Full-Stack Blogging Platform Assessment

Project Overview

You will be building a "Multi-User Blogging Platform" that allows users to create, edit, and delete blog posts with category management. This application will demonstrate your understanding of full-stack development using Next.js 15, PostgreSQL, Drizzle ORM, tRPC, and other modern web technologies.

Timeline: 7 days from assignment receipt
Expected Time Investment: 12-16 hours

Technical Requirements

Backend Development

1. Database Design and Implementation

- Set up a PostgreSQL database
- Implement database schema using Drizzle ORM
- Create necessary tables for:
    - Blog posts (title, content, slug, published status, timestamps)
    - Categories (name, description, slug)
    - Post-category relationships (many-to-many)

2. API Development (tRPC with Next.js App Router)

- Implement type-safe APIs using tRPC for:
    - CRUD operations for blog posts
    - CRUD operations for categories
    - Assigning categories to posts
    - Filtering posts by category
- Implement proper error handling and validation using Zod schemas
- Use tRPC middleware for request validation
- Implement slug generation for posts and categories
- Leverage tRPC's automatic type inference for end-to-end type safety

Frontend Development

1. User Interface

- Create a responsive blog layout with navigation
- Implement a content editor for post creation/editing (rich text OR markdown)
- Design forms for post and category management
- Create a category management interface
- Build a blog post listing page with filtering
- Design individual blog post view pages

2. State Management and Data Fetching

- Implement global state management using Zustand (where appropriate)
- Use React Query (via tRPC's React Query integration) for API data fetching and caching
- Handle loading and error states appropriately
- Implement optimistic updates for better user experience
- Leverage tRPC's built-in React hooks for seamless data fetching

Feature Priority Guide

To help you manage your time effectively over the 7-day period, features are prioritized as follows:

🔴 Must Have (Core Requirements - Priority 1)

- Blog post CRUD operations (create, read, update, delete)
- Category CRUD operations
- Assign one or more categories to posts
- Blog listing page showing all posts
- Individual post view page
- Category filtering on listing page
- Basic responsive navigation
- Clean, professional UI (doesn't need to be fancy, just functional and clean)

🟡 Should Have (Expected Features - Priority 2)

- Landing page with at least 3 sections (Header/Hero, Features, Footer)
- Dashboard page for managing posts
- Draft vs Published post status
- Loading and error states
- Mobile-responsive design
- Content editor (choose ONE: rich text editor OR markdown support - markdown is faster)

🟢 Nice to Have (Bonus Features - Priority 3)

Only if you have extra time and core features are polished.

- Full 5-section landing page (Header, Hero, Features, CTA, Footer)
- Search functionality for posts
- Post statistics (word count, reading time)
- Dark mode support
- Advanced rich text editor features
- Image upload for posts
- Post preview functionality
- SEO meta tags
- Pagination

Technical Stack Requirements

- **Next.js 15** with App Router
- **PostgreSQL** (local or hosted, e.g., Supabase, Neon)
- **Drizzle ORM**
- **tRPC** (for type-safe API layer)
- **Zod** (for schema validation with tRPC)
- **React Query** (TanStack Query, integrated via tRPC)
- **Zustand** (for global state where needed)
- **TypeScript**
- **Tailwind CSS** (strongly recommended for faster styling)
- **Content editor**: Choose ONE:
    - Markdown editor (faster: textarea + markdown parser)
    - Rich text editor (e.g., Tiptap, Lexical)

Important Notes

- **Authentication system is NOT required** - focus on core blogging features
- **Focus on code quality over feature quantity** - we value well-architected, type-safe code
- **Choose markdown over rich text** if you want to save 2-3 hours
- **Use a component library** (shadcn/ui) if you want to speed up UI development
- **Prioritize properly** - a polished core feature set beats a rushed complete feature set

Evaluation Criteria

We will assess your submission based on:

1. **Code Organization and Architecture** (20%)
    - Clean separation of concerns
    - Proper folder structure
    - Reusable components
    - Well-organized tRPC router structure
2. **UI/UX - Overall Design** (20%)
    - Professional and clean design
    - Intuitive navigation
    - Responsive layout across devices
    - Good user feedback (loading states, error messages)
3. **TypeScript Implementation** (15%)
    - Proper use of TypeScript and type safety
    - Effective use of tRPC's automatic type inference
    - Minimal use of `any` types
    - Well-defined interfaces and types
4. **React Best Practices** (15%)
    - Implementation of modern React patterns and hooks
    - Effective use of tRPC React hooks
    - Component composition
    - Performance considerations
5. **Database Design** (10%)
    - Database schema design and relationships
    - Appropriate use of Drizzle ORM
    - Data integrity
6. **API Design (tRPC)** (10%)
    - Well-structured tRPC routers and procedures
    - Proper input validation with Zod
    - Error handling in tRPC context
    - Logical organization of endpoints
7. **State Management** (5%)
    - Efficient use of Zustand for global state
    - React Query implementation via tRPC
    - Appropriate cache management
8. **Error Handling** (5%)
    - Input validation with Zod schemas
    - User-friendly error messages
    - Graceful error recovery

What We're NOT Looking For

- Pixel-perfect designs (clean and functional is enough)
- Every single bonus feature implemented
- Over-engineered solutions
- Excessive premature optimization

What We ARE Looking For

- Clean, readable, maintainable code
- Proper TypeScript usage with tRPC
- Working core features that are well-implemented
- Good understanding of the tech stack
- Thoughtful architecture decisions

Submission Guidelines

Required:

- GitHub repository with clear README documentation
- Setup instructions with all environment variables documented
- Brief explanation of your tRPC router structure
- Live deployment link (Vercel recommended - it's free and fast)
- Instructions on how to seed the database (if applicable)

README should include:

- Setup steps (how to run locally)
- Tech stack used
- Features implemented (checklist from Priority 1, 2, 3)
- Any trade-offs or decisions you made
- Time spent (optional but helpful)

Time Management Suggestions

Day 1-2: Setup & Backend

- Project initialization and dependencies
- Database setup and Drizzle schema
- Basic tRPC setup with routers
- Core CRUD operations for posts and categories

Day 3-4: Core Features

- Blog listing page
- Individual post view
- Post creation/editing form
- Category management
- Category filtering

Day 5-6: Polish & Priority 2 Features

- Dashboard implementation
- Landing page (3 sections minimum)
- Mobile responsiveness
- Loading and error states
- Bug fixes

Day 7: Final Polish & Deployment

- Code cleanup
- README documentation
- Deployment to Vercel
- Final testing
- (Optional) Add bonus features if time permits

Recommended Shortcuts to Save Time

1. Use **shadcn/ui** for pre-built components (saves 3-4 hours)
2. Choose **markdown** over rich text editor (saves 2-3 hours)
3. Use **Neon or Supabase** for quick PostgreSQL setup (saves 1 hour)
4. Start with a **simple 3-section landing page**, expand if time allows
5. Focus on **desktop-first**, then add mobile responsiveness
6. Use **Tailwind's default theme** instead of custom design system

Questions?

If anything is unclear about the requirements, please document your assumptions in your README. We value your ability to make reasonable decisions when faced with ambiguity.

Remember: We're evaluating your ability to build production-quality code with modern tools. A well-implemented core feature set with clean architecture is much more valuable than a rushed application with every feature.