DevBoard - Test Report

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

Testing Strategy

The DevBoard application implements a comprehensive testing strategy across multiple layers:

1. **Unit Tests**: Using Jest to test individual components and functions
2. **Integration Tests**: Testing API resolvers with mock data
3. **End-to-End Tests**: Using Cypress to simulate real user interactions

This multi-layered approach ensures high code quality, robust functionality, and a smooth user experience.

Unit & Integration Testing with Jest

Jest is used for testing backend components, particularly the GraphQL resolvers that form the application's API layer.

Test Structure

The Jest tests are organized by resolver type:

```
server/__tests__/
  ├── resolvers/
  │   ├── channel.test.js
  │   ├── message.test.js
  │   ├── rating.test.js
  │   ├── reply.test.js
  │   ├── search.test.js
  │   └── user.test.js
  └── mocks/
      └── various mock implementations
```

Test Coverage

The Jest tests cover all GraphQL resolvers:

Resolver Type	Coverage Areas
User	Authentication, CRUD operations, field resolvers
Channel	Creation, updates, deletion, queries
Message	CRUD operations, nested data
Reply	Nested replies, CRUD operations
Rating	Upvotes, downvotes, rating toggles
Search	Content search, user search, sorting

This amounts to a total of 52 unit tests across the 6 suites.

Testing Approach

Each test uses mock implementations of the database models to isolate the resolver logic:

- **Mock Context**: Simulates the GraphQL context with authentication state
- **Mock Models**: Replaces actual database operations with controlled responses
- **Assertions**: Verifies correct model calls and response formatting

This approach ensures the business logic is tested independently from the database, making tests faster and more reliable.

End-to-End Testing with Cypress

Cypress tests simulate real user interactions with the application, testing the entire stack from UI to database, using a separately created test database.

Test Structure

The Cypress tests are organized into four main categories:

```
server/cypress/e2e/
  ├── 1-navigation/
  │   └── navigation.cy.js
  ├── 2-auth/
  │   └── auth.cy.js
  ├── 3-channels/
  │   └── channels.cy.js
  └── 4-search/
      └── search.cy.js
```

This amounts to a total of 16 integration tests.

Test Scenarios

1. Navigation Tests

The navigation tests verify that users can navigate between different pages:

- **Landing Page Validation**: Checks that the landing page displays correctly with all components
- **Login Navigation**: Verifies navigation to the login page
- **Register Navigation**: Verifies navigation to the registration page

2. Authentication Tests

The authentication tests verify user account management:

- **User Registration**: Tests creating a new account with valid credentials
- **User Login**: Tests logging in with valid credentials
- **Admin Login**: Tests logging in as an administrator
- **Invalid Login**: Verifies proper error handling with incorrect credentials
- **Logout Functionality**: Tests that users can successfully log out

3. Channel & Content Tests

The channel tests verify the core content functionality:

- **Channel Creation**: Tests creating a new discussion channel
- **Channel Navigation**: Verifies navigation to channel details
- **Message Posting**: Tests creating new messages in a channel
- **Rating System**: Verifies upvoting and downvoting functionality
- **Reply System**: Tests creating replies to messages
- **Nested Replies**: Verifies the nested reply functionality

4. Search Tests

The search tests verify the search functionality:

- **Search Interface**: Checks that all search components are present
- **Channel Search**: Tests searching for channels by name
- **User Search**: Tests searching for users by username
- **Result Display**: Verifies proper display of search results

Testing Methodology

Cypress tests follow a user-centric workflow:

1. Setup the test environment (login, navigate to starting point)
2. Perform user actions (click buttons, fill forms, submit data)
3. Verify expected outcomes (UI changes, notifications, data presence)
4. Test error cases and edge conditions

This approach ensures that all user workflows function properly from end to end.

Test Automation

Tests are automated through npm scripts and can be run in various modes:

- **Unit Tests**: `npm run test:units` runs all Jest tests
- **E2E Tests (Headless)**: `npm run test:e2e` runs Cypress tests in CI mode
- **E2E Tests (Browser)**: `npm run cypress` opens Cypress for interactive testing

Test Reporting

Both Jest and Cypress generate comprehensive test reports:

- **Jest**: Console output and optional HTML coverage reports
- **Cypress**: Video recordings of test runs and screenshots of failures
- **Test Logs**: Detailed logs for troubleshooting