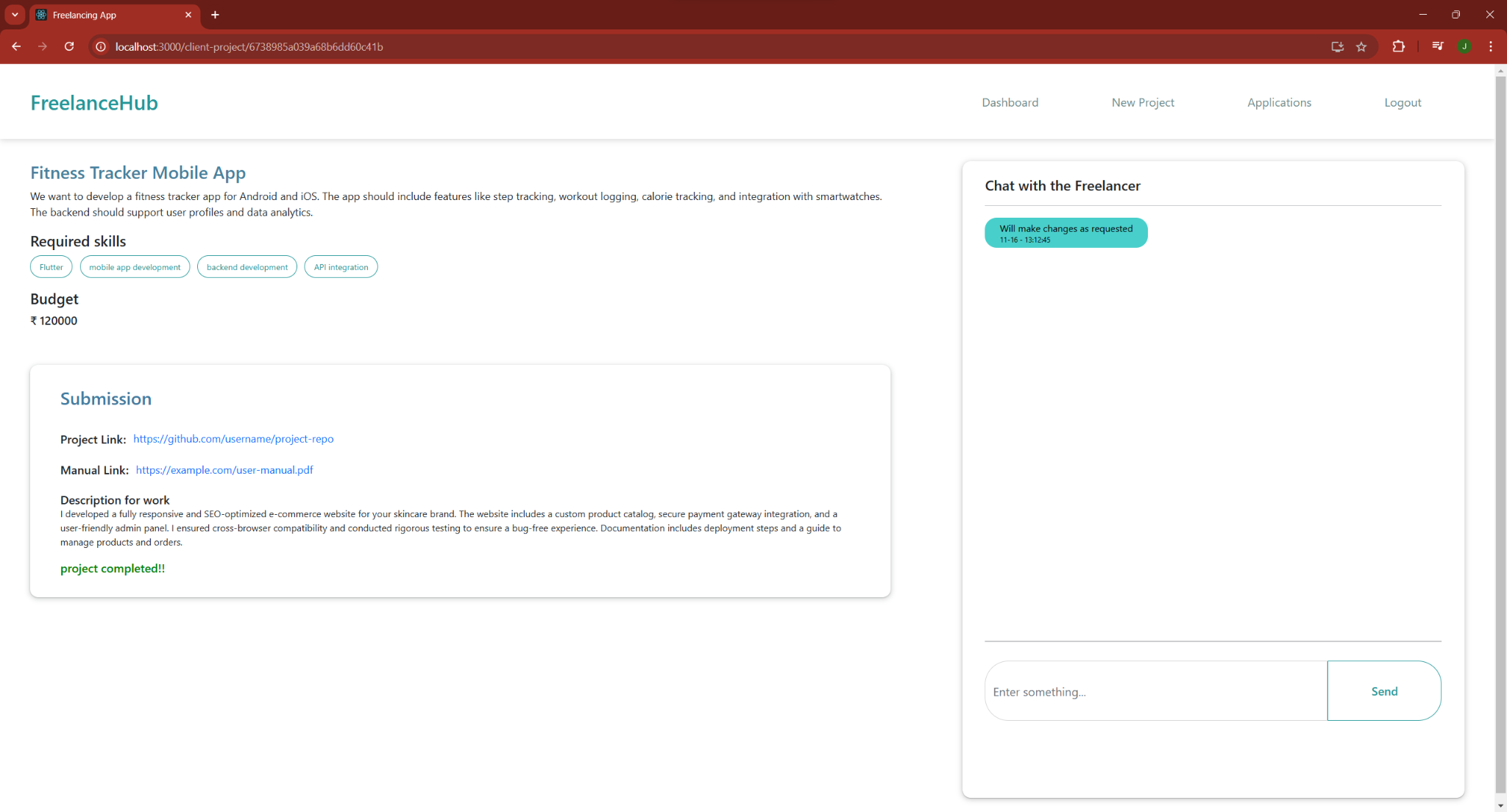
**Adding projects:**

****

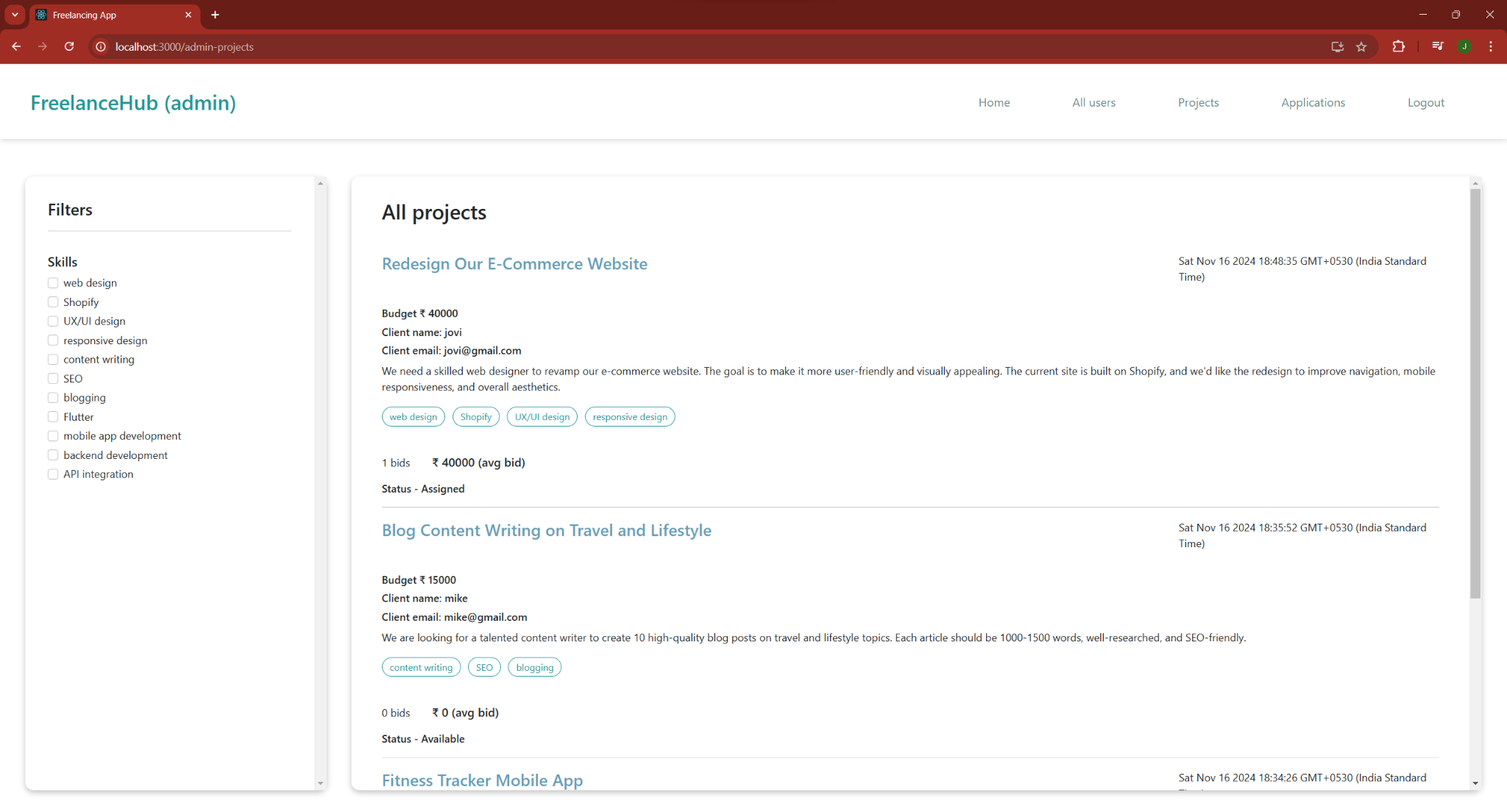
**Proposal accepting:**

****

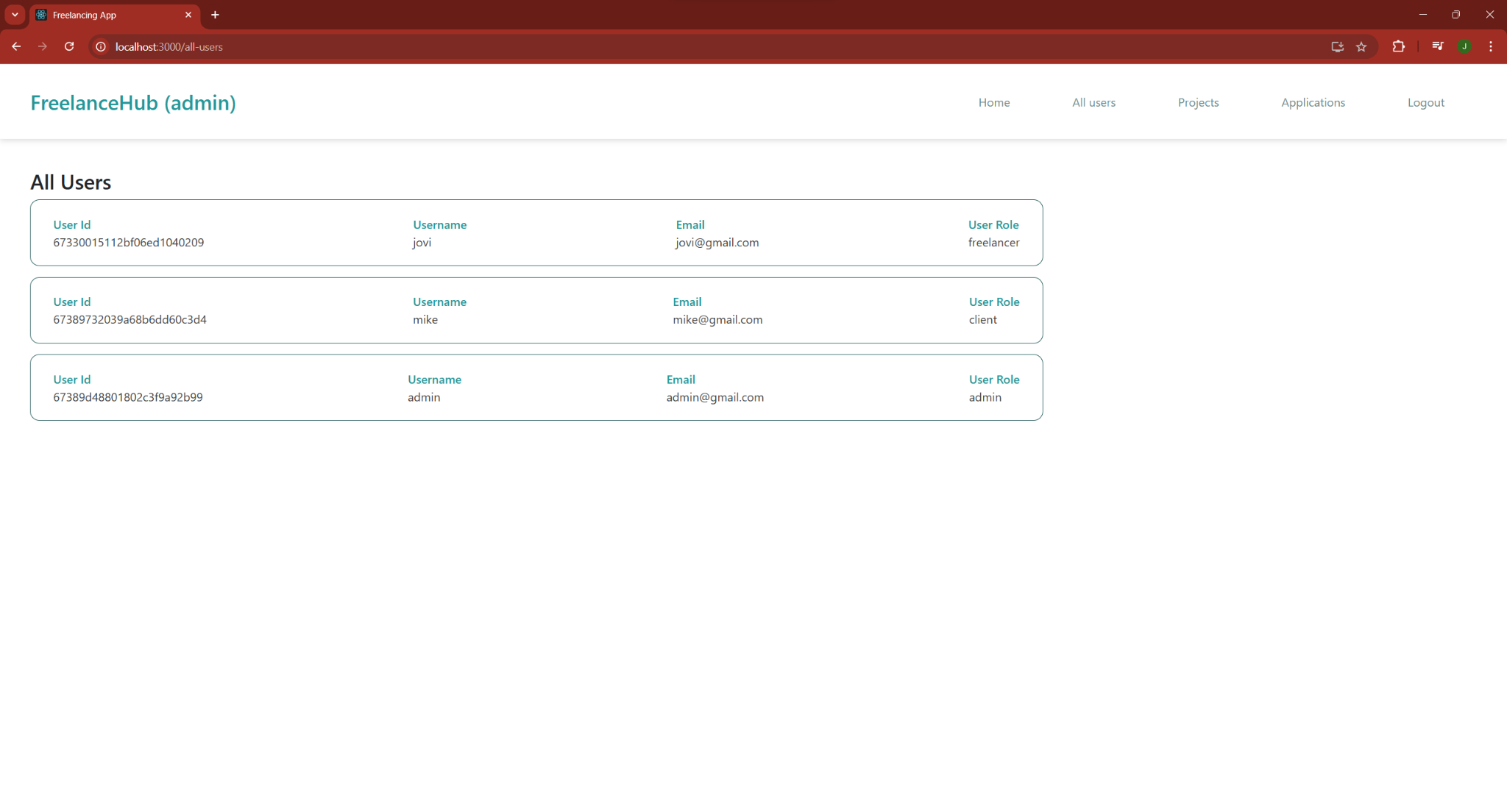
****

**Project approval : **

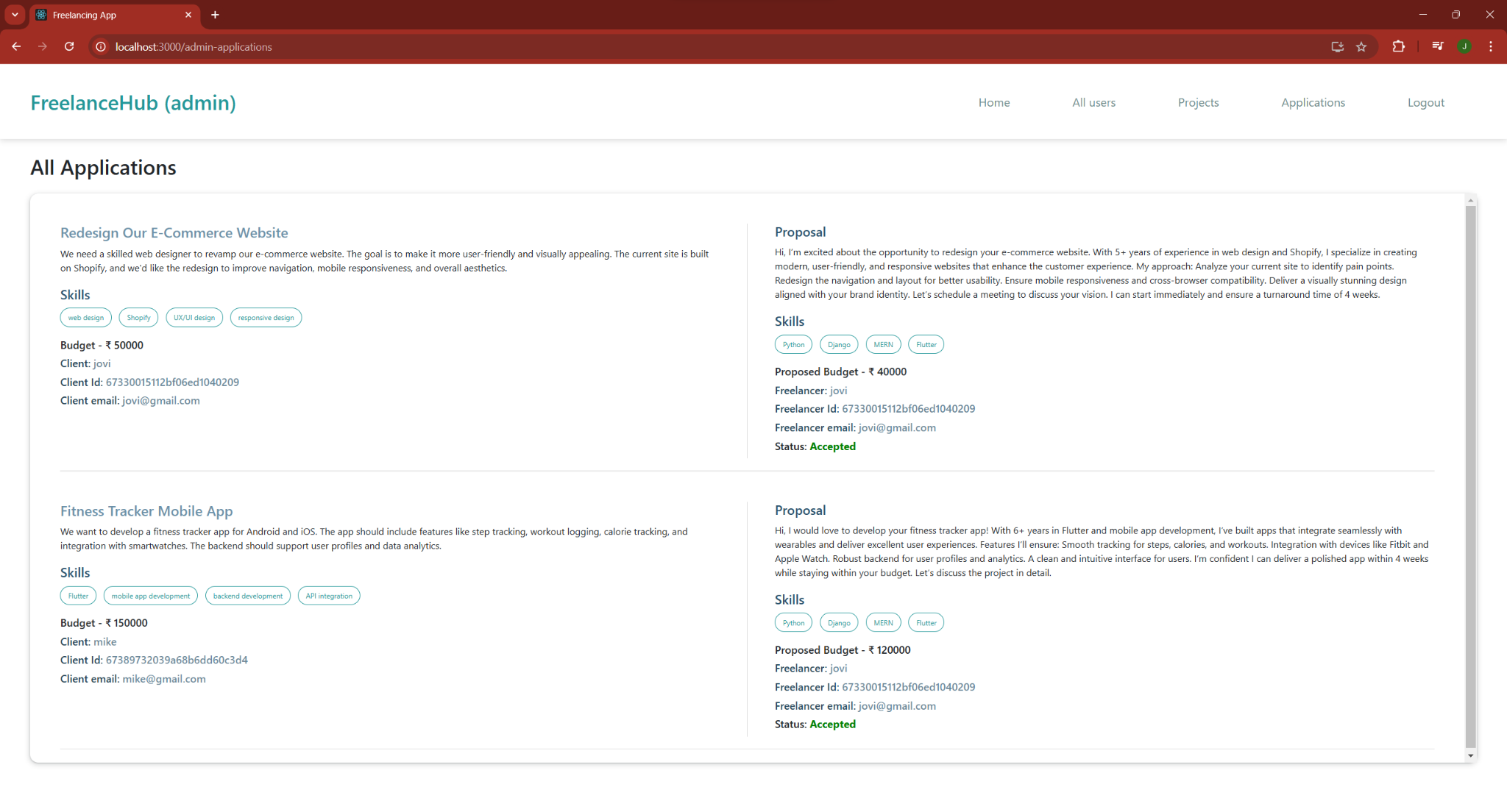
**Admin:**

**Projects : **

**Users:**

****

**Applications:**

****

#### 

#### **Testing :**

#### **1. Manual Testing:**

* Test key user flows such as project creation, proposal submission, project assignment, and payment processing.
* Ensure that all user roles (Client, Freelancer, Admin) can perform their respective tasks correctly.
* Verify the UI for usability, responsiveness, and proper functionality across different devices and screen sizes.

#### **2. Unit Testing (with Jest):**

* Test individual backend functions such as user registration, login, project creation, proposal handling, and submission functionalities.
* Mock database interactions to isolate logic and validate edge cases, such as incomplete project details or invalid user inputs.
* Test validation logic for forms, such as budget limits, required fields, and password criteria.

#### **3. Integration Testing:**

* Ensure seamless communication between APIs and the database, validating correct storage, retrieval, and updates of project and user data.
* Test the complete workflow, such as:
  1. A client creating a project.
  2. Freelancers submitting proposals.
  3. Clients reviewing and accepting a proposal.
  4. Freelancers submitting the completed project.
  5. Payment and feedback processes.
* Verify notification systems, ensuring users are informed at appropriate stages (e.g., proposal acceptance, payment requests).

### **Demo Link**

### **Known Issues:**

1. **Concurrent Proposal Submissions:**
   * Multiple freelancers submitting proposals simultaneously may cause inconsistencies in the proposal queue or project status. Implementing locking mechanisms can resolve this.
2. **Delayed Updates in Project Status:**
   * Clients or freelancers may see outdated project statuses due to delayed updates. Real-time syncing and improved data consistency are needed.
3. **Performance Issues with Large Data Sets:**
   * A growing number of projects and users can slow down searches and filter operations. Database optimization and caching techniques are necessary to maintain performance.
4. **Session Management Issues:**
   * Users may experience unexpected logouts or lose progress while creating or submitting proposals. Consistent session handling across devices is essential.
5. **Edge Case Handling:**
   * Payment failures or invalid project details may lead to unexpected behavior. Robust error handling and clear user feedback can prevent disruptions.

### **Future Enhancements:**

1. **Mobile Application Development:**
   * Building a mobile app using React Native to enable clients and freelancers to access the platform on-the-go, improving user engagement.
2. **Real-Time Notifications:**
   * Integrating push or SMS notifications for updates like proposal acceptances, project status changes, and payment confirmations to enhance communication.
3. **Payment Gateway Integration:**
   * Adding support for multiple payment options (e.g., UPI, credit/debit cards, wallets) to streamline the payment process for clients and freelancers.
4. **AI/ML-Powered Project Recommendations:**
   * Utilizing AI/ML algorithms to suggest projects to freelancers based on their skills, past submissions, and activity patterns, personalizing the platform experience.
5. **Load Testing:**
   * Conducting load testing to ensure the platform can handle high traffic during peak usage times, maintaining speed and reliability.

These enhancements aim to make the freelancing platform more user-friendly, scalable, and efficient for all stakeholders.