



Republic of the Philippines

Region V Bicol

BICOL UNIVERSITY

Polangui, Albay

Computer Studies Department



Lab Submission for Week 14

(Phase 4 and Phase 5)

ProFolio

Where you let your work shine and watch opportunities follow

Member	Role
Ofracio, Aran Joshua P.	Project Manager
Dolon, Ella Mae R.	Frontend Developer
Lumayad, Victor Ronel P.	Backend Developer
Dela Cruz, Paulina C.	Database Manager
Odoño, Alona P.	GitHub Manager
Noble, Noelissa S.	Documentation Officer

BSIT 2A

Dr. Guillermo V. Red, Jr.

Course Professor


May 17, 2025


Screenshot of the HTML form interface

Login Form

- The login form is the initial point of access for users entering the ProFolio platform, whether they are freelancers or clients. It features two main input fields: one for the username or email and another for the password. These fields are accompanied by intuitive icons that enhance usability and guide the user visually. This form is designed with validation features that prevent empty or incorrect submissions, ensuring that only registered users can log in. Aside from usability, the form also addresses basic security concerns by masking the password input. Its simplicity, combined with clear labeling and proper feedback, makes the login process user-friendly while still maintaining control over authenticated access to personalized user dashboards.

Log in to ProFolio

 Username or Email

 Password

Registration Form

- This form enables a new freelancer or client to register an account on the platform. Fields include the client’s first name, last name, email, password, country, and confirmation fields. The registration process is streamlined to encourage easy onboarding.


First name*

Last name*

Email*

Password *

Password (8 or more characters)



Country*

Philippines


☐ Send me helpful emails to find rewarding projects and opportunities.

☐ Yes, I understand and agree to the ProFolio [Terms of Service](#), including the [User Agreement](#) and [Privacy Policy](#).

Create my account

Freelancer Profile Section

- This form section allows freelancers to view their personal and professional information such as name, email, job title, and professional summary (with the job title and summary being editable). It is designed to be editable so that users can make changes as needed, whether they're switching industries, updating contact details, or refining their bio. Keeping profile information current is essential in a freelance platform where visibility and credibility are important. This section ensures freelancers have full control over how they present themselves to potential clients, allowing for both flexibility and professionalism.



Change Photo

Full Name

Aran Joshua Ofracio

Email Address

aranofraccio@gmail.com

Job Title

Virtual Assistant

Professional Summary

I'm a Virtual Assistant who helps busy entrepreneurs stay organized and get things done. I take care of admin work, emails, scheduling, research, and whatever else you need so you can focus on the big stuff. Reliable, efficient, and easy to work with.

Save Changes

Cancel

Freelancer 'My Portfolio' Section

- The 'My Portfolio' section is a dynamic form-based interface that allows freelancers to input, manage, and update entries related to their previous work. Each entry includes the project title, a description, skills, work experience with timeline and work samples with external links like websites or repositories. This feature serves as a live résumé, giving freelancers a way to showcase their experience and capabilities directly within the platform. Editable and user-controlled, the portfolio section empowers freelancers to keep their profiles current and relevant, which is a key to attracting job offers from interested clients browsing the system.

1

Basic Info

2

Skills

3

Experience

4

Work Samples

Basic Information

Portfolio Name *
e.g., Web Development Projects

Description *
Describe your portfolio and what makes it special...

Next →

1

2

3

4

Basic Info

Skills

Experience

Work Samples

Skills

Skills *

e.g., JavaScript

+

Add skills relevant to this portfolio. Press the + button after typing each skill.

← Previous

Next →

1

2

3

4

Basic Info

Skills

Experience

Work Samples

Work Experience

* At least one work experience is required

Experience #1

Work Title *

Position or project title

Company/Client *

Company or client name

Start Date *

End Date *

Description *

Describe your responsibilities and achievements...

+Add Another Experience

← Previous

Next →

1

2

3

4

Basic Info

Skills

Experience

Work Samples

Work Samples

* At least one work sample is required

Sample #1

Project Title *

Project name

Project Description *

Describe your project...

Project URL *

https://...

+Add Another Work Sample

← Previous


Create Portfolio

Client Profile section

- The client profile section allows clients to view their name and email and modify their company name which is displayed in the sidebar. Keeping this information updated not only enhances the client’s credibility but also helps freelancers learn more about the people or businesses offering jobs. A well-detailed client profile fosters transparency, builds trust, and improves communication between parties.

Your Profile

Manage your professional information



Change Photo

Full Name

Ella Mae Dolon

Email Address

engrsarah2@gmail.com


Company (Optional)

Enter your company name

May 17, 2025

Client Find Talents Section

- This is the search bar located in the "Find Talents" section of the client dashboard. It allows users to quickly search for professionals by entering the name of a specific freelancer who’s portfolio they want to review. The feature is designed to make talent discovery fast and efficient, helping clients connect with the right individuals for their projects or job opportunities



Search for talents by name, expertise or skills...

Screenshot of JS code used for API connection

Login and Register

- This code is used to simplify the sign-in process by allowing users to log in with their Google accounts instead of creating a new one. It uses Google's OAuth 2.0 system to securely get basic user information, like name and email, which is then sent to the backend for account handling. This approach improves user experience by making sign-up faster and more secure, while also reducing the need to manage passwords or manual form inputs. It also ensures a smooth flow between authentication and user account processing on the server side.

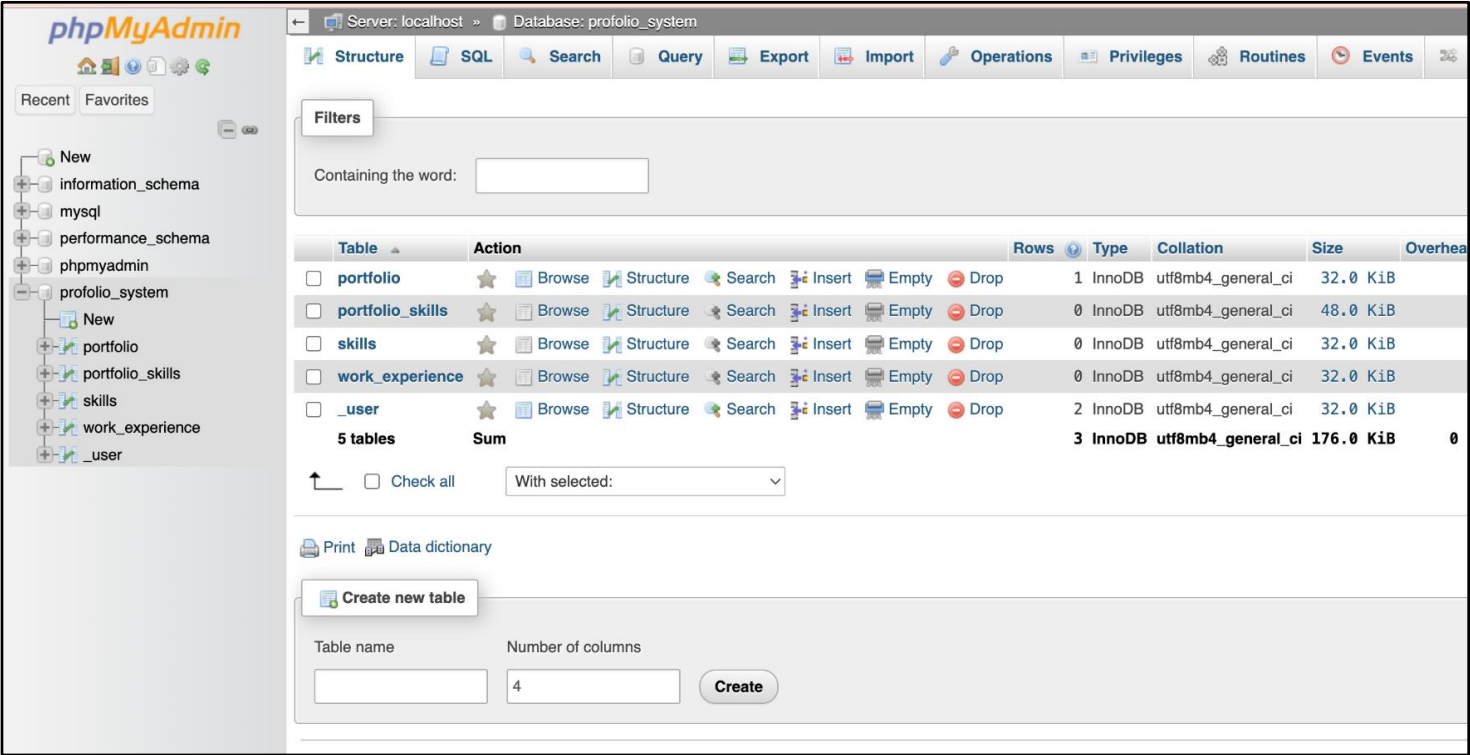
```
201 function handleGoogleSignIn() {
202   const client = google.accounts.oauth2.initTokenClient({
203     client_id: '524108211758-j11nlvkhu866ub9m7024aecundfrbu51.apps.googleusercontent.com',
204     scope: 'profile email',
205     callback: (response) => {
206       fetch('https://www.googleapis.com/oauth2/v3/userinfo', {
207         headers: {
208           'Authorization': `Bearer ${response.access_token}`
209         }
210       })
211       .then(res => res.json())
212       .then(profile => {
213         console.log('Google profile:', profile);
214
215         // Prepare data
216         const params = new URLSearchParams();
217         params.append("google_signup", "1");
218         params.append("first_name", profile.given_name || '');
219         params.append("last_name", profile.family_name || '');
220         params.append("email", profile.email);
221
222         // Send data to PHP backend
223         const xhr = new XMLHttpRequest();
224         xhr.open("POST", "", true);
225         xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

```
227     xhr.onreadystatechange = function () {
228       if (xhr.readyState === XMLHttpRequest.DONE) {
229         const res = JSON.parse(xhr.responseText);
230
231         if (xhr.status === 200) {
232           if (res.status === "success") {
233             alert("Google Sign-Up Successful!");
234             window.location.href = "dashboard.php"; // redirect new user
235           } else if (res.status === "redirect") {
236             alert(res.message);
237             window.location.href = res.redirect_url; // redirect existing user to login.php
238           } else {
239             alert("Google Sign-Up Failed: " + res.message);
240           }
241         } else {
242           alert("Something went wrong.");
243         }
244       }
245     };
246   }
247 }
```

Screenshots of XAMPP (MySQL + phpMyAdmin)

document inserted

The `profolio_system` database is designed to support a dynamic portfolio management application by organizing and maintaining key user-related data. At its core is the `_user` table, which stores essential user account information and acts as a reference point for connecting other data throughout the system. The `portfolio` table manages details related to individual user portfolios, including overarching information that defines a user's professional profile. To represent user abilities, the `skills` table contains a list of predefined skills, while the `portfolio_skills` table creates a many-to-many relationship between portfolios and skills, allowing users to associate multiple skills with each portfolio. The `work_experience` table records users' professional histories, such as job titles, employers, durations, and descriptions of roles held. Complementing this, the `work_samples` table holds references to user projects or samples that serve as practical demonstrations of their experience and expertise. Together, these tables form a relational framework that enables comprehensive and structured portfolio data management.



This shows a MySQL database table displayed in phpMyAdmin (part of the XAMPP stack), containing user data. Each row represents a user record, including fields like id, first_name, last_name, email, password, country, role, created_at, summary, and job_title. This data appears to be the result of user registration or profile creation, likely collected via a web form. The password field contains a securely hashed password, which indicates good security practice. The created_at timestamps show when the entries were added. This table helps manage user profiles in a structured way, useful for applications like job platforms, portfolios, or content management systems.

id	first_name	last_name	email	password	country	role	created_at	summary	job_title
82	Victor Ronel	Lumayad	vplumayad@gmail.com			freelancer	2025-04-27 18:39:30	nye nyee	Web Dev
83	viron	lumayad	viron@gmail.com	\$2y\$10\$ky52HmOXMYOMxtQRX9aWHOcyjPu0nntzGSO8P3zUZzl...	Philippines	freelancer	2025-04-27 18:51:01	ahahahhhahaahahahah	CYBERSECURITY

portfolio

This table holds the main portfolio entries for each user. Each record is associated with a specific user via a foreign key (**user_id**). The table typically includes a title and a description, allowing users to name and explain their portfolio. This helps users organize their work into distinct themes or showcases that highlight their professional or creative achievements.

Server: localhost » Database: profolio_system » Table: portfolio

Table structureRelation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	portfolio_id	int(11)		No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2	user_id	int(11)		Yes	NULL			Change Drop More
<input type="checkbox"/>	3	name	varchar(255) utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	4	category	varchar(255) utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	5	description	text utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	6	created_at	timestamp		No	current_timestamp()			Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext Add to central columns Remove from central columns

portfolio_skills

The **portfolio_skills** table is a linking table that establishes a many-to-many relationship between portfolios and skills. Each entry connects a portfolio to a specific skill using foreign keys referencing the **portfolio** and **skills** tables. This setup allows users to tag multiple skills to a single portfolio and reuse the same skill across multiple portfolios, creating a flexible and scalable skill-tagging system.

Server: localhost » Database: profolio_system » Table: portfolio_skills

Table structureRelation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	id	int(11)		No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2	portfolio_id	int(11)		Yes	NULL			Change Drop More
<input type="checkbox"/>	3	skill_id	int(11)		Yes	NULL			Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext Add to central columns Remove from central columns

skills

The **skills** table acts as a centralized repository of all possible skills a user can possess. Each skill has a unique ID and a name, and it may also include a category field to group skills. By managing skills in a separate table, the system ensures consistent tagging and easier management of user competencies across the platform.

Server: localhost » Database: profolio_system » Table: skills

Table structureRelation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	skill_id	int(11)		No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2	skill_name	varchar(255) utf8mb4_general_ci		Yes	NULL			Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext Add to central columns Remove from central columns

work_experience

The **work_experience** table documents the professional background of each user. It includes the company name, job title, start and end dates, and a description of the role or responsibilities. This allows users to showcase their employment history and provide context about their career journey, helping potential clients or employers understand their qualifications and growth.

Server: localhost » Database: profolio_system » Table: work_experience

Table structure

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 experience_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 portfolio_id	int(11)			Yes	NULL			Change Drop More
<input type="checkbox"/>	3 title	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	4 company	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	5 start_date	date			Yes	NULL			Change Drop More
<input type="checkbox"/>	6 end_date	date			Yes	NULL			Change Drop More
<input type="checkbox"/>	7 description	text	utf8mb4_general_ci		Yes	NULL			Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext Add to central columns

Remove from central columns

_user

The **_user** table serves as the core of the system, storing essential user profile data. It includes fields such as user ID, name, email, password, profile picture, and a short bio. This table connects to most other tables in the database, acting as the foundation for building and managing user accounts and their related portfolio content. Every user in the system can own portfolios, upload work samples, list their work experiences, and associate specific skills with their profiles.

Table structure

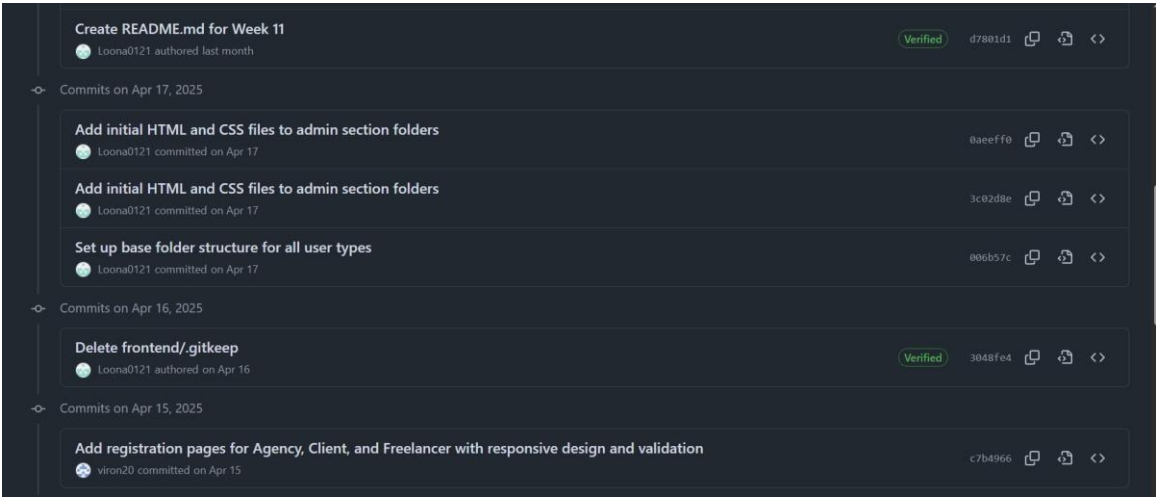
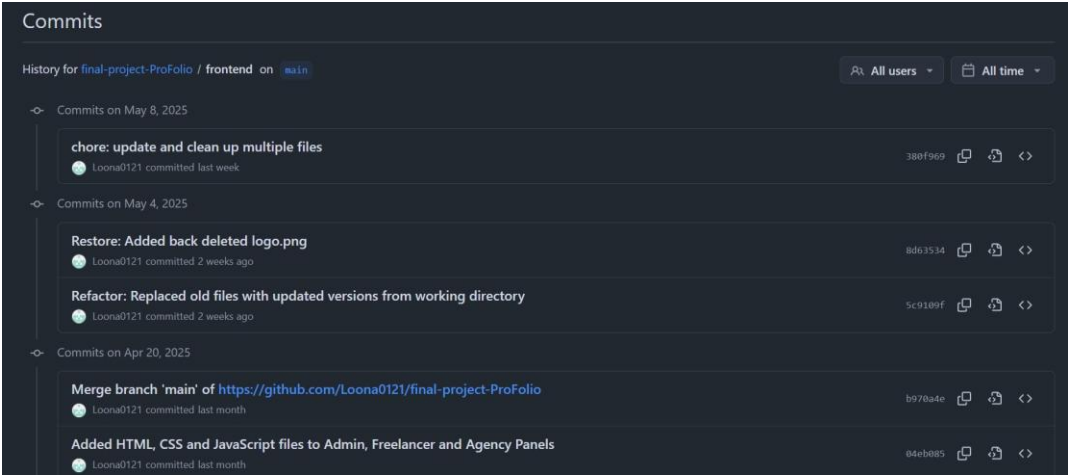
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 first_name	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	3 last_name	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	4 email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	5 password	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	6 country	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	7 role	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	8 created_at	timestamp			No	current_timestamp()			Change Drop More
<input type="checkbox"/>	9 summary	text	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	10 job_title	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	11 profile_photo	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	12 updated_at	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More
<input type="checkbox"/>	13 company_name	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More

☐ Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext Add to central columns

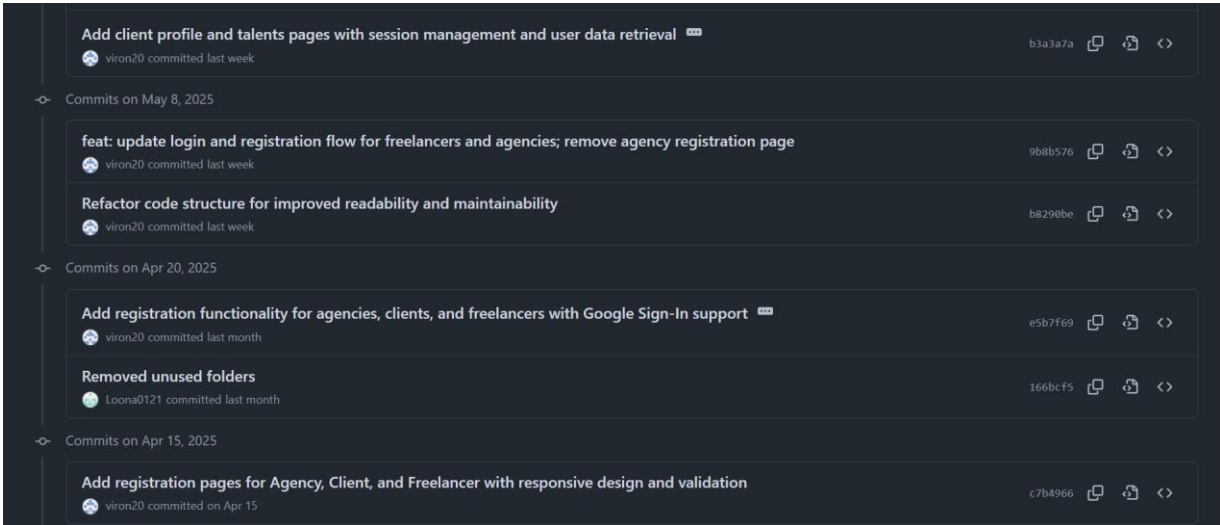
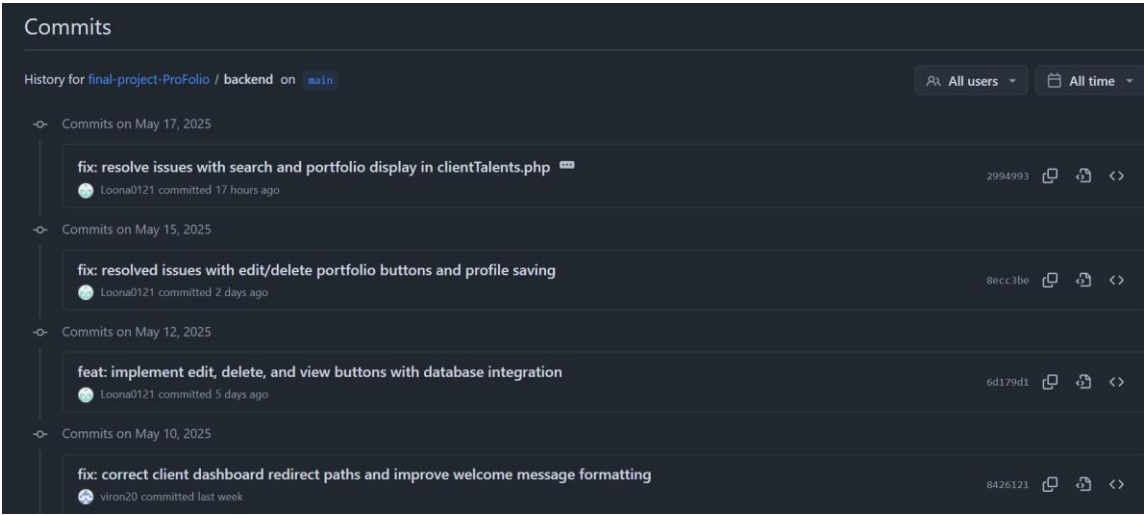
Remove from central columns

Screenshots of GitHub Commits (Frontend and Backend)

Frontend



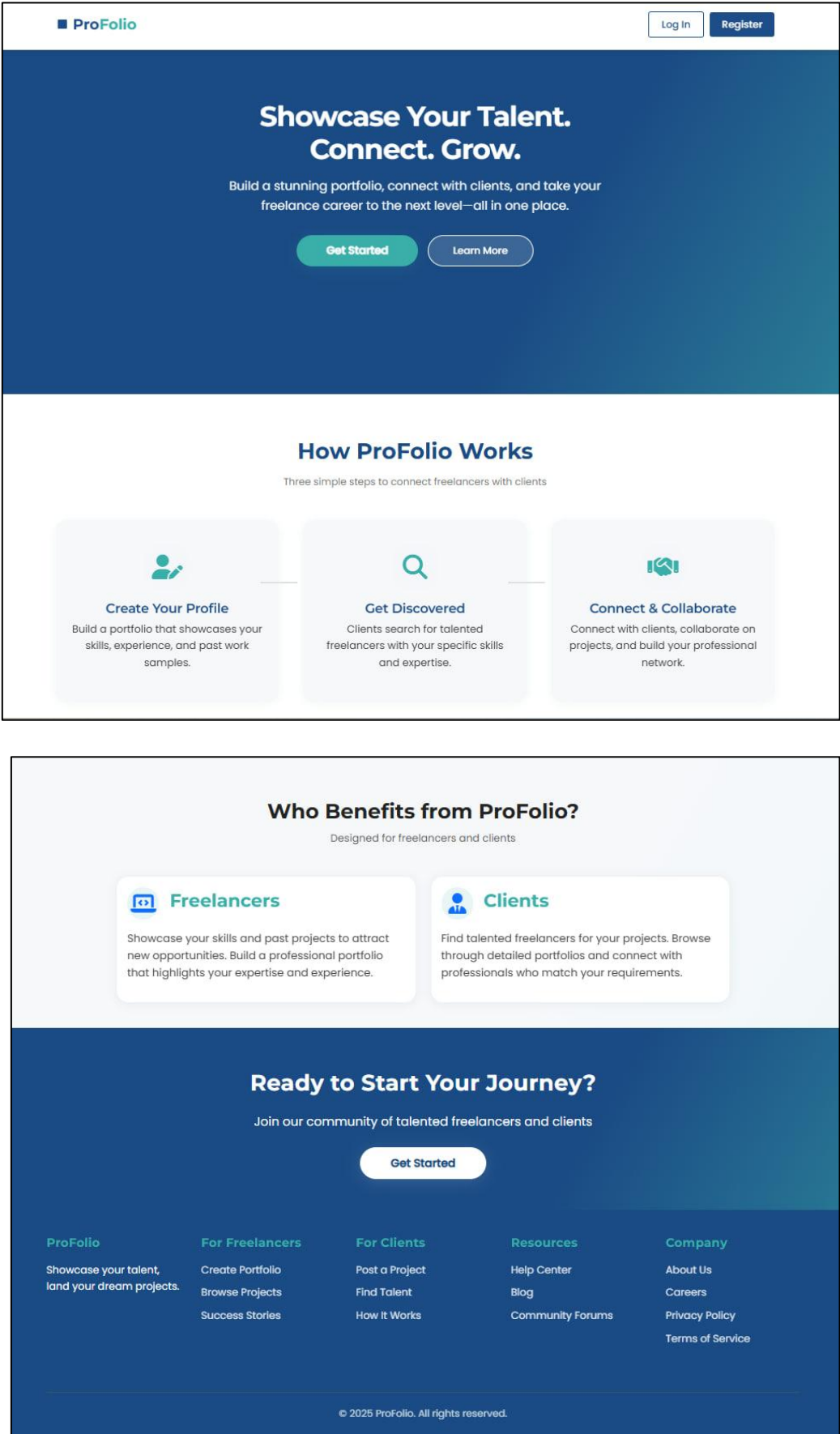
Backend



Screenshots of At Least 2 Display Pages

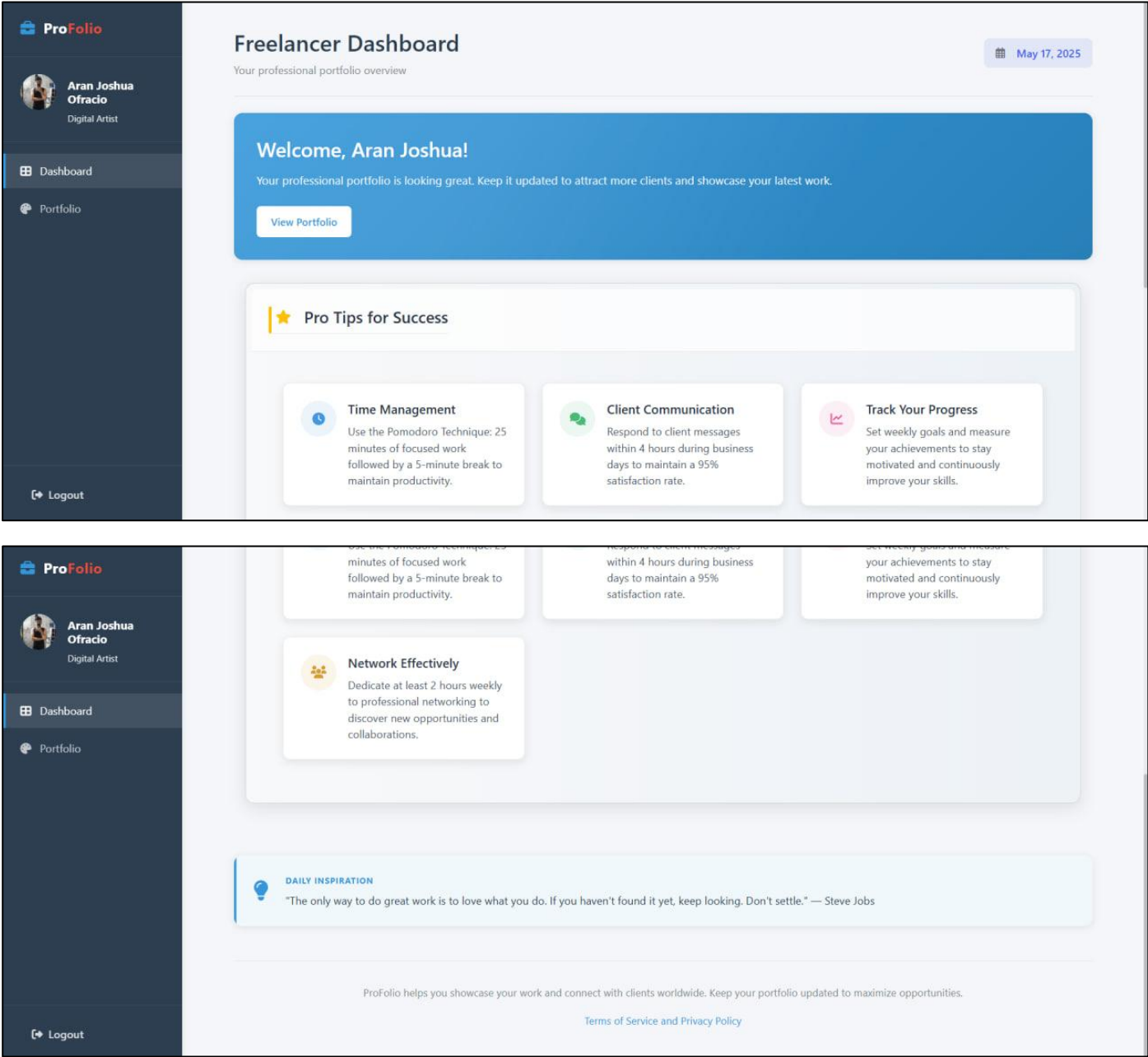
Landing Page

- The Landing Page of the ProFolio web application serves as the welcoming gateway for both freelancers and clients. Designed with clarity and ease-of-use in mind, it features a bold call-to-action encouraging users to showcase their talent, connect with others, and grow professionally. At the top, visitors can quickly register or log in, while the center highlights the platform’s core value, building a standout portfolio and getting discovered. Scrolling down, users are guided through a simple overview of how ProFolio works, with clean icons and concise steps.



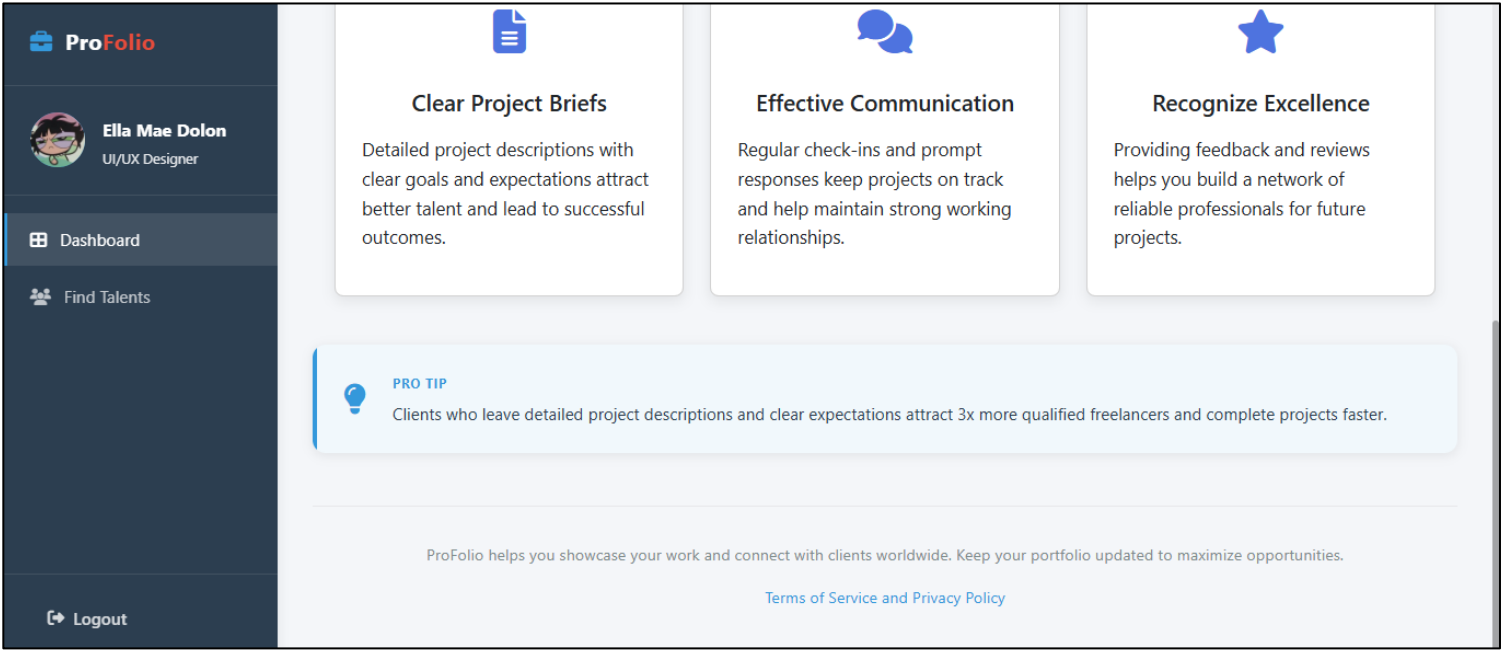
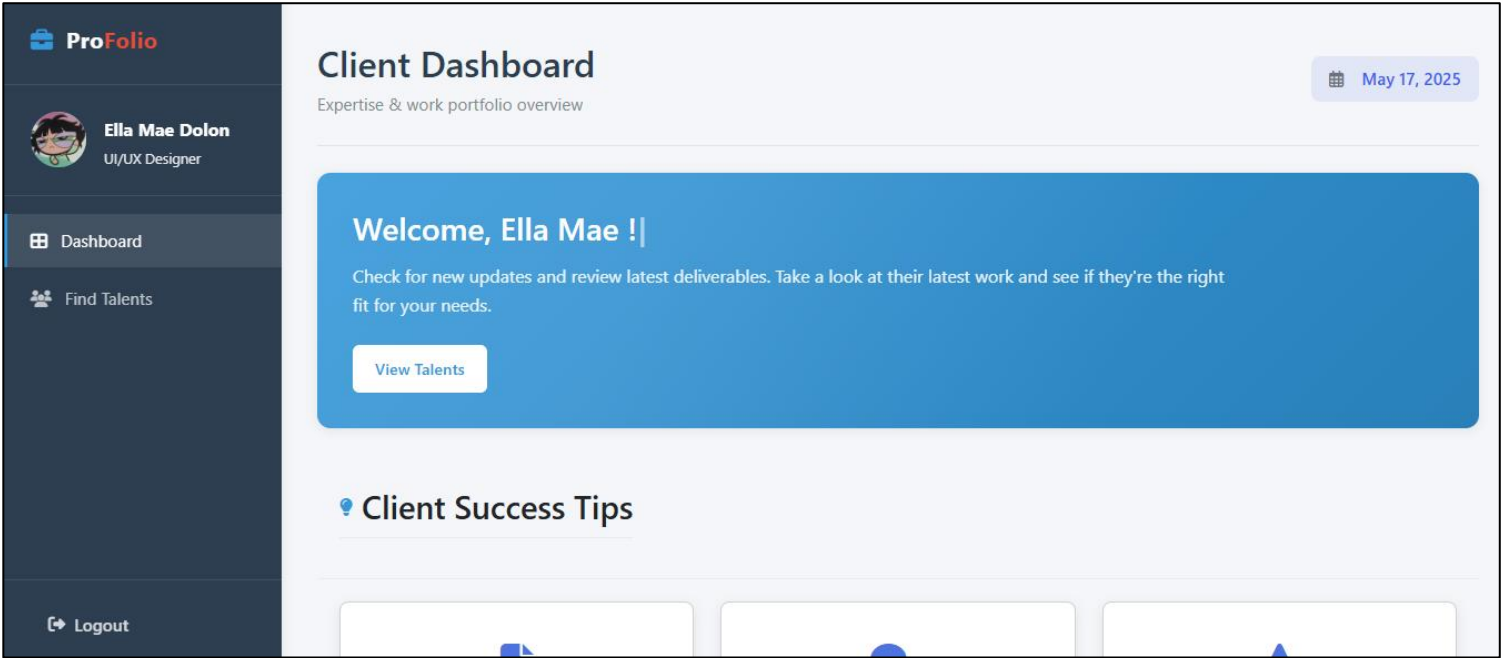
Freelancer Dashboard

- The Freelancer Dashboard of the ProFolio web application, designed specifically for freelancers using the platform. Similar to the client version, the dashboard features a personalized welcome message at the top to greet the freelancer and provide a quick overview of their workspace. On the left-hand sidebar, there are key navigation options including “Dashboard” and “Portfolio”, allowing freelancers to easily manage their profile, showcase their work, and browse available opportunities. The main section also includes a “Pro Tip for Success”, offering helpful guidance or best practices to improve the freelancer’s experience and chances of landing jobs. The layout emphasizes usability and efficiency, supporting freelancers in tracking tasks and maintaining a strong presence on the platform.



Client Dashboard

- The Client Dashboard page of the ProFolio web application, serves as the main interface for clients using the platform, providing them with an overview of their workspace. On the left-hand sidebar, users can navigate between different sections such as "Dashboard" and "Find Talents". The center of the page welcomes the client by name and encourages them to review updates and deliverables. There are action buttons labeled "View Talents" to help clients quickly access relevant sections. The dashboard also includes a section for Client Success Tips, aimed at guiding users on how to get the most out of the platform. This display page highlights a clean, user-friendly design focused on accessibility and functionality.



Sample Backend API Response

Login

- This backend script manages user authentication through two methods: traditional email/password login and Google sign-in. For standard login, it checks the submitted credentials against the database using prepared statements to ensure security. If the user exists and the password matches, it starts a session, stores the user's ID, email, and role, and redirects them to the appropriate dashboard based on their role (e.g., freelancer or client). If authentication fails, it sets an error message and redirects back to the login page. For Google login, it checks if the user's email already exists in the system. If it does, the user is logged in and redirected just like a regular login. If the user is new, their Google profile information is stored temporarily in the session, and they are redirected to a registration page to complete their signup. This approach ensures secure login, supports role-based access, and provides a smooth flow for both existing and new users.

```
6  if ($_SERVER["REQUEST_METHOD"] === "POST" && !isset($_POST['google_signup'])) {
7      $email = $_POST['email'];
8      $password = $_POST['password'];
9
10     $stmt = $con->prepare("SELECT * FROM _user WHERE email = ?");
11     $stmt->bind_param("s", $email);
12     $stmt->execute();
13     $result = $stmt->get_result();
14
15     if ($result && $result->num_rows > 0) {
16         $user = $result->fetch_assoc();
17
18         // Verify the hashed password
19         if (password_verify($password, $user['password'])) {
20             $_SESSION['email'] = $user['email'];
21             $_SESSION['role'] = $user['role'];
22             $_SESSION['id'] = $user['id'];
23
24             $redirect = match ($user['role']) {
25                 'freelancer' => '../FreelancerDashboard/freelancerDashboard.php',
26                 'client' => '../clientDashboard/clientDashboard.php',
27                 default => 'login.php'
28             };
29
30             header("Location: $redirect");
31             exit();
32         } else {
33             $_SESSION['error'] = "Invalid email or password.";
34         }
35     }
```

```
36     } else {
37         $_SESSION['error'] = "Invalid email or password.";
38     }
39
40     header("Location: login.php");
41     exit();
42 }
43 if ($_SERVER["REQUEST_METHOD"] === "POST" && isset($_POST['google_signup'])) {
44     header('Content-Type: application/json');
45
46     $email = $_POST['email'];
47     $firstName = $_POST['first_name'];
48     $lastName = $_POST['last_name'];
49
50     // Check if user exists
51     $stmt = $con->prepare("SELECT * FROM _user WHERE email = ?");
52     $stmt->bind_param("s", $email);
53     $stmt->execute();
54     $result = $stmt->get_result();
55
56     if ($result->num_rows > 0) {
57         $user = $result->fetch_assoc();
58         $_SESSION['email'] = $user['email'];
59         $_SESSION['role'] = $user['role'];
60         $_SESSION['id'] = $user['id'];
```

```
62         // Redirect based on role
63         $redirect = match ($user['role']) {
64             'freelancer' => '../FreelancerDashboard/freelancerDashboard.php',
65             'client' => '../clientDashboard/clientDashboard.php',
66             default => 'login.php'
67         };
68
69         echo json_encode([
70             'status' => 'redirect',
71             'message' => 'Welcome!!',
72             'redirect_url' => $redirect
73         ]);
74         exit();
75     } else {
76         // New user, redirect to registration
77         $_SESSION['google_email'] = $email;
78         $_SESSION['google_first_name'] = $firstName;
79         $_SESSION['google_last_name'] = $lastName;
80
81         echo json_encode([
82             'status' => 'redirect',
83             'message' => 'Complete your registration.',
84             'redirect_url' => 'register.php'
85         ]);
86         exit();
87     }
88 }
```


Register

- This backend script handles user registration through two flows: Google sign-up and standard manual registration. When a Google sign-up request is detected, it retrieves the user's first name, last name, and email, and checks if that email already exists in the database. If not, it inserts a new user record with an empty password and country (since Google doesn't provide those directly), and assigns a default role of "client". If the email already exists, it prevents duplicate sign-ups and redirects the user to the login page via a JSON response. For traditional registration, the script collects user input such as name, email, password, and country. It first checks if the email is already registered. If it is, it alerts the user. If not, it hashes the password for security and stores the new user details in the database with the provided role. A success message is then shown, and the user is redirected to the login page. Overall, the script ensures that new users can register securely and prevents duplicate accounts whether registering manually or through Google.

```
10 if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['google_signup'])) {
11     $firstName = isset($_POST['first_name']) ? mysqli_real_escape_string($con, $_POST['first_name']) : '';
12     $lastName = isset($_POST['last_name']) ? mysqli_real_escape_string($con, $_POST['last_name']) : '';
13     $email = isset($_POST['email']) ? mysqli_real_escape_string($con, $_POST['email']) : '';
14     $role = 'client';
15
16     if (!empty($email)) {
17         $checkUserSql = "SELECT * FROM _user WHERE email = '$email'";
18         $result = $con->query($checkUserSql);
19
20         if ($result->num_rows === 0) {
21             // No user yet → proceed with Google sign-up
22             $sql = "INSERT INTO _user (first_name, last_name, email, password, country, role)
23                 VALUES ('$firstName', '$lastName', '$email', '', '', '$role')";
24             $con->query($sql);
25             echo json_encode(['status' => 'success', 'message' => 'Signed in with Google!', 'redirect_url' => 'login.php']);
26             exit;
27         } else {
28             // Email already exists → do NOT allow Google sign-up again
29             echo json_encode([
30                 'status' => 'redirect',
31                 'message' => 'Email already registered. Redirecting to login...',
32                 'redirect_url' => 'login.php'
33             ]);
34         }
35         exit;
36     }
37 }
38 }
```

```
44 if ($_SERVER["REQUEST_METHOD"] == "POST") {
45     // Retrieve POST data
46     $firstName = isset($_POST['first_name']) ? mysqli_real_escape_string($con, $_POST['first_name']) : '';
47     $lastName = isset($_POST['last_name']) ? mysqli_real_escape_string($con, $_POST['last_name']) : '';
48     $email = isset($_POST['email']) ? mysqli_real_escape_string($con, $_POST['email']) : '';
49     $password = isset($_POST['password']) ? password_hash($_POST['password'], PASSWORD_DEFAULT) : '';
50     $country = isset($_POST['country']) ? mysqli_real_escape_string($con, $_POST['country']) : '';
51
52     // Check if the email already exists
53     $checkEmailSql = "SELECT * FROM _user WHERE email = '$email'";
54     $result = $con->query($checkEmailSql);
55
56     if ($result->num_rows > 0) {
57         echo "<script>alert('Email already exists. Please use a different email.');";
58     } else {
59         // Insert the data if email doesn't exist
60         $sql = "INSERT INTO _user (first_name, last_name, email, password, country, role)
61             VALUES ('$firstName', '$lastName', '$email', '$password', '$country', '$role')";
62
63         if ($con->query($sql)) {
64             echo "<script>alert('Registration successful! Please login using your account.');"; window.location.href='login.php';</script>";
65         } else {
66             echo "Error: " . $con->error;
67         }
68     }
69 }
70 ?>
```

Annotated JavaScript Snippet Used for Rendering Data

- This code listens for the server's response after a Google sign-up request is sent. Once the response is complete, it checks if the request was successful. If the backend indicates a new user, it shows a success message and redirects to the dashboard. If the user already exists, it displays a message and redirects to the appropriate page. If there's an error, it shows an alert.

Set up a function to handle changes in the XMLHttpRequest state.

```
xhr.onreadystatechange = function () {  
  if (xhr.readyState === XMLHttpRequest.DONE) {  
    const res = JSON.parse(xhr.responseText);
```

Check if the request has completed (state 4 = DONE).

Parse the JSON response from the server into a JavaScript object.

Check if the HTTP response status is 200 (OK/success).

If the backend responded with status 'success' (new user registered).

```
if (xhr.status === 200) {  
  if (res.status === "success") {  
    alert("Google Sign-Up Successful!");  
    window.location.href = "dashboard.php"; // redirect new user
```

Show a success alert to the user.

Redirect the newly registered user to the dashboard page.

If the backend indicates this is an existing user.

Redirect the user to the appropriate page (usually login.php).

```
} else if (res.status === "redirect") {  
  alert(res.message);  
  window.location.href = res.redirect_url; // redirect existing user to login.php  
} else {  
  alert("Google Sign-Up Failed: " + res.message);  
}
```

If the backend returned an unexpected status or error.

Show an error message from the backend.

Show the backend-provided message (e.g., "Email already registered").

Show a general error alert.

```
} else {  
  alert("Something went wrong.");  
}
```

If the HTTP status is not 200 (something went wrong server-side).

• Client getting information of the freelancers

Sends a request to the backend URL stored in ajaxUrl.

Checks if the HTTP response is OK (status 200–299). If not, throws an error with status code and message for debugging.

```
// Fetch portfolios via AJAX from the server
fetch.ajaxUrl)
.then(response => {
  console.log('AJAX response received:', response);
  if (!response.ok) {
    throw new Error(`Network response was not ok: ${response.status} ${response.statusText}`);
  }
  return response.json();
})
.then(data => {
  console.log('Portfolio data received:', data);
  portfolioContainer.innerHTML = '';

  // Check if response has expected structure
  if (!data || typeof data !== 'object') {
    throw new Error('Invalid response format received from server');
  }

  const portfolios = data.portfolios || [];
  console.log('Found', portfolios.length, 'portfolios');
```

Converts the response into a usable JavaScript object.

Ensures the data is in the expected object format before using it. Prevents rendering errors if the server returns unexpected content.

Gets the portfolios array from the response, or assigns an empty array if not found.

Clears existing content in the portfolioContainer to prepare for fresh data rendering.

Useful for developers to verify how many items were loaded and to trace issues in development.

Sends an HTTP request to get portfolios for a specific freelancer using their talentId.

Converts the server's response into JSON format for further processing.

```
fetch('clientTalents.php?action=get_portfolios&user_id=' + talentId)
.then(response => response.json())
.then(data => {
  console.log('Portfolio data received:', data);
  const portfolios = data.portfolios || [];
  portfolioContainer.innerHTML = '';

  if (portfolios.length === 0) {
    portfolioContainer.innerHTML = `
      <div class="col-12 text-center py-5">
        <i class="fas fa-folder-open fa-3x text-muted mb-3"></i>
        <h5 class="text-muted">No portfolios available</h5>
        <p>This freelancer hasn't created any portfolios yet.</p>
      </div>
    `;
    return;
  }
});
```

Extracts the portfolios array from the data. If not found, defaults to an empty array to avoid errors. Clears any existing HTML in the container.

Checks if there are no portfolios. If true, inserts a styled message into the portfolioContainer to inform the user that no portfolios are available