CSIT128 Assignment 2

DELIVERABLE + REPORT

INTERNLINKT

Assignment Cover Sheet

Subject Code:	CSIT128
Subject Name:	Introduction to Web Technology
Submission Type:	Report
Assignment Title:	Assignment 2
Student Name:	Fatima Mirza Mariam Noor Kulsoom Mubeen Sushma Geddam
Student Number:	8909143 8888395 8972631 9071994
Student Email:	
Lecturer Name:	Dr Soly Mathew Biju Dr Haitham Yaish
Due Date:	22 June 2025
Date Submitted:	22 June 2025

PLAGIARISM:	**DECLARATION:**
The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism. The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation. **PLEASE NOTE:** STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED	I/We certify that this is entirely my/our own work, except where I/we have given fully-documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism. **Signature of Student:**

Optional Marks:

Comments:

Lecturer Assignment Receipt (To be filled in by student and retained by Lecturer upon return of assignment)

Subject:	**Assignment Title:**
Student Name:	**Student Number:**
Due Date:	**Date Submitted:**

Table of Contents

Problem Statement and Description of Website

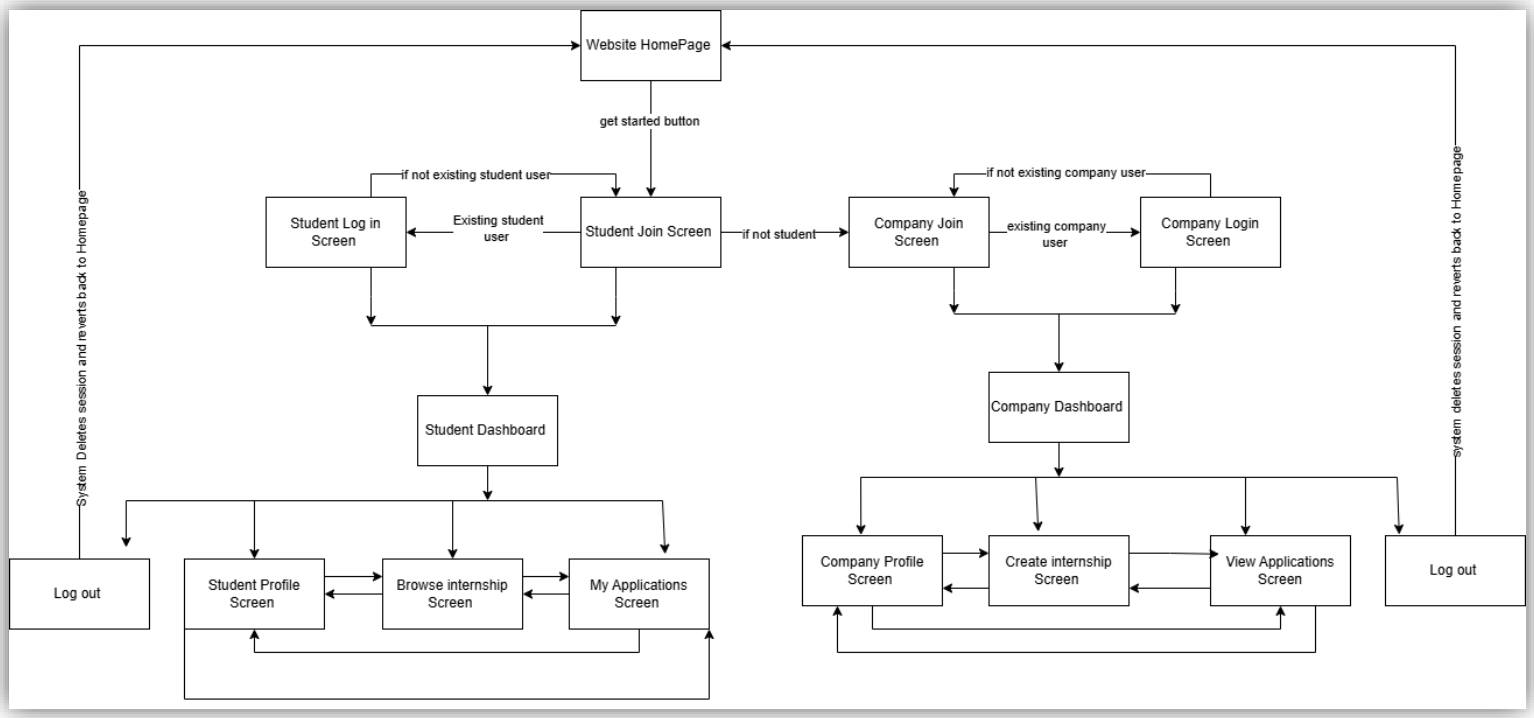
In a generation where everything is just a click away, the overwhelming process of finding the perfect internship remains outdated, exhausting, and melancholy. Current students and fresh graduates are often forced to roam around offices, career fairs printing and handing out endless CVS only to hear an unsure reply “We will let you know soon” at the same time Companies receive scattered applications, lack a structured system where work is maintained well and miss out on promising talents due to an unorganized process. In addition to wasting time and effort, this manual hustle puts obstacles in the way of opportunity and desire.

InternLinkt was created to break those obstacles. No more Office Hopping, no more Paper Trails, Just Connecting Interns to their Dream Internship!

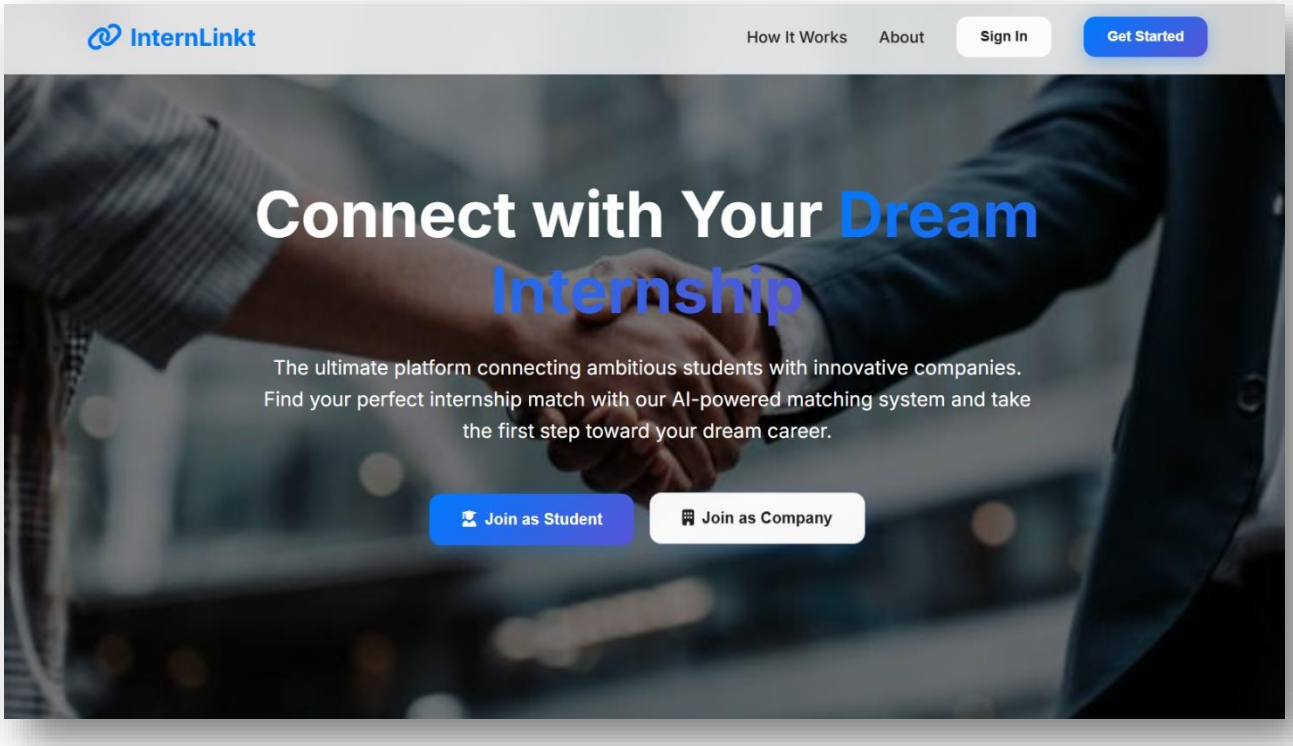
InternLinkt is a dynamic online platform for managing internships that aims to make connections between businesses and students more intelligent and effective. It offers specialized dashboards for both businesses and students and was constructed with HTML, CSS, JavaScript, Node.js, and MySQL.

From the comfort of their own homes, students may register, edit their profiles, upload resumes, look through internships, apply, and monitor the status of their applications at any time and from any location. In addition to managing applicants by changing their status as accepted, rejected, or under review, companies can register, post internships, and evaluate applications, and get in touch with the best applicants—all inside a single, intelligent interface.

The design and logic of the User Interface flow.



Website Display/logos



InternLinkt

How It Works About Sign In Get Started

About InternLinkt

InternLinkt is revolutionizing the way students connect with internship opportunities. We believe every student deserves access to meaningful work experiences that shape their future careers.

Founded with the mission to bridge the gap between talented students and innovative companies, InternLinkt combines cutting-edge technology with human-centered design to create seamless internship matching experiences.

95%
Success Rate

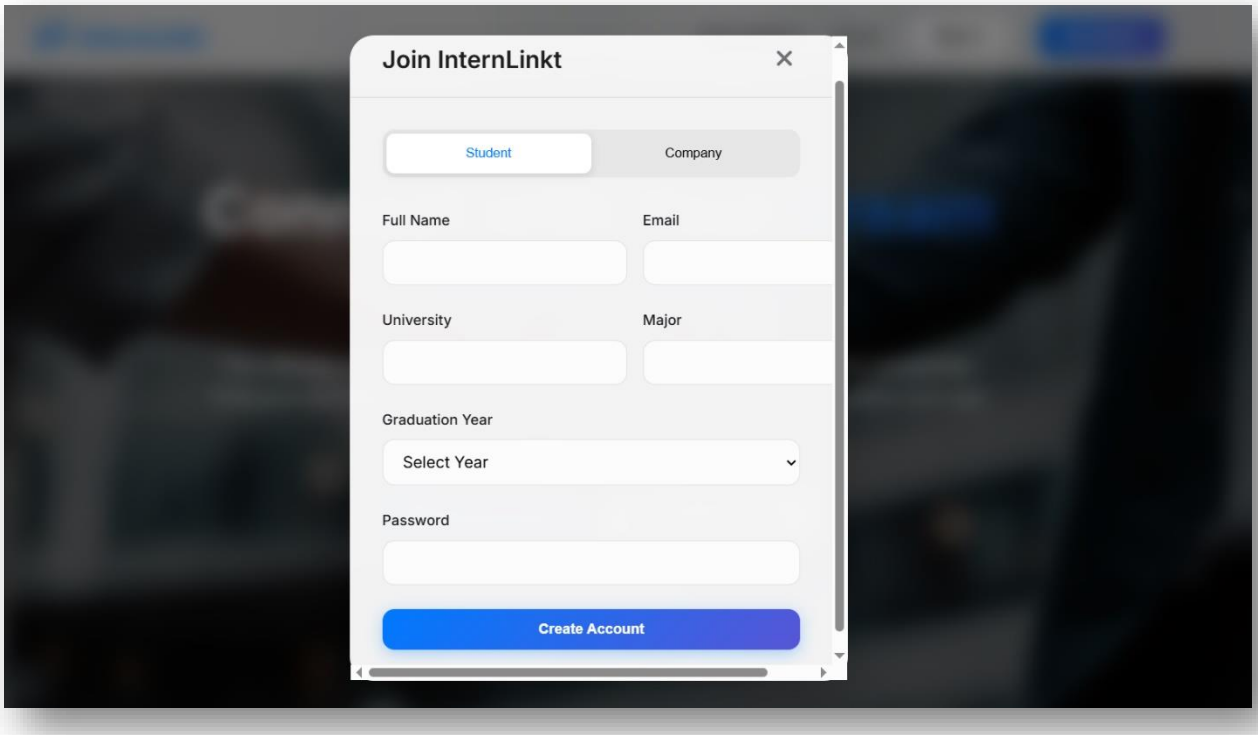
24hr
Average Response Time

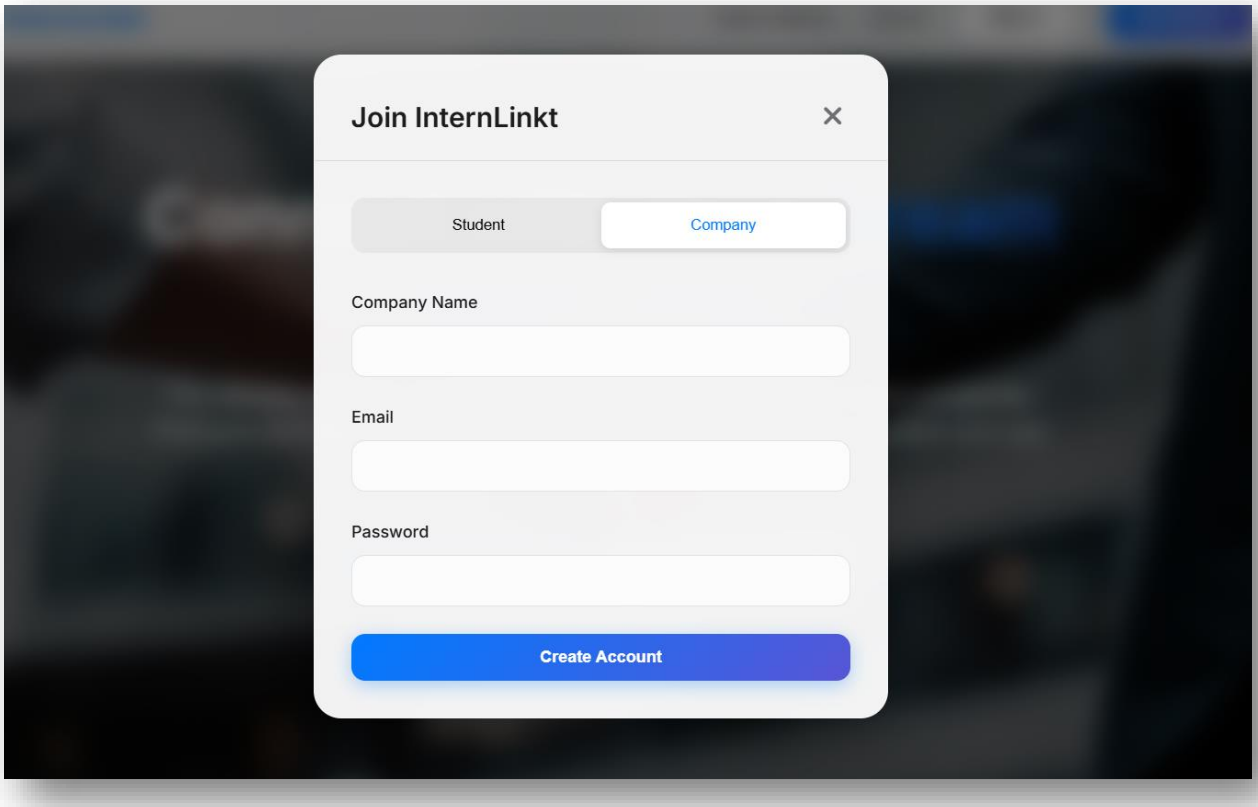
50+
Universities Partnered

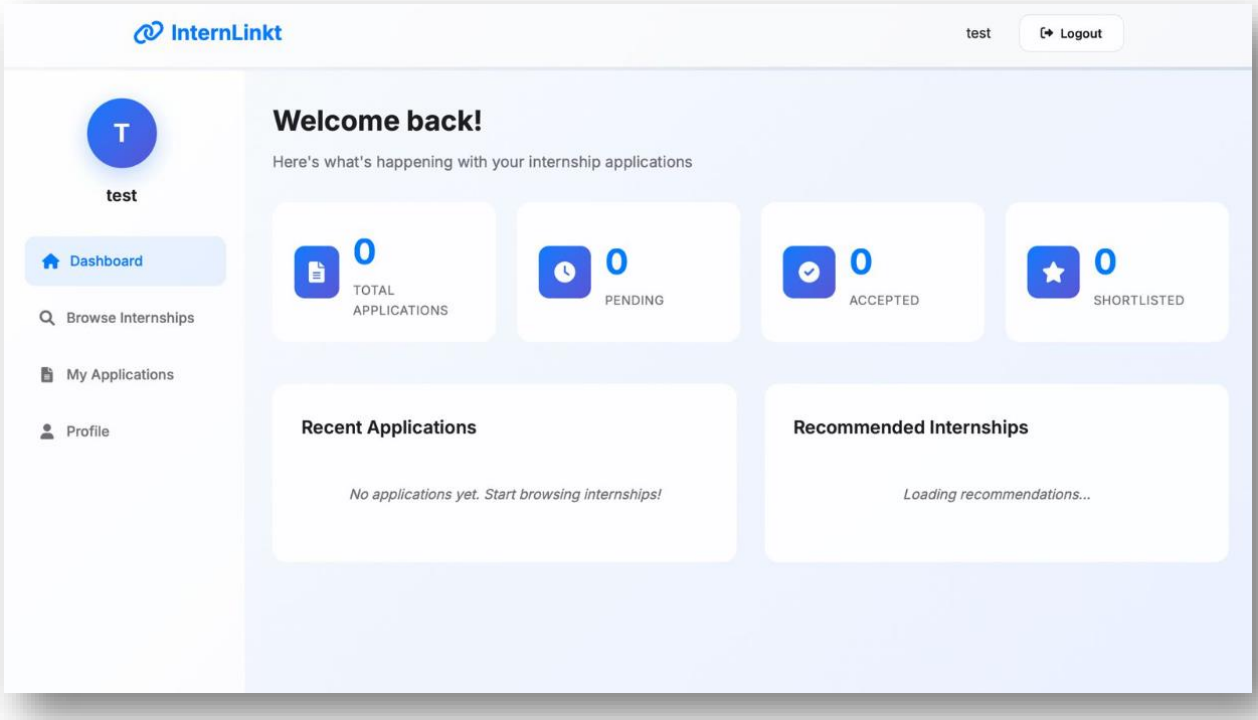
Trusted by Leading Companies

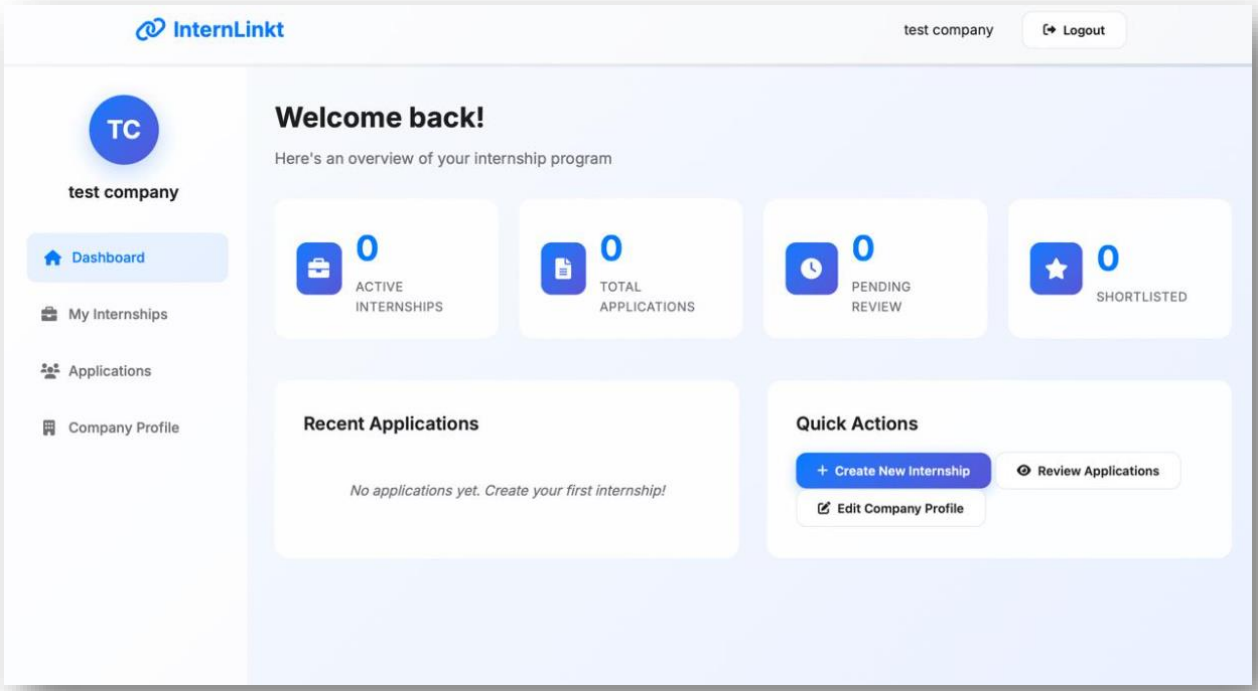
TechCorp InnovateLab

StartupXYZ FutureTech




Join InternLinkt
Student
Company
Company Name
Email
Password
Create Account


InternLinkt
test
Logout
T
test
Dashboard
Browse Internships
My Applications
Profile
Welcome back!
Here's what's happening with your internship applications
0
TOTAL APPLICATIONS
0
PENDING
0
ACCEPTED
0
SHORTLISTED
Recent Applications
No applications yet. Start browsing internships!
Recommended Internships
Loading recommendations...



HTML

Use of <meta>

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This ensures the website is responsive on any device.

Use of <nav>

```
<!-- Navigation -->
<nav class="navbar">
  <div class="nav-container">
    <div class="nav-logo">
      <i class="fas fa-link"></i>
      <span>InternLinkt</span>
    </div>
    <div class="nav-links">
      <a href="#how-it-works">How It Works</a>
      <a href="#about">About</a>
      <button class="btn-secondary" onclick="showLoginModal()">Sign In</button>
      <button class="btn-primary" onclick="showSignupModal()">Get Started</button>
    </div>
  </div>
</nav>
```

The website's primary navigation area is defined by the <nav> elements. It makes it easier for visitors to navigate between the homepage, about us page , and student company login, among other areas of the website.

About us section and Stats

```
<section id="about" class="about">
  <div class="container">
    <div class="about-content">
      <div class="about-text animate-on-scroll">
        <h2>About InternLinkt</h2>
        <p class="about-lead">
          InternLinkt is revolutionizing the way students connect with internship opportunities.
          We believe every student deserves access to meaningful work experiences that shape their future careers.
        </p>
        <p>
          Founded with the mission to bridge the gap between talented students and innovative companies,
          InternLinkt combines cutting-edge technology with human-centered design to create seamless
          internship matching experiences.
        </p>
        <div class="about-stats">
          <div class="about-stat">
            <span class="about-stat-number">95%</span>
```

About section describes the features of our website to users using headings paragraphs

About stats show impressive platform statistics, such as the number of partners and users. Aids in establishing credibility and trust with guests.

Use of <footer>

```
    <div class="footer-bottom">
      <p>&copy; 2025 InternLinkt. All rights reserved.</p>
    </div>
  </div>
</footer>
```

The Footer sets a professional copyright notice stating that InternLinkt is the owner of the website's content completes the page. In addition to providing consumers with a clear indication that they have reached the end of the page; it guarantees legal protection and improves professionalism.

Student Dashboard page

```
<div class="dashboard" id="dashboard-content" style="display: none;">
  <!-- Sidebar -->
  <div class="sidebar">
    <div class="avatar" id="user-avatar">JD</div>
    <div class="sidebar-name" id="sidebar-name">John Doe</div>
    <nav class="sidebar-nav">
      <a href="#dashboard" class="sidebar-link active" onclick="showSection('dashboard')">
        <i class="fas fa-home"></i>
        Dashboard
```

The sidebar provides quick access to sections like DashBoards, application and profile. It also shows recent stats and activities in card format

Internship browser with filters - Lets students search internships with filters.

```
<!-- Internships Section -->
<section id="internships-section" class="dashboard-section" style="display: none;">
  <div class="section-header">
    <h2>Browse Internships</h2>
    <div class="search-filters">
      <input type="text" id="search-input" placeholder="Search internships..." class="input" />
      <select id="location-filter" class="input">
        <option value="">All Locations</option>
        <option value="Remote">Remote</option>
        <option value="New York">New York</option>
        <option value="San Francisco">San Francisco</option>
        <option value="London">London</option>
      </select>
      <select id="type-filter" class="input">
        <option value="">All Types</option>
        <option value="on-site">On-site</option>
```

My application section lists the internships students have applied for and shows their status updates

```
<div id="applicationModal" class="modal">
  <div class="modal-content large">
    <div class="modal-header">
      <h2>Apply to Internship</h2>
      <button class="modal-close" onclick="closeModal('applicationModal')">
        <i class="fas fa-times"></i>
      </button>
    </div>
    <div class="modal-body">
      <form id="applicationForm" class="form">
        <input type="hidden" id="internship-id" name="internship_id" />
        <div class="form-group">
          <label>Cover Letter</label>
```

Student profile – lets students edit and update their details.

```
<!-- Profile Modal -->
<div id="profileModal" class="modal">
  <div class="modal-content">
    <div class="modal-header">
      <h2>Edit Profile</h2>
      <button class="modal-close" onclick="closeModal('profileModal')">
        <i class="fas fa-times"></i>
      </button>
    </div>
    <div class="modal-body">
      <form id="profileForm" class="form">
        <div class="form-group">
          <label>Full Name</label>
          <input type="text" name="name" required />
        </div>
        <div class="form-group">
          <label>Email</label>
          <input type="email" name="email" required />
        </div>
        <div class="form-group">
          <label>University</label>
          <input type="text" name="university" required />
        </div>
        <div class="form-group">
          <label>Major</label>
          <input type="text" name="major" required />
        </div>
        <div class="form-group">
          <label>Graduation Year</label>
          <input type="number" name="graduation_year" min="2020" max="2030" req
        </div>
        <div class="form-group">
          <label>Phone (Optional)</label>
```

Company Dashboard page

```
<!-- Dashboard Layout -->
<div class="dashboard" id="dashboard-content" style="display: none;">
  <!-- Sidebar -->
  <div class="sidebar">
    <div class="avatar" id="user-avatar">CD</div>
    <div class="sidebar-name" id="sidebar-name">Company Name</div>
    <nav class="sidebar-nav">
      <a href="#dashboard" class="sidebar-link active" onclick="showSection('dashboard')">
        <i class="fas fa-home"></i>
        Dashboard
      </a>
      <a href="#internships" class="sidebar-link" onclick="showSection('internships')">
        <i class="fas fa-briefcase"></i>
        My Internships
      </a>
      <a href="#applications" class="sidebar-link" onclick="showSection('applications')">
        <i class="fas fa-users"></i>
        Applications
      </a>
      <a href="#profile" class="sidebar-link" onclick="showSection('profile')">
        <i class="fas fa-building"></i>
        Company Profile
      </a>
    </nav>
  </div>
```

Company Dashboard has sidebars to enable switching quickly between different Dashboard, Internships, Applications, and Profile.

Company stats displays important metrics, such as the number of applications, pending reviews, shortlisted, and ongoing internships.

```
<div class="main-content">
  <!-- Dashboard Section -->
  <section id="dashboard-section" class="dashboard-section">
    <div class="section-header">
      <h2>Welcome back!</h2>
      <p>Here's an overview of your internship program</p>
    </div>

    <div class="dashboard-stats">
      <div class="stat-card">
        <div class="stat-icon">
          <i class="fas fa-briefcase"></i>
        </div>
        <div class="stat-info">
          <span class="stat-number" id="total-internships">0</span
          <span class="stat-label">Active Internships</span>
        </div>
```

Create internships modal – create internships with all required details making sure user can view and apply accordingly.

```
<div id="internshipModal" class="modal">
  <div class="modal-content large">
    <div class="modal-header">
      <h2 id="internship-modal-title">Create New Internship</h2>
      <button class="modal-close" onclick="closeModal('internshipModal')">
        <i class="fas fa-times"></i>
      </button>
    </div>
    <div class="modal-body">
      <form id="internshipForm" class="form">
        <input type="hidden" id="internship-id" name="id">
        <div class="form-row">
          <div class="form-group">
            <label>Internship Title</label>
```

Form validation for Sign up forms

```
// Basic empty check
if (!userData.name || !userData.email || !userData.password || !userData.university || !userData.major) {
    showToast("Please fill in all required fields", "error");
    return;
}

// Email format validation
const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailPattern.test(userData.email)) {
    showToast("Invalid email format", "error");
    return;
}

// Password validation: minimum 8 characters, at least one uppercase, one lowercase, one number, and one sp
const passwordPattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;
if (!passwordPattern.test(userData.password)) {
    showToast("Password must be at least 8 characters and include uppercase, lowercase, number, and special
    return;
}

// Submit to server if valid
try {
    const response = await fetch('/api/students/signup', {
        method: 'POST',
```

We have implemented form validation to

- Make sure that the relevant fields are filled in.
- Prevent users from inputting incorrect email addresses or other formats.
- Boost security (strong passwords, for example)
- Instant feedback enhances the customer experience.

```
// Password validation: minimum 8 characters, at least one uppercase, one lowercase, one number, and one special character
```

CSS

1. **Reset and Base Styles**

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Resets default browser styles, ensuring consistent layout. box-sizing: border-box makes width/height include padding and border, preventing overflow.

```css
body {
  font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
  line-height: 1.6;
  color: #1d1d1f;
  background: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
  overflow-x: hidden;
}
```

The body has a clean sans-serif font, a subtle gradient background, and hidden horizontal overflow to maintain layout integrity.

2. Navigation Bar

```css
.navbar {
  position: fixed;
  top: 0;
  width: 100%;
  background: rgba(255, 255, 255, 0.8);
  backdrop-filter: blur(20px);
  -webkit-backdrop-filter: blur(20px);
  border-bottom: 1px solid rgba(255, 255, 255, 0.2);
  z-index: 1000;
  padding: 1rem 0;
}
```

The navigation bar creates a **sticky transparent navigation bar** with a blur effect using backdrop-filter, giving a modern "glassmorphism" feel.

The logo and links (.nav-logo, .nav-links a) are styled with spacing, hover color changes, and font weight to enhance clarity and interaction.

3. **Dashboard Layout**

```css
/* Dashboard Stats */
.dashboard-stats {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 1.5rem;
  margin-bottom: 3rem;
}
```

```css
/* Dashboard Content */
.dashboard-content {
  margin-top: 2rem;
}
```

It is a full-screen **flexbox layout** with top padding (to offset the fixed navbar). *A Guide to Flexbox* (n.d.).

The sidebar is:Fixed, vertically scrollable, semi-transparent with blur, contains avatar, username, and navigation links

Hover effects like transform: translateX(4px) and gradient animations on .sidebar-link::before make the UI feel interactive.

4. Main Content

```css
.main-content {
  flex: 1;
  margin-left: 280px;
  padding: 2rem;
  background: linear-gradient(135deg, #f7fafd 0%, #eaf1ff 100%);
  min-height: calc(100vh - 80px);
}
```

Shifts right to account for the fixed sidebar and uses a soft gradient background. This section holds the actual page content.

5. **Cards and Sections**

Sections like .stat-card, .content-card, and .internship-card use:

- backdrop-filter, rounded corners, and soft shadows

- Hover transitions: slight translateY, box shadows
- Responsive grid layouts with grid-template-columns

This creates **clean, modular blocks** ideal for dashboards or portals.

6. **Buttons**

```css
/* Buttons */
.btn-primary, .btn-secondary {
  padding: 0.75rem 1.5rem;
  border: none;
  border-radius: 12px;
  font-weight: 600;
  font-size: 0.9rem;
  cursor: pointer;
  transition: all 0.3s ease;
  text-decoration: none;
  display: inline-flex;
  align-items: center;
  gap: 0.5rem;
}
```

Buttons are visually distinct using gradients, shadows, and scaling hover effects. .btn-primary is the action button; .btn-secondary is a subtle alternative.

7. **Cards for Internships, Applications, and Profiles**

```css
.internship-card {
  background: rgba(255, 255, 255, 0.9);
  backdrop-filter: blur(20px);
  border-radius: 16px;
  padding: 1.5rem;
  margin-bottom: 1.5rem;
  border: 1px solid rgba(255, 255, 255, 0.3);
  transition: all 0.3s ease;
  position: relative;
  overflow: hidden;
}
```

```css
.application-card {
  background: rgba(255, 255, 255, 0.9);
  backdrop-filter: blur(20px);
  border-radius: 16px;
  padding: 1.5rem;
  margin-bottom: 1rem;
  border: 1px solid rgba(255, 255, 255, 0.3);
  transition: all 0.3s ease;
}
```

```css
.profile-card {
  background: rgba(255, 255, 255, 0.9);
  backdrop-filter: blur(20px);
  border-radius: 16px;
  padding: 2rem;
  border: 1px solid rgba(255, 255, 255, 0.3);
  max-width: 600px;
  margin: 0 auto;
}
```

These maintain visual consistency with modern white-transparent themes. Status labels (like .application-status.pending) use color-coded badges to indicate application state clearly.

```css
/* Hero Section */
.hero {
  min-height: 100vh;
  display: flex;
  background-image: url('internlinkt.jpg');
  align-items: center;
  justify-content: center;
  position: relative;
  padding: 8rem 2rem 4rem;
  overflow: hidden;
  background-repeat: no-repeat;
  background-size: cover;
  background-position: center;
}

/* Dark overlay */
.hero::before {
  content: "";
  position: absolute;
  top: 0; left: 0;
  width: 100%; height: 100%;
  background: rgba(0, 0, 0, 0.5); /* semi-transparent black */
  z-index: 1;
}

/* Make sure hero content is above the overlay */
.hero-content {
  position: relative;
  z-index: 2;
  color: rgb(255, 255, 255); /* Make text white for contrast */
}
```

The **hero section** centers content and overlays background image using @keyframes float animates orbs for a soft movement effect.

background-repeat: no repeat; Stops the background from repeating (no tiling).
background-dimension: fill; Ensures the image fills the whole section.
background-position: middle; Guarantees the background image remains centered within the section.
Overlay adds a dark translucent layer over the background image to enhance text visibility.

9. **Animations**

```css
.animate-on-scroll {
  opacity: 0;
  transform: translateY(30px);
  transition: all 0.8s cubic-bezier(0.25, 0.46, 0.45, 0.94);
}

.animate-on-scroll.animate-in {
  opacity: 1;
  transform: translateY(0);
}
```

Reusable animations like .animate-on-scroll: fade-in + slide-up on scroll,

.loader: rotating multi-color ring for loading state,

Buttons and cards: scaling and box shadows

These enhance user experience without JavaScript.

10. **Responsive Design**

```css
@media (max-width: 768px) {
  .about-content {
    grid-template-columns: 1fr;
    gap: 2rem;
  }
```

Media queries ensure:

- Sidebar turns horizontal or collapses
- Layouts become single-column on mobile
- Font sizes and paddings adjust accordingly

JavaScript

App Javascript

This JavaScript file implements the **front-end logic** for managing user sessions, handling form-based authentication (login and signup), navigating between dashboards for students and companies, and dynamically updating the user interface with internship and profile data. It also provides modular support for interacting with backend APIs, managing modals, and rendering dashboard elements.

1. Global Variables and DOM References

```
// Global variables
let currentUser = null;
let currentRole = null;
```

These two variables are used to store information about the currently logged-in user and their role—either a 'student' or 'company'. By storing this data globally, the application can easily access it throughout different parts of the code (e.g., to load personalized dashboards or handle form submissions).

```
// DOM Elements
const loginModal = document.getElementById('loginModal');
const signupModal = document.getElementById('signupModal');
const toast = document.getElementById('toast');
```

These are references to key HTML elements:

- loginModal and signupModal control the visibility of login and signup popups.
- toast is a UI element used for displaying temporary notification messages (e.g., "Login successful" or "Network error").

  2. App Initialization

  ```
  // Initialize app
  document.addEventListener('DOMContentLoaded', function() {
      checkAuthStatus();
      setupEventListeners();
  });
  ```

When the page finishes loading (DOMContentLoaded), the application:

- Checks if a user is already logged **in** using localStorage by calling checkAuthStatus().
- Sets up listeners for form submissions and modal interactions via setupEventListeners().

This ensures that the page is immediately ready to respond to user actions without requiring a manual refresh. Mozilla. (n.d.).

3. Authentication Status Check

This function verifies whether a user session exists by retrieving currentUser and userRole from localStorage. If found, it tries to parse and validate the data to confirm it's still usable. If the data is invalid (e.g., corrupted or missing required fields), it resets everything.

This function does not auto-redirect the user, giving the developer more control over where and when redirection should happen (for example, only after manual login).

4. Event Listeners for Forms and Modals

```
// Setup event listeners
function setupEventListeners() {
    // Login form
    document.getElementById('loginForm').addEventListener('submit', handleLogin);

    // Signup forms
    document.getElementById('studentSignupForm').addEventListener('submit', handleStude
    document.getElementById('companySignupForm').addEventListener('submit', handleCompa

    // Modal close events
    window.addEventListener('click', function(event) {
        if (event.target === loginModal) closeModal('loginModal');
        if (event.target === signupModal) closeModal('signupModal');
    });
}
```

This function attaches submit handlers to:

- Login form
- Student signup form

- Company signup form

It also handles click events outside of the modal, so the modals close if the user clicks the background (improving UX).

5. Modal Management

Functions like showLoginModal(), showSignupModal(), closeModal(modalId), and switchSignupTab(role) control the visibility and content of modals.

For example:

- showLoginModal() makes the login modal visible and disables background scrolling.
- switchSignupTab('student') changes the visible form to the student signup version.

These functions make the UI feel dynamic and interactive, giving users smooth transitions between different account types and actions.

6. Login & Signup Handling

There are three main form handlers:

handleLogin(event)-

- Prevents form's default submit behavior.
- Reads form data (email, password, role) and sends a POST request to either /api/students/login or /api/companies/login.
- If successful, stores the user info in localStorage, shows a success toast, and redirects to the correct dashboard.

handleStudentSignup(event) and handleCompanySignup(event

- Collect registration details and send them to respective signup APIs.
- If account creation is successful, it prompts the user to log in.

These functions abstract the API interaction and give a consistent user experience for both account types.

7. Dashboard Redirection and Logout

```
// Navigation
function redirectToDashboard() {
    if (currentRole === 'student') {
        window.location.href = '/student_dashboard.html';
    } else {
        window.location.href = '/company_dashboard.html';
    }
}

function logout() {
    localStorage.removeItem('currentUser');
    localStorage.removeItem('userRole');
    currentUser = null;
    currentRole = null;
    window.location.href = '/';
}
```

- redirectToDashboard() sends students to /student_dashboard.html and companies to /company_dashboard.html.
- logout() clears the saved session data and returns the user to the homepage.

This role-based redirection ensures that users see **relevant features and content** based on their role.

8. Toast Notifications

```
function showToast(message, type = 'success') {
    toast.textContent = message;
    toast.className = `toast ${type}`;
    toast.classList.add('show');

    setTimeout(() => {
        toast.classList.remove('show');
    }, 3000);
```

Creates floating notification boxes that appear temporarily on the screen. The type (success/error) changes the color/styling. For example:

- Green for success messages like “Account created!”
- Red for error messages like “Invalid credentials.”

This feedback helps users understand what’s happening without reloading the page.

9. Reusable API Helper

```
async function apiCall(endpoint, options = {}) {
    try {
        const response = await fetch(endpoint, {
            headers: {
                'Content-Type': 'application/json',
                ...options.headers
            },
            ...options
        });
```

A universal function that sends a fetch() request to any API endpoint with automatic JSON parsing and error handling. It improves code reusability and maintainability, as other functions don't need to repeat the same fetch boilerplate.

10. Dashboard Data Loading

Functions like:

loadStudentDashboard()

loadCompanyDashboard()

These use the apiCall() helper to:

Load and display internships, applications, and profile info.

Handle cases where the user isn't authenticated by redirecting to the homepage.

They ensure that each user sees their own data, personalized to their identity and role.

11. Rendering Functions

These functions use template literals and .innerHTML to inject HTML into the DOM:

renderInternships(internships) creates internship cards.

renderApplications(applications) shows applied positions.

renderStudentProfile(profile) and renderCompanyProfile(profile) populate user details.

Each rendering function creates a visually informative layout for the user, often including buttons like "Apply Now" or "Edit Profile."

12. Utility Function

```
// Utility Functions
function getInitials(name) {
    return name.split(' ').map(n => n[0]).join('').toUpperCase().slice(0, 2);
}
```

This small utility extracts and returns the initials from a user's full name (e.g., “John Doe” → “JD”). It’s used for avatar display and user identification in the UI.

13. Action-Based Functions

These manage specific features:

- applyToInternship(id) opens the application modal with the selected internship.
- viewInternshipDetails(id) (stubbed) would show full internship info.
- createInternship() opens a modal to create a new internship.

These functions make the **internship platform interactive and functional**, allowing companies and students to manage their experience.

14. Application Viewing and Acceptance

```
async function viewApplications(internshipId) {
```

```
async function handleApplicationSubmit(event) {
```

These functions allow companies to:

- See which students applied to an internship.
- Accept a student’s application.

The UI dynamically updates based on the response and shows success/error messages. It also calls viewApplications() again to refresh the list after accepting someone.

15. Global Function Export

```
window.InternLinkt = {
```

This exposes selected functions globally so that inline HTML (like onclick="InternLinkt.logout()") or other external JS files can call them. This technique keeps your code **organized in one file** but still accessible as needed.

Company Dashboard JavaScript

1. Section Navigation and UI Transition

```
function showSection(section) {
```

This function allows the user to switch between dashboard views: internships, applications, and profile. It hides all sections, shows the selected one, highlights the current navigation link, and loads relevant data dynamically.

A smooth fade-out and fade-in effect is applied when loading finishes and the dashboard becomes visible, enhancing user experience without page reloads.

2. Dashboard Statistics

```
function updateDashboardStats(internships, applications) {
  document.getElementById('total-internships').textContent = internships.length;
  document.getElementById('total-applications').textContent = applications.length;
  document.getElementById('pending-applications').textContent = applications.filter(app
  document.getElementById('shortlisted-applications').textContent = applications.filter(
}
```

This updates the key stats on the dashboard: total internships, total applications, and counts of applications in “Pending” or “Shortlisted” status. It extracts values from fetched data and updates the respective HTML elements.

3. Recent Applications Panel

```
function renderRecentApplications(applications) {
  const container = document.getElementById('recent-applications');
  if (!container) return;
  if (applications.length === 0) {
    container.innerHTML = '<p class="empty-state">No applications yet. Create your first
    return;
  }
```

Displays the 5 most recent student applications in card format. Each card includes the student's name, university, application status, and the date applied. A “Review” button is added for future actions.

4. Internship Management

a. Loading & Displaying

These functions fetch and visually display all internships created by the company. Each internship is shown in a card with meta info (location, salary, deadline) and includes buttons for viewing applications, editing, or deleting.

b. Editing Existing Internship

When a company clicks “Edit,” the form fields are populated with existing internship details so the employer can make changes easily.

c. Creating/Updating via Form

Submits either a new or edited internship. If internshipId is present, it sends a PUT request (update); otherwise, it posts a new internship to the server.

d. Deleting Internship

Asks for confirmation and deletes an internship. If successful, it reloads the dashboard and internships list.

5**. Application Filtering and Management**

a. Filtering Applications

This enables HR staff to filter student applications by internship or application status using dropdowns. The filtered list is re-rendered in real time.

b. Loading All Applications

Gathers applications from *all* internships posted by the company. It maps each application to its corresponding internship title and populates filter dropdowns.

c. Updating Application Status

Updates the status of an application (e.g., to Shortlisted or Accepted). It sends a PUT request to the API and reloads the application list on success.

6. Company Profile Rendering and Editing

```
function renderCompanyProfile(profile) {
  const container = document.getElementById('profile-container');
  if (!container) return;
  container.innerHTML = `
```

These functions render the company's profile (name, email, description, etc.), allow the company to open an edit form, and handle form submissions to update the profile on the server.

7. **Review Stub**

```
function reviewApplication(applicationId) {
  // Implementation for reviewing application details
  showToast('Application review feature coming soon!', 'success');
}
```

This placeholder function is meant to handle detailed application review logic in the future. Currently, it just shows a toast as a temporary notification

Index Page JavaScript

1. **Hover Lift Effect on Cards**

```
  document.querySelectorAll('.feature-card, .step-card').forEach(card => {
    card.addEventListener('mouseenter', function() {
      this.style.transform = 'translateY(-8px) scale(1.02)';
    });
    card.addEventListener('mouseleave', function() {
      this.style.transform = 'translateY(0) scale(1)';
    });
  });
});
```

This applies a subtle 3D "lift" animation when a user hovers over .feature-card or .step-card elements. It translates the card upwards and slightly scales it for a modern, tactile feel. When the mouse leaves, the element returns smoothly to its original state.

Use case: Highlights interactive elements to draw attention and indicate they're clickable or important.

2. **Animated Statistics with Count-Up Effect**

```
function animateStats() {
```

```
 function animateNumber(element, start, end, duration) {
```

These functions animate numerical values (e.g., number of users, downloads) as the user scrolls. The number increases from 0 to a target value over a smooth 2-second interval.

- Uses the **Intersection Observer API** to trigger only when the element becomes visible.
- animateNumber() calculates small increments per frame and uses requestAnimationFrame for performance-friendly updates
-

Use case: Makes static stats feel dynamic and trustworthy by drawing user focus through movement.

3. **Hero Section Parallax Scrolling**

```
function setupParallax() {
  window.addEventListener('scroll', () => {
    const scrolled = window.pageYOffset;
    const parallax = document.querySelector('.hero-background');
    if (parallax) {
      const speed = scrolled * 0.5;
      parallax.style.transform = `translateY(${speed}px)`;
    }
  });
}
```

Creates a **parallax effect** on the background of the hero section, meaning the background image scrolls more slowly than the rest of the page content. This is done by adjusting the vertical transform based on scroll offset.

Effect: Adds a sense of depth and movement to the page's first impression.

4. **Scroll-Triggered Element Reveal**

```
function setupScrollAnimations() {
  const observer = new IntersectionObserver((entries) => {
    entries.forEach(entry => {
      if (entry.isIntersecting) {
        entry.target.classList.add('animate-in');
      }
    });
```

Reveals page elements with animation as they enter the viewport. Uses another **Intersection Observer** to detect when .animate-on-scroll elements are 10% visible, then adds the animate-in class.

This class is typically paired with CSS animations like fade-ins or slide-ins.

Advantage: Delays animations until needed, improving performance and giving a polished, interactive scroll experience

Student Dashboard JavaScript

This script powers the **student dashboard**, enabling students to view internships, track applications, and manage profiles dynamically. Here's a breakdown of what's unique in this script

Load Guarding & Console Debuggin

```
if (dashboardDataLoading) {
  console.log('Dashboard data already loading, skipping duplicate call');
  return;
}
dashboardDataLoading = true;

try {
  console.log('Loading dashboard data...');
  await loadDashboardData();
  console.log('Dashboard data loaded');

  updateUserInfo();
  setupEventListeners();

  const dashboard = document.getElementById('dashboard-content');
  if (dashboard) {
    dashboard.style.display = 'flex'; // or block if you prefer
    console.log('Dashboard shown');
  }
```

- dashboardDataLoading prevents duplicate API calls by tracking load state.
- Multiple console.log() statements help developers trace execution flow and data issues during debugging.

Parallel API Requests

```
try {
  console.log('Loading dashboard data...');
  await loadDashboardData();
  console.log('Dashboard data loaded');
```

Uses Promise.all() in loadDashboardData() to fetch internships, applications, and student profile **simultaneously**—improving performance by avoiding sequential waits.

User Info Rendering

```
function updateUserInfo() {
  if (!currentUser) return;
  const userNameEl = document.getElementById('user-name');
  const sidebarNameEl = document.getElementById('sidebar-name');
  const userAvatarEl = document.getElementById('user-avatar');
  if (userNameEl) userNameEl.textContent = currentUser.name;
  if (sidebarNameEl) sidebarNameEl.textContent = currentUser.name;
  if (userAvatarEl) userAvatarEl.textContent = getInitials(currentUser.name)
  console.log('User info updated:', currentUser.name);
}
```

updateUserInfo() updates the name and initials in the sidebar and header using getInitials() for visual personalization.

Dashboard Section Switching

```
// Section Navigation
function showSection(section, event) {
  console.log('Switching to section:', section);
```

showSection() switches between **internships**, **applications**, and **profile** views.
Only relevant content is displayed, improving UX and keeping the dashboard clean

Internship and Application Rendering

```
function renderInternships(internships) {
  const container = document.getElementById('internships-container');
  if (!container) return;
  if (internships.length === 0) {
    container.innerHTML = '<p class="empty-state">No internships available at the
    return;
  }
```

- renderInternships() and renderApplications() dynamically create HTML cards based on fetched data.
- Internship cards include "Apply" and "View Details" buttons.
- Application cards show company, status, and date.

Student Profile Display

- ```
  function renderStudentProfile(profile) {
    const container = document.getElementById('profile-container');
  ```
  renderStudentProfile() shows user bio, university, major, and graduation year in a styled layout.
- Includes an "Edit Profile" button for easy updates.

Application Submission

```
async function handleApplicationSubmit(event) {
  event.preventDefault();
  console.log('Submitting application...');
  const formData = new FormData(event.target);
  formData.append('student_id', currentUser.id);
```

handleApplicationSubmit() sends a POST request using FormData, appending the student ID and form inputs.
On success, a toast message appears and the dashboard reloads after a short delay.

Internship Filtering

```
function filterInternships() {
```

- filterInternships() lets students narrow internships using:
- Text search (by title/company),
- Location, and
- Type (e.g., remote/full-time).
```

- Matching results stay visible, while others are hidden using .style.display.

Node.js

Company Backend Javascript

This file sets up **server-side routes** for **company-related operations** such as sign-up, login, internship posting, and profile management using Express, MySQL, and bcrypt.

1. **Company Signup**

```
// 1. Company Signup POST /api/companies/signup
router.post('/signup', async (req, res) => {
  const { name, email, password } = req.body;

  if (!name || !email || !password) {
    return res.status(400).json({ error: 'All fields are required.' });
  }
```

- Validates input (name, email, password).
- Checks for duplicate email.
- Hashes password with bcrypt.
- Inserts a new record into both the users and companies tables.

2. Company Login

```
// 2. Company Login POST /api/companies/login
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
```

- Finds user by email and role='company'.
- Verifies password using bcrypt.compare.
- Retrieves related company profile info and returns combined data (user + company).

3. **Create Internship**

```
// 3. Create Internship POST /api/companies/:companyId/internships
router.post('/:companyId/internships', async (req, res) => {
  const { title, location, type, salary, duration, deadline, skills, descri
  const companyId = req.params.companyId;
```

- **Purpose:** Allow a company to post a new internship.
- **Validates** required fields like title, location, type, and skills.
- **Inserts** internship data into the internships table.

4. **Get All Company Internships**

```
router.get('/:companyId/internships', async (req, res) => {
```

Fetches internships created by a specific company, ordered by creation time.

5. **Get Applications for an Internship**

```
router.get('/internship/:internshipId/applications', async (req, res) => {
```

- Uses an SQL **JOIN** to combine application data with student details.
- Returns a rich dataset for reviewing applicants.

6. **Update Application Status**

```
// 8. Update Application Status PUT /api/applications/:id
router.put('/application/:applicationId/status', async (req, res) => {
  const { status } = req.body;
```

- Updates the status of a student's application (Pending, Shortlisted, etc.).
- Includes validation to prevent invalid status updates.

7. **Update Company Profile**

```
// 9. Update Company Account PUT /api/companies/:id
router.put('/:id', async (req, res) => {
  const { name, email, description, logo, website, location } = req.body;
```

Updates both:

- **User email** in the users table.
- **Company profile** in the companies table (name, logo, website, etc.).

8. **Get Company Info**

```
// 10. Get Company Info GET /api/companies/:id
router.get('/:id', async (req, res) => {
```

- Joins companies and users tables to fetch full profile (including email).
- Used for profile views and form pre-filled

Internship Routes Javascript

1. **Resume Uploads via Multer**

```
// Set up multer for resume uploads
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, path.join(__dirname, '../public/resumes'));
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, uniqueSuffix + '-' + file.originalname);
  }
});
```

multer is used to handle **multipart/form-data**, especially for file uploads (e.g., resumes).

2. **Create Internship**

```
// Create Internship
router.post('/', async (req, res) => {
  const { company_id, title, location, type, skills, salary, duration, deadline, des
  if (!company_id || !title || !location || !type) {
    return res.status(400).json({ error: 'Missing required fields.' });
```

- Validates required fields.
- Inserts a new internship entry into the database.

3. **Advanced Internship Filtering**

```
// Get All Internships (for browsing)
router.get('/', async (req, res) => {
  const { location, type, salaryMin, salaryMax, skills } = req.query;
```

- Dynamically builds an SQL query with optional filters:
- location, type, salaryMin, salaryMax, and skills.
- Uses **parameterized queries** to prevent SQL injection.
- **Flexible design** enables frontend users to search internships by any combination of filters.

4. **View Internship Details**

Retrieves one internship, along with the company's name, logo, and description using a **JOIN** between internships and companies.

5. **Edit Internship**

```
// Edit Internship
router.put('/:id', async (req, res) => {
  const { title, location, type, salary, duration, deadline, skills, description }
```

Updates internship information by ID using values from the request body.

6. **Delete Internship**

```
// Delete Internship
router.delete('/:id', async (req, res) => {
  try {
```

- **Two-step deletion**:
  - Deletes all applications related to the internship (maintains referential integrity).
  - Deletes the internship itself.

7. **Student Application Submission**

```
// Student applies to internship
router.post('/:id/apply', upload.single('resume'), async (req, res) => {
  const internship_id = req.params.id;
  const { student_id, cover_letter } = req.body;
  let resume_path = null;
  if (req.file) {
    resume_path = '/resumes/' + req.file.filename;
  }
  if (!student_id || !cover_letter) {
    return res.status(400).json({ error: 'Missing required fields.' });
  }
  try {
    await db.query(
      'INSERT INTO applications (student_id, internship_id, cover_letter, resume) VA
      [student_id, internship_id, cover_letter, resume_path]
    );
    res.json({ success: true, message: 'Application submitted successfully' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Server error.' });
  }
});

module.exports = router;
```

- Allows a student to apply to a specific internship.
- Accepts form data + uploaded resume.
- Stores metadata (cover letter and resume path) into the applications table.

Student Javascript

Authentication & Security

- **bcrypt** is used to securely hash passwords during signup and to verify credentials during login.
- Password hashes are never stored or returned in plain text.

Resume Upload with Multer

```
// Multer setup for resume upload
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, path.join(__dirname, '../public/resumes'));
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname);
  }
});
const upload = multer({ storage });
```

- Multer handles resume uploads.
- Files are saved in a public/resumes folder with filenames that include a timestamp to ensure uniqueness.

POST /signup – Student Registration

```
// 1. Student Signup POST /api/students/signup
router.post('/signup', async (req, res) => {
  const { name, email, password, university, major, graduation_year } = req.body;
```

- Creates a new user in the users table with the role "student".
- Then inserts profile info into the students table.
- Validates all required fields.

. POST /login – Student Login

```
// 2. Student Login POST /api/students/login
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
```

- Verifies the student's credentials.
- Returns a JSON object with complete student info on successful login (email, university, skills, etc.).

PUT /:id – Update Student Profile

```
// 7. Update Student Profile PUT /api/students/:id
router.put('/:id', async (req, res) => {
  const { name, email, university, major, graduation_year, bio, skills, experience
```

- Updates both user email and student profile fields like bio, skills, experience, phone, and location.
- Checks for student existence first using id.

GET /:id – Fetch Student Profile

```
// 2. Student Login POST /api/students/login
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
```

- Retrieves student profile details along with associated user email.
- Uses a join between students and users.

POST /apply – Apply to Internship

```
// 5. Apply to Internship POST /api/applications
router.post('/apply', upload.single('resume'), async (req, res) => {
  const { student_id, internship_id, cover_letter } = req.body;
  const resume = req.file ? req.file.filename : null;
```

- Requires student_id, internship_id, and optionally a resume and cover_letter.
- Prevents duplicate applications by checking if the student already applied to the same internship.
- Default status for applications is "Pending".

GET /:studentId/applications – Fetch Student's Applications

```
// 6. Get Student Applications GET /api/students/:studentId/applications
router.get('/:studentId/applications', async (req, res) => {
```

- Returns a list of all internships the student has applied to.
- Joins applications, internships, and companies to show relevant details like internship title, company name, and type of work

Database Configuration Javascript

```
const mysql = require('mysql2');

const pool = mysql.createPool({
  host: 'localhost',
  user: 'root',
  password: 'kulsoom',
  database: 'internlinkt',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

module.exports = pool.promise();
```

This script sets up a **MySQL connection pool** using the mysql2 library and exports a **promise-based interface** for executing queries.

Server Javascript

backend > JS server.js > ...

```
const express = require('express');
const cors = require('cors');
const path = require('path');
const studentRoutes = require('./routes/student');
const companyRoutes = require('./routes/company');
const internshipRoutes = require('./routes/internship');

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname, 'public')));

// API Routes
app.use('/api/students', studentRoutes);
app.use('/api/companies', companyRoutes);
app.use('/api/internships', internshipRoutes);

// Serve the main page
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

This script initializes the **Express.js web server** and connects all routes and middleware needed for the InterLinkt web application. Express. (n.d.).

(Casciaro, 2016) explains that Express sets up the web server and cors allows frontend to call backend even if hosted separately. path manages the file paths

studentRoutes, companyRoutes, internshipRoutes: Import route handlers for various API endpoints Expressjs. (n.d.)

- app.use(cors());: Activates CORS, allowing the frontend (even from a different location) to reach the backend.
- app.use(express.json());: parses requests that have JSON formatted data.
- app.use(express.static(path.join(__dirname, 'public')));: Provides static assets such as HTML, CSS, and JS from the public folder.

- The studentRoutes module processes routes that begin with /api/students.
- The companyRoutes module manages routes that begin with /api/companies.
- Requests that begin with /api/internships are processed by the internshipRoutes module

My SQL database

1. Database Creation

```
-- Create the database
CREATE DATABASE IF NOT EXISTS internlinkt;
USE internlinkt;
```

- Creates the database only if it doesn't already exist.
- Switches to use it.

2. Drop Old Tables

```
-- Drop tables if they exist (order matters due to foreign keys)
DROP TABLE IF EXISTS saved_internships;
DROP TABLE IF EXISTS applications;
DROP TABLE IF EXISTS internships;
DROP TABLE IF EXISTS students;
DROP TABLE IF EXISTS companies;
DROP TABLE IF EXISTS users;
```

Ensures a clean reset by deleting tables in reverse dependency order (due to foreign key constraints).

3. Creating of Tables

```
-- Create users table
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('student', 'company') NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- Stores login credentials and the role (student or company).

- All users must have a unique email.

4. students Table

- Linked to the users table using user_id.
- Holds student-specific info like name, major, university, skills, etc.

5. Insert Sample Data

```
-- Insert sample data for testing
-- Insert users first (parent table)
INSERT INTO users (email, password_hash, role) VALUES
('john.smith@example.com', 'hashed_password1', 'student'),
('sarah.johnson@example.com', 'hashed_password2', 'student'),
('company@example.com', 'hashed_password3', 'company')
ON DUPLICATE KEY UPDATE password_hash=VALUES(password_hash), role=VALUES(rol
```

- Adds two students and one company user with hashed passwords.
- Adds three company profiles, all tied to the same user for demo purposes.
- Adds profiles for John Smith and Sarah Johnson linked to their user accounts.

Declaration of Contribution

Members	Member Contribution
Fatima	Javascript and node+ sql, Connect the frontend to backend using SQL database and NodeJS. made certain that the website could securely and efficiently store, retrieve, and manage user and company information.
Mariam	Javascript and node+ sql. Created dynamic webpages integrated with my SQL database. wrote SQL queries and integrated the database with the backend using MySQL2 and connection pooling.

Kulsoom	CSS and Javascript, oversaw the whole display of the website using CSS and javascript animations for every page included animations and icons to improve the overall user experience and responsiveness of the site
Sushma	HTML Files, in charge of Structure of Website Homepage, Signups Form Validation JavaScript, Dashboards and establishing a solid visual base for the platform while maintaining usability and accessibility.

Conclusion

In summary, creating our internship management website was a joint effort that unified various technical abilities—from frontend architecture, design, and animations to backend logic and database administration. Every team member made significant contributions, making sure the platform is both easy to use and robust in functionality. More than a class assignment, we genuinely aspire to expand this website to a broader audience in the future. We aim to establish InternLinkt as a trustworthy and influential platform that effectively aids students in obtaining significant internship experiences and helps businesses discover potential talent. We are confident that this website can significantly simplify, expedite, and enhance accessibility to the internship process for all participants.

References

- Yaish, D.H. (2025, June 9) CSIT128_Sample Node.js Code Using Different Kinds of Files Example [online] Available at: https://moodle.uowplatform.edu.au/course/view.php?id=43186 (accessed: 12 June 2025)
- W3Schools.com (n.d.) *JavaScript JSON*. [online] Available at: https://www.w3schools.com/js/js_json.asp (accessed: 17 June 2025).
- Casciaro, M., 2016. *Node.js Design Patterns*. 2nd ed. Birmingham: Packt Publishing.
- A Complete Guide to CSS Grid Layout (n.d.). *CSS-Tricks*. [online] available at: https://css-tricks.com/snippets/css/complete-guide-grid/ [accessed 19 Jun 2025].

- Mozilla Developer Network (n.d.) *Template literals*. [online]available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals [Accessed 22 Jun 2025].

- *A Guide to Flexbox* (n.d.). *CSS-Tricks*.[online] available at: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ [accessed 18 Jun 2025].

- Mozilla Developer Network (n.d.) *CSS: backdrop-filter*.[online] Available at: https://developer.mozilla.org/en-US/docs/Web/CSS/backdrop-filter [Accessed 22 Jun 2025].

- Mozilla Developer Network (n.d.) *JavaScript reference: Destructuring assignment*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring [Accessed 22 Jun 2025].

- Oracle Corporation (n.d.) *MySQL 8.0 Reference Manual: Optimization*. [online] Available at: https://dev.mysql.com/doc/refman/8.0/en/optimization.html [Accessed 22 Jun 2025].

- dcodeIO (n.d.) *bcrypt.js*. [online]Available at: https://github.com/dcodeIO/bcrypt.js [Accessed 22 Jun 2025].

- W3Schools (n.d.) *SQL JOIN*. [online]Available at: https://www.w3schools.com/sql/sql_join.asp [Accessed 22 Jun 2025].

- MDN Web Docs. (n.d.). *Event.bubbles*. Mozilla. [online]Available at: https://developer.mozilla.org/en-US/docs/Web/API/Event/bubbles [Accessed 20 June 2025].