TEST PLAN FOR

WORDEO

Changelog

Version	Change Date	By	Description
1.0.0.	2023.10.11	All Members	Initial Test Plan

1. Introduction

1.1. Scope

- Account Creation & Management
  - Create an account
  - Login to an account
  - Updating username
- Leaderboard
  - Retrieve leaderboard for the core game
  - Search on leaderboard
- Core game
  - Retrieve words

    - Custom Game Settings
- Multiplayer Rooms
    - Global/In-Game Chat
- Currency System
    - Power-ups
    - In-game Store

1.2. Roles and Responsibilities

Name	Net ID	GitHub username	Role
Alyssa Gregorash	gregoraa	c4ke2	Frontend Developer
Benedict Agupitan	aguptiab	BenedictAg	Frontend Developer
Dani Youn	yound	DaniYoun	Backend Developer, DevOps Engineer
Hamdi Elzard	elzardh	hamdielzard	Frontend Developer, Project Manager, Level Designer
Souvik Ray	rays1	raysofhopes	Backend Developer

Role Description

- **Frontend Developer**: Develops the user interactable component of the project.
- **Backend Developer**: Develops the database and data access layers of the project.
- **Project Manager**: Oversees the project requirements and specifications.
- **DevOps Engineer**: Develops and maintains the infrastructure for CI/CD of the project.
- **Level Designer**: Responsible for creating the different levels of the game. Specifically for our project, responsible for creating word data and setting their difficulty.

2. Test Methodology

2.1. Test Levels

2.1.1. Account Creation & Management

2.1.1.1. Unit Tests

1. POST for a new account should create a record of an account in the database and should return the created document with an HTTP 200 response
2. POST for an account with a preexisting username should not create a new account in the database and should return an HTTP 409 (conflict) response
3. GET for an account that exists in the database should return the document for the account with an HTTP 200 response
4. GET for an account that does not exist in the database should return an HTTP 404 response
5. PATCH for an account should update the account information (excluding username) and should return the update status with an HTTP 200 response
6. The home page should render a sign in button for the users to enter the sign in/up page
7. The sign in page should render a sign in box with username and password input fields available
8. The sign up page should render the sign up box with username and password input fields available
9. The sign in box should render a warning text when username or password field is empty
10. The sign up box should render a warning text when username or password field is empty
11. The account page should render the user's name, buttons to log out and edit account information, statistics for the user, and a list of achievements.
12. The account page should render apply and delete account buttons when in edit mode.
13. The account page should render an updated description when 'edit' is pressed, the textfield is changed, and 'apply' is pressed.
14. The account page should render 'delete' and 'cancel' buttons when delete account button is pressed in edit mode.

2.1.1.2. Integration Tests

1. The sign in page should display an error text when the username and password combination does not exist
2. The sign up page should display an error text when the given username already exists
3. An invalid request to a non-existing page should render a "page not found" webpage.

2.1.1.3. Acceptance Tests

1. User enters the website and creates a new account
    - Given that I am in a role of a website user
    - When I click on the "not logged-in" button on the top right side of the page
    - Then the system shows a sign-up & sign-in page with a username and password input fields

- When I fill in all the text fields on the sign-up form
- And I click on the ‘create account’ button
- Then the system saves my username and password pair
- And the system takes me to the account screen with my account information available
- And the system shows me a button to navigate back to the home page

2. User enters the website and updates their account information
   - Given that I am in a role of a website user
   - When I click on the “Account” button on the top right side of the page
   - Then the system shows an Account Information page that lists all my achievements earned, games played, and highest score earned.
   - When I click on “Edit” button at the top right
   - Then the system shows me a popup to change my account description
   - Then the system saves my account’s description
   - And the system takes me to the Account Information page

2.1.2. Leaderboard

2.1.2.1. Unit Tests

1. POST for a score with an existing user should update a record of a score in the database and should return the created document with an HTTP 200 response
2. POST for a score with a non-existing user should not create a record of a score and return an HTTP 404 response
3. POST for a score with an invalid game mode should not create a record of a score and return an HTTP 400 response
4. POST for a score with an invalid score (negative) should not create a record of a score and return an HTTP 400 response
5. GET on all scores should return a list of all scores in the database with an HTTP 200 response
6. GET on all scores with filters should return a filtered list of scores from the database with an HTTP 200 response
7. The leaderboard page should render a button to return to the home page
8. The leaderboard page should render a list of scores with usernames ordered in a descending order by the score
9. The leaderboard page should render a button to switch between different game modes
10. The leaderboard page should render a button to filter the list by a username
11. The leaderboard page should render a button to undo the filter when the list is filtered by a username

2.1.2.2. Integration Tests

1. At the completion of a game, if the user is signed in, the scores should be posted to the server

2.1.2.3. Acceptance Tests

1. User checks the leaderboard after playing a game
    - Given that I am in a role of a signed-in user
    - When I click the leaderboard icon from the home screen,
    - Then the system shows a list of users with their score and username ordered by the score
    - And the system shows my highest score highlighted on the list with my rank

2.1.3. Core game

2.1.3.1. Unit Tests

1. GET for a game should return the number of rounds requested and a variety of categories or a specific category with varying difficulties with an HTTP 200 response.
2. GET for a game outside the client should return with an HTTP 405 response.
3. GET for a round outside the client should return with an HTTP 405 response.
4. POST for a solved round should include the time it took, and the user that solved it and return with an HTTP 200 response
5. POST for a solved round with missing parameters: time to solve and user, should return an HTTP 400 response.
6. POST for a completed game should include the user's score, total time (for score verification), and the user and return an HTTP 200.
7. POST for a completed game without the required parameters should return an HTTP 400 response.
8. POST for a solved round outside the client should return with an HTTP 405 response.
9. POST for a completed game outside the client should return with an HTTP 405 response.
10. POST for a word should create a single word in the database and should return the created document with an HTTP 200 response
11. GET with a given "word" should return a list of word documents with an HTTP 200 response
12. PATCH with a given "word" should update a single word and return the update status with an HTTP 200 response
13. DELETE for a given word should delete a single word in the database and return with the delete status with an HTTP 200 response
14. POST for multiple words should return the list of words created with an HTTP 200 response
15. GET on all words should return a list of all words in the database with an HTTP 200 response
16. GET on all words with filters should return a filtered list of words from the database with an HTTP 200 response

17. The home page should render a play button for the users to enter the core game
18. The game page should render a timer
19. The game page should render a hint text
20. The game page should render remaining letters and guesses
21. The number of word boxes should be the same as length of words given
22. Correct letters should update the letter boxes. Boxes hosting correct letters should turn green and have visible letters
23. All letter boxes should contain each letter of the word
24. Incorrect letter boxes should contain the correct letter
25. Unguessed letter boxes should be white and invisible
26. When all correct words have been guessed it should call a function for game end
27. Number of incorrect boxes should match number of incorrect letters
28. Timer should match time given
29. When timer ends it should call a function for round end

2.1.3.2. Integration Tests

1. When a game is started, the client should retrieve 10 words from the server
2. When a round starts, the client should set the timer and stop taking input after the timer has reached 0
3. When the word is complete (guessed all letters), the client should calculate the score based on the remaining time
4. When the game is complete, the client should see the resulting score.

2.1.3.3. Acceptance Tests

1. User plays a game in solo mode
    - Given that I am in a role of a signed-in user
    - When I click on the play button on the home page
    - Then the system shows me the game page with white empty boxes and a description of a word
    - And the system shows a timer on the top center of the screen along with the game information: current score, current round
    - When I press a key,
    - Then the system either updates the white empty box to a green box with my key if it was correct, or if it was wrong adds a new red box with my key to the bottom of the screen
    - When I finish guessing, or the timer reached 0,

- Then the system shows me my score for the current round
- When I finish playing 10 rounds
- Then the system shows the total score for all 10 rounds
- And the system submits my score to the leaderboard
- And the system shows a button to play again and a button to return to the home screen

2.1.4. Multiplayer Rooms

2.1.4.1. Unit Tests

1. POST to create a new multiplayer room should return the created document with an HTTP 200 response
2. GET for an existing multiplayer room should return the document with an HTTP 200 response
3. GET for a non-existing multiplayer room should return an HTTP 404 response
4. PATCH to a multiplayer room should return the update status with an HTTP 200 response
5. DELETE to a multiplayer room should return the delete status with an HTTP 200 response
6. POST to a chat for a multiplayer room should return an HTTP 200 response
7. GET to a chat for a multiplayer room should return the list of documents with an HTTP 200 response
8. The home page should render a 'multiplayer' button to access the multiplayer page
9. The multiplayer page should render a list multiplayer rooms and a button to create a new room
10. The create new room form should render a form with an input text for the room name
11. The multiplayer room page should render a list of users
12. The multiplayer room page should render the room chat

2.1.4.2. Integration Tests

1. When a message is sent to the room chat, the message should be broadcasted to all members in the room
2. When a new room is created, the multiplayer room should be displayed to all users
3. When a puzzle (round) is solved, all clients should wait for the timer or until everyone has solved the puzzle.
4. When a puzzle (round) is solved and all clients have finished, all clients should move to the same next puzzle.
5. When a game ends, all clients should see the option to play again or exit.
6. When a game ends, all clients will see all their opponent's score and who has won or lost the game.

2.1.4.3. Acceptance Tests

1. User opens a multiplayer room

- Given that I am in a role of a signed-in user
- When I click on the ‘multiplayer’ button from the home page
- Then the system shows the multiplayer page with a list of rooms and a button to create a new room
- When I click on the ‘create room’ button
- Then the system shows me a form with one input text for the room name and two buttons to ‘create’ and ‘cancel’
- When I fill in the input text and click the create button
- Then the system creates a new multiplayer room
- And the system shows me the multiplayer room with a list of users conaining my username
- And the system shows me the multiplayer chat on the bottom left side of the screen

2.1.5. Currency System

2.1.5.1. Unit Tests

1. PATCH for coins with a valid user should update the user’s coins and return with an HTTP 200 response
2. PATCH for a new item with a valid user should update the user’s inventory with the requested item and return with an HTTP 200 response
3. PATCH for an invalid item with a valid user should not update the user’s inventory and return with an HTTP 400 response
4. GET for coins of a valid user should return the coin balance of the user with an HTTP 200 response
5. GET for user inventory for a valid user should return the list of items with an HTTP 200 response
6. The shop page should render the number of coins a user has
7. The home page should render a button for the users to navigate to the store page
8. The store page should render a list of available power ups and cosmetics
9. The store page should render a text box containing the description of an item when hovered on the item
10. The game page should render available power ups

2.1.5.2. Integration Tests

1. The number of coins should be adjusted appropriately when purchasing an item from the store
2. The user inventory should be adjusted appropriately when purchasing an item from the store

2.1.5.3. Acceptance Tests

1. User purchases an item
    - Given that I am in a role of a signed-in user

- When I click the button ‘store’ from the home page
- The system shows a screen with a list of items
- And the number of coins that I currently own
- When I hover my mouse on an item on the list
- Then the system shows a text with the description of the item
- When I click on the item
- The system shows a button ‘purchase’ and ‘cancel’
- When I click on the button ‘purchase’
- Then the system updates my inventory
- And the system updates my number of coins
- And the system shows the store again with an updated amount of coins

2.2. Test Completeness

Testing will be considered complete when,

- 100 % back-end code coverage
- At least 70 % front-end code coverage
- All automated builds & tests are completed successfully

3. Resource & Environment Needs

3.1. Testing Tools

Here are the tools used for testing and tracking bugs for our project:

Tracking & Automations

- GitHub Actions
- GitHub Project & Issues

Client (frontend)

- React Testing Library

Server (backend)

- Jest
- Supertest

3.2. Test Environment

Here are the minimum requirements for testing the Wordeo client and server projects:

- Operating System Requirements:
    - Windows (x64, ARM64)
    - macOS (x64, ARM64)
    - Ubuntu Linux (x64, ARM64)
- Software Requirements:
    - Node.js 20 or higher
    - MongoDB Community Server 7.0 or higher
- Packages (Node Package Manager):
    - React
    - Express.js
    - Mongoose

4. Terms/Acronyms

TERM/ACRONYM	DEFINITION
Puzzle	A word puzzle played under each Round
Round	A time-based period consisting of a puzzle for the user to play.
Game	A full-game of 10 rounds, played either in solo play or multiplayer.
CI/CD	Continuous Integration/Continuous Development
Document	A single record in a MongoDB database