Title: Marvel Comic Review

Marvel-Themed Community Platform - A Web Application for Comic Enthusiasts

Jared Preyer, Paul Rodriguez, Noah Mamo, Max Knauss, Alexis Mollet

Project Description: Overview and Core Technologies

Introduction:

The Marvel-Themed Community Platform is a web application designed to create an engaging space for comic book enthusiasts. It leverages the power of modern web technologies to provide a user-friendly interface for exploring Marvel comics, discussing favorite series, and interacting with fellow fans.

Core Technologies:

- Express.js: Used for building the application server and API, enabling efficient handling of requests and responses.
- PostgreSQL with pg-promise: This SQL database system is integrated for robust data management and storage, with pg-promise facilitating seamless connection and query execution.
- Body-Parser and Express-Session: These middleware tools parse incoming request bodies in JSON format and manage user sessions, enhancing user experience and security.
- bcrypt: Implements password hashing for secure user authentication and data protection.
- EJS (Embedded JavaScript Templates): Serves as the templating engine to render dynamic HTML pages for the client-side.
- Marvel API Integration: A custom module, MarvelAPI, interfaces with the Marvel Comics API, fetching and displaying comics-related data.
- Socket.io: A JavaScript library that enables real-time, bidirectional, and event-based communication between web clients and servers, integral for implementing features like live chat and notifications. Both Socket.io and Feathers.js were used in tandem with each other in order to implement real-time functionality to the messaging system on the groups page.
- Feathers.js: An open-source web framework for building real-time applications and REST APIs using JavaScript or TypeScript, which we used to enhance our application's real-time features, like chat. Both Socket.io and Feathers.js were used in tandem with each other in order to implement real-time functionality to the messaging system on the groups page.

Application Functionality and User Interaction

User Registration and Authentication:

The platform offers a straightforward user registration and login process. It includes robust validation and encrypted password storage, ensuring a secure and trustworthy user experience.

Session Management:

User sessions are efficiently managed, allowing for persistent, personalized experiences across the application. The session data is crucial for access control and user-specific interactions.

Dynamic Content Rendering:

Users can explore a wide array of Marvel comics. The integration with the Marvel API allows the application to fetch, display, and search for comics and series data dynamically.

Interactive Features:

- Discover Page: Users can discover new comic series, with the functionality to filter and search based on various criteria like start year.
- Account Management: Users can update their profile, including changing usernames or passwords.
- Community Engagement: The application includes group creation and messaging features, fostering a community space for discussions and interactions among comic fans.

Advanced Features, Security Measures, and Future Scope

Advanced User Interactions:

- Comic Reviews: Users can post and view reviews on specific comics, enhancing community engagement.
- Series and Group Exploration: Users can explore specific comic series and participate in group discussions, further enriching the community experience.
- Real-time chat

Security and Data Integrity:

- The application ensures data security and integrity, using bcrypt for hashing passwords and implementing middleware for authentication and protection against unauthorized access.

Future Enhancements:

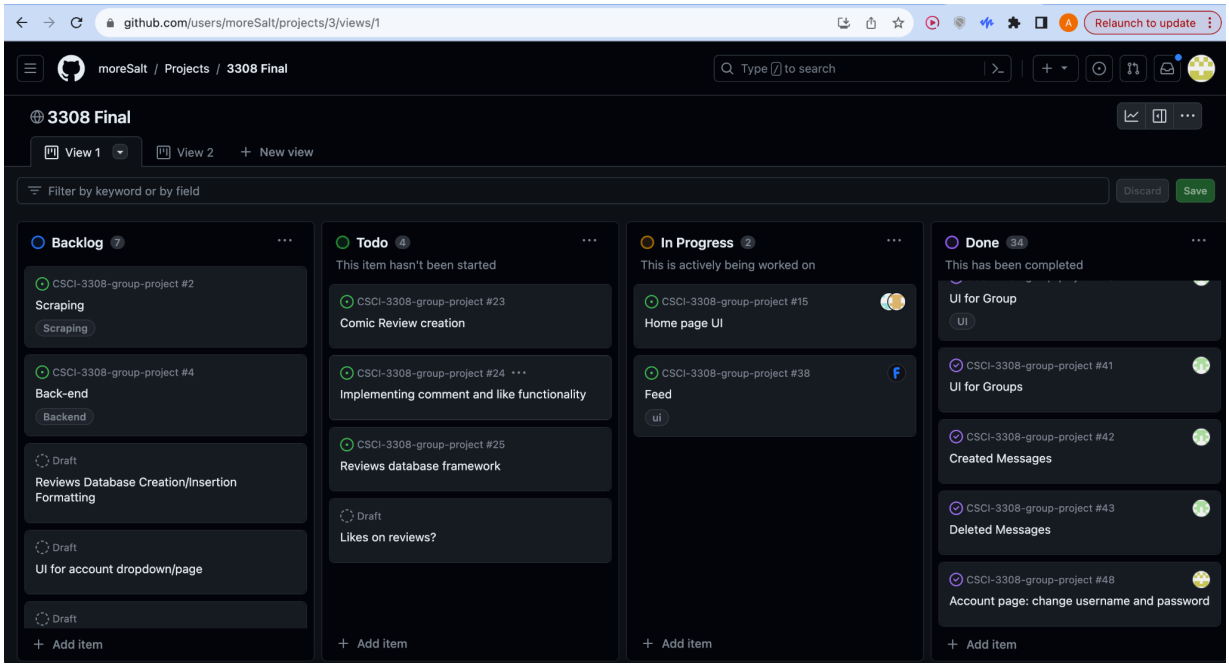
- Expansion of the Marvel API integration to include more detailed information and interactive elements.
- Implementation of real-time chat functionality for groups.
- Introduction of personalized recommendations based on user activity and preferences.
- Implement a request to groups feature that had to verify belonging to a super hero group.

Conclusion:

The Marvel-Themed Community Platform is a versatile and user-centric web application that offers a unique space for comic book enthusiasts to explore, interact, and engage with content and each other. Combining modern web technologies with a passion for comics, this platform stands out as an innovative solution in the realm of community-driven web applications.

Project Tracker - GitHub project board:

https://github.com/users/moreSalt/projects/3/views/1



Video: 5 minute or less video demonstrating your project. Your audience is a potential customer or person interested in using your product.

https://www.loom.com/share/fb82c45bcdf0420292e07c01ea08ac25?sid=0fdda033-df1a-4241-9488-988c54b3949a

VCS: Link to your git Repository. Instructor/TAs will check, weekly, to ensure the following are stored in your VCS repository:

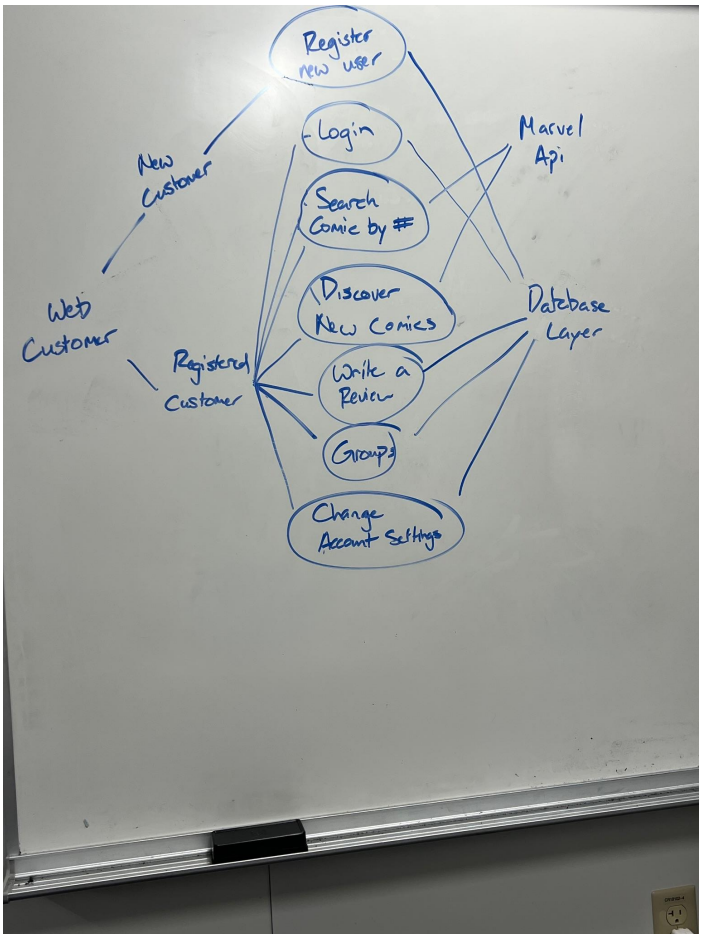
https://github.com/moreSalt/CSCI-3308-group-project

Contributions:

- A brief (not more than 100 words) from each team member about their contributions. This should include the technologies worked on and features that they have contributed to. You can also include: A screenshot of the project Board and A screenshot of the contributions on GitHub
    - Jared Preyer, This Node.js web application uses Express and EJS for dynamic web pages. EJS templates display group and message details, with forms for posting and deleting messages. Express routes handle functionalities like showing groups, creating new groups, managing messages, and interacting with a database. This structure, combining front-end templates for user interaction and back-end routes for logic and data handling, is a common web development pattern. Postman, node.js, express, socket.io, feathers.js)
    - Paul Rodriguez. Added in the initial skeleton of the project (ejs, node.js, express), created the Marvel API wrapper class (postman, node.js), Built routes and pages for comics and series (ejs, node.js, express, axios). Updated groups to use websockets (socket.io, feathers.js)
    - Noah Mamo - worked on various aspects of the project. Technologies i used to implement the task assigned were PostgreSQL, Express js, EJS, Postman, Git and Bootstrap. SQL to create tables for comics , reviews also to link them with a weak link table. EJS for templating when implementing the feed and search features. Bootstrap utilized for the UI of the feed, Review box and Search feature. Postman to test if data was being sent in the correct format for Login, Registration, Creating review and Getting review. Git to create branches such as the feed branch and to collaborate with Max on it. Express js for implementing the routes and interacting with the database.
    - Max Knauss, I primarily worked on the EJS, Express.js, and PostgreSQL technologies throughout this project. I implemented message alert functionality to the website, set up input data validation in login and registration, which was then carried over to the account page. I worked with Noah to implement the feed on the website homepage, and I also fully implemented the comic id search. I also occasionally debugged/polished the website as a whole.
    - Alexis Mollet had to execute on a diverse set of skills in both front-end and back-end development for our web application. On the front-end, he skillfully utilized EJS for templating, creating a seamless user interface for account settings,

including forms for username and password changes. In the back-end, Alexis implemented robust Express.js routes, ensuring secure and efficient user data handling. He integrated session-based authentication, data validation, and complex SQL transactions for user information updates. Additionally, Alexis configured the Express.js server, managed session states, and enabled JSON parsing. Their contribution was critical in connecting the database operations with a user-friendly interface, demonstrating a comprehensive understanding of full-stack development.

Use Case Diagram:



Test results: In Lab 11, you created a Test Plan. You need to include the test results and observations in the project report. Refer to this for more information

// TEST 1 - Registration //
User is on the /register page

Successful Registration:
Inputs:
- Username: 'testing'
- Password: 'supersafe'
Expected results:
- update to the database
- confirmation message that the user registration worked well
- user gets redirected to /login page

Leaving Required fields open:
Inputs: leave one or more required fields open
Expected result:
- Error message: 'missing required field'
- redirect to empty registration page again

Duplicate Entry:
Inputs: uses a username which already exists
Expected result:
- Error message 'Username is not available'
- redirect to empty registration page again

// TEST 2 - Login //
User is on the /login page
Successful Login
Inputs(that match those existing within the database):
- Username:
- Password:
Expected result:
- user enters the app and gets redirected to the /feed page.

Invalid username/ Password
Inputs: username and/or password does not exist within the database
Expected result:
- Error message: 'Invalid Username/password'
- redirect to empty /login to try again

Empty fields
Inputs: leave username/ password as empty
Expected result:
- Error message: 'Invalid Username/password'
- redirect to empty /login to try again

// TEST 3 - Adding a Comment // Likely need to change this into some test on groups,
User is on their /feed page and attempts to type in a Comment

Valid Comment
Input: "This comic is my favorite!"
Expected result: comment is added to the database and shows up at the top of the comment list
Empty Comment
Input: (empty)
Expected result: nothing happens we dont want to fill the database with empty comments
Special Characters
Inputs: try special characters like numbers, punctuation and emojis
Expected result:
- comment is added to database and posts to comment section like normal

Deployment: Link to deployment environment or a written description of how the app was deployed and how one might access/run the app. The app must be live, working, and accessible to your TA.

Postgres and the Node application were containerized in Docker, and deployed on an Azure compute instance. To run on localhost:3000, clone the github repo and after going into its directory, run `docker compose up -v` in your terminal.

Azure link: http://recitation-16-team-03.eastus.cloudapp.azure.com:3000/
Github: https://github.com/moreSalt/CSCI-3308-group-project