# 

{Word Guessing Game Design and Development}

Abstract:

This report is about the design and development of a word-guessing game that use optimized data structures and algorithms to make sure the game is scalable, interactive, and works good. It talks about the proposed solution, methodology, the technologies used in backend and frontend, and also some future improvements we could do.

Prepared By:

Akar Hassan (Back-end Development)

Sana Latif (Front-end Development)

Muslim Mahmood (Front-end Development)

Muhammad Jalal (Database Design and Development)

Table of content:

1. Introduction

2. Problem Statement

3. Proposed Solution

4. Methodology

4.1Used Data Structures and Algorithms

4.2 Flow Chart

4.3 Backend

* (Flask)
* (JavaScript)

5. Conclusion

6. Future Work

# 1. Introduction

We’re making a word-guessing game where players try to figure out the letters in a word to earn points. The game will keep players interested with things like leaderboards, word queues, and tracking which letters they’ve guessed. This document explains how we’ll build the game using the right data structures and algorithms.

# 2. Problem Statement

A lot of word-guessing games struggle with managing words, keeping track of guesses, and handling leaderboards properly. When the design isn’t done right, the game can end up slow and not work well if more players or features are added. This makes it harder to keep things running smoothly as it grows. The challenge here is to create a game that’s not just fun and interactive but also works efficiently and can handle more players or updates without any issues.

# 3. Proposed Solution

The idea is we build a web-based word guessing game that’s efficient and also fun to play. It’ll use smart data structures and algorithms to handle words dynamically, so the word list can be updated or changed easily. The game will keep track of guesses players make, so they don’t repeat stuff, and the leaderboard will be fast and interactive to keep things exciting.

For the backend we’ll use MySQL to store stuff like words, scores, and guesses in an organized way. On the frontend JavaScript will handle the real-time parts so the game feels responsive. This setup should make sure the game runs smooth even if more players join or we add new features later on.

# 4. Methodology

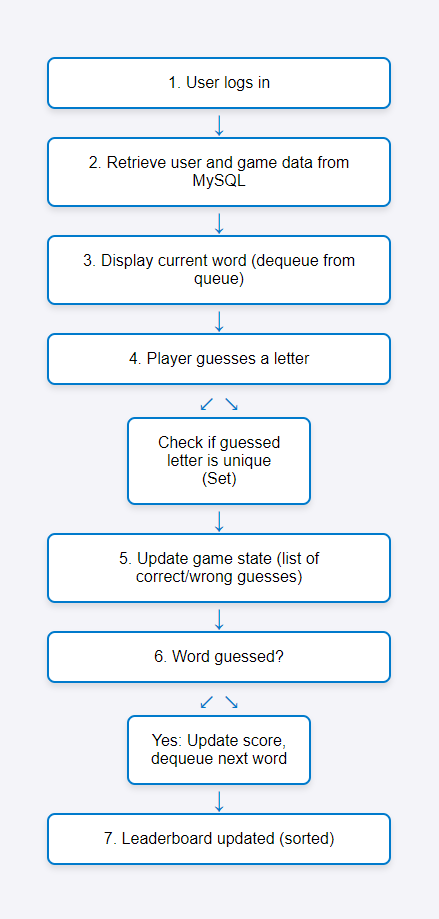
## 4.1Used Data Structures and Algorithms

We gonna use relational tables in MySQL for the main data. The users table is for player info, scores table will track how everyone doing, and the words table is where all the game words will be stored.

For managing user sessions, we’ll use a dictionary in Flask to handle things like login states and user-specific info. Lists or arrays will be used to store things like the word list, guessed words, wrong letters, and scores, since they’re simple and easy to work with.

A queue will help manage the flow of words during the game, making sure it’s handled in order. A set will be used to track guessed letters, so players don’t accidentally repeat guesses. And for the leaderboard, we’ll use a sorting algorithm to keep scores ranked properly.

## 4.2 Flow Chart



## 4.3 Backend

### (Flask)

from flask import Flask, session, jsonify

import mysql.connector

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key'

# Database connection function

# This function establishes a connection to the MySQL database.

def get\_db\_connection():

return mysql.connector.connect(

host='localhost', user='root', password='password', database='game\_db')

# Login endpoint

# This endpoint handles user login. It simulates a login and sets the session data for the logged-in user.

@app.route('/login', methods=['POST'])

def login():

# Example login logic (simplified)

session['loggedin'] = True # Set the logged-in state in the session.

session['id'] = 1 # User ID (for example)

session['username'] = 'Player1' # Username for the logged-in player

return jsonify({'status': 'logged\_in'}) # Responds with a JSON object indicating the login status

# Get Word endpoint

# This endpoint retrieves a word from the database to be used in the game.

@app.route('/get\_word', methods=['GET'])

def get\_word():

connection = get\_db\_connection() # Establishes a connection to the database

cursor = connection.cursor() # Create a cursor to execute SQL queries

cursor.execute("SELECT word FROM words LIMIT 1") # SQL query to get a single word from the 'words' table

word = cursor.fetchone() # Fetches the first word from the query result

connection.close() # Closes the database connection

return jsonify({'word': word[0]}) # Returns the word in JSON format\_in'}) # Responds with a JSON object indicating the login status

### (JavaScript)

// Initialize word queue

// The array stores a list of words to be guessed by the player. A queue-like behavior is simulated by shifting the first word when needed.

let words = ["apple", "banana", "cherry"]; // Array of words for the game

let wordQueue = [...words]; // Copy of the words array to manage the words dynamically

let currentWord = wordQueue.shift(); // Dequeue the first word from the wordQueue

// Initialize guessed letters set

// The set is used to keep track of the letters that have already been guessed by the player to avoid duplication.

let guessedLetters = new Set(); // Set data structure ensures that each guessed letter is unique

// Handle guesses

// This function processes the player's letter guess, checks if the letter has been guessed before, and provides feedback.

function guessLetter(letter) {

if (guessedLetters.has(letter)) { // Check if the guessed letter is already in the set of guessed letters

console.log("Letter already guessed!"); // Feedback to the user if the letter is a duplicate

return;

}

guessedLetters.add(letter); // Add the letter to the set to mark it as guessed

if (currentWord.includes(letter)) { // Check if the guessed letter is in the current word

console.log(`Correct guess: ${letter}`); // Feedback for correct guess

} else {

console.log(`Wrong guess: ${letter}`); // Feedback for incorrect guess

}

}

// Leaderboard logic

// The leaderboard is an array of objects, and sorting is done using a sorting algorithm.

let scores = [{name: "Player1", score: 50}, {name: "Player2", score: 75}]; // Example scores array

scores.sort((a, b) => b.score - a.score); // Sorting the leaderboard based on the score in descending order

console.log("Leaderboard:", scores); // Display the sorted leaderboard

# 5. Conclusion

The proposed game is gonna mix simple gameplay with strong backend and frontend systems. using the right data strctures is important because it helps make sure the game can grow bigger and still work fine. we need it to be fast, interative, and keep perform well even if more people play. the backend is gonna be organized so it can handel a lot of users without crashing, and the frontend will make sure players have fun and enjoy the game. chosing the right data structures will help the game be effcient and keep it run smooth no matter how big it gets.

# 6. Future Work

We plan to add real-time multiplayer features. Also, we want to make it so the game supports differrent languages. And we might add some power ups and more gameplay modes to make it more fun.

7.References:

Flask Documentation: https://flask.palletsprojects.com/

MySQL Documentation: https://dev.mysql.com/doc/

JavaScript MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript

can you make a cover page template for this report