The Nacho Libre's

Project Report

Breanna Caudle, Keegan Lloyd, Ben Shinnick, Zachary Tisdale

Capstone Project Fall, 2023

The University of West Florida

11/13/2023

CIS4595 Capstone Project

Dr. Bernd Owsnicki-Klewe

Table of Contents

Table of Contents

1 Project Description....3
1.1 Ideas and Concepts.... 3
1.2 Functions.... 3
2 Timeline....4
2.1 Final Timeline vs Initial Timeline.... 4
3 Project Results....5
3.1 Implemented Use Cases.... 5
4 Software Evaluation....8
4.1 Functionality Overview....8
4.2 Functionality Documentation.... 9
4.3 Security Overview....9
4.4 Security Documentation....10
4.4.1 Threat Identification.... 10
4.4.2 Security Requirements....11
4.4.3 Data Protection Methods.... 12
4.4.4 Secure Coding Standards and Code Verification.... 12
4.4.5 Security Testing.... 12
5 Work to be Done....13

1 Project Description

1.1 Ideas and Concepts

Well-List is a web application designed to simplify and enhance the way users manage their daily mundane tasks. The goal of juggling work, school, a social life and our own wellness can often lead to stress. Well-List addresses this challenge by providing users with a user-friendly interface to organize, track, and complete their tasks efficiently. This application gamifies wellness by rewarding completion of goals with points that can be redeemed for virtual plants, encouraging the user to have fun with daily tasks without becoming overwhelmed. Users are able to create and prioritize tasks in the hopes that nothing will fall through the cracks. Well-List aims to transform task management into a rewarding and enjoyable experience.

1.2 Functions

Using Well-List allows users to create and login to secure accounts that contain their tasks and reward data. From their profile page they will be able to create, prioritize and update tasks. On completion of tasks points will be rewarded to the user, which can be redeemed in a separate shop page of our app that contains virtual plants that are up for sale. Users are able to go back and view tasks and rewards for previous days.

2 Timeline

2.1 Final Timeline vs Initial Timeline

Our team was able to follow the original expected timeline very closely with only a few alterations. Differences from the original timeline are highlighted below:

09/11/23	Presentation 1 Project Plan
09/18/23	Finalizing Project Ideas
09/25/23	Distributing Planned Work/Group Meeting
10/02/23	Sprint 1. <Applying Chosen Model / Tech Document, Finalize Project Details>
10/09/23	Sprint 1 Cont. <Initialize Development Environment, Design Choices (UI Appearance, Logo, etc.)> Started Implementing UI, Work from Sprint 2
10/16/23	Sprint 2 <Implement UI, Account Creation>
10/23/23	Sprint 2 Cont. Presentation 2
10/30/23	Sprint 2 Review, Sprint 3 <List creation/Management, Rewards, ~~Progress Tracking~~>

11/06/23	Sprint 3 Cont.
11/13/23	Sprint 3 Review, Sprint 4 <Project Finalization, Documentation Review> Continued List Creation/Management from Sprint 3
11/20/23	Sprint 4 Cont.
11/27/23	Final Presentations

3 Project Results

3.1 Implemented Use Cases

Within our Technical documentation we provided more than 2 Use cases that have been implemented (with one dropped), for that reason we are providing the 2 most pertinent use cases for this submission. Both use cases were implemented according to the original expectations outlined in the technical documentation.

01. **Use Case Name**: Managing Daily Tasks

    Priority: TOP

    Actors:

    Primary Actor: User

    Secondary Actor: System

    Description: The user can create, view, edit, and delete their daily tasks using

the system's interface, allowing them to better organize their tasks.

Preconditions:

The user is logged into the system

The user is on the tasks management interface

Flow of Events:

1.) Creating a Daily Goal:

- user selects the option to create a new task

- The system will display a form with fields for their task description, due date, and priority

- The user enters the task details and submits form

- The system validates the input and creates the new task, associating it with the user's account

2.) Viewing Daily Goals:

- The user access the tasks management interface

- The system retrieves and displays a list of the user's tasks, sorted by priority

3.) Editing a Daily Goal:

- The user selects a task from the list

- The system displays the task details in an editable form

- The user modifies the task details and saves the changes

- The system validates the updated information and updates the goal

4.) Deleting a Daily Goal:

- The user selects the tasks from the list

- The system displays a confirmation prompt

- The user confirms deletion

- The system removes the goal from the user's list of tasks

Alternate Flow:

1.) Task Validation Error:

If any error with the creation or modification of the task, the system will display the appropriate error messages, and the user will be prompted to correct the errors

Postconditions: The user's tasks are updated based on their interactions

02. **Use Case Name**: Gaining Points for Task Completion

Priority: Top

Actors:

Primary Actor: User

Secondary Actor: System

Description: The user can earn spendable points through the completion of their tasks.

Preconditions:

The user is logged into the system

The user has daily tasks assigned in the system

Flow of Events:

1.) Completing a Daily Task:

- The user accesses their list of daily tasks

- The user selects a task that they have completed

- The system validates the completion status of the task

- If the task is completed, the system awards points to the user's account based on task's points value

- The system updates the user's total points

2.) Viewing Point Balance:

- The system retrieves and displays the user's current points balance

Postconditions:

- The user points balance is updated after completing tasks, reflecting the earned points

- Points are stored securely and persistently in the system database

4 Software Evaluation

4.1 Functionality Overview

The main testing tool used throughout the duration of the project is Jest. Whilst using Jest, the primary testing methods that we have followed is integration testing as well as regression testing. With integration testing, there were tests to make sure that every functionality of MongoDB that we would use was working correctly as well as help us acquire a better understanding of the tool itself. We employed regression testing as well since we have changed up the design of our schemas. Regression testing helped make sure that the previous tests were still passing and that nothing failed upon the addition and edits of new schemas as the application evolved.

4.2 Functionality Documentation

The two main forms of documentation were a Requirements Traceability Matrix (RTM) and a Test Case Documentation (TCD). The RTM helped outline the main expectations from the web app and how it should behave. Each requirement from the RTM was then linked to a specific test case id that is then expanded upon further in the TCD. As stated, the TCD goes more in depth by explaining what each test is doing, the test data provided, the expected result, the actual result, and any comments on the test and/or how the test should be performed. Both forms are then color-coded to provide a quick glance at how many tests are passing, failing, partially passing, or needing to be implemented as whole.

4.3 Security Overview

The Well-List application prioritizes security by implementing authentication mechanisms, incorporating secure data handling practices and by adhering to secure coding standards. The security tools and methods used include Bcrypt, Passport and Mongoose, all of which are used to safeguard user data and prevent unauthorized user access. Bcrypt and Passport provide secure password hashing, while Mongoose in connection with MongoDB's inherent security features allow the user data integrity and confidentiality to remain secure. Since Well-List focuses on taking user provided information, our main priority is data protection.

However, we have also completed thorough documentation on potential risks and misuses of the app in order to better understand potential attack vectors. Clear code security requirements have been given out to assist in making sure any new addition of

code is kept within the best practice standards. A detailed Risk Assessment document has been prepared, addressing the potential threats and vulnerabilities to our system. Additionally, the Misuse Case document outlines potential misuse cases with provided solutions to improve Well-Lists' security

The next step for the security of Well-List is the addition of auditing and regular penetration testing. Auditing will be used to check any abuses to the point system, along with potential attempts at malicious actions. While pentesting will be conducted to identify and rectify any vulnerabilities additional code has caused.

4.4 Security Documentation

4.4.1 Threat Identification

Our application has a few *external* threats that could drastically affect our CIA. This includes network attacks. This comes in the form of potential risks of unauthorized access or even disruption of communication channels between the network and the database. A form of protection for this is to create firewall rules within MongoDB to control the list of IP addresses that are allowed to connect to the server. Another external threat comes in the form of Denial of Service (DoS) or a Distributed Denial of Service (DDoS). This can occur if someone tries to target our database by creating a lot of traffic, and since MongoDB does not come with protection against a threat like this, our server could go down. If we had the materials, we could invest in load balancing resources to assist in mitigating DDoS attacks.

Some *internal* threats are with unauthorized access. We need to create a requirement in the system that does not allow a user to access the homepage without

confirmation on login. We have already incorporated password encryption and the use of logins. Another potential risk is insider threat, a user with access to our database could abuse the rewards system. This can be prevented with user monitoring and audit trails. Along with restricting user access privileges.

Finally Well-List does have some untested *application-specific* threats. This includes Input Validation Attacks, which will need to be tested and mitigated through penetration testing SQL injections in the task creation page. And since we are using sessions for each user, some users are at risk of Session Hijacking. We have implemented session serialization and deserialization through passport.

4.4.2 Security Requirements

Authentication and Authorization are our main focus in this assignment. We created user authentication by adding a password requirement, however our password requirement does not implement strong password policies. This will be added upon completion of the create account page. Another important security requirement that we focus on is *Data Protection*. In order to do this we have encrypted data using a hashing algorithm (bcrypt).

Secure Coding Standards is another security requirement. This is why we implemented the Code Review Checklist to make sure that the code is following best practices.

4.4.3 Data Protection Methods

For this section we hash the user passwords within the database storage. We will also be applying the hash to critical information that the user inputs to make sure the data integrity remains. Passport should handle secure communication to the database.

4.4.4 Secure Coding Standards and Code Verification

We created and used a Code Review Checklist to enhance understanding and implementation of secure coding principles. The checklist allows us to inspect the code manually before submitting it to Github. We have gone over secure guidelines when dealing with secure error handling to avoid disclosing sensitive information.

4.4.5 Security Testing

After the web application is finished in order to test its security we will be using penetration testing to identify and address any vulnerabilities. The next step after pentesting is to automate the security testing through the use of automated tools. Regular penetration testing (internal and external) will assist in simulating real world attacks and identify vulnerabilities in Well-List. This continuous testing, combining manual and automated methods, ensures the resilience of our application against security threats.

5 Work to be Done

If the group had additional time to create a finalized build and product, various extra features would be included within Well-List.

1. Finish server and database integration:

   For development convenience we saved some task and reward data locally on the browser instead of the MongoDB where it belonged. We were hoping to get all task and reward data fully integrated this last week before the final submission, but were not able to due to time constraints. However, all account handling and account data is fully integrated and set up on the server.

2. Progress visualization:

   Initially intended but dropped due to time constraints, Well-List would benefit from the inclusion of progress based tracking and visualization. Allowing users to see a chart outlining the summary of their activity over various durations of time, week to week, monthly, or even yearly activity would be modeled to provide users with at a glance summaries of their overall progress in doing daily tasks that they defined.

3. Project Deployment:

At current Well-List is deployed by a user deploying their own server and database. While the current build automates much of the communication for the user a finalized build of Well-List would come with the inclusion of a deployed web app users could simply access without the need of individual setup.