Gisma University of Applied Sciences

Final Project Report: ScrollCast

Professor: **Prof. Alireza Mahmoudi**

M607 Computer Science Application Lab

Author

GH1031360 Sharan Thakur
sharan.thakur@gisma-student.com

GitHub https://github.com/c2p-cmd/scroll_cast

YouTube https://youtu.be/UjMPYivrAcI

Date: September 2025

Contents

1. ScrollCast – Project Report

The following report details the development and implementation of ScrollCast, a full-stack web application developed for the M607 Computer Science Application Lab.

2. Introduction

ScrollCast is a modern, full-stack news aggregation and weather platform built using React and Express.js. The application provides users with a comprehensive set of features, including the ability to aggregate news from various RSS feeds, manage personalized news sources, and access real-time weather information. A key feature is the administrative panel, which offers robust content and user management capabilities. The platform was engineered with a focus on performance, scalability, and delivering an optimal user experience.

3. Assignment Overview

This project was designed as a comprehensive full-stack web application to demonstrate a deep understanding of modern software development practices and principles. It successfully addresses several core learning objectives:

- **Full-stack Development**: The application is a complete web solution with a clearly separated frontend and backend.
- **RESTful API Design**: The backend is built with well-structured API endpoints that adhere to REST principles, ensuring a logical and scalable data flow.
- **Database Design & ORM**: A relational database model was designed and implemented using Prisma ORM for type-safe database access and streamlined migrations.
- **User Authentication**: Secure, JWT-based authentication and authorization were implemented to manage user sessions and permissions.
- **Frontend Framework Mastery**: The frontend showcases modern React development practices, utilizing hooks, state management, and a robust routing system.
- **Third-party Integration**: The application seamlessly integrates with external services for RSS feed parsing and real-time weather data.
- **Code Quality**: Best practices for code organization, testing, and comprehensive documentation were applied throughout the development lifecycle.

4. Technical Competencies

The development of ScrollCast demonstrates proficiency in several key technical areas, essential for modern software engineering.

- **Backend API Development**: The backend was crafted using Express.js and Node.js, providing a robust and scalable foundation for the application's data layer.
- **Frontend UI Development**: The user interface was meticulously developed using React and the Material-UI component library, ensuring a responsive, accessible, and visually appealing design.
- **Database Schema Design**: The project involved careful database schema design and management of migrations to support complex data relationships, such as user favorites and feed sources.
- **Authentication & Security**: A secure authentication system was implemented using JWT tokens and bcrypt encryption to protect user data and control access.
- **Real-time Data Processing**: The application's core functionality relies on processing real-time data from various RSS feeds efficiently.

- **Data Visualization**: Interactive charts for the weather dashboard were created using Plotly.js, showcasing skills in data representation and visualization.
- **Code Testing & Documentation**: The project includes backend unit tests for critical functionality and is supported by comprehensive documentation.

4.1. Key Features

4.1.1 User Management

The platform includes a robust user management system designed for secure and efficient user handling.

- **Secure Authentication**: A secure, JWT-based login and registration system ensures stateless authentication.
- **Role-based Access Control**: The application implements distinct roles for standard users and administrators, each with specific permissions and access levels.
- **Password Security**: User passwords are encrypted using the industry-standard bcrypt hashing algorithm for secure storage.

4.1.2 RSS Feed Management

This core feature enables the aggregation and management of diverse news content.

- **Multi-source Aggregation**: Users can consume multiple RSS feed sources from various publishers.
- **Real-time Parsing**: The application parses and displays live RSS feed content, ensuring data freshness.
- **Favorites System**: Users can save and organize their preferred news feeds for easy access.
- **Admin Controls**: Administrators have complete CRUD (Create, Read, Update, Delete) capabilities over feed sources.

4.1.3 Weather Dashboard

A dedicated dashboard provides interactive weather information.

- **Location-based Weather**: The dashboard provides real-time weather information for multiple cities.
- **Weather Metrics**: Key weather data, including temperature, humidity, and general conditions, is displayed clearly.

4.1.4 Analytics and Admin Panel

The administrative features provide essential insights and control over the platform.

- **Usage Statistics**: The admin panel provides analytics to monitor platform usage and user engagement.
- **Feed Performance Tracking**: Administrators can track the performance and popularity of various RSS feeds.
- **Interactive Charts**: Data is visualized using interactive charts powered by Plotly.js, offering a dynamic view of important metrics. See Figure ??
- **Administrative User Management**: The admin panel allows for the control and monitoring of feed sources.

4.2. Architecture

The project's architecture is organized into distinct, modular components to ensure maintainability and scalability. The project directory structure reflects the clear separation of concerns between the frontend and backend services.

```
ScrollCast/
| backend/          # Express.js API Server
    routes/       # Defines API endpoints for data retrieval and manipulation.
    utils/        # Contains utility functions, including database and RSS processing logic.
    tests/        # Houses unit tests for the backend API.
| frontend/         # React Web Application
    src/          # Core React components, pages, and application logic.
    public/       # Static assets like images and HTML templates.
    components/   # Reusable UI components used across the application.
| prisma/           # Database schema and migration files.
 generated/        # Auto-generated Prisma client for type-safe database access.
```

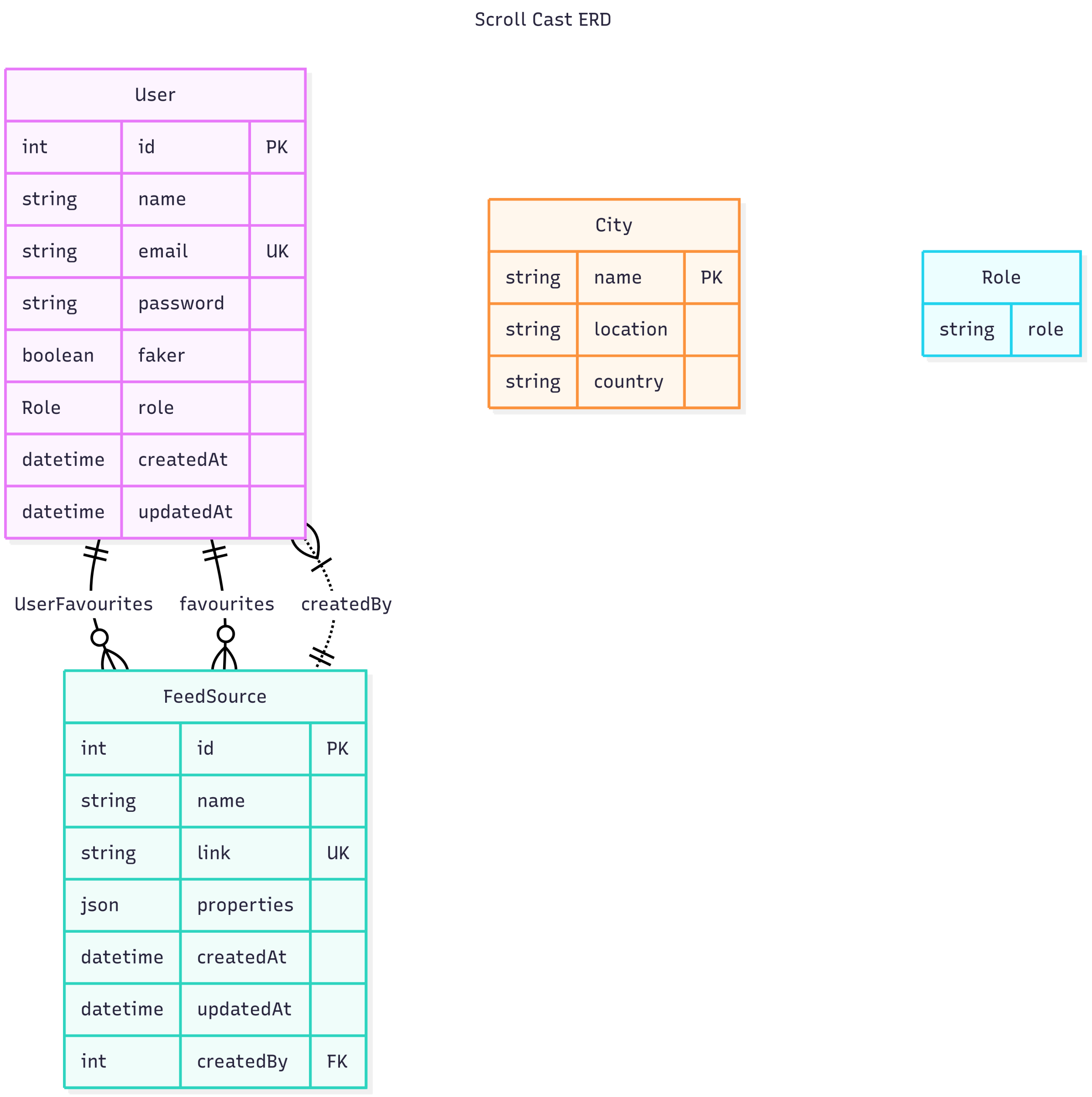
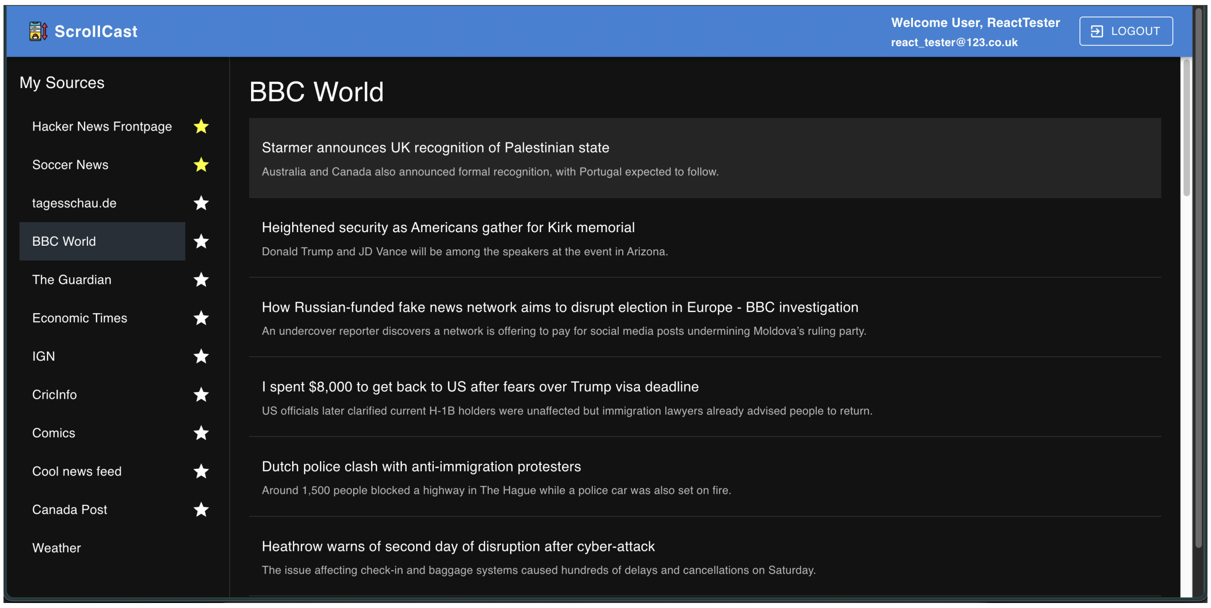


Figure 1: ER Diagram

4.3. Outputs

4.3.1 User View



ScrollCast

Welcome User, ReactTester
react_tester@123.co.uk

LOGOUT

My Sources

- Hacker News Frontpage ★
- Soccer News ★
- tagesschau.de ★
- BBC World ★
- The Guardian ★
- Economic Times ★
- IGN ★
- CricInfo ★
- Comics ★
- Cool news feed ★
- Canada Post ★
- Weather

BBC World

Starmer announces UK recognition of Palestinian state
Australia and Canada also announced formal recognition, with Portugal expected to follow.

Heightened security as Americans gather for Kirk memorial
Donald Trump and JD Vance will be among the speakers at the event in Arizona.

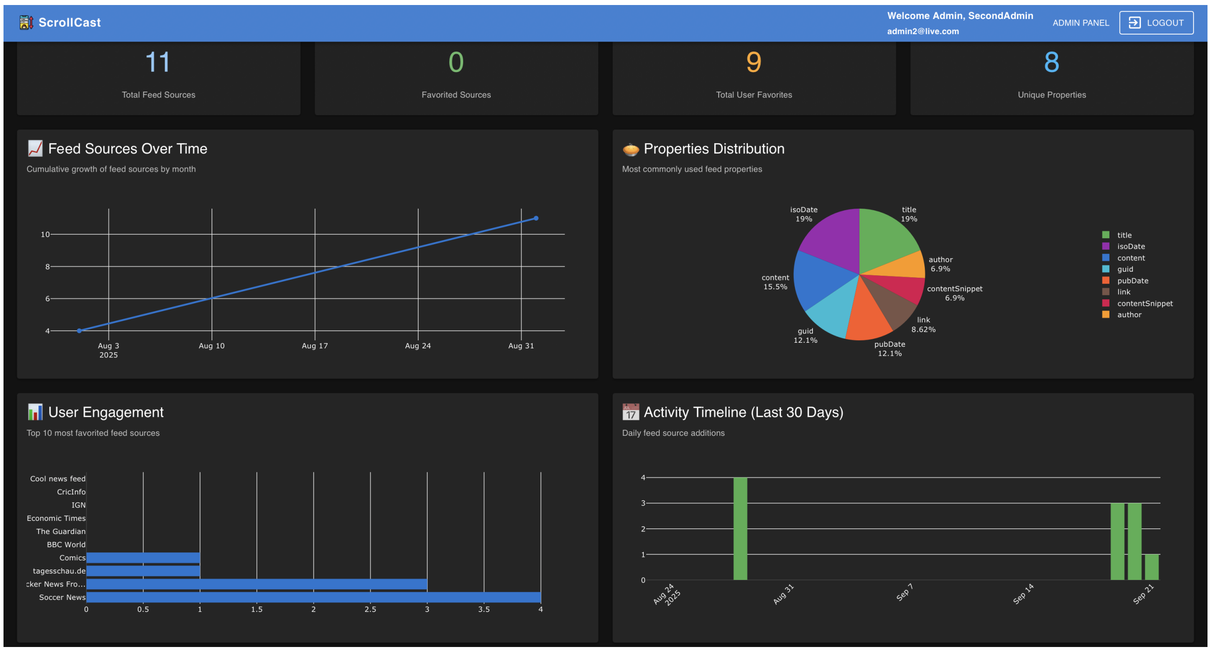
How Russian-funded fake news network aims to disrupt election in Europe - BBC investigation
An undercover reporter discovers a network is offering to pay for social media posts undermining Moldova's ruling party.

I spent $8,000 to get back to US after fears over Trump visa deadline
US officials later clarified current H-1B holders were unaffected but immigration lawyers already advised people to return.

Dutch police clash with anti-immigration protesters
Around 1,500 people blocked a highway in The Hague while a police car was also set on fire.

Heathrow warns of second day of disruption after cyber-attack
The issue affecting check-in and baggage systems caused hundreds of delays and cancellations on Saturday.

4.3.2 Admin View



5. Technical Challenges and Solutions

The development process involved navigating several technical challenges, each with a principled solution.

5.1. Challenge 1: RSS Feed Parsing Reliability

Problem: RSS feeds from different sources often have varying structures and can contain malformed XML or missing data fields. This inconsistency makes it difficult to parse and display content reliably. **Solution**: To address this, a robust error-handling mechanism was implemented using the `rss-parser` library. The solution included validation layers and fallback mechanisms to ensure a consistent data structure, regardless of the quality of the source feed.

5.2. Challenge 2: Role-based Access Control

Problem: The application required different levels of access for standard users and administrators to protect sensitive features and data. **Solution**: A secure, middleware-based authentication system was created using JWT tokens. This system implemented role-based route protection on the backend and conditional rendering on the frontend to ensure that only authorized users could access specific content and functionalities.

5.3. Challenge 3: Weather Data Integration

Problem: External weather APIs can be prone to rate limits and may have inconsistent response formats across providers. **Solution**: The weather dashboard was designed to use a unified service layer that processes RSS feeds from a reliable source like AccuWeather. This approach provided a consistent data format and included comprehensive error handling for API failures.

6. Tech Stack

The project leverages a modern and high-performance tech stack, chosen for its efficiency, scalability, and strong developer ecosystem.

6.1. Backend

- **Runtime**: **Bun** was chosen for its superior performance and significantly faster package management.
- **Framework**: **Express.js** was used for its lightweight and flexible nature, with a proven track record for building scalable APIs.
- **Database**: **SQLite** was selected for its zero-configuration, serverless nature, and **Prisma ORM** was used to provide type-safe database access and automated migrations.
- **Authentication**: **JWT tokens** and **bcrypt** were used to provide a secure and stateless authentication system.
- **RSS Processing**: The **rss-parser** library was used for reliable and efficient RSS feed parsing.

6.2. Frontend

- **Framework**: **React 19** was used to build a modern and performant user interface, leveraging its concurrent features and hook-based architecture.
- **Build Tool**: **Vite** provides a lightning-fast development server with hot module replacement and optimized production builds.
- **UI Library**: **Material-UI (MUI)** was used to create a production-ready, accessible, and responsive design system.
- **HTTP Client**: **Axios** was chosen as the HTTP client for its rich features, including request/response interceptors and robust error handling.

- **Routing**: **React Router DOM** was implemented for declarative and dynamic client-side routing.
- **Data Visualization**: **Plotly.js** was integrated to create the interactive and informative charts in the weather dashboard.

6.3. DevOps & Tools

- **Package Manager**: **Bun** offers a significant performance advantage over traditional package managers like npm.
- **Database Migrations**: **Prisma Migrate** was used to manage database schema versions and apply changes systematically.

7. Key Learnings and Reflections

The project was an invaluable learning experience, reinforcing key concepts and introducing new skills essential for full-stack development.

7.1. Technical Skills Developed

- **Advanced JavaScript (ES6+)**: This project provided practical experience with modern JavaScript features like async/await, modular programming, and destructuring.
- **Database Design and Query Optimization**: The use of Prisma allowed for the design of a relational database and the optimization of queries to improve application performance.
- **Secure Authentication and Authorization**: The implementation of JWT and bcrypt provided hands-on experience in building secure, stateless authentication systems.
- **React Component Architecture**: A modular and scalable component architecture was developed, utilizing React hooks and best practices for state management.
- **Full-stack Integration**: The project solidified the understanding of how to seamlessly connect a frontend and backend, managing data flow and error handling.

7.2. Best Practices Applied

- **Modular Code Organization**: The project's structure is highly modular, with a clear separation of concerns between the frontend, backend, and database layers.
- **Comprehensive Documentation**: Extensive documentation, including a detailed README and inline code comments, was a key focus.
- **Error Handling and Fallback Mechanisms**: Graceful error handling was implemented throughout the application to ensure a smooth user experience even when external services fail.
- **Security Measures**: The application incorporates multiple security measures, including input validation and password hashing, to protect against common vulnerabilities.
- **Backend Unit Testing**: Unit tests were written for critical backend functions and API endpoints to ensure code quality and reliability.

7.3. Development Methodology

The project followed an iterative development process, starting with core features and progressively adding advanced functionality. An API-first approach was adopted, with backend endpoints designed and tested before frontend implementation. A strong focus on user experience was maintained, with the inclusion of loading states and intuitive navigation.

7.4. Future Enhancements

While complete, the project offers several opportunities for future improvement to enhance its scalability and functionality.

- **Scalability**: Implementing Redis for distributed caching and session management would improve performance and scalability.
- **Testing Coverage**: Expanding the testing suite to include frontend component testing and end-to-end testing would increase application robustness.
- **Deployment**: Integrating Docker containerization and a CI/CD pipeline would enable automated, consistent deployments.
- **Performance**: Implementing pagination for large datasets and lazy loading for improved user experience.
- **Mobile App Support**: The next step could be to develop a React Native mobile application or enhance the existing web app with Progressive Web App (PWA) capabilities.