NON FUNCTIONAL TESTING

STOCKFELLOW

DEVOPPS

BRIGHTBYTE ENTERPRISES

DEMO 4

Name and Surname	**Student Number**
*Tinotenda Chirozvi	22547747
Diyaana Jadwat	23637252
Dean Ramsey	22599012
Naazneen Khan	22527533
Lubabalo Tshikila	22644106

Contents

1 Overview

This document outlines the **load and concurrency, usability, and security testing** performed on StockFellow services. The goal is to validate system performance, reliability, usability, and security.

2 Load and Concurrency Testing

2.1 Tools Used

- hey (HTTP load testing tool)
- Bash scripts for automated testing and reporting

2.2 Testing Goals

- Identify performance bottlenecks
- Measure average response time (Avg RT) and requests per second (RPS)
- Ensure endpoints handle concurrent requests efficiently

2.3 Test Environment

Component	**Details**
Test Machines	Localhost environment (Docker)
Services Tested	Notifications, Auth, Admin, Groups, Transactions, Users, MFA, etc.
Concurrent Workers	10
Requests per Endpoint	100
Redirect Handling	Disabled (`-disable-redirects`)
Sample Path Parameters	`notificationId=1`, `userId=1`, `groupId=1`, `cycleId=1`, `transactionId=1`

2.4 Methodology

1. **Endpoint Collection:** All service endpoints were collected from StockFellow APIs.
2. **Parameter Substitution:** Sample values were substituted for dynamic path parameters (e.g., `{userId}`, `{notificationId}`).
3. **Load Testing:** Each endpoint was tested using `hey` with:

   ```
   hey -n 100 -c 10 --disable-redirects <endpoint>
   ```

4. **Metrics Captured:** Avg RT and RPSs.

2.5 Load and Concurrency Results

Here are some of the results captured during the test:

Figure 1: Load and Concurrency Results Part 1

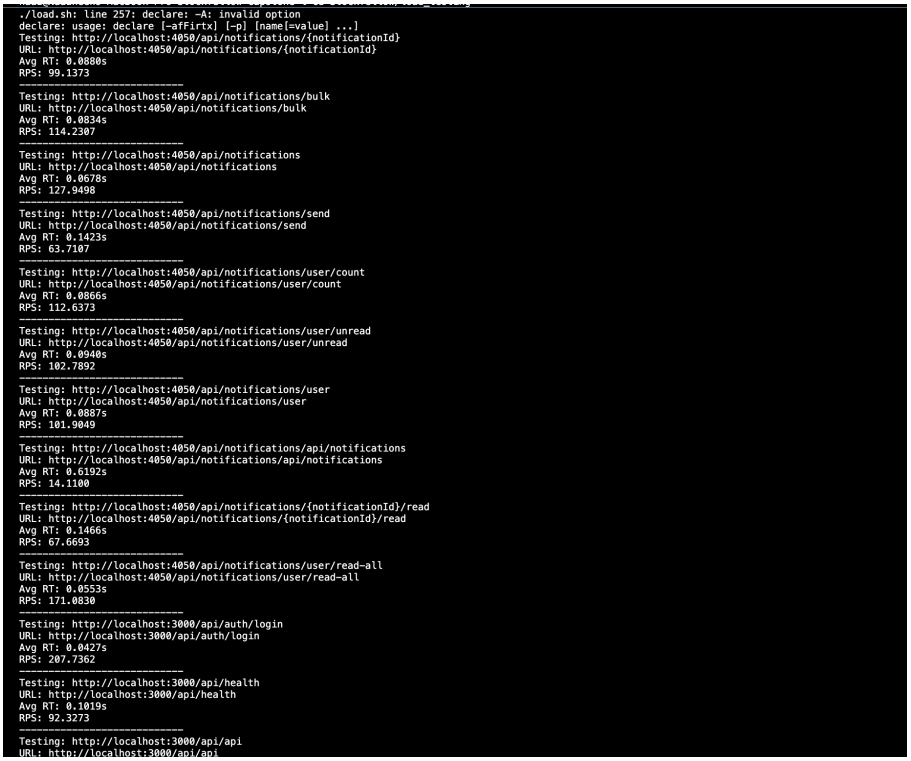

Figure 2: Load and Concurrency Results Part 2

2.6 Observations

The load and concurrency tests show that the system handles requests efficiently with generally low average response times and high request throughput. Most endpoints demonstrate excellent performance under concurrent access, indicating that the services are capable of supporting high user load. A few endpoints show slightly higher latency, but overall, the system is stable and performs well under the tested conditions.

3 Usability Testing

3.1 Usability Testing Scenarios

The following scenarios were used for task-based usability testing. Participants performed these tasks using the StockFellow app, after which feedback was collected via a Google Form to evaluate their experience.

1. **User Registration & Onboarding**
   Scenario: Opening StockFellow for the first time to create an account.
   Goal: Successfully sign up, complete verification steps, and reach the home screen.

2. **Login & Multi-Factor Authentication**
   Scenario: Logging in as an existing user to access the dashboard.
   Goal: Log in using credentials and complete multi-factor authentication successfully.

3. **Auto-Join Prompt for New Users**
   Scenario: A new user is prompted to join a suggested group.

Goal: Decide whether to accept the suggested group or explore joining/creating another.

4. **Viewing and Searching Stokvels**
   Scenario: Searching for stokvel groups to join.
   Goal: Search for groups, view details, and decide whether to request to join.

5. **Creating a New Stokvel**
   Scenario: Starting a new stokvel group for friends or colleagues.
   Goal: Create a group, fill in details, and ensure all information is accepted correctly.

6. **Managing Join Requests**
   Scenario: Managing membership requests as a group admin.
   Goal: Approve or reject requests and verify that the member list updates correctly.

7. **Add Card for Transactions**
   Scenario: Adding a card to enable contributions and payouts.
   Goal: Successfully add the card and verify it is ready for transactions.

8. **Making and Viewing Transactions**
   Scenario: Contributing to a stokvel or viewing past transactions.
   Goal: Complete contributions/payouts and view transaction history accurately.

9. **Profile & Settings Management**
   Scenario: Updating personal information.
   Goal: Make updates successfully and confirm changes persist after logging out and back in.

10. **Help & Support**
    Scenario: Seeking assistance or guidance for app features.
    Goal: Locate helpful information in the FAQ or contact support successfully.

3.2 User Feedback Collection

- Feedback was gathered using a Google Form after completing the scenarios.

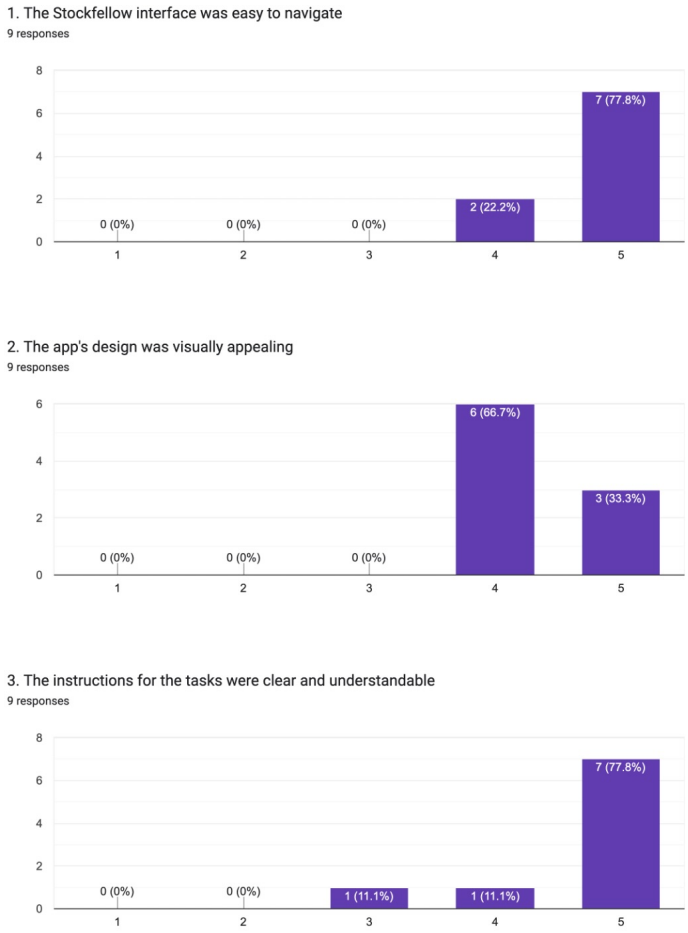

Figure 3: Responses Part 1

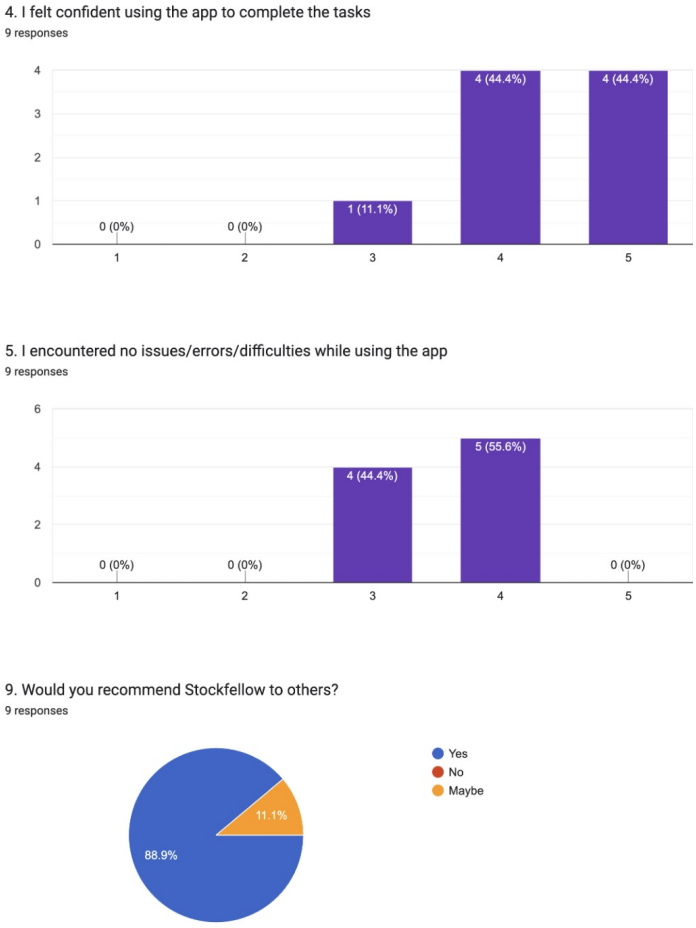

Figure 4: Responses Part 2

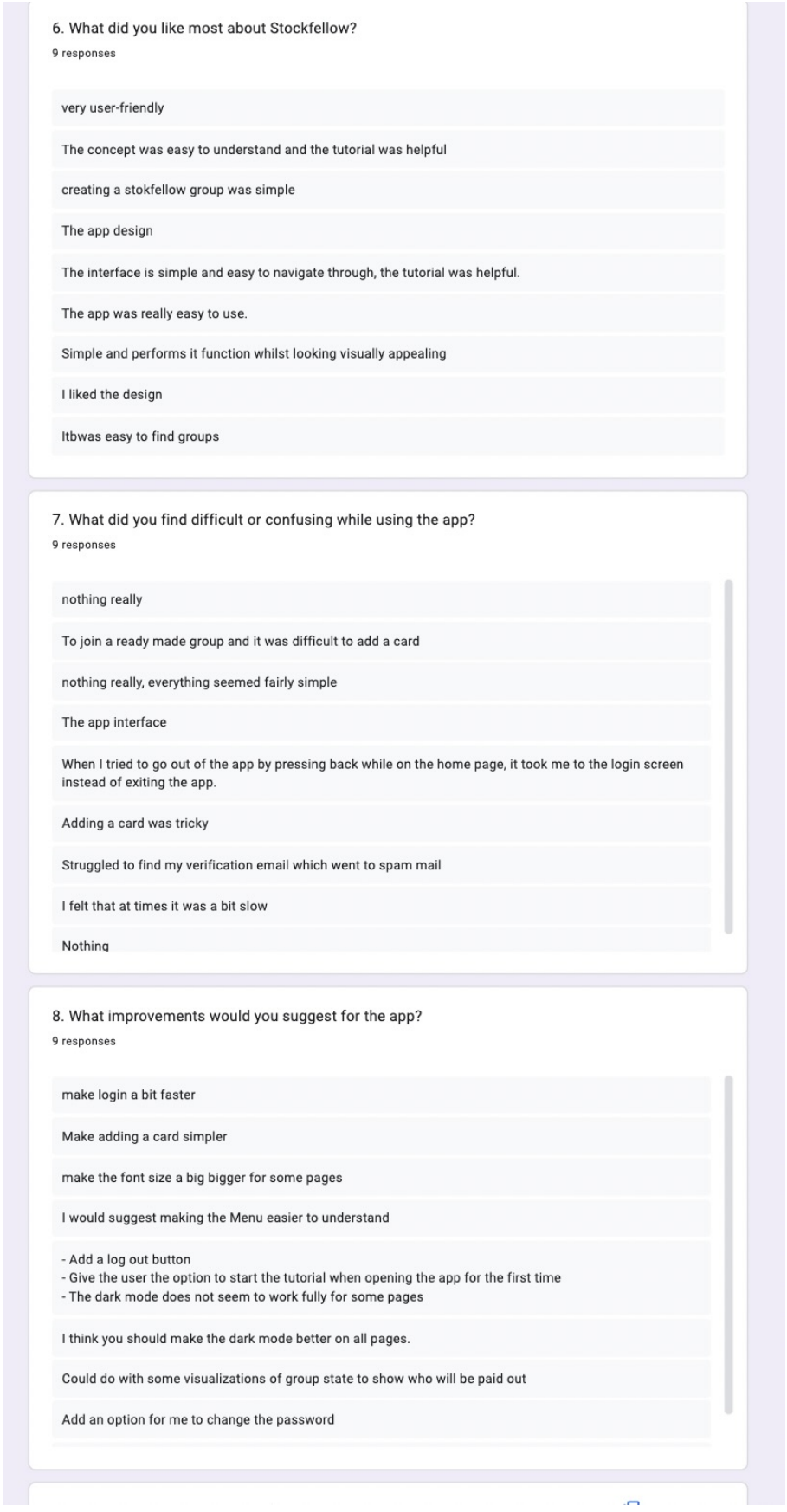

Figure 5: Responses Part 3

3.3 Summary of Observations

- Users were able to complete tasks successfully and efficiently, demonstrating good usability.
- The app's design and interface received positive feedback, indicating a generally favourable user experience.
- Minor usability issues were identified, which could be addressed in future updates.
- Overall, user feedback suggests satisfaction with the app, while highlighting areas for potential improvement.

4 Security Testing

4.1 Overview

Security testing was conducted to evaluate the resilience of StockFellow services and containerized infrastructure against vulnerabilities and misconfigurations. The goal was to ensure data confidentiality, integrity, and availability.

- **Trivy:** Used for scanning container images and project dependencies for vulnerabilities and misconfigurations.

4.2 Methodology

The assessment leveraged the **Trivy Security Scanner** to perform container security scans across all service images. The scan included:

- Detection of known vulnerabilities in OS packages, libraries, and application dependencies.
- Identification of container configuration issues, such as running as root, missing `HEALTHCHECK`, and missing `-no-install-recommends` flags.
- Secret scanning was enabled to check for exposed credentials in the filesystem.
- Classification of vulnerabilities by severity (Critical, High, Medium, Low).

4.3 Results

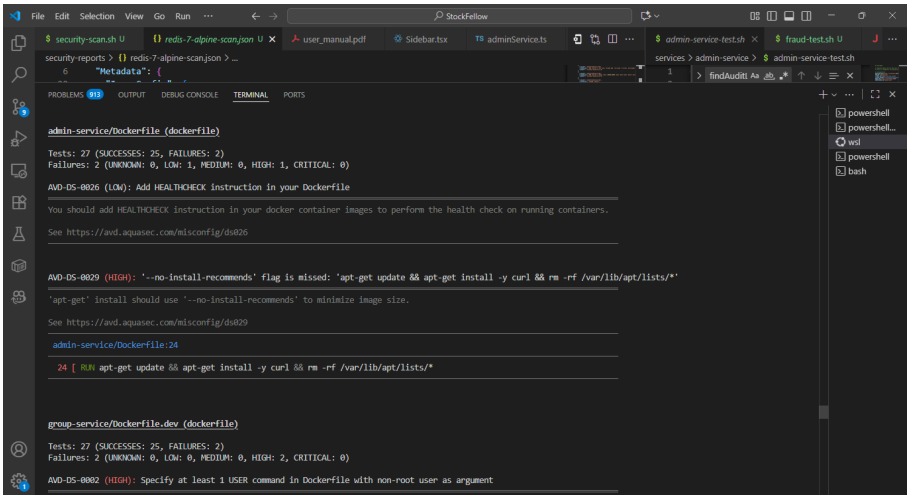

Figure 6: Security Testing Part 1

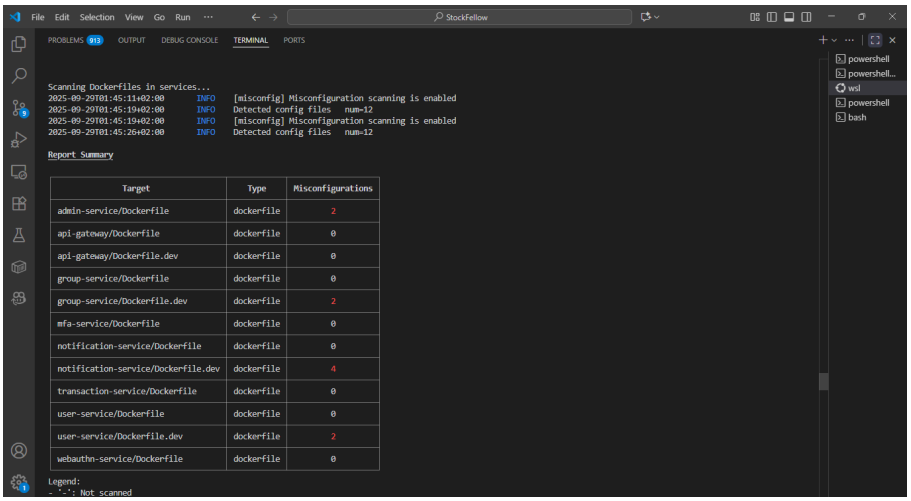

Figure 7: Security Testing Part 2

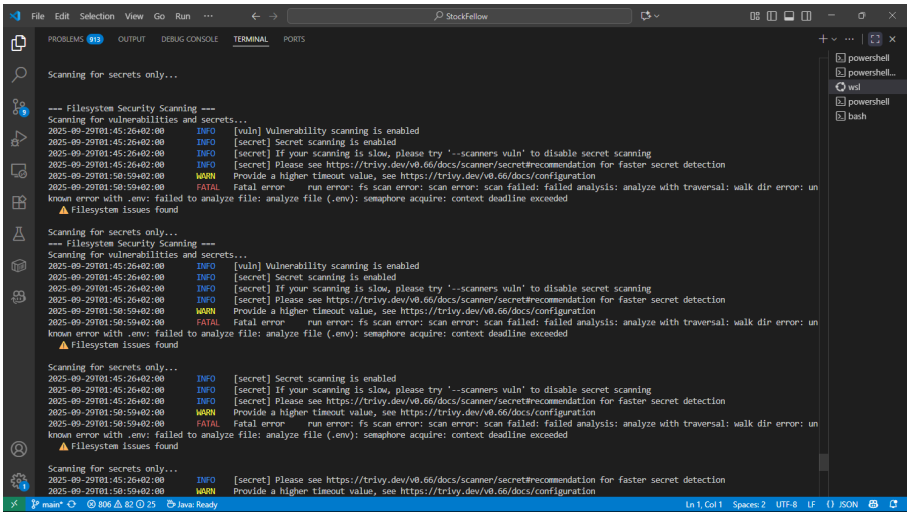

Figure 8: Security Testing Part 3

4.4 Observations

- Several containers passed scans without issues, but a few images had critical and high-severity vulnerabilities that require immediate attention (e.g., SQLite, Jetty, Netty, Expat).
- Common Dockerfile misconfigurations were detected, including missing `HEALTHCHECK` instructions and running containers as root without a non-root `USER` command.
- Secret scanning did not reveal any critical secrets; however, some filesystem scan errors occurred due to timeout settings.
- Overall, while many components show good security practices, the critical and high-severity vulnerabilities found need to be fixed promptly to maintain a strong security posture.