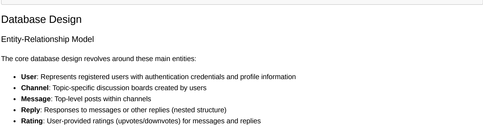
DevBoard - Design Report

Name	NSID	Student Number	Course
Ruel Nathaniel Alarcon	ozg921	11341033	CMPT 353

Architecture Overview

DevBoard is a community platform for discussing programming issues, sharing knowledge, and connecting with developers. The application follows a modern full-stack architecture with clear separation of concerns:



Database Design

Entity-Relationship Model

The core database design revolves around these main entities:

- **User**: Represents registered users with authentication credentials and profile information
- **Channel**: Topic-specific discussion boards created by users
- **Message**: Top-level posts within channels
- **Reply**: Responses to messages or other replies (nested structure)
- **Rating**: User-provided ratings (upvotes/downvotes) for messages and replies

Relationships

- Users create channels (one-to-many)
- Users author messages (one-to-many)
- Users author replies (one-to-many)
- Users provide ratings (one-to-many)
- Channels contain messages (one-to-many)
- Messages have replies (one-to-many)
- Replies can have nested replies (self-referential)
- Messages and replies can receive ratings (polymorphic relationship)

ORM Implementation

The database entities are implemented using Sequelize ORM, which provides:

- Automated table creation and relationship management
- Data validation and type enforcement
- Transaction support
- Migration capabilities

Backend Architecture

Server Framework

The backend uses Express.js, a lightweight and flexible Node.js web application framework, providing:

- HTTP server capabilities
- Middleware support for request processing
- Session management
- Static file serving

API Design

The application uses GraphQL through Apollo Server v4, chosen for:

- Flexible querying capabilities
- Strongly typed schema
- Efficient data fetching (clients can request exactly what they need)
- Built-in documentation and playground

GraphQL Schema

The GraphQL schema defines:

- Data types (User, Channel, Message, Reply, Rating)
- Query operations for data retrieval
- Mutation operations for data modification
- Polymorphic relationships (SearchResult union type)

Authentication

User authentication is implemented with:

- Express-session for session management
- Password hashing using bcryptjs
- Session-based authentication rather than JWT tokens
- Special admin account with elevated privileges

File Storage

The system supports image uploads for:

- User avatars
- Message screenshots
- Reply screenshots

Files are stored in the filesystem with references in the database.

Frontend Architecture

Component Structure

The React application follows a component-based architecture:

- **Context providers**: Auth context for user state management
- **Pages**: Main route components representing full pages
- **Components**: Reusable UI elements
- **Hooks**: Custom React hooks for shared logic

State Management

- Apollo Client manages GraphQL data fetching, caching, and updates
- React Context API manages authentication state
- Component-level state for UI interactions

Routing

React Router DOM v7 provides client-side routing with:

- Route protection based on authentication status
- Nested routes for related views
- Dynamic routing with parameters

UI Framework

Mantine UI was selected as the component framework for:

- Modern, accessible components
- Theming capabilities
- Responsive design support
- Comprehensive UI component library

Performance Considerations

- Code splitting for reduced initial load time
- Optimistic UI updates for improved perceived performance
- React's virtual DOM for efficient rendering
- Custom font (Geist Sans) with variable font technology for responsive typography

Search Functionality

The search system is designed to provide comprehensive content discovery:

- Full-text search across messages and replies
- User-based content filtering
- Advanced sorting options based on ratings
- Channel-specific search capabilities

Design Patterns & Principles

Several key design patterns and principles were applied:

- **Repository Pattern**: Abstraction layer between data models and business logic
- **MVC Architecture**: Separation of concerns between models, views, and controllers
- **Context Provider Pattern**: For state management across the component tree
- **Higher-Order Components**: For sharing common functionality
- **RESTful Resource Naming**: For predictable API endpoints

Scalability Considerations

The architecture was designed with scalability in mind:

- Separation of frontend and backend allows for independent scaling
- GraphQL reduces over-fetching and under-fetching of data
- Database indexing for performance optimization
- Stateless authentication for horizontal scaling potential

Security Measures

- Password hashing for secure credential storage
- Content sanitization to prevent XSS attacks
- Input validation at both client and server levels
- CSRF protection via express-session
- Authorization checks for all operations

Package Dependencies

Server Dependencies

Package	Purpose
@apollo/server	GraphQL server implementation for Node.js
bcryptjs	Secure password hashing for user authentication
cors	Enables Cross-Origin Resource Sharing for API access
dotenv	Environment variable management from .env files
express	Web server framework for handling HTTP requests
express-session	Session management for authentication
graphql	GraphQL implementation for JavaScript
graphql-tag	Template literal tag for parsing GraphQL queries
multer	Middleware for handling file uploads (screenshots)
mysql2	MySQL client for database connectivity
sanitize-html	Prevents XSS attacks by sanitizing HTML content
sequelize	ORM for database interaction and model definition

Testing & Development (Server)

Package	Purpose
cypress	End-to-end testing framework
jest	JavaScript testing framework
nodemon	Auto-restarts server during development
prettier	Code formatting tool
supertest	HTTP assertion library for API testing
start-server-and-test	Coordinates test server startup and test execution

Client Dependencies

Package	Purpose
@apollo/client	GraphQL client with caching capabilities
@mantine/core	UI component library for React
@mantine/form	Form handling for Mantine UI
@mantine/hooks	Custom React hooks for common UI patterns
@mantine/notifications	Toast notification system
@mantine/code-highlight	Code syntax highlighting for programming content
@tabler/icons-react	Icon pack for UI elements
graphql	GraphQL implementation for JavaScript
react	Core UI library for component-based development
react-dom	DOM-specific methods for React
react-router-dom	Client-side routing library

Development & Build (Client)

Package	Purpose
vite	Modern build tool for faster development
typescript	Type safety for JavaScript development
eslint	JavaScript/TypeScript linting
stylelint	CSS linting
prettier	Code formatting
@testing-library/*	Testing utilities for React components
postcss	CSS processing and transformation