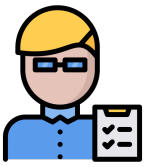
Techunity

Osama Ghaliah - 322509357

Lamar Atamleh - 212760482



Dr. John English

Abstract

To be continued - at the end.

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.

Programmers Interaction

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

Enhancing Coding Abilities

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

- Leetcode-like.

To be continued - at the end.

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 assication 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 Overcame

- 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.

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 Overcame

- 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.

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 Overcame

- 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.

Critical Step #5 ~ Advanced Filtering (Searching & Sorting)

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 Overcame

- 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 ~ Advanced Filtering (Searching & Sorting)

To be continued - at the end.

Future Work & Discussion

To be continued - at the end.