Nosedive



Table of Contents

[6.1 Final Implementation 2](#_Toc165272764)

[6.2.1 introduction 2](#_Toc165272765)

[6.2 Application Design preparation 2](#_Toc165272766)

[6.3 Tools Used 4](#_Toc165272767)

[6.4 Iteration and improvements 4](#_Toc165272768)

[6.6 Database Design & Configuration 6](#_Toc165272769)

[6.7 Business Rules/Algorithm 8](#_Toc165272770)

[6.7 Features 9](#_Toc165272771)

[6.7.1 Index.js 9](#_Toc165272772)

[6.7.2 Homepage.js 10](#_Toc165272773)

[6.7.3 Explore.js 11](#_Toc165272774)

[6.7.4 Rating.js 12](#_Toc165272775)

[6.7.5 Profile.js 12](#_Toc165272776)

[6.7.6 Login.js 13](#_Toc165272777)

[6.7.7 APPLICATION SCREEENSHOTS 14](#_Toc165272778)

[6.8 Testing 16](#_Toc165272779)

[6.8.1 Whitebox testing 16](#_Toc165272780)

[6.8.2 Blackbox testing 17](#_Toc165272781)

[6.8.3 User Testing 17](#_Toc165272782)

[6.9 Evaluation 18](#_Toc165272783)

[6.9.1 Challenges 18](#_Toc165272784)

[6.9.2 Limitations 19](#_Toc165272785)

[6.9.3 PROJECT OUTCOMES 19](#_Toc165272786)

[6.9.4 PERSONAL REFLECTION 20](#_Toc165272787)

[6.9.5 Future enhancements 20](#_Toc165272788)

[7. Demonstration Video 20](#_Toc165272789)

[7.1 Link to Application 21](#_Toc165272790)

[7.2 References 21](#_Toc165272791)

[7.3 APPENDIX 21](#_Toc165272792)

# 6.1 Final Implementation

## 6.2.1 introduction

This section delves into the project’s implementation stage. It includes the code for the system’s main components and explains how it achieves the desired outcome. The section is divided into sections that discuss each system component in terms of relevant programming and scripting languages.

## 6.2 Application Design preparation

**GitHub Setup:** In the initial phase of the project development, a GitHub repository was established to leverage the advantages of version control and collaborative features.

**A screenshot of a computer

Description automatically generated**

Figure 1. Shows initial GitHub repository.

**Node modules:** I organized the backend code into different Node.js modules, each corresponding to specific functionalities of the application's web pages. This modularization approach ensured a clean and efficient architecture by encapsulating distinct functionalities within separate files [1].

**A screen shot of a computer program

Description automatically generated**

Figure 2. Showcases the different modules.

**Libraries installed:**

Several external modules were installed via Node.js's package manager using npm. These include essential libraries such as Express for server-side routing, EJS for templating, and MySQL for database interactions. Security-oriented modules like Bcrypt for hashing and securing passwords and 'dotenv' for managing environment variables were also integrated.

A computer code with text

Description automatically generated

Figure 3. Showcases imported Libraries.

Table E below shows the list of routes that will be required for the application to function.

| **Route** | **Method** | **Purpose** | **Authorization** | **Validation** |
| --- | --- | --- | --- | --- |
| / | GET | Home/landing page |  |  |
| /profile | GET | User profile | Signed in users only |  |
| /login | GET | Login form | Signed out users only |  |
| /login | POST | Login form handler | Signed out users only | Username: present, Password: present |
| /signup | GET | Register form | Signed out users only |  |
| /signup | POST | Register form handler | Signed out users only | Username: not taken, Password: valid format |
| /logout | GET | Logout user | Signed in users only |  |
| /contact | GET | Contact form |  |  |
| /explore | GET | Explore page |  |  |
| /rate | POST | Rating form handler | Signed in users only | Rating: valid range |

Figure 4. Required routes for the application.

## **6.3 T**ools Used

The implementation of the web application was done by using the following tools:

**Node.js**: Served as the runtime environment for my application, handling server-side logic.

**Express.js**: A web application framework for Node.js for routing and middleware functionalities.

**EJS (Embedded JavaScript Templating):** I used it to generate HTML templates with embedded JavaScript, simplifying the creation of dynamic web pages.

**CSS:** I used it to style the application's front end, ensuring the web pages are visually appealing and functional.

**Bootstrap**: I used this front-end framework to design responsive web pages. It includes various ready-to-use components that enhance UI design and user experience.

**JavaScript**: The primary scripting language for both client-side and server-side development.

## 6.4 Iteration and improvements

**Front-End improvements:** In response to user feedback, front-end improvements to the web application included a revised colour scheme for better visual appeal and readability, alongside enlarged buttons to enhance usability and accessibility, particularly on mobile devices.

A screenshot of a computer

Description automatically generated

Figure 5 Showcases Survey conducted on Design choices.

A pie chart with different colors

Description automatically generated

Figure 6. Showcases the results of the Survey.

**Backend improvements**: Backend changes involved a shift from NoSQL to MySQL for the database system. I decided to do this because of the anticipated low volume of data, which MySQL handles more efficiently. Additionally, MySQL provides robust relational data integrity and supports complex queries.

**Rating system changes:** After user feedback that the old 5-star system was too simple, I upgraded it to encompass various evaluation aspects. The altered system now enables people to assess competency, likability, and influence on 4 options namely: very, yes, somewhat and no. This methodology is like what Photo Feeler employs to offer intricate resonance between the feedback given by users as well as their engagement patterns on this site.

**A screenshot of a computer

Description automatically generated**

Figure 7. Showcases updated Rating system.

## 6.6 Database Design & Configuration

**Database Structure Overview:** The database is structured in MySQL, creates a normalized design to minimize redundancy and ensure data integrity. My design comprises of two primary entities, users, and ratings. My setup begins by setting the SQL mode and transaction properties.

A black screen with white text

Description automatically generated

Figure 8 Showcases transaction setup.

**Users Entity:**

The user's table is designed to store user account information. Each record in the table represents a unique user, identified by an auto-incrementing primary key (id). The table includes:

* Fields for storing the user's full name (full\_name).
* Email address (email).
* Profile picture URL (profile).
* Hashed password (password).

The email field is also indexed with a unique constraint to ensure no two users can register with the same email address. This table is fundamental for user authentication and profile management functionalities within the application.

A black background with white text

Description automatically generated

Figure 9. Showcases User Table.

**Ratings Entity:**

In the ratings table, comments left by users for each other are recorded. This table has the user (the rated user's ID), rated by (the rater's user ID), and an auto-incrementing ID as the primary key for every rating instance. The ID of the user table was linked to both foreign keys. Such an arrangement ensures relational integrity in my web app. Ratings can be categorized using three different attributes:

competent\_rating, likable\_rating, and influential\_rating, each restricted to store integers. A text field called comment permits users to give textual feedback. The time field automatically captures the timestamp of when the rating was made and provides chronological context to this input.

A computer screen shot of a program code

Description automatically generated

Figure 10. Showcases Rating table.

**Normalization and Query Utilization:** The database structure adheres to at least the Third Normal Form (3NF) [2]:

* **1NF** is achieved as each table has a primary key, and the values in each column are automatic.
* **2NF** and **3NF** are maintained by ensuring that all attributes depend only on the primary key and there are no transitive dependencies.

Queries using this structure efficiently retrieve, insert, and manage user and rating data. It enables my application to show a comprehensive user profile with ratings and feedback.

## 6.7 Business Rules/Algorithm

**Overview of Rating Algorithm:** The Nosedive web application's algorithm determines average user ratings based on three factors: influence, likability, and competence. By giving their assessments more weight, my algorithm gives preference to users who have higher ratings, thereby recognizing their greater standing or reliability in the rating system. It was modified for this project using the Bayesian Algorithm [3] during its construction.

**Algorithm Explanation:** The weighted averages for the three rating categories are calculated by the getAverageRatings function. By multiplying each rating by ten and dividing the result by three, a weight is assigned to each rating. This process naturally scales and modifies the impact of each rating according to the strength of its corresponding attribute. Next, the outcomes are averaged across all ratings:

* **Competence:** This is calculated by summing the adjusted competence ratings of all reviews for a user and dividing by the number of ratings.
* **Likability:** Competence, the likability score is grouped by summing the adjusted values and then averaging them.
* **Influence:** The influence score is computed by accumulating the weighted influence ratings and averaging them.

A screen shot of a computer program

Description automatically generated

Figure 11. Shows Rating Algorithm.

**Time Complexity Analysis:** The getAverageRatings function has an O(n) time complexity, where n is the number of ratings. All ratings are processed in a single iteration through the array, and each rating takes a fixed amount of time to process.

**Influence on Community Dynamics**: The way in which the algorithm assigns greater weight to user ratings has a significant impact on the social dynamics of my web application. Through a positive feedback loop that increases the influence of highly rated users. This could lead to a stratification within the user base, making it harder for new or inactive users to change their reputation or visibility. I implemented a decaying influence algorithm [4] which ensures that newer interactions have more significance than older ones. This addresses the possibility of stratification within the "Nosedive" application.

A screen shot of a computer program

Description automatically generated

Figure 12. Showcases Decaying Algorithm.

## 6.7 Features

### 6.7.1 Index.js

**Purpose and Interactions**: index.js is the central configuration file, setting up the server, database connections and middleware.

* **Server and Database Setup**: Created the Express server that connects to my MySQL database which manages application data.
* **Middleware Integration**: Configured body parser for form data handling and session middleware for user session management.
* **Modules Setup**: The different modules for each page are initialised.

**Key Features:**

* Express server initialization
* Middleware configuration for body parsing and session management
* Route definitions and management

A screen shot of a computer program

Description automatically generated

Figure 13. Shows Express and Body parser setup.

### 6.7.2 Homepage.js

**Purpose and Interactions:** homepage.js manages the main landing page of the Nosedive application. It dynamically renders content based on whether the user is logged in or not. For logged-in users, it redirects them to their profile page. For visitors, it displays top users based on their ratings.

**Detailed Functionality:**

* **User Redirection:** It will redirect logged-in users to their profile page by providing direct access to personal content.
* **Database Query Execution:** For visitors, it executes a SQL query to retrieve all users from the database. For logged-in users, it fetches all other users except the logged-in user.
* **Dynamic Content Rendering:** After fetching user data, it calculates average ratings for each user and displays a sorted list of top users based on their ratings.

**Key Features:**

**Dynamic SQL Querying:** Depending on the user’s session status, it tailors SQL queries to fetch appropriate user data.

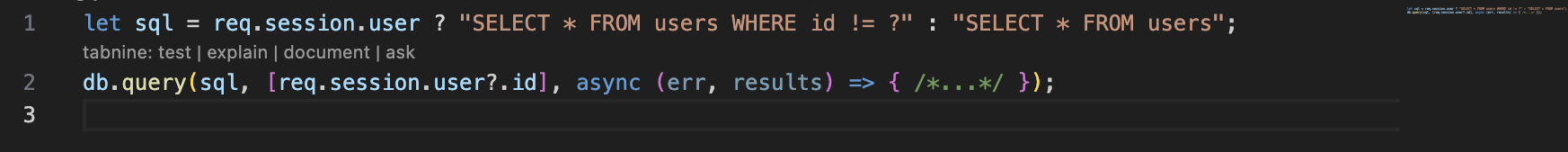


Figure 14. Showcases fetching user data.

**Top Users Display:** Sorts users by the number of ratings and their average ratings, then selects the top four to display on the homepage.

### 6.7.3 Explore.js

**Purpose and Interactions:** Explore.js this feature creates user exploration of other profiles within the Nosedive application, allowing users to search, filter, and view profiles based on specific criteria. My web application provides dynamically filtered results and redirects non-authenticated users to the login page.

**Detailed Functionality:**

* **Authentication Check:** Redirects non-authenticated users to the login page, making sure that only logged-in users can explore other profiles.
* **Dynamic SQL Querying:** Executes a SQL query to fetch all users except the logged-in user.
* **Search and Filter:** Functionality to filter the results based on a search query entered by the user.

**Key Features:**

**User Filtering Logic:** Implements logic to filter users dynamically based on user input from a search bar. This allows users to find profiles by names that contain the typed query.

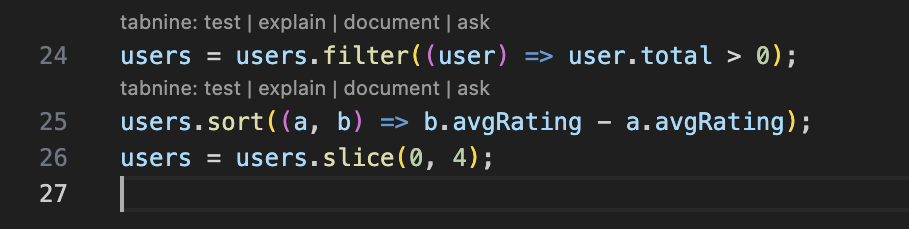


Figure 15 Showcases top user’s filter.

**Average Rating Calculation and Sorting:** After fetching ratings for each user, it calculates average ratings and sorts users based on the specified sort criteria ('suggested' or 'featured').

A screen shot of a computer program

Description automatically generated

Figure 16. Showcases rating calc code.

### 6.7.4 Rating.js

**Purpose and Interactions:** rating.js manages the rating interactions within the Nosedive application, allowing users to rate other users based on different criteria. This file handles both the display of user ratings and the submission of new ratings.

**Detailed Functionality:**

* **Rating Display:** When a user navigates to the rating page, the application fetches existing ratings for a specific user from the database. It displays these ratings, or a default message if no ratings exist.
* **Rating Submission:** Allows logged-in users to submit ratings for other users, capturing several aspects of user performance such as competence, likability, and influence.

**Key Features:**

**User Authentication and Redirection:** Redirects non-authenticated users to the login page, ensuring that only logged-in users can access the rating functionalities.

**SQL Query for User Ratings:** Executes a SQL query to fetch the user details along with their ratings. If ratings exist, it calculates average ratings; if not, it initializes average ratings to zero.

**Handling Rating Submissions:** Processes POST requests for new ratings, inserting them into the database and then redirecting the user back to the explore page.

A screen shot of a computer code

Description automatically generated

Figure 17. Showcasing post request for rating submissions.

### 6.7.5 Profile.js

**Purpose and Interactions:** profile.js handles user profile interactions within the Nosedive application, focusing on displaying user ratings and managing profile updates.

**Detailed Functionality:**

* **Profile Display:** When a user accesses their profile, the application queries the database for their ratings calculating average ratings if any exist. If no ratings are found it displays the default values.
* **Profile Update:** Functionality for users to update their profile picture using Multer middleware, which handles file uploads securely.

**Key features:**

**Authentication Check and Redirection:** Redirects non-authenticated users to the login page.

**SQL Query for User Ratings:** Executes a SQL query to fetch the user's ratings and calculates an average based on the competent, likable, and influential scores.

A screen shot of a computer

Description automatically generated

Figure 18. Fetching User ratings data showcase.

### 6.7.6 **Login.js**

**Purpose and Interactions:** login.js manages user authentication within my web application, securely verifying user credentials and handling user sessions.

**Detailed Functionality:**

* **Login Page Rendering:** Displays the login page, redirecting already logged-in users directly to their profile, enhancing user flow and security.
* **User Authentication:** Validates user credentials against stored data in the database and handles session setup upon successful login.

**Key Features:**

**Session Management and Redirection:** Checks if a user is already logged in; if so, redirects to the profile page to avoid redundant login attempt.

A screenshot of a computer

Description automatically generated

Figure 19. Redirection Showcase.

**Credential Verification:** Verifies email and password during the login process. It uses bcrypt to safely compare the provided password with the hashed password stored in the database.

### 6.7.7 APPLICATION SCREEENSHOTS

The below figures show the key components of the application:

A screenshot of a web page

Description automatically generated

Figure 20. Homepage Screenshot of Web application

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a login screen

Description automatically generatedA screenshot of a login form

Description automatically generated

Figure 21. Explore Users Page.

Figure 22. User profile Page.

Figure 23. Shows the Sign in page.

Figure 24. Shows the Login page.

A screenshot of a computer

Description automatically generated

Figure 25. Showcases user rating system.

A screenshot of a contact us

Description automatically generated

Figure 26. Showcases the contact page.

## **6.8 Testing**

Three tests will be completed to test the functionalities of Nosedive thoroughly: White Box testing, Black Box testing, and User testing. It is worth noting that this project was completed using an Agile methodology, Test-Driven Development (TDD) [5]. This involved testing every function as it was created and remediating any issues identified.

### 6.8.1 Whitebox testing

White Box testing confirms that the internal structure and functionalities are working as expected [6]. The testing was completed by a software developer who understands the code. The table below shows the main functionalities of the Nosedive web application and the results of White Box testing (creating various use cases that relate to different parts of the app to ensure they all work correctly). The table can be viewed on ([Link](#Whitebox))

### 6.8.2 Blackbox testing

Black-Box testing is performed after the application has been developed, compared to White-Box testing during development. Black-Box is used to test that the application works as expected but not by testing the internal components [7]. Below you will find the table of tests conducted. The table can be viewed on ([Link](#Blackbox))

### 6.8.3 User Testing

User Testing was also used as it is essential to gain feedback from individuals, and any improvements (such as design or extra features) would be considered in future releases of this app. User testing can also provide valuable results about the usability of the app and system acceptance (as users can state afterward whether they had an issue using the app).

The individuals were briefed on the application but were not given any instructions or walkthroughs; this would help test whether the app was intuitive and whether the usability was acceptable (as the individuals would be asked questions regarding the app's navigation). The individuals were given 30 minutes to use the app and then provide feedback via an online anonymous survey (rather than face-to-face, to ensure that results were more reliable as individuals may not give accurate opinions in person). The complete survey can be found in ([Link](#Form)). The results are shown below:

**Question 1:**

A pie chart with different colored circles

Description automatically generated

Figure 28. How easy was it to navigate through the app?

**Question 2:**

A pie chart with different colored circles

Description automatically generated

Figure 29. How fair did you find the rating system in the application?

**Question 3:**

**A pie chart with text

Description automatically generated**

Figure 30. How responsive do you find the application during use.

## 6.9 Evaluation

### 6.9.1 Challenges

**Scheduling and Project Management:**

One of the primary challenges I encountered during the development of the web application was managing the project timeline effectively. Balancing the demands of the project with academic responsibilities and personal commitments proved to be particularly taxing. The iterative nature of the development process, including time allocated for planning, designing, coding, testing, and revising, often conflicted with other obligations. This misalignment sometimes led to rushed sessions and extended work hours, impacting both the project's progress and my personal well-being.

**Ensuring Fairness in the Rating System:**

Creating a fair and unbiased rating system presented a considerable challenge. The initial model tended to favour users who were early participants in the platform, skewing the ratings disproportionately. Adjusting the algorithm to weight ratings based on a variety of factors—such as the number of ratings, the recency of feedback, and the diversity of raters—required careful thought and multiple iterations. This adjustment aimed to produce a more balanced and equitable system, where all users, regardless of when they joined, had a fair representation in the ratings.

### 6.9.2 Limitations

**Scalability Concerns**

One of the inherent limitations of the current implementation of the web application is its scalability. While the application performs adequately under the current user load, the backend architecture and database design may not effectively handle a significant increase in user numbers or data volume. This limitation is partly due to the simplistic query structures and the lack of distributed database solutions, which could impede performance as the system scales.

**Real-Time Data Processing:**

Currently, the application does not support real-time data processing, which limits the interaction dynamics of the web. Particularly in features like live ratings or updates. Users must manually refresh pages to see updated data, which can detract from the user experience and decrease the system's efficiency and appeal in scenarios where real-time feedback is crucial.

### 6.9.3 PROJECT OUTCOMES

**Result Analysis:**

The final iteration of my web application notably advanced from the initial concept in terms of functionality and user interaction. Provides an integrated profile management, rating systems, and advanced search features, the application met the fundamental objectives set at the project's outset. However, while the core functionalities were implemented as planned, certain advanced features like real-time notifications and a fully adaptive user interface were deferred for future updates due to time constraints.

**Aims and Objectives Fulfilment:**

My project successfully achieved its primary aims, including creating a secure, user-friendly platform where users could interact, rate each other and update profiles. The integration of third-party libraries for features like password hashing and image uploading enhanced the application’s security and usability. My project still has room to grow given only the MVP was met. Other enhancements are planned for future iterations of the application.

### 6.9.4 PERSONAL REFLECTION

**Development Methodology Efficacy:**

The iterative development approach adopted for this project proved highly effective, allowing for continuous feedback and incremental improvements. Regular reviews and adaptations helped address issues promptly, though at times the pace felt rushed due to tight deadlines.

**Reflecting on Time Management:**

Managing the project timeline was a continual balancing act. While I met most deadlines, the experience underscored the importance of effective time management and its impact on project quality and stress levels.

### 6.9.5 Future enhancements

**Adoption of NoSQL Databases:**

To better accommodate growing user volumes and the need for scalable, flexible data storage solutions, future developments will include transitioning to a NoSQL database. This shift will facilitate more efficient handling of large data sets and unstructured data.

**Improved Time Management Techniques:**

Further enhancements will focus on refining project management and scheduling techniques. Adopting proven strategies such as the Pomodoro Technique will enhance productivity and efficiency, ensuring timely delivery of updates and maintenance, while also mitigating the stress associated with tight deadlines.

## 7. Demonstration Video

To be added.

## 7.1 Link to Application

Github Link: <https://github.com/Mo2225z/Final-Year-Project>

## 7.2 References

[1] A. Singh, ‘Code Like a Pro: The Benefits of Modular Coding in Data Science’, Medium. Accessed: Apr. 29, 2024. [Online]. Available: https://medium.com/@Ambarish\_224/code-like-a-pro-the-benefits-of-modular-coding-in-data-science-5c00ae46be49

[2] ‘What is Database Normalization?’, Data Management. Accessed: Apr. 29, 2024. [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/normalization

[3] ‘Using the Bayesian average in custom ranking | Algolia’, Algolia Documentation. Accessed: Apr. 29, 2024. [Online]. Available: https://www.algolia.com/doc/guides/managing-results/must-do/custom-ranking/how-to/bayesian-average/

[4] N. Ohsaka, Y. Yamaguchi, N. Kakimura, and K. Kawarabayashi, ‘Maximizing Time-Decaying Influence in Social Networks’, in *Machine Learning and Knowledge Discovery in Databases*, P. Frasconi, N. Landwehr, G. Manco, and J. Vreeken, Eds., Cham: Springer International Publishing, 2016, pp. 132–147. doi: 10.1007/978-3-319-46128-1\_9.

[5] Asana, ‘What Is Agile Methodology? (A Beginner’s Guide) [2024] • Asana’, Asana. Accessed: Apr. 29, 2024. [Online]. Available: https://asana.com/resources/agile-methodology

[6] ‘What Is White Box Testing | Types & Techniques for Code Coverage | Imperva’, Learning Center. Accessed: Apr. 29, 2024. [Online]. Available: https://www.imperva.com/learn/application-security/white-box-testing/

[7] ‘What is Black Box Testing | Techniques & Examples | Imperva’, Learning Center. Accessed: Apr. 29, 2024. [Online]. Available: https://www.imperva.com/learn/application-security/black-box-testing/

## 7.3 APPENDIX

Table 1. Showcases Whitebox testing.

| **Function** | **Input** | **Process** | **Expected Output** | **Actual Output** | **Result** |
| --- | --- | --- | --- | --- | --- |
| **login() in login.js** | Correct email and password | Queries database with email, bcrypt compares hashed passwords | User is redirected to profile page | User is redirected to profile page | Pass |
| **login() in login.js** | Incorrect password | Queries database with email, bcrypt compares hashed passwords | Error message: "Incorrect password" displayed | Error message: "Incorrect password" displayed | Pass |
| **login() in login.js** | Email not found in database | Queries database with email | Error message: "User not found" displayed | Error message: "User not found" displayed | Pass |
| **logout() in login.js** | User selects logout | Session is destroyed, user redirected to login page | User is redirected to login page | User is redirected to login page | Pass |
| **signUp() in signup.js** | Enter all fields correctly | Insert user data into database, hash password with bcrypt | User is registered successfully | User is registered successfully | Pass |
| **signUp() in signup.js** | Input an email that is already taken | Attempt to insert duplicate email into database | Error message: "Email already exists" displayed | Error message: "Email already exists" displayed | Pass |
| **signUp() in signup.js** | Input a password shorter than required length | Validation fails due to password length check | Error message: "Password too short" displayed | Error message: "Password too short" displayed | Pass |
| **profile() in profile.js** | Access profile with valid session | Fetch user details and ratings from the database | User profile and ratings displayed | User profile and ratings displayed | Pass |
| **profile() in profile.js** | Access profile without valid session | Redirect to login due to missing session | User is redirected to login page | User is redirected to login page | Pass |
| **uploadProfile() in profile.js** | Upload a valid image file | File is stored, user profile updated in database | Success message: "Profile Updated" displayed | Success message: "Profile Updated" displayed | Pass |
| **uploadProfile() in profile.js** | Upload an invalid file format | File format check fails | Error message: "Invalid file format" displayed | Error message: "Invalid file format" displayed | Pass |
| **explore() in explore.js** | Valid session and no query | Fetch and display all user profiles except the user's own | Profiles displayed sorted by default | Profiles displayed sorted by default | Pass |
| **explore() in explore.js** | Valid session and search query present | Fetch profiles and filter based on query | Profiles matching query displayed | Profiles matching query displayed | Pass |
| **rating() in rating.js** | Valid session and post valid rating | Store rating in database, update user's average rating | Rating successfully submitted, redirected to explore | Rating successfully submitted, redirected to explore | Pass |
| **rating() in rating.js** | No session when attempting to rate | Redirect to login page | User is redirected to login | User is redirected to login | Pass |
| **homepage() in homepage.js** | Valid session, user logged in | Redirect to user profile | User is redirected to profile page | User is redirected to profile page | Pass |
| **homepage() in homepage.js** | No session, visitor access | Display homepage with top users based on ratings | Homepage displays top users | Homepage displays top users | Pass |

Table 2. Showcases Blackbox Testing.

| **Description** | **Process** | **Expected Result** | **Actual Result** | **Test Result** |
| --- | --- | --- | --- | --- |
| **Login Redirect** | The user accesses the homepage and logs in. | User is redirected to their profile page after login. | User is redirected to their profile page after login. | Pass |
| **Profile Edit** | User edits their profile information. | The profile page updates with new information. | The profile page updates with new information. | Pass |
| **Upload Profile Picture** | User selects to upload a new profile picture. | User selects an image, and it updates their profile picture. | User selects an image, and it updates their profile picture. | Pass |
| **Explore Profiles** | Logged-in user accesses the explore page. | Displays profiles other than the user’s own. | Displays profiles other than the user’s own. | Pass |
| **Search Users on Explore** | User types into search bar on the explore page. | Displays profiles matching search criteria. | Displays profiles matching search criteria. | Pass |
| **Post Rating** | User posts a rating for another user. | Rating is saved and reflected in the user's ratings. | Rating is saved and reflected in the user's ratings. | Pass |
| **Logout Functionality** | User clicks logout. | User is logged out and redirected to the login page. | User is logged out and redirected to the login page. | Pass |

**Conducted google survey:**

<https://docs.google.com/forms/d/e/1FAIpQLSfXzeh27tNUN3ix5Ndvu8jho1gSbK19EC656u22YVcPxaET_A/viewform?usp=sf_link>

<https://docs.google.com/forms/d/e/1FAIpQLSc2L1akxkk8mMtNggTnIIdwMMY9iYichXW1rpf5ByHocG9e9A/viewform?usp=sf_link>

A pie chart with text

Description automatically generated

A pie chart with different colored circles

Description automatically generated

A pie chart with different colored circles

Description automatically generated