Final Project Report

Beri Peric, Selena Chiang, Dakota Bell, and Caleb Shafer

Capstone Project Spring 2024

The University of West Florida

April 25, 2024

CIS4595 - 12716

Dr. Bernd Owsnicki-Klewe

Table of Contents

Table of Contents

1 Project Description

1.1 Ideas and Concepts:

HeartZone is an innovative online dating app targeted at individuals with a passion for computer science and technology. The app aims to connect users based on shared interests like programming, AI, cybersecurity, etc., using efficient matching algorithms. The app focuses on providing an intuitive user interface and a personalized experience for tech enthusiasts. The project emphasizes quality priorities such as usability, accessibility, efficiency, security, and user feedback.

1.2 Functions of the project:

- User profile creation with advanced filters for match preferences (programming language, sexuality, fields of study, professional experience).
- Client-side app for Android devices.
- S3 Buckets for user account and data management.
- Integration with third-party APIs for location data (Maps SDK, Google Geofencing API) to assist with proximity matching.
- Chat feature for users to communicate with their matches.
- Matching algorithm to connect users based on shared interests and preferences.
- User feedback and evaluation mechanisms (surveys, task-based testing, expert reviews) to ensure usability and continuously improve the app.
- Security measures such as input validation, static analysis, and penetration testing to protect user data and withstand threats.
- Performance monitoring and load testing to ensure responsiveness and stability of the app.

2 Final and Initial Timeline Comparison

Week	Initial Timeline	Final Timeline
1-4	Brainstormed ideas Project initiation phase	Brainstormed ideas Project initiation phase
5-6	Creating technical and functional requirements Frontend Design with Figma	Creating technical and functional requirements Frontend Design with Figma (Ran into Figma formatting frontend wrong)
7-8	Frontend development Start on backend	Started frontend development Researched backend
9-10	Continue backend development	Continued frontend development Started backend development Tested on Android emulator Cutbacks on features Created test cases
11-12	Adjusting functionality and appearance Testing	API integration Fixed appearance and test cases Started matchmaking algorithm

3 Project Results Compared with Expectations

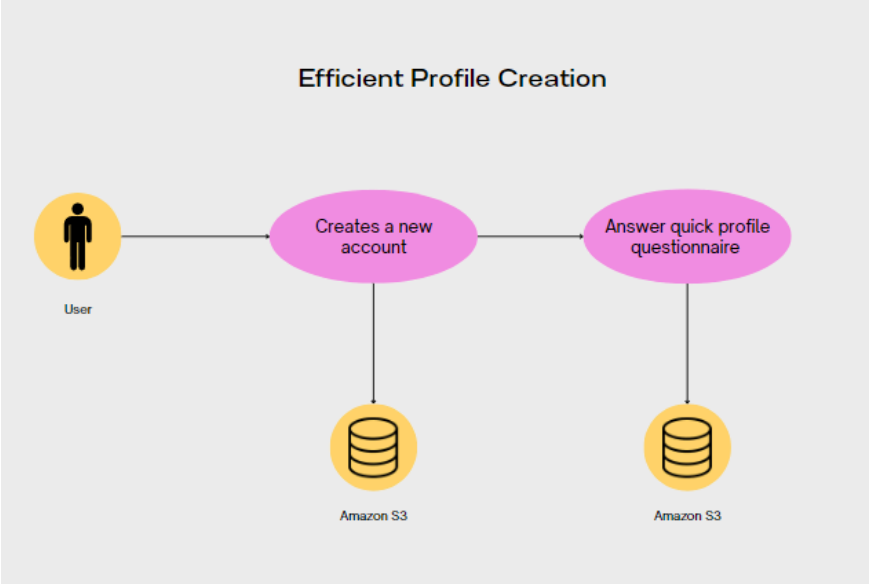

Table 1: Efficient Profile Creation Use Case

This use case is a MUST-have. Assuming this is a new user, the user will create a new account, once the user enters the appropriate information, it will be sent to the Amazon S3 buckets to store the account information. Users will be prompted with a quick profile questionnaire that they will answer. After completing the questionnaire, it will lead them to their profile view page, where they can view other user's profiles.

Compared to initial expectation: there is no Amazon EC2 instance of the application.

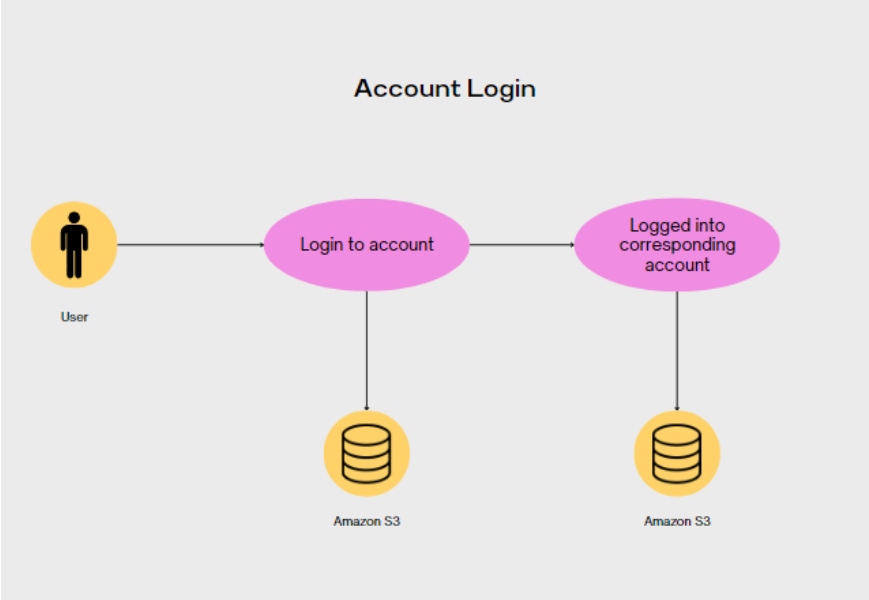

Table 2: Account Login

This use case is a MUST-have. Users will attempt to log into their account, the user cannot enter null values. Users will be logged into their corresponding account if that account already exists in the Amazon S3 bucket.

Compared to initial expectation: there is no Amazon EC2 instance of the application.

4 Software Evaluation

Testing initially began with Java JUnit testing but was instead moved to Postman. Given the use of AWS, the focus was primarily on valid file input and retrieval. Error handling was of great concern as well. Because of the high sensitivity of the data being stored, security and testing are a priority. Strategies in both must be kept up to date with future features in the event of new exploit attempts.

4.1 Functionality

The primary data being handled were users' profile data stored in JSON files, so content had a consistent storage pattern for verification. Testing consisted of manual and automatic function testing, and began after the initialization of AWS which allowed files to be stored and sent via URLs. Postman allowed for URL input and the creation of test cases, which ensured that data stored in HeartZone's S3 buckets was correct. Other than data validation, server responses, response times, and error handling were also tested. Issues were initially present with access compilations utilizing GET and PUT, but with further understanding of the bucket system, these were solved. Certain permissions had to be granted to permit testing, and server output had to be standardized so test cases could expect the correct output. Functionally there are no current issues found from testing, however as future features are added, concerns over scalability with more users and volume with chats/pictures may arise. Based on the current implementation, simple storage, and function have allowed for simple testing and production for development. Testing documentation is stored as well as a JSON import file which can be used on Postman.

Test example:

```
{
"name": "John Doe",
"email": "john.doe@example.com",
"password": "SecurePassword123!",
"dob": "1990-01-01"
}
```

i. User data input example: Test cases would check for these headers and ensure no additional content was not added that could exploit system functionality.

```
var settings = {
  "url": "https://t4fh12f682.execute-api.us-east-2.amazonaws.com/v1/heartzonedb?file=test/calebtest.json&method=PUT",
  "method": "PUT",
  "timeout": 0,
  "headers": {
    "Content-Type": "application/json"
  },
  "data": JSON.stringify({
    "name": "John Doe",
    "email": "john.doe@example.com",
    "password": "SecurePassword123!",
```

ii. Code used in Postman for API testing: Content from test JSON files was imported and sent to AWS.

```
{"name": "John Doe", "email": "john.doe@example.com", "password": "securepassword123", "dob": "1990-01-01"}
```

iii. Server return message to verify uploaded content: Return was also used in use cases to check for correct storage.

```
pm.test("Verify request body contains required headers", function () {
    const requestData = pm.request.body;


    const jsonData = JSON.parse(requestData);

    pm.expect(jsonData).to.have.property("name");
    pm.expect(jsonData).to.have.property("email");
    pm.expect(jsonData).to.have.property("password");
    pm.expect(jsonData).to.have.property("dob");

        const expectedHeaders = ["name", "email", "password", "dob"];
    const actualHeaders = Object.keys(jsonData);
    const extraProperties = actualHeaders.filter(header => !expectedHeaders.includes(header));

    pm.expect(extraProperties).to.eql([], `Extra properties found: ${extraProperties.join(', ')}`);

});
```

iv. Javascript code for use case: Each header is individually checked, followed by a scan for unnecessary content.

4.2 Security

Security methods and tools that have been employed:

- Input validation
    - Only alphabetic characters for name, school, profession
- Password encryption on the client side
    - AES
- File hashing on the server side in Lamdba
    - SHA2-256 with 64 rounds
- Re-encrypt file after hashing

Security issues that are still open:

- Directory traversal
- Cookies
- Threat identification:
    - A threat identification model will be created when a new feature is released, a security incident occurs, or changes to architecture or infrastructure. The threat identification model will answer questions like; what is being worked on? What could go wrong? What will be done about it? etc.
- Security requirements:
    - Data- personal information and communication are:
        - Hashed with SHA2- 256 with 64 rounds
        - Encrypted with AES
    - Unauthorized access- audit and monitor user access to the application regularly. Revoke access for inactive or compromised accounts.

    - Amazon S3 buckets- used bucket policies and ACLs in order to control access to the buckets. S3 server captures access logs with detailed records for requests made to the buckets.
- Secure coding standards:
    - Input validation- prevent injection attacks like cross site scripting by only allowing alphabetic characters and numbers in text input fields.
    - Error handling- informative error message popups
    - Secure file handling- validate and sanitize file uploads by only allowing .jpeg and .png files to be uploaded in profile creation and chats.
- Code verification:
    - Code review- team members review each other's code to identify issues.
    - Verify code follows consistent styling to enhance readability and maintainability.
    - The code includes sufficient comments for understanding, maintenance, and future development.
    - Verify new code changes do not have unintended side effects or changes to existing functionality through repression testing.

5 Work to be Done

- Matching algorithm
- Swiping animation
- Geolocation API
- Additional security measures
- Chat room:
    - live updates

    - sending images
- Make it visually and functionally effective via a phone. Currently works better on a web browser.
- Bugs to fix:
    - sizing issue with the screen
    - emulator issue due to the type of react we used:
        - react.js is optimized for websites
        - react native is optimized for phone apps.

6 Guides

Product Name: React.js

User/Administrator guides:

Steps to run our React.js app:

- Install node.js and npm
- Our GitHub repository should have all the React.js libraries already.
    - Clone our repo into your IDE
- With our app's code on your IDE, go to the command line and make sure you are in the correct directory (/HeartZone/heart-zone) and run 'npm start' to boot up a web server locally.
    - We used VisualStudioCode and ran it from the command prompt inside the IDE

Product Features:

Component-Based: React.js allows developers to build UI components that can be reused throughout an application, promoting code reusability and maintainability.

Virtual DOM: React.js uses a virtual DOM to efficiently update the UI. It compares the virtual DOM with the actual DOM and only updates the parts that have changed, reducing unnecessary re-rendering and improving performance.

JSX: JSX is a syntax extension for JavaScript that allows developers to write HTML-like code within JavaScript, making it easier to create and visualize UI components.

Support for Server-Side Rendering: React.js provides support for server-side rendering, which improves performance and SEO by rendering the initial HTML on the server before sending it to the client.

Product Functions and Capabilities:

UI Development: React.js is used for building user interfaces for web applications.

Component Reusability: React.js promotes component reusability, allowing developers to create modular UI components that can be reused across different parts of an application.

Performance Optimization: React.js optimizes performance through features like virtual DOM, which reduces the number of DOM manipulations and re-renders, resulting in faster UI updates.

Integration with Other Technologies: React.js can be integrated with other technologies and libraries, such as React Native for building mobile applications, or backend frameworks like Node.js for full-stack development.

Testing: React.js applications can be easily tested using various testing frameworks and tools, such as Jest and Enzyme, to ensure the reliability and quality of the codebase.

7 Deliverables

Github URL: https://github.com/BeriPeric/HeartZone