

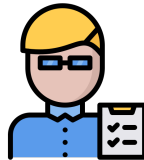
Techunity



Osama Ghaliah - 322509357



Lamar Atamleh - 212760482



Dr. John English

Abstract

In today's digital world, communities thrive where **connection** and **learning** come together. **Techunity** was born from this vision: a platform where **technology enthusiasts, developers, and learners** can not only interact socially but also grow their technical skills in meaningful ways.

Built with **modern web technologies** — **React 18**, **TypeScript**, and **Appwrite's Backend-as-a-Service** — Techunity unites the strengths of a **social media network** with an **educational coding hub**. At its heart lies a **multi-layered architecture**: **React Query** ensures smooth state management on the client side, while **Appwrite's NoSQL database** powers reliable, real-time data on the server side. This design makes every interaction seamless, from posting content to collaborating on coding challenges.

What sets Techunity apart are its **core experiences**:

- **Real-time messaging** that supports both direct and group conversations with administrative oversight.
- An **interactive coding challenge system**, enhanced with **gamification** — achievements, progress tracking, and learning paths that keep users motivated.
- A **content-sharing system** where posts, comments, likes, saves, and personalized profiles create a dynamic social environment.

The **technical foundation** behind these experiences follows **atomic design principles**, combining **Radix UI components**, **Tailwind CSS styling**, and a custom **responsive design system**. Data flows securely and efficiently through **WebSockets**, **optimistic updates**, and **intelligent caching**, while performance is boosted with **code splitting**, **lazy loading**, and **Vite optimization**.

Security is deeply embedded in the platform. **Authentication**, **role-based access control**, and **secure file storage** protect users, while **Zod validation**, **granular permissions**, and error-recovery mechanisms safeguard data integrity and privacy.

Yet, the most distinctive feature of Techunity is its **educational integration**. By weaving **coding challenges** directly into the social experience, it allows users to practice in **multiple programming languages**, receive **instant feedback**, and celebrate progress through **achievements**. Learning becomes not an isolated task but a **shared journey**, where users collaborate, motivate one another, and grow together.

In conclusion, **Techunity** is more than a platform — it is a **community-driven ecosystem** that blends **the engagement of social networking** with the **practical value of coding education**. Designed with scalability, performance, and security at its core, it represents a forward-looking approach to how communities in technology can **connect, learn, and evolve together**.



Introduction

Motivation & Problem Statement

Existing social media platforms don't work well for tech professionals. They're filled with irrelevant content that makes it hard to find useful technical information, share code, or connect with other developers. Tech professionals need a dedicated space where they can focus on what matters to them: coding, learning new technologies, and building meaningful professional relationships.

Objectives

It empowers coders to **connect, collaborate, and grow** by sharing posts, completing challenges, showcasing code, and contributing to developer-centric conversations.

Technology Stack

Frontend:

- **React.js** – Component-based UI development
- **Tailwind CSS** – Utility-first styling for rapid, responsive design
- **React Query** – Data fetching, caching, and synchronization

Backend:

- **Appwrite** – Backend-as-a-service (BaaS) for authentication, databases, file storage, and functions

Other Libraries:

- **TypeScript** – Type-safe development
- **ShadCN UI** – Pre-built accessible UI components
- **React Router** – Page routing and nested layouts

Features

Authentication & Profiles

- Sign-up and sign-in via email/password.
- Editable user profiles with avatars and bios.

Post System

- Create, edit, and delete posts with image uploads.
- Tags, captions, locations.
- Like and save posts.
- Infinite scroll on feed.

Social Graph

- View other users.
- Follow and explore posts by peers.

Storage & Media

- Secure file upload and preview via Appwrite Storage.
- Proper cleanup of unused files on post/user updates.

Advanced Filtering

- Search posts by caption.
- Sort by recency or popularity (likes).

Following System

- Users can easily follow or unfollow other tech professionals with a simple click, building their professional network.
- A dedicated "People" section where users can browse and discover other members of the tech community.
- Follow buttons are seamlessly integrated throughout the platform, appearing on user profiles and user cards for easy access.

Chat System

- A dedicated space for users to interact, discuss, and engage with other tech professionals.

Coding-Challenges System

- Structured coding problems and challenges that help developers sharpen their problem-solving skills.
- Leetcode-like.

Development Process

Critical Step #1 ~ Project Setup & Choosing Tech-Stack

Objective

- Providing us a strong start in developing our idea.

The Process

- Choosing the **backend** tech-stack: **Typescript** and **Appwrite** (BaaS).
- Choosing the **frontend** tech-stack: **Tailwind**, **CSS** and **React**.
- Via **Appwrite**, a comprehensive and accurate database was successfully designed; gathering collections such as:
 - **'followsCollectionId'**: crucial collection for the following system.
 - **'Saves'**: Stores posts that were bookmarked by the users.
 - **'Posts'**: Stores posts that were created by the users successfully.
 - **'Users'**: Stores the platform's all registered users.
 - **'Media'**: Stores created-posts' image/video contents.
- Implementing the code's setup and preparing the API;
E.g.; configs, server, routing, authentication... etc.
- Project's **'src'-focused** schema:

```
src
├── _auth # Authentication-related
│       │ components and layouts
├── assets # Static association routes
│       │ and layouts
├── components # Form-related components
│   ├── forms # Form-related components
│   ├── shared # Shared/common components
│   └── ui # Base UI components (buttons,
│         │ inputs, etc.)
├── constants # Application constants
│       │ and configuration
├── context # React context providers
├── hooks # Custom React hooks
├── lib
│   ├── appwrite # Appwrite backend configuration
│   ├── react-query # React Query configuration
│   └── validation # Form validation utilities
├── types # TypeScript type definitions
├── App.css # Main application styles
├── App.tsx # Root application component
├── globals.css # Global CSS styles
├── index.css # Base CSS styles
├── main.tsx # Application entry point
└── vite-env.d.ts # Vite environment type
    definitions
```

Critical Step #2 ~ Implementing Authentication System

Objective

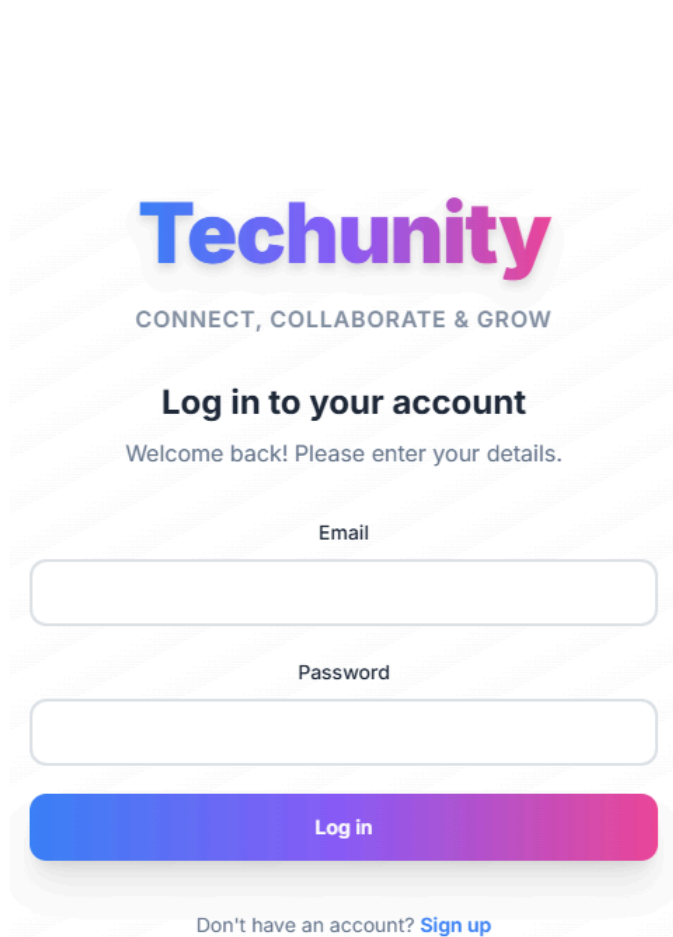
- To establish a secure and user-friendly authentication system that allows users to create accounts, sign in to existing accounts, and securely sign out, providing the foundation for user-specific features and data protection within the **Techunity** platform.

The Process

- The authentication system was implemented through a comprehensive approach involving:
 - **Authentication Context Setup:** Created **AuthContext.tsx** to manage global authentication state, including user data, authentication status, and login/logout functions across the entire application.
 - **Appwrite Integration:** Leveraged **Appwrite** as the backend authentication service, configuring it in **src/lib/appwrite/config.ts** and **api.ts** to handle user registration, login, and session management securely.
 - **Form Components Development:** Built dedicated authentication forms (**SignInForm.tsx** and **SignUpForm.tsx**) with proper validation, error handling, and user feedback mechanisms.
 - **Layout Structure:** Implemented **AuthLayout.tsx** to handle authentication routing logic, ensuring users are redirected appropriately based on their authentication status using **React Router's Navigate** component.
 - **Protected Routes:** Integrated authentication guards to prevent unauthorized access to protected features while maintaining a smooth user experience.

A Challenge That Was Overcome

- One significant challenge was implementing proper authentication state persistence and route protection.
- The team had to carefully balance security requirements with user experience, ensuring that users remained logged in across browser sessions while preventing unauthorized access to protected routes.
- This was resolved by implementing a robust authentication context that properly managed user sessions, handled token refresh logic, and integrated seamlessly with **React Router's** navigation system to provide smooth redirects based on authentication status.



The login form features the Techunity logo at the top, followed by the tagline 'CONNECT, COLLABORATE & GROW'. Below this is the heading 'Log in to your account' and a welcome message. The form includes input fields for 'Email' and 'Password', a 'Log in' button with a blue-to-pink gradient, and a link to 'Sign up' for users without an account.

Techunity
CONNECT, COLLABORATE & GROW

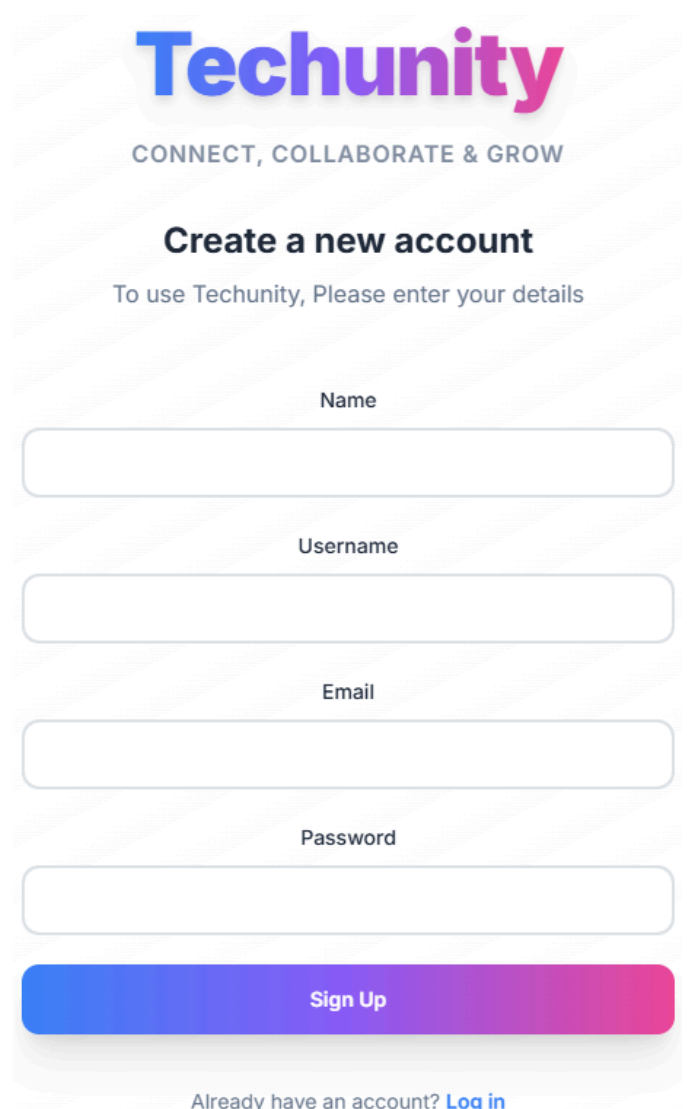
Log in to your account
Welcome back! Please enter your details.

Email

Password

Log in

Don't have an account? [Sign up](#)



The sign-up form features the Techunity logo at the top, followed by the tagline 'CONNECT, COLLABORATE & GROW'. Below this is the heading 'Create a new account' and a prompt to enter details. The form includes input fields for 'Name', 'Username', 'Email', and 'Password', a 'Sign Up' button with a blue-to-pink gradient, and a link to 'Log in' for users who already have an account.

Techunity
CONNECT, COLLABORATE & GROW

Create a new account
To use Techunity, Please enter your details

Name

Username

Email

Password

Sign Up

Already have an account? [Log in](#)

Critical Step #3 ~ Providing Editable Profiles

Objective

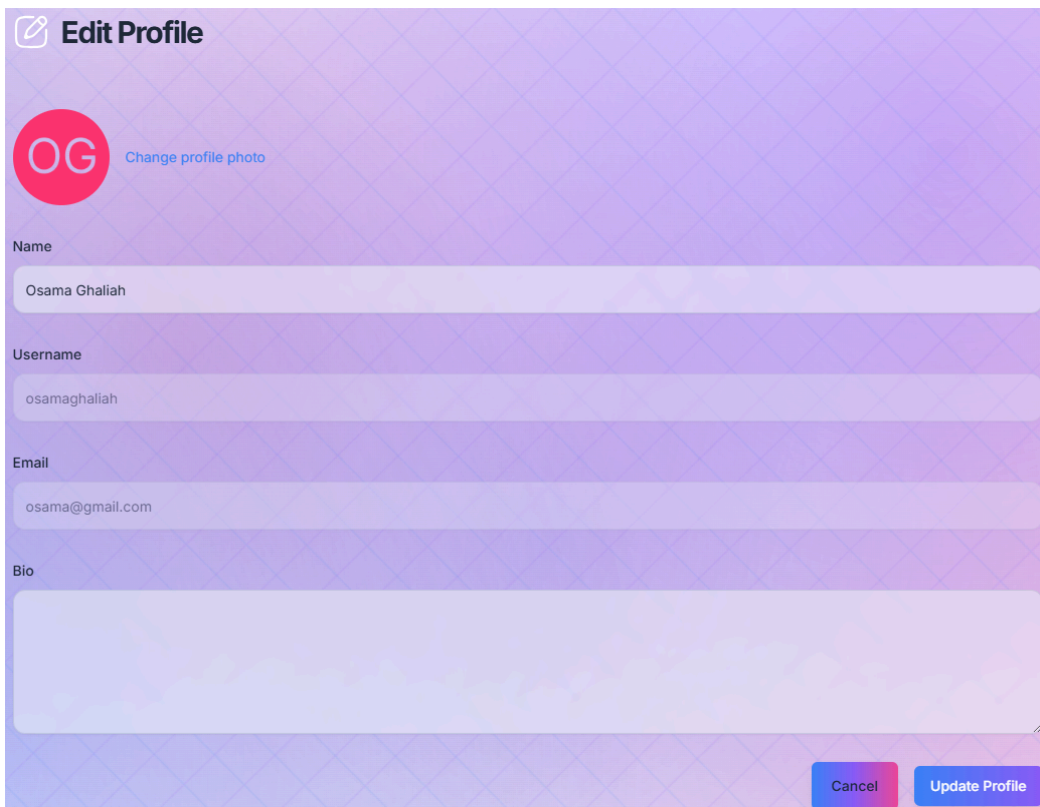
- To create a comprehensive user profile management system that allows users to view, edit, and customize their personal information, profile pictures, and account details, enabling personalization and identity establishment within the **Techunity** community.

The Process


- The editable profile system was implemented through a multi-layered approach:
 - **Profile Display Components:** Developed **Profile.tsx** and **UserCard.tsx** components to showcase user information in an attractive and organized manner, displaying profile pictures, usernames, bio information, and other relevant details.
 - **Profile Editing Interface:** Created **UpdateProfile.tsx** page with comprehensive form controls that allow users to modify various profile attributes including personal information, profile pictures, and account preferences.
 - **Image Upload Functionality:** Implemented **ProfileUploader.tsx** component using the **FileUploader.tsx** utility to handle profile picture uploads, ensuring proper image validation, compression, and storage management.
 - **Form Validation:** Integrated robust validation systems using the validation utilities in **src/lib/validation/** to ensure data integrity and provide real-time feedback to users during profile updates.
 - **State Management:** Leveraged **React Query** for efficient data fetching and caching of profile information, ensuring that profile updates are immediately reflected across the application while maintaining data consistency.


A Challenge That Was Overcome

- A major challenge was implementing seamless profile picture updates while maintaining application performance and user experience.
- The team had to develop an efficient image handling system that could process various image formats, implement proper compression to reduce storage requirements, and ensure that profile picture changes were immediately reflected across all components without causing unnecessary re-renders or performance degradation.
- This was resolved by creating a dedicated **ProfileUploader** component with optimized image processing, implementing proper caching strategies through **React Query**, and ensuring that the file upload system could handle different image formats gracefully while providing immediate visual feedback to users.



The image shows a user interface for editing a profile. It features a purple background with a subtle diamond pattern. At the top left, there is a pencil icon and the text "Edit Profile". Below this is a circular profile picture placeholder with the letters "OG" in white. To the right of the placeholder is a blue link that says "Change profile photo". Below the profile picture are four input fields: "Name" with the value "Osama Ghaliah", "Username" with the value "osamaghaliiah", "Email" with the value "osama@gmail.com", and "Bio" which is an empty text area. At the bottom right, there are two buttons: a pink "Cancel" button and a blue "Update Profile" button.

 **Edit Profile**

 [Change profile photo](#)

Name
Osama Ghaliah

Username
osamaghaliiah

Email
osama@gmail.com

Bio

[Cancel](#) [Update Profile](#)

Critical Step #4 ~ Posting System

Objective

- To establish a comprehensive content creation and sharing platform that enables users to create, publish, and manage various types of posts, fostering content generation and community engagement within the **Techunity** ecosystem.

The Process

- The posting system was implemented through a systematic development approach:
 - **Post Creation Interface:** Developed **CreatePost.tsx** page with an intuitive form interface that allows users to compose posts with text content, media attachments, and relevant metadata, providing a seamless content creation experience.
 - **Form Components:** Built reusable **PostForm.tsx** component that handles post input validation, media uploads, and form submission logic, ensuring consistent post creation across different contexts.
 - **Media Upload Integration:** Integrated **FileUploader.tsx** component to handle various media types (images, videos, documents) with proper validation, compression, and storage management for optimal performance.
 - **Post Management:** Implemented **EditPost.tsx** functionality that allows users to modify existing posts, update content, and manage their published materials with full editing capabilities.
 - **Data Persistence:** Utilized **Appwrite** backend services through the configured API layer to store post data, media files, and user associations, ensuring reliable content storage and retrieval.
 - **Form Validation:** Applied comprehensive validation rules using the validation utilities to ensure post content meets quality standards and platform requirements before publication.

A Challenge That Was Overcome

- A significant challenge was implementing efficient media handling for posts while maintaining fast upload times and optimal storage usage.
- The team had to develop a robust file processing system that could handle multiple file formats, implement intelligent compression algorithms to reduce file sizes without quality loss, and create a seamless user experience where users could upload multiple media files simultaneously.
- This was resolved by creating an optimized **FileUploader** component with progress indicators, implementing proper error handling for failed uploads, and developing a backend storage strategy that efficiently managed media files while maintaining quick access times for content retrieval.

What's on your mind?

Share your thoughts, ideas, or experiences...

0/2200

Add Photos

Upload Photo
Drag & drop or click

PNG JPG SVG

Choose File

Details

Location

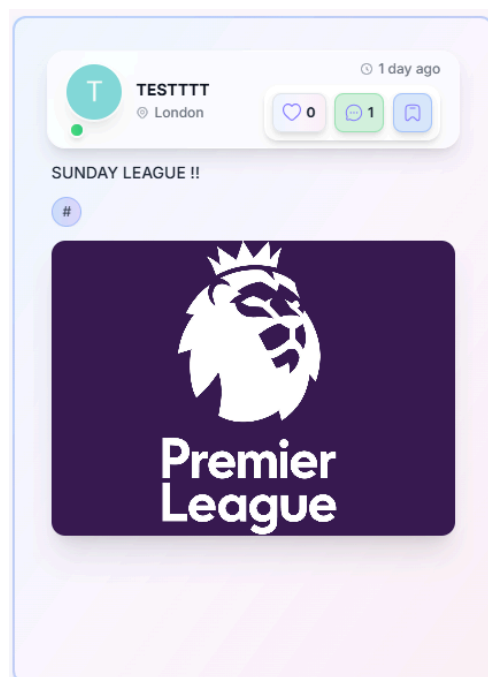
Where are you?

Tags

Art, Expression, Learn

Separate with commas

Create



Critical Step #5 ~ Advanced Filtering

Objective

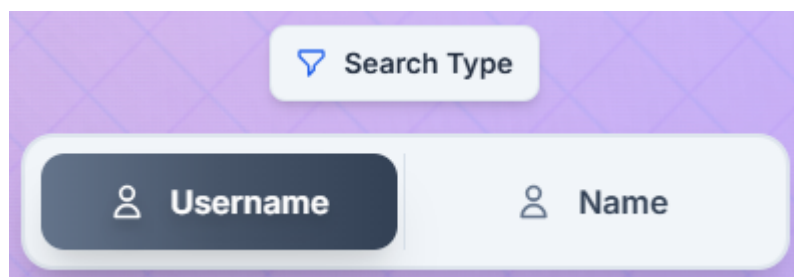
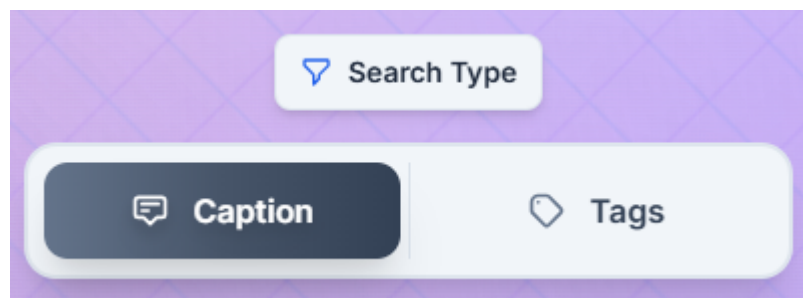
- To implement a sophisticated content discovery system that enables users to efficiently search, filter, and sort through posts based on various criteria, enhancing content accessibility and user experience within the **Techunity** platform.

The Process

- The advanced filtering system was implemented through a comprehensive development approach:
 - **Search Functionality:** Developed robust search capabilities that allow users to find posts by keywords, content, author names, or tags, implementing both real-time search suggestions and comprehensive search results.
 - **Filter Implementation:** Created multiple filtering options including content type filters (text, image, video), date-based filters, author filters, and category-based filtering to help users narrow down content according to their preferences.
 - **Sorting Mechanisms:** Implemented various sorting algorithms including chronological sorting (newest/oldest), popularity-based sorting (likes, comments, shares), and relevance-based sorting to present content in the most useful order for users.
 - **Performance Optimization:** Utilized the **useDebounce.ts** hook to implement debounced search functionality, preventing excessive API calls during user typing and ensuring smooth search performance.
 - **UI Components:** Integrated filter and search controls into the main interface using the **filter.svg** icon and search components, providing intuitive access to filtering options across different pages.
 - **Query Management:** Leveraged **React Query** for efficient data fetching and caching of filtered results, ensuring that search and filter operations are fast and responsive while maintaining data consistency.

A Challenge That Was Overcome

- A major challenge was implementing real-time search functionality without overwhelming the backend with excessive API requests.
- The team had to balance search responsiveness with system performance, ensuring that users could search efficiently while preventing unnecessary server load.
- This was resolved by implementing a debounced search system using the **useDebounce** hook, which delays API calls until users stop typing, significantly reducing server requests while maintaining a responsive user experience. Additionally, the team developed intelligent caching strategies using **React Query** to store search results and filter combinations, minimizing redundant API calls and improving overall search performance.



Critical Step #6 ~ Following System

Objective

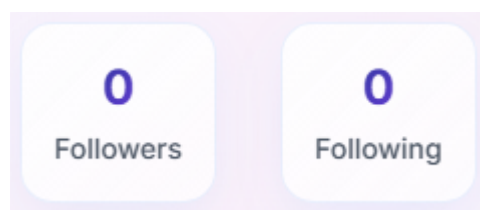
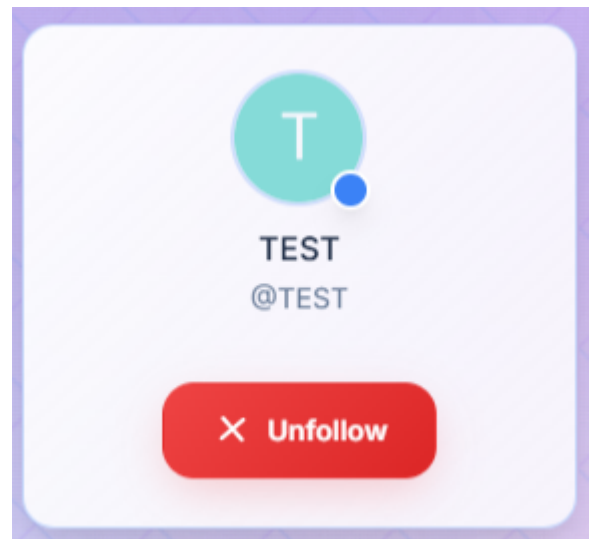
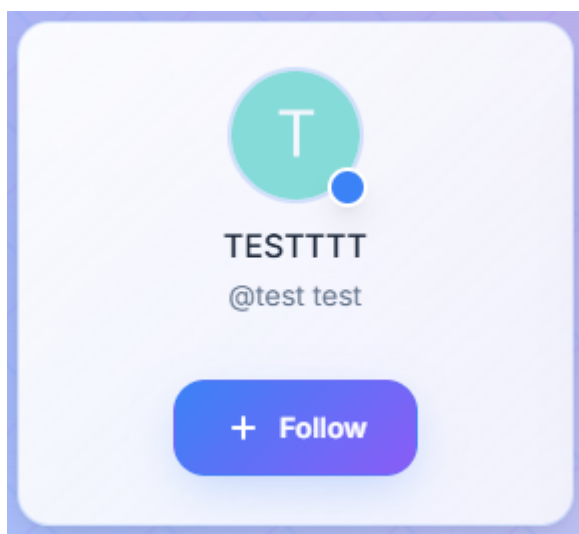
- To implement a social networking feature that enables users to follow other users, create personalized content feeds, and establish meaningful connections within the **Techunity** community, fostering user engagement and content discovery through social relationships.

The Process

- The following system was implemented through a comprehensive social networking approach:
 - **Follow/Unfollow Functionality:** Developed **FollowButton.tsx** component that allows users to follow or unfollow other users with a single click, providing immediate visual feedback and updating relationship status in real-time.
 - **User Discovery:** Created **AllUsers.tsx** page that displays a comprehensive list of users, enabling users to discover and connect with other community members through an intuitive user interface.
 - **User Cards:** Implemented **UserCard.tsx** components that showcase user profiles with follow/unfollow options, displaying essential user information and relationship status to facilitate informed following decisions.
 - **Follow State Management:** Utilized **React Query** to manage follow relationships efficiently, ensuring that follow/unfollow actions are immediately reflected across the application while maintaining data consistency.
 - **Social Integration:** Connected the following system with user profiles and post displays, showing follower counts, following status, and enabling users to see who they're following and who follows them.

A Challenge That Was Overcome

- A significant challenge was implementing real-time follow/unfollow functionality while maintaining optimal performance and preventing race conditions when multiple users interact with the same follow buttons simultaneously.
- The team had to develop a robust state management system that could handle concurrent follow/unfollow requests, ensure data consistency across different components, and provide immediate visual feedback without causing UI flickering or inconsistent states.
- This was resolved by implementing **optimistic updates** in the follow system, where the UI immediately reflects the user's action while the backend processes the request, and then gracefully handling any potential conflicts or failures by reverting to the correct state if needed.



Critical Step #7 ~ Chat System

Objective

- To implement a comprehensive real-time communication system that enables users to engage in both direct messaging and group conversations, fostering deeper social connections and enabling private communication within the **Techunity** platform.

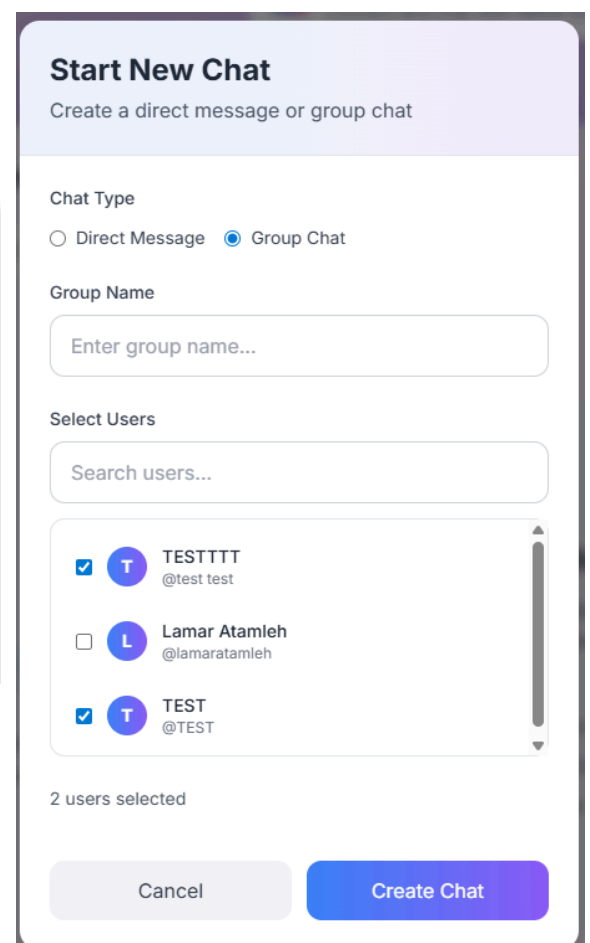
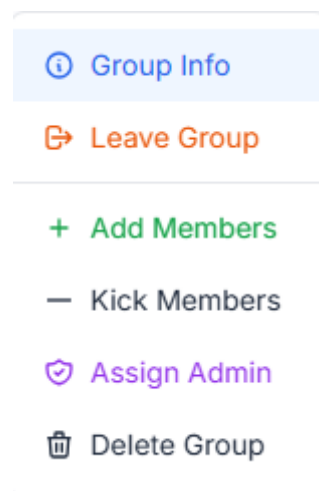
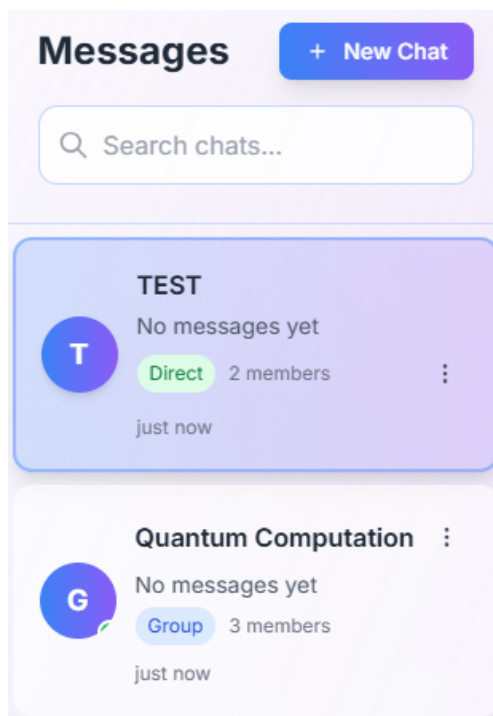
The Process

- The chat system was implemented through a sophisticated real-time communication approach:
 - **Chat Management System:** Developed a complete chat infrastructure using two dedicated **Appwrite** collections — **chats** and **messages** collections — to handle both direct and group conversations with proper data separation and organization.
 - **Chat Creation and Management:** Implemented **createChat()** functionality that supports both direct messaging (one-on-one) and group chats, with features like chat naming, participant management, and group image handling for enhanced user experience.
 - **Message System:** Built a robust system using the **messages** collection that handles different message types (text, image, file) with proper sender identification, timestamping, and chat association.
 - **Advanced Group Features:** Developed comprehensive group chat functionality including member management (**addMemberToGroup**, **removeParticipantFromChat**, **kickMemberFromGroup**), admin controls (**assignAdminToGroup**, **removeAdminFromGroup**), and group customization (**updateGroupDescription**, **updateGroupImage**).
 - **Real-time Message Handling:** Implemented **sendMessage()** and **getChatMessages()** functions with automatic chat updates (last message tracking) to ensure real-time conversation flow.
 - **User Chat Discovery:** Created **getUserChats()** functionality that efficiently retrieves all chats where a user is a participant, enabling users

to access their conversation history and continue ongoing discussions.

A Challenge That Was Overcome

- A major challenge was implementing efficient group chat management while maintaining data consistency and proper authorization controls.
- The team had to develop a complex permission system that distinguishes between direct chats (where only participants can remove themselves) and group chats (where admins can manage members), while also handling edge cases like automatic chat deletion when no participants remain.
- This was resolved by implementing sophisticated authorization checks in functions like **removeParticipantFromChat** and **kickMemberFromGroup**, creating clear role-based permissions, and developing a cascading deletion system that properly cleans up both messages and chat records when conversations are terminated, ensuring data integrity throughout the chat lifecycle.



Critical Step #8 ~ Coding-Challenges System

Objective

- To implement a comprehensive coding education platform that enables users to practice programming skills through interactive challenges, track their learning progress, earn achievements, and compete with others while building a gamified learning experience within the **Techunity** community.

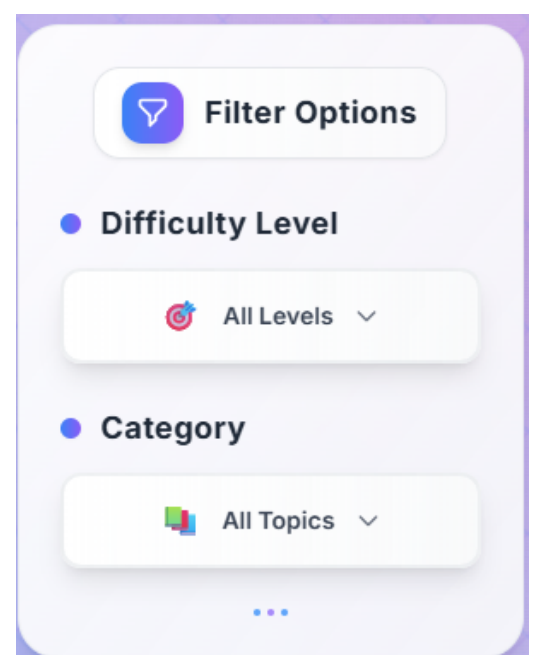
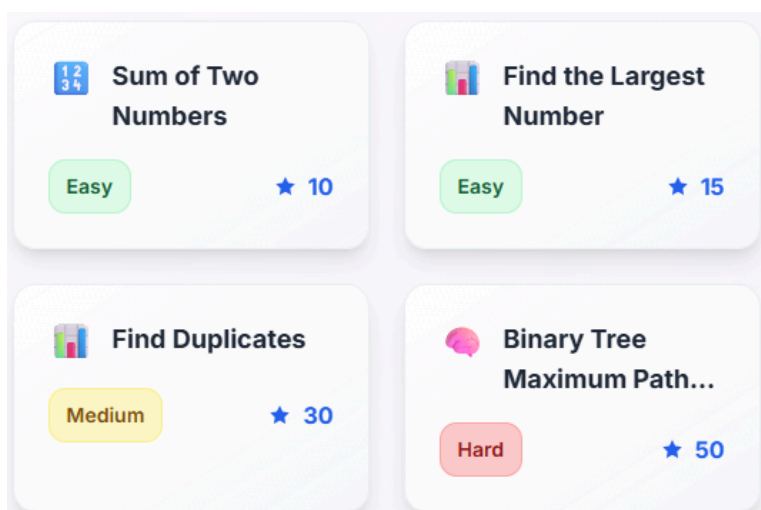
The Process

- The coding challenges system was implemented through a sophisticated educational platform approach:
 - **Challenge Management Infrastructure:** Developed a complete challenge system using dedicated **Appwrite** collections (**challenges**, **challengeAttempts**, **userProgress**, **achievements**) to handle challenge storage, user submissions, progress tracking, and achievement management.
 - **Interactive Challenge Interface:** Created **Coding.tsx** page with a comprehensive tabbed interface featuring challenge browsing, progress tracking, and achievement display, providing users with an intuitive learning environment.
 - **Advanced Challenge Components:** Built specialized components including **ChallengeCard** for displaying challenge details, **ChallengeFilters** for difficulty and category filtering, **ProgressStats** for user analytics, and **AchievementsList** for gamification elements.
 - **Code Validation System:** Implemented **validateCode()** function with test case validation, supporting multiple programming languages (**JavaScript**, **Python**, **Java**, **C++**) and providing real-time feedback on code submissions.
 - **Progress Tracking System:** Developed comprehensive user progress tracking with **getUserProgress()** and **updateUserProgress()** functions that monitor challenges solved, points earned, streaks maintained, and category-specific achievements.

- **Gamification Features:** Created an achievement system with different rarity levels (**common, rare, epic, legendary**) and various achievement types including milestone achievements, streak rewards, category mastery, and special accomplishments.
- **Sample Data Integration:** Implemented fallback sample data system to ensure the platform remains functional even when backend collections are not fully configured, providing immediate access to learning content.

A Challenge That Was Overcome

- A major challenge was implementing a robust code validation system that could safely execute and test user-submitted code while maintaining security and performance.
- The team had to develop a sophisticated validation approach that could handle multiple programming languages, execute test cases safely, and provide meaningful feedback without compromising system security.
- This was resolved by implementing a simulated validation system using pattern matching and code analysis techniques, combined with a comprehensive test case framework that validates code against expected outputs. Additionally, the team created a fallback system that gracefully handles cases where backend collections aren't fully configured, ensuring users can still access and practice with sample challenges while the full system is being deployed.



System Architecture

Application & API-Layer Structure

```
src/lib/appwrite/  
├─ config.ts          # Appwrite configuration  
├─ api.ts             # API functions  
└─ tailwind.config.js # Styling config
```

```
src/  
├─ _auth/              # Authentication layer  
│   ├─ AuthLayout.tsx  # Auth route wrapper  
│   └─ forms/          # Sign-in/Sign-up forms  
├─ _root/              # Main application layer  
│   ├─ RootLayout.tsx  # Main layout wrapper  
│   └─ pages/          # Route components  
├─ components/         # Reusable UI components  
│   ├─ ui/             # Base UI components (Radix-based)  
│   ├─ shared/         # Business logic components  
│   └─ forms/          # Form-specific components  
├─ context/            # React Context providers  
├─ lib/                # Utility libraries  
│   ├─ appwrite/       # Backend integration  
│   ├─ react-query/    # Data fetching layer  
│   └─ validation/     # Form validation schemas  
└─ types/              # TypeScript type definitions
```

Data Flow Pattern

```
User Action → Component → Custom Hook → React Query → API → Appwrite → Database  
↓  
UI Re-render ← Context Update ← State Update ← Cache Update ← Response ←
```

Final Database Schema

Users (users)

└─ accountId, name, username, email, imageUrl, bio

Posts (posts)

└─ creator, caption, imageUrl, imageId, location, tags, likes

Saves (saves)

└─ user, post

Follows (follows)

└─ followerId, followingId

Chats (chats)

└─ name, type, participants, createdBy, admins, lastMessage

Messages (messages)

└─ chatId, senderId, content, type

Comments (comments)

└─ postId, userId, content

Coding Challenges Collections

└─ challenges (definitions)

└─ challengeAttempts (submissions)

└─ userProgress (learning status)

└─ achievements (gamification)

Testing & Futuristic TO-DOs

Methodology

The **testing approach** for **Techunity** employed a comprehensive **manual testing strategy** focused on **user experience validation** and **functional verification** across all platform features. The **testing process** was conducted through systematic **exploration** of the application's **user interface**, ensuring that all implemented **functionalities** operate as intended and provide a seamless **user experience**.

Scope & Coverage

- **Manual UI Testing:** The primary testing method involved navigating through the application's user interface, interacting with various components, and verifying that features respond as expected. This included testing form submissions, button interactions, navigation flows, and visual feedback mechanisms across different sections of the platform.
- **Functional Validation:** Testing focused on verifying that implemented features work according to their intended design. This included validating user authentication flows, content creation and management, social interaction features, and the coding challenges module through direct user interaction and observation of system responses.
- **Feature-By-Feature Testing:** Each major feature was tested individually by interacting with the user interface and observing the system's response. This included testing user registration and login processes, profile management, post creation and editing, social features like following and liking, messaging functionality, and coding challenges.
- **Workflow Testing:** Complete user workflows were tested to ensure logical progression through the application. This included testing the flow from user registration through content creation, social interactions, and engagement with educational features.

Bottom-line

The **manual testing process** successfully **validated** that the **core platform features function** as intended and provide users with a **functional** and **engaging experience**. The **testing** confirmed the **effectiveness** of the **user interface design** and the successful **implementation** of major platform **functionalities**.

While the **testing approach** was primarily **manual** and **UI-focused**, it provided effective **validation** of the platform's **core functionality** and **user experience**, ensuring that the **application** meets its intended purpose as a **social media** and **learning platform** for the **technology community**.

Future Work

- Allowing the users to initialize a chat under a specific opened coding-challenge via effectively adapting both Coding and Chat systems. This feature provides users to solve challenges together.
- Implementing duel-like feature so users can challenge each other for a specific coding-challenge; providing users with more motives to enhance their abilities. Ensuring it is integrated with the achievements functionalities.
- Adding Hi-Tech industry newsfeed & articles.
- Allowing users to attach/upload files, documents, images... etc.

Q & A

What did you learn from doing the project?

The Techunity project was a valuable learning experience that demonstrated the power of modern web technologies and BaaS platforms like Appwrite for rapid development.

While the manual testing approach and UI-focused development process successfully delivered a functional platform, the project highlighted several areas for improvement including the need for comprehensive automated testing, better state management architecture, and more robust error handling strategies.

The experience reinforced the importance of balancing feature development with technical debt management and user experience considerations throughout the development lifecycle.

Did the project come out the way you imagined it ?

Unfortunately, no.

The imagination was production-ready-like. However, the summer semester is short. On the other hand, we accomplished significant progress for the project; well-designed architecture and fully functional, using various beneficial technologies.